UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MECANICA Y ELECTRICA

DIVISION DE ESTUDIOS DE POST-GRADO

MICROCONTROLADORES, HERRAMIENTAS DE DISEÑO Y SIMULACIÓN PARA EL DESARROLLO DE PROTOTIPOS

POR

ING. JUAN MANUEL GARCÍA ZÚÑIGA.

TESIS
EN OPCION AL GRADO DE MAESTRO EN CIENCIAS DE LA INGENIERIA ELECTRICA CON ESPECIALIDAD EN ELECTRONICA.

SAN NICOLAS DE LOS GARZA, N.L., ENERO DEL 2008

UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MECANICA Y ELECTRICA

DIVISION DE ESTUDIOS DE POST-GRADO

MICROCONTROLADORES, HERRAMIENTAS DE DISEÑO Y SIMULACIÓN PARA EL DESARROLLO DE PROTOTIPOS.

POR

ING. JUAN MANUEL GARCÍA ZÚÑIGA.

EN OPCION AL GRADO DE MAESTRO EN CIENCIAS DE LA INGENIERIA ELECTRICA CON ESPECIALIDAD EN ELECTRONICA.

SAN NICOLAS DE LOS GARZA, N.L., ENERO DEL 2008.
Universidad Autónoma de Nuevo León

Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica
División de Estudios de Posgrado

Los miembros del Comité de Tesis recomendamos que la tesis "**Microcontroladores, herramientas de diseño y simulación para el desarrollo de prototipos**", realizada por el alumno(a) Juan Manuel García Zúñiga con numero de matricula **1032008** sea aceptada para su defensa como opción al grado de Maestro en Ciencias De La Ingeniería Eléctrica Con Especialidad En Electrónica

El Comite De Tesis.

_____
Asesor
M.C José Manuel Rocha Núñez

_____
Revisor
M.C Juan Ángel Garza Garza

_____
Revisor
M.C Guadalupe Ignacio Cantú Garza

V.o B.o

_____
Dr. Guadalupe Alan Castillo Rodríguez
División de Estudios de Posgrado

Ciudad Universitaria a 8 De Enero del 2008.

AGRADECIMIENTOS

A MIS PADRES

Andrés García Reyes y Juana Zúñiga Morales, por el gran esfuerzo que hicieron por educarme, por el buen ejemplo que siempre me dieron y por todos esos momentos que de una u otra forma siempre estuvieron y están cerca de mí para aconsejarme o apoyarme.

CONTENIDO

I N T R O D U C I Ó N 1
1.3 Importancia del estudio 3
1.4 Historia 3
1.5 Limites del estudio 3
1.6 Justificación de la tesis 4
1.7 Metodología 4
MICROPROCESADORES Y MICROCONTROLADORES 6
2.1 Programas de apoyo para mejorar la Educación Básica 6
2.2 ¿Porque Elegir Un Microcontrolador? 7
2.3 Diferencia Entre Microcontrolador Y Microprocesador 9
2.4 Arquitecturas 9
2.5 MEMORIA 11
2.6 Tipos de Memoria. 11
2.7 UNIDAD ARITMETICO LOGICA (ALU) 13
2.8 RELOJ (CLOCK) 15
2.9 UNIDADES DE ENTRADAS Y SALIDAS 16
MICROCONTROLADOR AVR 17
3.1 Microcontrolador AVR 17
3.2 Arquitectura 17
3.3 Memoria 18
3.4 Interfase JTAG 19
3.5 Reloj 20
3.6 Ejecución de operaciones 21
3.7.1 Registro directo (Un Registro) 22
3.7.2 Registro directo (Dos Registros) 24
PERIFÉRICOS DEL MICROCONTROLADOR 27
4.1 Convertidor Analógico – Digital 27
4.1.1 Tiempo de conversión: 27
4. 1.2 Referencias de voltaje. 28
4.1.3 Definiciones. 28
4.1.4 Configuración del ADC 30
4.2 PWM 32
4.2.1 Tipos de PWM 33
4.2.2 Características de PWM con corrección de fase 34
4.4.3 Configuración de la PWM. 36
4.3 USART 38
4.3.1 NORMA RS-232 38
4.3.2 CONFIGURACION DE LA USART 39
4.4 SPI 44
HERRAMIENTAS DE DISEÑO 47
5.1 ENSAMBLADOR 47
5.1.1 Funcionamiento 47

5.1.2 Tipos de ensambladores 48
5.1.3 ventajas de los lenguajes de ensamblador. 49
5.2 COMPILADOR 50
5.2.1 Tipos de compiladores 51
5.3 DEPURADOR 52
5.3.1 Usos 52
5.4 EMULADOR 55
SISTEMAS OPERATIVOS 56
6.1 Que Es Un Sistema Operativo 56
6.2 Tipos de los Sistemas Operativos 57
6.3 Principio de funcionamiento de un Sistema Operativo 58
6.4 Sistemas operativos de tiempo real 58
6.4.1 Características generales 58
6.5 RTOS mas comunes 59
LENGUAJES DE PROGRAMACIÓN 60
7.1 LENGUAJE DE PROGRAMACIÓN. 60
7.2 Lenguajes de bajo nivel. 60
7.3 Lenguajes de alto nivel, ... 61
7.4 Lnguaje C. 62
7.4.1 Preprocesado: 63
7.4.2 Compilación: 63
7.4.3 Enlazado: 63
7.4.4 Ventajas: 64
7.5 lenguaje C++. 64
7.5.1 Ventajas y desventajas 66
7.6 PASCAL. 67
7.6.1 Tipos de datos: 67
7.7 BASIC. 68
7.8 Lenguajes Visuales 68
EJEMPLOS 69
8.1 Introducción 69
8.2 Ejemplo 1 (Led intermitente) 71
8.2.1 Descripción: 71
8.2.2 Diagrama de flujo: 71
8.2.3 Código ensamblador 72
8.2.4 Código C (Avr Studio) 73
8.2.5 Programa en FlowCode 75
8.2.6 Código generado por FlowCode 76
8.2.7 Simulación en Flowcode 80
8.3 Reloj Digital 81
8.3.1 Diagrama De Flujo 81
8.3.2 Simulación 82
SISTEMAS DE DESARROLLO 83
9.1 ¿Que es un sistema de desarrollo? 83
9.2 Sistemas De Desarrollo Comerciales 84
9.3 Programadores Universales, ¿Son una necesidad para el estudiante? 86

9.4 Construya su propio sistema de desarrollo. 87
APLICACIONES 89
10.1 Control de matriz de leds 89
10.1.2 Código 93
10.1.3 Diagrama esquemático 98
10.2 Control de matriz de interruptores y matriz de leds. 99
10.2.1 Funcionamiento 99
10.2.2 Código fuente 101
10.2.3 Diagrama esquemático 111
10.3 Aplicación 1 (Alarma con pantalla LCD reprográmable) 112
10.3.1 Descripción: 113
10.3.2 Funcionamiento: 113
10.3.3 Diagrama de Flujo 114
10.3.4 Código Fuente: 115
10.3.5 Diagrama en PROTEUS 123
FREERTOS 124
11.1 Que es FreeRTOS? 124
11.2 Filosofía de diseño de FreeRTOS: 125
11.3 Características De Las Tareