ESCUELA POLITÉCNICA DEL EJÉRCITO

FACULTAD DE INGENIERÍA ELECTRÓNICA

PROYECTO DE GRADO PARA LA OBTENCIÓN DEL TÍTULO EN INGENIERÍA ELECTRÓNICA

"DISEÑO E IMPLEMENTACION DE UN DISPLAY ROTATIVO RGB PROGRAMABLE CON INTERFASE INALAMBRICA SIRC Y CONTROL A TRAVES DE TECLADO REMOTO ABC2"

ANTONIO JAVIER RUALES RÍOS

QUITO - ECUADOR

ABRIL 2006.

CERTIFICACIÓN

Certificamos que la presente Tesis de Grado, "DISEÑO E IMPLEMENTACION DE UN DISPLAY ROTATIVO RGB PROGRAMABLE CON INTERFASE INALAMBRICA SIRC Y CONTROL A TRAVES DE TECLADO REMOTO ABC2" fue realizada en su totalidad por el señor Antonio Ruales Ríos bajo nuestra dirección, como requerimiento parcial a la obtención del título de Ingeniero Electrónico con especialidad en Telecomunicaciones.

Sr. Ing. Byron Navas	Sr. Ing. Víctor Proaño
DIRECTOR DE TESIS	**CODIRECTOR DE TESIS**

AGRADECIMIENTO

Primero que todo quisiera agradecer al Sr. Decano de nuestra facultad que muestra un verdadero interés por el bienestar y la superación de la misma dando su apoyo incondicional tanto a profesores y personal administrativo así como a los estudiantes.

A los profesores que conforman el cuerpo académico de nuestra facultad que además de sentir una gran pasión por los temas que enseñan también despliegan un gran esfuerzo para que los estudiantes descubran la misma fascinación.

Un agradecimiento especial para mi Director y Codirector de Tesis que me brindaron su apoyo para salir adelante con este proyecto aportando con ideas y guiándome a lo largo de su desarrollo para poder realizar un trabajo satisfactorio.

Este proyecto no hubiera sido realidad sin la iniciativa del Ing. Byron Navas que con su gran dedicación al estudio de los Sistemas Digitales y de Microcontroladores, en especial sus investigaciones sobre los Microcontroladores Atmel, ha brindado un importante aporte en este campo de la Electrónica.

Finalmente quisiera agradecer a mis padres Antonio F. Ruales y Narcisa E. Ríos que son las personas que hacen que todo lo que yo haga sea posible ya que sin su apoyo y su confianza en mi no encontraría las fuerzas para superarme.

Gracias Señor por permitirme vivir este momento.

DEDICATORIA

Para mis padres y hermanos que son la fuerza que me obliga a seguir adelante ya que por ellos soy capaz de hacer cosas que no haría por mí mismo.

Antonio Ruales Ríos.

PROLOGO

El tema mostrado en el presente proyecto consiste en el diseño un Display Rotativo para desplegar imágenes prediseñadas y mensajes con distintos colores programados a través de la interfase IR de los controles remotos de TV de la Sony y su implementación utilizando un microcontrolador AVR-Atmega, LEDs RGB y un teclado similar al sistema de ingreso de datos de los teléfonos celulares conocido como ABC2.

Un Display Rotativo es un dispositivo que hace girar un arreglo vertical de LEDs alrededor de su punto central. Mediante el control del tiempo de encendido y apagado de cada LED se forma un cilindro de imágenes luminosas alrededor que pueden representar figuras o letras.

Debido a que el microcontrolador AVR-Atmega que controla los LEDs se encuentra girando junto con los mismos esto representa un reto al momento de buscar la manera de darle energía para su funcionamiento y de comunicarlo con el usuario para su configuración. La solución al problema de la alimentación de energía se logra a través de un diseño mecánico ingenioso y la parte de la comunicación a través de una interfase IR.

Para desarrollar el prototipo, realizar el programa y grabarlo en el microcontrolador AVR-Atmega se emplean nuevas herramientas de trabajo que también serán explicadas junto con el proyecto: CodeVisionAVR C Complier, AVR Studio y PonyProg2000.

El diseño y la implementación de este proyecto ha sido detallada paso a paso a manera de tutorial o guía de elaboración a fin de lograr un mejor entendimiento del mismo y para permitir que sirva de referencia para aquellos que deseen saber más sobre el manejo de los microcontroladores AVR-Atmega.

ÍNDICE

CAPÍTULO 3: AMBIENTES DE DESARROLLO INTEGRADO (IDE)

CAPÍTULO 1

INTRODUCCIÓN. LED DISPLAYS

1. INTRODUCCIÓN.

Todos alguna vez han ido a un estadio o a un espectáculo público. En estos lugares siempre es necesario mostrar mensajes publicitarios como el anuncio de un artículo deportivo, una marca de ropa o un nuevo servicio de telefonía celular como se puede apreciar en la Figura 1.1. También es necesario desplegar mensajes informativos como el marcador de los equipos que se enfrentan o el tiempo de juego. Estos anuncios deben ser lo suficientemente grandes, brillantes y claros para que puedan ser vistos por toda la gente alrededor y para esto se utilizan las más variadas formas de presentaciones ya sea meramente con anuncios o letreros pintados o utilizando la más avanzada tecnología en despliegue de mensajes luminosos valiéndose de la electrónica moderna.

Figura 1.1. Minneapolis Target Center. El tablero central de este estadio además de mostrar el marcador y las estadísticas de un partido tiene una pantalla de 3 m de alto por 5 m de ancho con una resolución de 10 mm^2 por punto para mostrar imágenes y video.

Además de los lugares públicos que albergan una gran cantidad de personas, también las oficinas y otros lugares cerrados requieren de distintas formas de mostrar información o publicidad a las personas que se encuentran dentro de una habitación, como por ejemplo en un banco o en una oficina gubernamental que necesitan indicar el número del próximo cliente a ser atendido. De igual manera en este caso la información puede ser mostrada mediante el uso de letreros rotativos que cambian su contenido continuamente.

Tampoco se debe olvidar que todas las personas utilizan en sus hogares: relojes digitales, calendarios e identificadores de llamadas, entre otros artefactos. La mayoría de estos aparatos ya tienen en la actualidad pantallas electrónicas incorporadas, generalmente displays de siete segmentos o de cristal líquido.

Todos estos lugares ya sean diseñados para recibir a miles de personas, o solo algunas decenas, o sean de uso público o privado tienen en común su necesidad de presentar información importante a las personas presentes y para lograrlo utilizan muchos recursos entre ellos las pantallas luminosas que son básicamente superficies cubiertas por cientos o miles de pequeñas luces controladas electrónicamente.

Para implementar las pantallas luminosas se utilizan diferentes materiales como es el caso de los LEDs[1], luces de neón, fibra óptica, pantallas de cristal líquido o simplemente pequeños focos. Hasta la actualidad los elementos más utilizados son las luces de neón que han sido la elección preferida al momento de fabricar letreros luminosos por su fácil elaboración y su brillantez. Sin embargo en la actualidad han surgido varios nuevos competidores que presentan muchas ventajas sobre las luces de neón, uno de ellos es la fibra óptica que garantiza una alta seguridad de funcionamiento inclusive en ambientes lluviosos y que promete ser una solución a largo plazo ya que es una tecnología que recién está surgiendo.

[1] LED significa Light Emitting Diode

En el caso de los LEDs, estos ofrecen una solución más inmediata y entre sus ventajas está el hecho de que tienen una mayor duración. Realizando un arreglo rectangular con miles de ellos se pueden formar figuras y letras. Al principio estos LED Displays eran bastante simples, generalmente estaban compuestos por LEDs de un solo color y de esta manera se obtenían figuras y letras monocromáticas. En la actualidad algunos de estos dispositivos son bastante complejos y cada punto de luz o píxel es bastante pequeño lo cual le da más resolución a la imagen y además cada píxel está compuesto por los tres colores básicos que son rojo, verde y azul que al ser mezclados forman los demás colores lo cual nos permite obtener imágenes a color.

Una nueva propuesta para el despliegue de mensajes luminosos a través de LED Displays es el uso de dispositivos en movimiento. Dichos dispositivos aprovechan el hecho de que una imagen se queda impresa en la retina del ojo humano por cierto tiempo, aún cuando la fuente de luz que produjo dicha imagen ya haya desaparecido. Los beneficios que ofrecen estos dispositivos en movimiento son: bajos precios y una forma llamativa de presentación del mensaje. Los precios bajan ya que se pueden utilizar menos fuentes luminosas y la presentación del mensaje se vuelve llamativa ya que se logra presentar dichos mensajes de una forma que las personan no han visto antes ya que están acostumbradas a las conocidas pantallas rectangulares. Un mensaje girando y flotando en el espacio sin un eje ni cables visibles es algo que seguramente llamaría la atención de las personas que pasen alrededor.

1.1. Evolución de los LED Displays.

Los primeros LED Displays comenzaron a ser utilizados hace 45 años combinando tres elementos básicos: galio, arsénico y fósforo para obtener un pequeño encapsulado de 5 mm de diámetro capaz de emitir una luz roja de 655 nm de longitud de onda no muy brillante por lo que solo se utilizaba como luz indicadora en algunos tableros de control. Recién en 1970 se introdujeron nuevos colores como el verde, naranja y amarillo, el azul era un color difícil de elaborar en aquel entonces. Luego en 1990 surgieron los LEDs de alto brillo que permitían su

uso para la elaboración de letreros luminosos a gran escala y no solo para funcionar como luces indicadoras que era su aplicación principal hasta ese entonces[2].

Al principio los LED Displays no eran muy populares debido a que la gente no veía la necesidad de cambiar los comúnmente utilizados letreros hechos a base de materiales tan simples como acero, madera y pintura o en casos más modernos hechos con luces de neón por estos costosos dispositivos *electrónicos* de 30W. No había necesidad de cambiar ya que los LED Displays no mostraban en si ninguna ventaja sobre los letreros sin elementos electrónicos ya que solo presentaban mensajes de texto de 18 a 24 caracteres sin ninguna novedad[3]. Es más estos Displays eran bastante complicados de programar, mantener o alterar; tener un LED Display con varias líneas ya era un gran avance. El uso más común de los LEDs era generalmente como para representar texto a través del formato de 7 segmentos en los tableros de las fábricas o de los centros de control, en los relojes digitales, salas de espera en oficinas públicas, bancos y aeropuertos como se muestra en la Figura 1.2. Siempre en lugares cerrados y no muy grandes ya que los LED Displays no podían ser vistos a una gran distancia.

Figura 1.2. LED Display utilizado generalmente en los bancos

Con el tiempo y el avance de la electrónica los LED Displays fueron mejorando. Se redujo su costo y complejidad haciéndolos más accesibles al público que comenzó a verse más atraído por el surgimiento de una amplia diversidad de

[2] Información obtenida del artículo "LED Displays: A Formidable Foe" escrito por Johnny Duncan que puede ser encontrado en la página de Inernet www.signindustry.com/led/articles/2005-04-15-JD_LED_Displays.php3

[3] Información obtenida del artículo "Coming of Age" escrito por Bob Klausmeier que puede ser encontrado en la página de Inernet www.signweb.com/moving/cont/comingofage.html

modelos mucho más llamativos que hacían cosas que los letreros comunes y corrientes no podían hacer. Además de actualizar la información que presentaban, mostrar mensajes en movimiento y con efectos cromáticos. Incluso se había aumentado la intensidad con que brillaban los LEDs lo que incrementaba la distancia a la cual los Displays podían ser vistos como es el caso del letrero mostrado en la Figura 1.3.

Figura 1.3. LED Display moderno hecho a pedido

En la actualidad los LED Displays forman parte de una gran área de la electrónica que se conoce como EDS[4] y abarca todos los tipos de tecnologías utilizadas para la presentación de información e imágenes tales como LED, OLED, LCD, CRT[5], plasma y otros dispositivos de proyección digital. Entre estas opciones, una que llama la atención por la madurez que ha alcanzado su diseño y elaboración es el uso de pantallas de matrices de LEDs que permiten utilizar arreglos de LEDs para presentar texto e incluso imágenes ya sean estáticos o en movimiento.

Las pantallas de LEDs más avanzadas de estos días como la mostrada en la Figura 1.4., son capaces de mostrar imágenes a gran escala, en movimiento, a todo color y con una alta definición que pueden ser vistas por miles de personas. Con esta innovación se abre una nueva puerta en el mercado de anuncios publicitarios para dispositivos manejados electrónicamente gracias a los adelantos en los usos de los LEDs. Atrás quedaron los días en que no se veía ninguna aplicación práctica para el descubrimiento realizado en 1907 por un Ingeniero que trabajaba para Marconi, llamado HJ Round que notó que algunos cristales que

[4] EDS significa Electronic Digital Signage.
[5] OLED significa Organic Led Emitting Diode, LCD significa Liquid Crystal Display y CRT significa Cathode Ray Tube.

contenían galio emitían una luz cuando una corriente eléctrica pasaba a través de ellos y lo publicó en una revista de la época llamada *Electrical World*[6].

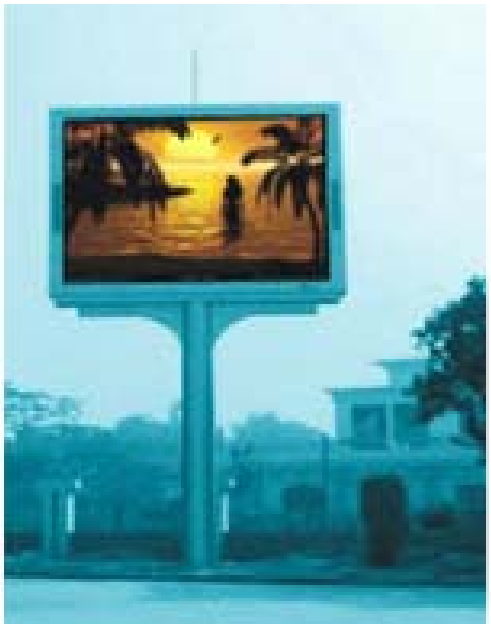


Figura 1.4. Pantalla de LEDs a color colocada en la vía pública

1.2. LEDs RGB.

Los LEDs RGB son LEDs especiales que tienen la capacidad de emitir los tres colores básicos: rojo, azul y verde dentro del mismo encapsulado lo cual significa un gran avance en la elaboración de estos elementos. Un ejemplo actual de LEDs RGB es el LED *RL5-RGB-D* con cubierta difusa fabricado y comercializado por la compañía *Super Bright Leds*.

Las características de cada color emitido por el LED *RL5-RGB-D* se presentan a continuación en las Tablas 1.1., 1.2. y 1.3. [7] En algunos casos cuando se quiere que el brillo de los LEDs se vea desde cualquier ángulo de vista y no solo de frente es preferible utilizar LEDs con cubiertas difusas que dirigen la luz emitida por el LED en todas direcciones.

[6] Información obtenida del artículo "The light emitting diode" que puede ser encontrado en la página de Inernet www.radio-electronics.com/info/data/semicond/led/light_emitting_diode.php.
[7] Las Tablas 1.1., 1.2. y 1.3. fueron obtenidas de la página de Internet www.superbrightleds.com.

Color: SUPER RED (AlGaIpn)				
Parámetros	**MAX**	**TYP**	**UNIT**	**TEST**
Angulo de vista a media potencia (claro)		15	Grados	
Voltaje directo	2.6	2.0	V	IF=20mA
Longitud de onda		630	nm	
Intensidad luminosa		1200	mcd	IF=20mA

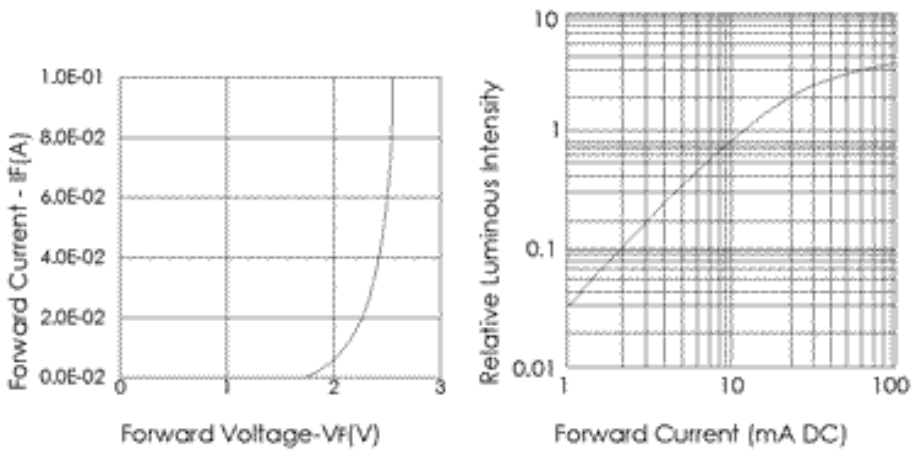


Tabla 1.1. Características del color rojo del RL5-RGB-D

Color: SUPER GREEN (GaInN/GaN))				
Parámetros	**MAX**	**TYP**	**UNIT**	**TEST**
Angulo de vista a media potencia (claro)		15	Grados	
Voltaje directo	4.0	3.5	V	IF=20mA
Longitud de onda		525	nm	
Intensidad luminosa		3700	mcd	IF=20mA

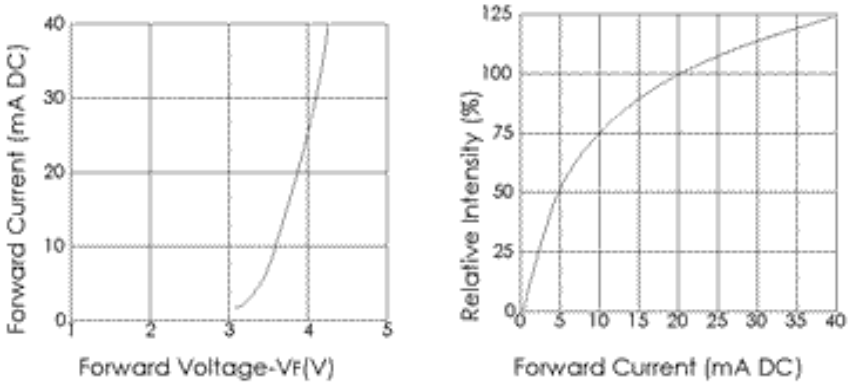


Tabla 1.2. Características del color verde del RL5-RGB-D

Color: SUPER BLUE (GaN)				
Parámetros	**MAX**	**TYP**	**UNIT**	**TEST**
Angulo de vista a media potencia (claro)		15	Grados	
Voltaje directo	4.0	3.5	V	IF=20mA
Longitud de onda		472	nm	
Intensidad luminosa		700	mcd	IF=20mA

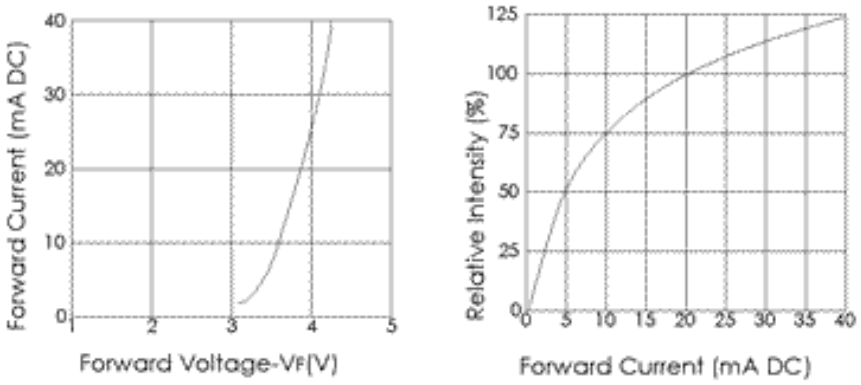


Tabla 1.3. Características del color azul del RL5-RGB-D

La estructura del *RL5-RGB-D* es similar a la de los LEDs normales con la diferencia de que tienen más terminales. A continuación se muestran en las Figuras 1.5. y 1.6. [8] los esquemas de dos LEDs RGB, uno con cátodo común y otro con ánodo común. Cuando se emplea microcontroladores es mejor utilizar LEDs con ánodo común para que la corriente utilizada por los LEDs no sea provista por el microcontrolador sino por la fuente directamente.

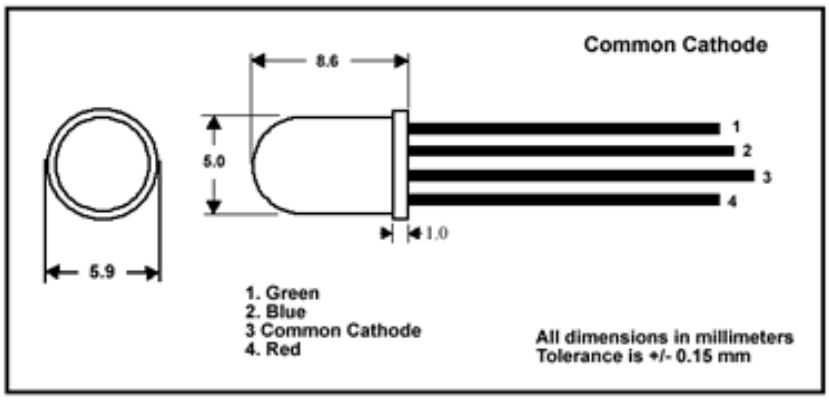


Figura 1.5. Esquema de un RL5-RGB-D con cátodo común

[8] Las Figuras 1.5. y 1.6. fueron obtenidas de la página de Internet www.superbrightleds.com.

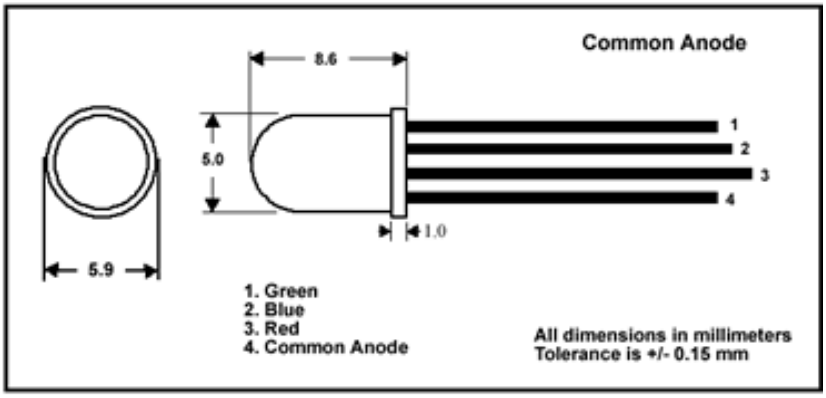


Figura 1.6. Esquema de un RL5-RGB-D con ánodo común

El hecho de que este tipo de LEDs tenga tantas terminales dificulta su uso cuando se trata de implementar grandes arreglos de LEDs como por ejemplo en una pantalla que despliega mensajes a través de LEDs encendidos y apagados de manera controlada.

Otra diferencia entre un LED RGB y uno normal es el precio ya que un LED RGB vale aproximadamente 10 veces más que un LED normal por lo que no están a la venta en nuestro mercado y si se quiere utilizarlos deben ser importados.

Estas razones justifican diseñar e implementar un dispositivo que permita mostrar mensajes luminosos utilizando la menor cantidad de LEDs RGB y una de las formas de lograr este objetivo es a través de un Display Rotativo que apenas utiliza una fila de ocho LEDs.

1.3. Matrices de LEDs.

Con el tiempo el formato de 7 segmentos se ha vuelto obsoleto ya que se limita a presentar su texto siempre de la misma manera. En este sentido un arreglo matricial de LEDs es más versátil ya que puede dibujar cada letra punto a punto cambiando su forma y estilo como se muestra en la Figura 1.7. Una matriz de LEDs inclusive es capaz de crear imágenes cuya resolución dependerá del tamaño y número de LEDs que contenga.

Figura 1.7. In-Car LED Message Bar

Si además se logra que cada punto de luz, conocido como píxel, contenga tres LEDs, cada uno con un color básico (rojo, verde y azul) se puede lograr una presentación multicolor de texto e imágenes como en la Figura 1.8.

Figura 1.8. LED Display multicolor

Controlando el tiempo y la manera en que los LEDs se encienden dentro de la matriz se puede dar muchos efectos cromáticos y de movimiento a la información presentada. Disminuyendo el tamaño de cada píxel y aumentando su número se puede conseguir una verdadera *pantalla* que si bien no tendrá la alta definición de la tecnología CRT o plasma, puede ser capaz de presentar imágenes en movimiento agradables a la vista y a todo color como se muestra en la Figura 1.9.

Figura 1.9. LED Message Ball

1.4. Pantallas de LEDs a color.

Debido a que el tamaño de los LEDs solo puede ser reducido hasta cierto límite, para que una pantalla de LEDs tuviera la resolución de una pantalla CRT o plasma, debería ser inmensa y ser vista desde muy lejos. Esto más que un inconveniente, es una ventaja en el caso de las vallas publicitarias. Construir pantallas CRT o plasma gigantes resulta costoso y poco práctico, por este motivo el mercado de las vallas publicitarias se ha visto satisfecho en gran manera con el surgimiento de las pantallas de LEDs que son una solución real para dicho mercado.

Como se puede apreciar en la Figura 1.10. las pantallas de LEDs están compuestas por miles de píxeles, cada uno de los cuales como ya se dijo anteriormente pueden contener varios LEDs (generalmente tres LEDs de colores: rojo, verde y azul)

Figura 1.10. Pantallas de LEDs apagadas

Su funcionamiento es igual al de cualquier otro tipo de pantalla, las imágenes se forman dependiendo del color que tome cada píxel. A corta distancia la imagen se ve igual que cuando uno se acerca demasiado a un televisor, sin embargo a una distancia lo suficientemente grande se puede apreciar una imagen de alta calidad como se puede ver en la Figura 1.11.

Figura 1.11. Anuncio publicitario mostrado en un Pantalla de LEDs

1.5. Usos de los LEDS Displays en la actualidad.

Como se ha podido apreciar el mercado de los LED Displays ha crecido bastante durante los últimos años y haciendo un recuento de sus aplicaciones no solo se usan en tableros o relojes como se hacía originalmente sino además se utilizan en vallas publicitarias, marcadores electrónicos en los estadios, en dispositivos de mensajes y entretenimiento visual tanto para el uso público y privado. Ahora Los LED Displays se utilizan hasta en gafetes electrónicos que muestran el nombre de la persona y alguna frase o identificación personal. Algunas de las aplicaciones de los LED Displays se presentan a continuación en la Figura 1.12.

<table>
<tr>
<td> 
a) Tableros de control</td>
<td> 
b) Displays de servicio a clientes</td>
</tr>
<tr>
<td> 
c) Displays para lugares públicos o privados</td>
<td> 
d) Displays para automotores</td>
</tr>
<tr>
<td> 
e) Vallas publicitarias</td>
<td> 
f) Gafetes personales</td>
</tr>
<tr>
<td> 
g) Entretenimiento visual para lugares públicos o privados</td>
<td> 
h) Relojes y alarmas</td>
</tr>
</table>

Figura 1.12. Aplicaciones de los LED Display

Así se puede apreciar en la Figura 1.12. que los LED Displays tienen las siguientes aplicaciones principalmente:

a) Tableros de control. Este se puede considerar el primer uso que se le dio a los LEDs, los cuales se utilizaban como luces indicadoras. Posteriormente también se comenzó a utilizar LED Displays para mostrar mensajes de texto que indicaban determinados valores como niveles, cantidad de productos, etc.

b) Displays de servicio a clientes. Los LED Displays siempre han sido muy comunes en lugares públicos o privados donde se atiende a varias personas. Los LED Displays generalmente eran utilizados en formato 7 segmentos aunque en la actualidad este formato ha sido reemplazado por los Displays matriciales. En este caso su uso es bastante simple, lo único que se necesita mostrar es el número del cliente a ser atendido y el número de ventanilla.

c) Displays para lugares públicos y privados. Los Displays matriciales son ahora utilizados no solo en salas de espera, sino también en otros lugares como tiendas y hoteles. Estos dispositivos son muy útiles para mostrar información o publicidad que requiere sor constantemente cambiada, así por ejemplo, las promociones vigentes en una tienda, mensajes de bienvenida o despedida para los huéspedes de un hotel, etc.

d) Displays para automotores. La ventaja de los LED Displays sobre otros dispositivos luminosos es que consumen poca corriente y trabajan a un bajo voltaje por lo que pueden ser utilizados hasta en vehículos. Así por ejemplo en las grandes ciudades se emplean en los buses de la transportación pública para mostrar su ruta de ida o de vuelta. Además también hay Displays disponibles para vehículos personales, si el conductor así lo desea para alguna función específica o simplemente por diversión.

e) Vallas publicitarias. El avance más importante en el uso de los LEDs es la creación de pantallas gigantes que pueden ser utilizadas como vallas publicitarias electrónicas. Las pantallas de LEDs superan en muchos aspectos a una valla publicitaria normal, se puede cambiar su contenido, hacerlas más llamativas y ser vistas en la noche con gran nitidez.

f) Gafetes personales. Estos gafetes muestran el nombre de la persona que lo está utilizando, su cargo en la empresa, número de identificación y hasta alguna frase personal que haya sido grabada.

g) Entretenimiento visual para lugares públicos y privados. Debido a que la matriz de LEDs de un Display ahora puede tomar varias formas, es posible crear Displays que muestran mensajes e imágenes de manera cilíndrica o circular. El uso de estos dispositivos no solo se limita a la estricta y sobria presentación de información sino también a entretener al espectador.

h) Relojes y alarmas. No se debe olvidar el uso más básico de los LEDs que es mostrar números como en el caso de la hora en un reloj o alarma. Sin embargo esta aplicación se ha visto innovada por los Displays Rotativos como el mostrado en la Figura 1.12.h que además utiliza una implementación mecánica para hacer girar los LEDs y así dar la impresión de que el mensaje está flotando en el aire.

1.6. Displays Rotativos.

El pionero de los mensajes con LEDs en movimiento rotativo es Bob Blick quien creó su primer *Propeller Clock* en 1996 el cual a través de 7 LEDs daba la ilusión de mostrar números en el aire y su objetivo era dar la hora[9]. En la Figura 1.13.[10] se puede apreciar como se veían los números desplegados por el *Propeller Clock* de Bob Blick.

Figura 1.13. Imagen mostrada por el Propeller Clock

[9] Información obtenida de la página de Internet "Propeller Clock" Mechanically Scanned LED Clock by Bob Blick. www.bobblick.com/techref/projects/ propclock/propclock.html

[10] Las Figuras 1.13., 1.14. y 1.15. son fotos que fueron obtenidas de la página de Internet "Propeller Clock" Mechanically Scanned LED Clock by Bob Blick. www.bobblick.com/techref/projects/ propclock/propclock.html

Para su proyecto Bob Blick utilizó un PIC 16C84 y un motor de VCR junto con otros elementos electrónicos básicos como resistencias y capacitores. La construcción del dispositivo no era muy complicada así como el programa creado para controlar los LEDs a través del PIC16C84. Ambos, el diseño del circuito y el código de programa son fáciles de conseguir a través del Internet[11] o pueden ser desarrollados por uno mismo una vez que se haya comprendido su principio de funcionamiento. A continuación en las Figuras 1.14. y 1.15. se muestran dos fotos del *Propeller Clock* detenido para poder apreciar como estaba construido.

Figura 1.14. Vista lateral del Propeller Clock

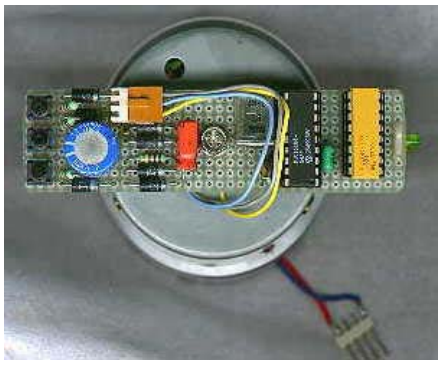


Figura 1.15. Vista Superior del Propeller Clock

Luego del proyecto realizado por Bob Blick muchos otros dispositivos similares que generalmente son denominados *Displays Rotativos* han sido desarrollados por otras personas ya sea con fines experimentales, didácticos o comerciales. Un ejemplo de un Display Rotativo que recientemente ha salido al mercado es el *Olimpia OLY-OL3000 Info Globe Caller ID Blue*[12] mostrado en la Figura 1.16. Este dispositivo tiene una presentación más completa de mensajes y además ofrece otras funciones además del reloj, como calendario, reconocimiento de llamadas, mensajes pregrabados, entre otras opciones.

[11] Para más información sobre el Propeller Clock hecho por Bob Blick se puede visitar la página de Internet www.bobblick.com/techref/projects/propclock/propclock.html

[12] Para mayor información sobre el Olimpia OLY-OL3000 Info Globe Caller ID Blue se puede ingresar a la página de Internet www.makophone.com/ololinglcaid.html o a cualquier otra página de telemercadeo.

Figura 1.16. Olimpia OLY-OL3000 Info Globe Caller ID Blue

1.7. Tendencias en el mercado de los avisos luminosos.

Debido al gran avance tecnológico en la fabricación de avisos luminosos, cada vez son menos utilizados los letreros que no contengan algún elemento lumínico, en el peor de los casos simples reflectores que iluminen el anuncio por la noche. De igual manera el uso de avisos de neón pronto quedará en el pasado debido a que brindan pocas posibilidades de configuración, por no decir ninguna ya que el usuario se tiene que conformar que el mensaje original formado por los delgados tubos de cristal que contienen los gases luminosos, si quisiera cambiar o dar movimiento al aviso tendría que emplear un complicado conjunto de tubos que se enciendan y apaguen sincrónicamente.

Entre todas las posibilidades electrónicas que existen en la actualidad, los LED Displays y las fibras ópticas son las más prometedoras. Sin embargo la tecnología en fibras ópticas está todavía en desarrollo y su uso en avisos luminosos se ve limitado debido a su poco brillo.

En la actualidad los LED Displays son la mejor opción debido al gran avance que han tenido durante los últimos años. Los avisos digitales como es el caso de los LED Displays tienen una amplia variedad de usos en restaurantes, tiendas por departamentos, presentaciones de negocios, salas de espera o de

conferencias en grandes edificios o centros de reuniones, hospitales, hoteles, teatros, casinos, aeropuertos, museos, escuelas y centros de aprendizaje.

Otra ventaja de los LED Display es que pueden ser colocados casi en cualquier sitio, en torres de anuncios a lo largo de la carretera, en marcadores deportivos en el centro o a uno de los extremos laterales de los estadios, alrededor de los estadios a manera de mensajes rotativos, en carteleras en medio del centro urbano, en mostradores o a la entrada de lugares públicos y hasta en vehículos como camiones o buses de transporte.

Además un hecho relevante a favor de los LED Displays es que nuestra sociedad se va acostumbrando cada vez más a ver imágenes en movimiento y no simples figuras estáticas sobre una superficie poco llamativa. Cada vez es más común el uso de Videos en muchas de las actividades cotidianas de cada individuo ya sea para entretenimiento, educación, comunicación o información.

Aunque actualmente de acuerdo a *CAP Ventures*, el total de dinero invertido por grandes compañías en anuncios publicitarios que utilizan avisos digitales es de 149 mil millones de dólares una cifra que apenas ocupa un 1% del mercado publicitario donde la TV, radio y revistas ocupan el 80%. El poderío de estos últimos medios mencionados está disminuyendo debido a que la gente ahora se ve más identificada con avisos más sofisticados, mostrados en lugares más llamativos y enfocados a las necesidades y gustos de cada persona[13]. Así por ejemplo una propaganda televisiva de un artículo deportivo al ser transmitida por un determinado canal a una determinada hora (sin considerar la transmisión de un evento deportivo), aunque llegue a millones de personas, no todas las personas que la vean estarán interesadas en deportes. Es mejor si se anuncia el mismo producto en un estadio donde todas las personas que asisten sienten una afinidad por los deportes y además si el evento a realizarse en dicho estadio es televisado

[13] Información obtenida del artículo “Dynamic Digital Signage - The New Age of Advertising” escrito por Yvonne Li & Greg Gilbert, ADurance Inc que puede ser encontrado en la página de Inernet www.signindustry.com/led/articles/2005-05-02-DigitalSignageNewAge.php3

el alcance de la propaganda es mayor y llegará a un público interesado en el producto anunciado como se muestra a continuación en la Figura 1.17[14].

Figura 1.17. Anuncio de GMC en el Pepsi Arena Center. Utilizando el SurroundVision puesto en el mercado por YESCO. El anuncio publicitario *gira* y se muestra alrededor de todo el estadio.

[14] La Figura 1.17 es una foto obtenida de la página de Internet www.signindustry.com/led/articles/2003-08-29-LEDConcourse.php3

CAPÍTULO 2

MICROCONTROLADORES AVR. ATMEGA32

2. MICROCONTROLADORES AVR.

Los microcontroladores permiten realizar muchas aplicaciones. Estos chips básicamente son pequeños computadores programables que pueden cumplir casi cualquier función imaginable lo único que los limita es su velocidad de procesamiento y capacidad de memoria que no son tan amplias como las de un computador personal, sin embargo pueden satisfacer necesidades más específicas. En la actualidad además se utilizan microcontroladores para los más diversos proyectos como por ejemplo sistemas de sensores IR de movimiento, sistemas de visión 3D para robots, sistemas de audio avanzados, sistemas de acceso y manejo de la red, lectores de discos duros, lectores de MP3s, sistemas de sensores de temperatura y sistemas de anuncios publicitarios digitales como los LED Displays entre otros.

En la actualidad el mercado está dominado por el microcontrolador PIC de la compañía Microchip. Los más conocidos son el PIC16F84 y el PIC16F877 siendo este último de mayor capacidad de memoria y mayor velocidad de procesamiento. Existen muchos programas, como es el caso de MPLAB y PIC C, que son utilizados para desarrollar proyectos para este tipo de microcontroladores y además hay circuitos grabadores de PICs de fácil implementación que están a la disposición del usuario.

Aunque los PICs son más conocidos en nuestro mercado, también hay otras opciones no muy conocidas como es el caso de los microcontroladores ATmega de la compañía Atmel. Los microcontroladores más conocidos y modernos de esta compañía son el ATtiny13, ATmega16, ATmega32 y ATmega128.

Haciendo una comparación como la mostrada en la Tabla 2.1[1] de las características más notables en cuanto la cantidad de memoria disponible y la velocidad de procesamiento de microcontroladores similares en precio y capacidad de la Microchip y de la Atmel se puede apreciar que los microntroladores ATmega ofrecen más ventajas y un menor costo.

Características	PIC16F877	ATmega8	ATmega16	ATmega32
Número de instrucciones disponibles	35	131	131	131
Max. I/O pines	32	23	32	32
Velocidad de procesamiento	5 MIPS[2] a 20 MHz	16 MIPS a 16 MHz	16 MIPS a 16 MHz	16 MIPS a 16 MHz
Capacidad de Memoria Flash (programa)	8 KB	8 KB	16 KB	32 KB
Capacidad de Memoria RAM (datos)	368 Bytes	1 KB	1 KB	2 KB
Capacidad de EEPROM (datos)	256 Bytes	512 Bytes	512 Bytes	1 KB
Precio	$9,49 Jameco	$3,66 Jameco	$6,25 Jameco	$11,95 Kanda

Tabla 2.1 Comparación entre PIC16F877, ATmega8, ATmega16 y ATmega32

Por esta razón es necesario explotar más el uso de los microcontroladores de la Atmel en nuestro mercado. Además es necesaria la búsqueda y familiarización con Software y Hardware que se pueda utilizar en el desarrollo de proyectos para microcontroladores Atmel.

[1] Información obtenida de los manuales de los microcontroladores citados. Los precios fueron obtenidos de los sitios de Internet de venta de elementos electrónicos JAMECO y KANDA.

[2] MIPS significa Million Instructions Per Second y es la medición de la velocidad y el poder de un procesador a través del número de instrucciones de máquina que puede ejecutar en un segundo

2.1. Características principales de los microcontroladores AVR.

Un AVR (Advanced Virtual RISC)[3] es un microcontrolador de 8 bits y representa la propuesta más avanzada de la Atmel en esta área. Los microcontroladores AVR tienen todas las características de los microcontroladores modernos como los conocidos PICs. Tienen una arquitectura Harvard, un reloj interno, escalabilidad y manejo de señales analógicas, temporizadores e interrupciones externas.

Emplear una arquitectura Harvard significa que tienen el bus de datos separado del bus de instrucciones lo cual les da una mayor velocidad de procesamiento de instrucciones.

Además los microcontroladores AVR tienen un oscilador interno por lo que pueden trabajar con un reloj interno o externo. Esta cualidad puede ser muy útil en aplicaciones que tienen poco espacio dentro del circuito como para agregar un oscilador externo o en el caso que se necesite variar la frecuencia del reloj interno solo bastaría con hacer ciertas modificaciones en el ATmega en el momento de grabar el código de programa en vez de cambiar de cristal, con todas las complicaciones que eso implica.

Otra cualidad de los microcontroladores AVR es que son escalables, es decir, se puede comenzar las aplicaciones con microcontroladores AVR de pequeña capacidad y si se requiere incrementar las funciones de la aplicación, se puede emigrar a un microcontrolador AVR de mayor capacidad sin problema ya que la arquitectura y la forma en que se realiza la configuración y programación del microcontrolador se mantiene igual.

Los microcontroladores AVR además tienen la capacidad de manejar señales analógicas, temporizadores, contadores e interrupciones externas entre otras

[3] RISC significa Reduced Instruction Set Computer. Son procesadores que contienen una colección reducida de instrucciones que les permite lograr una velocidad de procesamiento de intrucciones más alta. AVR también significa Alf Vegard RISC en referencia a sus creadores Alf Egil Bogen y Vegard Wollan.

cualidades. Existen diferentes tipos de microcontrolador AVR en el mercado con distintas capacidades de memoria, número de puertos y características especiales.

En la actualidad inclusive han salido nuevas versiones del microcontrolador AVR que han reemplazado a las antiguas versiones así por ejemplo la caduca sub-familia AT90 ha sido reemplazada por la sub-familia ATmega con sus modelos ATmega16, ATmega32 y ATmega128.

Los AVR-ATmega tienen bastantes cualidades que los convierten en una buena opción al momento de escoger un microcontrolador. En primer lugar ofrecen gran velocidad de procesamiento, 1 MIPS por MHz, debido a su arquitectura RISC que les permite ejecutar una instrucción por ciclo y a sus 32 Registros para múltiples propósitos.

Tienen un bajo consumo de energía. Operan con voltajes que pueden oscilar entre 1.8[4] y 5.5V. Tienen varios tipos de *Sleep Modes*: *Idle, ADC Noise Reduction, Power-save, Power-down, Standby* y *Extended Standby* que se ajustan a cada necesidad del usuario. Permiten también distintos tipos de configuración en lo que se refiere al manejo de eventos en tiempo real y a la manera en que se realiza el *Wake-up*.

Aparte de los microcontroladores ATmega, la Atmel también produce los microcontroladores ATtiny diseñados para satisfacer las necesidades de aplicaciones pequeñas que no requieren de gran capacidad de memoria o de funciones complicadas. Los modelos más comunes son el ATtiny13 y el ATtiny26 que apenas tienen 1 y 2 Kbytes de memoria para programación respectivamente.

Los microcontroladores AVR puede ser programados aún estando dentro del circuito de la aplicación a través de un ISP[5] y poseen la capacidad de auto-

[4] 1.8V es el voltaje mínimo de trabajo solo para ciertos AVR-ATmega como por ejemplo el ATmega48, el ATmega88 y el ATmega1280. Para el ATmega8, el ATmega16 y el ATmega32 el voltaje mínimo de trabajo es de 2.7V. Para mayor información ver las Tablas 2.2. y 2.3.
[5] ISP significa In-System Programmer

programarse comunicándose con una central a través de ciertos pines especiales cuando es necesario, como por ejemplo cuando una actualización es requerida. Todas estas cualidades van encaminadas a evitar tener que extraer el microcontrolador del circuito ya que con una constante manipulación el chip se puede dañar a largo plazo.

Además Atmel proporciona varias herramientas para el desarrollo de proyectos basados en microcontroladores AVR como por ejemplo el Software de libre distribución AVR Studio que contiene las herramientas básicas de edición, compilación y simulación de programas escritos para microcontroladores AVR y el kit de desarrollo STK500 que posee varios módulos de prueba e interfases capaces de simular cualquier ambiente en el cual se quiere que el microcontrolador se desenvuelva.

A continuación se presenta las Tablas 2.2. y 2.3.[6] donde se puede apreciar las características principales de los microcontroladores AVR pertenecientes a las sub-familias ATmega y ATtiny, tales como su capacidad de memoria, número de pines I/O, frecuencia de trabajo y número de Timers e interrupciones externas que poseen.

[6] Las Tablas 2.2. y 2.3. fueron obtenidas de la página de Internet de la Atmel: www.atmel.com.

Características de los AVR-ATmega									
Device	**Flash (Kbytes)**	**EEPROM (Kbytes)**	**SRAM (Bytes)**	**Max I/O Pins**	**F.max (MHz)**	**Vcc (V)**	**16-bit Timer**	**8-bit Timer**	**Ext Interr**
AT90PWM2	8	0.5	512	19	16	2.7-5.5	1	1	4
AT90PWM3	8	0.5	512	19	16	2.7-5.5	1	1	4
ATmega128	128	4	4096	53	16	2.7-5.5	2	2	8
ATmega1280	128	4	8192	86	16	1.8-5.5	4	2	32
ATmega1281	128	4	8192	51	16	1.8-5.5	4	2	32
ATmega16	16	0.5	1024	32	16	2.7-5.5	1	2	3
ATmega162	16	0.5	1024	35	16	1.8-5.5	2	2	3
ATmega165	16	0.5	1024	54	16	1.8-5.5	1	2	17
ATmega168	16	0.5	1024	23	20	1.8-5.5	1	2	26
ATmega169	16	0.5	1024	53	16	1.8-5.5	1	2	17
ATmega2560	256	4	8192	86	16	1.8-5.5	4	2	32
ATmega2561	256	4	8192	51	16	1.8-5.5	4	2	32
ATmega32	32	1	2048	32	16	2.7-5.5	1	2	3
ATmega325	32	1	2048	53	16	1.8-5.5	1	2	17
ATmega3250	32	1	2048	68	16	1.8-5.5	1	2	17
ATmega329	32	1	2048	53	16	1.8-5.5	1	2	17
ATmega3290	32	1	2048	68	16	1.8-5.5	1	2	32
ATmega406	40	0.512	2048	18	1	abr-25	1	1	4
ATmega48	4	0.256	512	23	20	1.8-5.5	1	2	26
ATmega64	64	2	4096	53	16	2.7-5.5	2	2	8
ATmega640	64	4	8192	86	16	1.8-5.5	4	2	32
ATmega645	64	2	4096	53	16	1.8-5.5	1	2	17
ATmega6450	64	2	4096	68	16	1.8-5.5	1	2	17
ATmega649	64	2	4096	53	16	1.8-5.5	1	2	17
ATmega6490	64	2	4096	68	16	1.8-5.5	1	2	32
ATmega8	8	0.5	1024	23	16	2.7-5.5	1	2	2
ATmega8515	8	0.5	512	35	16	2.7-5.5	1	1	3
ATmega8535	8	0.5	512	32	16	2.7-5.5	1	2	3
ATmega88	8	0.5	1024	23	20	1.8-5.5	1	2	26

Tabla 2.2. Características de los AVR-ATmega

Características de los AVR-ATiny									
Device	**Flash (Kbytes)**	**EEPROM (Kbytes)**	**SRAM (Bytes)**	**Max I/O Pins**	**F.max (MHz)**	**Vcc (V)**	**16-bit Timer**	**8-bit Timer**	**Ext Interr**
ATtiny11	1			6	6	2.7-5.5		1	1
ATtiny12	1	0.0625		6	8	1.8-5.5		1	1
ATtiny13	1	0.064	64B+32 reg	6	20	1.8-5.5		1	6
ATtiny15L	1	0.0625		6	1.6	2.7-5.5		2	1(+5)
ATtiny2313	2	0.128	128	18	20	1.8-5.5	1	1	2
ATtiny25	2	0.128	128	6	20	1.8-5.5		2	7
ATtiny26	2	0.125	128	16	16	2.7-5.5		2	1
ATtiny28L	2		32	11	4	1.8-5.5		1	2(+8)
ATtiny45	4	0.256	256	6	20	1.8-5.5		2	7
ATtiny85	8	0.512	512	6	20	1.8-5.5		2	7

Tabla 2.3. Características de los AVR-ATiny

Como se puede apreciar en las Tablas 2.1. y 2.2. existe una amplia gama de microcontroladores AVR-ATmega y AVR-ATiny que son capaces de ajustarse a las necesidades específicas de cada aplicación determinada por el usuario desde el poderoso ATmega2560 hasta el práctico ATiny11.

En la numeración colocada sobre los chips ATmega su información está plenamente mostrada de manera que se puede saber sus características principales como su tipo, fuente de poder, velocidad, tipo de empaquetado y rango de temperatura con un simple vistazo. A continuación se muestra en la Figura 2.1.[7] la manera en que se debe leer la numeración de un microcontrolador ATmega.

[7] La Figura 2.1. está basada en la información obtenida del manual del ATmega32, capítulo: Packing Information

ATmega XXX X - X X X

XXX	X	X	X	X
Tipo	**Fuente de Poder**	**Velocidad**	**Tipo de Empaquetado**	**Rango de Temperatura**
8, 48, 8515, 8535 = 8kx8 In-System programmable FLASH	Vacío = 4.5 a 5.5 V	8 = 8 MHz	A = TQFP[1]	C = Comercial
16, 161, 162, 163, 169 = 16kx8 In-System programmable FLASH	L = 2.7 a 5.5 V	16 = 16 MHz	P = Plástico DIP[2]	I = Industrial
32, 323 = 32kx8 In-System programmable FLASH	V = 1.8 a 3.6 V		J = PLCC[3]	
64 = 64kx8 In-System programmable FLASH			M = MLF[4]	
103, 128 = 128kx8 In-System programmable FLASH				

[1] TQFP significa Thin Quad Flat Pack
[2] DIP significa Dual In-line Package
[3] PLCC significa Plastic Leadless Chip Carrier
[4] MLF significa Micro Lead Frame

Figura 2.1. Características de los AVR-ATmega

2.2. Microcontrolador ATmega32.

El ATmega32 es un microcontrolador AVR de 8 bits de alto rendimiento y bajo consumo de energía. Su avanzada Arquitectura RISC maneja 131 instrucciones en lenguaje Assembly y tiene una velocidad de procesamiento de hasta 16 MIPS cuando está configurado a 16 MHz que es su frecuencia máxima (solo alcanzada con reloj externo). También hay en el mercado el ATmega32L que solo llega hasta 8 MHz.

El ATmega32 tiene tres tipos de memoria: Una memoria Flash autoprogramable de 32 KB para programación que puede resistir hasta 10.000 ciclos de lectura/escritura, una memoria SRAM de 2 KB para datos generados durante la ejecución del programa, y una EEPROM de 1 KB para datos que permanecen guardados aún cuando el microcontrolador esté apagado que igualmente puede resistir hasta 10.000 ciclos de lectura/escritura. Cada una de

estas memorias puede ser bloqueada independientemente a través de los Bits de Configuración y Seguridad.

En cuanto a sus otras características principales se puede decir que el microcontrolador ATmega32 posee cuatro Puertos de 8 bits denominados PORTA, PORTB, PORTC y PORTD que ofrecen 32 pines I/O. ya que cada pin de dichos puertos puede ser configurado individualmente como de entrada o de salida.

Este microcontrolador también tiene tres Interrupciones Externas INT0, INT1 e INT2 y tres Timers, el Timer0 de 8 bits, el Timer1 de 16 bits y el Timer2 de 8 bits cada uno de los cuales posee su propio prescaler. El Timer1 asimismo tiene la opción de Captura de Datos que permite medir el tiempo transcurrido entre eventos que ocurren fuera del microcontrolador.

A esto se le suma las cualidades ya mencionadas anteriormente en cuanto al consumo de energía, el reloj interno y la programación mediante sistemas ISP que son comunes entre todos los microcontroladores ATmega.

A continuación en la Figura 2.2[8] se muestra la configuración de pines del microcontrolador ATmega32 en el encapsulado 40-pin PDIP[9] utilizado en circuitos de ensayo ya que puede ser insertado en zócalos o protoboards. En esta figura se puede apreciar los pines asignados para VCC, GND, el reloj externo y los Puertos I/O con sus funciones especiales entre paréntesis.

[8] La Figura 2.2. fue obtenida del manual del ATmega32, capítulo: Pin Configurations
[9] PDIP significa Plastic Dual In-line Package

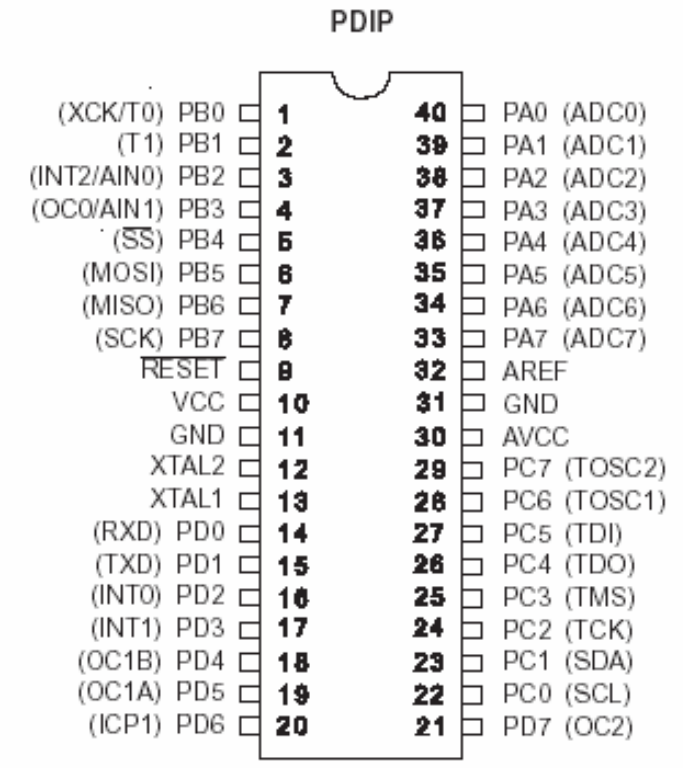


Figura 2.2. Configuración de Pines del ATmega32

2.3. Consumo de energía del microcontrolador ATmega32.

El consumo de energía del microcontrolador ATmega32 se puede calcular en base al voltaje Vcc y a la frecuencia de trabajo del microcontrolador. Para esto se utiliza las gráficas de las Figuras 2.3. y 2.4.[10] que muestra la relación existente entre estos dos parámetros. Como se puede apreciar en la gráfica, el consumo de energía del microcontrolador ATmega32 aumenta en relación directa al voltaje Vcc y a la frecuencia de trabajo. Su consumo mínimo cuando Vcc = 3V y la frecuencia de trabajo es 1 MHz es de 1,1mA[11]. Su consumo máximo cuando Vcc = 5.5V y la frecuencia de trabajo es 16 MHz es de más de 25 mA aproximadamente.

[10] Las Figuras 2.3. y 2.4. fueron obtenidas del manual del ATmega32, capítulo: Electrical Characteristics
[11] Información obtenida del manual del ATmega32, capítulo: Features.

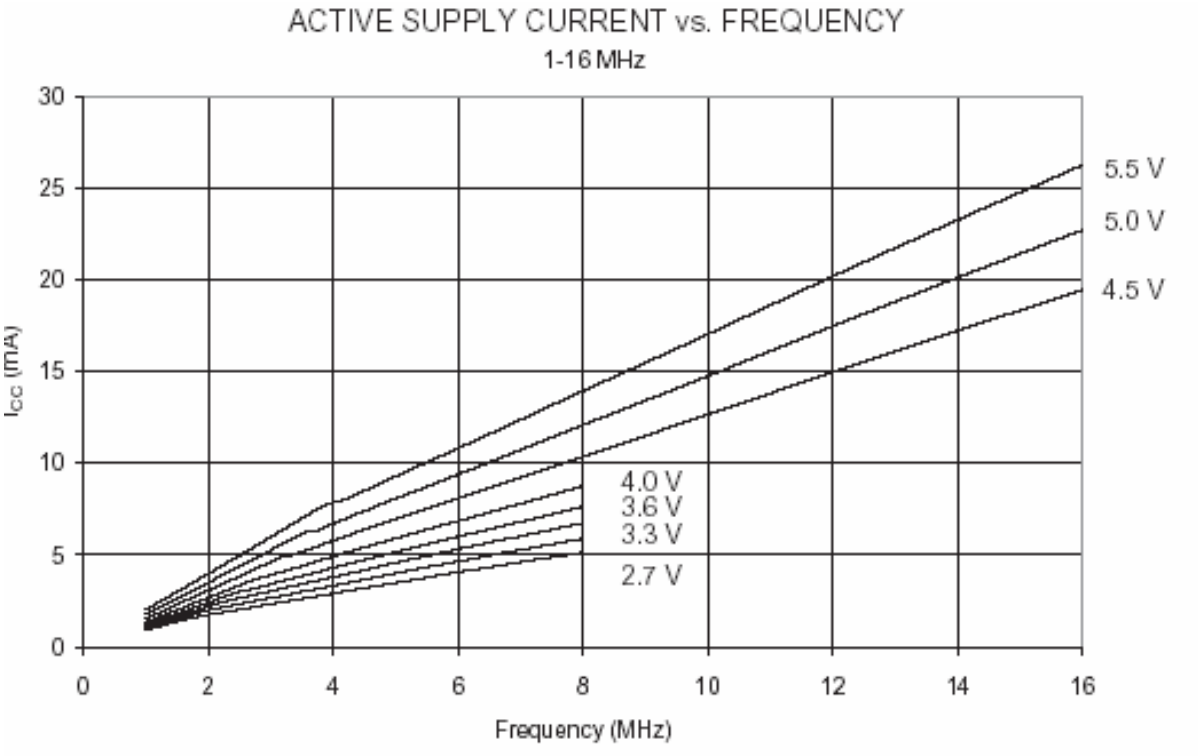


Figura 2.3. Consumo de corriente vs. Frecuencia en el ATmega32 en modo Activo

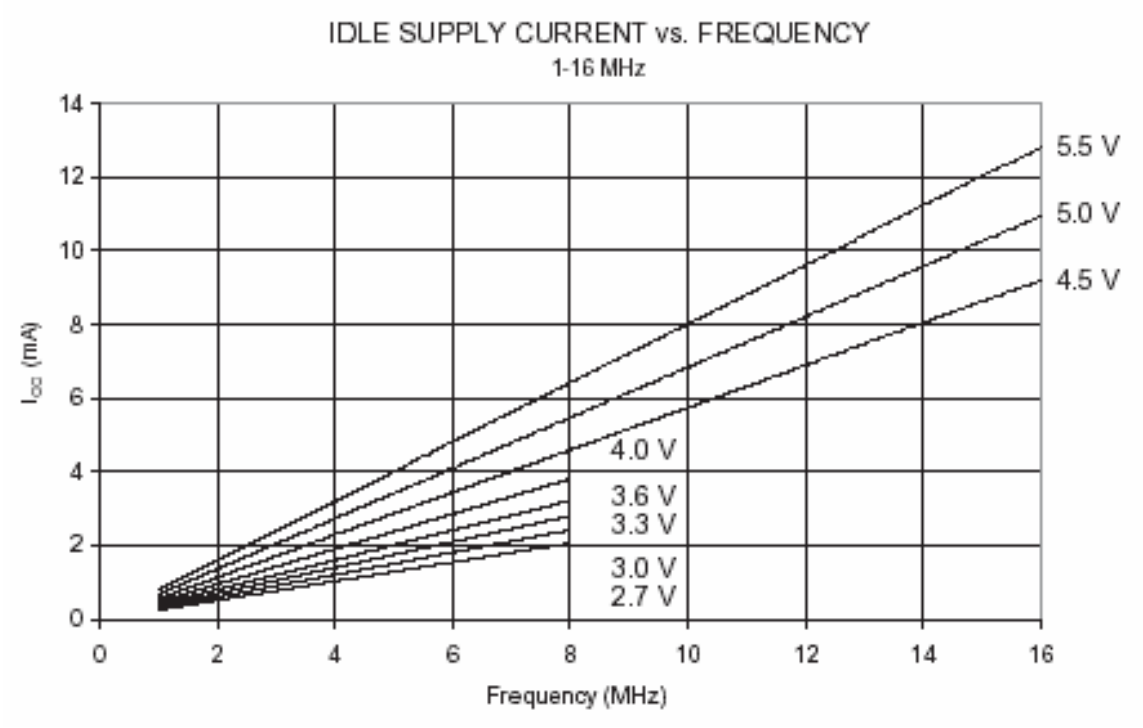


Figura 2.4. Consumo de corriente vs. Frecuencia en el ATmega32 en modo *Idle*

Asimismo utilizando las gráficas de las Figuras 2.3. y 2.4. se puede establecer que el valor medio de consumo de corriente del microcontrolador ATmega32 trabajando a un voltaje Vcc = 5V y a una frecuencia de 8 MHz es de 12 mA en modo Activo y 5 mA en modo *Idle* aproximadamente.

Además se debe considerar que cada puerto I/O en DC maneja máximo hasta 40 mA[12]. De esta manera, al implementar un circuito se puede determinar cuanta corriente consumirá el circuito en total sumando el consumo propio del microcontrolador y el consumo de cada puerto.

2.4. Reloj del sistema.

Todos los microcontroladores ATmega vienen con un oscilador interno incluido que puede ser utilizado como reloj del sistema. Además si se desea se puede utilizar una fuente externa de sincronismo como por ejemplo un cristal conectado a los pines XTAL1 y XTAL2 del microcontrolador.

La configuración del Reloj no se hace dentro del programa sino con los Bits de Configuración y Seguridad. Los bits que determinan la fuente de reloj son 4 y se denominan *CKSEL* (Clock Selection). Se debe tener en consideración que en el programa grabador utilizado en este proyecto cuando estos bits tienen un visto en su respectivo *checkbox* de configuración, significa que están activados y esto se representa con un cero y no con un uno como se podría creer en un principio.

Las opciones de reloj que existen son *External Cristal/Ceramic Resonador, External Low-frequency Cristal, External RC Oscilador. Calibrated Internal RC Oscilador* y *External*. Cada opción a su vez comprende un rango de valores que varían según la frecuencia de trabajo escogida. Esto se puede apreciar en la Tabla 2.4. a continuación.

[12] Información obtenida del manual del ATmega32, capítulo: I/O Ports.

Device Clocking Option	CKSEL3..0
External Crystal/Ceramic Resonator	1111 - 1010
External Low-frequency Crystal	1001
External RC Oscillator	1000 - 0101
Calibrated Internal RC Oscillator	0100 - 0001
External Clock	0000

Tabla 2.4. Opciones de Reloj

Así por ejemplo en el caso de la opción *Calibrated Internal RC Oscillator* el valor de los Bits *CKSEL* debe estar en el rango 0100 - 0001. Para cambiar la frecuencia del reloj interno se deben configurar los bits *CKSEL* como se indica en las Tablas 2.5.[13] que se muestra a continuación.

CKSEL3..0	Nominal Frequency (MHz)
0001(1)	1.0
0010	2.0
0011	4.0
0100	8.0

Note: 1. The device is shipped with this option selected.

Tabla 2.5. Frecuencias de operación del oscilador interno RC

Además están los 2 bits *SUT* (Start-up Time) mostrados en la Tabla 2.6. que determinan el retardo o tiempo que se deja pasar luego de energizar el ATmega32 para que éste comience a funcionar y así darle mayor estabilidad.

[13] Las Tablas 2.4., 2.5. y 2.6. fueron obtenidas del manual del ATmega32, capítulo: System Clock and Clock Options

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	6 CK	–	BOD enabled
01	6 CK	4.1 ms	Fast rising power
10(1)	6 CK	65 ms	Slowly rising power
11	Reserved		

Note: 1. The device is shipped with this option selected.

Tabla 2.6. Tiempos de inicio para el oscilador interno RC

Así los 4 bits *CKSEL* del microcontrolador ATmega32 están configurados de fábrica como “0001”, es decir, los tres primeros están con visto (activados) y el último no tiene visto (desactivado). Bajo esta configuración el microcontrolador ATmega32 está establecido con un reloj interno de 1 MHz. Los 2 bits *SUT* están configurados “10” para un retardo lento y dar suficiente tiempo al microcontrolador para estabilizarse.

2.5. Configuración y Manejo de Puertos I/O.

Como fue mencionado anteriormente, el microcontrolador ATmega32 tiene 32 pines I/O (de *Input/Output*) agrupados en cuatros puertos: PORTA, PORTB, PORTC y PORTD. Cada pin tiene una funcionalidad Read-Modify-Write, es decir, puede ser programado como de entrada o de salida sin afectar los otros pines del mismo puerto. Además cada uno tiene una resistencia de Pull-up para protección del puerto que también puede ser habilitada o deshabilitada individualmente. El buffer de salida de cada puerto tiene la capacidad de manejarse simétricamente como si fuese una fuente o un drenaje, es decir, dando o recibiendo corriente. Además los pines de salida son lo suficientemente resistentes a situaciones extremas para manejar LED Displays directamente ya que tienen diodos conectados a VCC y tierra como se muestra en la Figura.

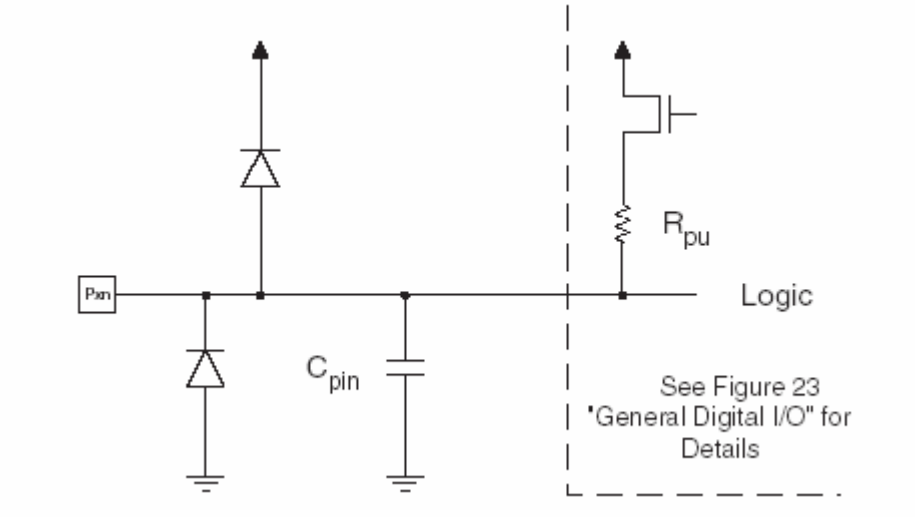


Figura 2.5. Esquema equivalente de un Pin I/O

Los puertos se representan de manera general con una “x” y cada pin del puerto con una “n”, así de manera genérica su representación puede ser PORTx para puertos y PORTxn para pines, o simplemente Pxn. Así por ejemplo PORTB3 o PB3 significa el pin 3 del puerto B.

Cada puerto tiene tres registros relacionados: el Registro de Datos PORTx, el Registro de Dirección de Datos (Data Direction Register) DDRx y el Registro del Pin de Puerto de Entrada (Port Input Pins) PINx. PINx es un registro de solo lectura mientras que los otros dos son de lectura y escritura.

La mayoría de puertos están multiplexados con funciones alternas especiales que se describirán a continuación. En algunos casos, al habilitar la función especial de los pines, no se altera el uso de los otros pines en el puerto como pines digitales de uso general I/O.

2.5.1. Configuración de Puertos.

Cada uno de los ocho pines de los cuatro puertos del microcontrolador ATmega pueden ser configurados de cuatro modos distintos en cualquier momento durante la ejecución del programa cargado en el microcontrolador, los modos son:

- IN, *Tri-State* (Tres Estados)
- IN, *Pull-Up*
- OUT, Cero
- OUT, Uno

Estas 4 configuraciones se logran modificando los Registros PORTxn y DDRxn. Primero se configura PORTxn y luego DDRxn. PORTxn indica el tipo de entrada del pin en IN o el estado inicial del pin en OUT. DDRxn indica la dirección del pin del puerto si es IN ó OUT.

Si PORTxn está en bajo (cero) y luego se pone DDRxn en bajo (cero) el pin está configurado como IN Tres Estados.

```
PORTA.1=0;
DDRA.1=0;
```

Si PORTxn está en alto (uno) y luego se pone DDRxn en bajo (cero) el pin está configurado como IN Pull-Up.

```
PORTA.1=1;
DDRA.1=0;
```

Si PORTxn está en bajo (cero) y luego se pone DDRxn en alto (uno) el pin está configurado como OUT y su valor inicial es cero.

```
PORTA.1=0;
DDRA.1=1;
```

Si PORTxn está en alto (uno) y luego se pone DDRxn en alto (uno) el pin está configurado como OUT y su valor inicial es uno.

```
PORTA.1=1;
DDRA.1=1;
```

En los ejemplos anteriores se configuraron solo el primer pin del Puerto A aunque se podría configurar los demás de igual manera o todos a la vez solo utilizando la palabra PORTA sin especificar ningún pin.

2.5.2. Manejo de Puertos.

Para leer o escribir información desde y hacia los puertos es necesario utilizar correctamente los Registros relacionados a cada puerto, estos Registros son PINx y PORTx. A su vez los puertos pueden ser manejados básicamente en tres modos: modo IN, modo OUT y modo especial.

Puertos en Modo IN

Cuando los puertos están en modo IN, el puerto debe ser leído a través del Registro PINx y no a través del Registro PORTx como se podría suponer en un principio. Esto se debe a que el Registro PINx y su precedente *latch* constituyen un sincronizador que evitan la *metastabilidad*[14] si el valor físico del pin cambia cerca de un flanco del reloj interno, pero esto introduce un retraso que puede ser entre 0,5 y 1,5 ciclos del reloj del sistema.

Puertos en Modo OUT

Cuando los puertos están en Modo OUT, el puerto debe ser escrito a través del Registro PORTx. El Registro PINx guarda el valor de PORTx en el siguiente ciclo de reloj. Aunque ambos Registros tienen el mismo valor es recomendable utilizar el Registro PORTx para las operaciones internas del programa, ya que este es de lectura y escritura, en cambio el Registro PINx es solo de lectura y su uso está reservado para el modo IN.

[14] Metastabilidad es cuando se tiene un estado de no equilibrio por un largo periodo de tiempo.

Usos especiales de los Puertos.

Los pines de los puertos en modo digital a parte de ser utilizados simplemente como de entrada o de salida, también pueden tener usos especiales relacionados con los Timers, Relojes externos y Módulos de depuración como el JTAG.

Un ejemplo es el pin PC2 que está deshabilitado de fábrica debido a que se utiliza con el Módulo JTAG. Para poder utilizar este pin hay que deshabilitar el bit de control JTAGEN.

Otro ejemplo es el pin PD6 también llamado ICP (Input Capture Pin) que captura los cambios de nivel o los flancos que activan el Timer1 cuando está en modo de captura de datos. También se tiene el pin PD2 que activa por flancos la interrupción externa INT0.

Además los pines PB7/SCK, PB6/MISO y PB5/MOSI son utilizados para grabar el programa al microcontrolador a través de un SPI[15] por lo que deben estar libres al momento de grabar. Para mayor información sobre los usos especiales de los pines de los puertos del microcontrolador ATmega32 se recomienda revisar el manual del ATmega32[16].

2.6. Manejo de Registros.

Los microcontroladores AVR poseen Registros para funciones específicas como por ejemplo el TCCR0 encargado de configurar e inicializar el Timer0 y Registros de propósito múltiple que pueden ser utilizados libremente en el programa creado por el usuario. La mayoría de los Registros Específicos se caracterizan por ser de 8 bits, sin embargo existen Registros Especiales que ocupan un par de Registros de 8 bits, es decir, 16 bits en total.

[15] SPI significa System Programming Interface
[16] Manual del ATmega32, capítulo: I/O Ports, Alternate Port Functions

2.6.1. Registros de propósito multiple.

Las variables generadas en un programa son guardadas en los 32 Registros de propósito múltiple de 8 bits que tiene el ATmega32 para uso general que van desde el Registro R0 hasta el R31. Los primeros 16 Registros son para Variables Globales y los 16 Registros restantes son para Variables Locales. Si hiciera falta más variables se las coloca en la SRAM.

Cuando una variable utiliza 8 bits como en el caso de las variables tipo *char*, ésta es guardada en un solo Registro de 8 bits. Cuando una variable utiliza 16 bits es guardada en dos Registros de 8 bits.

2.6.2. Registros especiales de 16 bits.

Algunos registros especiales son de 16 bits como el contador del Timer1 TCNT1. En este caso se lo divide en dos registros TCNT1L y TCNT1H, bajo y alto respectivamente. Este tipo de registros se manejan de una manera especial ya que siempre deben seguir una secuencia de lectura y escritura.

Según el manual del ATmega32, para escribir un Registro de 16 bits, primero se debe escribir el Registro Alto y luego el Bajo. Para leer un Registro de 16 bits, primero se debe leer el Registro Bajo y luego el alto. Si no se sigue esta secuencia, la lectura o escritura da 0x00 o valores erróneos.

Secuencia de Lectura

```
Var1 = TCNT1L;
Var2 = TCNT1H;
```

Secuencia de Escritura

```
TCNT1H = 0x3F;
TCNT1L = 0x2E;
```

2.7. Interrupciones Externas.

El microcontrolador ATmega32 posee 3 fuentes de Interrupción Externa, activadas por los pines INT0 (PD2), INT1 (PD3) e INT2 (PB2) respectivamente. Cuando las Interrupciones Externas están habilitadas las mismas se ejecutan aún cuando los pines estén configurados como de salida.

La interrupción puede ser activada por un flanco ascendente o descendente o por un nivel bajo (INT2 solo se activa por flancos). Para INT0 e INT1 esta configuración se puede especificar en el Registro MCUCR (MCU Control Register), mostrado en la Figura 2.6.[17], en los bits 0, 1, 2 y 3 llamados ISC (Interrupt Sense Control). ISC00 e ISC01 configuran INT0 mientras que ISC10 e ISC11 configuran INT1 como se puede apreciar en las Tablas 2.7. y 2.8. [18]

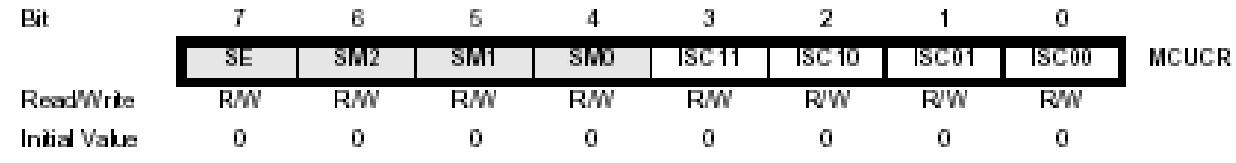


Figura 2.6. Registro MCUCR

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

Tabla 2.7. Bits de Control de INT1

[17] La Figura 3.7.1 fue obtenida del manual del ATmega32, capítulo: External Interrupts
[18] Las Tablas 3.7.1 y 3.7.2 fueron obtenidas del manual del ATmega32, capítulo: External Interrupts

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

Tabla 2.8. Bits de Control de INT0

Para INT2 su configuración se hace a través del bit 6 del Registro MCUCSR (MCU Control and Status Register) llamado ISC2 mostrado en la Figura 2.7.[19] Cuando este bit está en bajo (cero) la interrupción se activa por flanco descendente y cuando está en alto (uno) por flanco ascendente.

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

Figura 2.7. Registro MCUCSR

Todas las Interrupciones Externas se habilitan a través del Registro GICR (General Interrupt Control Register) mostrado en la Figura 2.8.[20] Solo se debe manipular los bits 5, 6 y 7 y dejar los demás en cero. Al poner en alto (uno) cualquiera de los bits del 5 al 7 se habilita la correspondiente interrupción.

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 2.8. Registro GICR

En el Registro GIFR (General Interrupt Flag Register), los bits 5, 6 y 7 llamados INTF2, INTF0 e INTF1 representan las banderas de cada interrupción. Al

[19] La Figura 2.7. fue obtenida del manual del ATmega32, capítulo: External Interrupts
[20] La Figura 2.8. fue obtenida del manual del ATmega32, capítulo: External Interrupts

producirse una interrupción externa se activa la correspondiente bandera, el MCU salta al correspondiente Vector de Interrupción, se ejecuta la rutina, se encera la bandera automáticamente y se regresa al programa principal.

2.8. Timers / Contadores.

El microcontrolador ATmega32 tiene 3 Timers / Contadores, 2 de 8 bits y 1 de 16 bits. El Timer0 es de 8 bits, el Timer1 es de 16 bits y el Timer2 es de 8 bits. Cada Timer tiene varios modos de uso, pero el más simple y más utilizado es el modo Normal que meramente incrementa el contador respectivo hasta que éste se desborda y comienza nuevamente desde cero. Los Timers trabajan con banderas que se activan cuando se produce un desborde o cuando se alcanza un valor establecido de comparación. Cuando se activa una bandera se ejecuta una interrupción que a su vez realiza una rutina determinada.

El Timer0 se utiliza generalmente para operaciones internas de conteo. El Timer1 además se puede utilizar para medir el tiempo entre eventos externos ya que puede capturar datos a través del PD6 llamado ICP (Input Capture Pin). El Timer2 se emplea para operaciones asincrónicas que utilizan un reloj externo.

Todos los Timers pueden trabajar a la frecuencia del microcontrolador o a la frecuencia de una fuente externa, además esta frecuencia puede ser modificada por un *prescaler* que divide la frecuencia de conteo para un factor de 1 / 8 / 64 / 256 / 1024 en el caso del Timer0 y Timer1 y 1 / 8 / 32 / 64 / 128 / 256 / 1024 para el Timer2.

El *prescaler* de cada Timer se configura a través de los Registros TCCR (Timer/Counter Control Register) modificando los 3 últimos bits denominados CS (Clock Select). La cuenta de cada Timer se lleva en los Registro TCNT (Timer Counter).

De esta manera el prescaler del Timer0 y del Timer1 se configuran en los últimos 3 bits (llamados CS02, CS01 y CS00 para el Timer0 y CS12, CS11 y Cs10

para el Timer1) de los Registros TCCR0 y TCCR1LB respectivamente. La configuración se puede hacer como se indica en la Tabla 2.9. a continuación.

CS02	CS01	CS00	Descripción
0	0	0	Temporizador detenido
0	0	1	Clk/1
0	1	0	Clk/8
0	1	1	Clk/64
1	0	0	Clk/256
1	0	1	Clk/1024

Tabla 2.9. Configuración del prescaler del Timer0

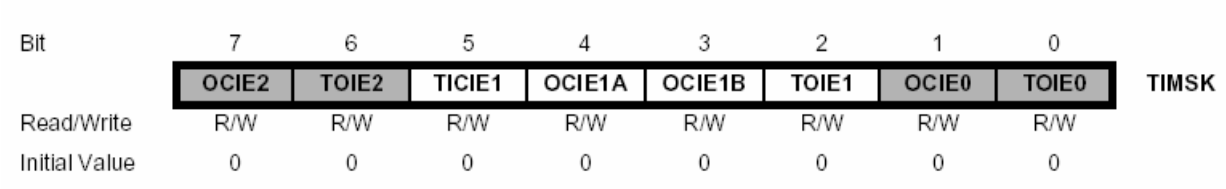
Para el Timer2 la configuración de los bits CS22, CS21 y CS20 del Registro TCCR2 se muestra en la Tabla 2.10.

CS22	CS21	CS20	Descripción
0	0	0	Temporizador detenido
0	0	1	Clk/1
0	1	0	Clk/8
0	1	1	Clk/32
1	0	0	Clk/64
1	0	1	Clk/128
1	1	0	Clk/256
1	1	1	Clk/1024

Tabla 2.10. Configuración del prescaler del Timer2

El tipo de interrupción puede ser por: desborde, comparación y captura; cada tipo de interrupción se habilita a través de los bits: TOIE (Timer Overflow Interruption Enable), OCIE (Ouput Comparison Interruption Enable) y TICIE (Timer Input Capture Interruption Enable) respectivamente, ubicados en el

Registro TIMSK (Timer/Counter Interrupt Mask). Las interrupciones se habilitan poniendo unos en los bits mencionados como se muestra en la Figura 2.9.[21].



Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	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	

Figura 2.9. Registro TIMSK

Así por ejemplo si se tiene un reloj interno de 8 MHz y se quiere configurar el Timer0 a una frecuencia de 1 MHz para que se produzca una interrupción cuando el Registro contador TCNT0 se desborde, se debe configurar los Registros mostrados a continuación:

```
TCCR0 = 0x02;     // Prescaler 8
TCNT0 = 0x00;     // Por seguridad se inicializa el contador en cero
                  // aunque cada vez que se enciende o resetea el ATmega
                  // todos los Registros se enceran
TIMSK=0x01;       // Se habilita la Interrupción por desborde del Timer0
```

Es recomendable escribir el Registro TIMSK al final luego de que todas las configuraciones han sido establecidas

2.9. Captura de datos con el Timer1.

La captura de datos a través del Timer1 se utiliza para poder determinar el lapso de tiempo transcurrido entre dos eventos externos al microcontrolador. La captura del evento se realiza a través un flanco ascendente o descendente detectado en el pin PD6 también conocido como ICP1 (Input Capture Pin).

En el Registro TCCR1B del Timer1 en modo de Captura además de la configuración del prescaler se debe determinar si se desea que el evento externo se capture a través de un flanco ascendente o descendente. Esto se configura a

[21] La Figura 3.8.1 fue obtenida del manual del ATmega32, capítulo: Timers/Counters

través del bit 6 del Registro TCCR1B llamado ICES1 (Input Capture Edge Select), cuando está en cero captura el flanco descendente y cuando está en uno el ascendente.

También es recomendable habilitar un cancelador de ruido a través del bit 7 llamado ICNC1 (Input Capture Noise Canceler) en el registro TCCR1B que obliga al ATmega32 a tomar 4 muestras seguidas de la señal por lo que hay un retardo de 4 ciclos de reloj.

Así por ejemplo si se desea establecer el prescaler en 256, captura de flanco ascendente y el Cancelador de Ruido activado se debería escribir lo siguiente en el Registro TCCR1B:

```
TCCR1B = 0xC4;
```

2.10. Uso de Interrupciones y Timers.

Para activar los Timers / Contadores, así como las Interrupciones Externas se debe habilitar las interrupciones globales en el microcontrolador a través de una instrucción en Assembly, que se debe colocar dentro del código principal del programa luego de inicializar todos los Timers e Interrupciones Externas que se desee utilizar. La instrucción en Assembly es:

```
#asm("sei")
```

Además es necesario tener en cuenta lo que ocurre cuando los Timers y las Interrupciones Externas son activados. Estos interactúan entre si de la siguiente manera:

1. Cuando una Interrupción Externa se activa mientras el programa se encuentra ejecutando una rutina de interrupción de un Timer, la Interrupción Externa espera a que la rutina del temporizador termine para ejecutarse.

2. Cuando un Timer (que cuenta todo el tiempo) se activa mientras el programa se encuentra ejecutando una rutina dentro de una Interrupción Externa, la Interrupción provocada por el Timer espera a que la rutina de la Interrupción Externa termine para ejecutarse.

3. Si se producen varias Interrupciones Externas (por rebote) se ejecutan las rutinas de Interrupción Externa una tras otra lo cual puede producir resultados no deseados.

CAPÍTULO 3

AMBIENTES DE DESARROLLO INTEGRADO (IDE)

3. AMBIENTES DE DESARROLLO INTEGRADO (IDE).

Antes de empezar cualquier Proyecto que utilice microcontroladores, lo primero que se debe hacer es buscar las herramientas necesarias para desarrollar dicho Proyecto. Una de esas herramientas es el Software utilizado para crear el programa de aplicación. Luego probar que el programa de aplicación funcione correctamente y finalmente cargarlo en el microcontrolador. Si se emplea el Software correcto o un conjunto adecuado de ellos, el desarrollo del Proyecto se puede facilitar en gran manera.

El Software usado para el desarrollo de proyectos de aplicación para microcontroladores se denomina *IDE (Integrated Development Environment).* La Atmel proporciona gratuitamente el IDE llamado *AVR Studio* que ofrece un Compilador en Assembly, un Depurador, un Emulador y un Grabador dentro del mismo Software.

Además existen otras compañías que también ponen a disposición del usuario otros IDEs, como por ejemplo el Software conocido como *CodeVisionAVR C Compiler* de características semejantes al AVR Studio, con la diferencia de que su Compilador trabaja en Lenguaje C.

Algunas veces para un mejor resultado es necesario combinar el uso de varios IDEs, lo cual es posible con los IDE: AVR Studio y CodeVisionAVR que son totalmente compatibles entre si. Además al momento de elegir el IDE se debe

tener en cuenta la plataforma de desarrollo y/o grabador que lo complementa, sin embargo ese punto se desarrollará en el capítulo siguiente.

3.1. CodeVisionAVR C Compiler.

CodeVisionAVR C Compiler es un programa que ofrece la oportunidad de programar en Lenguaje C para microcontroladores AVR, lo cual permite generar aplicaciones de una manera más simple que si se trabajase en Lenguaje Assembly.

3.1.1. Descripción.

CodeVisionAVR C Compiler es un programa que fue desarrollado por Pavel Haiduc, HP InfoTech en 1998. CodeVisionAVR es un Ambiente Integrado de Desarrollo (IDE), Generador de Programa Automático y un Programador In-System para la familia de los Microcontroladores AVR de Atmel. El programa está diseñado para trabajar bajo sistemas operativos como Windows 98, Me, NT 4, 2000 y XP. La versión utilizada en este proyecto fue la 1.24.6 Standard[1].

La ventaja de este programa sobre otros existentes es que permite crear programas en Lenguaje C y su compilador implementa casi todos los elementos del Lenguaje ANSI C, según lo permita la arquitectura del AVR. Además permite añadir algunas instrucciones y funciones propias de la Arquitectura AVR. Cuando CodeVisionAVR compila un archivo de programa crea varios archivos entre ellos los más importantes son el archivo .COFF y el archivo .HEX.

Los archivos objeto .COFF pueden ser utilizados para depuración y emulación a través de otros programas como el AVR Studio (CodeVisionAVR no realiza ni emulación ni depuración).

[1] Información obtenida directamente del programa CodeVisionAVR C Compiler

Los archivos .HEX, pueden ser cargados directamente en el microcontrolador AVR a través de un ISP (In-System Programmer) como el STK500/AVRISP/AVRProg de la Atmel o el Kanda Systems STK200 entre otros.

Además de poseer las librerías C estándar, CodeVisionAVR tiene librerías dedicadas entre otras cosas para:

- Módulos alfanuméricos LCD
- El Sensor de Temperatura LM75 de la Nacional Semiconductors
- Manejo de Poder
- Retardos
- Conversión de Código Gray

Además CodeVisionAVR posee el Generador Automático de Programa CodeWizardAVR que permite escribir en pocos minutos el código necesario para implementar muchas funciones básicas.

3.1.2. Ventana Principal.

Inmediatamente al abrir CodeVisionAVR aparece la ventana principal mostrada en la Figura 3.1. que contiene las siguientes partes:

a) **Menús de Comandos.** El Menú de Comandos de CodeVisionAVR es similar al de cualquier otro programa de Windows con las opciones: File, Edit, Project, Tools, Settings, Windows y Help.

b) **Barra de Herramientas.** Todos los comandos del Menú pueden también ser ejecutados de una manera más rápida a través de estos Botones de Comandos.

c) **Ventana del Navegador, Code Templates y Clipboard History.** En la opción *Navigator* se puede manejar los distintos archivos, funciones y variables que

forman parte del Proyecto. En la opción *Code Templates* se puede copiar el formato de las funciones en Lenguaje C más comunes como por ejemplo *if* y *switch*. En la opción *Clipboard History* se mantiene un registro de todos los bloques de texto copiados en el Clipboard para poder ser pegados posteriormente.

d) **Ventana del Editor del código del programa.** En esta ventana se escribe el código del programa de la aplicación en desarrollo en Lenguaje C.

e) **Ventana de Mensajes.** En esta ventana se muestran al usuario los mensajes de compilación y de posibles errores al momento de compilar el código del programa.

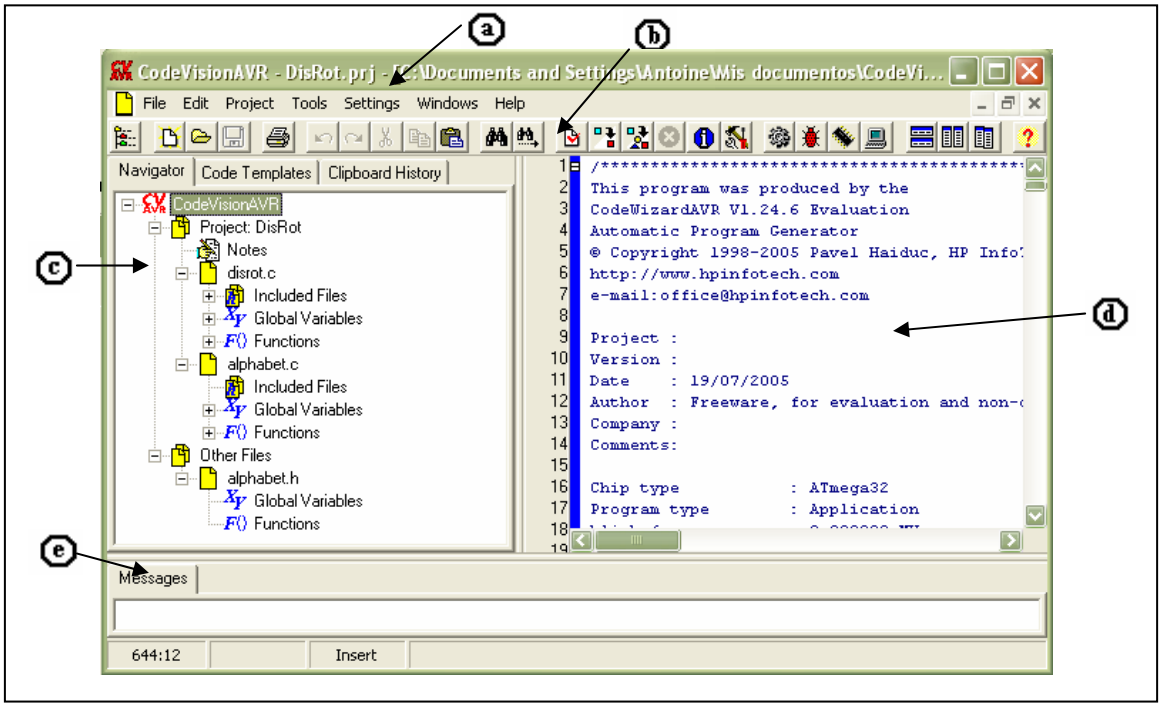


Figura 3.1. Ventana principal de CodeVisionAVR

3.1.3. Creando un Proyecto.

Para crear un nuevo proyecto se debe seleccionar el comando *File/New* o simplemente presionar el botón *Create New File* de la barra de herramientas. A

continuación aparece una ventana, como la mostrada en la Figura 3.2., donde se debe escoger la opción *File Type/Project* y luego presionar OK.

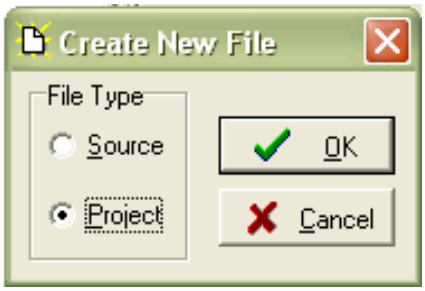


Figura 3.2. Ventana Create New File

Luego aparece una ventana de diálogo, como la mostrada en la Figura 3.3., para preguntar si se desea utilizar CodeWizardAVR para crear un nuevo Proyecto. Si se presiona *Yes* se ejecuta CodeWizardAVR. Si se presiona *No,* CodeWizardAVR no se ejecuta.

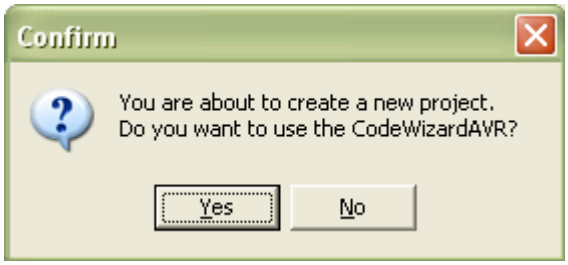


Figura 3.3. Ventana Confirm (CodeWizardAVR)

Después de que se haya ejecutado o no CodeWizardAVR aparece la ventana *Create New Project*, como se muestra en la Figura 3.4., donde se debe ingresar el nombre del Proyecto. Es recomendable que se cree una carpeta para guardar allí únicamente el Proyecto y sus archivos generados posteriormente para evitar que se mezclen con los de otros Proyectos.

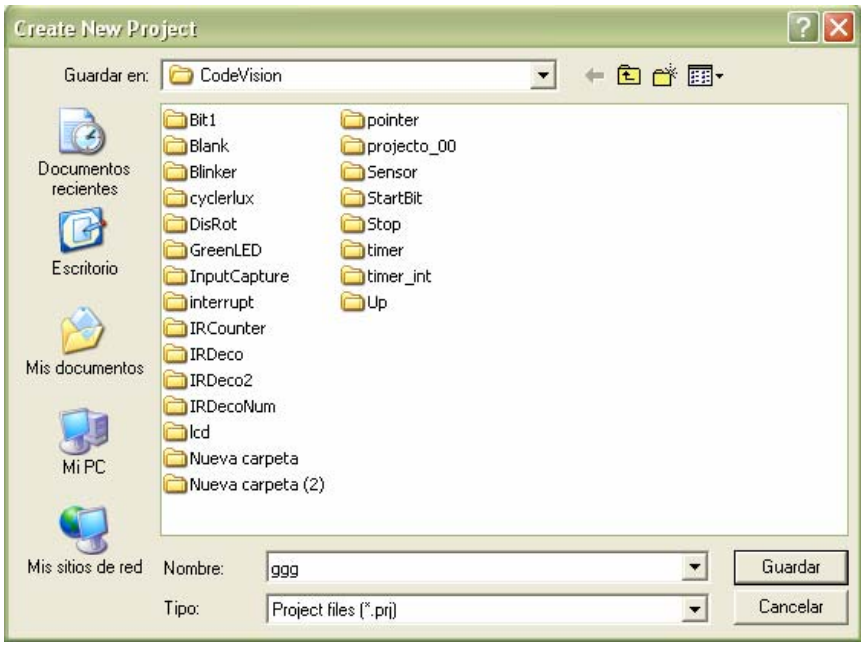


Figura 3.4. Ventana Create New Project

El Proyecto se guardará con la extensión .prj. Si se desea hacer cambios en el Proyecto se puede usar el comando *Project/Configure*.

3.1.4. CodeWizardAVR. Código inicial de un programa.

Si se desea ingresar al Generador Automático de Programa conocido como CodeWizardAVR dentro de un Proyecto que ya haya sido creado sin tener que crear un nuevo Proyecto, se lo puede hacer a través del comando *Tools/CodeWizardAVR* en el Menú o presionando el botón *CodeWizardAVR*. Esta función especial de CodeVisionAVR permite escribir automáticamente el código de inicialización de varias operaciones y funciones básicas como:

- Acceso a memoria externa
- Identificación de la fuente de reset del Chip
- Inicialización de Puertos I/O
- Inicialización de Interrupciones Externas
- Inicialización de Timers
- Inicialización del WatchDog Timer
- Inicialización de la comunicación serial UART (USART)

- Inicialización del Comparador Análogo
- Inicialización del ADC
- Inicialización de la Interfase SPI
- Inicialización de la Interfase Two Wire
- Inicialización de la Interfase CAN
- Inicialización de Sensores de Temperatura
- Inicialización de módulo LCD

Para generar el código inicial de un programa se debe ingresar los parámetros deseados en las sub-ventanas correspondientes dentro de la ventana de CodeWizardAVR.

Entre las distintas opciones de inicialización brindadas por CodeWizardAVR, aquellas utilizadas en este proyecto fueron las siguientes:

- Chip
- Ports
- External IRQ
- Timers
- LCD

Chip

En esta ventana mostrada en la Figura 3.5. se puede especificar el tipo de microcontrolador utilizado y la frecuencia del reloj. La frecuencia escogida es solo para la descripción del programa ya que la frecuencia del microcontrolador realmente se configura mediante los Bits de Configuración.

Además para los microcontroladores AVR que permiten la identificación de las fuentes de reset, hay un check box suplementario. Si está activado entonces el CodeWizardAVR generará un código que permita identificar las condiciones que causaron el reset. Para los microcontroladores AVR que permiten la auto

programación, se puede escoger el tipo de programa entre Aplicación y Boot Loader.

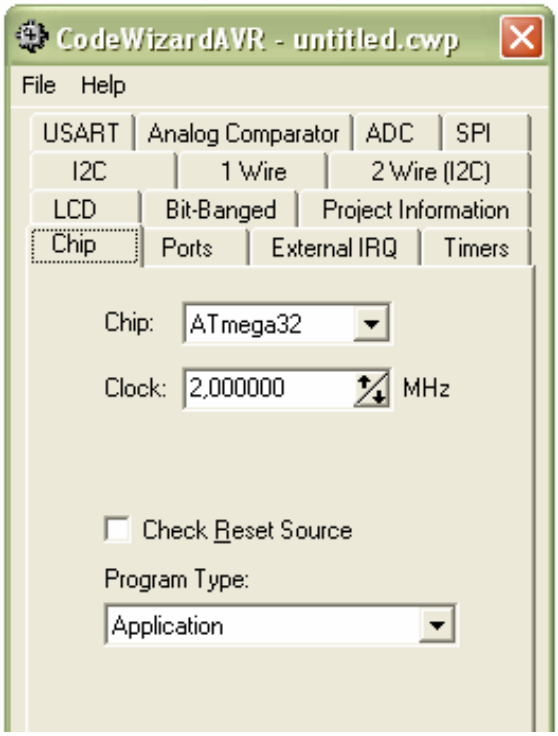


Figura 3.5. Ventana CodeWizardAVR/Chip

Ports

En esta ventana mostrada en la Figura 3.6. se puede escoger que puerto PORT x se quiere configurar, el número de Puertos disponibles dependerá del tipo de AVR que se esté utilizando. Se puede determinar si cada Bit del Puerto escogido va a ser de entrada o de salida individualmente.

Si se configura un Puerto de entrada además se debe determinar si tendrá una resistencia de Pull-Up (P) o si será Tres-estados (T). Si se configura un Puerto como de salida además se debe determinar su estado inicial que puede ser 1 ó 0.

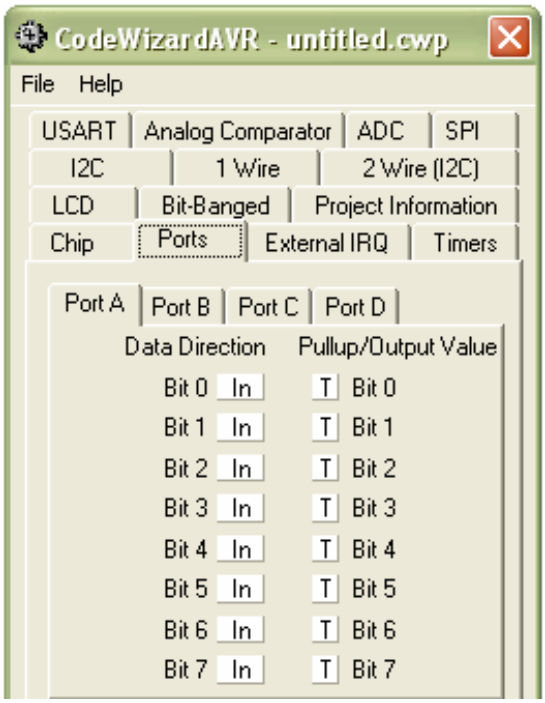


Figura 3.6. Ventana CodeWizardAVR/Ports

External IRQ

En esta ventana mostrada en la Figura 3.7. se escoge las Interrupciones Externas que se quieran habilitar. El número de Interrupciones Externas disponibles dependerá del tipo de AVR que se esté utilizando. También se puede escoger el modo en que será activada la Interrupción: por flancos, por niveles bajos o por cualquier cambio dependiendo de la Interrupción escogida.

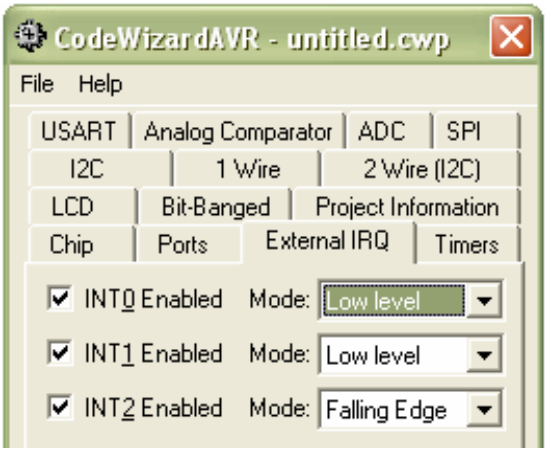


Figura 3.7. Ventana CodeWizardAVR/External IRQ

Timers

En esta ventana mostrada en la Figura 3.8. se configuran los Timers y el Watchdog. Se escoge la fuente de reloj que puede ser el Reloj del Sistema o una fuente externa, el prescaler, el modo de trabajo, los valores de comparación y el valor inicial del contador.

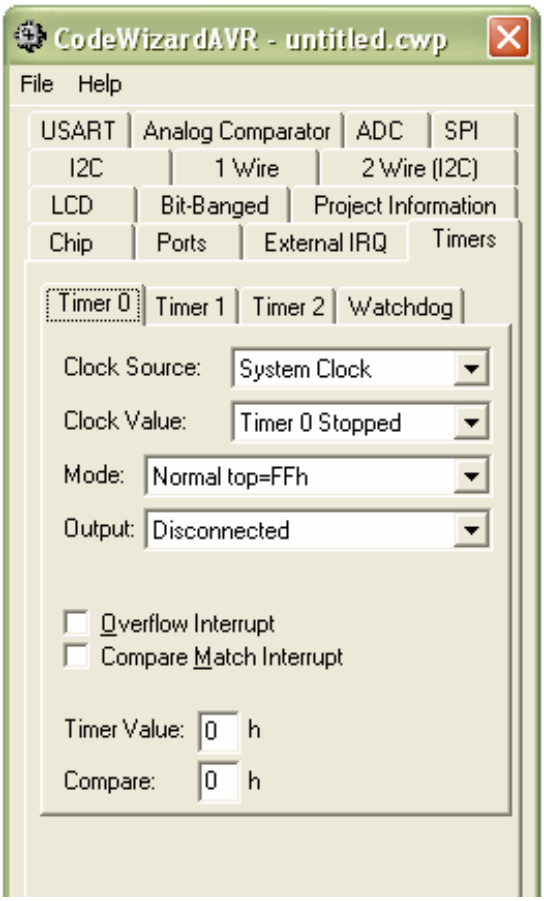


Figura 3.8. Ventana CodeWizardAVR/Timers

LCD

En esta ventana mostrada en la Figura 3.9. se puede inicializar la interfase para un LCD conectado al microcontrolador AVR. Primero se debe escoger el Puerto que controlará el LCD y luego el número de caracteres por línea.

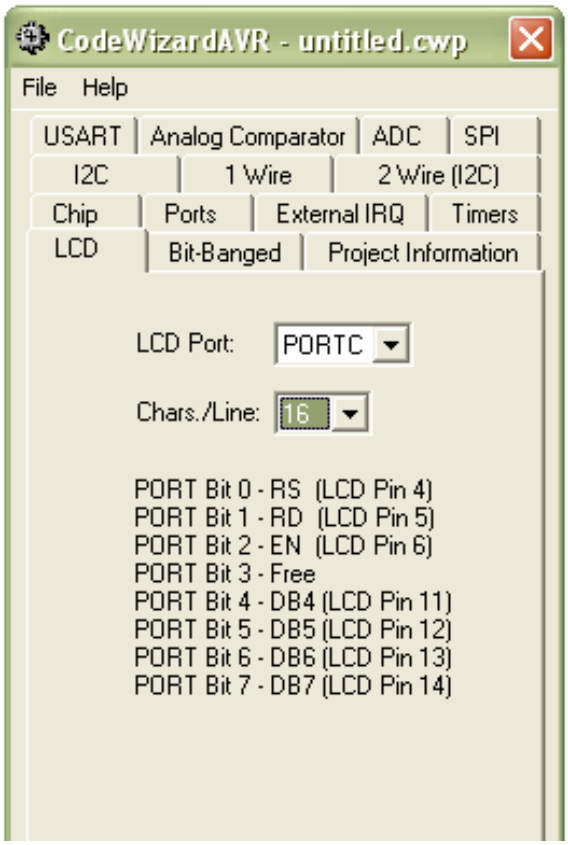


Figura 3.9. Ventana CodeWizardAVR/LCD

Generando los archivos.

Luego de establecer los parámetros de inicialización deseados se debe escoger el comando *File/Generate, Save and Exit* para generar tres archivos con las extensiones .c, .prj y .cwp que deben estar dentro de la misma carpeta. El archivo .c contiene todas las configuraciones establecidas a través de los Registros respectivos. CodeWizardAVR evita tener que memorizar el nombre de cada Registro que se debe alterar para habilitar e inicializar una determinada función del microcontrolador AVR, sin embargo es recomendable saber como funciona cada registro para poder hacer cambios sin tener que acudir al CodeWizardAVR a cada momento.

3.1.5. Relación entre Variables y Registros.

Otra ventaja de CodeVisionAVR es que muestra en la Ventana del Navegador la relación entre las variables creadas en el programa y los 32 Registros utilizados en el chip AVR conocidos como R1, R2, R3,… y R31.

Las Variables Globales tipo char, int y pointer y las Variables Locales tipo bit se guardan desde el Registro R2 hasta el R15. Las Variables Locales tipo char, int y pointer se guardan desde el Registro R16 hasta el R31. El orden en que se guardan las variables en los registros no es necesariamente secuencial sino que responde a configuraciones avanzadas de CodeVisionAVR que pueden ser establecidas por el usuario.

Si se crean más variables que las que se pueden guardar en los Registros, las variables recientemente creadas se guardan en la Memoria SRAM y simplemente se muestra su dirección.

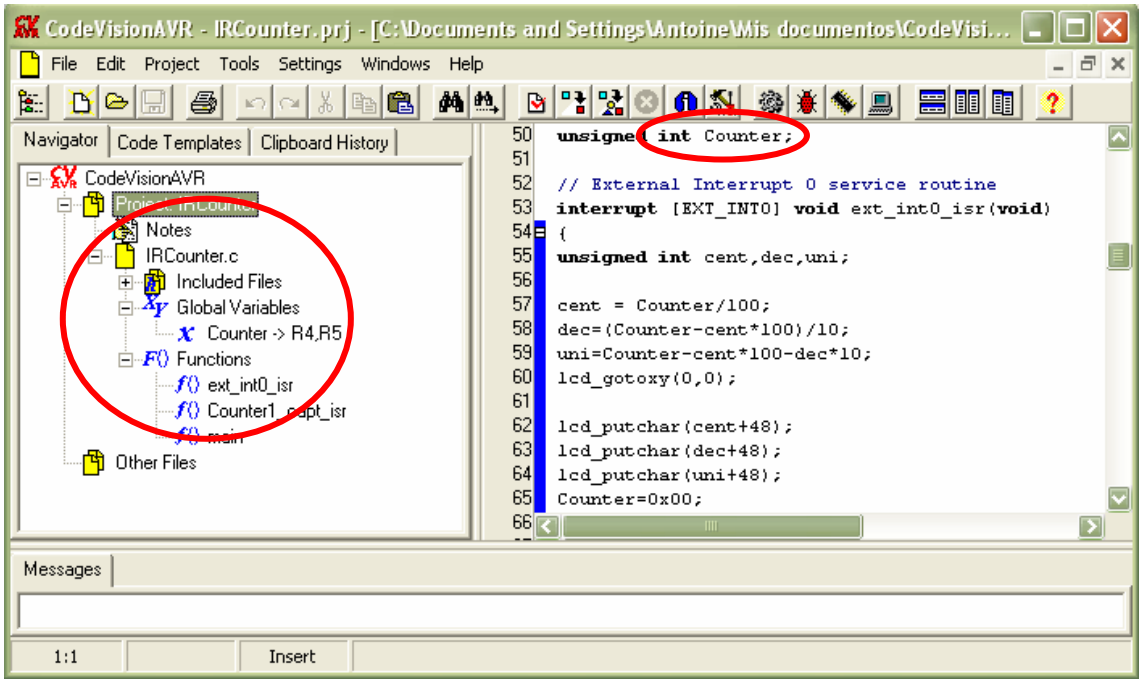


Figura 3.10. Ventana principal de CodeVisionAVR. Registros

En la Figura 3.10. se muestra por ejemplo que la Variable Global Counter está guardada en los Registros R4 y R5 debido a que es tipo int y ocupa 16 bits (cada Registro solo tiene 8 bits).

3.1.6. Ingresando a la EEPROM.

Cuando se necesita que las variables alteradas durante la ejecución del programa grabado en el chip AVR no pierdan su información al apagarlo, se debe utilizar variables localizadas en la EEPROM.

El uso de la EEPROM se realiza utilizando la palabra clave eeprom al declarar la variable que se quiere guardar en dicha memoria, como se muestra a continuación:

```
eeprom unsigned char alpha;
```

3.1.7. Compilación.

Al compilar el archivo en C se generan varios archivos entre ellos el .COFF y el .HEX. El archivo .COFF (Common Object File Format) puede ser utilizado para depuración y el .HEX se graba en la Memoria Flash del microcontrolador AVR.

Luego de compilar el archivo editado a través del comando *Project/Make* o a través del botón *Make*, CodeVisionAVR presenta una ventana de resultados, como la mostrada en la Figura 3.11. Esta ventana muestra las características del archivo compilado como por ejemplo el número de líneas del programa, el número de palabras, el porcentaje de la EEPROM y de la Memoria Flash utilizados.

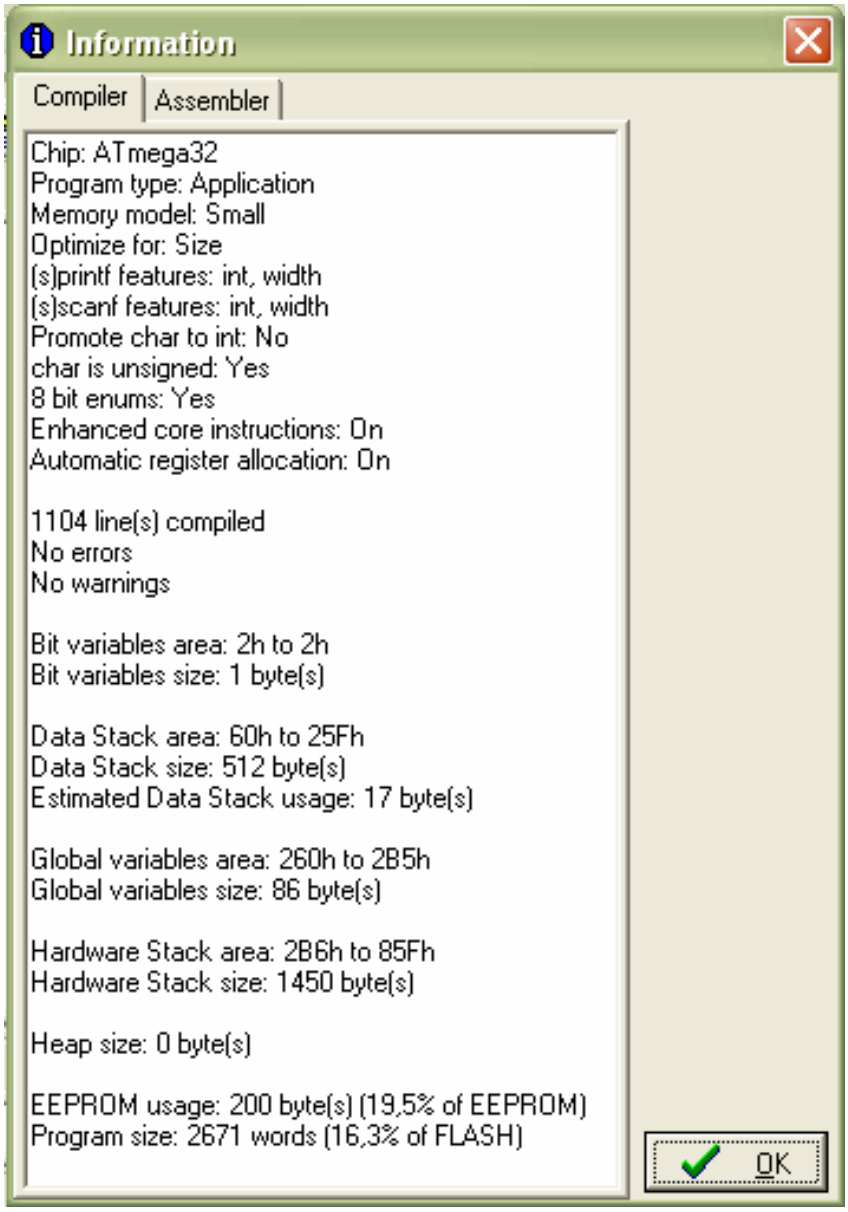


Figura 3.11. Ventana de Información luego de la compilación

En el caso de producirse algún error se puede ver en la ventana del Navegador la línea del programa en que ocurrió el problema. De esta manera se pueden hacer las correcciones necesarias como se muestra en la Figura 3.12.

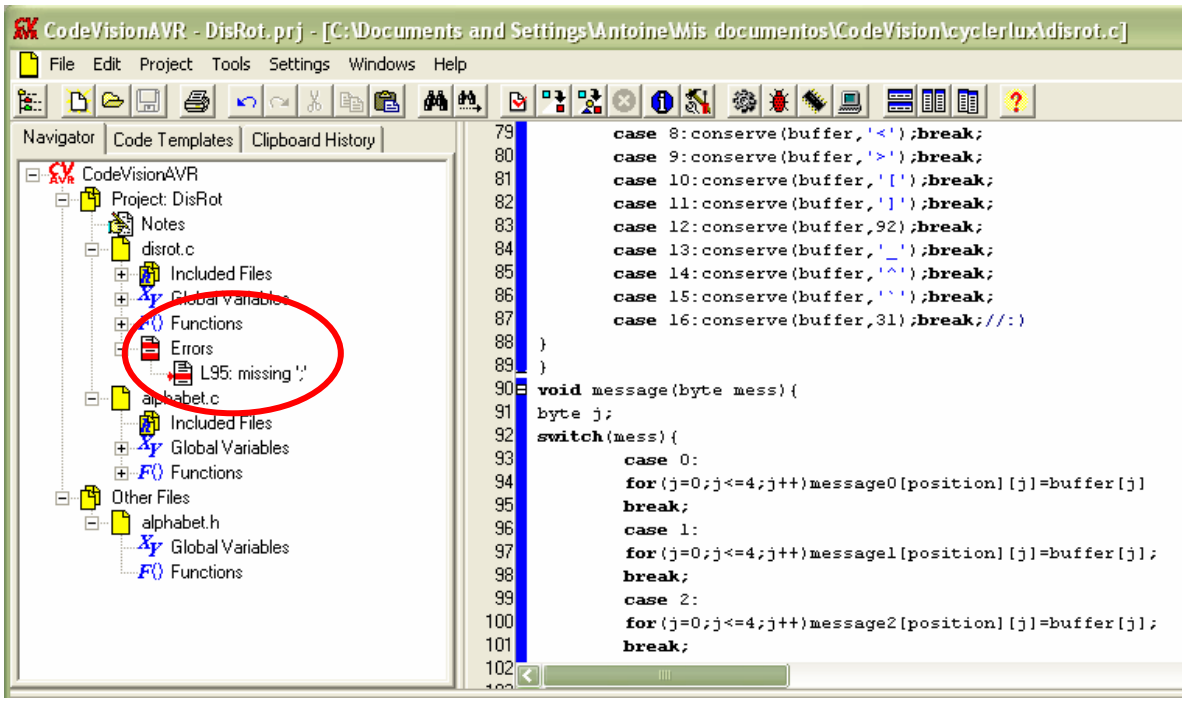


Figura 3.12. Ventana principal de CodeWizardAVR. Errores

3.1.8. Depuración.

CodeVisionAVR no realiza la depuración del programa, pero puede llamar a otros programas para realizarlo, lo más común es utilizar AVR Studio.

AVR Studio es el IDE creado por la misma Atmel para los microcontroladores AVR, su inconveniente es que únicamente compila programas escritos en Lenguaje Assembly, por ese motivo no fue escogido para la realización de este proyecto.

Sin embargo AVR Studio es capaz de realizar la depuración de programas escritos en Lenguaje C siempre y cuando se genere el archivo .COFF al compilar. La depuración a través de AVR Studio se tratará más adelante.

3.1.9. Programación en Lenguaje C.

La programación en Lenguaje C en CodeVisionAVR utiliza las mismas instrucciones y operadores estándar de C, acepta estructuras de control tan

simples como if, else y switch, e incluso funciones matemáticas si se incluye la librería adecuada. Se pueden escribir funciones propias, incluir a otros archivos C, hacer estructuras, matrices de hasta dos dimensiones, utilizar punteros, etc.

Las Interrupciones Externas, Timers, LCDs, y demás elementos relativos al chip AVR se manejan a través de funciones especiales que trabajan como cualquier otra función escrita en C, es decir, pueden recibir parámetros, retornar valores, ser tipo void, etc.

Además con el uso de CodeWizardAVR, no es necesario escribir las funciones ya que estás se generan automáticamente al igual que su parámetros de inicialización, lo único que se debe hacer es ingresar el código que realice la operación que se desee grabar en el microcontrolador AVR.

Creación de un Código de Programa. Ejemplo práctico.

Si se desea escribir un programa que utilice un ATmega32 con un reloj de 4 MHz (aunque la frecuencia del reloj no se configura en el programa sino a través de los Bits de Configuración) se puede ingresar estos datos en la ventana de CodeWizardAVR, en la opción *Chip.* Esto generaría el siguiente código que podría servir como una plantilla:

```
#include <mega32.h>

// Declare your global variables here

void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
```

```
DDRB=0x00;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
```

```
MCUCR=0x00;
MCUCSR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

while (1)
    {
    // Place your code here

    };
}
```

Basta con analizar con cuidado el código generado gracias a los comentarios también generados automáticamente para entender los parámetros con los cuales se ha generado el programa. Se puede apreciar los valores dados a los Registros de los Timers, Interrupciones Externas, Comparadores Análogos, Puertos. Además ya quedan establecidos los puntos donde se debe ingresar el código que definirá el funcionamiento del chip AVR.

Modificando el Código del Programa.

Si se quisiera además habilitar, por ejemplo el Timer0, se puede volver a presionar el botón de CodeWizardAVR y aparecerá una ventana como la de la Figura 3.13. donde se puede habilitar el Timer0 para que utilice el Reloj del Sistema como fuente, que trabaje en modo normal a la misma frecuencia que el Reloj interno del microcontrolador AVR y que la interrupción se produzca por desborde.

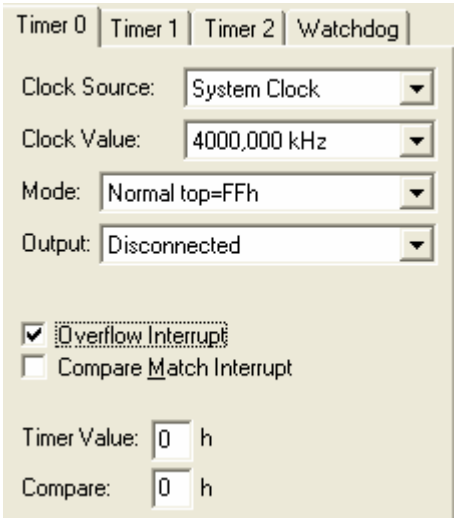


Figura 3.13. Ventana de CodeWizardAVR/Timers

En vez de volver a generar otro código, se puede realizar una vista previa del código generado a través del comando *File/Program Preview* y copiarlo en el programa original, claro que para hacer esto es necesario conocer como funcionan los Registros del microcontrolador AVR por lo que es recomendable siempre tener a mano el manual del microcontrolador que se esté utilizando.

Al ver la vista previa del programa, lo único que se añade es la función de Interrupción por desborde del Timer0:

```
include <mega32.h>
// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Place your code here
}
// Declare your global variables here
……
```

y se varían los Registros correspondientes, en este caso TCCR0 y TIMSK

```
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 4000,000 kHz
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x01;
TCNT0=0x00;
OCR0=0x00;
……
```

```
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x01;
```

De esta manera, CodeWizardAVR y el manual del microcontrolador AVR utilizados en conjunto son herramientas muy útiles para realizar cambios en el programa, sin tener que memorizarse los Registros ni las funciones, ni tener que generar constantemente nuevos archivos gracias a la vista previa.

Uso de Instrucciones en Assembly.

Algunas operaciones requieren que se trabaje al nivel más bajo de programación, como por ejemplo la habilitación de las Interrupciones Globales. Para esto se tienen dos opciones dependiendo de cuantas líneas de código se desean ingresar.

Si solo se desea ingresar una línea de código o instrucción se puede utilizar #asm(“nop”), por ejemplo:

```
// Global enable interrupts
#asm("sei")
```

Sei es una instrucción en Assembly que habilita las Interrupciones Globales. Si se desea ingresar varias instrucciones se puede utilizar #asm(“nop/nop/nop”) o se lo puede hacer de la siguiente manera:

```
#asm
        nop
        nop
        nop
#endasm
```

3.1.10. Inclusión de Archivos de Cabecera .H.

Muchas veces es necesario dividir un programa en varios archivos para hacerlo más estructurado y entendible. Todo programa siempre tiene un archivo principal

con extensión .c que incluye a los demás archivos denominados *de cabecera* con extensión .h.

Para dividir un programa en varios archivos lo primero que se debe hacer es crear los archivos .c que contengan sus propias variables y funciones. La diferencia entre un archivo principal y uno de cabecera es que el archivo principal tiene el mismo nombre que el proyecto, contiene las definiciones requeridas para compilar el programa a través de la instrucción *#include* y además la función principal *main()*.

Por ejemplo, dentro de la carpeta *EXAMPLES* de *CodeVisionAVR* se puede encontrar el Proyecto *MULTFILE* que contiene varios archivos .c, el principal es llamado *Mainfile.c* y los auxiliares *file1.c, file2.c* y *file3.c*. Además existen los archivos de cabecera *file1.h, file2.h* y *file3.h.*

El código de *Mainfile.c* es el siguiente:

```
/* Example of a simple multifile project

   CodeVisionAVR C Compiler
   (C) 2000-2002 HP InfoTech S.R.L.
   www.hpinfotech.ro
*/

// this is the main file of the project

// include the header files with function declarations
#include "file1.h"
#include "file2.h"
#include "file3.h"

main() {
// declare a local variable
int i;
// call a function with the prototype declared in "file1.h"
// the function itself is located in "file1.c"
i=func1(10);
// call a function with the prototype declared in "file2.h"
// the function itself is located in "file2.c"
i=func2(i);
// call a function with the prototype declared in "file3.h"
// the function itself is located in "file3.c"
i=func3(i);
// stop here
while (1);
}
```

El código de *file1.c* es el siguiente:

```
int func1(int a) {
return a+a;
}
```

El código de *file1.h* contiene:

```
int func1(int a);
```

El código de *file2.c* es el siguiente:

```
int func2(int a) {
return a*a;
}
```

El código de *file2.h* contiene:

```
int func2(int a);
```

El código de *file3.c* es el siguiente:

```
int func3(int a) {
return a*3;
}
```

El código de *file3.h* contiene:

```
int func3(int a);
```

Como se puede apreciar en el archivo *Mainfile.c*, la función *func1(int a)* que pertenece al archivo *file1.c* es llamada dentro de la función *main()* mediante. *i = func1(10).* Para que esto funcione se debe incluir al principio del archivo *Mainfile.c* el archivo *file1.h* a través de la instrucción *#include*.

Para hacer que los archivos *file1.c* y *file1.h* formen parte del proyecto *MAINFILE* deben estar dentro de la misma carpeta del proyecto *MAINFILE.prj*. Luego se debe abrir en *CodeVisionAVR* la ventana *Configure Project* a través del comando *Project/Configure* y añadir el archivo *file1.c* al archivo *mainfile.c* a través del botón *Add* como se muestra en la Figura 3.14.

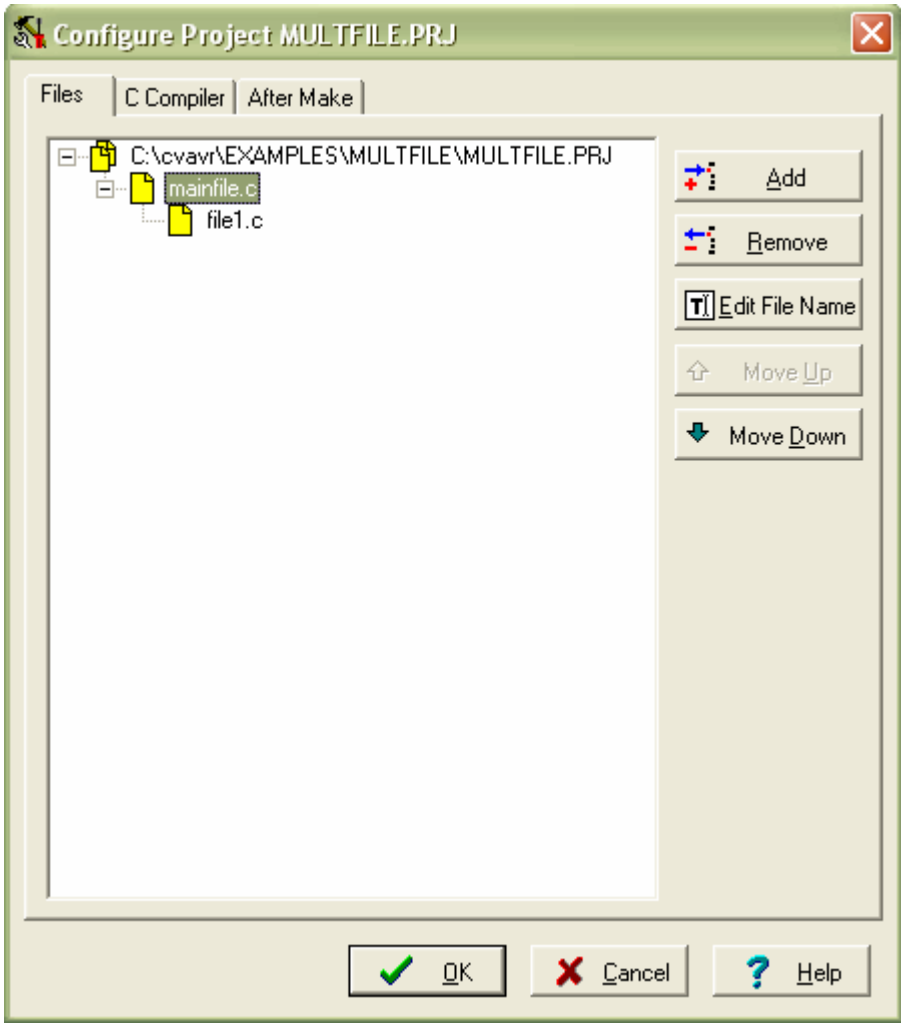


Figura 3.14. Ventana de Configure Project (MULTFILE.PRJ)

El mismo funcionamiento se aplica para los archivos *file2.c, file2.h, file3.c* y *file3.h*

3.2. AVR Studio.

AVR Studio es un programa distribuido gratuitamente por Atmel que aunque trabaja con proyectos escritos en Assembly tiene la facultad de simular aplicaciones escritas en Lenguaje C para microcontroladores AVR.

3.2.1. Descripción.

AVR Studio es un Ambiente de Desarrollo Integrado (IDE) para escribir y depurar aplicaciones AVR en Windows 9x/Me/NT/2000/XP. AVR Studio ofrece: herramientas para la administración de proyectos, un editor de archivos fuente, un simulador y emulador *In-circuit* para los microcontroladores AVR de 8 bits[2].

AVR Studio trabaja perfectamente con el STK500[3] que permite la programación de los chips AVR y además es compatible con interfases JTAG[4]. La versión utilizada en este proyecto es el AVR Studio 4.11.

3.2.2. Simulación y Depuración.

Debido a que el programa del proyecto fue escrito en Lenguaje C y no en Assembly, no se puede utilizar el editor ni el administrador de proyectos de AVR Studio. Sin embargo si se puede utilizar el simulador para hacer una depuración del programa.

Para realizar la depuración estando en CodeVisionAVR, se presiona el botón de *Debugger* o se selecciona el comando *Tools/Debugger* en el Menú. Si se lo hace por primera vez, se debe ingresar la ruta que se debe seguir para ejecutar el AVR Studio.

---

[2] Información obtenida directamente del programa AVR Studio.
[3] El STK500 es la plataforma de desarrollo oficial de Atmel.
[4] JTAG significa Joint Test Action Group. Esta interfase permite realizar pruebas de diagnóstico y continuidad en microcontroladores y otros circuitos impresos.

Al ingresar a AVR Studio, lo primero que se debe hacer es escoger el archivo .COFF del programa que se desee simular y el tipo de chip AVR que se quiere utilizar.

Para cambiar el reloj del sistema del microcontrolador AVR que generalmente está en 8 MHz se debe ingresar al Menú *Debug/AVR Simulator Options* y cambiar la frecuencia del reloj.

La ventana principal de AVR Studio se muestra en la Figura 3.15. y contiene las siguientes partes:

a) **Menús de Comandos.** El Menú de Comandos de CodeVisionAVR es similar al de cualquier otro programa de Windows con las opciones: File, Project, Edit, View , Tools, Debug, Windows y Help.

b) **Barra de Herramientas.** Todos los comandos del Menú pueden también ser ejecutados de una manera más rápida a través de estos Botones de Comandos.

c) **Ventana Workspace.** En esta ventana se puede trabajar con los Registros de propósito múltiple, el Procesador, el Stack Monitor y los Puertos I/O del microcontrolador que se desee simular.

d) **Ventana del Editor del código del programa.** En esta ventana se puede ver el código del programa de la aplicación simulada.

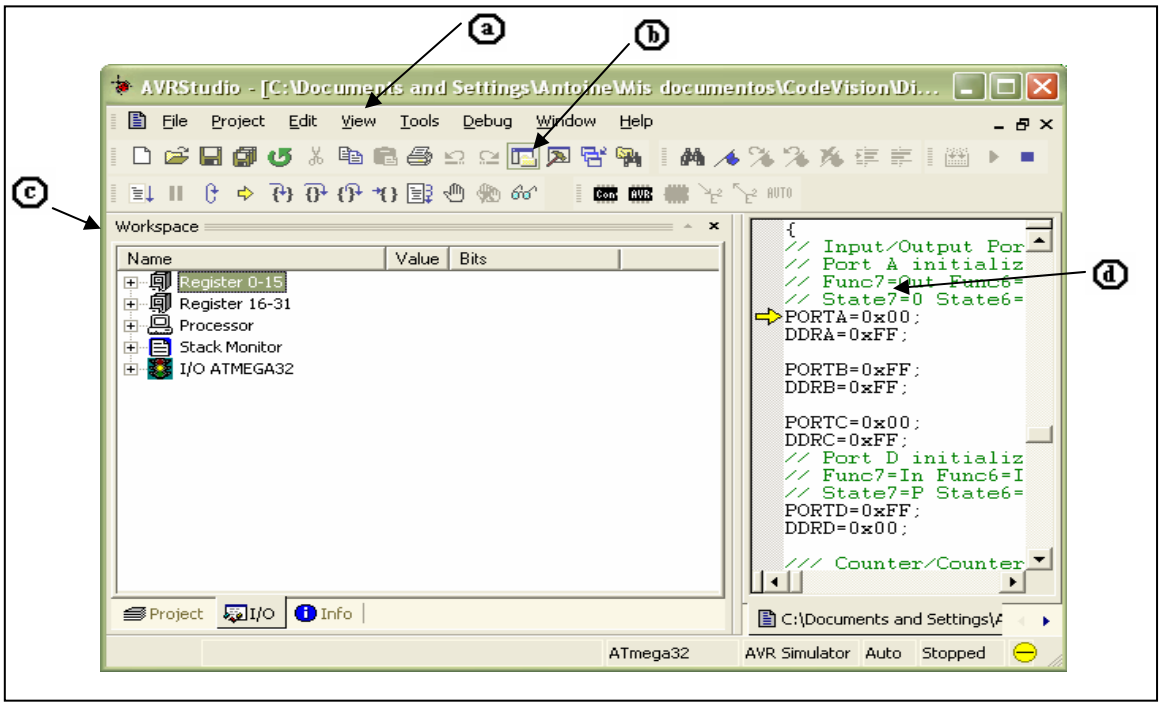


Figura 3.15. Ventana principal de AVR Studio

3.2.3. Manejo de Registros.

En la ventana *Workspace,* al hacer clic en cualquiera de los dos *Registers,* como se muestra en la Figura 3.16. se puede apreciar los valores guardados en los Registros del 0 al 15 o en los Registros del 16 al 31.

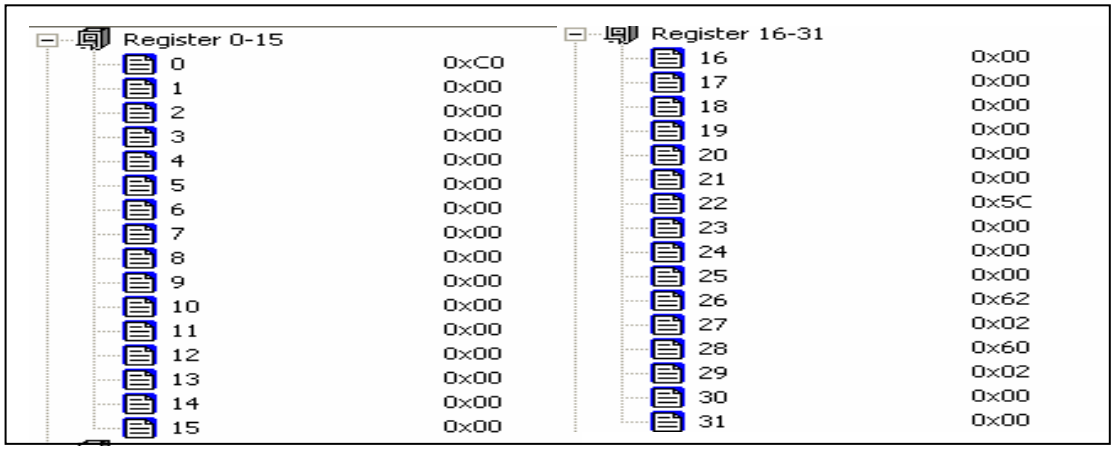


Figura 3.16. Ventana Workspace/Registers

3.2.4. Manejo del Procesador.

En la ventana *Workspace* al hacer clic en *Processor,* como se muestra en la Figura 3.17. se puede apreciar datos básicos sobre el Microprocesador mientras se esté ejecutando la simulación como por ejemplo el Program Counter, la frecuencia del reloj y un cronómetro para medir el tiempo transcurrido al ejecutarse cada línea del programa.

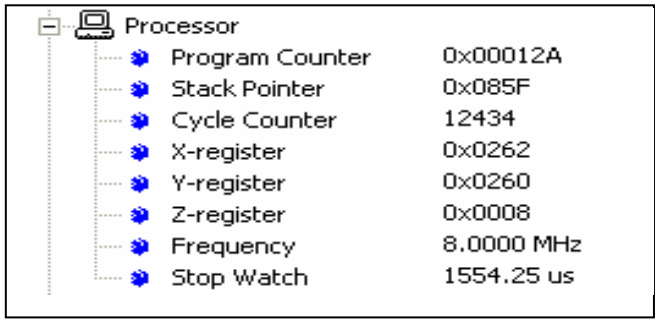


Figura 3.17. Ventana Workspace/Processor

3.2.5. Manejo de puertos.

En la ventana *Workspace* al hacer clic en *I/O ATMEGA32,* como se muestra en la Figura 3.18. se puede ver el estado de los Registros de los Puertos, los cuales se pueden manejar de manera gráfica. Simplemente se debe cambiar el estado de cada bloque con el ratón, cada bloque representa un bit de un determinado Puerto.

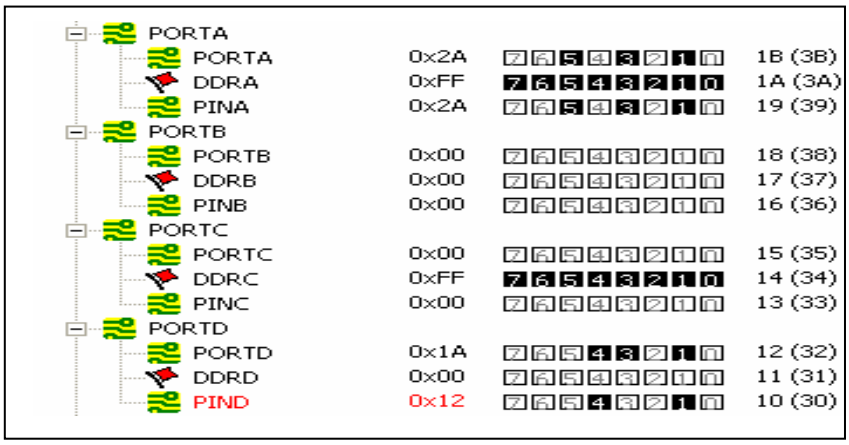


Figura 3.18. Ventana Workspace/I/O ATMEGA32

Como ya se mencionó anteriormente (*2.5. Configuración y Manejo de Puertos I/O*), cada puerto tiene tres registros relacionados: el Registro de Datos PORTx, el Registro de Dirección de Datos (Data Direction Register) DDRx y el Registro del Pin de Puerto de Entrada (Port Input Pins) PINx.

El Registro PINx es un registro de solo lectura mientras que los otros dos Registros, PORTx y DDRx, son de lectura y escritura.

Si el Puerto está configurado como entrada se puede simular un cambio de estado cambiando el color de cada bloque que representa uno de los 8 bits del Registro PINx, siendo blanco 0 y negro 1. En el siguiente ciclo de reloj este cambio se hará presente automáticamente en los bloques que representan los 8 bits de los Registros PORTx y DDRx.

Si el Puerto está configurado como salida, su estado cambiará dependiendo del programa y su valor se podrá apreciar en los bloques que representan los 8 bits de los Registros PORTx y DDRx.

3.2.6. Propiedades.

La ventana de trabajo del AVR Studio puede ser configurada para que se adapte mejor a nuestros requerimientos. Al hacer clic derecho con el ratón aparece la ventana de opciones mostrada en la Figura 3.19.

En esta ventana se puede activar la presentación de datos de manera decimal o hexadecimal. Entre otras características también permite contraer los elementos no utilizados dentro de la ventana del *Workspace* para tener a la vista solo los elementos que el usuario está manejando en ese momento.

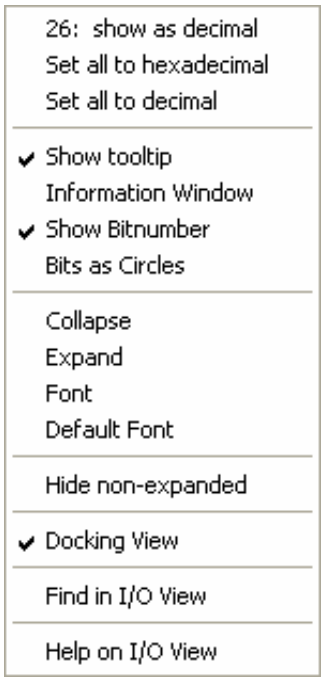


Figura 3.19. Ventana de Propiedades de AVR Studio

3.2.7. Simulación de un Programa.

Para realizar la simulación de un programa en CodeVisionAVR, lo primero que se debe hacer, como ya se mencionó anteriormente, es ejecutar el comando *Tools/Debugger* en el Menú o presionar el botón de Debugger en la Barra de Herramientas.

Si es la primera vez que se hace una simulación, se debe ingresar la ruta que se debe seguir para ejecutar el AVR Studio. Luego se ingresa el archivo .COFF del programa que se desee simular y el tipo de microcontrolador AVR que se quiere utilizar.

Se puede controlar la ejecución de la simulación a través de los botones de la Barra de Herramientas de AVR Studio mostrados en la Figura 3.20.

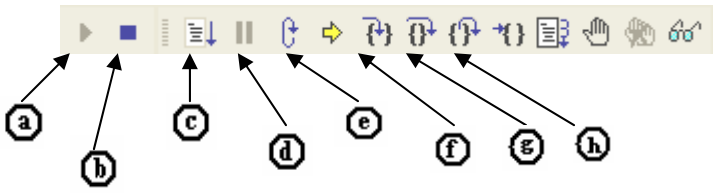


Figura 3.20. Botones del Simulador en la Barra de Herramientas de AVR Studio

Dichos botones permiten comenzar y detener la depuración, resetearlo, ejecutarlo línea por línea o automáticamente, entre otras opciones. Las funciones más importantes son:

a) Start Debugging. Carga el archivo .COFF y comienza la simulación

b) Stop. Detiene la simulación completamente. La única forma de comenzar la simulación nuevamente es presionando *Start Debugging*.

c) Run. Ejecuta cada línea de código del programa una tras otra de manera automática. El programa se ejecutará hasta que se presione *Stop*.

d) Break. Cuando se está ejecutando *Run*, detiene la simulación en la línea donde se haya quedado.

e) Reset. Resetea la simulación encerando todos los Registros y regresando a la primera línea del código del programa.

f) Step Into. Ejecuta una línea de comando cada vez que es presionado. Si encuentra una subrutina, ingresa a la misma.

g) Step Over. Ejecuta una línea de comando cada vez que es presionado. Si encuentra una subrutina la ejecuta como si fuese una línea de comando y continúa.

h) Step Out. Cuando está dentro de una subrutina ejecuta cada línea de comando hasta salir de la misma y luego se detiene. Si está en el nivel

superior de un programa ejecuta cada línea de comando hasta que se presione *Stop*.

Además si se cambia el archivo original en CodeVisionAVR y se lo vuelve a compilar, en AVR Studio sale una ventana como la mostrada en la Figura 3.21. que pregunta si se desea actualizar el archivo de simulación y luego lo hace automáticamente.

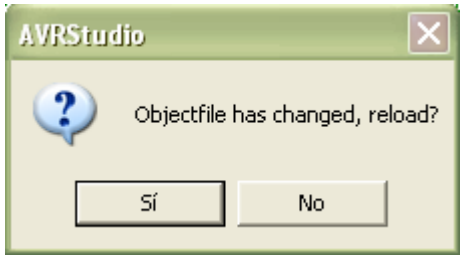


Figura 3.21. Ventana de pregunta para actualizar el archivo de simulación

CAPÍTULO 4

GRABADORES DE MICROCONTROLADORES AVR. PONYPROG2000

4. GRABADORES DE MICROCONTROLADORES AVR.

Una vez que se tiene el IDE adecuado para desarrollar el proyecto con microcontroladores AVR, el siguiente paso es conseguir el Grabador que mejor se adapte a las necesidades y a las posibilidades económicas del Proyecto. La selección del IDE y el Grabador no necesariamente debe ir en este orden, sino más bien en forma paralela. Aunque en este Proyecto se presenta el análisis del IDE elegido antes que la elección del Grabador, la selección de ambos se hizo siempre tomando en cuenta su compatibilidad, de nada serviría un IDE que nos genere un archivo .HEX para grabarlo en el microcontrolador si no se dispone del medio físico para hacerlo.

El Hardware en si consiste del circuito o los circuitos necesarios para grabar el código de programa de la aplicación, generado a través de un IDE, en el microcontrolador. En el mercado existen muchos tipos de grabadores, entre ellos el más completo es el STK500 distribuido por la propia Atmel. El STK500 no solo es un grabador sino además es un kit de pruebas para las aplicaciones creadas por el usuario con varios módulos e interfases que facilita bastante la creación de nuevos Proyectos.

Como es de esperarse el precio del STK500 es bastante elevado debido a todas las funciones que realiza con el microcontrolador. Esta situación obliga a buscar circuitos más simples y por lo tanto más económicos que simplemente

cumplan con la función de grabar el código de programa del archivo .HEX creado en el IDE.

4.1. Grabadores en paralelo y en serie.

Básicamente los grabadores de microcontroladores AVR se pueden dividir en dos grupos: en paralelo y en serie. Los grabadores de Kanda Systems son generalmente en paralelo y los grabadores de Atmel, como el STK500, son seriales. Los grabadores en paralelo suelen ser más simples de construir, pero tienen el inconveniente de que la mayoría de computadoras solo tienen un puerto paralelo para la impresora. Los grabadores seriales son más complejos, pero permiten utilizar el puerto serial de la computadora que generalmente siempre está disponible.

El grabador PonyProg2000 utiliza una interfase serial no muy compleja que puede ser construida utilizando pocos elementos electrónicos y que funciona perfectamente con los microcontroladores AVR, inclusive con la sub-familia ATmega.

4.2. Elección del grabador utilizado.

Existen muchos tipos de plataformas de desarrollo y grabadores de microcontroladores AVR. Entre ellos los mejores son la plataforma de desarrollo STK500 mostrada en la Figura 4.1. y el grabador AVRISP[1] mostrado en la Figura 4.2. distribuidos por Atmel. Además son emuladores con muchos elementos extras para realizar pruebas bastante completas con el microcontrolador AVR.

[1] AVRISP significa AVR In-System Programmer

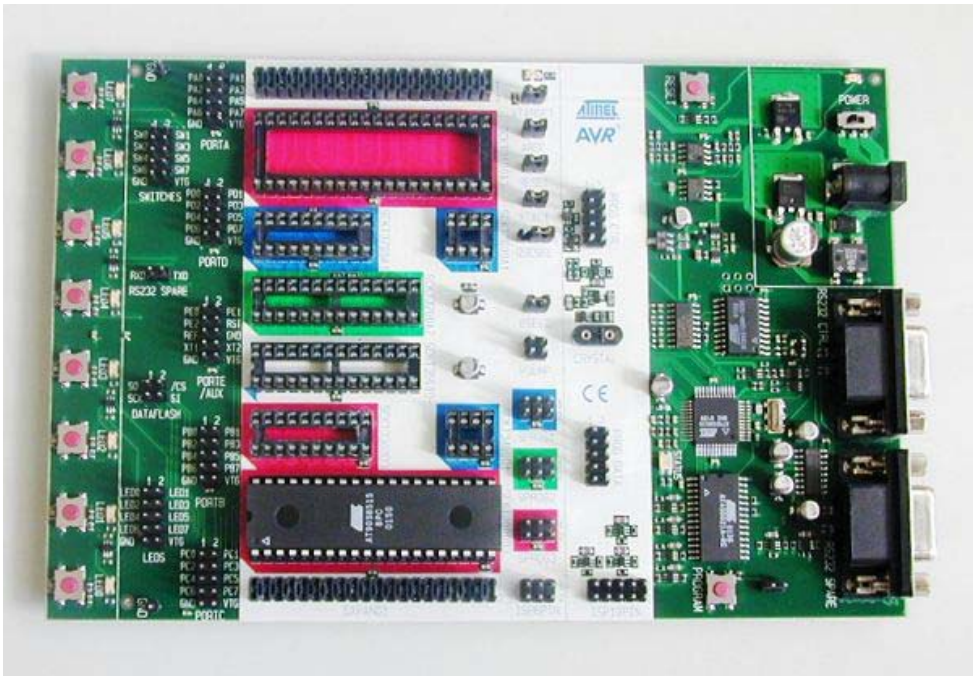


Figura 4.1. Vista superior del STK500

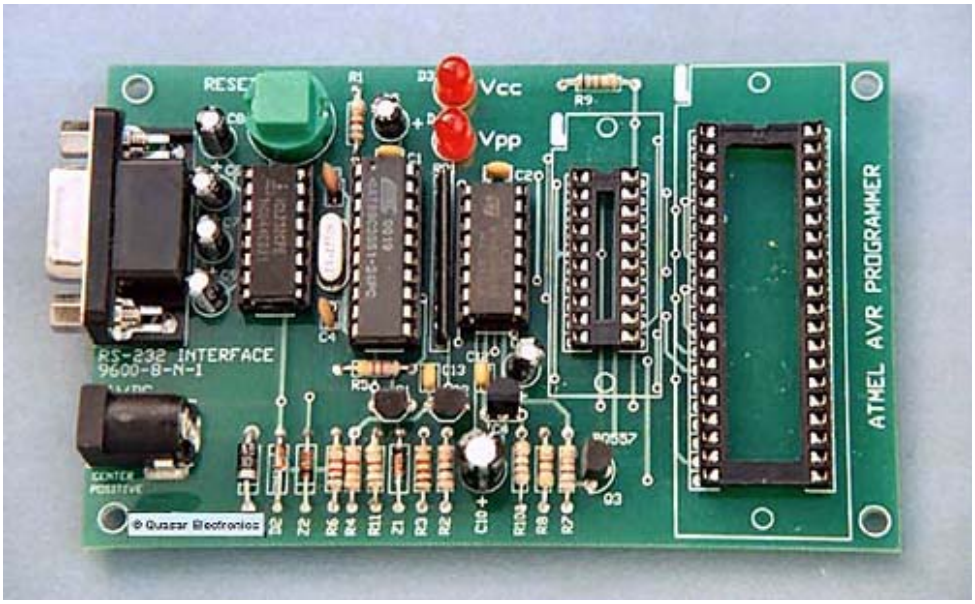


Figura 4.2. Vista superior del AVRISP

También existe la opción de utilizar la interfase J-TAG. Finalmente existen otros grabadores desarrollados por otras compañías como Kanda Systems. La

desventaja de todos estos grabadores es su precio, que es bastante elevado y además deben ser importados ya que no los venden en nuestro mercado.

Otro inconveniente con muchos programadores de microcontroladores AVR es que muchos de ellos son algo anticuados y no trabajan con los modernos microcontroladores AVR de la sub-familia ATmega, en especial con el microcontrolador ATmega32 utilizado en este Proyecto.

4.3. PonyProg2000.

Considerando precios, posibilidades de adquisición en el mercado y funcionalidad se optó por el grabador PonyProg2000 desarrollado por la compañía italiana Lanconelli Open Systems para la elaboración de este proyecto. PonyProg2000 es un programa para Windows que permite grabar y leer microcontroladores y que trabaja con su propio ISP (In-System Programmer). No solo el programa es gratuito y de libre distribución, sino que además el diseño del ISP está disponible en Internet y es bastante simple de construir por lo que no hay necesidad de comprarlo.

Tanto el programa como el diseño del circuito grabador pueden ser obtenidos de la página de Internet de la compañía Lanconelli Open Systems que los distribuye[2]. Además este programa trabaja perfectamente con el archivo .HEX generado por CodeVisionAVR.

4.3.1. Descripción.

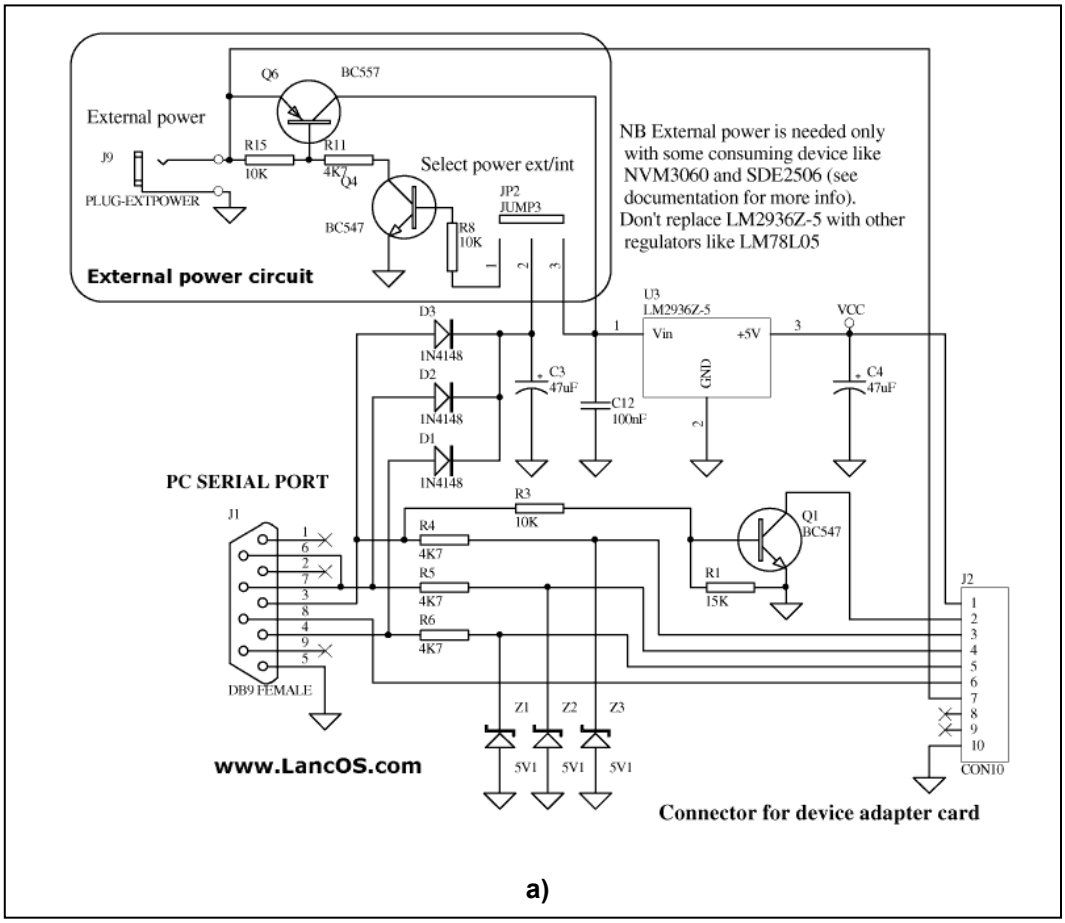
PonyProg2000 fue desarrollado por la compañía italiana Lanconelli Open Systems. Existen varias versiones de PonyProg, sin embargo se escogió la más actual, PonyProg2000 Versión 2.06f Beta debido a que versiones anteriores no son compatibles con los ATmega ya que siguen trabajando con los antiguos AT90S de Atmel que ya salieron de circulación.

[2] Para mayor información visitar la página de Internet www.lancos.com/prog.html.

4.3.2. In-System Programmer (ISP).

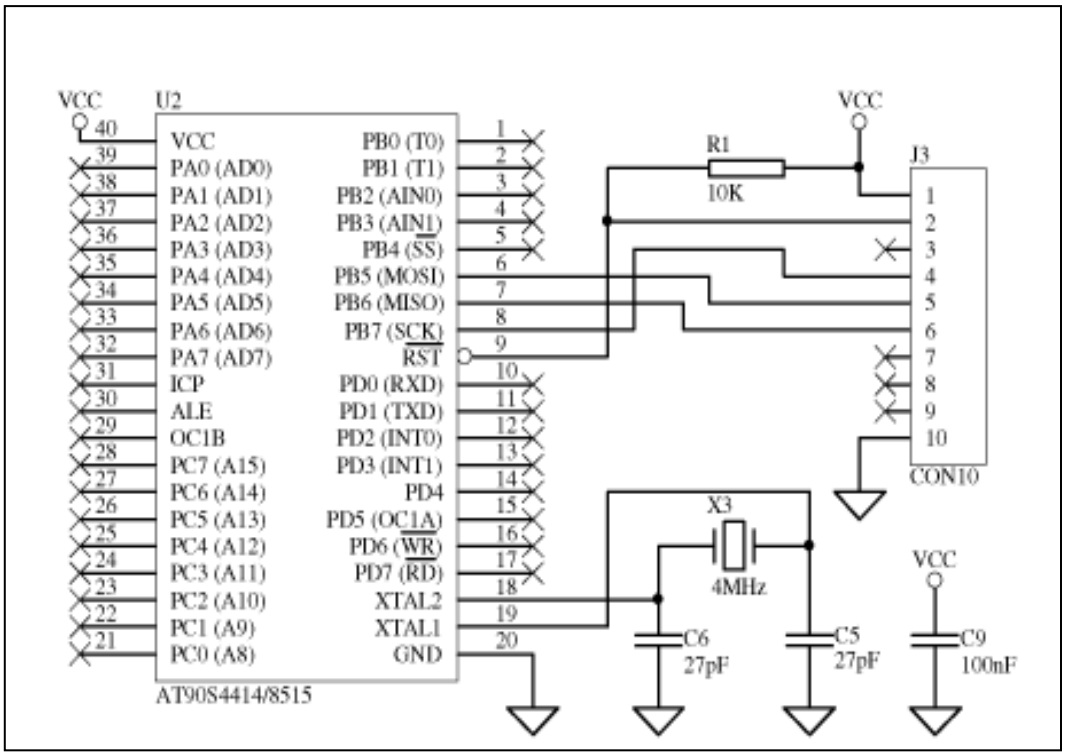
Un ISP es un grabador que no requiere que el microcontrolador sea extraído del circuito de la aplicación para grabar o leer su código de programa, lo cual facilita el manejo del microcontrolador. El diseño del ISP utiliza elementos fáciles de conseguir y ocupa poco espacio. Se divide en dos bloques, el circuito principal y el adaptador del microcontrolador AVR. También puede trabajar con PIC y otros tipos de microcontroladores. El circuito principal es siempre el mismo y el adaptador del microcontrolador depende del tamaño del microcontrolador que determina a su vez el tamaño del socket utilizado.

A continuación se muestra el diagrama del circuito del ISP en la Figura 4.3. junto con la lista de elementos requeridos para su construcción en la Tabla 4.1.[3].



a)

[3] En las Figura 4.3. y en la Tabla 4.1. se muestra el diseño original que se puede encontrar en www.lancos.com/prog.html. En el desarrollo de este Proyecto se hicieron ciertas modificaciones.



b)

Figura 4.3. a) Diseño del circuito principal b) Adaptador del microcontrolador

Resistencias:

- 3 x 4K7
- 2 x 10K
- 1 x 15K

Capacitores:

- 2 x 47uF
- 2 x 100nF
- 2 x 27pF

Diodos:

- 3 x 1N4148
- 3 x Zener 5V1
- Otros Semiconductores:
- 1 x BC547 (NPN transistor)
- 1 x LM2936Z-5 (voltage regulator)

Xtal:

- 1 x 4MHz

Conectores:

- 1 x DB9 Female
- 1 x Strip female SIL 10p
- 1 x Strip male SIL 40p

Sockets:

- 1 x DIL 40p

Tabla 4.1. Lista de elementos del ISP de PonyProg2000

Aunque en la Figura 4.3. a) se recomienda no reemplazar el LM2936Z-5 con un LM78L05, en nuestro mercado no existe el integrado LM2936Z-5 así que se utilizó el más próximo que es el LM78L05 que funciona sin problemas con el ATmega32 utilizado en este Proyecto.

En la Figura 4.3. b) se muestra el diagrama del adaptador del microcontrolador AT90S4414/8515 debido a que no hay todavía un diseño propio para el ATmega32 en la página de Internet de Lanconelli Open Systems.

Sin embargo considerando que ambos tipos de microcontroladores (el AT9024414/8515 y el ATmega32) tienen 40 pines distribuidos de manera casi similar y ambos utilizan los mismos pines del Puerto B para su grabación en serie se puede usar el mismo diseño.

Lo único que se debe cambiar es la ubicación de los pines de VCC y GND que es diferente. Debido a que el ATmega32 tiene su propio reloj interno, el cristal de 4 MHz también puede ser omitido. Así mismo ya que solo se va a trabajar con el ATmega32, el adaptador del microcontrolador puede ser implementado en el mismo circuito de aplicación.

La ventaja de usar un ISP es que permite grabar el microcontrolador sin tener que extraerlo del circuito de aplicación del proyecto. Sin embargo para poder grabar el microcontrolador es necesario que los pines PB5, PB6 y PB7 estén desconectados del resto del circuito de aplicación. Para este fin se puede utilizar un *jumper* para conectar y desconectar dichos pines del resto del circuito de aplicación durante el proceso de grabación.

Con todas las modificaciones mencionadas, el circuito principal junto con el adaptador del ATmega32 implementados de manera conjunta en este Proyecto, quedan establecidos como se muestran en la Figura 4.4.

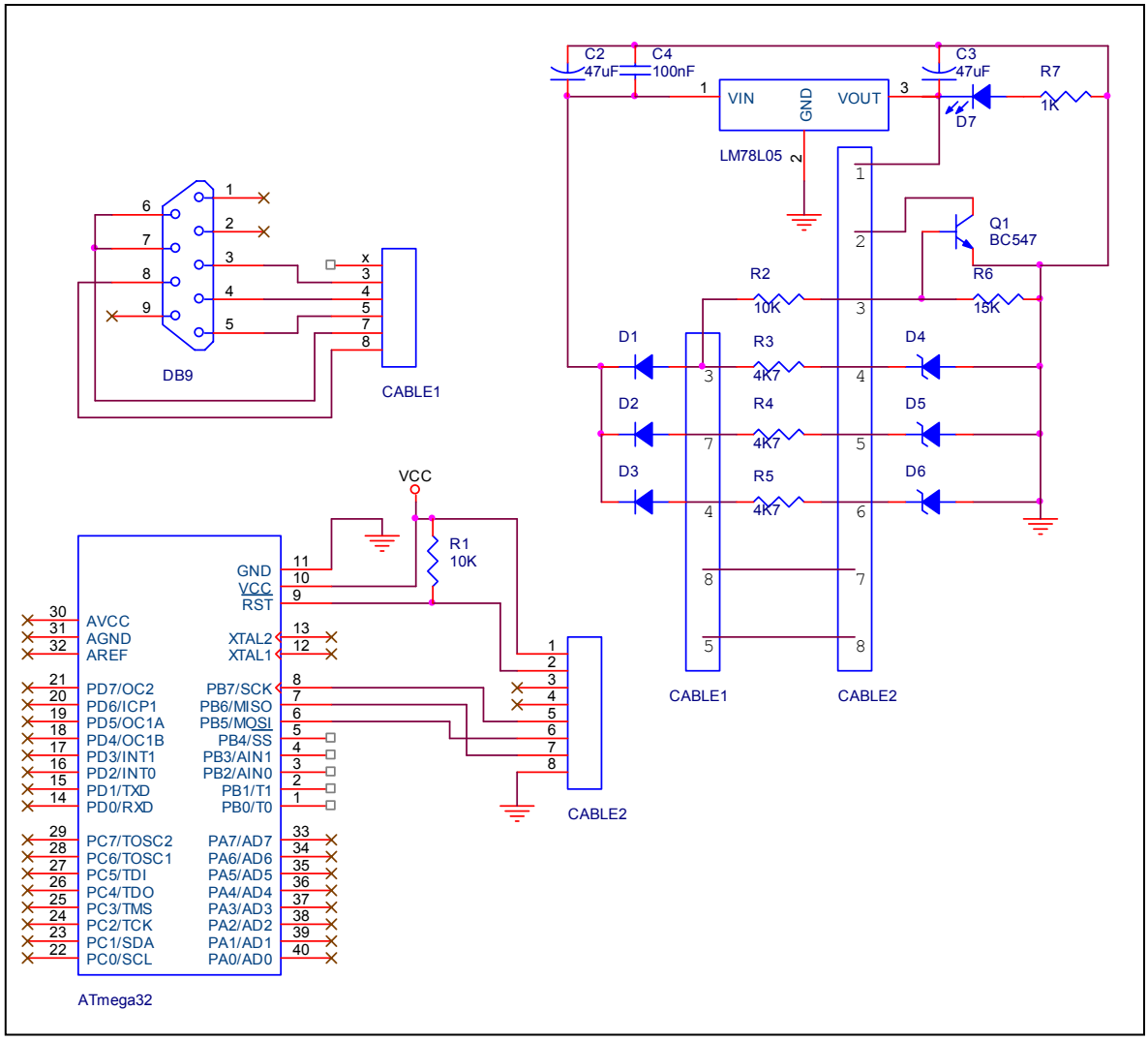


Figura. 4.4. Esquema del ISP implementado

En la Figura 4.5. se muestra el circuito principal construido:

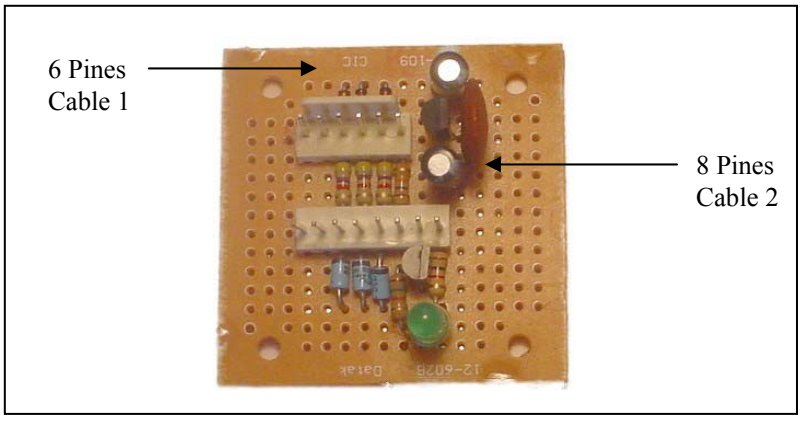


Figura 4.5. ISP construido

Los Cables 1 y 2 deben ser armados como se muestra en la Figura 4.6. a continuación:

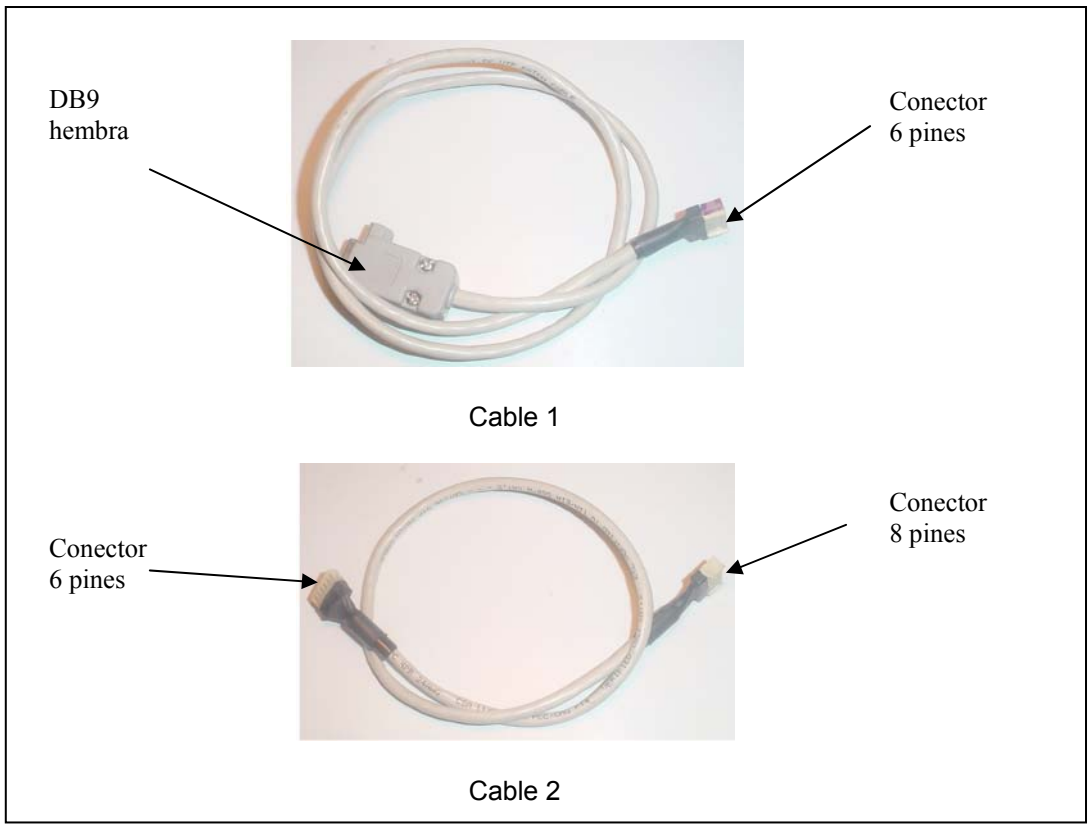


Figura 4.6. Cables del ISP

El circuito principal puede ser colocado dentro una caja para su protección como se muestra en la Figura 4.7. a continuación:

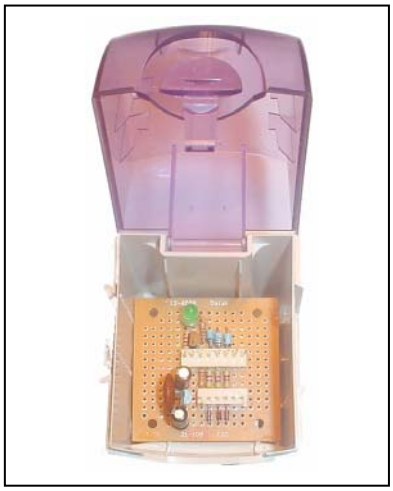


Figura 4.7. Caja protectora del ISP

En la Figura 4.8. se muestran la manera en que se deben conectar los cables para grabar el ATmega 32 del Display Rotativo que es la aplicación de este proyecto:

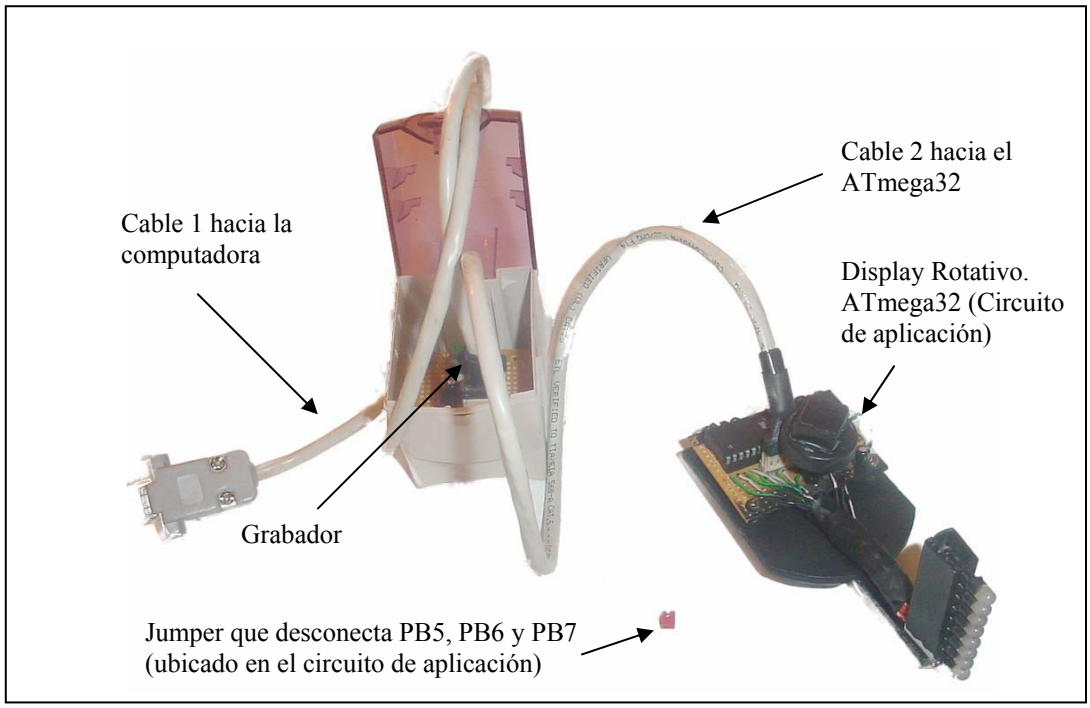


Figura 4.7. Conexiones del ISP

Como se puede apreciar en la Figura 4.8. el circuito principal se conecta al puerto serial de la computadora a través del Cable 1 y al circuito de aplicación que contiene el ATmega32 a través del Cable 2.

4.3.3. Configuración de PonyProg2000 y del ISP.

Una configuración errónea de PonyProg2000 o del ISP podría dañar permanentemente el microcontrolador AVR por lo que se debe tener bastante precaución en este punto. En la Figura 4.8. se muestra la ventana principal de PonyProg2000 y contiene las siguientes partes:

a) **Menús de Comandos.** El Menú de Comandos de PonyProg2000 es similar al de cualquier otro programa de Windows con las opciones: File, Project, Edit, Device, Command, Script, Utility, Help y Window.

b) Barra de Herramientas. Todos los comandos del Menú pueden también ser ejecutados de una manera más rápida a través de estos Botones de Comandos.

c) Ventana del archivo .HEX. En esta ventana se puede ver el código en hexadecimal del archivo .HEX que se leyó o se va a escribir en el microcontrolador.

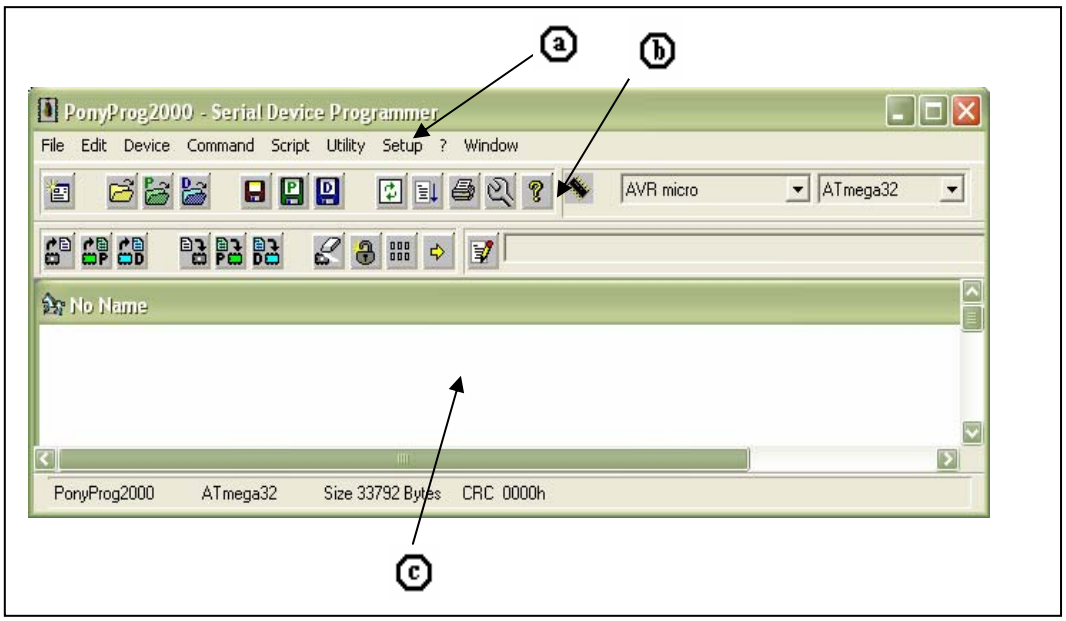


Figura 4.8. Ventana principal de PonyProg2000

Lo primero que se debe hacer es escoger el microcontrolador adecuado, en este caso el ATmega32 dentro de la lista de AVR micro en la esquina superior derecha de la ventana principal de PonyProg2000.

A continuación se debe configurar el ISP. Primero se debe conectar el ISP al puerto serial, el ATmega32 no debe estar conectado. Después a través del comando *Setup/Interface Setup* o a través del botón *Setup* se ingresan los parámetros correspondientes al ISP utilizado mediante la ventana mostrada en la Figura 4.9. El *I/O port Setup* debe ser Serial/SI Prog API. Luego se escoge el puerto serial utilizado, generalmente el COM1, y se debe dejar los *checkboxs* en blanco. Finalmente se presiona *OK*.

Si se desea se puede probar que el ISP funcione correctamente presionando el Botón *Probe*, luego de lo cual debe salir un mensaje de *OK* si no hay problemas o de error si hubo alguno.

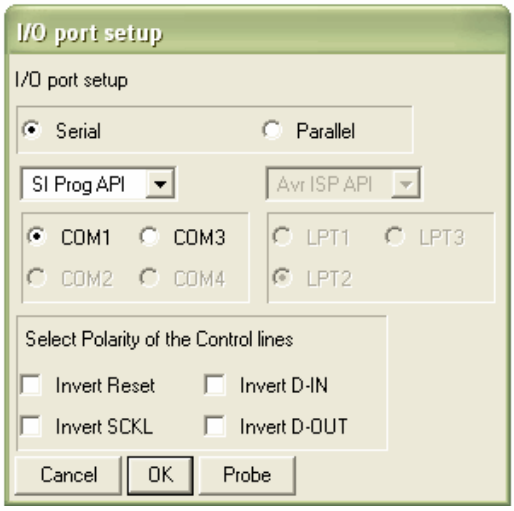


Figura 4.9. Ventana I/O port Setup

4.3.4. Cargando un archivo .HEX en un microcontrolador AVR.

Para cargar un archivo con extensión .HEX en un microcontrolador AVR se deben seguir los siguientes pasos:

- Escoger el microcontrolador adecuado, en este caso el ATmega32 dentro de la lista de AVR micro en la esquina derecha superior de la ventana principal de PonyProg2000.

- Desconectar la alimentación del microcontrolador AVR así como los pines del Puerto B involucrados en la grabación del Microcontrolador. En el caso específico de este proyecto se debe apagar el switch principal del Display Rotativo y sacar el jumper que habilita a los Puertos del microcontrolador AVR.

- Conectar el microcontrolador al grabador a través del Cable 2, comprobando que el borde de tierra, marcado con negro, corresponda en cada lado.

- Conectar el grabador a la computadora a través del Cable 1 que tiene en un extremo un conector DB9 hembra. Comprobar que la conexión del otro extremo el borde de tierra, marcado con negro, corresponda en cada lado.

- Abrir el archivo .HEX a través del comando *File/Open Device File*. Se recomienda no usar las opciones *Open Program File* ni *Open Data File* ya que estas opciones no aseguran que se grabe el programa correctamente en el microcontrolador AVR.

- Grabar el archivo .HEX a través del comando *Command/Write All*. Se recomienda no usar las opciones *Write Program* ni *Write Data* ya que estas opciones no aseguran que se grabe el programa correctamente en el microcontrolador AVR.

La duración del proceso de grabación del microcontrolador AVR depende del tamaño de la Memoria Flash del microcontrolador. Luego de la grabación, PonyProg2000 verifica que se haya grabado correctamente el programa. Si no hubo problemas aparecerá el mensaje *Write Successful*, caso contrario se verá un mensaje de error y en consecuencia se debería comprobar cada uno de los pasos mencionados.

4.3.5. Cargando un archivo .EEP en un microcontrolador AVR.

Una de las grandes cualidades de *CodeVisionAVR* es que además de generar el archivo .HEX que contiene el código del programa que se guardará en la memoria Flash del microcontrolador, también crea un archivo .EEP que contiene los datos a grabar en la EEPROM del microcontrolador. Para cargar un archivo con extensión .EEP en un microcontrolador AVR se deben seguir los siguientes pasos:

- Escoger el microcontrolador adecuado, en este caso el ATmega32 dentro de la lista de AVR micro en la esquina derecha superior de la ventana principal de PonyProg2000.

- Desconectar la alimentación del microcontrolador AVR así como los pines del Puerto B involucrados en la grabación del Microcontrolador. En el caso específico de este proyecto se debe apagar el switch principal del Display Rotativo y sacar el jumper que habilita a los Puertos del microcontrolador AVR.

- Conectar el microcontrolador al grabador a través del Cable 2, comprobando que el borde de tierra, marcado con negro, corresponda en cada lado.

- Conectar el grabador a la computadora a través del Cable 1 que tiene en un extremo un conector DB9 hembra. Comprobar que la conexión del otro extremo el borde de tierra, marcado con negro, corresponda en cada lado.

- Abrir el archivo .EEP a través del comando *File/Open Data File*. Se recomienda no usar las opciones *Open Program Memory File* ni *Open Device File* ya que estas opciones no aseguran que se graben los datos correctamente en el microcontrolador AVR.

- Grabar el archivo .EEP a través del comando *Command/Write Data*. Se recomienda no usar las opciones *Write Program* ni *Write All* ya que estas opciones no aseguran que se graben los datos correctamente en el microcontrolador AVR.

La duración del proceso de grabación del microcontrolador AVR depende del tamaño de la EEPROM del microcontrolador, sin embargo es mucho menor que cuando se graba un archivo .HEX. Luego de la grabación, PonyProg2000 verifica que se haya grabado correctamente el programa. Si no hubo problemas aparecerá el mensaje *Write Successful*, caso contrario se verá un mensaje de

error y en consecuencia se debería comprobar cada uno de los pasos mencionados.

En caso de no grabar el archivo .EEP en el microcontrolador después de grabar el archivo .HEX, la memoria EEPROM quedará llena de valores 0xFF. Si no se inicializa alguna variable tipo *eeprom* en el programa correspondiente al archivo .HEX grabado en el microcontrolador, al momento de grabar el archivo .EEP en dicho microcontrolador, el sector de la EEPROM asignado a dicha variable se llenará de valores 0x00.

4.3.6. Bits de Configuración y de Seguridad.

El Reloj de Sistema utilizado por el microcontrolador AVR junto con otras características especiales y bloqueos de acceso al microcontrolador, se configuran a través de los Bits de Configuración y Seguridad mostrados en la Figura 4.10.

Los Bits de Configuración y de Seguridad se pueden leer o escribir a través del comando *Command/Security and Configuration Bits*.

Figura 4.10. Ventana Configuration and Security Bits

En la ventana mostrada anteriormente, cuando el *checkbox* está vacío significa que está configurado con un 1 y si está marcado, con un 0. La función de cada bit para configurar el Reloj del Sistema ha sido explicada anteriormente (2.4. Reloj del Sistema) y la función de cada bit de Configuración y Seguridad se explica en el manual del microcontrolador AVR utilizado, en este caso el ATmega32. Se debe tener cuidado al cambiar estos bits ya que se podría bloquear involuntariamente el acceso al microcontrolador AVR. La configuración utilizada en este proyecto se verá en capítulos posteriores.

CAPÍTULO 5

CODIFICACIÓN SIRC

5. CODIFICACIÓN SIRC (SONY INFRARED CODING).

El primer paso para comenzar el diseño del Display Rotativo es encontrar la manera de comunicarse con el mismo ya sea para configurarlo o ingresar un mensaje de texto. Debido a que el Display Rotativo se encuentra en constante movimiento, una conexión por cable sería imposible. La única forma de hacer cambios sería deteniendo las partes que giran, entre ellas el microcontrolador, extraer el microcontrolador para conectarlo al ISP, realizar las modificaciones necesarias y colocar el microcontrolador nuevamente en su lugar. Debido a que hacer esto resultaría poco práctico e incomodo para el usuario, se debe buscar otras alternativas.

Es fácil concluir que lo que se requiere es una interfase inalámbrica que comunique al microcontrolador con el usuario. Una opción sería usar transmisores y receptores RF[43], que brindarían una comunicación a gran distancia, pero esto no es necesario ya que el usuario y el dispositivo generalmente están a pocos metros de distancia. A corta distancia la solución más común es el uso de comunicación Infra-Roja (IR).

Para evitar tener que construir un transmisor IR para comandar al microcontrolador que trabajaría como receptor, se puede utilizar un Control Remoto de televisión y simplemente programar en el microcontrolador las funciones a realizar cada vez que se presione un botón del Control Remoto. De

[43] RF significa Radio Frequency

esta manera inclusive se disminuye las fuentes de errores al momento de entablar una comunicación IR ya que se asegura que el transmisor está construido bajo ciertos estándares de calidad.

Los Controles Remotos más comunes son los Controles Universales fabricados por Sony. Estos controles utilizan la Codificación SIRC que básicamente es una modulación digital por ancho de pulso (DPWM[44]). En este proyecto se utilizará el Control Remoto Universal Sony RM V8.

Antes de programar el microcontrolador, es necesario conocer más sobre la Codificación SIRC para saber el formato de datos que recibirá el microcontrolador cada vez que se presione un botón del Control Remoto Universal.

5.1. Emisión IR.

La transmisión IR utiliza la emisión de luz con una longitud de onda superior a la luz visible (el color rojo está aproximadamente en 630 nm y la luz utilizada en la transmisión IR está al rededor de 945 nm) como se muestra en la Figura 5.1.[45]

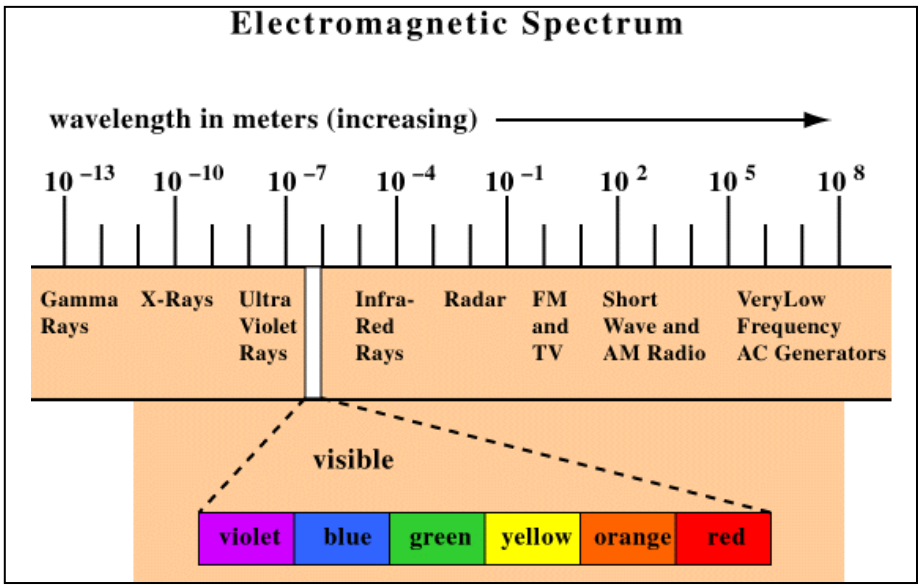


Figura 5.1. Espectro Electromagnético

[44] DPWM significa Digital Pulse-Width Modulation
[45] La Figura 5.1. fue obtenida de la página de Internet http:// www.cramersoftware.com/ illustr.html

En el ambiente existen muchas fuentes de transmisión IR tales como el Sol, fuentes luminosas artificiales como los focos incandescentes y fluorescentes y además los cuerpos cálidos que emiten ondas IR junto con el calor. Debido a que el calor y la emisión IR están relacionados, se puede utilizar sensores IR para la medición de temperatura o movimiento.

La emisión IR no puede atravesar cuerpos muy gruesos. Otra característica de la emisión IR es que utilizando el transmisor adecuado (LEDs IR con una cubierta alrededor) es altamente directiva y además no puede atravesar cuerpos muy gruesos, ambas características la diferencia de la emisión de ondas electromagnéticas de RF lo cual hace que sus aplicaciones sean distintas.

La emisión IR (Infrarroja) es muy útil en el caso de que la comunicación directa por cable no sea posible. La transmisión y recepción IR generalmente se utiliza para controlar o comunicar equipos que cumplan con las siguientes características:

- Dificultad de ser conectados a través de un cable debido a razones mecánicas o de comodidad.

- Una ubicación no muy distante, de preferencia en la misma habitación.

- Posibilidad de línea de vista. Si algún objeto interfiere entre el transmisor y el receptor IR no debe ser muy grueso.

Considerando los puntos mencionados anteriormente, el Display Rotativo es un equipo ideal para el uso de comunicación IR.

5.2. Módulo Detector IR.

Equipos electrodomésticos como Televisores y Equipos de Audio y Video utilizan Módulos Detectores IR para recibir comandos desde Controles Remotos.

Estos módulos reciben la señal de IR con lógica inversa, es decir, se mantienen en alto o 1 lógico mientras no hay señal y cambian a bajo o 0 lógico cada vez que reciben un pulso IR. Un Módulo Detector IR, como el mostrado en la Figura 5.2.[46], tiene tres terminales V_{OUT}, GND y V_{CC}.

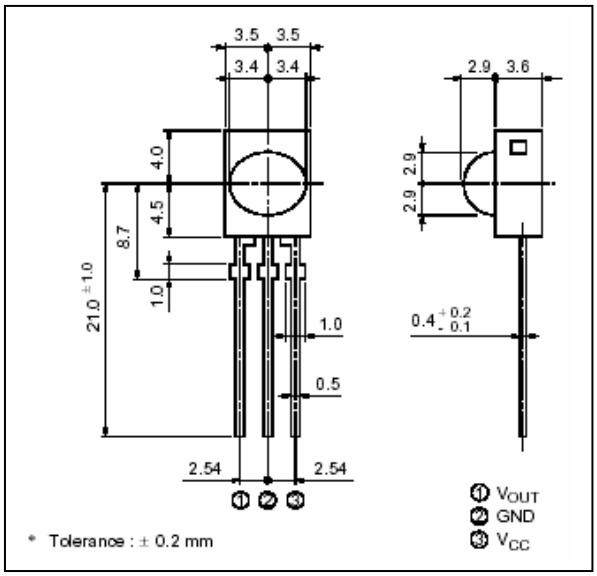


Figura 5.2. Esquema del Módulo Detector IR

Un inconveniente a enfrentar en el uso de detectores IR es que el ambiente contiene varios emisores de luz IR, como por ejemplo el sol, luces artificiales, focos incandescentes, lámparas fluorescentes, etc.

Para evitar recibir estás señales no deseadas, los Módulos Detectores IR dentro de los equipos domésticos reciben la señal IR de una portadora modulada por la información que se desea enviar. Esta señal portadora tiene una frecuencia fija de alrededor de 38 KHz. Al llegar la señal al Módulo Detector IR la portadora es suprimida y se obtiene la información transmitida como se muestra en la Figura 5.3.[47]

[46] La Figura 5.2. fue obtenida de la hoja técnica del Módulo detector IR IS1U621.
[47] La Figura 5.3. fue obtenida de la hoja técnica del Módulo detector IR IS1U621.

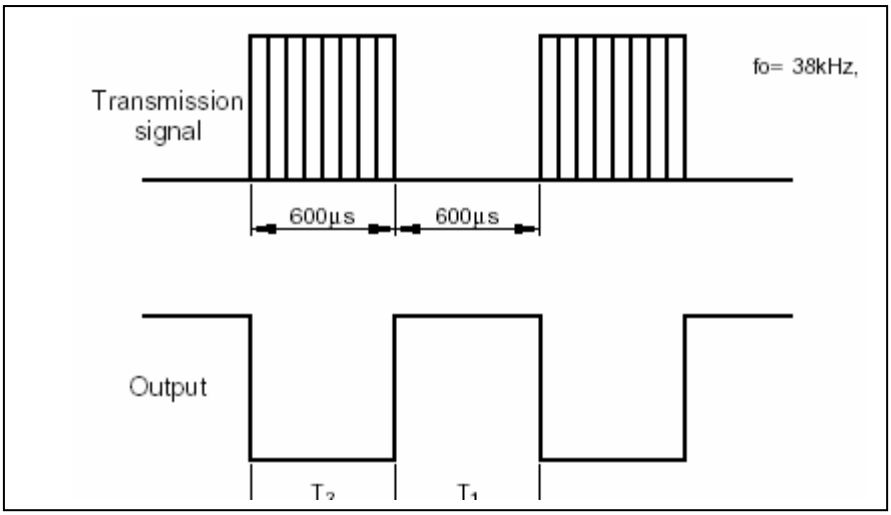


Figura 5.3. Señal transmitida y señal demodulada en el Módulo Detector IR

Otra ventaja de estos Módulos Detectores IR es que pueden trabajar con lógica TTL directamente y sin la necesidad de etapas de amplificación. Simplemente se debe conectar los pines correspondientes y además un Capacitor de 47 µF entre VCC y GND para suprimir el ruido tal como se indica en la sección 5.4.2.

5.3. Descripción de la Codificación SIRC.

SIRC es una codificación por ancho de pulso para poder enviar paquetes de datos. Los receptores de IR de equipos domésticos como Televisores, Radios, CD Players trabajan con lógica inversa, es decir, cuando no reciben una señal infrarroja están en estado alto o 1 lógico y cuando la reciben cambian a estado bajo o 0 lógico.

La Codificación SIRC consta de paquetes de 13 pulsos que equivalen a un STARTBIT y 12 BITS. Generalmente se denomina T al tiempo mínimo que dura un pulso en bajo, este valor suele estar entre 550 y 650 µs[48].

[48] Información obtenida de la página de Internet Remote Control Technical Specifications SONY. www.ustr.net/infrared/sony.shtml

5.3.1. Transmisor SIRC.

La señal de un transmisor IR bajo una Codificación SIRC es básicamente un tren de pulsos como el mostrado en la Figura 5.4. Cuando el transmisor IR no está transmitiendo, la señal se mantiene en bajo. Cuando el transmisor emite una señal envía un tren de pulsos como el que se muestra en la gráfica.

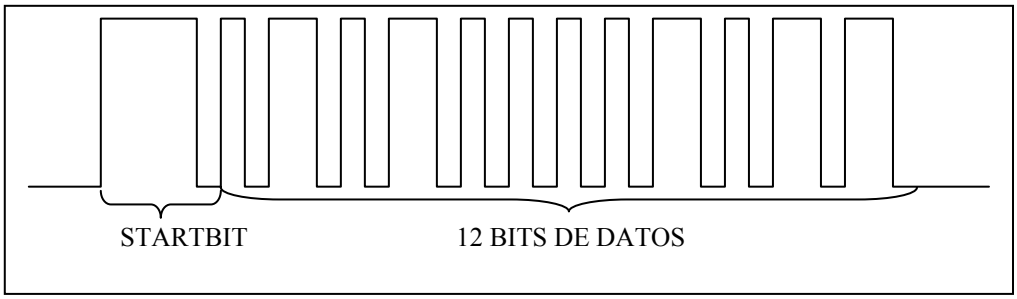


Figura 5.4. Señal SIRC transmitida

Como se puede apreciar en el gráfico anterior, cada bit consta de un tiempo en alto y uno en bajo. El tiempo en bajo es constante e igual a T, mientras que el tiempo en alto puede ser igual a T, 2T y 4T dependiendo del tipo de bit que sea. El STARTBIT indica el inicio del paquete enviado y la duración del tiempo en alto es de de 4T. La diferencia entre un 1 lógico y un 0 lógico en los bits de datos depende del ancho del pulso en alto, siendo de T para un 0 lógico y de 2T para un 1 lógico. A continuación se presentan los diferentes tipos de bits transmitidos en la Figura 5.5.

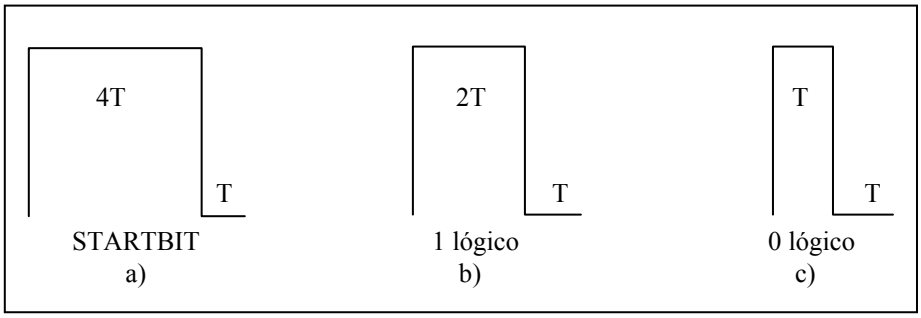


Figura 5.5. Bits transmitidos: a) STARTBIT b) 1 lógico c) 0 lógico

5.3.2. Receptor SIRC.

Los receptores IR funcionan de manera inversa a los transmisores. Cuando no reciben una señal se mantienen en alto o 1 lógico. Cuando reciben un pulso cambian a estado bajo mientras dure el pulso, por lo que el tren de pulsos recibidos queda como se muestra en la Figura 5.6.

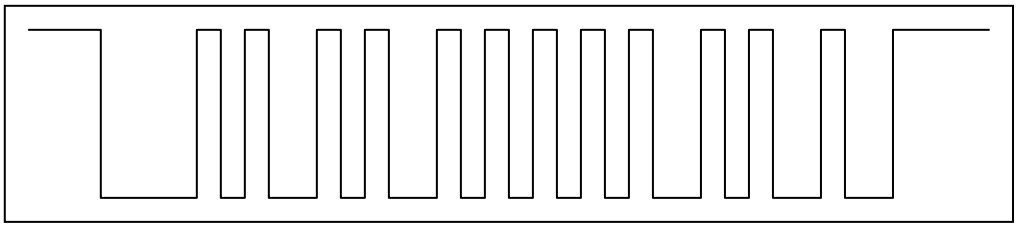


Figura 5.6. Señal SIRC recibida

De ésta forma los pulsos para STARTBIT, 0 y 1 se reciben de la siguiente manera en el receptor IR según se muestra en la Figura 5.7.

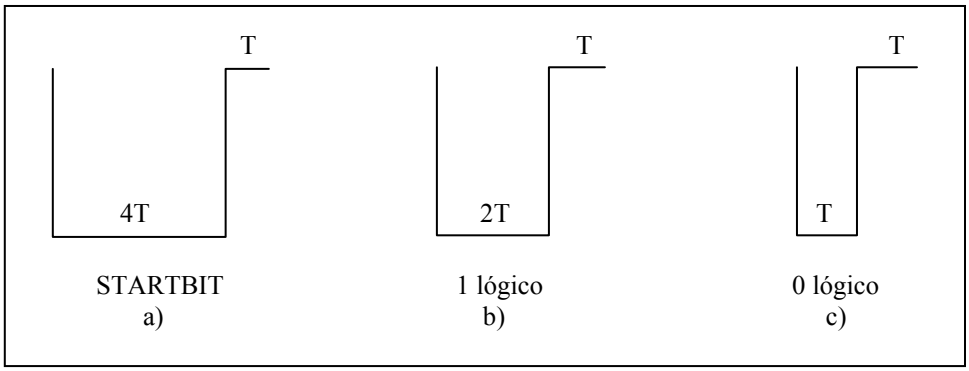


Figura 5.7. Bits recibidos a) STARTBIT b) 1 lógico c) 0 lógico

5.3.3. Codificación del Tren de Pulsos.

El primer bit denominado STARTBIT indica el inicio de la transmisión de datos. Los 12 bits de datos enviados posteriormente se dividen en 7 bits de Código y 5 bits de Dirección.

Los primeros 7 bits son de Código, el primer bit enviado es el LSB y el último es el MSB. Estos bits sirven para indicar que botón se ha presionado en el Control Remoto y por lo tanto el correspondiente comando a ejecutarse.

Los siguientes 5 bits son de Dirección, el primer bit enviado es el LSB y el último es el MSB. Estos bits sirven para indicar que equipo debe ejecutar la instrucción, ya sea un VCR, un Televisor, un CD Placer, etc. En el caso de configurar el Control Remoto para manejar un Televisor estos 5 bits se mantendrán constantes por lo que no es necesaria su decodificación. A continuación se muestra en la Figura 5.8. un esquema de las partes que conforman el tren de pulsos en la Codificación SIRC.

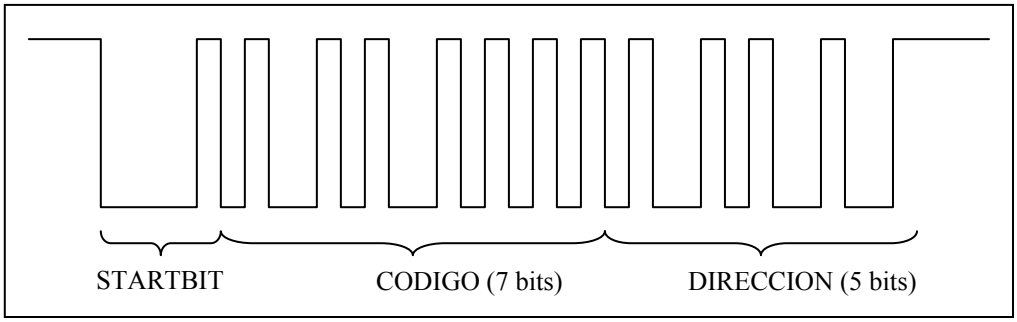


Figura 5.8. Tren de pulsos en la Codificación SIRC (Receptor)

5.4. Diseño e Implementación de un Detector SIRC para Control Remoto Universal RM V8.

Como ejercicio experimental para aprender a manejar el microcontrolador ATmega32 con las herramientas de desarrollo descritas en capítulos anteriores se puede proponer el diseño de una aplicación que reciba la señal de un Control Remoto Universal RM V8 fabricado por la Sony y muestre en una pantalla LCD el botón presionado.

Para llegar al objetivo final es recomendable ir por pasos, de esta manera se logra un mejor entendimiento de la Codificación SIRC del Módulo Detector IR conectado al ATmega32 y de sus herramientas de desarrollo.

5.4.1. Utilización del Timer 1.

Uno de los objetivos propuestos para realizar un decodificador IR y presentar un mensaje en un LCD es utilizar un solo Timer. El Timer1 permite una amplia variedad de usos en este programa debido a que maneja un registro de 16 bits (FFFF) y puede medir periodos de tiempo en µs y ms llegando hasta 2,097 s con un reloj interno de 8 MHz. Además permite generar interrupciones cuando ocurre un evento fuera del ATmega32 a través del ICP (Input Capture Pin - PD6) pudiendo capturar flancos descendentes o ascendentes e inclusive cambios de nivel cuando sea necesario y permite medir el tiempo que transcurre entre eventos. Para mayor información sobre el uso del Timer1 se puede consultar la sección 2.9. Captura de datos con el Timer 1.

5.4.2. Conexión del Módulo Detector IR al ATmega32.

El Módulo Detector IR se conecta al pin ICP del ATmega32 (PD6) como se muestra en la Figura 5.9. El ICP activa la interrupción del Timer1 del ATmega32 cada vez que se recibe produce un cambio en el pin PD6 del microprocesador. Esta interrupción puede servir para futuras aplicaciones ya que es muy útil para medir el tiempo entre dos eventos.

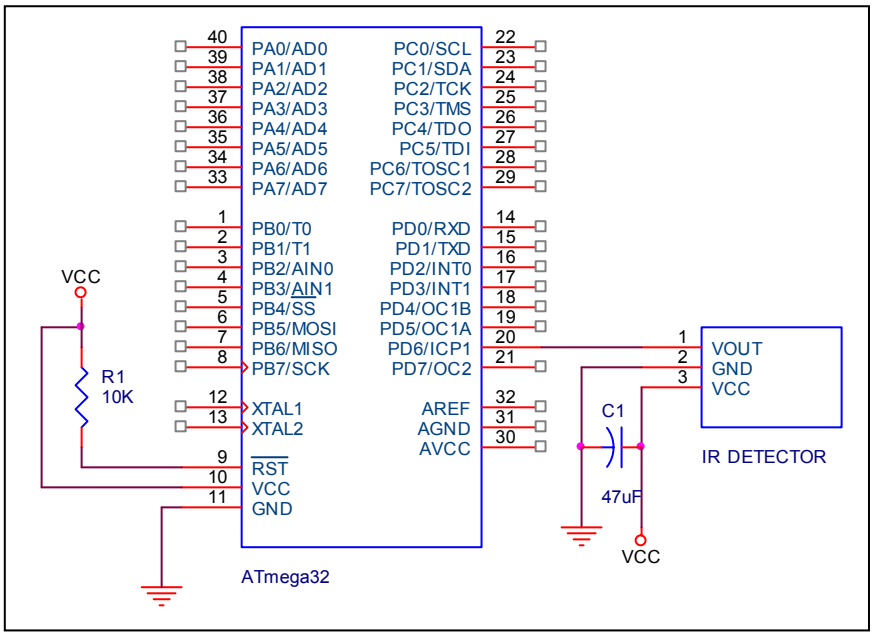


Figura 5.9. Conexión del Módulo Detector IR al ATmega32

5.4.3. Detección IR. Proyecto GREENLED.

El primer paso es determinar si se detectan los pulsos desde el Control Remoto y si el cambio en su nivel de voltaje es suficiente para interactuar con el ATmega32. Esto se determina simplemente realizando un programa que encienda o apague un LED en el pin PB4 cuando se reciba una señal desde el Módulo Detector IR como se muestra en la Figura 5.10. Como se trata de un solo LED se lo activará en alto, es decir, tomará la corriente desde el microcontrolador.

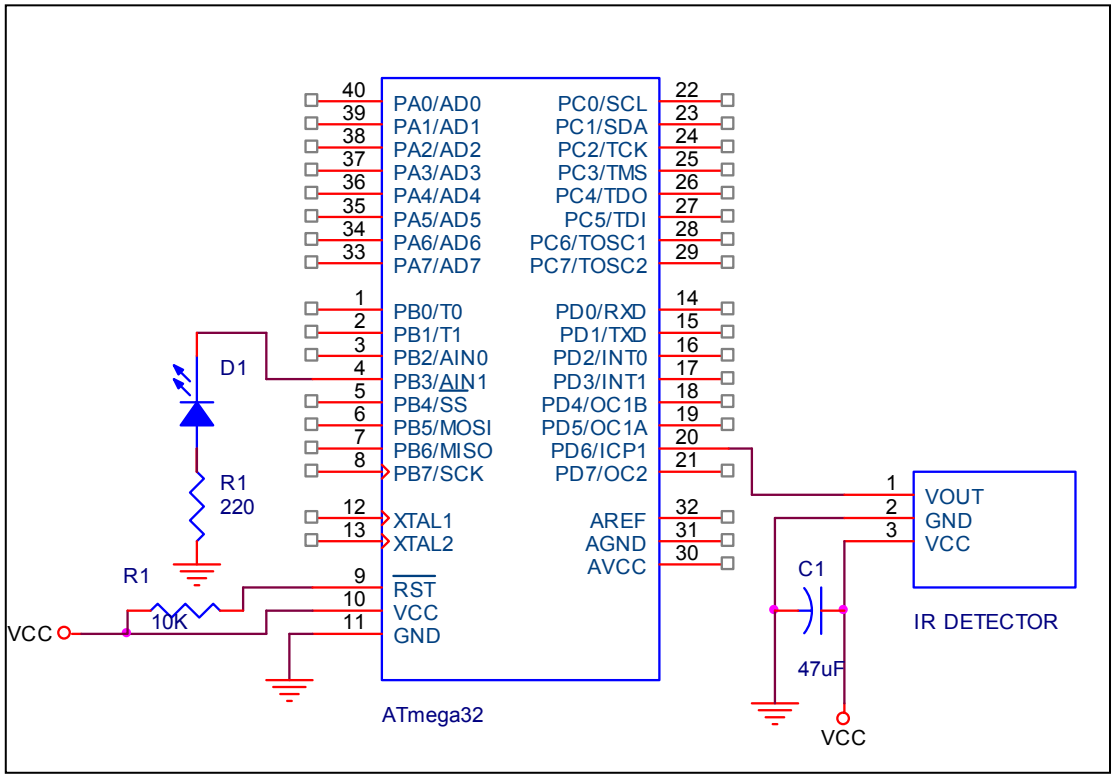


Figura 5.10. Circuito Detector IR

Lo primero que se debe hacer es configurar el ATmega32 para que utilice el reloj interno a 8 MHz. Esto se logra configurando los Bits de Configuración y Seguridad como ya se explicó en los capítulos respectivos: 2.4. Reloj del sistema y 4.3.6. Bits de Configuración y de Seguridad.

Luego utilizando CodeWizardAVR se crea el Proyecto *GREENLED* dentro de su carpeta respectiva. No se requiere hacer mayor modificación en las opciones

de la ventana de CodeWizardAVR, únicamente se debe establecer el tipo de microcontrolador como un ATmega32 a 8 MHz. También se debe escoger que el Puerto D es de entrada con resistencia de Pull-up y el Puerto B es de salida con un valor inicial en alto para todos los pines, la configuración de los demás puertos es indiferente. El código del archivo *GreenLed.c* es el siguiente:

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

Este programa recibe la señal IR-TV de un Control Remoto SONY RM V8 universal.
El sensor IR (de TV) está conectado al pin PD6 (ICP).

Se utiliza PORTB.4 para mostrar el nivel de PD6

*****************************************************/

#include <mega32.h>

void main(void)
{
PORTA=0x00;
DDRA=0xFF;

/// Puerto de salida que enciende el LED en alto y con un valor inicial en alto o 1 logico
PORTB=0xFF;
DDRB=0xFF;

PORTC=0x00;
DDRC=0xFF;

/// Puerto de entrada del Modulo Detector IR con Pull-Ups.
PORTD=0xFF;
DDRD=0x00;

/// Counter/Counter 0 initialization
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x0C;

/// Counter/Counter 1 initialization
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
```

```
MCUCSR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;

// Analog Comparator initialization
ACSR=0x80;
SFIOR=0x00;

while (1){
PORTB.4=PIND.6;
};
}
```

Como se puede apreciar en el código del archivo *GreenLed.c* se inicializa el ATmega32 con todas sus funciones apagadas. Lo único que se debe añadir es el código que le ordena al microcontrolador mostrar el nivel de PD6 en el pin PB4. Este programa a su vez sirve para comprobar que el ATmega32 y las herramientas utilizadas funcionen perfectamente.

Una vez que el archivo sea compilado y cargado en el ATmega32 se podrá ver que el LED conectado al pin PB4 titila cada vez que se presiona un botón del Control Remoto Universal RM8. No es posible ver los pulsos ya que estos son muy rápidos, por lo que se recomienda utilizar un osciloscopio, si se dispone de uno.

También se puede apreciar que a veces el LED titila aún cuando no se presiona ningún botón del Control Remoto. Esto se debe a las emisiones de luz IR en el ambiente que actúan como ruido. Para evitar esto se debe proteger el Módulo Detector IR con una cubierta que evite que la luz del ambiente incida directamente sobre el mismo.

5.4.4. Conteo de pulsos IR. Proyecto IRCOUNTER.

Una vez detectados los pulsos, el siguiente paso es contarlos. Una forma de hacerlo es contando los flancos descendentes ocurridos al enviar una señal IR desde el Control Remoto al pin PD6 que a su vez captura datos para el Timer1.

Luego de enviar un comando desde el Control Remoto Universal Sony RM V8 se puede utilizar un LCD para mostrar el número de flancos descendentes contados. El LCD puede ser actualizado cada vez que se presione un pulsador conectado a la interrupción externa INT0, indicando cuantos pulsos fueron contados. Este pulsador es experimental aunque posteriormente puede servir como un Reset general, pero al final se debe lograr que todo el funcionamiento del microcontrolador se pueda manejar remotamente a través del Módulo Detector IR.

Diseño del Circuito. Proyecto IRCOUNTER.

El circuito implementado es más complejo que el anterior ya que se va a utilizar un LCD. Las conexiones del LCD con el ATmega32, junto con el pulsador de actualización de datos conectado a INT0 se muestran en la Figura 5.11.

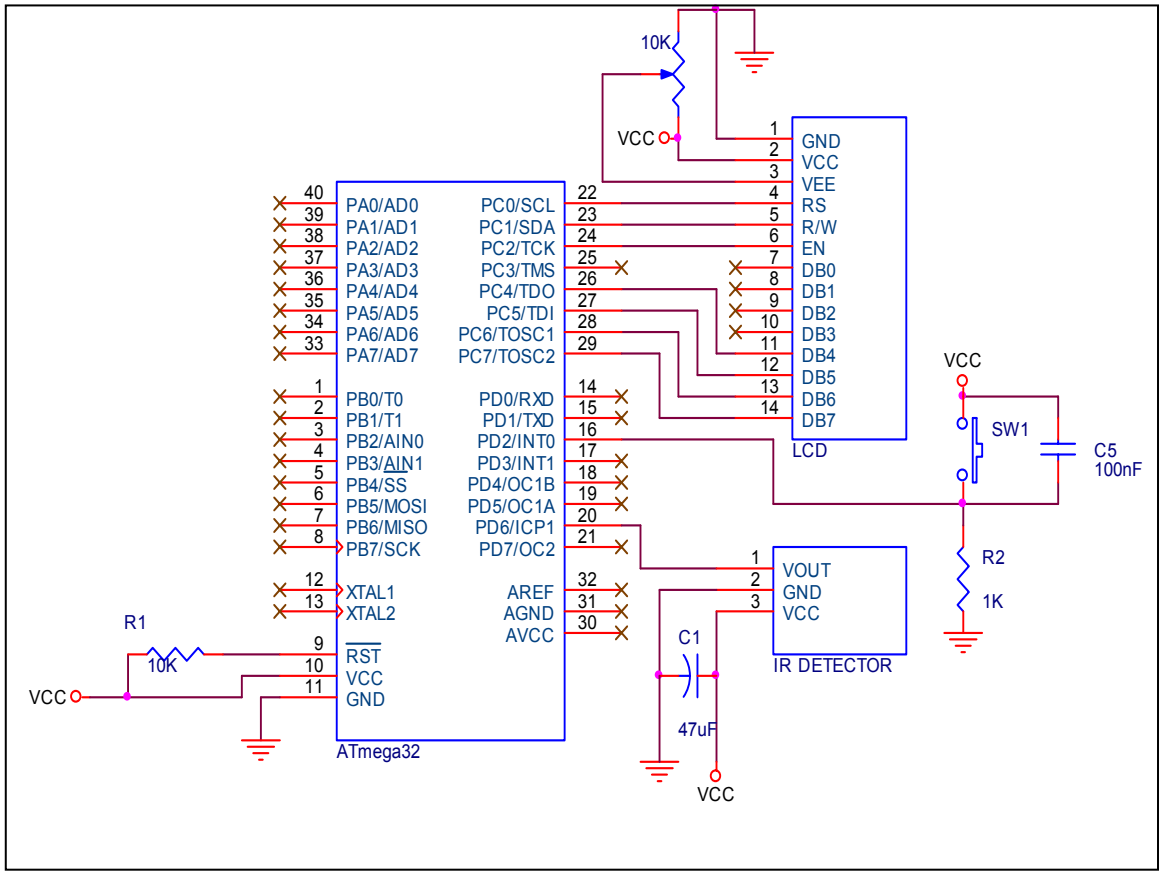


Figura 5.11. Circuito Contador de Pulsos IR

Funcionamiento del Programa. Proyecto IRCOUNTER.

Cuando el usuario pulse un botón del Control Remoto, éste enviará un tren de pulsos continuo mientras el botón esté presionado como se muestra en la Figura 5.12.

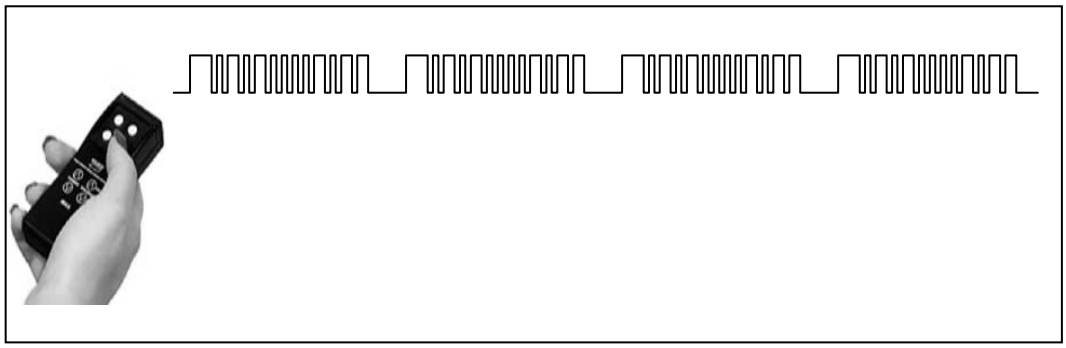


Figura 5.12. Tren de pulsos transmitidos

Para contar los pulsos basta con contar uno de sus flancos, ya sea el ascendente o el descendente. En este caso se contará los flancos descendentes como se muestra en la Figura 5.13., recordando que la señal enviada por el Módulo Receptor IR al ATmega32 es invertida, por lo cual lo que se cuenta es pulsos invertidos.

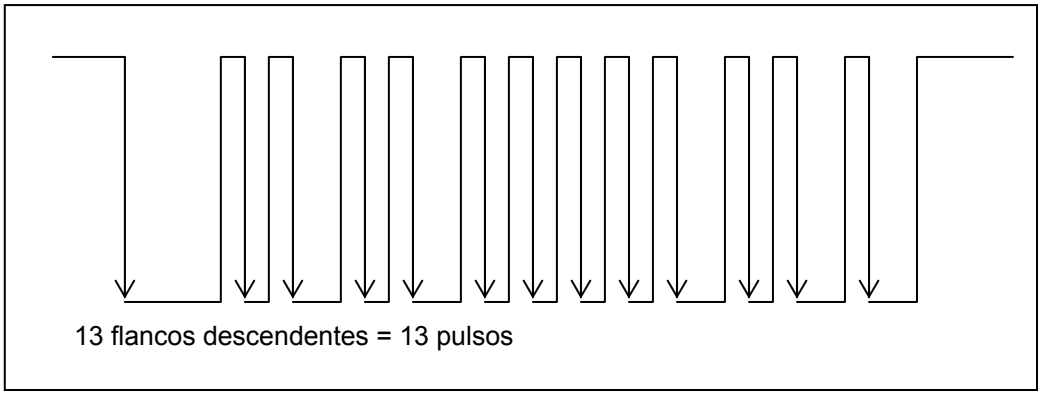


Figura 5.13. Conteo de pulsos

Cada vez que se ingrese un flanco descendente en el ICP (PD6) se activará la *Interrupción1* la cual le indicará al ATmega32 que debe incrementar su variable *Contador* de pulsos. Al final, el usuario debe presionar el pulsador para activar la

Interrupción2 que le indica al ATmega32 que muestre el valor del *Contador* en la pantalla LCD y luego lo encere para un nuevo conteo. Este proceso se muestra en la Figura 5.14.

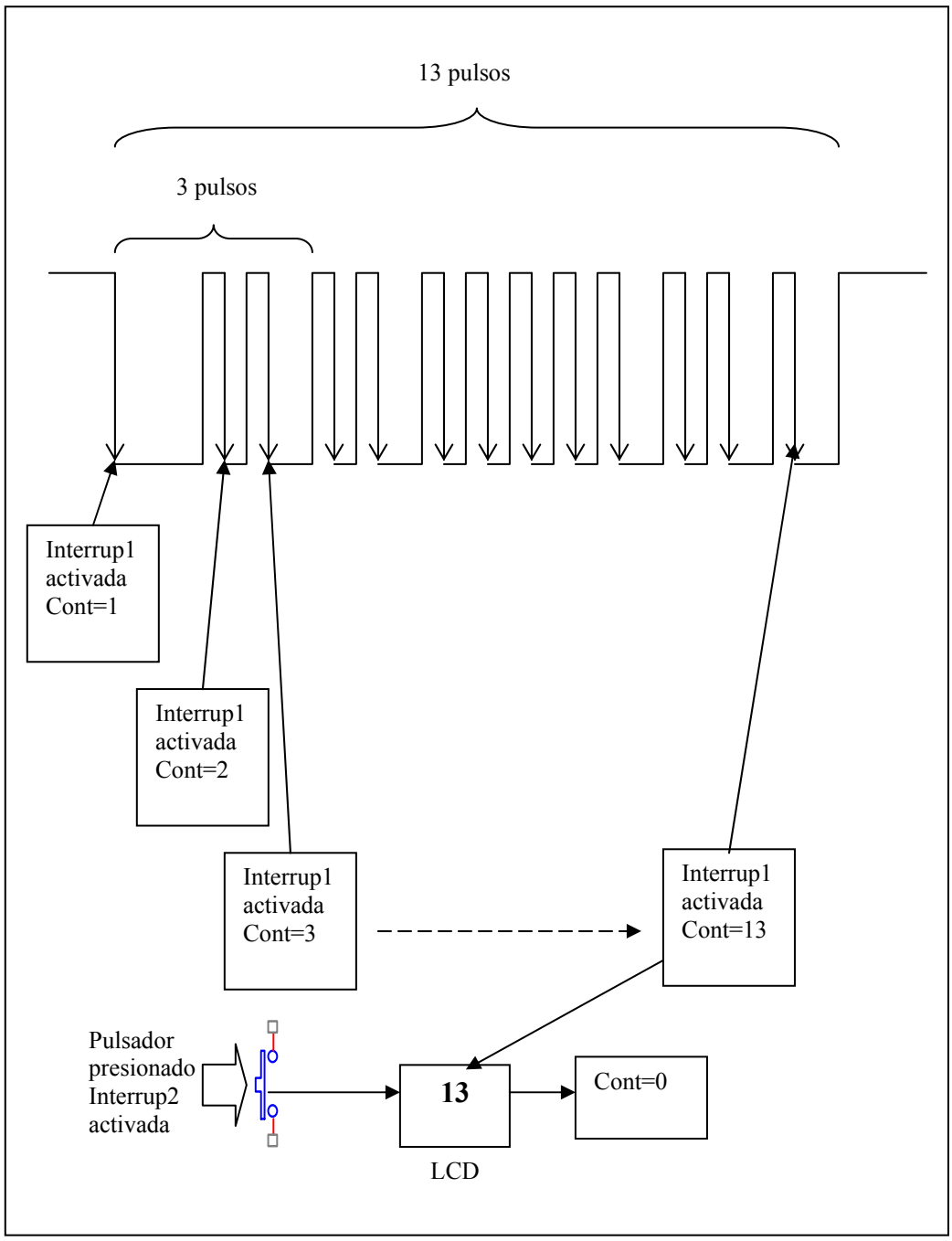


Figura 5.14. Funcionamiento del Contador de Pulsos

Código del Programa. Proyecto IRCOUNTER.

Al comenzar se debe configurar el ATmega32 para que utilice el reloj interno a 8 MHz. Esto se logra configurando los Bits de Configuración y Seguridad como ya se explicó en los capítulos respectivos: 2.4. Reloj del sistema y 4.3.6. Bits de Configuración y de Seguridad.

Luego utilizando CodeWizardAVR se crea el Proyecto *IRCOUNTER* dentro de su carpeta respectiva. Entre las opciones de la ventana de CodeWizardAVR, se debe establecer el tipo de microcontrolador como un ATmega32 a 8 MHz.

También se debe especificar que el Puerto D es de entrada con resistencia de Pull-up y el Puerto C es de salida con un valor inicial en bajo para todos los pines, la configuración de los demás puertos es indiferente. El Puerto D se va a emplear para recibir la señal desde el Módulo Detector IR (ICP – PD6) y el Puerto C se va a utilizar para manejar el LCD, por lo se debe escoger esta opción en la parte respectiva de CodeWizardAVR.

Además se van a utilizar dos interrupciones: INT0 y Timer1 Input Capture. INT0 se debe activar por flanco descendente (Falling Edge). Timer1 se debe configurar en Modo Normal a una frecuencia de 8 MHz y un *prescaler* de 1024 lo que da una frecuencia de conteo para el Timer 1 de 7,8125 KHz. La interrupción Input Capture y la opción Noise Cancel deben ser activadas, Se debe especificar que la interrupción Input Capture se active por flanco descendente.

El archivo *IRCounter.c* generado con CodeWizardAVR, además de inicializar el ATmega32 con las opciones especificadas, crea dos nuevas funciones ISR (Interrupt Service Routine) para atender las interrupciones interrupt[EXT_INT0] e interrupt[TIM1_CAPT].

Luego de generar el archivo .c se puede apreciar que el código necesario para habilitar las interrupciones y el manejo del LCD ya ha sido creado. Sin embargo

se debe ingresar el código que le indique al ATmega32 como debe funcionar. El código que se debe añadir se muestra en la Tabla 5.1.

Código añadido	Función
// Declare your global variables here unsigned char Counter	*Counter* es la variable global que guarda el número de pulsos. Al ser *Counte*r una variable tipo unsigned char puede ir desde 0 hasta 255

Ext_int0_isr(void)
{
unsigned char cent,dec,uni;
cent = Counter/100;
dec=(Counter-cent*100)/10;
uni=Counter-cent*100-dec*10;
lcd_gotoxy(0,0);
 	
lcd_putchar(cent+48);
lcd_putchar(dec+48);
lcd_putchar(uni+48);
Counter=0x00;
}
	Al activarse INT0 se debe presentar el valor de *Counter* en el LCD. Como este valor puede llegar hasta 255, el número presentado será de 3 dígitos.
	
	La función lcd_putchar() escribe símbolos en el LCD uno a la vez por lo que para escribir un número de 3 dígitos se lo debe hacer uno por uno. Para hacer esto se utiliza las variables locales *cent*, *dec* y *uni* que descomponen el valor de *Counter* en centenas, decenas y unidades.
	
	Además se debe considerar que la función lcd_putchar() escribe símbolos ASCII por lo que para escribir el valor de cada variable se le debe añadir 48 ya que por ejemplo el código ACII para el número 1 es 49 = 48 + 1.

interrupt [TIM1_CAPT] void Counter1_capt_isr(void) { Counter++;}	En este programa lo único que debe ocurrir cuando la interrupción TIMER1 Input Capture es activada, es que *Counter* se incremente.

Tabla. 5.1. Código de IRCounter.c. (Solo se incluye nuevas modificaciones a GreenLed.c)

Luego de las modificaciones respectivas el código del archivo *IRCounter.c* es el siguiente:

```
/*****************************************************
Este programa recibe la señal IR de un Control Remoto.
El sensor IR (de TV) está conectado al pin PD6(ICP).
El programa cuenta el número de pulsos (Falling edge) transmitidos al presionar
cualquier botón del Control Remoto.
Se muestra el resultado en un LCD al presionar el pulsador
conectado a PD2(INT0).

Cada símbolo del Control Remoto consta de 13 pulsos (Falling edge)
(STARTBIT + 12 BITS)

Según los resultados obtenidos los controles remotos están diseñados para
```

```
enviar múltiplos de 13 (mínimo 3) por lo que los resultados
suelen ser 039, 052, 078, 091 dependiendo de cuanto tiempo se presione
el botón
*****************************************************/

#include <mega32.h>
// the LCD is connected to PORTC outputs
// see the file lcd.h in the ..\inc directory
#asm
.equ __lcd_port=0x15 ;PORTC
#endasm

// include the LCD driver routines
 #include <lcd.h>
// Declare your global variables here
unsigned char Counter;

// External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{
unsigned char cent,dec,uni;

cent = Counter/100;
dec=(Counter-cent*100)/10;
uni=Counter-cent*100-dec*10;
lcd_gotoxy(0,0);

lcd_putchar(cent+48);
lcd_putchar(dec+48);
lcd_putchar(uni+48);
Counter=0x00;
        }

// Counter 1 input capture interrupt service routine
interrupt [TIM1_CAPT] void Counter1_capt_isr(void)
{
Counter++;
}

void main(void)
{

// Declare your local variables here
Counter=0x00;

// Input/Output Ports initialization
// Port A initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTA=0x00;
DDRA=0xFF;

// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x00;

// Port C initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
```

```
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTC=0x00;
DDRC=0xFF;

// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x00;

// Counter/Counter 0 initialization
// Clock source: System Clock
// Clock value: Counter 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

/// Counter/Counter 1 initialization
// Clock source: System Clock
// Clock value: 8000,000 kHz/1024
// Mode: CTC top=ICR1
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: On
// Input Capture on Falling Edge
// Counter 1 Overflow Interrupt: Off
// Input Capture Interrupt: On
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off

TCCR1A=0x00;
TCCR1B=0x85;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Counter/Counter 2 initialization
// Clock source: System Clock
// Clock value: Counter 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: On
// INT0 Mode: Falling Edge
// INT1: Off
// INT2: Off
GICR|=0x40;
MCUCR=0x02;
```

```
MCUCSR=0x00;
GIFR=0x40;

// Counter(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x20;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Counter/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// Global enable interrupts
#asm("sei")

lcd_init(16);
lcd_clear();

while (1)

    {
    // Place your code here
    };
}
```

Resultados obtenidos. Proyecto IRCOUNTER.

De este paso se puede aprender dos cosas muy importantes en cuanto al Control Remoto Universal Sony RM V8 y al Módulo Detector IR. El Control Remoto RM V8 envía pulsos mientras se presione cualquier botón. El número de pulsos es de mínimo 39, y dependiendo del tiempo que se presione el determinado botón pueden ser 52, 65, 78, etc. Siempre múltiplos de 13 ya que cada paquete en Codificación SIRC consta de 13 pulsos.

Además estos valores son obtenidos únicamente en condiciones ideales, cuando no hay una luz natural o artificial que incida sobre el Módulo Detector IR. Cuando esto ocurre los valores obtenidos varían en algunos pulsos más, lo cual indica que algo de ruido ambiental perturba al Módulo. El uso de un capacitor entre Vcc y GND en el Módulo Detector IR ayuda bastante a reducir este ruido de ambiente. Además el ruido empeora cuando se toca el Módulo.

5.4.5. Ancho de Pulso. Proyecto STARTBIT.

En este paso se puede mejorar el uso de la Interrupción por Captura del Timer1 del ATmega32, que trabaja con el pin ICP (PD6). Básicamente lo que se hace es medir el tiempo que transcurre entre un flanco descendente y un flanco ascendente, esto da la duración de los pulsos en bajo que en Codificación SIRC varían dependiendo del tipo de bit (los pulsos en alto se mantienen constantes) como se muestra en la Figura 5.15.

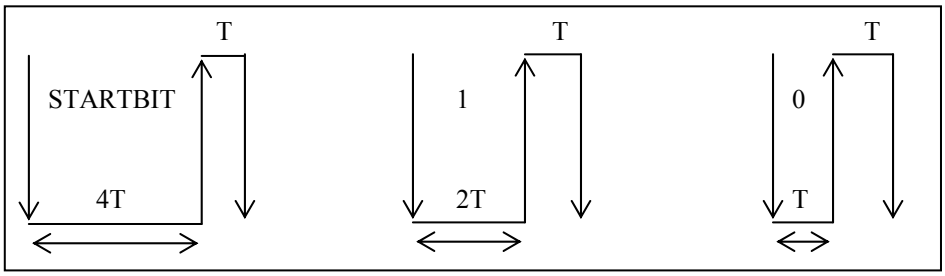


Figura 5.15. Ancho de distintos pulsos (Receptor)

Cada Timer del ATmega32 tiene un registro contador conocido como TCNTx que se incrementa cada cierto tiempo. Si se guarda en otro registro el valor de este contador cada vez que hay un flanco descendente o ascendente en el ICP, al restar estos valores se puede obtener el ancho del pulso en bajo.

Se puede comenzar por identificar y contar los STARTBITs que son más fáciles de diferenciar debido a que son más largos que los 1 y 0 lógicos.

Funcionamiento del Programa. Proyecto STARTBIT.

En el programa anterior, *GreenLed.c*, se determinó como se realiza el conteo de pulsos, ahora además se debe medir el ancho de los pulsos en bajo para lo cual se necesita que la *Interrupción1* se active ya sea con flancos ascendentes o descendentes. Sin embargo ya que resulta más simple contar y medir el ancho de pulso al mismo tiempo, esta vez se hará el conteo de los flancos ascendentes en vez de los descendentes.

Como todos los pulsos son recibidos de manera invertida en el Receptor, primero se debe capturar el flanco descendente y luego el ascendente; por lo que al inicio la *Interrupción1*, debe detectar un flanco descendente.

Si se detecta un flanco descendente, la variable *Bajada* debe guardar el valor del registro contador y la próxima vez la *Interrupción1* debe activarse con un flanco ascendente.

Si se detecta un flanco ascendente, la variable *Subida* debe guardar el valor del registro contador. Luego se resta el valor de *Subida* menos el de *Bajada* para obtener la *Longitud* del pulso. Si la *Longitud* es igual a 4T se considera al pulso un STARTBIT y se incrementa la variable *Contador* de STARTBITs. La próxima vez la *Interrupción1* debe activarse con un flanco ascendente.

Al final el usuario debe presionar el pulsador para activar la *Interrupción2* que muestre el *Contador* en la pantalla LCD y luego lo encere para un nuevo conteo. Este proceso se muestra en la Figura 5.16.

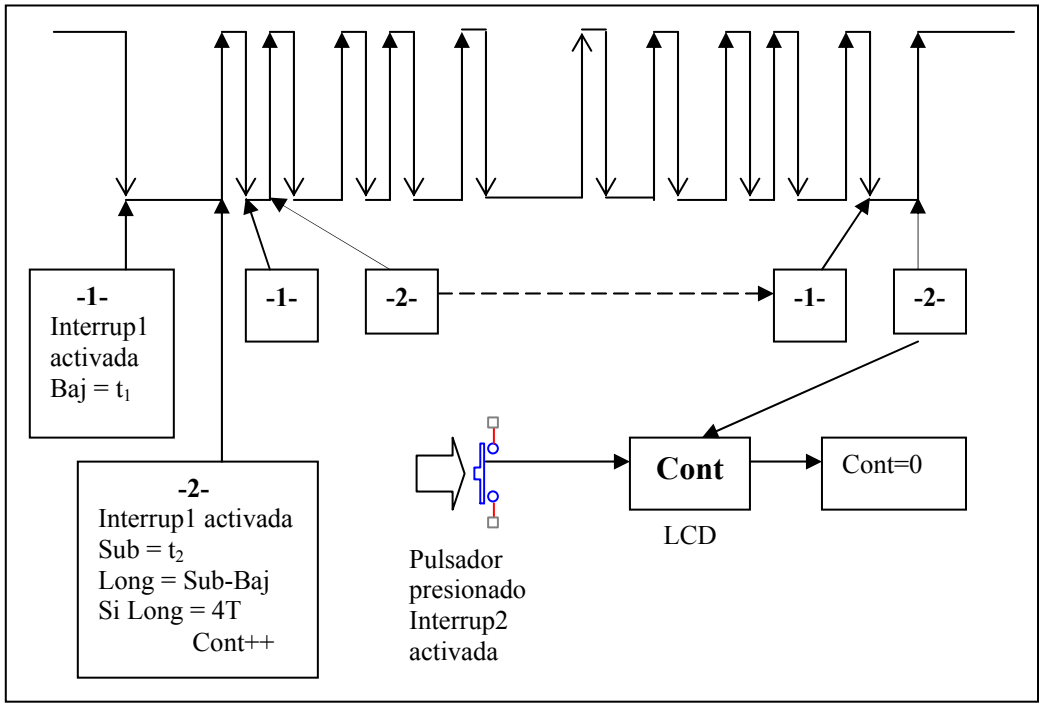


Figura 5.16. Funcionamiento del Contador de STARTBITs

Código del Programa. Proyecto STARTBIT.

Al momento de realizar el programa se deben hacer modificaciones al algoritmo para que se ajuste a la realidad del proceso. Es difícil que la *Longitud* de un STARTBIT sea exactamente de 4T por lo que hay que darle un margen de tolerancia.

Considerando un reloj de trabajo para el ATmega32 de 8 MHz y que el prescaler esté configurado a 256, esto da una frecuencia de conteo de 31,250 KHz (8MHz/256) que significa que el registro de conteo del Timer1, llamado TCNT1, incrementa su valor cada 32 µs (1/31,250 KHz) pudiendo llegar a contar hasta FFFF es decir 2,097 s (32 µs x 65535). Los valores que se quieren medir están en el orden de los cientos de µs por lo que esta escala es lo suficientemente precisa y tendrá un error de ±32 µs.

En mediciones realizadas anteriormente con programas basados en este algoritmo se llegó a obtener los valores mostrados en la Tabla 5.2. relacionados con la Figura 5.17. Dichos valores son en realidad promedios de algunas muestras que variaban en ±32n µs siendo n = 1,2 ó 3.

Tiempo		**Ancho (decimal)**	**Duración (µs)**
ALTO	T1	19	608
BAJO	T2	21	672
CERO	T1+T2	40	1280
UNO	T1+2T2	61	1952
STARTBIT BAJO	4T2	84	2688

Tabla. 5.2. Mediciones de anchos de pulso

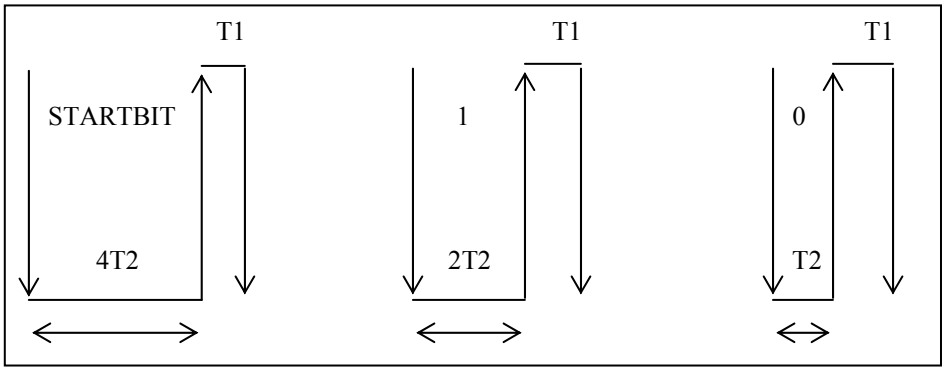


Figura 5.17. Mediciones de anchos de pulso

El ancho muestra la diferencia en decimal entre los valores del registro capturado en un flanco descendente y uno ascendente y así por ejemplo al multiplicar 19 por el periodo de 32 µs del Timer1 nos da el tiempo de duración del pulso en alto que es de 608 µs, valor que se encuentra dentro del rango utilizado en la Codificación SIRC (550 – 650 µs) como se indicó en la Sección 5.3.

Con los valores obtenidos se confirma que el pulso en bajo es de T2 para un cero, 2T2 para un uno y 4T2 para el STARTBIT. El pulso en alto T1 es ligeramente menor que T2. Cabe acotar que estos valores no son precisos debido el error propio del contador TCNT de ±32 µs y al ruido del ambiente por lo que los valores mostrados son el promedio de varios datos obtenidos.

Utilizando la información de la Tabla 5.2. se puede determinar que 4T es en promedio igual a 84 en decimal por lo que al momento de escribir el código del programa se puede considerar a un pulso un STARTBIT, si su tiempo en bajo está dentro del rango de 80 y 88.

La creación del proyecto *STARTBIT* y la generación del archivo *StartBit.c* son iguales a lo realizado en el proyecto *IRCOUNTER*. Sin embargo se deben hacer las siguientes modificaciones mostradas en la Tabla 5.3.

Código añadido	Función
// Declare your global variables here unsigned char Timer; bit flag; unsigned char Falling, Rising, Width;	*Timer* es la variable global que guarda el valor del registro de captura del Timer1, TCR1L. No se requiere guardar el valor de TCR1H ya que el ancho de un pulso no supera 0xFF = 255. *flag* es una variable tipo bit que al ser igual a 0 indica que se capturó un flanco ascendente y si es igual a 1 indica que se capturó un flanco descendente. *Falling, Rising* y *Width* son las variables que guardan los tiempos de Bajada, Subida y Ancho de pulso respectivamente.

interrupt [TIM1_CAPT] void
Counter1_capt_isr(void)
{
flag++;
Timer=ICR1L;
 	
if(flag==1)
{
TCCR1B=0xC4;
Falling=Timer;
};
 	
if(flag==0)
{
TCCR1B=0x84;
Rising=Timer;
Width=Rising-Falling;
if(Width>=80&&Width<=88)Counter++;
};
}
	Esta función ha sido modificada para que alterne la captura de flancos descendentes y ascendentes mediante el uso de la bandera *flag* y cambiando el registro TCCR1B.
	
	Debido a que *flag* es una variable tipo bit, cuando se ejecuta f*lag++*, su valor solo cambia entre 0 y 1 alternadamente.
	
	Cuando *flag* = 1 esto significa que se capturó un flanco descendente y por lo tanto se guarda en *Falling* el valor de *Timer*. Además se altera TCCR1B para que la próxima vez capture un flanco ascendente.
	
	Cuando *flag* = 0 esto significa que se capturó un flanco ascendente y por lo tanto se guarda en *Rising* el valor de *Timer*. Además se altera TCCR1B para que la próxima vez capture un flanco descendente. Luego se obtiene *Width* y si está dentro del rango 80 – 88 se incrementa *Counter*

Tabla 5.3. Código de StartBit.c. (Solo se incluye nuevas modificaciones a IRCounter.c)

Luego de las modificaciones respectivas, el código del archivo *StartBit.c* es el siguiente:

```
/*****************************************************
Este programa recibe la señal IR de un Control Remoto.
El sensor IR (de TV) está conectado al pin PD6(ICP).
Se muestra el resultado en un LCD al presionar el pulsador
conectado a PD2(INT0).
Este programa cuenta el número de StartBits recibidos.
*****************************************************/
#include <mega32.h>
```

```
// the LCD is connected to PORTC outputs
// see the file lcd.h in the ..\inc directory
#asm
.equ __lcd_port=0x15 ;PORTC
#endasm

// include the LCD driver routines
 #include <lcd.h>
// Declare your global variables here
unsigned char Counter;
unsigned char Timer;
bit flag;
unsigned char Falling, Rising, Width;

// External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{
unsigned char dec,uni;
dec=Counter/10;
uni=Counter1-dec*10;
lcd_gotoxy(0,0);
lcd_putchar(dec+48);
lcd_putchar(uni+48);
Counter=0x00;
        }

// Counter 1 input capture interrupt service routine
interrupt [TIM1_CAPT] void Counter1_capt_isr(void)
{
flag++;
Timer=ICR1L;

if(flag==1)
{
       TCCR1B=0xC4;
       Falling=Timer;
};

if(flag==0)
{
       TCCR1B=0x84;
       Rising=Timer;
       Width=Rising-Falling;
       if(Width>=80&&Width<=88)Counter++;
};
}

void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization

// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTA=0x00;
DDRA=0xFF;

// Port B initialization
```

```
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x00;

// Port C initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTC=0x00;
DDRC=0xFF;

// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x00;

// Counter/Counter 0 initialization
// Clock source: System Clock
// Clock value: Counter 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

/// Counter/Counter 1 initialization
// Clock source: System Clock
// Clock value: 8000,000 kHz/256
// Mode: CTC top=ICR1
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: On
// Input Capture on Falling Edge
// Counter 1 Overflow Interrupt: Off
// Input Capture Interrupt: On
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x84;     //0x84-->256
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Counter/Counter 2 initialization
// Clock source: System Clock
// Clock value: Counter 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
```

```
// INT0: On
// INT0 Mode: Falling Edge
// INT1: Off
// INT2: Off
GICR|=0x40;
MCUCR=0x02;
MCUCSR=0x00;
GIFR=0x40;
// Counter(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x20;
// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Counter/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;
// Global enable interrupts
#asm("sei")
lcd_init(16);
lcd_clear();

while (1)

    {
    // Place your code here
    };
}
```

Resultados obtenidos. Proyecto STARTBIT.

El proyecto STARTBIT es el compendio de varias pruebas basadas en el mismo algoritmo, describir cada una de ellas sería extenso. En vez de contar STARTBITs se puede medirlos ordenando al ATmega32 que muestre el ancho de pulso en bajo medido más largo. También se podrían medir o contar pulsos en alto o en bajo. Con los datos obtenidos se formó la Tabla 5.2. que será de gran utilidad al momento de decodificar la señal SIRC enviada por el Control Remoto Universal Sony RM V8.

Otro dato importante obtenido es que los pulsos en bajo o en alto en Codificación SIRC no son menores a 19 (608 µs) por lo que pulsos menores a 17 (544 µs) se pueden considerar como ruido ambiental y por lo tanto se podría implementar un algoritmo que discrimine estos pulsos.

5.4.6. Identificación de Códigos. Proyecto IRDECO.

Teniendo la capacidad de identificar STARTBITS, unos y ceros es posible recibir la información codificada enviada por el Control Remoto RM V8 en modo TV (la Dirección será constante). Lo que se debe hacer es una rutina que detecte un STARTBIT y guarde en memoria los 12 bits que lo siguen. Para simplificar el programa se puede tan solo guardar en memoria los 8 primeros bits para formar un número hexadecimal tipo 0xFF.

Funcionamiento del Programa. Proyecto IRDECO.

Como fue mencionado anteriormente en la Sección 5.4.5. Ancho de Pulso. Proyecto STARTBIT, la medida del ancho de los pulsos transmitidos por el Control Remoto Universal Sony RM V8 no es precisa debido a variaciones de ±32n µs siendo n = 1,2 ó 3. Por esta razón es recomendable definir rangos de tolerancia para los distintos anchos de pulso de la codificación SIRC. Como el peor caso es cuando n = 3, los rangos mostrados en forma decimal en la Tabla 5.4. tienen una tolerancia de ±3.

Tipo de pulso	Límite inferior	Límite superior
1 Pulso en bajo	39*	45
0 Pulso en bajo	18	24
STARTBIT Pulso en bajo	81	87
* Estos valores se deben multiplicar por 32 µs para obtener su duración en seg.		

Tabla 5.4. Rangos para distintos anchos de pulso

El reconocimiento del comando transmitido se realiza en el flanco ascendente del pulso en bajo durante la *Interrupción1*:

El primer paso es medir el ancho de pulsos de bajo mediante la operación *Longitud* = *Subida* – *Bajada* ya utilizada en proyectos anteriores.

Para comenzar el proceso de decodificación de comando se debe esperar a recibir un STARTBIT que será reconocido por su ancho de pulso en bajo que está dentro del rango 78-86.

Luego de reconocerlo hay que esperar al próximo pulso en bajo para comenzar a medir los siguientes 8 bits. Debido a que de los 12 bits transmitidos únicamente los 7 primeros cambian (los 5 restantes son constantes ya que corresponden a la dirección del equipo, en este caso TV), solo es necesario guardar los 8 primeros bits para descodificar el comando transmitido (además de guardar los 7 bits de código es recomendable guardar el primer bit de dirección para poder formar un número hexadecimal tipo 0xFF).

Cuando el siguiente pulso en bajo haya ocurrido, si está en el rango 38-46 es un 1 lógico y si está en el rango 18-26 es un 0 lógico.

Para guardar el comando transmitido se necesitan 8 variables *b0, b1, b2,…, b6* y *b7*, cada una de las cuales guardará un bit que puede ser 0 ó 1. La posición del bit actual de código se indica mediante una variable llamada *nbit*.

Cuando se haya llegado al bit 8 se debe concluir el proceso de captura de bits hasta que un nuevo STARTBIT aparezca.

Al final el usuario debe presionar el pulsador para activar la *Interrupción2* que muestre los 8 bits transmitidos en la pantalla LCD y luego encere todas las variables utilizadas para poder recibir y mostrar un nuevo comando. El proceso completo se muestra en la Figura 5.18.

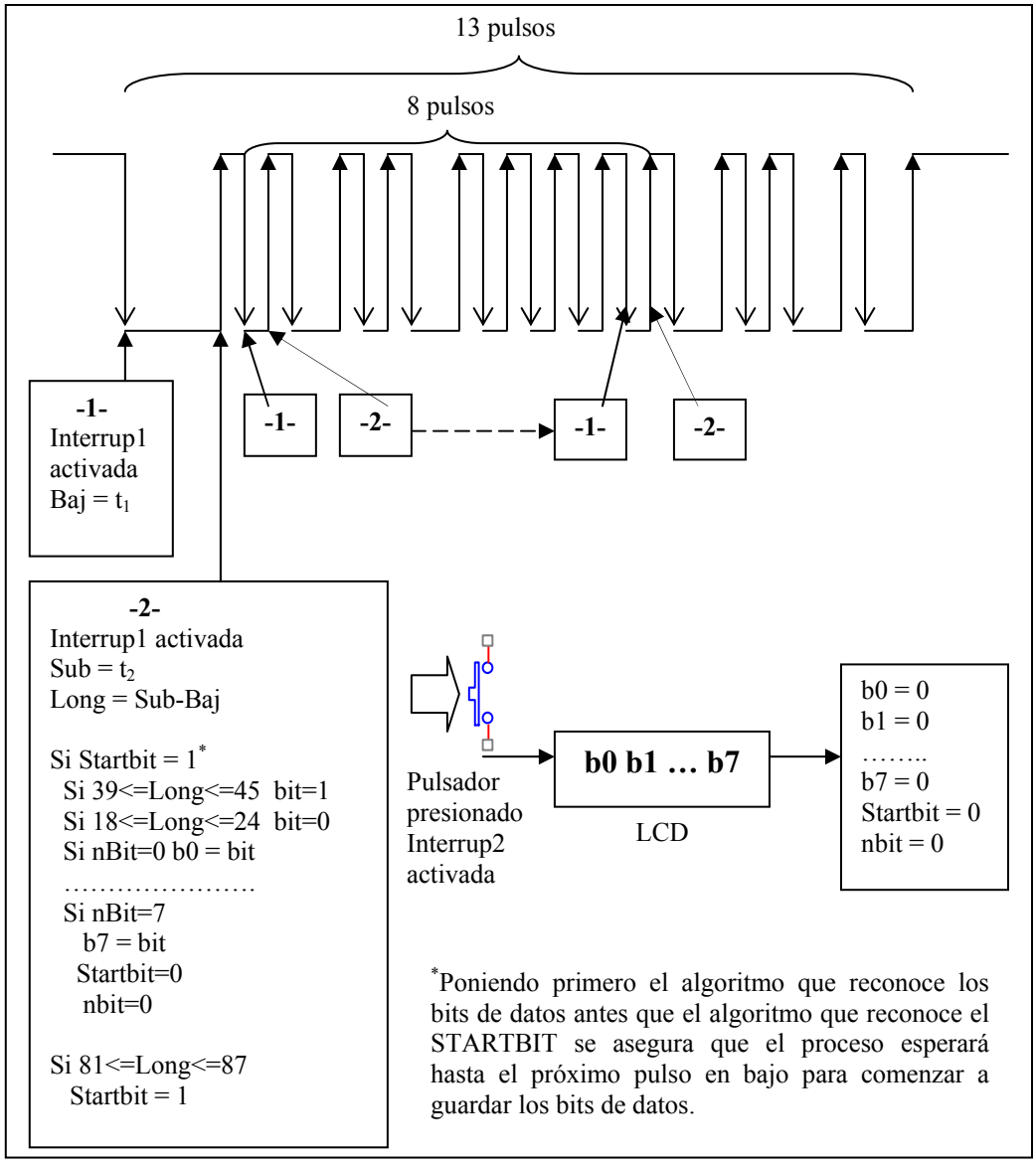


Figura 5.18. Funcionamiento del Identificador de Códigos

Código del Programa. Proyecto IRDECO.

La creación del proyecto *IRDECO* y la generación del archivo *IRDeco.c* son iguales a lo realizado en el proyecto *STARTBIT*. Sin embargo se deben hacer las siguientes modificaciones mostradas en la Tabla 5.5.

Código añadido	Función
// Declare your global variables here unsigned char nBit; bit StartBit, BitRx; bit b0,b1,b2,b3,b4,b5,b6,b7;	nBit indica la posición del bit de datos. b0 a b7 guardan los bits de datos StartBit cuando es 1 indica que un STARTBIT ha sido reconocido. BitRx guarda el bit recibido.

interrupt [TIM1_CAPT] void
Counter1_capt_isr(void){
Timer=ICR1L;
switch (flag) {
case 0:
flag=1;
TCCR1B=0xC4;
Falling=Timer;
break;
case 1:
flag=0;
TCCR1B=0x84;
Rising=Timer;
Width=Rising-Falling;
if(StartBit==1){
if(Width>=38&&Width<=46) BitRx=1;
if(Width>=18&&Width<=26) BitRx=0;
switch (nBit) {
case 0:b0=BitRx;nBit++;break;
case 1:b1=BitRx;nBit++;break;
case 2:b2=BitRx;nBit++;break;
case 3:b3=BitRx;nBit++;break;
case 4:b4=BitRx;nBit++;break;
case 5:b5=BitRx;nBit++;break;
case 6:b6=BitRx;nBit++;break;
case 7:b7=BitRx;nBit=0;StartBit=0;
break;
default:StartBit=0;nBit=0;break;
} }
if(Width>=78&&Width<=86) StartBit=1;
break;}}
	Utilizando las estructuras *if* y *switch* se puede implementar las operaciones explicadas en el algoritmo para guardar los 8 bits del comando transmitido

interrupt [EXT_INT0] void ext_int0_isr(void){
lcd_gotoxy(0,0);
lcd_putchar(b0+48);
lcd_putchar(b1+48);
lcd_putchar(b2+48);
lcd_putchar(b3+48);
lcd_putchar(b4+48);
lcd_putchar(b5+48);
lcd_putchar(b6+48);
lcd_putchar(b7+48);
StartBit=0;
b0=0;b1=0;b2=0;b3=0;b4=0;b5=0;b6=0;b7=0;
nBit=0;}
	Para mostrar el comando transmitido en el LCD simplemente se utiliza la instrucción lcd_putchar() 8 veces.
	
	Luego se encera las variables utilizadas.

Tabla. 5.5. Código de IRDeco.c. (Solo se incluye nuevas modificaciones a StartBit.c)

Luego de las modificaciones respectivas el código del archivo *IRDeco.c* es el siguiente:

```
/*****************************************************
Este programa recibe la señal IR de un Control Remoto.
El sensor IR (de TV) está conectado al pin PD6(ICP).

Se muestra el resultado en un LCD al presionar el pulsador
conectado a PD2(INT0).

Muestra el código enviado (los primeros 8 bits después del STARTBIT)
*****************************************************/
#include <mega32.h>
// the LCD is connected to PORTC outputs
// see the file lcd.h in the ..\inc directory
#asm
.equ __lcd_port=0x15 ;PORTC
#endasm
// include the LCD driver routines
 #include <lcd.h>
// Declare your global variables here
unsigned char Timer,nBit;
bit flag, StartBit, BitRx;
bit b0,b1,b2,b3,b4,b5,b6,b7;
unsigned char Falling, Rising, Width;
// External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{
lcd_gotoxy(0,0);
lcd_putchar(b0+48);
lcd_putchar(b1+48);
lcd_putchar(b2+48);
lcd_putchar(b3+48);
lcd_putchar(b4+48);
lcd_putchar(b5+48);
lcd_putchar(b6+48);
lcd_putchar(b7+48);
StartBit=0;
b0=0;
b1=0;
b2=0;
b3=0;
b4=0;
b5=0;
b6=0;
b7=0;
nBit=0;
        }

// Counter 1 input capture interrupt service routine
interrupt [TIM1_CAPT] void Counter1_capt_isr(void)
{
Timer=ICR1L;
switch (flag) {
   case 0:
      flag=1;
      TCCR1B=0xC4;
      Falling=Timer;
```

```
   break;
   case 1:
      flag=0;
      TCCR1B=0x84;
      Rising=Timer;
      Width=Rising-Falling;

      if(StartBit==1){
 if(Width>=38&&Width<=46) BitRx=1;
 if(Width>=18&&Width<=26) BitRx=0;
            switch (nBit) {
                  case 0:b0=BitRx;nBit++;break;
                  case 1:b1=BitRx;nBit++;break;
                  case 2:b2=BitRx;nBit++;break;
                  case 3:b3=BitRx;nBit++;break;
                  case 4:b4=BitRx;nBit++;break;
                  case 5:b5=BitRx;nBit++;break;
                  case 6:b6=BitRx;nBit++;break;
                  case 7:b7=BitRx;nBit=0;StartBit=0;break;
                  default:StartBit=0;nBit=0;break;
             };
       };

      if(Width>=78&&Width<=86) StartBit=1;

   break;
};

}
void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTA=0x00;
DDRA=0xFF;
// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x00;
// Port C initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTC=0x00;
DDRC=0xFF;
// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x00;

// Counter/Counter 0 initialization
// Clock source: System Clock
// Clock value: Counter 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
```

```
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;
/// Counter/Counter 1 initialization
// Clock source: System Clock
// Clock value: 8000,000 kHz/256
// Mode: CTC top=ICR1
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: On
// Input Capture on Falling Edge
// Counter 1 Overflow Interrupt: Off
// Input Capture Interrupt: On
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x84;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
// Counter/Counter 2 initialization
// Clock source: System Clock
// Clock value: Counter 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;
// External Interrupt(s) initialization
// INT0: On
// INT0 Mode: Falling Edge
// INT1: Off
// INT2: Off
GICR|=0x40;
MCUCR=0x02;
MCUCSR=0x00;
GIFR=0x40;
// Counter(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x20;
// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Counter/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;
// Global enable interrupts
#asm("sei")
lcd_init(16);
lcd_clear();

while (1)
      {
      // Place your code here
      };
}
```

Resultados Obtenidos. Proyecto IRDECO.

A través de este programa se logró descifrar los comandos transmitidos por el Control Remoto RM V8 en Modo TV. La Tabla 5.6. muestra los códigos obtenidos.

CONTRO REMOTO SONY RM V8 MODO TV																	
HEX	**Comando**	**b0**	**b1**	**b2**	**b3**	**b4**	**b5**	**b6**	**b7**	**b8**	**b9**	**b10**	**b11**	**Relleno**	**Pulsos**	**Total**	**T (seg)**
0x80	1	0	0	0	0	0	0	0	1	0	0	0	0	140	585	725	0,0232
0x81	2	1	0	0	0	0	0	0	1	0	0	0	0	120	606	726	0,023232
0x82	3	0	1	0	0	0	0	0	1	0	0	0	0	120	606	726	0,023232
0x83	4	1	1	0	0	0	0	0	1	0	0	0	0	100	627	727	0,023264
0x84	5	0	0	1	0	0	0	0	1	0	0	0	0	120	606	726	0,023232
0x85	6	1	0	1	0	0	0	0	1	0	0	0	0	100	627	727	0,023264
0x86	7	0	1	1	0	0	0	0	1	0	0	0	0	100	627	727	0,023264
0x87	8	1	1	1	0	0	0	0	1	0	0	0	0	80	648	728	0,023296
0x88	9	0	0	0	1	0	0	0	1	0	0	0	0	120	606	726	0,023232
0x89	0	1	0	0	1	0	0	0	1	0	0	0	0	100	627	727	0,023264
0x8b	ENT	1	1	0	1	0	0	0	1	0	0	0	0	80	648	728	0,023296
0x90	CH+	0	0	0	0	1	0	0	1	0	0	0	0	100	606	706	0,022592
0x91	CH-	1	0	0	0	1	0	0	1	0	0	0	0	80	627	707	0,022624
0x92	VOL+	0	1	0	0	1	0	0	1	0	0	0	0	120	627	747	0,023904
0x93	VOL-	1	1	0	0	1	0	0	1	0	0	0	0	80	648	728	0,023296
0x94	MUTE	0	0	1	0	1	0	0	1	0	0	0	0	40	627	667	0,021344
0x95	PWR	1	0	1	0	1	0	0	1	0	0	0	0	80	648	728	0,023296
0x98	=	0	0	0	1	1	0	0	1	1	0	1	0	60	669	729	0,023328
0x99	II	1	0	0	1	1	0	0	1	1	0	1	0	40	690	730	0,02336
0x9a	>	0	1	0	1	1	0	0	1	1	0	1	0	40	690	730	0,02336
0x9b	<<	1	1	0	1	1	0	0	1	1	0	1	0	20	711	731	0,023392
0x9c	>>	0	0	1	1	1	0	0	1	1	0	1	0	40	690	730	0,02336
0x9d	REC+>	1	0	1	1	1	0	0	1	1	0	1	0	20	711	731	0,023392
0xa5	FUNC	1	0	1	0	0	1	0	1	0	0	0	0	80	648	728	0,023296
0xb6	SLEEP	0	1	1	0	1	1	0	1	0	0	0	0	60	669	729	0,023328
0xba	DISP	0	1	0	1	1	1	0	1	0	0	0	0	60	669	729	0,023328
0xbb	RECALL	1	1	0	1	1	1	0	1	0	0	0	0	40	690	730	0,02336

Tabla. 5.6. Comandos del Control Remoto Sony RM V8 Modo TV

De los 12 bits los últimos 5 varían dependiendo del equipo al que van dirigidos (TV o VCR). Los bits de interés son los 8 primeros que pueden ser guardados en una variable que llegue hasta FF (unsigned char). El primer bit recibido b0 es el

LSB y el último bit b7 es el MSB por lo que el código correcto en hexadecimal es el de la primera columna. Como se puede apreciar en dicha columna cada tipo de comando está agrupado según su naturaleza, los números van desde 0x80 hasta 0x89, canales y volumen desde 0x90 hasta 0x93 y así.

Al calcular el tiempo que duraba cada comando (STARTBIT + 12 BITS) se encontró que son distintos, sin embargo si se utiliza un programa auxiliar para medir la duración del pulso en bajo (en el Receptor) entre códigos se encuentra que éste varía dependiendo del comando y sirve de relleno para hacer que todos los comandos duren el mismo tiempo que es de de 23,2 ms. Según mediciones realizadas con dicho programa auxiliar, el relleno lo que hace es añadir 20 (640 µs) por cada cero que hay en el comando hasta un máximo de 140 (4,48 ms).

5.4.7. Filtro por Software.

Partiendo del resultado del proyecto anterior el objetivo actual es añadir al programa un filtro que desprecie pulsos menores a 17 (544 µs) esto daría al sistema mayor inmunidad al ruido ambiental.

Una señal SIRC sin ruido se vería como el tren de pulsos de la Figura 5.19.

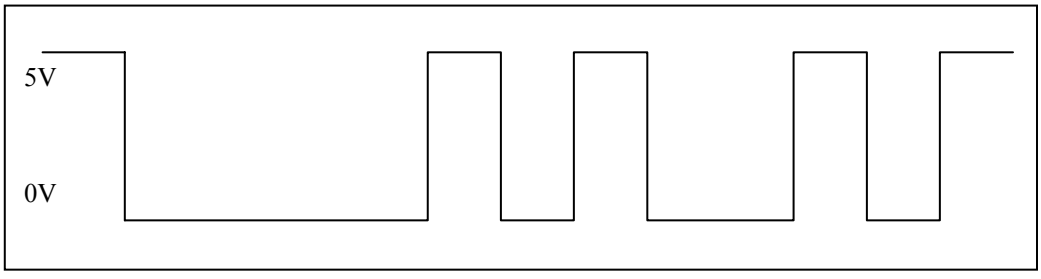


Figura 5.19. Tren de pulsos sin ruido

Una señal de ruido se caracteriza por ser un tren de pulsos muy cortos y aleatorios. Cabe recordar que a la entrada del ATmega32 llegan pulsos invertidos por lo que su forma sería como la que se muestra en la Figura 5.20.

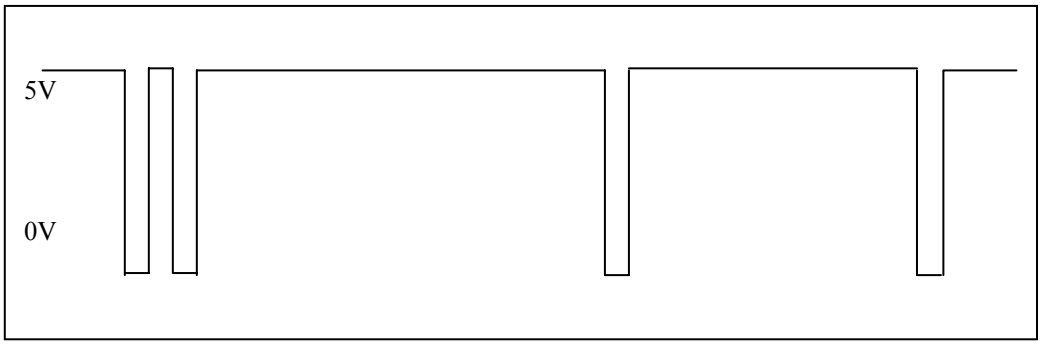


Figura 5.20. Forma del ruido

Al sumarse las señales, el ruido no afectará al ATmega32 mientras haya un pulso en bajo ya que la señal del Control Remoto solapa al ruido. Sin embargo cuando hay un pulso en alto esto significa que el Control Remoto no está emitiendo ninguna señal por lo que en este caso el ruido si ingresa al ATmega32 Una señal SIRC con ruido se vería como el tren de pulsos de la Figura 5.21.

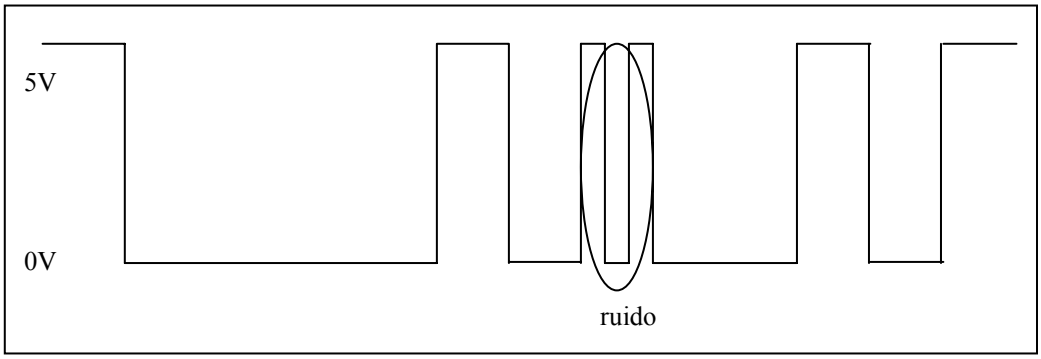


Figura 5.21. Tren de pulsos con ruido

La única modificación que se tiene que hacer en el archivo IRDeco.c es en la función interrupt [TIM1_CAPT]. Para facilitar el conteo se encera los registros TCNT1L y TCNT1H cada vez que se activa la interrupción Input Capture. Además se cambió la forma en que se alterna el valor de *flag* para evitar posibles desbordes. La nueva versión de la función interrupt [TIM1_CAPT] se muestra en la Tabla 5.7.

Código añadido	Función

interrupt [TIM1_CAPT] void
Counter1_capt_isr(void)
{
Timer=ICR1L;
switch (flag) {
case 0:
flag=1;
TCCR1B=0xC4;
if(Timer>=17){
TCNT1H=0x00;
TCNT1L=0x00;
};
break;
case 1:
flag=0;
TCCR1B=FALL;
if(Timer>=17){
Width=Timer;
TCNT1H=0x00;
TCNT1L=0x00;
};
break;
};
}
	En esta modificación, la variable *flag* se cambia dentro de la estructura *switch* que alterna el valor de dicha variable entre 0 y 1.
	
	Mediante la instrucción *if* se asegura que el *Timer* tenga un valor mayor o igual que 17 (544 µs) tanto en el flanco ascendente como en el descendente.
	
	Se encera los registros contadores del Timer1 TCNT1L y TCNT1H cada vez que se activa la interrupción Input Capture para tener siempre directamente los anchos de pulso.

Tabla 5.7. Código de filtro. (Solo se incluye nuevas modificaciones a IRDeco.c)

5.4.8. Representación numérica. Proyecto IRDNUM.

Una vez obtenidos los comandos, se debe presentar los símbolos que representan en el LCD. Al presionar el botón 1 del Control Remoto, no debe mostrarse 00000001 sino 1. Se puede comenzar de manera experimental con los comandos que representan los números del 0 al 9.

Funcionamiento del Programa. Proyecto IRDNUM.

La decodificación de los botones de números es sencilla ya que se puede utilizar un artificio matemático para calcular el número. De los 8 bits de datos, si se suma b3*8 + b2*4 + b1*2 + b0 + 1 se puede obtener el botón presionado desde

el 1 hasta el 9. Para obtener el 0 se debe considerar que es el único botón numérico que contiene un 1 en b0 y en b3 al mismo tiempo.

Además para dar un funcionamiento más efectivo al programa se debe presentar el número en el LCD cuando se presione el botón del Control Remoto correspondiente y no cuando se active la *Interrupción2*. El algoritmo final quedaría como el que se presenta en la Figura 5.22.

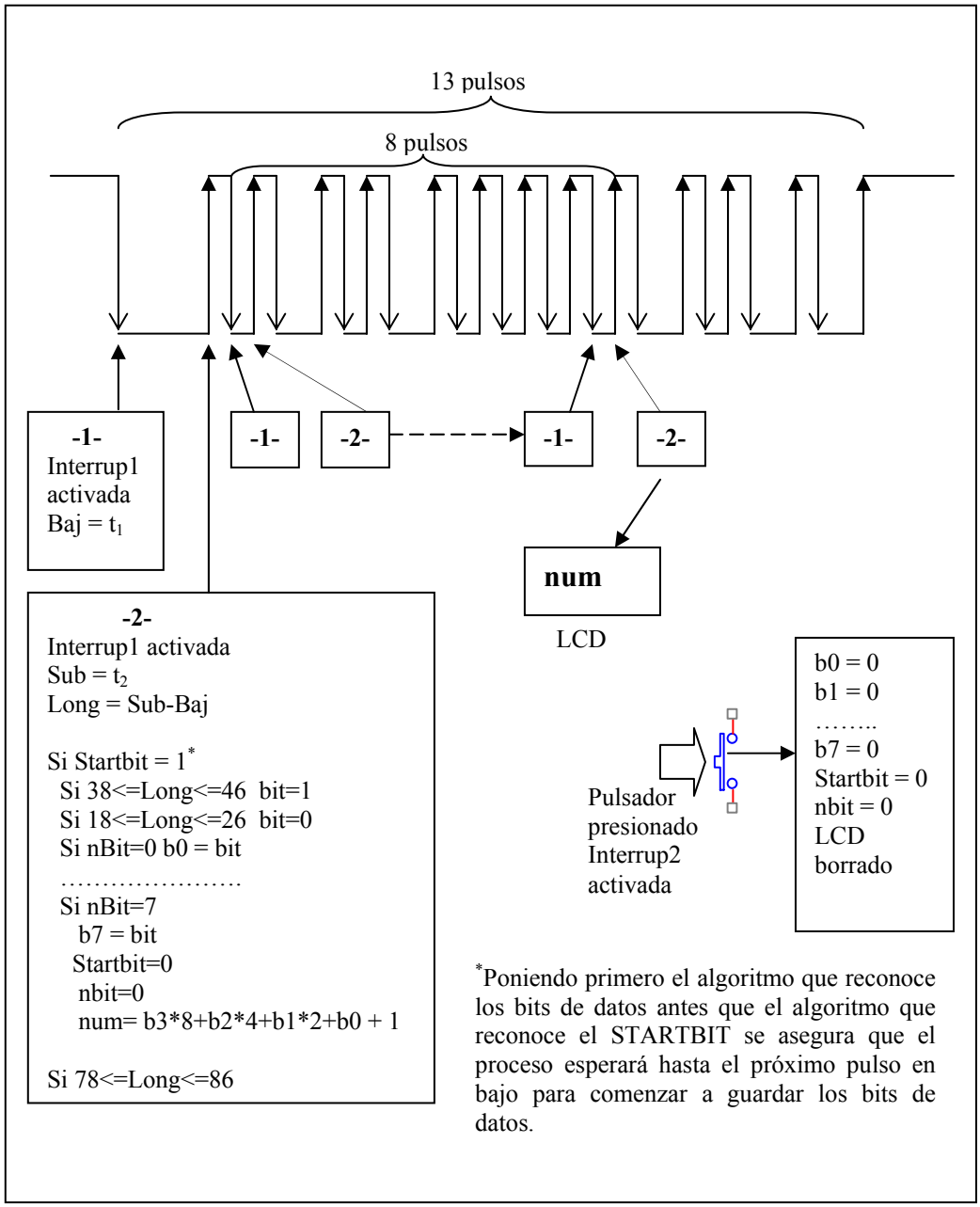


Figura 5.22. Funcionamiento del Decodificador Numérico

Código del Programa. Proyecto IRDNUM.

La única modificación que se debe hacer al proyecto anterior para crear el proyecto *IRDNUM* y el archivo *IRDNum.c* es que en vez de presentar el código del número se debe mostrar el número en si al momento en que se ha obtenido el bit de datos b7 y no cuando se presiona el pulsador de INT0 para darle un funcionamiento más real. El pulsador de INT0 solo se utilizaría para encerar las variables y borrar el LCD. Para lograr esto se debe borrar los lcd_putchar() dentro de interrupt [EXT_INT0], crear la variable global *unsigned char Num* y cambiar el *case 7* del *switch (nBit)* dentro de interrupt [TIM1_CAPT] de la siguiente manera:

```
 case 7:b7=BitRx;nBit=0;StartBit=0;
if(b0==1&&b3==1)Num=0;
else Num=b3*8+b2*4+b1*2+b0+1;
lcd_putchar(Num+48);
break;
```

Resultados parciales obtenidos. Proyecto IRDNUM.

En este proyecto se puede apreciar que al presionar un botón del Control Remoto se muestra en el LCD el número presionado repetido varias veces, ya que el Control Remoto sigue enviando paquetes mientras se tenga presionado el botón.

Además con cierta frecuencia el primer número suele ser errado y muestra un símbolo incorrecto, lo cual no pasa a partir del segundo número. Estos dos puntos sirven de base para el siguiente paso cuyo objetivo es mostrar un solo número al presionar cualquier botón entre 0 y 9 sin importar el tiempo que se lo haga. También hay que desechar el primer paquete y presentar el segundo paquete capturado que experimentalmente ha demostrado ser más confiable.

Modificaciones al programa. Proyecto IRDNUM.

Para lograr que solo se muestre un número al presionar cualquier botón entre 0 y 9 se debe implementar una rutina que solo muestre el segundo paquete recibido

y luego se deshabilite. Posteriormente debe medir constantemente la diferencia entre el tiempo actual y el último flanco ascendente. Si esta diferencia es muy grande, es decir, mayor que un paquete, esto significa que ya se dejó de presionar un botón y se debe habilitar la presentación de datos en el LCD nuevamente para mostrar el siguiente número cuando se presione otra vez cualquier botón.

Además se va a introducir un nuevo concepto, *definición de constantes*. Para no tener que lidiar con variables numéricas cada vez que se quiera cambiar un parámetro se puede utilizar constantes que representen un número para todo el programa, así en vez de escribir 0x84 en el registro TCCR1B para indicar la captura de un flanco descendente se podría tan solo escribir *FALL* siempre y cuando haya sido definido anteriormente que FALL = 0x84. Luego si en el futuro se desea cambiar 0x84 por otro valor solo sería necesario cambiarlo en la línea donde se define el valor de FALL y no a lo largo de todo el código del programa. Esto se logra a través de la directiva de pre-procesamiento llamada *#define*. Las modificaciones del programa son mostradas en la Tabla 5.8.

Código añadido	**Función**
#define RISE 0xC4 #define FALL 0x84 #define TDOWN 21 // 21 tiempo en alto #define TUP 19 // 19 tiempo en bajo #define K 3 // 3 factor de tolerancia #define WAIT 4 // 4 factor de espera 0,032768 seg > 0,023296 seg (duración de un paquete)	Mediante la directiva de pre-procesamiento #define se asignó los determinados valores de RISE, FALL, TDOWN, TUP, K y WAIT. RISE y FALL trabajan con el registro TCCR1B. TDOWN y TUP representan los anchos de pulso en alto y en bajo. K modifica el rango de tolerancia de los distintos tipos de bits: STARTBIT, 0 y 1. WAIT trabaja con el registro TCNT1H i sirve para determinar si un botón del Control Remoto sigue siendo presionado. Todos estos valores están calibrados con un reloj interno de 8 MHz. Si se cambia el reloj interno o el prescaler se debe también cambiar estos valores.
// Declare your global variables here unsigned char aux,counter,Gotit;	*aux* guarda el valor del registro TCCNT1H. *counter* se utiliza para asegurarse que solo se capture y presente en el LCD el segundo paquete transmitido. *Gotit* indica que ya se capturó el segundo paquete y evita que se muestren más números en el LCD aún cuando se mantenga presionado el botón del Control Remoto.

interrupt [TIM1_CAPT] void
Counter1_capt_isr(void)
{
Timer=ICR1L;
TCNT1H=0x00;
TCNT1L=0x00;
switch (flag) {
case 0:
flag=1;
TCCR1B=RISE;
Falling=Timer;
break;
case 1:
flag=0;
TCCR1B=FALL;
if(Timer>=TDOWN-K){
Rising=Timer;
Width=Rising;
}
if(Width>=TDOWN-
K&&Width<=TDOWN+K)BitRx=0;
if(Width>=2*TDOWN-
K&&Width<=2*TDOWN+K)BitRx=1;
if(StartBit==1&&Gotit==0){
switch (nBit) {
case 0:b0=BitRx;nBit++;break;
case 1:b1=BitRx;nBit++;break;
case 2:b2=BitRx;nBit++;break;
case 3:b3=BitRx;nBit++;break;
case 4:b4=BitRx;nBit++;break;
case 5:b5=BitRx;nBit++;break;
case 6:b6=BitRx;nBit++;break;
case 7:b7=BitRx;nBit=0;StartBit=0;
if(counter==1){
if(b0==1&&b3==1)Num=0;
else Num=b3*8+b2*4+b1*2+b0+1;
lcd_putchar(Num+48);
Gotit=1;
PORTB=0x01;
}
counter++;
break;
default:nBit=0;StartBit=0;break;
}}
if(Width>=4*TDOWN-
2*K&&Width<=4*TDOWN+K){
StartBit=1;
}
break;
}}
	En las modificaciones realizadas a esta interrupción, se le indica al ATmega32 que muestre el número obtenido solo luego de haber recibido el segundo paquete, es decir, cuando counter = 1.
	
	Luego de mostrar el número en el LCD, se guarda en Gotit el valor de 1 para indicar que ya se cumplió con la captura y presentación del número transmitido, y se deshabilita esta parte del programa (aunque se siguen capturando pulsos para saber que el botón del Control Remoto continua siendo presionado) hasta que Gotit vuelva a ser igual a 0.

while (1){
aux=TCNT1L;
aux=TCNT1H;
if(aux>=WAIT){
PORTB=0x00;
Gotit=0;
counter=0;
};
};
	Para guardar el valor de un Registro de 16 bits como es el caso de TCNT1, dividido en High y Low, es necesario primero guardar la parte menos significativa y luego la más significativa. En este caso no importa que se pierda el valor de TCNT1L guardado en la variable aux al sobrescribir el valor de TCNT1H en la misma variable ya que solo éste último valor será utilizado posteriormente.

	Ya que TCNT1L y TCNT1H se enceran cada vez que se ejecuta la interrupción Input Capture su valor se mantendrá pequeño mientras se siga presionando cualquier botón del Control Remoto. Cuando se suelte dicho botón, el valor de TCNT1L aumentará hasta desbordarse y consecuentemente incrementar TCNT1H. En el momento en que TCNT1H supere el valor de 4, 0,032768 s si se usa un reloj interno de 8MHz, nos aseguramos de que ningún botón está siendo presionado ya que los paquetes llegan a durar hasta 0,023296 s en los cuales TCNT1 y TCNT1H están constantemente siendo encerados. Si WAIT es igual a 4, cuando aux >= WAIT se encera Gotit y counter para que el programa esté listo para recibir un nuevo número.

Tabla. 5.8. Código añadido a IRDNum.c. (Solo se incluye nuevas modificaciones a IRDeco.c)

Luego de las modificaciones respectivas el código del archivo *IRDNum.c* es el siguiente:

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

Este programa recibe la señal IR-TV de un Control Remoto SONY RM V8 universal.
El sensor IR (de TV) está conectado al pin PD6(ICP).

Se muestra el número presionado en un LCD (solo funciona con botones del 0 al 9)
No importa si se mantiene presionado mucho tiempo el boton, solo
muestra un numero hasta que se presione otro
un led en PORTB1 muestra que un boton está siendo presionado
los números se muestran uno a continuación de otro
Para borrar la pantalla del LCD se debe presionar el pulsador conectado al pin PD2 (INT0)

*****************************************************/

#include <mega32.h>
#asm
.equ __lcd_port=0x15 ;PORTC
#endasm
 #include <lcd.h>
#define RISE 0xC4
#define FALL 0x84
#define TDOWN 21 // 21 tiempo en alto
#define TUP 19 // 19 tiempo en bajo
#define K 3 // 3 factor de tolerancia
#define WAIT 4 // 4 factor de espera 0,032768 seg > 0,023296 seg (duración de un paquete)

unsigned char Timer,nBit;
unsigned char aux,counter,Gotit;
bit flag, StartBit,BitRx;
bit b0,b1,b2,b3,b4,b5,b6,b7;
unsigned char Falling, Rising;
```

```
unsigned char Width,Num;

interrupt [EXT_INT0] void ext_int0_isr(void)
{
lcd_clear();
TCNT1H=0x00;
TCNT1L=0x00;
PORTB=0x00;
StartBit=0;
Gotit=0;
counter=0;
b0=0;
b1=0;
b2=0;
b3=0;
b4=0;
b5=0;
b6=0;
b7=0;
 }

interrupt [TIM1_CAPT] void Counter1_capt_isr(void)
{

Timer=ICR1L;
TCNT1H=0x00;
TCNT1L=0x00;

    switch (flag) {
          case 0:
             flag=1;
             TCCR1B=RISE;
             Falling=Timer;
          break;
          case 1:

             flag=0;
             TCCR1B=FALL;
             if(Timer>=TDOWN-K){
                   Rising=Timer;
                   Width=Rising;
             };
                   if(Width>=TDOWN-K&&Width<=TDOWN+K)BitRx=0;
                   if(Width>=2*TDOWN-K&&Width<=2*TDOWN+K)BitRx=1;

                   if(StartBit==1&&Gotit==0){
                         switch (nBit) {
                               case 0:b0=BitRx;nBit++;break;
                               case 1:b1=BitRx;nBit++;break;
                               case 2:b2=BitRx;nBit++;break;
                               case 3:b3=BitRx;nBit++;break;
                               case 4:b4=BitRx;nBit++;break;
                               case 5:b5=BitRx;nBit++;break;
                               case 6:b6=BitRx;nBit++;break;
                               case 7:b7=BitRx;nBit=0;StartBit=0;
                                  if(counter==1){
                                  if(b0==1&&b3==1)Num=0;
                                  else Num=b3*8+b2*4+b1*2+b0+1;
                                  lcd_putchar(Num+48);
                                  Gotit=1;
```

```
                                PORTB=0x01;
                                };
                                counter++;
                                break;
                            default:nBit=0;StartBit=0;break;
                        };
                    };
                    if(Width>=4*TDOWN-2*K&&Width<=4*TDOWN+K){
                        StartBit=1;
                    };
            break;
    };

}

void main(void)
{

// Input/Output Ports initialization
// Port A initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTA=0x00;
DDRA=0xFF;

PORTB=0x00;
DDRB=0xFF;

PORTC=0x00;
DDRC=0xFF;
// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=P State6=P State5=P State4=P State3=P State2=P State1=P State0=P
PORTD=0xFF;
DDRD=0x00;

/// Counter/Counter 1 initialization
TCCR1A=0x00;
TCCR1B=FALL;     //0x84-->256
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// External Interrupt(s) initialization
GICR|=0x40;
MCUCR=0x02;
MCUCSR=0x00;
GIFR=0x40;

// Counter(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x20;

// Analog Comparator initialization
ACSR=0x80;
SFIOR=0x00;
```

```
// Global enable interrupts
#asm("sei")

lcd_init(8);
lcd_clear();
while (1){
    aux=TCNT1L;
    aux=TCNT1H;
    if(aux>=WAIT){
          PORTB=0x00;
          Gotit=0;
          counter=0;
         };
      };
}
```

Resultados finales obtenidos. Proyecto IRDNUM.

Luego de las modificaciones realizadas y añadiendo un filtro a la recepción de comandos, se logra mostrar en el LCD los números presionados en el Control Remoto, sin que se repitan mientras se presione el botón.

Sin embargo, todavía es necesario utilizar el pulsador conectado a INT0 para borrar la pantalla del LCD y encerar las variables en caso de algún error. El siguiente paso debe ser manejar completamente la presentación de números y letras en el LCD a través del Control Remoto.

5.4.9. Representación Alfanumérica ABC2 y Comandos de Ejecución. Proyecto IRDECO2.

Una vez resueltos todos los objetivos referidos a la decodificación de la señal recibida desde el Control Remoto el siguiente paso es poder presentar un mensaje en el LCD. Debido a que un Control Remoto no tiene suficientes botones para representar cada letra del alfabeto se puede utilizar el método empleado por los teléfonos celulares que tienen incorporados varias letras o símbolos en cada número del teclado desde el 0 hasta el 9. Ahora se debe relacionar cada comando numérico con varios símbolos como se muestra en la Tabla 5.9.

Comando numérico	Símbolos
1	. ! ?
2	ABC
3	DEF
4	GHI
5	JKL
6	MNO
7	PQRS
8	TUV
9	WXYZ
0	espacio

Tabla. 5.9. Tabla de símbolos

La rutina implementada es la misma que la utilizada para mostrar un solo número en el LCD solo que esta vez la diferencia entre el tiempo actual y el último flanco ascendente debe ser relacionada al tiempo que el usuario se demora en soltar y volver a presionar un botón, tiempo que es mucho mayor que la duración de un paquete. De esta manera si el tiempo es corto significa que está presionando el mismo botón por lo que el cursor debe quedarse en la misma posición cambiando de símbolo, por ejemplo, entre 2, A, B y C. Si el tiempo es largo significa que el usuario va a presionar otro botón por lo que la pantalla debe mostrar el último símbolo escogido y el cursor debe pasar a la siguiente posición.

Los Comandos de Ejecución se refieren a los comandos que no muestran un símbolo sino que ejecutan una acción que, por ejemplo, puede ser de borrado del último símbolo o de borrado de pantalla del LCD. Se puede relacionar un Comando de Ejecución con un comando del Control Remoto como se muestra en la Tabla 5.10.

Comando Control Remoto	Comando de ejecución
DISP	Borrar último símbolo
ENT	Borrar pantalla

Tabla. 5.10. Tabla de comandos

Funcionamiento del Programa. Proyecto IRDECO2.

Para lograr que el Control Remoto funcione como el teclado de un teléfono celular se debe considerar el tiempo que una persona se demora presionando el mismo botón o presionando distintos botones.

El tiempo que una persona necesita para presionar, soltar y volver a presionar el mismo botón es bastante corto, pero no menor a 0,023296 s que es el tiempo que dura un paquete en Codificación SIRC por lo que el factor de espera WAIT = 4 (igual a 0,032768 s) puede seguir siendo utilizado para indicar cuando la persona ha soltado el botón.

El tiempo que una persona necesita para presionar un botón, soltarlo y presionar otro botón es de aproximadamente 0,4096 s. Este valor puede cambiar dependiendo de que tan lejos estén los botones y de la destreza del usuario.

Utilizando estos dos tiempos se obtienen tres posibilidades:

- Que el tiempo transcurrido entre paquete y paquete sea menor que 0,032768 s lo cual significaría que el mismo botón se sigue presionando. En tal caso se debe mostrar el símbolo del botón y el cursor debe quedarse en el mismo sitio.

- Que el tiempo transcurrido entre paquete y paquete sea mayor que 0,032768 s y menor que 0,4096 s lo cual significaría que se presionó el mismo botón dos veces. En tal caso se debe rotar al siguiente símbolo, mostrarlo y el cursor debe quedarse en el mismo sitio.

- Que el tiempo transcurrido entre paquete y paquete sea mayor que 0,4096 s lo cual significaría que se presionó un botón y luego otro. En tal caso se debe mostrar el símbolo del primer botón, mover el cursor a la siguiente posición y mostrar el símbolo del otro botón.

El manejo de varios símbolos con el mismo botón del Control Remoto se puede lograr utilizando una variable que funcione como indicador (buffer circular) que rote su valor cada vez que se presione el mismo botón durante un lapso menor que 0,4096 s. Si se pasa de este tiempo significa que el usuario quiere pasar a la siguiente posición o que se ha presionado otro botón en el cual el puntero debe comenzar nuevamente desde su posición inicial.

Código del Programa. Proyecto IRDECO2.

En esta ocasión se va a crear el proyecto *IRDECO2* y el archivo *IRDeco2.c* basados en el proyecto *IRDNUM*. Debido a que el programa va adquiriendo complejidad es necesario separar los procesos. La interrupción *interrupt [TIM1_CAPT]* se va a encargar únicamente de capturar los pulsos del comando transmitido en Codificación SIRC. Cuando tenga los 8 bits de datos, llamará a su vez a la función *deco()* que se encargará de determinar que símbolo se debe mostrar en el LCD. La interrupción *interrupt [EXT_INT0]* activada por el pulsador solo se utilizará como un reset en caso de un mal funcionamiento que deberá encerar las variables utilizadas y borrar la pantalla del LCD.

Para manejar las tres posibilidades en cuanto al lapso de tiempo entre eventos al presionar los botones se tiene que usar dos instrucciones *if* que controlen continuamente el tiempo transcurrido dentro del *while (1)* en *main()*. Estos dos if se encargan de habilitar o deshabilitar las operaciones realizadas dentro de las demás funciones guardando 0 ó 1 en las variables de control respectivas.

Función deco().

La función *deco()* consta básicamente de dos *switches* anidados que se encargan de determinar que símbolo será presentado en el LCD o que acción será ejecutada.

Luego de haber recibido el segundo paquete de datos, que es más confiable, la función *deco()* opera los 8 bits de datos para obtener su código en hexadecimal y

guardarlo en la variable *Code*. Dependiendo del código se elige un subconjunto de opciones. Por ejemplo si el código es 0x81 significa que se presionó el botón 2 del Control Remoto por lo que el subconjunto de posibilidades es A, B, C y 2.

El *switch* interno maneja este subconjunto de posibilidades dependiendo del valor de la variable *point_s*. Si su valor es 0 se escoge el primer símbolo, que en el caso de presionar el botón 2 sería A. Cabe anotar que el valor de *point_s* debe rotar así que al llegar a su valor máximo debe regresar a 0.

Sólo cuando se haya presionado un botón numérico la variable *point_s* debe incrementarse. En caso de que se presione otro botón como DISP o ENT se debe ejecutar la orden respectiva y *point_s* no debe cambiar.

Sólo cuando la variable *Next* sea igual a 1 significa que se ha presionado otro botón o que ha pasado un tiempo mayor a 0,4096 s y por lo tanto el cursor debe pasar a la siguiente posición indicada por la variable *position*.

Luego de las modificaciones indicadas el código del archivo IRDeco2.c queda de la siguiente manera:

```
/*****************************************************
Este programa recibe la señal IR-TV de un Control Remoto SONY RM V8 universal.
El sensor IR (de TV) está conectado al pin PD6(ICP).

Se muestra el símbolo presionado en un LCD (funciona como el teclado de un celular)
No importa si se mantiene presionado mucho tiempo el boton, solo
muestra un numero hasta que se presione otro

un led en PORTB1 cuando está prendido indica que
el cursor se está quedando en la misma posición
cuando esta apagado muestra que el cursor ha pasado a
la siguiente posición.

*****************************************************/

#include <mega32.h>
#asm
.equ __lcd_port=0x15 ;PORTC
#endasm
 #include <lcd.h>
#define RISE 0xC4
#define FALL 0x84
#define TDOWN 21 // 21 tiempo en alto
```

```
#define TUP 19 // 19 tiempo en bajo
#define K 3 // 3 factor de tolerancia
#define WAIT 4 // 4 factor de espera 0,032768 seg > 0,023296 seg (duración de un paquete)

unsigned char Code,Simbolo,Width;
unsigned char Timer,nBit;
unsigned char aux,counter,position,point_s;
bit flag, StartBit,BitRx,Gotit,Next;
bit b0,b1,b2,b3,b4,b5,b6,b7;

void deco(){
    if(counter==1){
          Code=b0+2*b1+4*b2+8*b3+16*b4+32*b5+64*b6+128*b7;
          switch(Code){
            case 0x80:
               switch(point_s){
                 case 0:Simbolo='.';break;
                 case 1:Simbolo='?';break;
                 case 2:Simbolo='!';break;
                 case 3:Simbolo='1';point_s=-1;break;
               };
               break;
            case 0x81:
               switch(point_s){
                 case 0:Simbolo='A';break;
                 case 1:Simbolo='B';break;
                 case 2:Simbolo='C';break;
                 case 3:Simbolo='2';point_s=-1;break;
               };
               break;
            case 0x82:
               switch(point_s){
                 case 0:Simbolo='D';break;
                 case 1:Simbolo='E';break;
                 case 2:Simbolo='F';break;
                 case 3:Simbolo='3';point_s=-1;break;
               };
               break;
            case 0x83:
               switch(point_s){
                 case 0:Simbolo='G';break;
                 case 1:Simbolo='H';break;
                 case 2:Simbolo='I';break;
                 case 3:Simbolo='4';point_s=-1;break;
               };
               break;
            case 0x84:
               switch(point_s){
                 case 0:Simbolo='J';break;
                 case 1:Simbolo='K';break;
                 case 2:Simbolo='L';break;
                 case 3:Simbolo='5';point_s=-1;break;
               };
               break;
            case 0x85:
               switch(point_s){
                 case 0:Simbolo='M';break;
                 case 1:Simbolo='N';break;
                 case 2:Simbolo='O';break;
                 case 3:Simbolo='6';point_s=-1;break;
```

```
        };
        break;
    case 0x86:
       switch(point_s){
         case 0:Simbolo='P';break;
         case 1:Simbolo='Q';break;
         case 2:Simbolo='R';break;
         case 3:Simbolo='S';break;
         case 4:Simbolo='7';point_s=-1;break;
        };
        break;
    case 0x87:
       switch(point_s){
         case 0:Simbolo='T';break;
         case 1:Simbolo='U';break;
         case 2:Simbolo='V';break;
         case 3:Simbolo='8';point_s=-1;break;
        };
        break;
    case 0x88:
       switch(point_s){
         case 0:Simbolo='W';break;
         case 1:Simbolo='X';break;
         case 2:Simbolo='Y';break;
         case 3:Simbolo='Z';break;
         case 4:Simbolo='9';point_s=-1;break;
        };
        break;
    case 0x89:
       switch(point_s){
         case 0:Simbolo=' ';break;
         case 1:Simbolo='0';point_s=-1;break;
        };
        break;
    case 0x8b:
          lcd_clear();
          lcd_putchar('_');
          position=0;
          point_s=0;
          break;
    case 0xba:
             if(position>=1){
                 position--;
                 lcd_gotoxy(position,0);
                 lcd_putchar('_');
                 lcd_putchar(' ');

             };
                 break;
    default:Simbolo='?';break;
};
if(Code>=0x80&&Code<=0x89){
    if(Next==1)position++;
    lcd_gotoxy(position-1,0);
    lcd_putchar(Simbolo);
    lcd_putchar('_');
    Next=0;
    point_s++;
};
Gotit=1;
```

```
            PORTB=0x01;
    };
    counter++;
}
interrupt [EXT_INT0] void ext_int0_isr(void)
{
lcd_clear();
lcd_putchar('_');
TCNT1H=0x00;
TCNT1L=0x00;
PORTB=0x00;
StartBit=0;
Gotit=0;
counter=0;
position=-1;
point_s=0;
b0=0;
b1=0;
b2=0;
b3=0;
b4=0;
b5=0;
b6=0;
b7=0;


 }

interrupt [TIM1_CAPT] void Counter1_capt_isr(void)
{
Timer=ICR1L;
TCNT1H=0x00;
TCNT1L=0x00;

    switch (flag) {
          case 0:
             flag=1;
             TCCR1B=RISE;
          break;
          case 1:

             flag=0;
             TCCR1B=FALL;
             if(Timer>=TDOWN-K){
                   Width=Timer;
             };
                   if(Width>=TDOWN-K&&Width<=TDOWN+K)BitRx=0;
                   if(Width>=2*TDOWN-K&&Width<=2*TDOWN+K)BitRx=1;
                   if(StartBit==1&&Gotit==0){
                         switch (nBit) {
                               case 0:b0=BitRx;nBit++;break;
                               case 1:b1=BitRx;nBit++;break;
                               case 2:b2=BitRx;nBit++;break;
                               case 3:b3=BitRx;nBit++;break;
                               case 4:b4=BitRx;nBit++;break;
                               case 5:b5=BitRx;nBit++;break;
                               case 6:b6=BitRx;nBit++;break;
                               case 7:b7=BitRx;nBit++;break;
                               case 8:nBit++;break;
                               case 9:nBit++;break;
```

```
                        case 10:nBit++;break;
                        case 11:nBit=0;StartBit=0;deco();break;
                        default:nBit=0;StartBit=0;break;
                    };
                };
                if(Width>=4*TDOWN-2*K&&Width<=4*TDOWN+K){
                    StartBit=1;
                };
        break;
    };

}

void main(void)
{
position=0;
// Input/Output Ports initialization
// Port A initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTA=0x00;
DDRA=0xFF;

PORTB=0x00;
DDRB=0xFF;

PORTC=0x00;
DDRC=0xFF;
// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=P State6=P State5=P State4=P State3=P State2=P State1=P State0=P
PORTD=0xFF;
DDRD=0x00;

/// counter/counter 1 initialization
TCCR1A=0x00;
TCCR1B=FALL;    //0x84-->256
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// External Interrupt(s) initialization
GICR|=0x40;
MCUCR=0x02;
MCUCSR=0x00;
GIFR=0x40;

// counter(s)/counter(s) Interrupt(s) initialization
TIMSK=0x20;

// Analog Comparator initialization
ACSR=0x80;
SFIOR=0x00;

// Global enable interrupts
```

```
#asm("sei")

lcd_init(8);
lcd_clear();
lcd_putchar('_');
while (1){
    aux=TCNT1L;
    aux=TCNT1H;
    if(aux>=WAIT){
          Gotit=0;
          counter=0;
         };
     if(aux>=50){
          PORTB=0x00;
          Next=1;
          point_s=0;
         };
};
}
```

Resultados obtenidos. Proyecto IRDECO2.

El resultado final de todos los proyectos elaborados en este capítulo ha sido la creación del proyecto IRDECO2 que permite mostrar en un LCD un mensaje de texto escrito a través de un Control Remoto Sony RM V8. El programa funciona sin inconvenientes. La velocidad de escritura al principio parece bastante rápida y no se alcanza a escribir los símbolos deseados en el LCD antes de que el cursor pase a la siguiente posición, sin embargo una vez que el usuario se acostumbra al teclado y con ayudas visuales que indiquen que símbolos hay disponibles en cada botón del Control Remoto, la destreza del usuario aumenta a tal punto que en cambio parece que la velocidad de escritura es algo lenta. Cualquier modificación de la velocidad de escritura para que se adapte mejor al usuario estándar se puede lograr cambiando el tiempo de 0,4096 s utilizado como referencia en este programa. Aunque el objetivo principal es mostrar mensajes de texto en un Display Rotativo y no en un LCD, la elaboración de IRDECO2 junto con sus predecesores ha servido para entender mejor el funcionamiento del ATmega32 y para tener lista la transmisión de comandos desde un Control Remoto para su futura aplicación en el Display Rotativo.

CAPÍTULO 6

IMPLEMENTACIÓN DEL DISPLAY ROTATIVO

6. IMPLEMENTACIÓN DEL DISPLAY ROTATIVO.

Al implementar el Display Rotativo se debe tener en cuenta que es un prototipo, es decir, las piezas que lo constituyen y el programa que lo hacen funcionar deben ser elaborados o adaptados para satisfacer los objetivos planteados.

Las piezas utilizadas para sustentar el proyecto han tenido que ser elaboradas casi artesanalmente a la medida de los requerimientos del Display Rotativo, utilizando derivados de madera, plástico y metal. Tanto el microcontrolador ATmega32, así como los LEDs RGB han sido importados ya que no hay distribuidores de estos productos conocidos en nuestro mercado. La construcción del prototipo en su forma más básica es necesaria antes de iniciar la programación, ya que no hay forma de probar si el programa del Display Rotativo funciona correctamente, sin hacer que los LEDs giren primero.

Ha sido muy útil para la implementación del Display Rotativo haber realizado con anterioridad el diseño y construcción de un Detector SIRC para un Control Remoto Universal RM V8 puesto que esto asentó una base de pruebas para el microcontrolador ATmega32 y afianzó los conocimientos sobre la comunicación IR.

Al momento de elaborar el código del programa, es necesario tener claro el funcionamiento planteado para el Display Rotativo, considerando las necesidades y comodidad del usuario, y la factibilidad de hacerlo. Además, debido a que hay

muchas actividades a realizar al mismo tiempo como por ejemplo: la comunicación IR, la presentación de Mensajes o imágenes a través de los LEDs, el sincronismo a través del sensor de movimiento, la hora actual, etc. es conveniente realizar el programa de la manera más estructurada y clara para que pueda ser entendido por cualquier persona y modificado si es necesario, sin realizar mayores alteraciones en su código.

El diseño y construcción del Display Rotativo representa un reto de diseño y construcción que mezcla varios elementos de mecánica, electrónica, efectos visuales y programación en C++. Además tiene dos objetivos claros. el primero es crear un dispositivo confiable llamativo y competitivo que represente una opción al momento de escoger entre las varias posibilidades que ofrece el mercado de los anuncios luminosos digitales y el segundo es brindar a las personas interesadas en el uso de microcontroladores, un guía útil y didáctica que puede ser una fuente de consulta para futuros proyectos.

Características propuestas para el Display Rotativo.

Para lograr que el Display Rotativo satisfaga las necesidades del usuario debe cumplir con ciertas características que lo hagan atractivo al momento de ser utilizado. Estas características son:

- El texto debe mostrarse nítido, fijo y sin alteraciones de la información mostrada. Es muy común ver en prototipos de Displays Rotativos que el texto no se muestra siempre en el mismo lugar, a veces gira sin control o parpadea. Estos problemas deben ser resueltos al diseñar e implementar este Proyecto.

- El Display Rotativo debe ser fácil de utilizar y programar. En la actualidad existen varios modelos de Displays Rotativos en el mercado cuya programación se complica debido a que utilizan teclados con pocas teclas y el usuario tiene que moverse de menú en menú y de letra en letra para programar alguna función o Mensaje en el Display Rotativo.

- El usuario debe tener varias opciones al momento de hacer funcionar el Display Rotativo. Estas opciones serán:

    - Mostrar al menos tres Mensajes distintos que pueden ser grabados por el usuario. Los Mensajes no deben perderse cuando se apague o desconecte el Display Rotativo.

    - Mostrar la hora actual con gran precisión.

    - Mostrar los Mensajes solo uno a la vez o de manera alternada uno tras otro para formar Mensajes más largos.

    - Hacer girar el Mensaje de manera controlada a la izquierda o a la derecha.

    - Explotar las características de los LEDs RGB para dar efectos cromáticos a los Mensajes, tales como alternar colores por Mensaje o por letras, de manera manual o automática.

- El Display Rotativo debe ser seguro. El hecho de que los LEDs estén girando a gran velocidad hace que su funcionamiento sea peligroso; ya que, si el usuario interfiere con el movimiento de los LEDs puede dañarlos y peor aún salir lastimado. Se deben tomar medidas al respecto; tales como, poner una cubierta plástica transparente alrededor de los LEDs.

- Como detalle final, el Display Rotativo debe tener una buena presentación. Cada detalle del dispositivo debe ser considerado desde el punto de vista funcional y de estética. Se debe tener en cuenta la forma y color de la base y la cubierta, la posición de los LEDs, el microcontrolador, interruptores y demás conexiones.

Descripción básica de funcionamiento.

Persistencia Visual.

Cuando una escena en movimiento es dividida en una secuencia de imágenes, y éstas se muestran a un observador en una sucesión rápida, el cerebro es capaz de captar el movimiento de la escena; este fenómeno se denomina Persistencia Visual. Para que el cerebro perciba el movimiento reproducido por las imágenes de forma natural, sin espasmos, su frecuencia de presentación debe ser de 15 o más imágenes por segundo[1], esta frecuencia se mide fps (frames per second).

Así por ejemplo, en el caso del formato de televisión americano conocido como NTSC[2], se trabaja con una frecuencia de 30 fps, es decir, se muestra 30 imágenes por segundo. En el caso del formato PAL[3] utilizado en Europa, la frecuencia es de 25 fps.

Las cámaras de video utilizadas en Internet, junto con las cámaras de seguridad, trabajan a 15 fps[4], lo cual todavía muestra una continuidad de movimiento aceptable y entendible. En el caso de las animaciones, los elementos mostrados en escena son mejor definidos, por lo que éstas pueden ser presentadas hasta con 12 fps, siempre y cuando, los movimientos que muestren no sean demasiado rápidos.

El Display Rotativo utiliza el fenómeno de la Persistencia Visual para presentar mensajes de texto e imágenes a través de una columna de 8 LEDs RGB que gira alrededor de un eje a una velocidad aproximada de 12 vueltas por segundo (el motor utilizado para hacer girar la columna de LEDs trabaja a 730 RPM). Estos mensajes de texto e imágenes dan la impresión de mantenerse flotando en el

[1] Información obtenida de la página de Internet http://www.wisegeek.com/how-does-a-television-work.htm.
[2] NTSC signifca National Television Standards Committee.
[3] PAL significa Phase Alternate Line. Información obtenida de de la página de Internet http://en.wikipedia.org/wiki/Frame_rate.
[4] Información obtenida de de la página de Internet http://psych.hanover.edu/nsfati/APASession1 /Sequences.ppt.

aire, ya se de manera estática o deslizándose hacia la derecha o izquierda, debido a que se muestra un *frame* de manera secuencial en cada vuelta.

Funcionamiento del Display Rotativo.

El dispositivo que se presenta a continuación en la Figura 6.1. es un Display Rotativo donde los 8 LEDs colocados a un extremo de una base, giran gracias al motor al cual están unidos. El chip que controla el funcionamiento del dispositivo es un ATmega32.

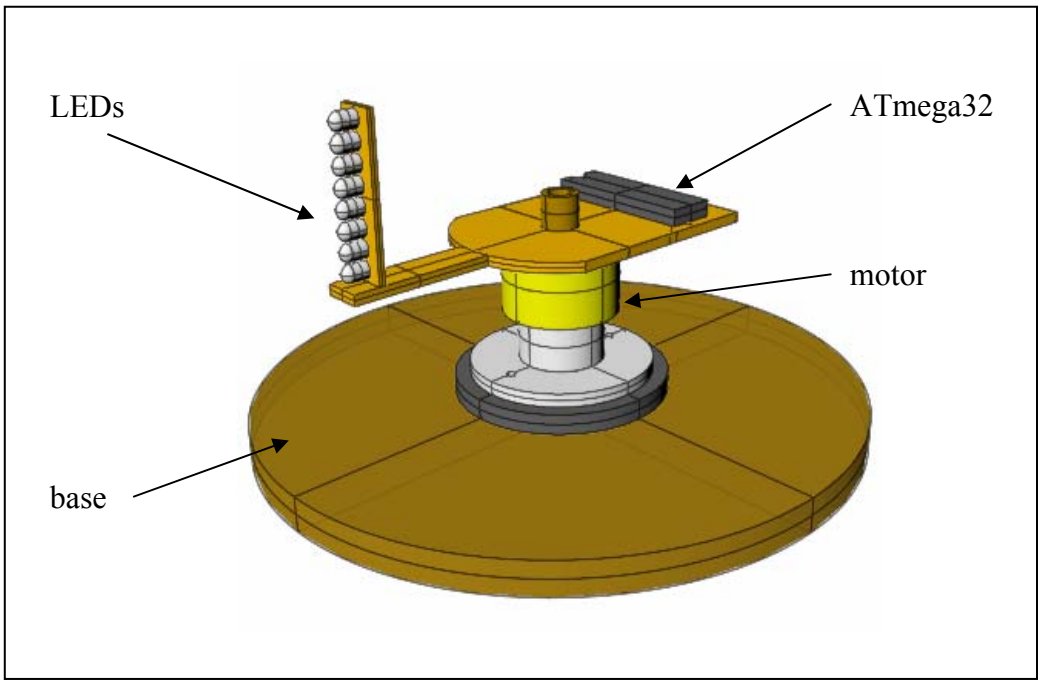


Figura 6.1. Display Rotativo básico

El prototipo se basa en el principio de la permanencia de imágenes en la retina, cuando éstas pasan rápidamente frente a un observador. Por ejemplo, si se encendiera solo el primer LED mientras el rotor está girando, se formaría un círculo luminoso como el mostrado en la Figura 6.2.

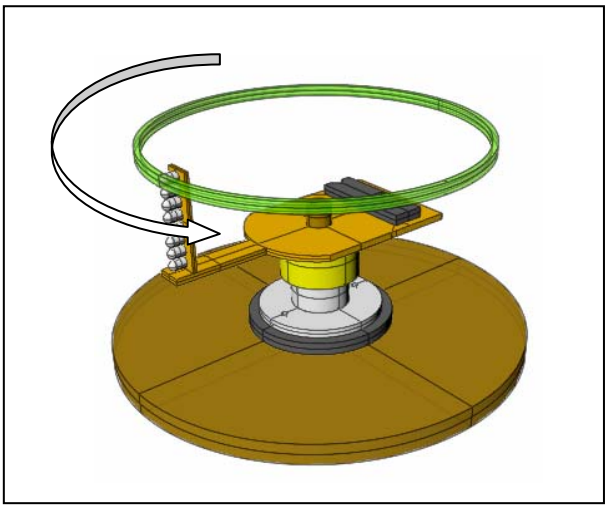


Figura 6.2. Círculo luminoso

Si se controla el tiempo que el LED está encendido haciendo que se encienda solo por pequeños lapsos de tiempo (cientos de microsegundos), se puede crear la ilusión de líneas girando en el espacio como se muestra en la Figura 6.3.

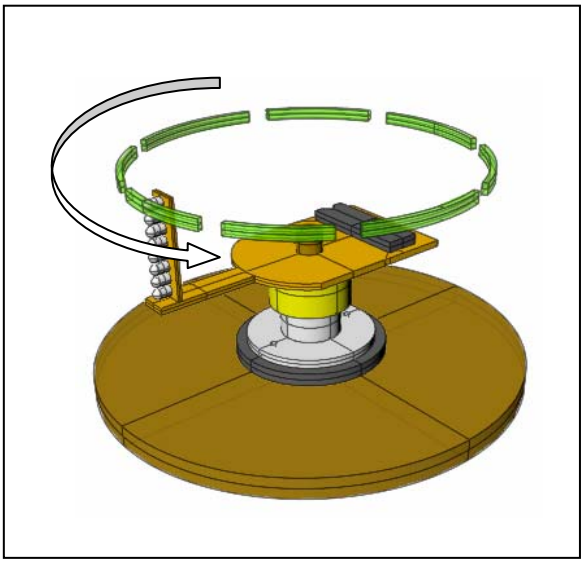


Figura 6.3. Líneas en el espacio

Utilizando el mismo principio en cada uno de los 8 LEDs y controlando de manera individual y precisa cada uno de ellos, es posible mostrar texto o imágenes a través del Display Rotativo. Así por ejemplo, para mostrar la letra “A”, inicialmente se muestra la primera columna de la letra, como se muestra en la Figura 6.4.

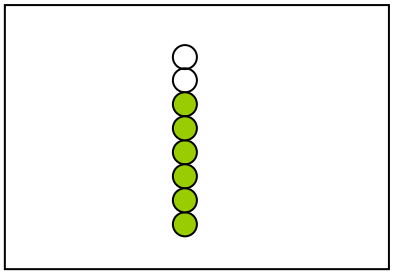


Figura 6.4. Primera columna de la letra “A”

La columna de LEDs se mantiene girando en sentido antihorario; es decir, si se mira de frente, de izquierda a derecha. Luego de 500 µs, se muestra la segunda columna cambiando los LEDs que se deben encender. Debido a que es un proceso muy rápido la impresión que dejó la primera columna en la retina del ojo del observador, sigue presente a la izquierda de la segunda columna, como se muestra en la Figura 6.5.

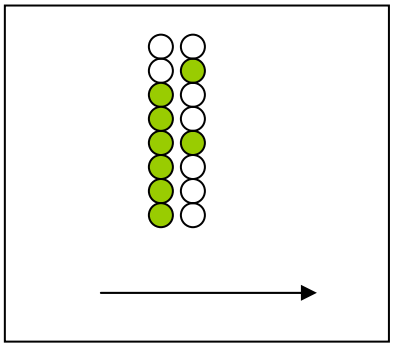


Figura 6.5. Primera y segunda columna de la letra “A”

Si se repite el mismo proceso algunas veces más cambiando los LEDs que se deben encender cada vez que se muestra una columna se puede completar la letra “A”. Se puede utilizar 5 columnas como se muestra en la Figura 6.6. para

formar caracteres de 8 x 5 píxeles. También se puede emplear 7 columnas para obtener una mejor resolución.

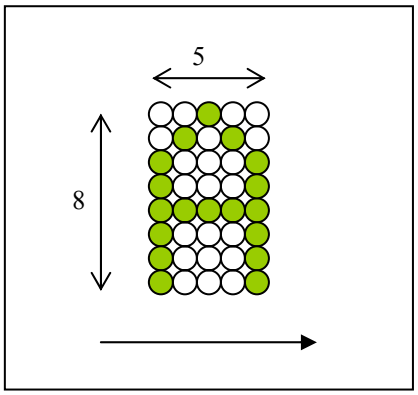


Figura 6.6. Letra "A" (8 x 5)

Siguiendo el mismo procedimiento se pueden formar palabras e inclusive Mensajes completos como el que se muestra en la Figura 6.7. Esos Mensajes dan la impresión de estar flotando en el aire debido a que las demás partes del Display Rotativo no se ven cuando el rotor está girando. El mismo principio se utiliza para mostrar imágenes, que además pueden ser de mejor calidad, si se cambia el color con que se enciende cada LED.

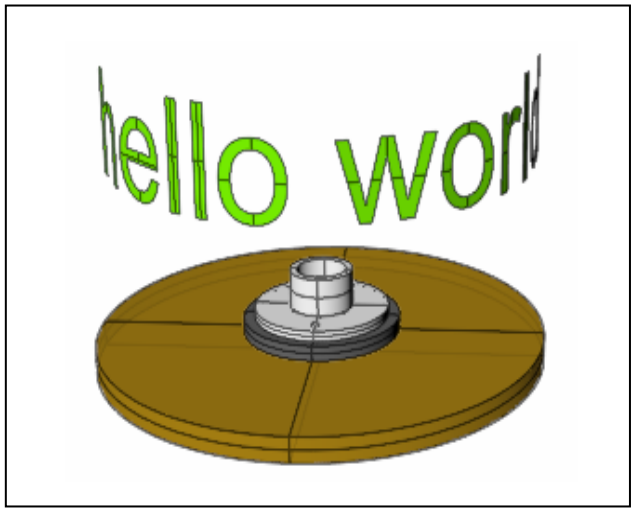


Figura 6.7. Mensaje "hello world"

Para que el Display Rotativo tenga un desempeño óptimo, requiere de más elementos como sensores, una interfase de control, cubiertas, entre otros. Sin embargo, su principio de funcionamiento es el ya mencionado.

Construcción del Display Rotativo.

Base Estática.

Debido a que el área barrida por los LEDs es bastante grande se produce una gran inercia. Por lo tanto, si la base del Display Rotativo que está en contacto con la superficie de apoyo (una mesa o un estante) no es lo suficientemente amplia, el dispositivo no podría mantener una posición estable.

La Base Estática es una pieza circular plana hecha de MDF[5]. En su parte inferior está cubierta por una moqueta de PVC para darle más adherencia con la superficie de contacto. En el centro de la base hay dos piezas circulares de moqueta adheridas entre sí para disminuir las vibraciones, y sobre ellas va el soporte del motor.

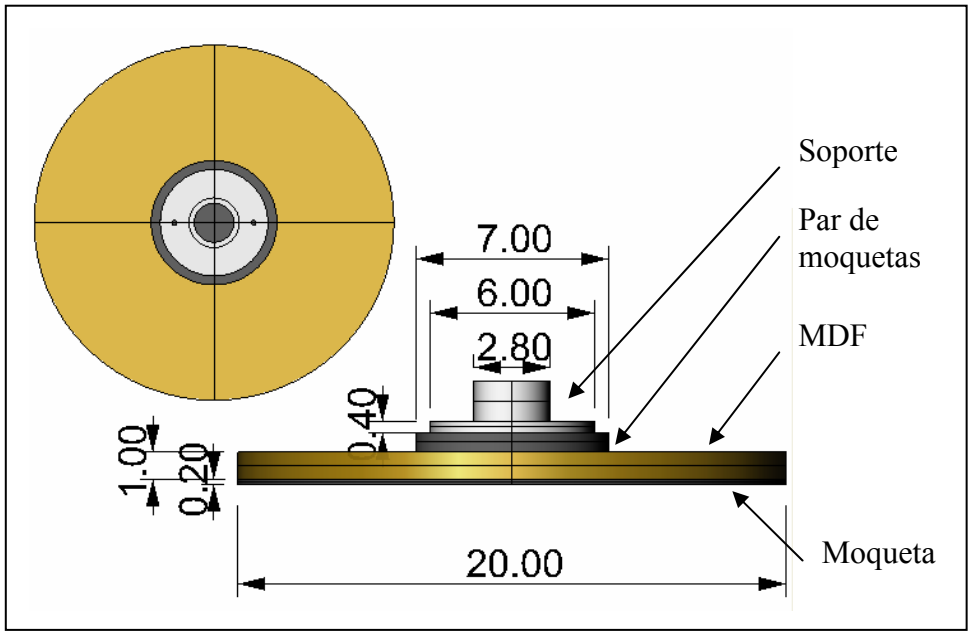


Figura 6.8. Base Estática

[5] Medium Density Fiberboard. Material hecho de fibras de madera, más duro que el cartón.

Motor sin escobillas.

Debido a que el Display Rotativo debe ser silencioso, ya que su uso será generalmente dentro de una habitación, se requiere de un motor que no haga ruido. No se puede utilizar un motor DC normal, puesto que estos motores tienen escobillas que producen ruido al rozar con el colector cuando el rotor está girando.

La mejor opción es utilizar un motor DC sin escobillas conocido también como *DC Brushless Motor*. Este tipo de motores son utilizados en los ventiladores que refrescan internamente las computadoras como el que se muestra en la Figura 6.9.

Figura 6.9. Ventilador de computadora

Los motores DC sin escobillas son pequeños y fáciles de conseguir ya que se puede utilizar el de un ventilador de computadora, nuevo o usado. Estos motores trabajan con un voltaje de 12 VDC, no hacen ruido y giran a gran velocidad, aproximadamente 730 RPM (12 vueltas por segundo).

Los motores DC sin escobillas funcionan al revés que los motores DC normales, las bobinas en el interior permanecen estáticas y el imán permanente externo es el que gira. Así, el estator está hecho generalmente de 4 bobinas y un circuito de control, mientras que el rotor consta de un imán permanente circular y una cubierta metálica.

La parte crucial en el funcionamiento del motor DC sin escobillas es el circuito de control debajo de las bobinas. Hay dos grupos de bobinas, cado uno formado por dos bobinas opuestas. El circuito de control está dividido en dos circuitos iguales, cada uno de los cuales maneja el encendido de cada par de bobinas a través de un transistor. La clave del circuito de control es un sensor capaz de detectar cambios en un campo magnético; llamado sonda de *efecto Hall*, el cual está colocado justo a lado del rotor. La sonda de Hall detecta la polaridad del campo magnético producido por el imán permanente que pasa frente a ella y según esto controla que par de bobinas se enciende para dar una rotación continua y en la misma dirección al rotor. Así, se consigue que el motor gire.

Búsqueda de una fuente de alimentación adecuada.

Uno de los retos más decisivos a enfrentar es la manera en que se va a alimentar a las partes que giran en el Display Rotativo; es decir, los LEDs RGB y el microcontrolador AVR junto con el resto de elementos.

La opción más simple es utilizar una batería que entregue el voltaje de 5V necesario para que los LEDs RGB y el microcontrolador AVR funcionen. Esto se podría lograr usando cuatro baterías de 1.5V o una de 9V junto a un regulador de 5V. El problema de usar baterías es que a pesar de que el microcontrolador utiliza tan solo 15 mA aproximadamente[6], las baterías se agotarían muy rápido debido al alto consumo de los LEDs, el cual es de 20 mA[7], cada uno (160 mA en total). Esto supera la corriente que pueden brindar la mayoría de pilas, en especial las recargables, en el caso de que se quiera utilizar este tipo de baterías para ahorrar dinero a largo plazo.

Existen soluciones más complejas, como por ejemplo el uso del devanado del rotor para obtener un voltaje AC y luego rectificarlo. Sin embargo esto obligaría a utilizar un motor normal con escobillas, lo cual sería un inconveniente para el

[6] Información obtenida del manual del ATmega32 del capítulo: ATmega32 Typical Characteristics.
[7] Información obtenida de las hojas de datos técnicos del RL5-RGB-D

usuario ya que estos motores son ruidosos en comparación a los motores sin escobillas.

En este proyecto se propone el uso del propio eje del rotor del motor sin escobillas para alimentar las partes giratorias. Esta solución es algo complicada de implementar ya que, además de la parte eléctrica, requiere de una adaptación especial de las partes mecánicas, pero una vez realizada ofrece la mejor solución al problema de alimentación ya que, brinda una fuente de voltaje DC estable y permite que el proyecto implementado trabaje de una manera silenciosa.

Fuente de Alimentación.

Para alimentar el circuito que está en el rotor es necesario hacer unos cambios en el eje del motor DC sin escobillas y en el anillo metálico a su alrededor. Desarmando el motor con mucho cuidado, en especial con las bobinas, se puede apreciar que el rotor gira dentro de un anillo metálico, como se muestra en la Figura 6.10.

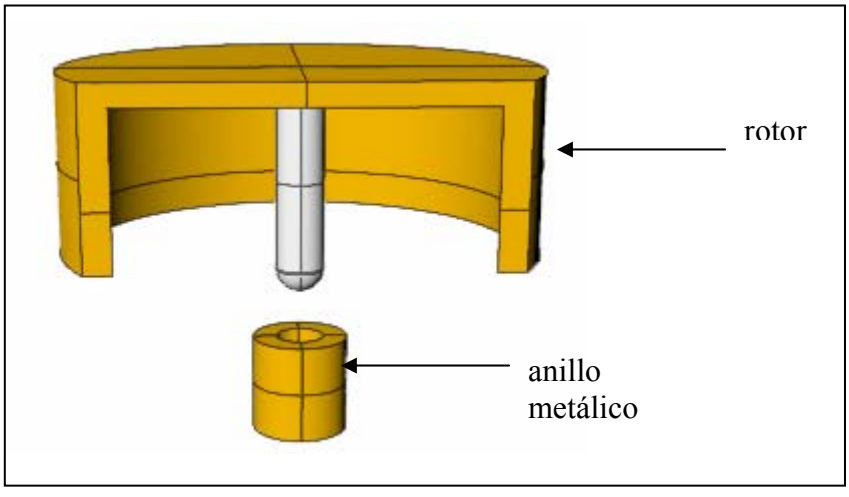


Figura 6.10. Rotor y anillo metálico del motor sin escobillas

Para poder alimentar el circuito que estará sobre el rotor, se va a utilizar el eje del propio rotor. El eje y el anillo están en contacto todo el tiempo, por lo que se podría utilizarlos para transmitir energía al circuito. El inconveniente está en que

se necesitan dos terminales aislados para así tener uno que sea positivo y otro negativo.

Para tener dos polos se debe hacer ciertas modificaciones al motor. Primero se debe cambiar el eje del rotor por un plug de audio monaural ¼ como el mostrado en la Figura 6.11. Los plug de audio monaurales están divididos internamente en dos terminales que se ajustan perfectamente a la necesidad del sistema.

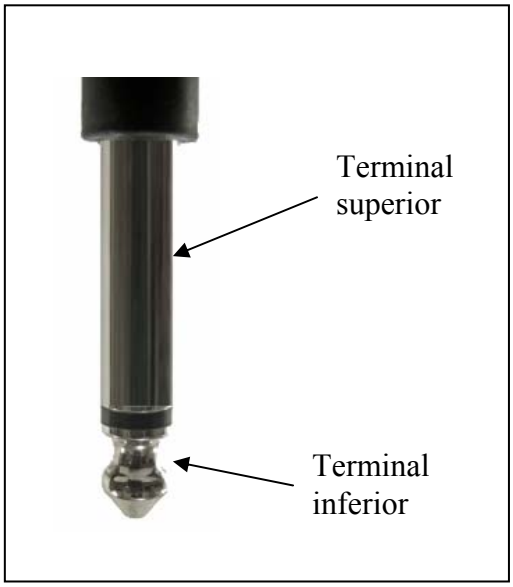


Figura 6.11. Plug de audio monaural ¼

Luego se debe disminuir el alto del anillo para que solo entre en contacto con unos de los terminales del nuevo eje, el terminal superior.

Finalmente se debe colocar el antiguo eje como una base sobre la cual el nuevo eje girará, como se muestra en la Figura 6.12. Debido a que la gravedad es la que mantiene el eje constantemente sobre su base, si se da la vuelta al motor la alimentación del circuito se cortaría.

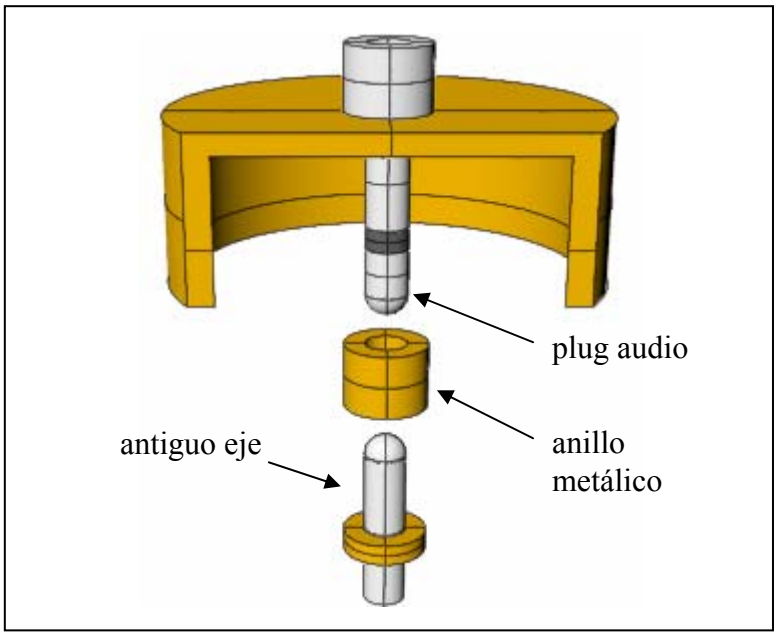


Figura 6.12. Eje del rotor modificado

Finalmente se sueldan cables a cada terminal y así se consigue llevar energía al circuito. Con todas las piezas modificadas ya ensambladas, el motor quedaría como se muestra en la Figura 6.13.

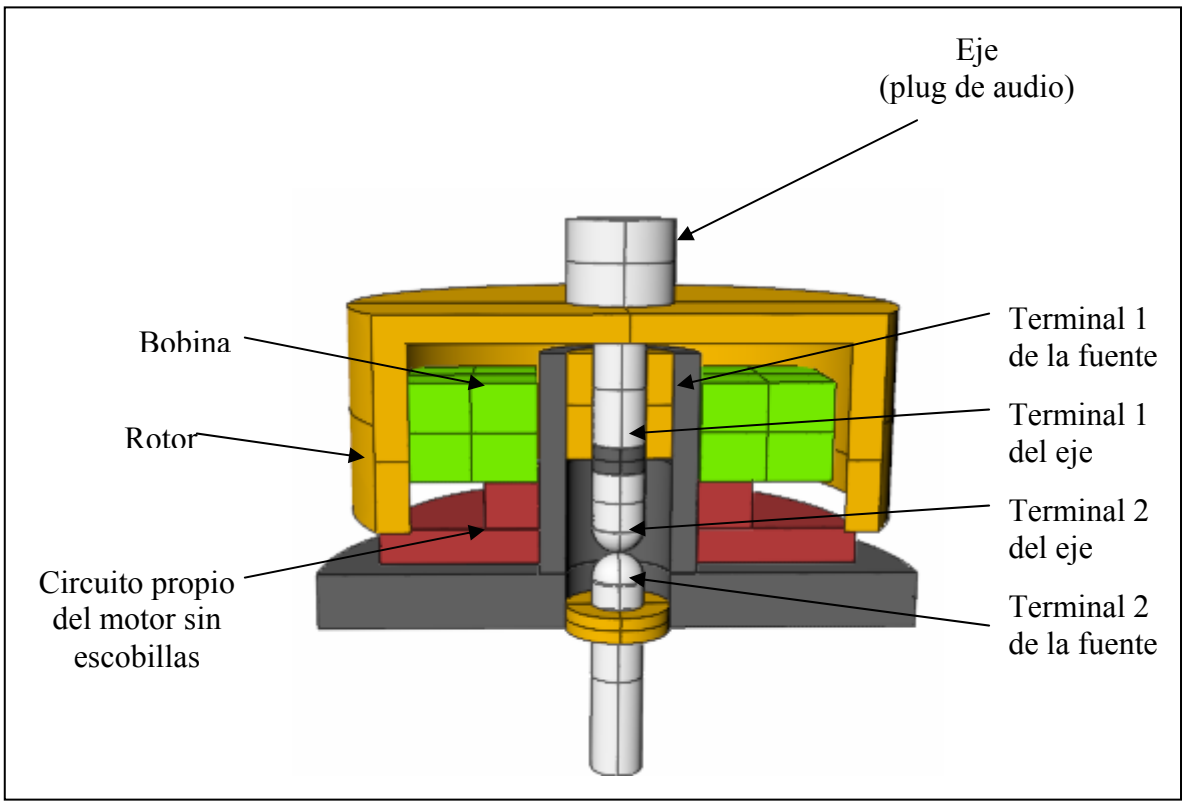


Figura 6.13. Motor sin escobillas modificado (completo)

Base del Rotor.

La Base del Rotor es simplemente una plancha de MDF de 3 mm de ancho cuya función es servir de soporte para el circuito del microcontrolador y la columna de ocho LEDs, como se muestra en la Figura 6.14.

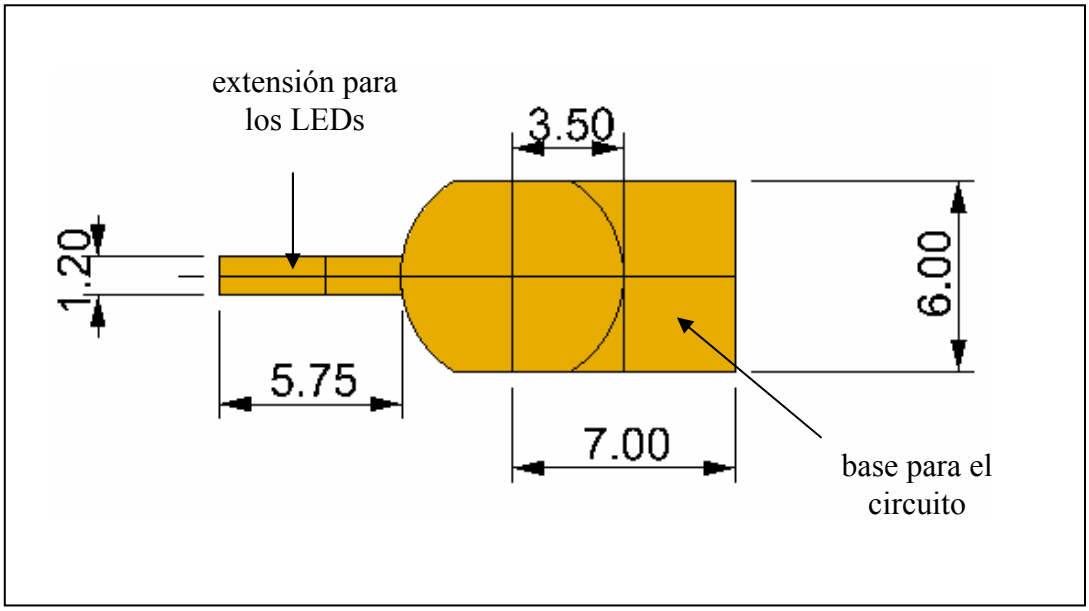


Figura 6.14. Base del Rotor

Circuito del Rotor.

El Circuito del Rotor se debe armar sobre una superficie lo suficientemente grande para que contenga al microcontrolador ATmega32, el sensor de la interfase IR, el jumper de grabado, cables y demás elementos electrónicos como resistencias y capacitores, sin sobrepasar las dimensiones de la base del Rotor. Una superficie de 4,5 x 5,5 cm es suficiente para cumplir con esta función. El circuito completo que debe ser implementado se muestra en la Figura 6.15.

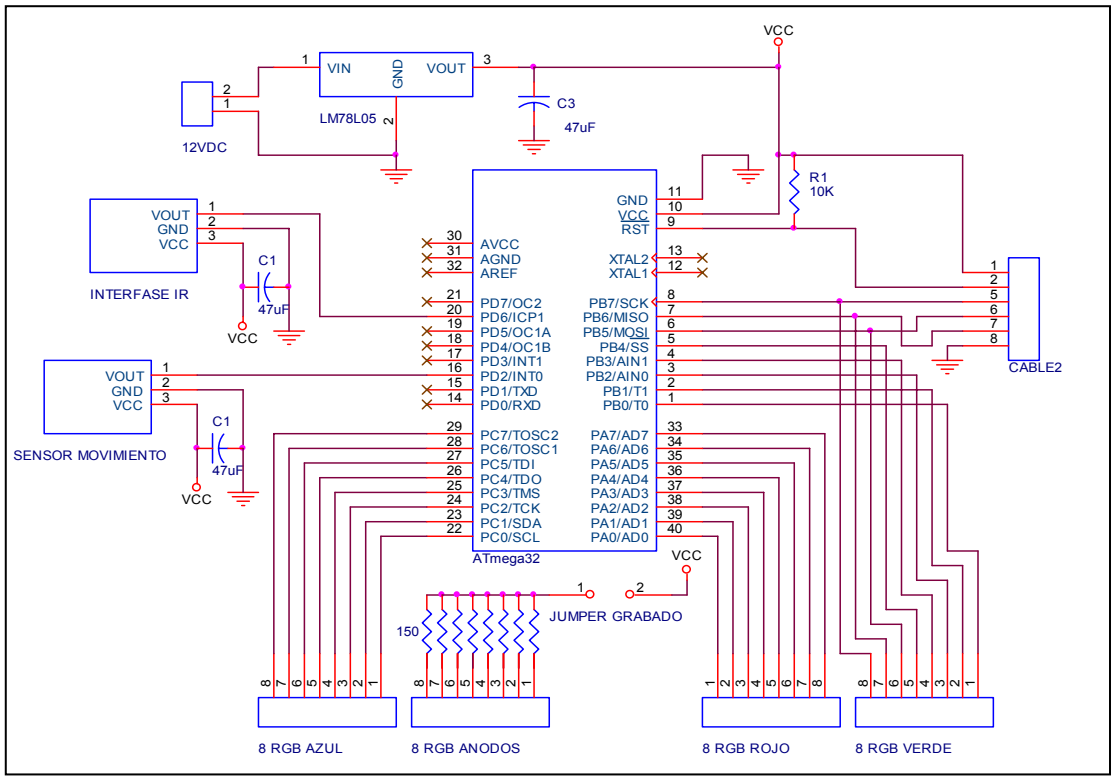


Figura 6.15. Circuito del Rotor

Al implementar el circuito queda de la manera mostrada en la Figura 6.16.:

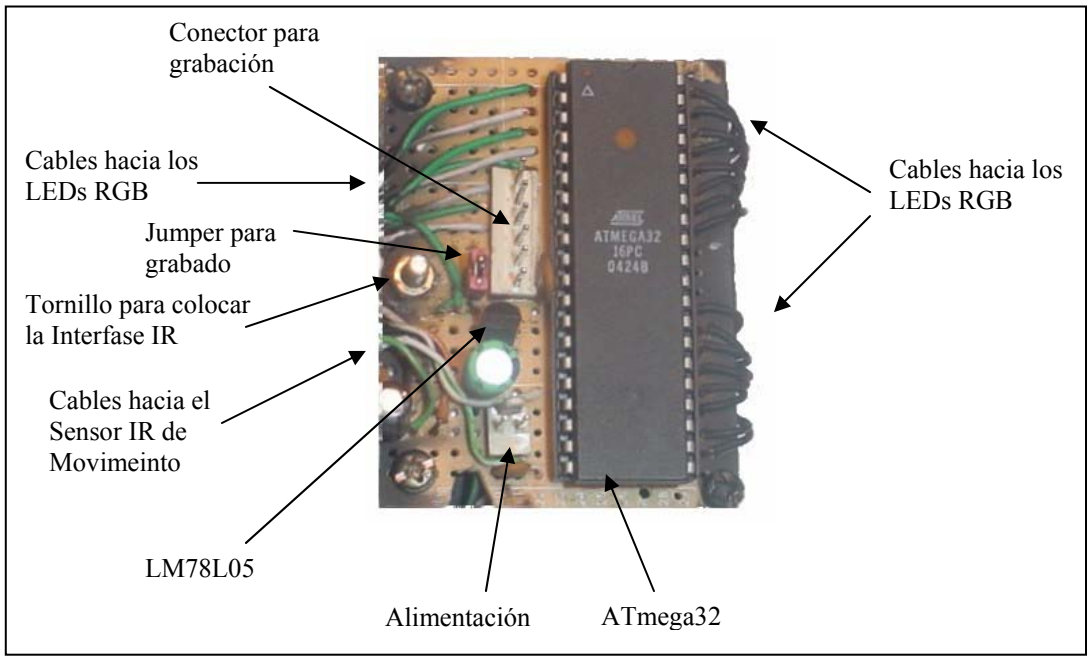


Figura 6.16. Circuito del Rotor implementado

Se puede dividir el Circuito del Rotor en las siguientes partes principales:

- **Microcontrolador ATmega32.** La parte fundamental del circuito, encargada de manejar las demás partes. Recibe los comandos enviados desde el Control Remoto a través del pin PD6/ICP1 y las señales del Sensor de Movimiento a través del pin PD2/INT0 cuando el eje del Rotor pasa sobre el LED IR. Controla los ocho LEDs RGB a través de tres puertos: PORTA para el color rojo, PORTB para el color verde y PORTC para el color azul.

- **Regulador LM78L05.** Se encarga de alimentar al circuito con un voltaje regulado a 5VDC. Con su capacitor de 47 uF ayuda a contrarrestar el ruido producido cuando el Rotor gira y ocasiona un rozamiento entre los contactos que alimentan de energía al circuito a través del eje del Rotor.

- **Conector para grabación.** A través de los seis pines de este conector se puede grabar el programa en el ATmega32 al conectar el Circuito del Rotor con el Grabador PonyProg2000 por medio del Cable 2.

- **Jumper para Grabado.** Debido a que los pines PB5, PB6 y PB7 son usados tanto para grabar el programa en el microcontrolador como para controlar el color verde de los LEDs RGB, se necesita usar un jumper que habilite una de las dos funciones. Cuando un puente une los dos pines del jumper, se habilita los LEDs RGB. Para poder grabar algún programa en el microcontrolador se debe retirar dicho puente.

- **Interfase IR.** Permite comunicar el Display Rotativo con el Control Remoto. Consiste en un Detector IR. De los tres pines del Detector IR, uno va conectado a VCC, otro a tierra y otro al pin PD6/ICP1. Un capacitor de 47 µF debe ser colocado entre VCC y tierra. Este bloque se encarga de recibir las señales desde el Control Remoto por lo que se debe colocar el Detector IR en una parte elevada en el centro de giro del circuito del Rotor.

- **Sensor de Movimiento.** Consiste en un Detector IR. De los tres pines del Detector IR, uno va conectado a VCC, otro a tierra y otro al pin PD6/ICP1. Un capacitor de 47 µF debe ser colocado entre VCC y tierra. Este Detector IR está fuera del circuito apuntando hacia un LED emisor IR con el que se debe alinear una vez por vuelta. Este sensor se encarga de indicar al microcontrolador que inicie el despliegue del Mensaje de texto o de la Imagen Prediseñada cada vez que la columna de LEDs pase sobre el LED emisor IR para que la presentación se mantenga estática.

- **LEDs RGB.** Se encuentran fuera del circuito del Rotor en la parte más extrema del eje horizontal de la base del Rotor. Cada uno de los ocho LEDs tiene cuatro pines, uno que es su ánodo común va conectado a VCC a través de una resistencia de 150 ohms y un jumper habilitador. Los otros tres pines de cada LED van conectados a los respectivos puertos de control en el microcontrolador.

Microcontrolador ATmega32.

El Display Rotativo maneja ocho LEDs RGB y dos sensores IR. Cada LED RGB tiene tres pines de control y un ánodo común, es decir que, para manejar 8 LEDs RGB se requieren 24 salidas en el microcontrolador además de una entrada por cada sensor IR. Para satisfacer este requerimiento se utiliza el ATmega32 que cuenta con 4 puertos de 8 bits cada uno, lo cual da en total 32 salidas / entradas.

Además, el ATmega32 tiene tres Interrupciones Externas y tres Timers que pueden ser muy útiles para el proyecto. También posee una EEPROM que puede guardar datos importantes aún cuando el ATmega32 este apagado.

Debido a que el programa del Display Rotativo no utiliza mucha memoria (aproximadamente 2 KB) se podría reemplazar al ATmega32 por un ATmega16 que tiene la mitad de memoria y un precio inferior, pero sus demás características son iguales como se muestra en la Tabla 6.1. Sin embargo el ATmega32 es ideal para hacer pruebas sin tener que preocuparse por limitaciones de memoria.

Device	ATmega16	ATmega32
Flash (KB)	16	32
EEPROM (KB)	0.5	1
SRAM (*Bytes*)	1024	2048
Max I/O Pins	32	32
F.max (MHz)	16	16
Vcc (V)	2.7-5.5	2.7-5.5
16-bit Timers	1	1
8-bit Timer	2	2
Ext Interr	3	3
Precio	$6,25 (Jameco)	$11,95 (Kanda)

Tabla 6.1. Comparación entre ATmega32 y ATmega16

Se puede establecer que cada puerto controle un color de los ocho LEDs RGB, así Port A controlaría el color rojo, Port B el verde y Port C el azul. Port D se utiliza para manejar los sensores IR. La señal del sensor IR que recibe la señal del control remoto trabaja con el Timer1, así que debe ir conectada al pin PD6 que corresponde al Input Capture del Timer1. La señal del sensor IR que sincroniza el funcionamiento del Display Rotativo trabaja con la Interrupción Externa INT0; así que, debe ir conectada en el pin PD2.

Ya que el ATmega32 tiene un oscilador RC interno, no será necesario conectar un cristal al microcontrolador y debido a que algunos procesos de temporización son largos, se calibrará el Reloj del Sistema en 2 MHz en vez de 8 MHz, como se ha hecho anteriormente.

Interfase IR.

La interfase IR se utiliza para ingresar y controlar los Mensajes desplegados en el Display Rotativo; así como, para configurar los efectos de movimiento y cromáticos del Mensaje.

El control se lo realiza a través de un control remoto de TV Sony, de preferencia Universal, aunque un control estándar de TV Sony también es aceptable.

La señal es recibida por un sensor IR conectado al ATmega32 a través del pin PD6, como se explicó en la sección 5.2. Módulo Detector IR. El sensor IR debe ser colocado en el centro de giro del rotor y a un nivel más alto que el resto del circuito para tener una mejor recepción. Además se debe poner una cubierta plástica para sensores IR para protegerlo de la interferencia IR del ambiente, en especial de luces artificiales como focos incandescentes o fluorescentes. La implementación del Módulo Detector IR se muestra en la Figura 6.17.

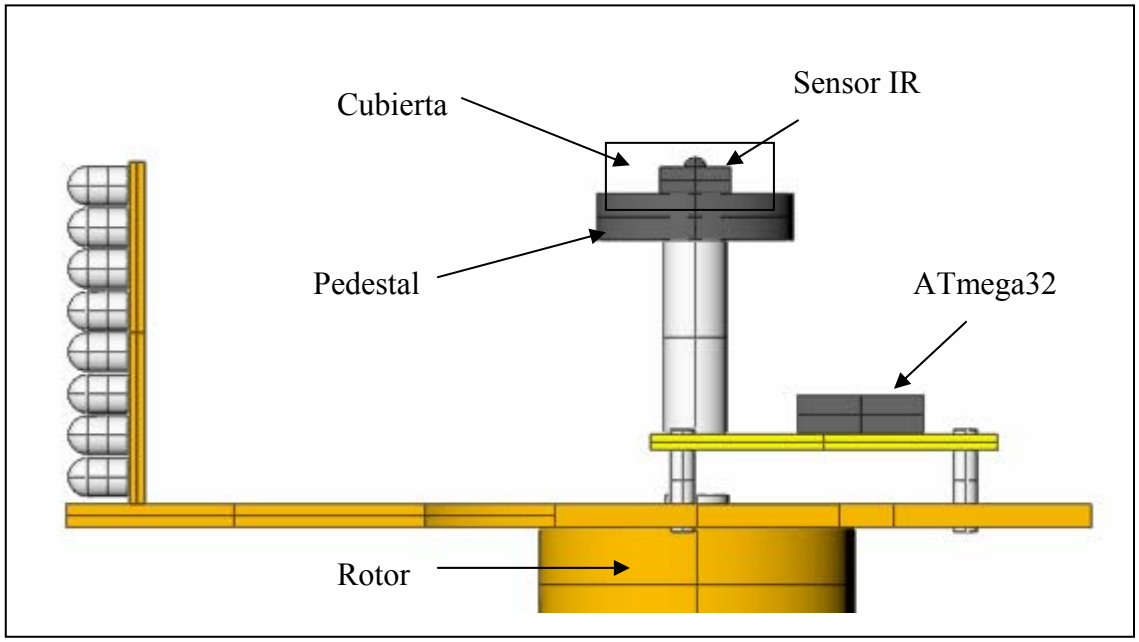


Figura 6.17. Módulo Detector IR

Sensor de Movimiento.

El sensor de movimiento da sincronismo al Display Rotativo; ya que, le indica donde debe empezar a mostrar el Mensaje. Consiste de un sensor IR que recibe la señal de un LED emisor IR colocado en la base del Display Rotativo. Para que su funcionamiento no interfiera con el sensor IR que trabaja con el Control Remoto, ni tampoco reciba interferencia del Control Remoto o del ambiente, el

sensor de movimiento y el LED emisor IR deben estar cubiertos parcialmente con *espagueti térmico* para darles mayor directividad. También es recomendable poner una cubierta de plástico transparente opaco para IR sobre el LED emisor IR. Tanto el Sensor de Movimiento IR como el LED emisor IR se muestran en la Figura 6.18.

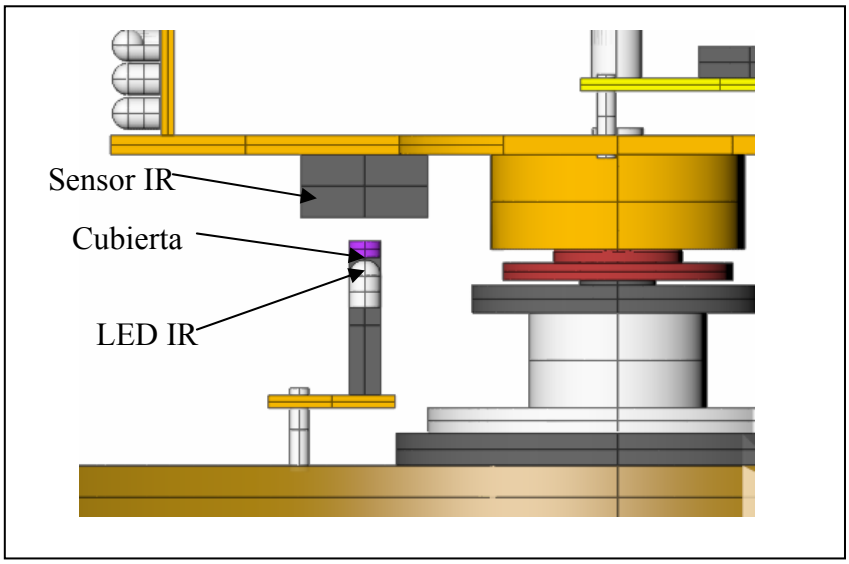


Figura 6.18. Sensor de Movimiento y LED emisor IR

Debido a que el sensor IR tiene varias cubiertas de *espagueti térmico* a su alrededor, se vuelve más grueso y ocupa más espacio. Además, se debe hacer un pequeño agujero a la entrada del sensor para que pueda entrar solo la señal del LED IR.

El LED IR es la única parte del circuito del Display Rotativo que no se encuentra en el Rotor sino en la base. Debido a que se necesita que el LED IR emita una señal IR modulada a una frecuencia de 38 KHz, se arma junto al mismo un circuito oscilador utilizando un Timer 555 como se muestra en la Figura 6.19.

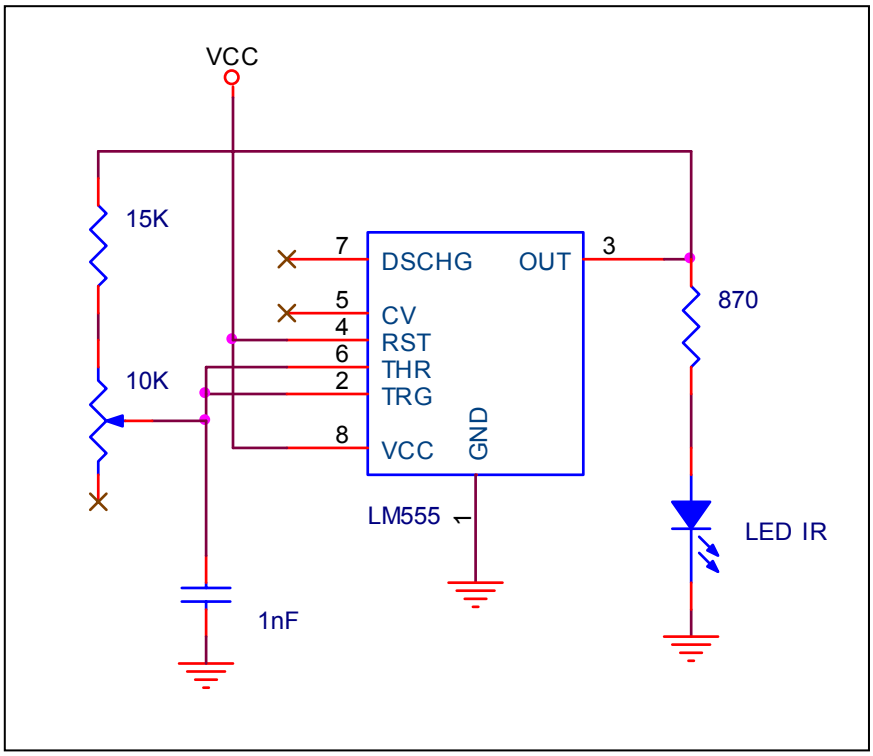


Figura 6.19. Circuito oscilador del LED emisor IR

Al construir el circuito oscilador del LED emisor IR, queda de la manera mostrada en la Figura 6.20.:

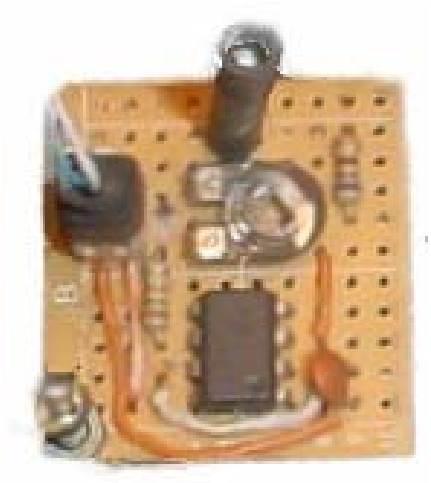


Figura 6.20. Circuito oscilador del LED emisor IR implementado

Se debe asegurar que el sensor IR y el LED IR estén alineados para que la señal que activa el sensor de movimiento se active cada vez que el senor IR pase sobre el LED IR.

Arreglo de LEDS Monocromáticos y RGB.

El Display Rotativo muestra el Mensaje a través de una columna de ocho LEDs colocados al extremo del Rotor. A su vez, cada LED es activado en bajo por lo que el ánodo de cada LED se conecta a VCC y cada cátodo va a uno de los ocho pines del puerto que maneja un determinado color a través de una resistencia de 220 ohms.

La diferencia entre usar LEDs monocromáticos y RGB está en el número de cables que se requiere para su manejo. Con LEDs monocromáticos solo se requieren nueve cables, uno que hace las veces de VCC común y ocho más que conectan los LEDs con un solo puerto de 8 bits.

En el caso de usar LEDs RGB, se requieren veinte y cinco cables, uno que hace las veces de VCC común y tres grupos de ocho cables, un grupo por cada puerto de 8 bits que maneja un respectivo color. Ya que colocar veinte y cuatro resistencias de 220 ohms resulta bastante complicado, es mejor poner una sola resistencia por LED conectada entre el mismo y VCC, además reduciendo su valor a 150 ohms que permitiría llegar al límite de trabajo de 20 mA de los LEDs RL5-RGB-D en el color Rojo que es el que a menor voltaje trabaja (2V).

Debido a que los cables y las resistencias dan una mala imagen a la presentación del Display Rotativo, es necesario poner una cubierta alrededor de toda esta parte del prototipo dejando libres solo los LEDs, como se muestra en la Figura 6.21.

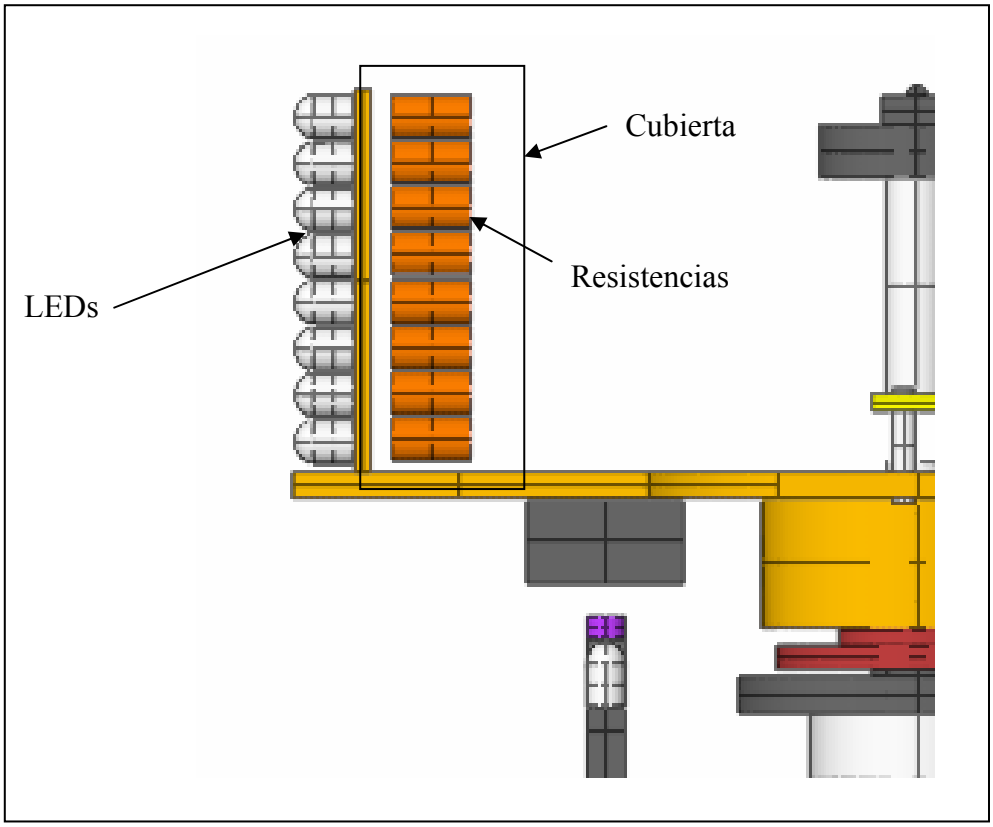


Figura 6.21. LEDs con cubierta

Ensamblaje del Display Rotativo.

El prototipo debe ser ensamblado de la manera más precisa ya que al girar el rotor se pueden producir anomalías como vibraciones, cortes de alimentación, interrupción del funcionamiento de lo sensores IR y principalmente ruido que puede ser molesto para el usuario. Todo esto debe ser considerado al momento de armar el prototipo.

La mejor opción para unir todas las partes del Display Rotativo es utilizar tornillos milimétricos con la longitud que se ajuste mejor a determinadas partes del proyecto, como los mostrados en la Figura 6.22.

Figura 6.22. Tornillos milimétricos utilizados

Primeramente se debe ensamblar es la Base Estática del Display Rotativo. La parte central de la base debe sostener el núcleo del motor sin escobillas que hace girar al circuito del Rotor junto con los LEDs RGB y el circuito oscilador de 38 KHz con el Timer 555 y el LED IR, como se muestra en las Figura 6.23. y 6.24.

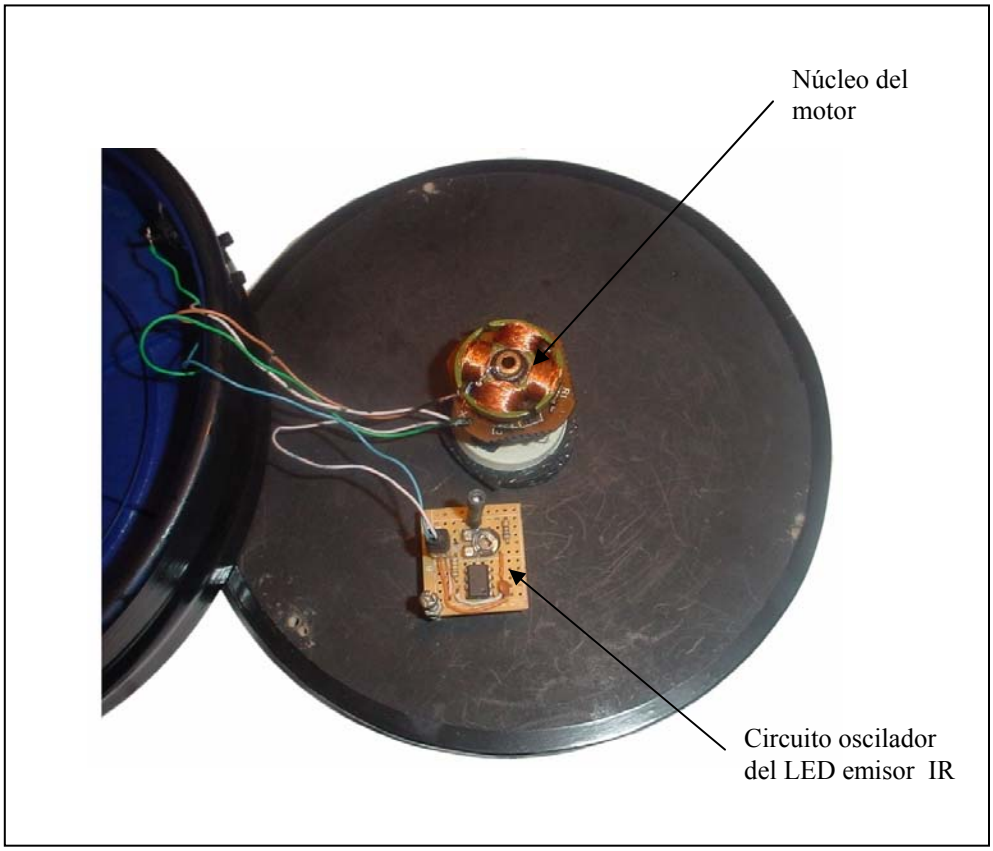


Figura 6.23. Base Estática ensamblada

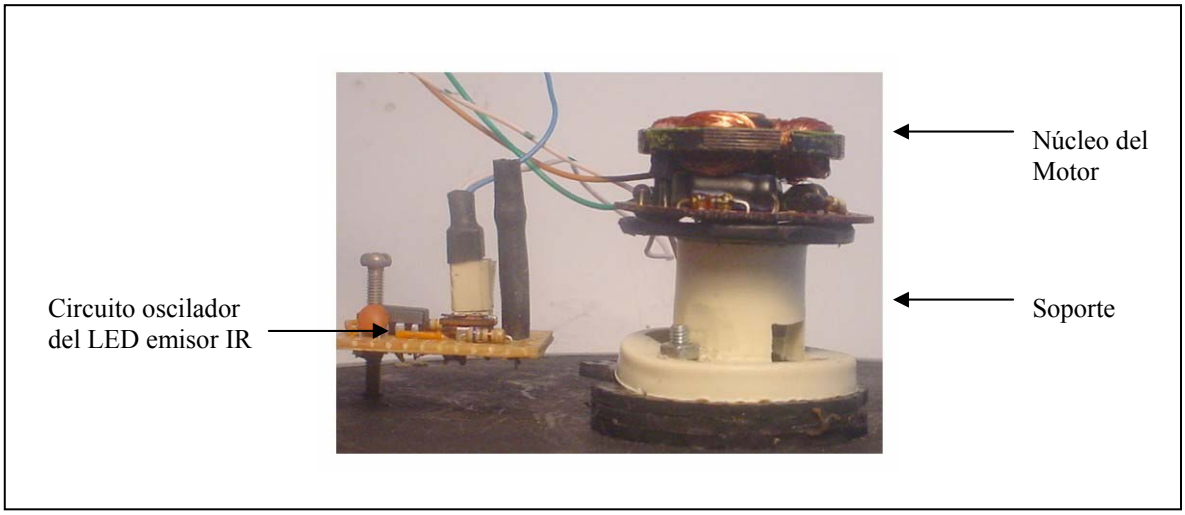


Figura 6.24. Vista lateral de la Base Estática ensamblada

Sobre la base debe ir una cubierta con un agujero en el centro para permite colocar el rotor posteriormente y que éste gire libremente. Se debe hacer un pequeño agujero a la cubierta de la base para que el LED IR pueda alinearse con el Detector IR de Movimiento del Rotor, como se muestra en la Figura 6.25.

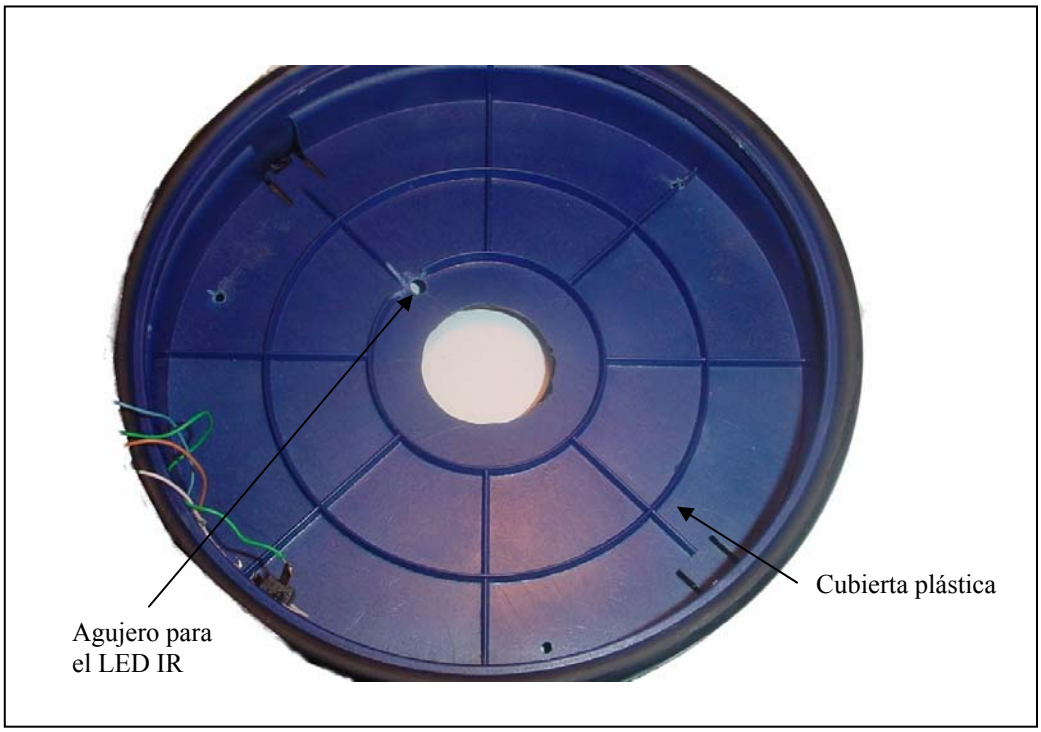


Figura 6.25. Vista inferior de la cubierta de la Base Estática ensamblada

Sobre el agujero para el LED IR se coloca una pieza circular de plástico para señales IR con su respectivo sujetador, como se muestra en la Figura 6.26.

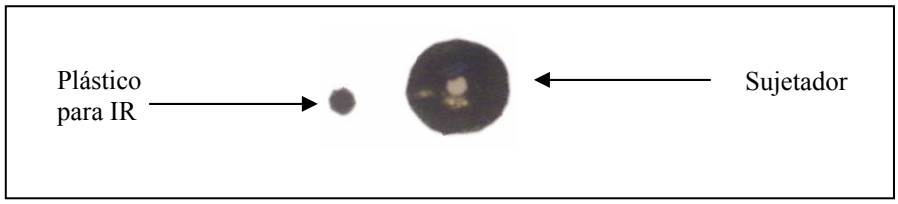


Figura 6.26. Plástico para señales IR y sujetador

Además, en la cubierta debe ir adherido el conector a la fuente de 12 VDC que alimenta al motor sin escobillas y al Circuito del Rotor. También se deben colocar dos interruptores, uno que encienda todo el Display Rotativo y otro que encienda el motor siempre y cuando el primer interruptor esté cerrado. Estas conexiones se muestran en la Figura 6.27.

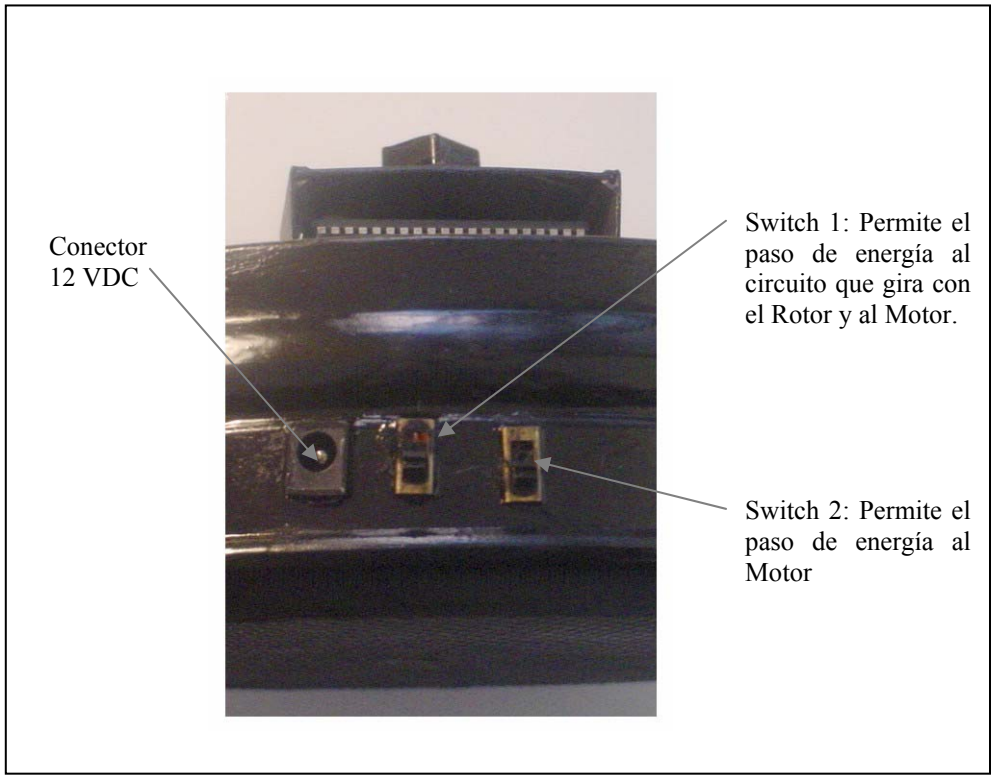


Figura 6.27. Conexiones de la Base Estática

Una vez lista la base se prosigue a construir el rotor, como se muestra en la Figura 6.28. El Circuito del Rotor se coloca sobre la Base del Rotor por medio de tornillos, dejando medio centímetro entre ambos para pasar cables después. Se debe tener en cuenta que el circuito debe ir hacia la parte trasera de la Base del Rotor dejando uno de sus extremos laterales sobre el eje de giro del rotor para colocar luego la Interfase IR. Luego se coloca el Sensor de movimiento debajo de la Base del Rotor apuntando hacia el LED IR que se encuentra en la base. Finalmente se conectan los cables respectivos desde el Sensor de Movimiento hacia el Circuito del Rotor.

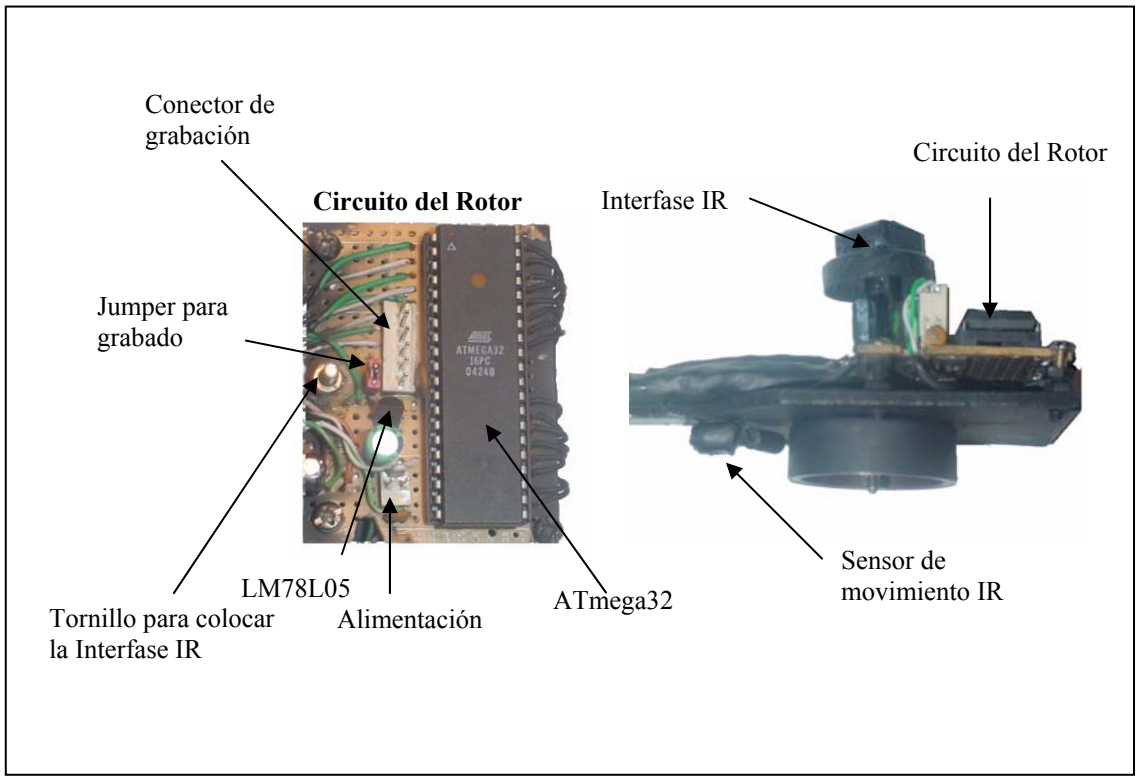


Figura 6.28. Ensamblaje del Circuito del Rotor

Luego de las modificaciones realizadas al motor sin escobillas para que pueda brindar alimentación al Circuito del Rotor, se suelda un cable al Terminal 1 del eje y otro cable al Terminal 2 del eje, como se muestra en la Figura 6.29. El otro extremo de cada cable va a un conector que a su vez alimentará al LM78L05 del Circuito del Rotor, recordando siempre mantener la polaridad correcta.

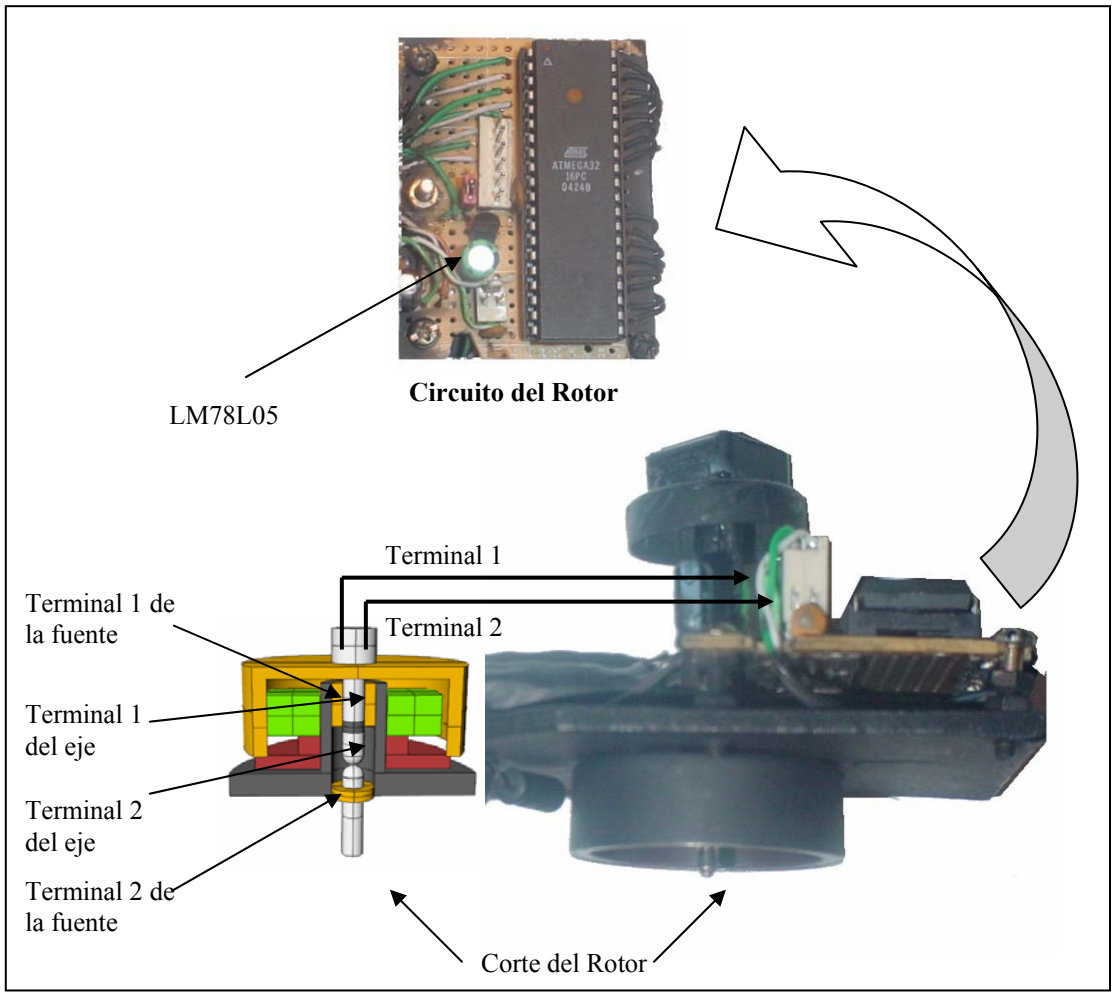


Figura 6.29. Alimentación del Circuito del Rotor

La alimentación del Motor y del circuito que gira con el rotor puede ser obtenida de una fuente de 12 VDC (adaptador), de mínimo 500 mA como el que se muestra en la Figura 6.30.

Figura 6.30. Adaptador 12 VDC

Para una mejor presentación y por seguridad se recomienda colocar una cubierta sobre el Circuito del Rotor, como la mostrada en la Figura 6.31.

Figura 6.31. Cubierta del Circuito del Rotor

Posteriormente se coloca la columna de 8 LEDs en el extremo del eje horizontal de la Base del Rotor, con sus resistencias y conexiones que unen cada LED con VCC y con los puertos respectivos. Al terminar de hacer las conexiones se coloca una cubierta sobre los cables y resistencias para ocultarlos. En la Figura 6.32. se muestra la columna de 8 LEDs.

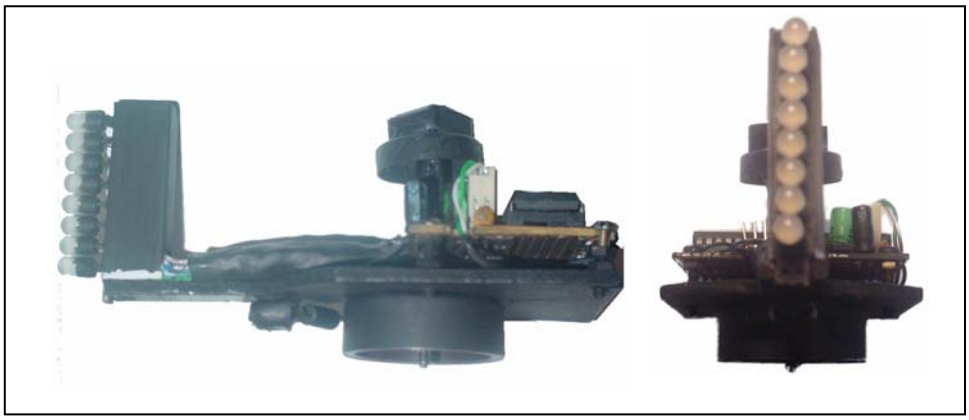


Figura 6.32. Columna de 8 LEDs

Finalmente se coloca el Rotor sobre la Base Estática, como se muestra en la Figura 6.33. El Rotor debe ser capaz de girar libremente y se debe asegurar que el Sensor de Movimiento pase sobre el LED emisor IR en cada vuelta. Se puede utilizar una cubierta semiesférica plástica obscura semitransparente, colocada sobre la Base Estática, para proteger el Rotor y para dar una mejor apariencia al prototipo.

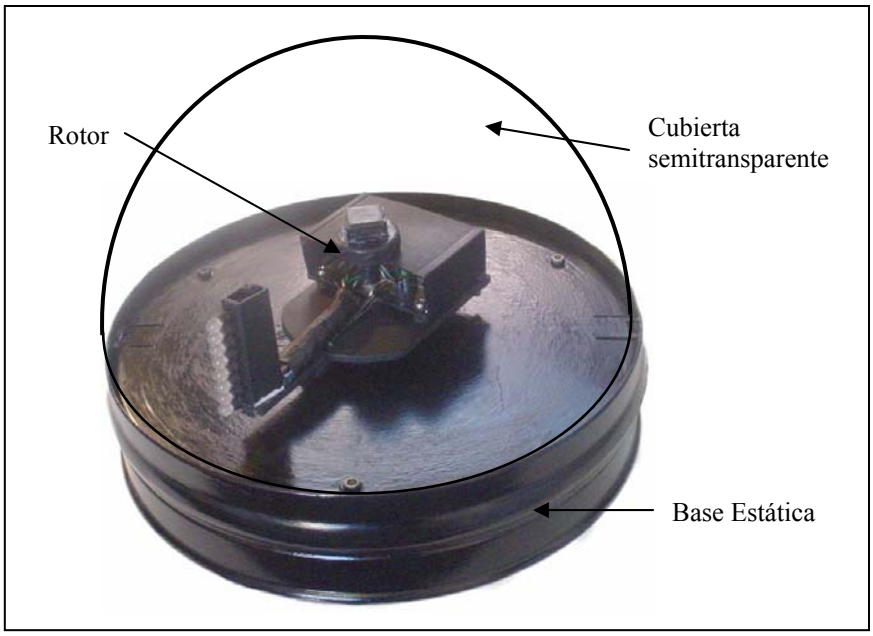


Figura 6.33. Display Rotativo terminado

Para dar la impresión de que los Mensajes o imágenes están girando en el aire es recomendable pintar todo el Display Rotativo de negro teniendo cuidado con las conexiones eléctricas y los Detectores IR. Además, no se deben utilizar pintura metálica sobre la superficie superior de la cubierta de la Base Estática; ya que, podría interferir con el Sensor de Movimiento.

Algoritmo del programa del Display Rotativo.

El funcionamiento planteado para el programa del Display Rotativo es el siguiente:

Todo el manejo del Display Rotativo se realizará a través de un Control Remoto Universal de Televisión Sony, excepto el encendido del Display Rotativo y del Motor que lo hace girar ya que esto último se realiza a través de interruptores conectados en el mismo aparato. Solo se utilizarán los botones estándar del Control Remoto para Televisión, es decir, *1, 2, 3, 4, 5, 6, 7, 8, 9, 0, sleep, display, enter, mute, recall, vol+, vol-, ch+* y *ch-*. No se usará el botón *Power*.

Para poder acceder al Display Rotativo se deberá presionar primero el botón *sleep* para que un cursor aparezca en la línea de Mensaje. Al presionar nuevamente el botón *sleep* el acceso al Display Rotativo se debe denegar y el cursor debe desaparecer. Esta medida de precaución, sumada al hecho de que no se usa el botón *Power,* se debe a que podría haber un televisor en el área y el usuario podría querer manejar cada uno individualmente.

En el caso de que haya un televisor en el lugar donde el Display Rotativo se encuentra, para manejar el Display Rotativo, el usuario tendrá primero que apagar el televisor momentáneamente. Luego si se quiere manejar el televisor, el usuario tendrá que presionar primero *sleep* para impedir el acceso al Display Rotativo y luego encender el televisor.

El Display Rotativo será capaz de mostrar 4 Mensajes independientes de 12 caracteres cada uno (cada Mensaje se considera una línea de Mensaje). Además de letras y números tendrá la capacidad de mostrar todos los símbolos ASCII más algunos símbolos extras, ya sean gestuales o llamativos.

El ingreso de Mensajes se realizará del mismo modo que se ingresan Mensajes en el teclado de un teléfono celular con la diferencia de que el movimiento del cursor hacia la posición siguiente no será automático sino controlado por los botones *ch+* y ch-, siendo utilizado ch+ para mover el cursor a la derecha y *ch-* para moverlo a la izquierda. Los símbolos más comunes y el número 1 se ingresarán a través del botón *1*, los demás botones desde el *2* hasta el *9* ingresarán los números y letras igual que un teclado telefónico. El número 0 y los espacios se ingresarán a través del botón *0*.

Los símbolos ASCII menos comunes y los símbolos gestuales y llamativos se escogerán de un menú que se activará con el botón *recall*, la selección se realizará a través de los botones *vol+* y *vol-*. Una vez que se tenga el símbolo deseado se podrá desactivar esta opción presionando *recall* nuevamente.

Además se podrá escoger el color de cada caracter entre las opciones rojo, verde y azul presionando repetidamente el botón *mute.* Los Mensajes no se perderán si el Display Rotativo se apaga o desconecta.

Un Reloj mostrará la hora actual en el formato HH:MM:SS de 24 horas y no en el de 12, de la manera más precisa posible. Al momento de configurar la hora actual, no se podrá ingresar horas ni minutos incorrectos, es decir, horas superiores a 23 ni minutos superiores a 59. En caso de ingresar un número incorrecto, el Display Rotativo deberá corregirlo automáticamente sustituyéndolo por 23 en el caso de las horas y por 59 en el caso de los minutos. Los segundos no serán configurados. La hora actual se perderá si el Display Rotativo se apaga o se desconecta.

Además de los Mensajes y el Reloj, el Display Rotativo será capaz de mostrar una Imagen Prediseñada y pregrabada de 8 x 90 píxeles, donde cada píxel podrá tener los colores negro (ausencia de color), rojo, verde o azul. La única manera de cargar una imagen nueva es mediante la reprogramación del microcontrolador.

Para escoger entre los Mensajes, el Reloj y la Imagen Prediseñada, el ususario podrá moverse a través de ellos utilizando los botones *vol+* y *vol-* siempre y cuando los símbolos no estén activados.

Entre los Modos de trabajo del Display Rotativo estarán: Normal, Deslizamiento y Alterno. En Modo Normal, el usuario tendrá la capacidad de ingresar y modificar los Mensajes así como el Reloj, y pudiendo mostrarse solo uno de ellos a la vez de manera estática.

En Modo Deslizamiento, el Mensaje o la hora actual se mostrará girando ya sea en sentido horario o antihorario y de manera rápida o lenta, lo cual podrá ser controlado a través de los botones *vol+* y *vol-. Vol+* controlará la velocidad del giro en sentido horario y *vol-* la velocidad de giro en el sentido antihorario.

En Modo Alterno, el Display Rotativo mostrará los cuatro Mensajes guardados en memoria, uno tras otro de manera continua. El Display Rotativo podrá mezclar los Modos Deslizamiento y Alterno si así el usuario lo desea; sin embargo, siendo que uno o ambos Modos están activados no se podrá modificar ni los Mensajes ni el Reloj. El Modo Deslizamiento se podrá activar o desactivar presionando el botón *enter* y el Modo Alterno se podrá activar o desactivar presionando el botón *display*.

Todas las funciones del Display Rotativo estarán validadas para evitar conflictos o errores en el ingreso de datos. Para ayudar al usuario a manejar mejor el Display Rotativo, se debe tener un Manual del Usuario disponible que explique el funcionamiento de este dispositivo de una manera simple y clara.

Programa del Display Rotativo.

Para un mejor entendimiento y funcionamiento del programa se lo ha dividido en funciones e interrupciones[8]. Se debe tener cuenta que el microcontrolador debe realizar dos actividades todo el tiempo, recibir comandos desde el Control Remoto y mostrar Mensajes e imágenes a través de los LEDs RGB. Si el microcontrolador se dedica a una actividad primero y luego al completarla se dedica a la otra, es posible que se pierda información o que ésta no se muestre correctamente, por lo que es mejor trabajar con interrupciones para la recepción de comandos desde el Control Remoto.

A su vez, cada actividad es realizada no solo por interrupciones sino por funciones que se llaman entre si, lo cual simplifica el programa y lo hace más entendible. De igual manera el programa principal se apoya en un programa secundario para el manejo de caracteres. También se emplean constantes definidas al principio del programa para permitir realizar cambios en el programa sin tener que hacer muchas modificaciones.

[8] Se puede revisar el capítulo anterior CODIFICACIÓN SIRC para comprender mejor como se llegó a la elaboración de algunas de estas funciones e interrupciones.

A continuación, en la Figura 6.34. se presenta un esquema que muestra la manera en que el programa principal del proyecto RAINBOW llamado Rainbow.c, junto con el archivo de código auxiliar Charlut.c están estructurados.

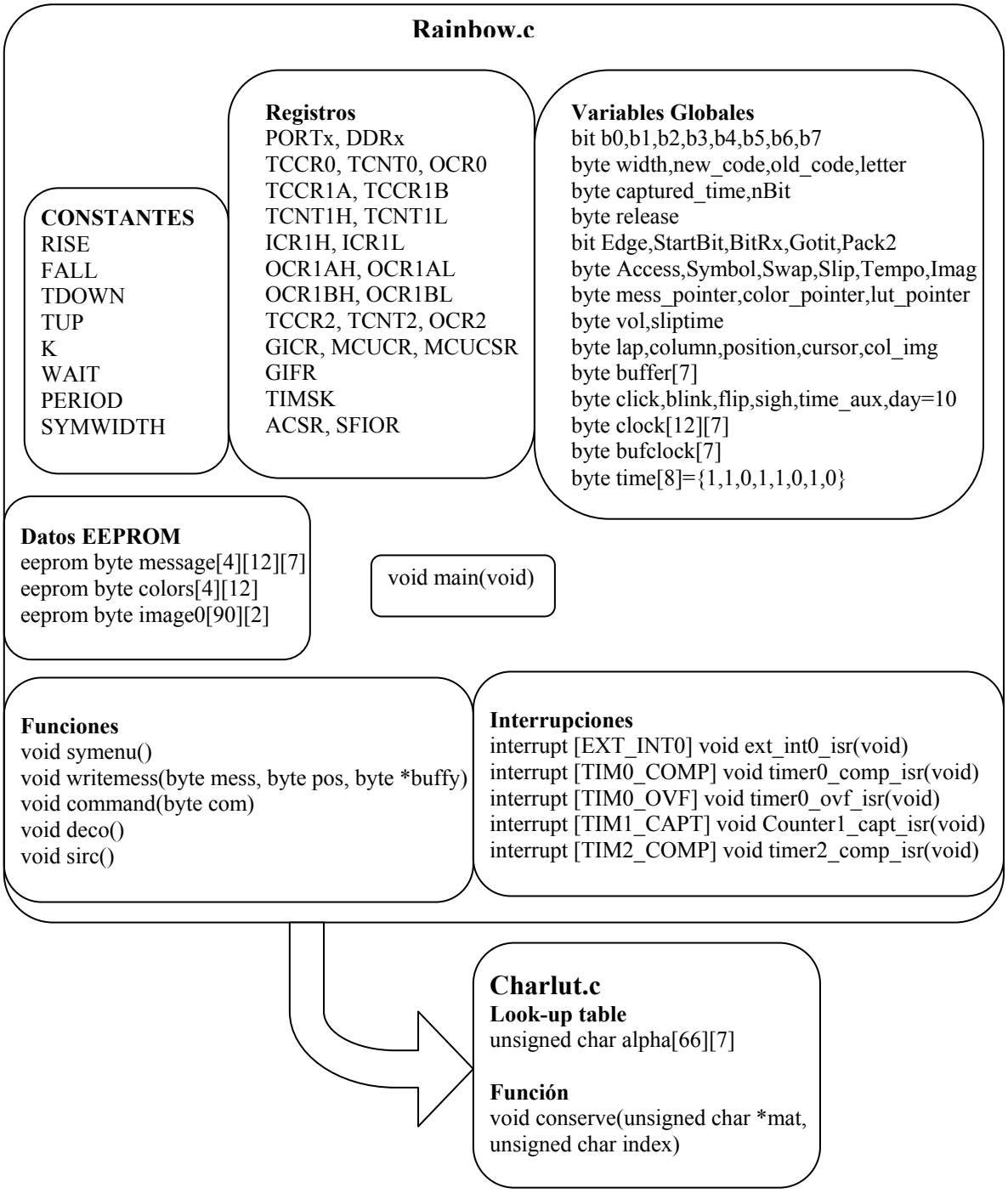


Figura 6.34. Esquema del Programa del Display Rotativo

6.5.1. Consideraciones iniciales.

Antes de realizar el programa se debe tener en cuenta los siguientes puntos:

- Frecuencia de trabajo
- Presentación de Mensajes
- Presentación de la Imagen Prediseñada
- Digitalización de la Imagen Prediseñada
- Tipos de variables

Frecuencia de trabajo.

En primer lugar se debe configurar el ATmega32 para que utilice el Reloj interno a 2 MHz. En programas anteriores se utilizó un Reloj interno de 8 MHz para utilizar el microcontrolador a toda su capacidad. Sin embargo, ahora se deben trabajar con lapsos de tiempo mayores; como es el caso del tiempo que el Display Rotativo demora en dar una vuelta completa, que es 0,09216 s aproximadamente[9]. Un prescaler de 1024 (su máximo valor) no alcanza para medir dicho lapso de tiempo a una frecuencia de 8 MHz.

La solución más simple es hacer que el microcontrolador sea más lento así que se debe bajar su frecuencia de trabajo a 2 MHz. Debido a que 2 MHz es una frecuencia de trabajo que aún se puede considerar rápida, en relación a las mediciones de tiempo con respecto a la transmisión de comandos desde el Control Remoto IR, el cambio de frecuencia no alterará esta parte del programa ya realizada anteriormente, lo único que se debe hacer es dividir todos los valores obtenidos para 4. El cambio de frecuencia de trabajo se logra modificando los Bits de Configuración y Seguridad como ya se explico en las secciones: 2.4. Reloj del Sistema y 4.3.6. Bits de Configuración y de Seguridad.

[9] Medición realizada con el Timer0, donde el contador al llegar aproximadamente a 180 mantenía el Mensaje mostrado por el Display Rotativo en el mismo sitio. Debido a que se cambio de un reloj interno de 8 MHz a 2 Mhz, se debe dividir el valor de 2MHz para el valor del prescaler del Timer0 que es 1024 y luego dividir 200 para dicho valor: 180/(2MHz/1024) = 0,09216 s.

Luego, utilizando CodeWizardAVR se crea el Proyecto RAINBOW dentro de su carpeta respectiva. Entre las opciones de la ventana de CodeWizardAVR, se debe establecer el tipo de microcontrolador como un ATmega32 a 2 MHz. También se debe escoger que el Puerto D es de entrada con resistencia de Pull-up y los Puerto A, B y C como de salida, con un valor inicial en alto para todos los pines. Es recomendable generar el archivo *Rainbow.c* sin más modificaciones, creando e inicializando las interrupciones manualmente.

Presentación de Mensajes.

Cada caracter que se vaya a presentar en los LEDs tendrá un área de 8 x 7 píxeles, así por ejemplo, para representar la letra “A” se encenderán los respectivos píxeles de la manera mostrada en la Figura 6.35.

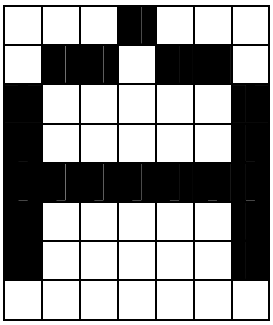


Figura 6.35. Matriz de la letra “A”

Se deja la última fila libre para el cursor, el cual indica la posición actual respecto a la línea de Mensaje. Cada símbolo mostrado en el Display Rotativo se muestra columna por columna conforme los LEDs se mueven de izquierda a derecha. Puesto que cada columna contiene 8 píxeles, como se muestra en la Figura 6.36., se le puede indicar al puerto del ATmega32 que controle los LEDs que están girando y que encienda solo ciertos LEDS para formar el caracter deseado. Así por ejemplo para formar la primera columna de la letra “A” solo se deben encender los LEDs 2,3,4,5 y 6. Al final de cada caracter se debe indicar al puerto que apague todos los LEDs por el mismo lapso que dura una columna del caracter para así tener un espacio entre caracter y caracter.

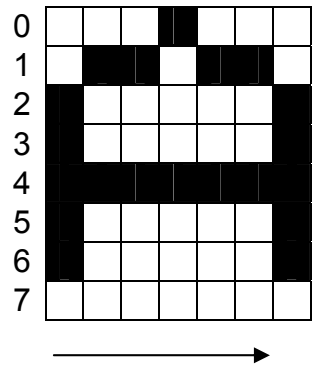


Figura 6.36. Formación de la matriz de la letra "A"

Además, para que los LEDs tomen su corriente desde la fuente y no desde el microcontrolador es mejor utilizar lógica negativa para su encendido. Por lo tanto, para encender la primera columna de la letra "A" el puerto que controla la columna de LEDs debería tener un valor en binario de 0b10000011. De esta manera, cada caracter requiere de 7 bytes de memoria para guardar la información necesaria para su representación a través de los LEDs del Display Rotativo.

Considerando que guardar todos estos caracteres ocupa bastantes líneas de código, es mejor guardarlos en otro archivo auxiliar al que se denominará Charlut.c, el cual se describirá posteriormente.

Finalmente, para presentar cada letra se debe definir su tipo de color. Los Puertos A, B y C del ATmega32 controlan los colores Rojo, Verde y Azul de los LEDs RGB respectivamente. Al escoger uno de dichos Puertos para formar la correspondiente letra, se determina su color.

Presentación de la Imagen Prediseñada.

Las imágenes mostradas a través de los LEDs del Display Rotativo se manejan de una manera algo distinta a los caracteres. Aunque el tiempo que la columna de LEDs se mantiene encendida para formar una columna de la imagen se mantiene igual que en el caso de los caracteres, en esta ocasión ya no debe haber un espacio entre caracter y caracter. Considerando que la imagen puede ser

mostrada a lo ancho de una media circunferencia de giro del Display Rotativo, lo cual equivale a noventa columnas, se puede decir que la imagen tendrá un área de 8 x 90 píxeles.

A diferencia de las letras, cuyo color se determina globalmente para cada una de ellas, es decir, toda la letra tiene un solo color; en una imagen, debe haber la posibilidad de determinar el color de cada píxel de manera individual entre las opciones negro (ausencia de luz), rojo, verde y azul. Siendo cuatro posibilidades de color por píxel se puede utilizar la lógica mostrada en la Tabla 6.2.:

Identificador	Color	Código
N	Negro	11
R	Rojo	10
G	Verde	01
B	Azul	00

Tabla 6.2. Lógica de colores para Imágenes Prediseñadas

Para mantener un mismo orden con respecto a la presentación de Mensajes se debe optar por un código de 11 para el negro (ausencia de luz) ya que en lógica negativa esto indica al puerto que debe apagar el respectivo LED, en vez de 00 como se podría pensar en un principio. De esta manera si se quisiera una columna de 8 píxeles con los colores R, G, G, N, B, N, R y G siendo el primer píxel el más alto de la columna y menos significativo tendríamos la Tabla 6.3.:

Orden	Identificador	Byte1	Byte0
0	R	1	0
1	G	0	1
2	G	0	1
3	N	1	1
4	B	0	0
5	N	1	1
6	R	1	0
7	G	0	1

Tabla 6.3. Lógica de colores de una columna de 8 píxeles

Como se puede ver en la Tabla 6.3., se forman dos *Bytes* de información por cada columna de píxeles. Mediante estos dos *Bytes* se le puede indicar al puerto respectivo que maneja cada color cuales LEDs deben encender y cuales no. Así

los puertos que manejan los colores: rojo (A), verde (B) y azul (C), deben considerar la Tabla 6.4.:

Orden	Identificador	Byte1	Byte0	Puerto rojo (A)	Puerto verde (B)	Puerto azul (C)
0	R	1	0	0	1	1
1	G	0	1	1	0	1
2	G	0	1	1	0	1
3	N	1	1	1	1	1
4	B	0	0	1	1	0
5	N	1	1	1	1	1
6	R	1	0	0	1	1
7	G	0	1	1	0	1

Tabla 6.4. Lógica de colores de una columna de 8 píxeles en los Puertos

Para lograr que cada puerto encienda solo determinados LEDs dando un efecto multicolor a cada columna de pixeles es necesario operar los bytes Byte1 y Byte2 para que den el resultado requerido para cada puerto. De esta manera la operación lógica que controla cada puerto quedaría de la siguiente manera:

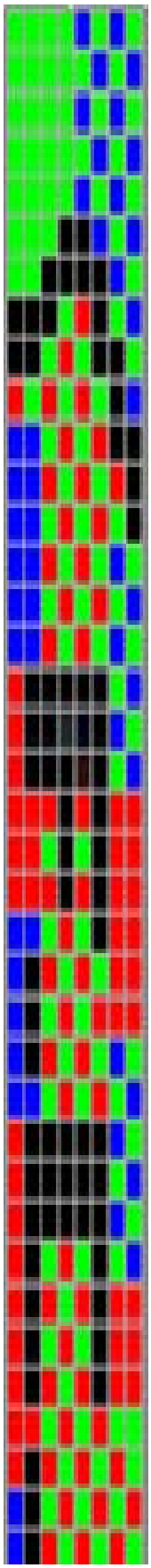
Puerto rojo = $\sim$Byte1 $\vee$ Byte0
Puerto verde = Byte1 $\vee$ $\sim$Byte0
Puerto azul = Byte1 $\vee$ Byte0

Si se guarda el valor de cada píxel en un arreglo de 90 x 2 llamado *image0[ ][ ]*, donde la variable *col_img* indica la columna actual, las líneas de código en lenguaje C serían:

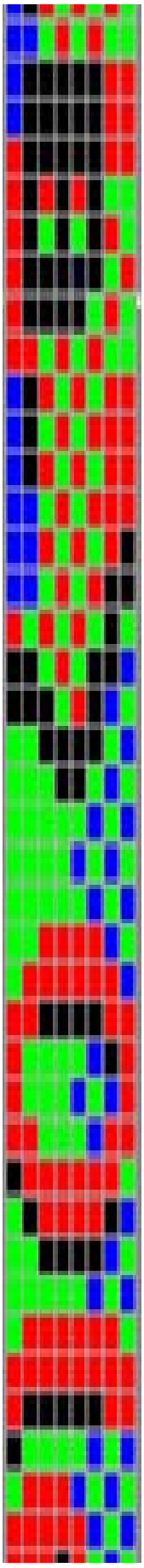
```
PORTA=~image0[col_img][1]|image0[col_img][0];
PORTB=image0[col_img][1]|~image0[col_img][0];
PORTC=image0[col_img][1]|image0[col_img][0];
```

Digitalización de la Imagen Prediseñada.

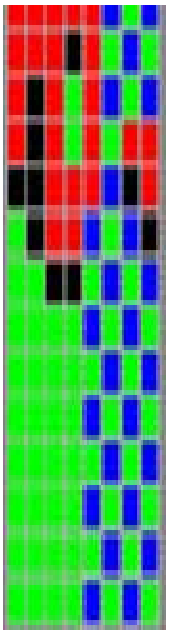
Utilizando una hoja de cálculo de Excel se puede codificar una imagen columna por columna según la lógica mencionada en la sección anterior, como se muestra en la Tabla 6.5.



		n	image0[n][1]	image0[n][0]
0 0 0 0 1 1 1 1	0 0 0 0 0 1 0 1	1	0	245
0 0 0 0 1 1 1 1	0 0 0 0 1 0 1 0	2	0	250
0 0 0 0 1 1 1 1	0 0 0 0 0 1 0 1	3	0	245
0 0 0 0 1 1 1 1	0 0 0 0 1 0 1 0	4	0	250
0 0 0 0 1 1 1 1	0 0 0 0 0 1 0 1	5	0	245
0 0 0 1 1 1 1 1	1 0 0 0 1 0 1 0	6	24	250
0 0 1 1 1 1 1 1	1 1 0 0 1 1 0 1	7	60	253
1 1 1 0 1 1 1 1	1 1 0 0 0 1 1 0	8	236	246
1 1 0 1 1 1 1 0	0 1 1 0 1 1 1 1	9	214	239
1 0 1 0 0 1 0 1	1 0 1 0 0 1 1 0	10	170	86
0 0 0 1 0 0 1 0	0 1 1 1 1 0 1 1	11	23	43
0 0 1 0 0 0 0 1	1 0 1 1 0 1 0 1	12	43	21
0 0 0 1 0 0 1 0	0 1 0 1 1 0 1 1	13	21	43
0 0 1 0 0 0 0 1	1 0 0 0 0 1 0 1	14	40	21
0 0 0 1 0 0 1 0	0 1 0 0 1 0 1 0	15	20	42
0 0 1 0 0 0 0 1	1 0 0 0 0 1 0 1	16	40	21
1 1 1 1 0 1 1 1	1 1 0 0 1 1 1 0	17	252	126
1 1 1 1 0 1 1 1	1 1 0 0 1 1 0 1	18	252	125
1 1 1 1 0 1 1 1	1 1 0 0 1 1 1 0	19	252	126
1 1 1 1 0 0 0 1	1 1 1 1 0 1 0 0	20	255	20
1 1 0 1 0 0 1 1	0 1 1 1 1 1 0 0	21	215	60
1 1 1 1 0 0 0 1	1 1 1 1 0 1 0 0	22	255	20
0 0 0 1 0 0 1 0	0 1 1 1 1 1 0 0	23	23	44
0 1 1 0 0 1 0 1	1 0 1 1 0 1 0 0	24	107	84
0 1 0 1 0 1 1 0	0 1 1 1 1 0 0 0	25	87	104
0 1 1 0 0 1 0 1	1 0 0 0 0 1 0 1	26	104	85
0 0 0 1 0 0 1 0	0 1 0 0 1 0 1 0	27	20	42
1 1 1 1 0 1 1 1	1 1 0 0 1 1 0 1	28	252	125
1 1 1 1 0 1 1 1	1 1 0 0 1 1 1 0	29	252	126
1 1 1 1 0 1 1 1	1 1 0 0 1 1 0 1	30	252	125
1 1 0 1 0 1 1 0	0 1 0 0 1 1 1 0	31	212	110
1 1 1 0 0 1 0 1	1 1 1 1 0 1 0 0	32	239	84
1 1 0 1 0 1 1 0	0 1 1 1 1 1 0 0	33	215	108
1 1 1 0 0 1 0 1	1 1 1 1 0 1 0 0	34	239	84
1 1 0 1 0 0 1 0	0 1 0 0 1 0 1 1	35	212	43
1 1 1 0 0 1 0 1	1 0 1 0 0 1 0 1	36	234	85
0 1 0 1 0 1 1 0	0 1 0 1 1 0 1 0	37	85	106
0 1 1 0 0 1 0 1	1 0 1 0 0 1 0 1	38	106	85



0 0 0 1 0 0 1 0	0 1 0 0 1 0 1 1	39	20	43
0 1 1 1 0 1 1 1	1 1 1 1 1 1 0 0	40	127	124
0 1 1 1 0 1 1 1	1 1 1 1 1 1 0 0	41	127	124
1 1 1 1 0 1 1 1	1 1 1 1 1 1 0 0	42	255	124
1 1 1 1 0 1 0 1	1 1 0 0 0 1 1 1	43	252	87
1 1 0 1 0 1 1 1	0 1 1 0 1 1 0 1	44	214	125
1 1 1 1 0 1 1 1	1 1 0 1 1 1 1 0	45	253	126
1 1 1 1 0 1 1 1	1 0 1 0 1 1 0 1	46	250	125
1 1 0 1 0 0 1 0	0 1 0 0 1 0 1 1	47	212	43
0 1 1 0 0 1 0 1	1 0 1 1 0 1 0 0	48	107	84
0 1 0 1 0 1 1 0	0 1 1 1 1 0 0 0	49	87	104
0 1 1 0 0 1 0 1	1 0 1 1 0 1 0 0	50	107	84
0 0 0 1 0 0 1 0	0 1 1 1 1 0 0 0	51	23	40
0 0 1 0 0 0 0 1	1 0 1 1 0 1 0 1	52	43	21
0 0 0 1 0 0 1 0	0 1 1 1 1 0 1 1	53	23	43
1 0 1 0 0 1 0 1	1 0 1 0 0 1 1 1	54	170	87
1 1 0 1 1 1 1 0	0 1 1 0 1 1 1 0	55	214	238
1 1 1 0 1 1 1 1	1 1 0 0 0 1 0 1	56	236	245
0 0 1 1 0 1 1 1	1 1 0 0 1 1 1 0	57	60	254
0 0 0 1 0 1 1 1	1 0 0 0 1 1 0 1	58	24	253
0 0 0 0 0 1 1 1	0 0 0 0 1 0 1 0	59	0	250
0 0 0 0 0 1 1 1	0 0 0 0 0 1 0 1	60	0	245
0 0 0 0 0 1 1 1	0 0 0 0 1 0 1 0	61	0	250
0 0 1 1 0 1 1 0	1 1 0 0 0 0 0 1	62	60	193
0 1 1 1 0 1 0 0	1 1 1 0 0 0 0 0	63	126	128
1 1 1 1 0 0 1 1	1 1 1 1 1 1 0 0	64	255	60
1 0 0 0 0 1 1 1	0 0 1 1 1 0 1 0	65	131	122
1 0 0 0 0 1 1 1	0 0 0 1 0 1 0 0	66	129	116
1 1 0 0 0 0 1 1	0 0 1 1 1 0 0 0	67	195	56
1 1 1 1 1 0 0 0	1 1 1 0 0 0 0 1	68	254	129
0 1 1 1 0 1 1 0	1 1 1 0 0 0 1 0	69	126	194
0 0 1 1 0 1 1 1	1 1 0 0 1 1 0 1	70	60	253
0 0 0 0 0 1 1 1	0 0 0 0 1 0 1 0	71	0	250
0 1 1 1 0 1 0 0	1 1 1 0 0 0 0 1	72	126	129
1 1 1 1 0 0 0 0	1 1 1 1 0 0 0 0	73	255	0
0 1 1 1 0 1 1 1	1 1 1 1 1 1 0 0	74	127	252
1 0 0 0 0 1 1 1	0 0 0 0 1 0 1 0	75	128	250
0 1 1 1 0 1 0 0	0 0 0 0 0 1 0 1	76	112	133
1 1 1 1 1 0 0 0	1 0 0 0 0 1 0 0	77	248	2



11000111 10010001	78	248	21
11010110 10010010	79	232	82
11010110 10101010	80	235	84
11110011 10011010	81	251	194
01100011 00010111	82	113	197
01110011 10100010	83	48	250
01111001 00010001	84	0	245
01111001 10100010	85	0	250
01111001 00010001	86	0	245
01111001 10100010	87	0	250
01111001 00010001	88	0	245
01111001 10100010	89	0	250
01111001 00010001	90	0	245

Tabla 6.5. Ingreso de la Imagen Prediseñada

En la Tabla 6.5. cada uno de los 8 píxeles que forman una columna son codificados según la Tabla 6.2. Los dos códigos binarios de 8 bits formados por cada columna de la Imagen Prediseñada son transformados a 2 *Bytes* escritos de manera decimal: image0[n][1] e image[n][0]. Finalmente concatenando los códigos de 2 *Bytes* de las 90 columnas de la Imagen Prediseñada, se crea el arreglo que contendrá la información de la imagen que se grabará en la EEPROM del microcontrolador, como se muestra en la Tabla 6.6.:

```
{0,245},{0,250},{0,245},{0,250},{0,245},{24,250},{60,253},{236,246},{214,239},{170,86}
{23,43},{43,21},{21,43},{40,21},{20,42},{40,21},{252,126},{252,125},{252,126},{255,20}
{215,60},{255,20},{23,44},{107,84},{87,104},{104,85},{20,42},{252,125},{252,126},{252,125}
{212,110},{239,84},{215,108},{239,84},{212,43},{234,85},{85,106},{106,85},{20,43},{127,124}
{127,124},{255,124},{252,87},{214,125},{253,126},{250,125},{212,43},{107,84},{87,104},{107,84}
{23,40},{43,21},{23,43},{170,87},{214,238},{236,245},{60,254},{24,253},{0,250},{0,245}
{0,250},{60,193},{126,128},{255,60},{131,122},{129,116},{195,56},{254,129},{126,194},{60,253}
{0,250},{126,129},{255,0},{127,252},{128,250},{112,133},{248,2},{248,21},{232,82},{235,84}
{251,194},{113,197},{48,250},{0,245},{0,250},{0,245},{0,250},{0,245},{0,250},{0,245}
```

Tabla 6.6. Código de la Imagen Prediseñada ingresado al microcontrolador

Tipos de variables utilizadas.

La gran mayoría de variables utilizadas en el programa son del tipo *unsigned char* que equivalen a un *Byte*; por lo que para no tener que escribir esta definición de tipo de variable que es muy larga y tan solo escribir *Byte*, se puede utilizar la siguiente línea de comando:

```
typedef unsigned char byte;
```

6.5.2. Constantes Pre-definidas.

Las constantes pre-definidas son de gran ayuda al momento de hacer modificaciones en el programa debido a que evitan tener que cambiar valores numéricos uno por uno. Las constantes son definidas al principio del programa *Rainbow.c* a través de la instrucción *#define* como se muestra a continuación:

```
#define RISE 0xC4
#define FALL 0x84
#define TDOWN 5 // 21 tiempo en alto
#define TUP 5 // 19 tiempo en bajo
#define K 1 // 3 factor de tolerancia
#define WAIT 1 // 4 factor de espera 0,032768 seg > 0,023296 seg (duración de un paquete)
#define PERIOD 170 // 180 sharp
#define SYMWIDTH 0x08 // ancho de cada letra
```

La función de cada constante pre-definida se explica a continuación:

RISE **FALL**	Trabajan con el Timer1. Además de inicializarlo también configuran a la Interrupción por Captura de Dato para capturar flancos ascendentes (RISE) o descendentes (FALL).
TDOWN **TUP**	Guardan la duración del ancho de pulso en bajo (TDOWN) y en alto (TUP) en la codificación SIRC, para las mediciones de tiempo realizadas en la Interrupción por Captura de Dato del Timer1.

K Es un factor de tolerancia que hace más flexible las mediciones de tiempo realizadas en la Interrupción por Captura de Dato del Timer1.

WAIT Determina el tiempo que el microcontrolador debe esperar para asegurarse de que el botón presionado en el Control Remoto ha sido soltado para aceptar un nuevo comando transmitido.

PERIOD Trabaja con el Timer0. Guarda el tiempo inicial de deslizamiento de Mensajes que hace que se muevan rápidamente en sentido antihorario debido a que su valor es menor que el periodo de rotación de los LEDs del Display Rotativo.

SYMWIDTH Determina el tiempo que la columna de LEDs debe estar encendida para formar cada columna de un determinado caracter; lo cual a su vez, influye en el ancho del mismo.

6.5.3. Uso de la EEPROM.

Los Mensajes mostrados en el Display Rotativo deben quedar guardados en memoria, aún cuando el dispositivo no esté encendido o conectado a una fuente de poder. La razón de esto es evitar que el usuario tenga que ingresarlos cada vez que se encienda el aparato. Lo mejor es guardar los Mensajes en la EEPROM del ATmega32.

Se requiere de 7 *Bytes* de memoria para guardar un caracter. Cada Mensaje contendrá 12 caracteres y en total el Display Rotativo tendrá la capacidad de mostrar cuatro Mensajes independientes. Para un mejor manejo de los Mensajes,

se los debe guardar en un arreglo de tres dimensiones 4 x 12 x 7 *Bytes* definido como *message[4][12][7]*.

Además el usuario tendrá la capacidad de escoger el color para cada caracter de cada Mensaje, las opciones son: rojo, verde y azul. Igualmente, dicha elección debe quedar guardada en la EEPROM. Si se considera que se puede guardar el color de cada caracter en un *Byte,* esto da como resultado otro arreglo de 4 x 12 *Bytes* definido como *colors[4][12]*.

La Imagen Prediseñada también debe ser guardada en la EEPROM a fin de que pueda ser alterada o cambiada a través del grabador del PonyProg2000, sin necesidad de volver a grabar todo el programa, ya que este último es un proceso que toma más tiempo de grabación. Puesto que las imágenes requieren de 2 *Bytes* de memoria por columna y en total se muestra 90 columnas, se requerirá un arreglo de 90 x 2 *Bytes* definido como *image0[2][90]*.

En total se utilizará 564 *Bytes* de memoria que alcanzan sin problema en la EEPROM de 1024 *Bytes* del ATmega32. Una variable guardada en la EEPROM se maneja igual que una variable guardada en la Memoria Flash junto con el resto del programa, la única diferencia es que al momento de definirla se utiliza la palabra especial *eeprom*. Además, es recomendable inicializarlas con algún valor caso contrario aparecerá un mensaje de WARNING al momento de compilar.

Debido a que las inicializaciones de las variables *eeprom* utilizadas en el programa *Rainbow.c* son bastante largas, por tratarse de arreglos de *Bytes*, se recomienda verlas en el código del programa completo que se muestra en el Apéndice. A continuación se muestra solo como deben ser definidas.

```
eeprom byte message[4][12][7];
eeprom byte colors[4][12];
eeprom byte image0[90][2];
```

6.5.4. Variables Globales.

Casi todas las variables globales utilizadas en el programa *Rainbow.c* son del tipo *unsigned char,* que ha sido redefinido como tipo *byte* a través de la instrucción *typedef.* Únicamente las variables que guardan los 8 bits de información del comando recibido desde el Control Remoto son del tipo *bit,* ya que sería un desperdicio de memoria darles mayor espacio siendo que solo guardarán un 0 o un 1.

Sería ideal que las variables globales que funcionan como banderas también sean del tipo *bit*, sin embargo el programa *CodeVisionAVR* solo permite crear 16 variables globales tipo *bit* por lo que es mejor solo definir como tipo *bit,* las banderas relacionadas con la comunicación IR.

Además, cabe señalar que el programa *CodeVisionAVR* no permite hacer arreglos con variables tipo *bit,* por lo que se debe definir por separado cada variable que guarde unos de los bits transmitidos desde el Control Remoto.

Las variables globales utilizadas en el programa *Rainbow.c* se pueden clasificar según la función que cumplen, o la parte del programa en que trabajan:

- Variables de comunicación IR
- Variables de presentación de Mensajes e Imagen Prediseñada
- Variables de Reloj

Variables de comunicación IR.

Las variables de comunicación IR se definen de la siguiente manera:

```
bit b0,b1,b2,b3,b4,b5,b6,b7;
byte width,new_code,old_code;
byte captured_time,nBit;
byte release;
bit Edge,StartBit,BitRx,Gotit,Pack2;
```

La función de cada variable se explica a continuación:

b0,b1,…,b7 Cada una de estas variables guarda un bit decodificado a partir del comando actualmente recibido desde el Control Remoto.

width Guarda el valor del ancho del pulso en bajo actualmente medido en la Interrupción por Captura de Datos en el Timer1, a partir del tren de pulsos recibido desde el Control Remoto.

new_code Guarda el número de 8 bits decodificado a partir del comando actualmente recibido desde el Control Remoto.

old_code Guarda el número de 8 bits decodificado a partir del anterior comando recibido desde el Control Remoto.

captured_time Guarda el valor del Registro ICR1L que contiene el tiempo en que ocurrío la captura de datos que activó la Interrupción por Captura de Datos del Timer1.

nBit Indica el orden del bit en que se debe guardar el dato al momento de guardar en las variables b0,b1,…,b7 el comando recibido desde el Control Remoto.

release Guarda el valor actual del contador del Timer1 para comprobar si algún botón del Control Remoto está todavía siendo presionado.

Pack2 Bandera que indica si se tiene el segundo paquete de datos recibido desde el Control Remoto.

Edge Bandera que indica al Timer1 si la próxima Inperrupción por Captura de Datos se activará por flanco ascendente o descendente.

StartBit Bandera que indica si se ha capturado algún STARTBIT recientemente para poder comenzar a decodificar el comando transmitido desde el Control Remoto.

BitRx Bandera que indica si se ha decodificado un 0 o un 1 a partir del pulso recibido actualmente para poder guardar dicho valor en la varible b0,b1,…,b7 respectiva.

Gotit Bandera que indica si se ha decodificado exitosamente un comando enviado desde el Control Remoto y cancela la decodificación de cualquier otro paquete recibido desde el Control Remoto hasta que se suelte el botón que se esté presionando.

Variables de Presentación de Mensajes e Imagen Prediseñada.

Las variables de presentación de Mensajes e Imagen Prediseñada se definen de la siguiente manera:

```
byte Access,Symbol,Swap,Slip,Tempo,Imag;

byte mess_pointer,color_pointer,lut_pointer;
byte vol,sliptime;
byte lap,column,position,cursor,col_img,letter;
byte buffer[7];
```

La función de cada variable se explica a continuación:

Access Bandera que indica si el acceso al Display Rotativo es denegado o permitido, dependiendo si su valor es 0 ó 1 respectivamente. Esta variable está relacionada con el botón *sleep* del Control Remoto.

Symbol Bandera que indica si el acceso a los símbolos ASCII menos conocidos, gestuales o llamativos está denegado o permitido, dependiendo si su valor es 0 ó 1 respectivamente. Esta variable está relacionada con el botón *recall* del Control Remoto.

Swap Bandera que indica si el Modo Alterno está apagado o encendido, dependiendo si su valor es 0 ó 1 respectivamente. Esta variable está relacionada con el botón *display* del Control Remoto.

Slip Bandera que indica si el Modo Deslizamiento está apagado o encendido, dependiendo si su valor es 0 ó 1 respectivamente. Esta variable está relacionada con el botón *enter* del Control Remoto.

Tempo Bandera que indica si se está mostrando el Reloj en los LEDs y por lo tanto si debe manejar los botones del *0* al *9* del Control Remoto como alfanuméricos o solo numéricos, dependiendo si su valor es 0 ó 1 respectivamente. Esta variable es utilizada para configurar la hora actual del Reloj del Display Rotativo.

Imag Bandera que indica si el Display Rotativo debe mostrar Mensajes o la Imagen Prediseñada, dependiendo si su

valor es 0 ó 1 respectivamente.

mess_pointer Apunta hacia uno de los cuatro Mensajes de 12 caracteres que se pueden mostrar en el Display Rotativo. También ayuda a elegir si se quiere mostrar el Reloj o la Imagen Prediseñada.

color_pointer Apunta hacia el color elegido para un determinado caracter que se guardará en la variable respectiva del arreglo *colors[4][12*].

lut_pointer Apunta hacia uno de los caracteres guardados en cada botón del Control Remoto.

vol Actúa a manera de un *selector* que puede ser variado a través de los botones VOL+ y VOL- del Control Remoto. Se utiliza para elegir caracteres especiales, regular la velocidad de deslizamiento de los Mensajes y para elegir cual de los cuatro Mensajes se desea mostrar.

sliptime Guarda el valor de la variable vol cuando el Display Rotativo está en Modo Deslizamiento para evitar pérdida de datos.

lap Guarda el número de vueltas que se muestra un Mensaje. Se usa cuando el Display Rotativo está en Modo Alterno y todos los Mensajes se muestran uno tras otro. Trabaja con la Interrupción por Desborde del Timer2.

column Indica el número de la columna del caracter que está

siendo mostrado en los LEDs del Display Rotativo a través de la Interrupción por Desborde del Timer2.

position Indica la posición actual del cursor cuando algún Mensaje o el Reloj están siendo modificados.

cursor Guarda el valor que será restado del caracter que será mostrado para hacer aparecer un cursor debajo del mismo. Trabaja con la Interrupción por Desborde del Timer2.

col_img Indica el número de la columna de la Imagen Prediseñada que está siendo mostrado en los LEDs del Display Rotativo a través de la Interrupción por Desborde del Timer2.

letter Indica el número del caracter que está siendo mostrado en los LEDs del Display Rotativo a través de la Interrupción por Desborde del Timer2.

buffer[7] Este arreglo guarda momentáneamente el caracter que a su vez será guardado en la variable respectiva del arreglo *message[4][12][7*].

Variables de Reloj.

Las variables de Reloj se definen de la siguiente manera:

```
byte bufclock[7];
byte click,blink,flip,sigh,time_aux,day=10;
byte time[8]={1,1,0,1,1,0,1,0};
byte clock[12][7];
```

El uso de cada variable se explica a continuación:

buclock[7] Guarda momentáneamente el caracter que representa el número que a su vez será guardado en la variable respectiva del arreglo *clock[4][12][7*].

click, blink, flip, sigh Estas variables ayudan a hacer más preciso el Reloj del Display Rotativo mediante conteos. Trabaja con la Interrupción por Desborde del Timer0.

time_aux Guarda el número del botón presionado en el Control Remoto cuando se está configurando la hora actual. Este valor luego será guardado en la variable respectiva del arreglo time[8].

day Esta variable ayuda al Reloj a hacer el cambio de 23 a 0 cuando un día termina. Su valor inicial debe ser 10.

time[8] Este arreglo guarda el valor numérico del Reloj del display Rotativo que puede ser incrementado cada segundo e inclusive modificado al momento de configuración de la hora actual. Las posiciones 0 y 1 guardan las horas, las posiciones 3 y 4 los minutos y las posiciones 6 y 7 los segundos. Las posiciones 2 y 5 son indiferentes ya que representan los separadores de horas y minutos que no tienen información relevante.

clock[12][7] Este arreglo guarda los caracteres que serán mostrados en el Reloj del Display Rotativo en el formato HH:MM:SS. Para ver sus datos de inicialización se recomienda revisar el Apéndice.

6.5.5. Registros.

Los Registros utilizados en el programa *Rainbow.c* ya han sido explicados en el momento de analizar el funcionamiento del ATmega32 en el Capítulo 2 por lo que simplemente se mostrará un resumen de sus funciones a continuación:

PORTx, DDRx	Configuración, inicialización y manejo de puertos
TCCR0, TCNT0, OCR0	Configuración, inicialización y manejo del Timer0
TCCR1A, TCCR1B, TCNT1H, TCNT1L, ICR1H, ICR1L, OCR1AH, OCR1AL, OCR1BH, OCR1BL	Configuración, inicialización y manejo del Timer1
TCCR2, TCNT2, OCR2	Configuración, inicialización y manejo del Timer2
GICR, MCUCR, MCUCSR, GIFR	Configuración, inicialización y manejo de Interrupciones externas
TIMSK	Configuración, inicialización y manejo de Timers
ACSR, SFIOR	Inicialización del Comparador Análogo. (no usado, pero generado por el CodeWizardAVR automáticamente)

6.5.6. Función Main.

La función *void main(void)* del programa *Rainbow.c* es utilizada básicamente para inicializar los Registros del ATmega32 que manejan los puertos, las interrupciones externas y los Timers, esto se debe a que todas las acciones que realiza el microcontrolador se activan a través de Interrupciones.

A parte de lo mencionado, esta función tiene como única actividad comparar todo el tiempo el *Byte* más significativo del Registro Contador del Timer1 de 16 bits, llamado TCNT1, guardado en la variable *release,* con el valor de la constante WAIT. Esto es necesario para saber si el botón que está siendo presionado en el Control Remoto ha sido soltado. Una vez que dicho botón ha sido soltado, las variables *Gotit* y *Pack2* deben ser enceradas para que el microcontrolador acepte un nuevo comando. Todo ésto se realiza a través de la siguiente rutina:

```
while (1){
    release=TCNT1L;
    release=TCNT1H; //solo el valor de TCNT1H es necesario para la comparación

    if(release>=WAIT){
          Gotit=0;
          Pack2=0;
         };
};
```

6.5.7. Funciones.

Las funciones empleadas en el programa R*ainbow.c* tienen como objetivo mejorar la estructura del mismo al reducir líneas de código y separar las rutinas según su labor. De esta manera, se logra un mejor entendimiento del programa y se puede dar un enfoque más claro a determinadas partes del mismo sin tener que preocuparse del resto, así por ejemplo, se puede analizar el manejo de los LEDs y cambiar esta parte del programa sin tener que considerar posibles alteraciones en la parte de comunicación IR, ya que ambos procesos están controlados por funciones distintas.

Las funciones utilizadas son las siguientes:

1. void symenu()
2. void writemess(byte mess, byte pos, byte *buffy)
3. void command(byte com)
4. void deco()
5. void sirc()

Función symenu().

La función *symenu()* sirve para elegir entre los símbolo ASCII menos usados y los símbolos gestuales y llamativos al momento de presionar el botón *mute* del Control Remoto a través de un *switch* que evalúa la variable *vol*. Una vez que se ha elegido un símbolo, se lo guarda en la variable *buffer* a través de la función *conserve().* El código de la función *symenu()* es el siguiente:

```
void symenu(){
switch(vol){
      case 0:conserve(buffer,'+');break;
      case 1:conserve(buffer,'-');break;
      case 2:conserve(buffer,'/');break;
      case 3:conserve(buffer,'(');break;
      case 4:conserve(buffer,')');break;
      case 5:conserve(buffer,'$');break;
      case 6:conserve(buffer,'*');break;
      case 7:conserve(buffer,'@');break;
      case 8:conserve(buffer,'<');break;
      case 9:conserve(buffer,'>');break;
      case 10:conserve(buffer,'[');break;
      case 11:conserve(buffer,']');break;
      case 12:conserve(buffer,92);break;//\
      case 13:conserve(buffer,'_');break;
      case 14:conserve(buffer,'^');break;
      case 15:conserve(buffer,`');break;
      case 16:conserve(buffer,31);break;//:)
}
```

Función writemess().

La función writemess() se encarga de guardar en el Mensaje respectivo el caracter guardado en la variable *buffer* o *bukclock*. El parámetro *mess* indica el Mensaje seleccionado, siendo desde 0 hasta 3 para Mensajes y 4 para el Reloj. El

parámetro *pos* indica la posición en el Mensaje o Reloj pudiendo ser este valor desde 0 hasta 11. Finalmente el parámetro *buffy* indica que arreglo tiene el caracter a guardar, si *buffer o bufclock*. La Variable Local *j* es utilizada para operar con las 7 columnas del caracter correspondiente. El código de la función *writemess()* es el siguiente:

```
void writemess(byte mess, byte pos, byte *buffy){
byte j;
if(mess==4)for(j=0;j<=6;j++)clock[pos][j]=buffy[j];
else for(j=0;j<=6;j++)message[mess][pos][j]=buffy[j];
}
```

Función command().

La función command() maneja los comandos enviados por el Control Remoto relacionados con los botones de funciones especiales o comandos propiamente dichos como *sleep, enter, display, ch+, ch- vol+, vol-, mute* y *recall*.

Cada botón de comando tiene una rutina propia que cumplir que además no debe interferir con las rutinas de los demás botones, por lo que, una validación de ingreso de datos y acciones a tomar es necesaria en cada uno de ellos. La elección de la rutina a ejecutarse según el botón presionado se realiza mediante un *switch* que evalúa el valor del parámetro *com*, el cual puede ir de 0 hasta 8, y cada valor se relaciona a los botones *sleep, enter, ch+, ch-, vol+, vol-, mute, display* y *recall*. El código de la función *command()* es el siguiente:

```
void command(byte com){
switch(com){
      case 0://SLEEP
          if(Access==0){
          Access=1;
          }
          else Access=0;
          break;
      case 1://ENT
          if(Slip==0){
          Slip=1;
          vol=sliptime;
          GICR=0x00;
          MCUCR=0x00;
          MCUCSR=0x00;
          GIFR=0x40;
```

```
    TIMSK=0xA3;
    }
    else{
    Slip=0;
    vol=0;
    GICR=0x40;
    MCUCR=0x02;
    MCUCSR=0x00;
    GIFR=0x40;
    TIMSK=0xA1;
    }
    break;
case 2://CH+
    if(Slip==0&&Swap==0&&Imag==0){
    if(Tempo==0&&position<=10)position++;
    if(Tempo==1&&position<=3){
      position++;
      if(position==2){
        if(time[0]==1){
          position=0;
          conserve(buffer,' ');
          writemess(mess_pointer,position,buffer);
        }
        if(time[0]*10+time[1]>=35){
          position=0;
          time[position]=3;
          conserve(buffer,'2');
          writemess(mess_pointer,position,buffer);
          day=4;
          position=1;
          time[position]=4;
          conserve(buffer,'3');
          writemess(mess_pointer,position,buffer);
        }
        position=3;
      }
      if(position==4){
        if(time[3]>=6){
          position=3;
          time[position]=6;
          conserve(buffer,'5');
          writemess(mess_pointer,position,buffer);
        }
        position=4;
      }
      time[position]=1;
      conserve(buffer,'0');
      writemess(mess_pointer,position,buffer);
    }
    }
    break;
case 3://CH-
    if(Slip==0&&Swap==0&&Imag==0){
      if(Tempo==0&&position>=1)position--;
      if(Tempo==1&&position>=1){
        time[position]=1;
        conserve(buffer,'0');
        writemess(mess_pointer,position,buffer);
        position--;
        if(position==2)position--;
```

```
            }
          }
          break;
      case 4:  //VOL+
          if(vol<=15)vol++;
          else vol=16;
          if(Slip==1)sliptime=vol;
          if(mess_pointer==4)Imag=1;
          if(Symbol==0&&Slip==0&&Swap==0&&mess_pointer<=3){
            mess_pointer++;
            position=0;
            lut_pointer=0;
            if(mess_pointer==4)Tempo=1;
            }
          if(Symbol==1&&Slip==0&&Swap==0&&Imag==0){
          symenu();
          writemess(mess_pointer,position,buffer);
          }
          break;
      case 5://VOL-
          if(vol>=1)vol--;
          else vol=0;
          if(Slip==1)sliptime=vol;
          if(Symbol==0&&Slip==0&&Swap==0&&mess_pointer>=1){
            if(Imag==1)Imag=0;
            else mess_pointer--;
            position=0;
            lut_pointer=0;
            if(mess_pointer!=4)Tempo=0;
            }
          if(Symbol==1&&Slip==0&&Swap==0&&Imag==0){
          symenu();
          writemess(mess_pointer,position,buffer);
          }
          break;
      case 6://MUTE
          if(Slip==0&&Swap==0&&Imag==0){
          if(color_pointer<=1)color_pointer++;
          else color_pointer=0;
          colors[mess_pointer][position]=color_pointer;
          }
          break;
      case 7://DISP
          if(Swap==0&&Imag==0){
          mess_pointer=0;
          Swap=1;
          }
          else Swap=0;
          break;
      case 8://RECALL
          if(Symbol==0&&Swap==0&&Slip==0&&Imag==0){
          Symbol=1;
          vol=0;
          conserve(buffer,'+');
          writemess(mess_pointer,position,buffer);
          }
          else Symbol=0;
          break;
}
}
```

Botón Sleep. Acceso al Display Rotativo

Al presionar el botón *sleep* se cambia el valor de la bandera *Access*, si es 0 el acceso al Display Rotativo es denegado, y si es 1 el acceso es permitido.

Botón Enter. Modo Deslizamiento.

Al presionar el botón *enter* se cambia el valor de la bandera *Slip*, si es 0 el Modo Deslizamiento está desactivado y si es 1 está activado. Al estar activado el Modo Deslizamiento, la variable *vol* toma el valor de la variable *sliptime* que guarda el último valor ingresado por el usuario para regular la velocidad de giro de los LEDs del Display Rotativo a través de *vol+* y *vol-*. Luego, se deshabilitan las interrupciones externas para que la Interrupción Externa INT0 activada por el sensor de movimiento IR deje de funcionar y se habilite la Interrupción por Comparación del Timer0. De esta manera, se logra que la presentación de caracteres o de la Imagen Prediseñada a través de los LEDs no sea estática sino que se mueva en sentido horario o antihorario dependiendo de la velocidad ingresada por el usuario a través de los botones *vol+* y *vol-* del Control Remoto.

Al estar deactivado el Modo Deslizamiento, se realiza un proceso inverso en el cual se habilita la Interrupción Externa INT0 y se deshabilita la Interrupción por Comparación del Timer0 para que la presentación en los LEDs vuelva a ser estática.

Botón Ch+. Movimiento del cursor.

Al presionar el botón *ch+* el cursor que indica la posición actual con respecto a la línea de Mensaje debe moverse hacia a la derecha. Debido a que el cursor solo debe moverse cuando el Display Rotativo está en Modo Normal, es necesario validar esta acción mediante la evaluación de las banderas *Slip, Swap* e *Imag*. El cursor solo debe llegar hasta la posición del caracter número 12.

Además, lo que hace más compleja esta rutina es el hecho de que hay que validar las acciones a tomar cuando se configure la hora actual del Reloj. Se puede conocer si se está mostrando el Reloj en los LEDs a través de la bandera *Tempo*. Primero hay que validar que al configurar el Reloj, el cursor salte las posiciones de los separadores de horas y minutos. También hay que validar que si se ingresan horas mayores a 23, se guarde el valor de 23; y si se ingresan minutos mayores a 59, se guarde 59 en las posiciones correspondientes de la variable *time[]* al momento de mover el cursor a la siguiente posición. También se debe validar que las horas menores a 10 no muestren un 0 al principio sino un espacio en blanco.

Botón Ch-. Movimiento del cursor.

Al presionar el botón *ch-* el cursor que indica la posición actual con respecto a la línea de Mensaje debe moverse hacia a la izquierda. Debido a que el cursor solo debe moverse cuando el Display Rotativo está en Modo Normal, es necesario validar esta acción mediante la evaluación de las banderas *Slip, Swap* e *Imag*. El cursor solo debe llegar hasta la posición del caracter número 1.

Igualmente como en el caso del botón *ch+*, lo que hace más compleja esta rutina es el hecho de que hay que validar las acciones a tomar cuando se configure la hora actual del Reloj. Se puede conocer si se está mostrando el Reloj en los LEDs a través de la bandera *Tempo*. Sin embargo, lo único que hay que validar en esta ocasión al configurar el Reloj es que el cursor salte las posiciones de los separadores de horas y minutos, puesto que la validación de ingreso de horas y minutos solo es requerida cuando se presiona *ch+*.

Botón Vol+. *Selector*.

Al presionar el botón *vol+* se debe incrementar el valor de la variable *vol* siempre y cuando no sobrepase el valor de 16. Debido a que los botones *vol+* y *vol-* son usados como un *selector* para distintas funciones, se debe validar la acción a tomar dependiendo de las banderas activadas.

Si la bandera *Slip* está activada se debe guardar el valor de la variable *vol* en la variable *sliptime* puesto que el Modo Deslizamiento está activado y su velocidad de giro se regula con el botón *vol+*.

Si *mess_pointer* es igual a 4, significa que el usuario quiere visualizar la Imagen Prediseñada; por lo que se debe activar la bandera *Imag*.

Si el Display está en Modo Normal y *mess_pointer* no supera el valor de 3 se debe incrementar el valor de *mess_pointer* para mostrar el siguiente Mensaje, poniendo el cursor en la primera posición del Mensaje. Si resulta que el nuevo valor de *mess_pointer* es igual a 4, se debe mostrar el Reloj con la hora actual, para lo cual se debe activar la bandera *Tempo*.

Si la bandera *Symbol* está activada y el Display Rotativo está en Modo Normal, significa que el usuario está cambiando de símbolo en el Menú de Símbolos, por lo que, se debe ejecutar la función *symenu()* y luego guardar el símbolo escogido en el Mensaje y posición actual mediante la función *writemess().*

Botón Vol-. *Selector*.

Al presionar el botón *vol-* se debe decrementar el valor de la variable *vol* siempre y cuando no sobrepase el valor de 0. Al igual que en el caso del botón *vol+, vol-* es usado como un *selector* para distintas funciones por lo que se debe validar la acción a tomar dependiendo de las banderas activadas.

Si la bandera *Slip* está activada, se debe guardar el valor de la variable *vol* en la variable *sliptime* puesto que el Modo Deslizamiento está activado y su velocidad de giro también se regula con el botón *vol-*.

Si el Display está en Modo Normal y *mess_pointer* no es menor que 1 se debe desactivar la bandera *Imag* y decrementar el valor de *mess_pointer* para mostrar el Mensaje anterior, poniendo el cursor en la primera posición del Mensaje. Si

resulta que el nuevo valor de *mess_pointer* es diferente a 4, no se debe mostrar el Reloj con la hora actual por lo cual se debe desactivar la bandera *Tempo*. De esta manera, además se puede utilizar la misma rutina para moverse de un Mensaje a otro, y desactivar la presentación del Reloj con la hora actual y de la Imagen Prediseñada sin necesidad de más líneas de código basándose en el hecho de que cada vez que el usuario presiona el botón *vol-* es para desactivar la presentación de la Imagen Prediseñada o del Reloj.

Si la bandera *Symbol* está activada y el Display Rotativo está en Modo Normal, significa que el usuario está cambiando de símbolo en el Menú de Simbolos por lo que se debe ejecutar la función *symenu()* y luego guardar el símbolo escogido en el Mensaje y posición actual mediante la función *writemess().*

Botón Mute. Selección de color.

Al presionar el botón *mute* se debe cambiar el color del caracter actual. Esto debe ocurrir solo si el Display Rotativo está funcionando en Modo Normal. El número del color que puede estar entre 0 y 2 se guarda en la variable *color_pointer*. Luego se guarda el valor de la variable *color_pointer* en el elemento respectivo del arreglo *colors[][]*, dependiendo de la posición del Mensaje y del caracter actual.

Botón Display. Modo Alterno.

Al presionar el botón *display* se cambia el valor de la bandera *Swap*, si es 0 el Modo Alterno está desactivado y si es 1 está activado. Debido a que el Modo Alterno y la presentación de la Imagen Prediseñada no pueden funcionar al mismo tiempo, se debe validar que el Modo Alterno no se active cuando la bandera *Imag* esté activada. Al estar encendido el Modo Alterno la variable *mess_pointer* debe apuntar al primer Mensaje para comenzar la presentación secuencial de Mensajes.

Botón Recall. Menú de Símbolos.

Al presionar el botón *recall* se cambia el valor de la bandera *Symbol*, si es 0 el acceso al Menú de Símbolos está negado y si es 1 está permitido. El Menú de Símbolos debe ser accesible solo cuando el Display Rotativo está funcionando en Modo Normal. Estando el ingreso al Menú de Símbolos permitido, se debe encerar la variable *vol* para que apunte al primer símbolo del menú que es "+" y se lo debe mostrar en la posición actual del cursor a través de las funciones *conserve()* y *writemess()* para indicar al usuario que puede escoger el símbolo que desee con los botones *vol+* y *vol-*. Una vez que haya escogido un símbolo puede desactivar el Menú de Símbolos presionando nuevamente el botón *recall* para que los botones *vol+* y *vol-* puedan volver a ser usados para moverse de un Mensaje a otro.

Función deco().

La función *deco()* utilizada en este capítulo es una adaptación ampliada de la función *deco()* empleada en el capítulo anterior. El código de la función *deco()* es el siguiente:

```
void deco(){
    if(Pack2==1){
          old_code=new_code;
          new_code=b0+2*b1+4*b2+8*b3+16*b4+32*b5+64*b6+128*b7;
          if(new_code!=old_code)lut_pointer=0;

if(new_code>=0x80&&new_code<=0x89&&Slip==0&&Swap==0&&Imag==0&&Access==1){
              switch(new_code){
                case 0x80:
                    if(Tempo==1)time_aux=1;
                    else{
                    switch(lut_pointer){
                      case 0:conserve(buffer,'1');break;
                      case 1:conserve(buffer,'.');break;
                      case 2:conserve(buffer,',');break;
                      case 3:conserve(buffer,'!');break;
                      case 4:conserve(buffer,'?');break;
                      case 5:conserve(buffer,'"');break;
                      case 6:conserve(buffer,39);lut_pointer=-1;break;
                    }}
                    break;
                case 0x81:
                    if(Tempo==1)time_aux=2;
```

```
    else{
    switch(lut_pointer){
      case 0:conserve(buffer,'A');break;
      case 1:conserve(buffer,'B');break;
      case 2:conserve(buffer,'C');break;
      case 3:conserve(buffer,'2');lut_pointer=-1;break;
    }}
    break;
case 0x82:
    if(Tempo==1)time_aux=3;
    else{
    switch(lut_pointer){
      case 0:conserve(buffer,'D');break;
      case 1:conserve(buffer,'E');break;
      case 2:conserve(buffer,'F');break;
      case 3:conserve(buffer,'3');lut_pointer=-1;break;
    }}
    break;
case 0x83:
    if(Tempo==1)time_aux=4;
    else{
    switch(lut_pointer){
      case 0:conserve(buffer,'G');break;
      case 1:conserve(buffer,'H');break;
      case 2:conserve(buffer,'I');break;
      case 3:conserve(buffer,'4');lut_pointer=-1;break;
    }}
    break;
case 0x84:
    if(Tempo==1)time_aux=5;
    else{
    switch(lut_pointer){
      case 0:conserve(buffer,'J');break;
      case 1:conserve(buffer,'K');break;
      case 2:conserve(buffer,'L');break;
      case 3:conserve(buffer,'5');lut_pointer=-1;break;
    }}
    break;
case 0x85:
    if(Tempo==1)time_aux=6;
    else{
    switch(lut_pointer){
      case 0:conserve(buffer,'M');break;
      case 1:conserve(buffer,'N');break;
      case 2:conserve(buffer,'O');break;
      case 3:conserve(buffer,'6');lut_pointer=-1;break;
    }}
    break;
case 0x86:
    if(Tempo==1)time_aux=7;
    else{
    switch(lut_pointer){
      case 0:conserve(buffer,'P');break;
      case 1:conserve(buffer,'Q');break;
      case 2:conserve(buffer,'R');break;
      case 3:conserve(buffer,'S');break;
      case 4:conserve(buffer,'7');lut_pointer=-1;break;
    }}
    break;
case 0x87:
```

```
      if(Tempo==1)time_aux=8;
      else{
      switch(lut_pointer){
        case 0:conserve(buffer,'T');break;
        case 1:conserve(buffer,'U');break;
        case 2:conserve(buffer,'V');break;
        case 3:conserve(buffer,'8');lut_pointer=-1;break;
      }}
      break;
  case 0x88:
      if(Tempo==1)time_aux=9;
      else{
      switch(lut_pointer){
        case 0:conserve(buffer,'W');break;
        case 1:conserve(buffer,'X');break;
        case 2:conserve(buffer,'Y');break;
        case 3:conserve(buffer,'Z');break;
        case 4:conserve(buffer,'9');lut_pointer=-1;break;
      }}
      break;
  case 0x89:
      if(Tempo==1)time_aux=0;
      else{
      switch(lut_pointer){
        case 0:conserve(buffer,' ');break;
        case 1:conserve(buffer,'0');lut_pointer=-1;break;
      }}
      break;
  }
  if(Tempo==1){
  time[position]=time_aux+1;
  if(time[0]>=3)day=4;
  else day=10;
  conserve(buffer,'0'+time_aux);
  }

  writemess(mess_pointer,position,buffer);
  lut_pointer++;
  }
  else{
  if(new_code==0xb6) command(0);//SLEEP
  else{
   if(Access==1){
    switch(new_code){
     case 0x8b://ENT
       command(1);
       break;
     case 0x90://CH+
       command(2);
       break;
     case 0x91://CH-
       command(3);
       break;
     case 0x92://<<VOL+
       command(4);
       break;
     case 0x93://VOL-
       command(5);
       break;
     case 0x94://MUTE
```

```
                command(6);
                break;
              case 0xba://DISP
                command(7);
                break;
              case 0xbb://<<RECALL
                command(8);
                break;
             }
            }
           }
           }

        Gotit=1;
    }
    Pack2++;
}
```

La función *deco()* primero se encarga de verificar si se trata del segundo paquete de datos enviado por el Control Remoto a través de la bandera *Pack2,* si es así guarda en la variable *old*_code el valor actual de la variable *new_code* y posteriormente une todos los bits de datos *b0, b1,…* y *b7* sumándolos para formar un número de ocho bits y lo guarda dentro de la variable *new_code*.

Luego se comparan las variables *old_code* y *new_code* para determinar si se está presionando el mismo botón o un botón diferente en el Control Remoto. Dependiendo del resultado se decide si se encera el valor guardado en la variable *lut_pointer*, que apunta hacia un determinado caracter dentro de la tabla de caracteres de cada botón. La variable *lut_pointer* se encarga; por ejemplo, en el caso de presionar el botón 2 del Control Remoto, de indicar si el caracter a guardar es A, B, C ó 2. Cada vez que se presione un nuevo botón, el valor de la variable *lut_pointer* se debe encerar para que la tabla de caracteres comience en su primera posición. Luego se divide la acción a tomar en dos partes, dependiendo si se presiona un botón numérico o un botón de función.

Solo si el Display Rotativo está en Modo Normal se evalúa si se presionó un botón numérico; es decir, del *0* hasta el *9*. En este caso se guarda su valor numérico en la variable *time_aux* si la bandera *Tempo* está activada o se guarda un caracter en la variable *buffer* a través de la función *conserve()* si la bandera *Tempo* está desactivada. El caracter a guardar dependerá del valor de la variable *lut_pointer* como ya se mencionó anteriormente. Además, cuando la variable

Tempo está activada, se debe guardar en la posición respectiva del arreglo *time[]* el valor de la variable *time_aux* más uno, también se debe evaluar el valor guardado en *time[0]* mediante la variable *day* para permitir que el Reloj cambie de 23 a 0 al terminar un día y luego se debe guardar en el *buffer* el número ingresado a través de la función *conserve().* Finalmente se guarda el caracter escogido en el Mensaje y posición actual mediante la función *writemess()* y se incrementa el valor de la variable *lut_pointer.*

Si se presiona un botón de comando como *sleep, display, enter,ch+, ch-, vol+, vol-, mute* o *recall* lo primero que se hace es comprobar si se presionó el botón *sleep* que permite o deniega el acceso al Display Rotativo. En el caso de que la bandera *Access* esté activada y por lo tanto el acceso esté permitido, se procederá a realizar la función escogida a través de un *switch* que evalúa el valor de la variable *new_code* y según eso ejecuta la función *command()* con el parámetro respectivo.

Al final de la función *deco()* se activa la bandera *Gotit* para indicar que ya se decodificó el comando enviado por el Control Remoto y se activa la bandera *Pack2* para indicar que se ha recibido el segundo paquete de datos desde el Control Remoto.

Función sirc().

El código de la función *sirc()* es el siguiente:

```
void sirc(){
if(width>=TDOWN-K&&width<=TDOWN+K)BitRx=0;
if(width>=2*TDOWN-K&&width<=2*TDOWN+K)BitRx=1;

if(StartBit==1&&Gotit==0){

    switch (nBit) {
        case 0:b0=BitRx;nBit++;break;
        case 1:b1=BitRx;nBit++;break;
        case 2:b2=BitRx;nBit++;break;
        case 3:b3=BitRx;nBit++;break;
        case 4:b4=BitRx;nBit++;break;
        case 5:b5=BitRx;nBit++;break;
        case 6:b6=BitRx;nBit++;break;
```

```
            case 7:b7=BitRx;nBit++;break;
            case 8:nBit++;break;
            case 9:nBit++;break;
            case 10:nBit++;break;
            case 11:nBit=0;StartBit=0;deco();break;
            default:nBit=0;StartBit=0;break;
        };
 };

if(width>=4*TDOWN-2*K&&width<=4*TDOWN+K)StartBit=1;

}
```

La función *sirc()* recopila lo aprendido en el Capítulo 5 sobre transmisión infrarroja. Esta función se encarga de traducir en ceros, unos o STARTBITs los comandos codificados por ancho de pulso y transmitidos desde el Control Remoto. Cada vez que la función *sirc()* es llamada, compara el valor de la variable *width* que contiene el ancho de pulso capturado con valores que resultan de operar TDOWN y K para determinar si se capturó un 0, un 1 o un STARTBIT en la variable *BitRx*.

La variable *StartBit* sirve como una bandera que indica cuando se ha capturado un STARTBIT, lo que alista a la función *sirc()* para capturar los siguiente ocho bits de datos en las variables *b0, b1,…* y *b7*. Luego de traducir a binario los ocho bits de datos, la función *deco()* se encargará de decodificar el comando transmitido desde el Control Remoto.

La variable *Gotit* impide que se siga operando el mismo comando del Control Remoto mientras se siga presionando cualquiera de sus botones, de esta manera una vez que se tiene el comando transmitido la función *sirc()* no altera los ocho bits de datos obtenidos ni los decodifica a través de la función *deco()* hasta que se suelte el botón que se está presionando y se vuelva a presionar el mismo u otro botón. De esta manera se evita que el microcontrolador esté trabajando todo el tiempo en esta parte del programa y le da más agilidad para dedicarse a otras funciones.

6.5.8. Modo Normal. Interrupt [EXT_INT0].

El código de la interrupción *interrupt [EXT_INT0]* es el siguiente:

```
interrupt [EXT_INT0] void ext_int0_isr(void){
TCCR2=0x05;
TCNT2=0x00;
letter=0;
column=0;
}
```

Cada vez que el sensor IR de movimiento recibe la señal del LED IR que se encuentra en la base del Display Rotativo se activa esta interrupción externa que a su vez activa la Interrupción por Comparación del Timer2 que controla la presentación de Mensajes o de la Imagen Prediseñada columna por columna e inicializa las variables *letter* y *column*. De esta manera se logra que la presentación se inicie siempre en la misma posición y se vea estática desde el punto de vista del usuario.

6.5.9. Modo Deslizamiento. Interrupt [TIM0_COMP].

El código de la interrupción *interrupt [TIM0_COMP]* es el siguiente:

```
interrupt [TIM0_COMP] void timer0_comp_isr(void){
TCCR2=0x05;
TCNT2=0x00;
OCR0=OCR0+PERIOD+5*sliptime;
letter=0;
column=0;
}
```

La Interrupción por Comparación del Timer0 se encarga de *deslizar* el Mensaje o la Imagen Prediseñada. Para realizar esta operación trabaja de manera similar a la Interrupción Externa INT0 activando la Interrupción por Comparación del Timer2 encargada de mostrar cada caracter del Mensaje o la Imagen Prediseñada columna por columna e inicializando las variables *letter* y *column*.

Si se quisiera que el Mensaje apareciera siempre en el mismo sitio se debería cosiderar el periodo de giro de la columna de ocho LEDs del Display Giratorio, el cual es cercano a 180 (0,09216 s aproximadamente) y activar una Interrupción por Comparación en el Timer0 cada vez que transcurra un periodo exacto. Como no se puede encerar el registro *TCNT0* porque también es utilizado para medir el tiempo mostrado en el Reloj, mejor se opera *OCR0=OCR0+PERIOD* con lo cual se obtiene el mismo efecto. Si el valor de *PERIOD* es de 170 en vez de 180 se consigue que el Mensaje gire rápidamente en sentido horario. Si además se incrementa el valor del registro *OCR0* también a través de *4*sliptime* se logra que disminuya la velocidad a la que gira el Mensaje en sentido horario hasta que llega a un punto en que comienza a girar en sentido antihorario.

6.5.10. Reloj HH:MM:SS. Interrupt [TIM0_OVF]

El código de la interrupción *interrupt [TIM0_OVF]* es el siguiente:

```
void timer0_ovf_isr(void){
click++;
if(click>=8){
click=0;
blink++;
if(time[7]>=10){
  if(time[6]>=6){
   if(time[4]>=10){
     if(time[3]>=6){
       if(time[1]>=day){
         if(time[0]>=3){
           time[0]=0;
           day=10;
         }
 if(time[0]==0)conserve(bufclock,' ');
         else conserve(bufclock,'0'+time[0]);
         writemess(4,0,bufclock);
         time[1]=0;
         time[0]++;
         if(time[0]>=3)day=4;
       }
       conserve(bufclock,'0'+time[1]);
       writemess(4,1,bufclock);
       time[3]=0;
       time[1]++;
     }
     conserve(bufclock,'0'+time[3]);
     writemess(4,3,bufclock);
     time[4]=0;
     time[3]++;
   }
```

```
   conserve(bufclock,'0'+time[4]);
   writemess(4,4,bufclock);
   time[6]=0;
   time[4]++;

 }
 conserve(bufclock,'0'+time[6]);
 writemess(4,6,bufclock);
 time[7]=0;
 time[6]++;
 click++;  //cada 10 seg
}
conserve(bufclock,'0'+time[7]);
writemess(4,7,bufclock);
time[7]++;
}
if(blink>=8){ //cada 8 seg
click++;
blink=0;
flip++;
}
if(flip>=18){ //
click++;
flip=0;
sigh++;
}
}
```

La Interrupción por Desborde del Timer0 se encarga de medir la hora actual que será mostrada por el Reloj del Display Rotativo y guardarla en los arreglos *time[]* y *clock[][]*. Para evitar tener que actualizar todos los números que conforman la hora actual cada segundo lo cual consumiría muchos recursos del ATmega32, se utiliza una anidamiento de estructuras *if* que solo actualizan los números que cambian a través de la variable *bufclock* y las funciones *conserve()* y *writemess()*.

Para permitir que se produzca el cambio de 23 a 0 al terminar un día, se utiliza la variable *day* que indica la diferencia entre contar hasta 19 y luego pasar a 20 y contar hasta 23 y luego pasar a 0. De igual manera, cuando el valor del elemento más significativo del arreglo *time[]* es 0, se determina que se presente un espacio en blanco en los LEDs en vez de un 0, para evitar mostrar “09:00”, sino “ 9:00”.

Para mejorar la precisión del Reloj se utilizan las variables *click, blink* y flip. El ATmega32 tiene una frecuencia de trabajo de 2 MHz con un prescaler de 1024. El

desborde en 0xFF del contador del Timer0 se produce cada 0,131072 s[10]. Cada vez que ocurre un desborde se incrementa el valor de la variable *click*. Cada 8 *clicks* equivalen a 1,048576 s por lo que se procede a aumentar un segundo en el Reloj guardado en el arreglo *time[]* y además se incrementa el valor de la variable *blink*. Para corregir el error de 0,048576 s que se produce cada segundo se resta 0,131072 s cada 10 segundos y cada 8 segundos incrementando nuevamente la variable *click.*

6.5.11. Recepción Infrarroja. Interrupt [TIM1_CAPT]

El código de la interrupción *interrupt [TIM1_CAPT]* es el siguiente:

```
interrupt [TIM1_CAPT] void Counter1_capt_isr(void)
{
captured_time=ICR1L;
    switch (Edge) {
          case 0:
             Edge=1;
             TCCR1B=RISE;
             if(captured_time>=TUP-K){
                  TCNT1H=0x00;
                  TCNT1L=0x00;
             };
          break;
          case 1:
             Edge=0;
             TCCR1B=FALL;
             if(captured_time>=TDOWN-K){
                    width=captured_time;
                    TCNT1H=0x00;
                    TCNT1L=0x00;
             };
             sirc();
          break;
    };
}
```

La Interrupción por Captura de Datos del Timer1 se encarga de capturar los datos transmitidos por el Control Remoto. Debido a que el envió de datos desde el Control Remoto debe ser analizado por parte del microcontrolador todo el tiempo,

[10] Este valor se obtiene mediante la fórmula 256/(2MHz/1024)

esta interrupción jamás se deshabilita. Lo primero que hace la interrupción es guardar en la variable *captured_time* el valor del registro ICR1L y a través de la variable *Edge* decide guardarlo o no en la variable *width* dependiendo si es un flanco ascendente o descendente.

Además esta interrupción se encargar de discriminar a manera de filtro todas las señales que sean menores que *TUP-K* o *TDOWN-K* para evitar que entre ruido al proceso.

6.5.12. Despliegue de Datos. Modo Alterno. Interrupt [TIM2_COMP]

El código de la interrupción *interrupt [TIM2_COMP]* es el siguiente:

```
interrupt [TIM2_COMP] void timer2_comp_isr(void){
TCNT2=0x00;
if(Imag==0){
if(letter==position&&Access==1)cursor=0x80;
else cursor=0x00;
if(column<=7){
 if(column<=6){
  if(mess_pointer==4){
   switch(letter){
    case 2:PORTC=clock[letter][column]-cursor;break;
    case 5:PORTC=clock[letter][column]-cursor;break;
    default:PORTB=clock[letter][column]-cursor;break;
    }
   }
  else{
   switch(colors[mess_pointer][letter]){
    case 0:PORTA=message[mess_pointer][letter][column]-cursor;break;
    case 1:PORTB=message[mess_pointer][letter][column]-cursor;break;
    case 2:PORTC=message[mess_pointer][letter][column]-cursor;break;
    default:PORTA=message[mess_pointer][letter][column]-cursor;break;
   }
  }
 }
 else {PORTA=0xFF;PORTB=0xFF;PORTC=0xFF;}
 column++;
}
else{
column=0;
letter++;
}
if(letter>=12){
 letter=0;
 PORTA=0xFF;PORTB=0xFF;PORTC=0xFF;
 TCCR2=0x00;
 TCNT2=0x00;
 if(Swap==1){
```

```
  if(lap<=12)lap++;
  else{
   lap=0;
   if(mess_pointer<=2)mess_pointer++;
   else mess_pointer=0;
  }
 }
}
}
else{
if(col_img<=89){
PORTA=~image0[col_img][0]|image0[col_img][1];
PORTB=image0[col_img][0]|~image0[col_img][1];
PORTC=image0[col_img][0]|image0[col_img][1];
col_img++;
}
else{
col_img=0;
TCCR2=0x00;
TCNT2=0x00;
PORTA=0xFF;
PORTB=0xFF;
PORTC=0xFF;
}
}
}
```

La Interrupción por Comparación del Timer2 se encarga de mostrar los Mensajes o la Imagen Prediseñada columna por columna. Si la frecuencia de trabajo del microcontrolador es de 2 MHz, el prescaler del Timer2 es 128 y el valor de comparación de la Interrupción por Comparación es 0x08, se puede determinar que la Interrupción por Comparación del Timer2 se ejecuta cada 0,512 ms[11]. Este tiempo es suficientemente rápido para crear el efecto de una línea de luz continua en el espacio, en el cual se basa el funcionamiento del Display Rotativo.

Además considerando que el periodo de giro de la columna de LEDs es 0,09216 s, 0,512 ms representan apenas 2º de la circunferencia descrita. Siendo que el observador tiene la capacidad de ver hasta 180º de dicha circunferencia, en teoría se podrían mostrar hasta 90 columnas de Mensajes o de la Imagen Prediseñada. Por este motivo el límite de caracteres para cada línea de Mensaje es 12 puesto que cada caracter ocupa 7 columnas más la columna de espacio, lo que da un total de 96 columnas como máximo[12], como se puede apreciar en la Figura 6.37.

[11] Este valor se obtiene mediante la fórmula 8/(2 MHz/128)
[12] Este valor se obtiene mediante la fórmula (7 x 1) x 12 = 96

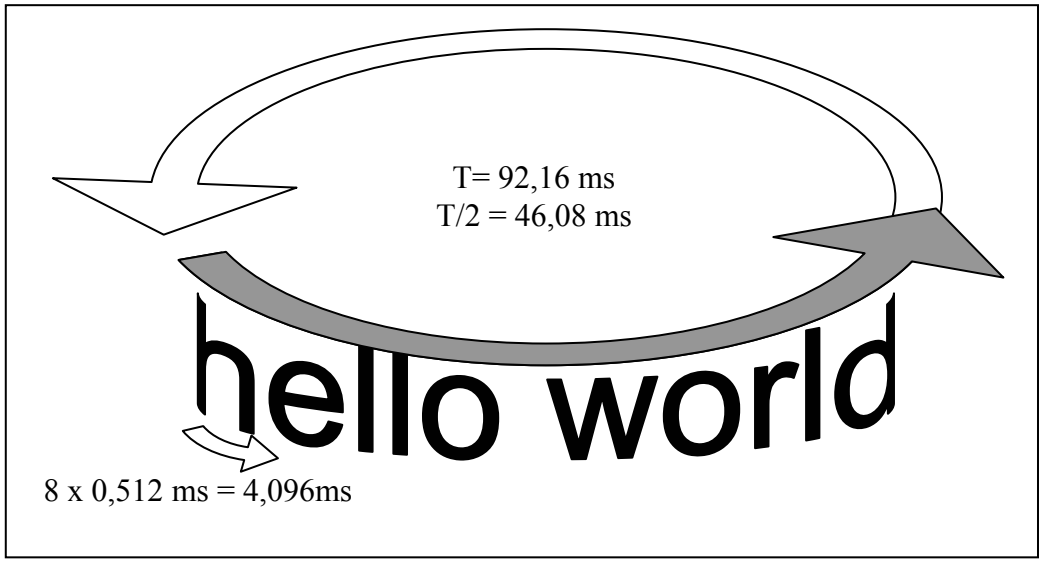


Figura 6.37. Tiempo de encendido de cada columna

Si el valor de la bandera *Imag* es 0 se presenta un Mensaje; caso contrario se presenta la Imagen Prediseñada. En la presentación de Mensajes lo primero que se hace es guardar en la variable *cursor* el valor de 0x80 que se encarga de encender el LEDs más bajo de la columna de LEDs para mostrar una línea que representa un cursor. Caso contrario se guarda en la variable *cursor* el valor de 0x00 que oculta el cursor. Luego se muestra la correspondiente columna del caracter.

Si la variable *mess_pointer* tiene el valor de 4, se debe mostrar el Reloj con la hora actual. Los números de presentan en color verde y los separadores de horas y minutos en color azul. Cabe recordar que el Puerto A maneja el color rojo, el Puerto B el color verde y el Puerto C el color azul.

Si se trata de cualquiera de los cuatro Mensajes restantes se debe mostrar cada caracter con su color respectivo por lo que se debe evaluar el valor del arreglo *colors[][]* a través de un *switch*. Para prevenir cualquier error en caso de no encontrar un valor entre 0 y 2 (0 para rojo, 1 para verde y 2 para aul) en la posición respectiva del arreglo *colors[][]*, se mostrará el caracter en color rojo.

Luego de mostrar las 7 columnas que forman un caracter, incrementando el valor de la variable *column*, se debe dejar una columna en negro (ausencia de color) para separar los caracteres. Para lograr esto se debe guardar el número 0xFF en los Puertos A, B y C.

Se repite el mismo proceso para los 12 caracteres de la línea de Mensaje incrementando el valor de la variable *letter.* Cuando se complete la línea de Mensaje se debe desactivar la Interrupción por Comparación del Timer2 y apagar todos los LEDs a través del número 0xFF.

En caso de que la bandera *Swap* esté activada y por lo tanto el Display Rotativo esté funcionando en Modo Alterno se debe cambiar de Mensaje cada 13 vueltas o ciclo. El conteo de ciclos se realiza a través de la variable *lap*. Cuando el valor de la variable *lap* llega a 13, se incrementa el valor de la variable *mess_pointer*, cambiando de Mensaje. Siendo el periodo de giro de los LEDs 0,09216 s, 13 ciclos equivalen a 1,19808 s, que es un tiempo prudencial para cambiar de Mensajes, permitiendo al observador entender cada uno de ellos.

La presentación de la Imagen Prediseñada se realiza mostrando cada columna de la misma a través del incremento del valor de la variable *col_img*. Debido a que cada LED de la columna de LEDs debe encenderse con un color diferente se utiliza la lógica explicada anteriormente para encender los Puertos A, B y C considerando el valor correspondiente del arreglo *image0[][]*. Al igual que con la presentación de Mensajes, al terminar de mostrar la imagen se debe desactivar la Interrupción por Comparación del Timer2 y apagar todos los LEDs a través del número 0xFF.

6.5.13. Matriz de Datos. Archivo auxiliar Charlut.c

El archivo auxiliar *Charlut.c* se encarga de manejar la tabla de caracteres ASCII y símbolos gestuales y llamativos disponibles para ser mostrados en los las líneas de Mensajes del Display Rotativo a través del programa *Rainbow.c*. Para incluir el

archivo *Charlut.c* en el archivo *Rainbow.c* se debe seguir el procedimiento explicado en la Sección 3.1.10. Inclusión de de Archivos de Cabecera .H.

De los 128 caracteres ASCII existentes, se utiliza aquellos que están dentro del rango 32 – 96 como se muestra en la Tabla 6.7.[13]

Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
32	20	040		Space	64	40	100	@	@	96	60	140	`	`
33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Tabla 6.7. Caracteres ASCII

El archivo *Charlut.c* contiene el arreglo *alpha[66][7]* que guarda los 66 caracteres disponibles en un formato de 7 *Bytes* por caracter, donde cada *Byte* representa una columna del caracter.

Además posee la función *conserve()* que es utilizada en el archivo *Rainbow.c* para guardar en la variable *buffer* o *bufclock* el caracter correspondiente a la elección del usuario. El parámetro **mat* es ocupado por la variable *buffer* o *bufclock*, y el parámetro *index* por el código ASCII del caracter que se quiere

[13] La Tabla 6.7. fue obtenida de la página de Internet http://www.LookupTable.com

guardar. Para los símbolos gestuales o llamativos se debe ingresar en el parámetro *index* el número correspondiente a su ubicación en el arreglo *alpha[][]* más 31.

Puesto que ingresar todos los caracteres en formato binario en vez de formato hexadecimal es más sencillo de visualizar, pero requiere de muchas líneas de código, a continuación solo se muestra como se inicia y se finaliza el archivo *Charlut.c*. Para ver el archivo completo se recomienda revisar el Anexo 2.

```
unsigned char alpha[66][7]={
{0b11000001,//:)
 0b10110110,
 0b10100110,
 0b10100000,
 0b10100110,
 0b10110110,
 0b11000001},
{0b11111111,//Space
 0b11111111,
 0b11111111,
 0b11111111,
 0b11111111,
 0b11111111,
 0b11111111},
……..//Codigo para los caracteres restantes: ! “ $.... Z [  \ ] ^
……..
{0b10111111,//_
 0b10111111,
 0b10111111,
 0b10111111,
 0b10111111,
 0b10111111,
 0b10111111},
{0b11111110,//`
 0b11111101,
 0b11111011,
 0b11111111,
 0b11111111,
 0b11111111,
 0b11111111}};
void conserve(unsigned char *mat, unsigned char index){
unsigned char i;
for(i=0;i<=6;i++)mat[i]=alpha[index-31][i];
}
```

Para poder incluir el archivo *Charlut.c* en el archivo *Rainbow.c* se debe crear el archivo *Charlut.h* cuyo código es el siguiente:

```
void conserve(unsigned char *mat,unsigned char index);
```

6.6. Diagrama de Bloques del Programa del Display Rotativo.

Debido a que el Programa del Display Rotativo se maneja básicamente a través Interrupciones Externas y Timers, el Diagrama de Bloques que se muestra a continuación en la Figura 6.38. está divido en partes independientes, las cuales funcionan solo cuando son requeridas.

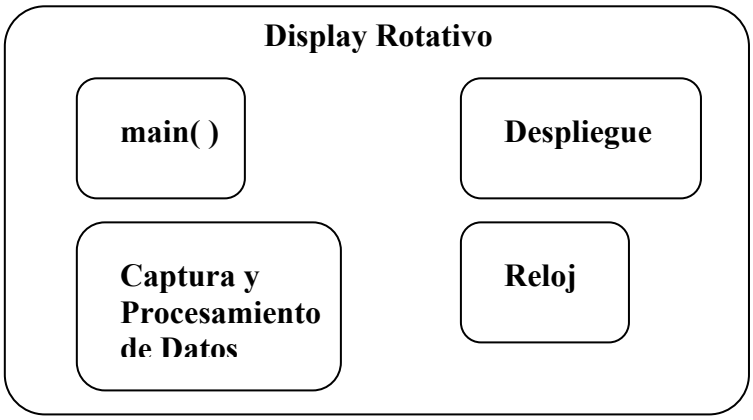


Figura 6.38. Diagrama de Bloques del Programa del Display Rotativo

Diagrama de Bloques de la Función main( ).

La función principal del programa del Display Rotativo, main( ), se encarga de inicializar las Variables y Registros. Además está constantemente monitoreando si se continúa enviando un comando desde el Control Remoto, como se muestra en la Figura 6.39.

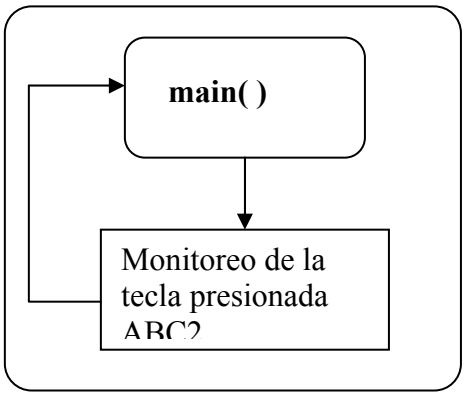


Figura 6.39. Diagrama de Bloques de la Función main( )

Diagrama de Bloques de la Captura y Procesamiento de Datos.

Este es el bloque encargado de recibir los comandos desde el Control Remoto, decodificarlos y ejecutar la acción correspondiente, como se puede ver en la Figura 6.40.

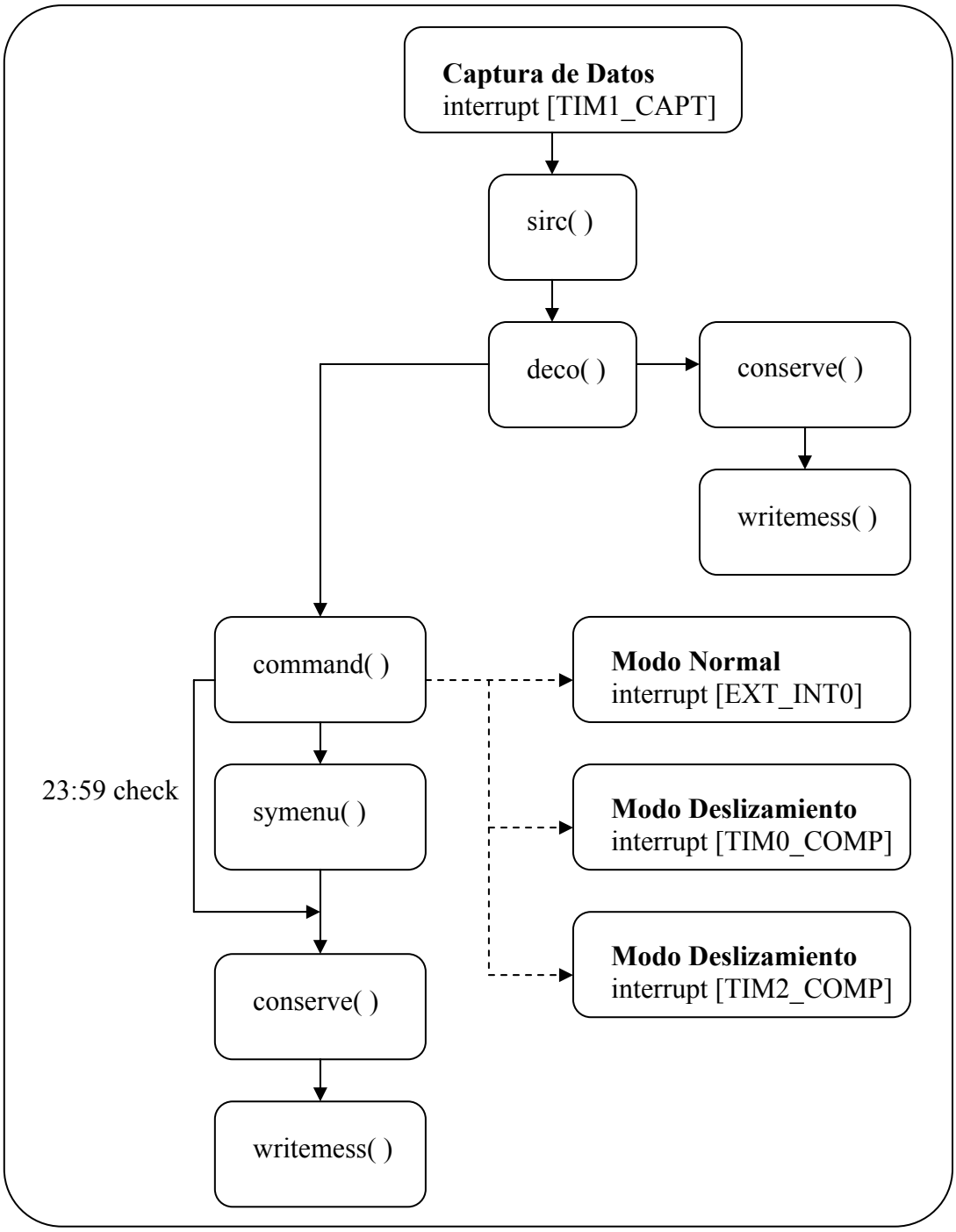


Figura 6.40. Diagrama de Bloques de la Captura y Procesamiento de Datos

Diagrama de Bloques del Despliegue.

La presentación del Mensaje o de las Imagen Prediseñada se realiza a través de una Interrupción por Comparación, como se muestra en la Figura 6.41. Su funcionamiento es independiente de la Captura y Procesamiento de Datos para permitir que el microcontrolador pueda realizar los dos procesos al mismo tiempo.

Figura 6.41. Diagrama de Bloques del Despliegue

Diagrama de Bloques del Reloj.

La hora actual mostrada por el Display Rotativo es determinada por una Interrupción por Desborde que mide cada segundo transcurrido. Luego hace las modificaciones respectivas en la hora desplegada, como se muestra en la Figura 6.41.

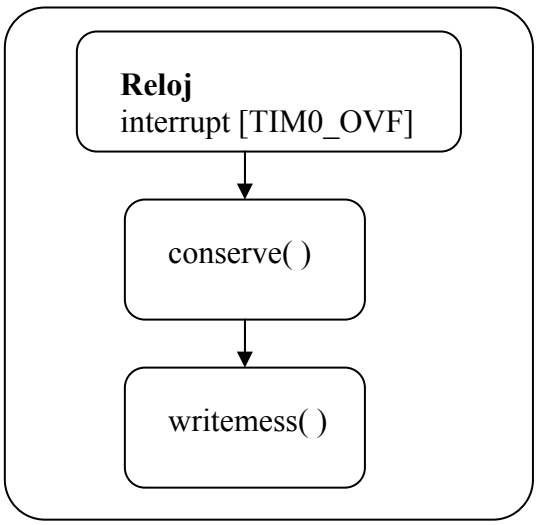


Figura 6.42. Diagrama de Bloques del Reloj

CAPÍTULO 7

CONCLUSIONES Y RECOMENDACIONES

7.1. CONCLUSIONES

- El diseño e implementación del Display Rotativo RGB programable con interfase SIRC y Teclado ABC2 fue satisfactorio puesto que se obtuvo un producto de alta calidad y bajo costo que representa una opción real en el mercado de los Anuncios Luminosos.

- La utilización del microcontrolador ATmega32 demostró ser una elección acertada debido a su confiabilidad, fácil manejo y a la disponibilidad de herramientas para el desarrollo de aplicaciones. De esta manera se logra abrir una nueva opción que puede representar una alternativa al uso de PICs en la creación de aplicaciones.

- La implementación del Display Rotativo emplea elementos de fácil adquisición en el mercado nacional y otros que solo pueden ser conseguidos a través de envíos desde el exterior como es el caso de los LEDs RGB y del Microcontrolador AVR.

- El Display Rotativo es la primera aplicación conocida en el país que utiliza LEDs RGB por lo que constituye una nueva forma de mostrar Anuncios Luminosos nunca antes utilizada en nuestro mercado.

- La Codificación SIRC representa una manera innovadora de comunicar equipos físicamente difíciles de acceder a través de cables ya que su codificación no es compleja y se puede adaptar a cualquier tipo de uso.

- La alimentación de poder al circuito giratorio dentro del Display Rotativo a través del eje es una solución original al problema de la forma en que brinda energía a elementos en movimiento que puede ser utilizada en otras aplicaciones.

- Los Ambientes de Desarrollo Integrado conocidos como IDEs utilizados en este proyecto: CodeVisionAVR C Compiler y AVR Studio son programas muy útiles, de simple manejo y con herramientas que facilitan la creación de aplicaciones para microcontroladores AVR.

- El grabador de microcontroladores AVR, PonyProg2000 es una opción práctica al momento de elegir la manera en que se trabajará entre el computador en que se desarrolla la aplicación y el microcontrolador AVR debido a su bajo costo y fiabilidad en comparación a las Plataformas de Desarrollo que existen en el mercado, que debido a su alto costo y dificultad de adquisición, se vuelven inaccesibles para el usuario.

- El desarrollo de este Proyecto ha sido realizado paso a paso para que las personas interesadas puedan entender la manera en que el Display Rotativo fue creado y además sirve de guía para implementar aplicaciones con microcontroladores AVR en general.

7.2. RECOMENDACIONES

- Al momento de implementar el Display Rotativo se debe tener precaución de seguir las indicaciones adecuadamente y utilizar las herramientas adecuadas ya que al ser un dispositivo en movimiento requiere que las piezas que lo conforman sea precisas.

- Si se desea implementar un Display Rotativo, los elementos importados, como el microcontrolador AVR y los LEDs RGB, deben ser enviados por una compañía confiable que asegure que dichos productos lleguen a su destino.

- Es posible reemplazar el ATmega32 por un ATmega16 para reducir costos ya que sus características son semejantes. De igual manera se puede reemplazar el sensor IR en la base del Display Rotativo por un sensor óptico (de luz) si así se desea.

- Cuando se utilice el Display Rotativo se deben seguir las recomendaciones establecidas en el Manual del Usuario (ver Anexo 1) para evitar daños tanto al dispositivo como a las personas que lo utilicen.

BIBLIOGRAFÍA

REVISTAS

- Atmel Applications Journal, Número 4, Primera edición, Atmel, USA, año 2005, pg 31-36, artículo: Wireless MP3 Remote Jukebox with Atmel AT90S2313-4PC 8-TRACKS TO CDS AND MP3S. Brian Miller.

INTERNET Y AYUDAS

- http://www.wisegeek.com/how-does-a-television-work.htm, Motion rate
- http://en.wikipedia.org/wiki/Frame_rate, Frame rate.
- http://en.wikipedia.org/wiki/Persistence_of_vision, Persistence of vision.
- http://psych.hanover.edu/nsfati/APASession1/Sequences.ppt, Image Sequence.
- http://www.LookupTable.com, Tabla de caracteres ASCII.
- http://www.atmel.com, Información técnica de los AVR ATmega.
- http://www.lancos.com/prog.html, PonyProg serial device programmer.
- http://www.bobblick.com/techref/projects/propclock/propclock.html, "Propeller Clock" Mechanically Scanned LED Clock by Bob Blick.
- http://www.makophone.com/ololinglcaid.html, Olimpia OLY-OL3000 Info Glob Caller ID Blue.
- http://www.rentron.com/Micro-Bot/IR_Nav.htm, Infrared Object Detection and Navigation.
- http://www.superbrightleds.com/RL5-IR2730, Super INFRA-RED 5mm LED specs.

- http://scv.bu.edu/GC/shammi/ir, SC546 Project, An analytical study of IR signals used by a SONY remote control.
- http://www.ustr.net/infrared/sony.shtml, Remote Control Technical Specifications SONY.
- http://www.overclockers.com/articles624, Brushless Motors.
- http://www.superbrightleds.com, LEDs RGB .
- http://www.alibaba.com/archives/product/150407/Electronics_Electrical.html, LED Displays Catalog.
- http://www.signweb.com/moving/cont/comingofage.html, “Coming of Age” by Bob Klausmeier.
- http://www.signindustry.com/led, Led Sign Industry.
- http:// www.cramersoftware.com/ illustr.html, Electromagnetic Spectrum.
- http://hem.bredband.net/robinstridh/avr/rot_led/index.html, An Introduction to Microcontrollers through a Practical Example: A Rotating LED Display. Robin Stridh. 29th April 2002. (archivo .pdf).
- CodeVisionAVR C Compiler Help.
- AVR Studio 4 Help.

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO

Assembly Lenguaje de programación básico, equivalente al lenguaje de máquina.

ATmega Subfamilia de los AVR, de gran capacidad.

ATtiny Subfamilia de los AVR, de menor capacidad.

AVR **Advanced Virtual RISC.** RISC Virtual Avanzado. Microcontrolador de 8 bits que representa la propuesta más avanzada de la Atmel en esta área. AVR también significa Alf Vegard RISC en referencia a sus creadores Alf Egil Bogen y Vegard Wollan.

AVRISP **AVR In-System Programmer.** Programador que permite grabar o leer un AVR sin tener que extraerlo de su aplicación.

CKSEL **Clock Selection.** Los 4 bits que determinan la fuente de reloj y frecuencia de los ATmega.

COFF **Common Object File Format.** Tipo de archivo utilizado en el proceso de depuración de un programa para un microcontrolador.

CRT **Cathode Ray Tube.** El tubo de rayo catódico, una de las tecnologías más usadas en pantallas o monitores de televisión y de ordenadores.

DC **Direct Current.** Corriente directa.

DIP **Dual In-line Package** . Tipo de Empaquetado de chips.

DPWM **Digital Pulse-Width Modulation.** Modulación Digital por ancho de pulso.

EDS	**Electronic Digital Signage.** Area de la electrónica dedicada a los Rótulos Digitales.
EEP	**EEPROM.** Tipo de archivo que se graba en la EEPROM interna de los microcontroladores.
EEPROM	**Electrically Erasable Programmable Read Only Memory.** Tipo de memoria de lectura en el ordenador que se borra únicamente por medio electrónico.
FPS	**Frames Per Second.** Medición de la cantidad de imágenes mostradas en un segundo en distintos sistemas de televisión.
HEX	**Hexadecimal.** Tipo de archivo que se graba en la Memoria Flash interna de los microcontroladores.
ICP	**Input Capture Pin.** Pin de captura de datos para la activación de Interrupciones en los ATmega.
IDE	**Integrated Development Environment.** Software usado para el desarrollo de proyectos de aplicación para microcontroladores.
INT	**Interruption.** Proceso de interrupción ocurrido en los ATmega.
IR	**Infrared.** Infrarrojo. Tipo de luz con una longitud de onda mayor a la luz visible.
ISP	**In-System Programmer.** Programador que permite grabar o leer un chip sin tener que extraerlo de su aplicación.
LCD	**Liquid Crystal Display.** Tipo de pantalla utilizada en relojes digitales y en ordenadores portátiles.
LED	**Light Emitting Diode. D**iodo emisor de luz.
Lenguaje C	Lenguaje de programación de alto nivel.
MDF	**Medium Density Fiberboard.** Material hecho de fibras de madera, más duro que el cartón.

Microcontrolador Microprocesador que comprende elementos fijos, como la unidad central, memorias y puertos I/O, y elementos personalizados en función de la aplicación.

MIPS **Million Instructions Per Second.** La medición de la velocidad y el poder de un procesador a través del número de instrucciones de máquina que puede ejecutar en un segundo.

MLF **Micro Lead Frame.** Tipo de Empaquetado de chips.

NTSC **National Television Standards Comitee.** Comité Nacional de Estándar para Televisión, comité que determina el estándar o patrones de radiodifusión y transmisión televisiva en Estados Unidos. En este estándar se muestras 30 fps

OLED **Organic Light Emitting Diode.** Diodo emisor de luz orgánico.

PAL **Phase Alternate Line.** Estándar para televisión utilizado en Europa y Australia. En este estándar se muestras 25 fps.

PDIP **Plastic Dual In-line Package.** Tipo de Empaquetado de chips.

PIC **Peripheral Interface Controller.** Controlador de interfase periférica. Microcontrolador fabricado por Microchip.

PLCC **Plastic Leadless Chip Carrier.** Tipo de Empaquetado de chips.

RF **Radio Frequency.** Radio Frecuencia. frecuencia de onda radiofónica a través de la cual es emitida una transmisión.

RGB **Red Green Blue.** Rojo verde azul, colores básicos utilizados para la creación de una imagen en colores.

RISC **Reduced Instruction Set Computer.** Procesadores que contienen una colección reducida de instrucciones que les permite lograr una velocidad de procesamiento de intrucciones más alta.

SIRC **Sony Infrared Coding.** Modulación DPWM utilizada por los controles remotos Sony.

SRAM **Static Random Access Memory.** Memoria de solo lectura estática (cuida su contenido sin necesidad de señal del procesador central).

SUT **Start-up Time.** Los 2 bits que determinan el retardo o tiempo que se deja pasar luego de energizar el ATmega para que éste comience a funcionar y así darle mayor estabilidad.

TQFP **Thin Quad Flat Pack.** Tipo de Empaquetado de chips.

TTL **Transistor Transistor Logic.** tipo común de circuito digital cuyo output se deriva de dos transistores.

Sangolquí, ……..

ELABORADO POR:

_____

Antonio Ruales R.

El Decano

El Secretario Académico

_____

Ing. Xavier Martínez Carrera
Tcrn. De E.M.

_____

Dr. Jorge Carvajal R.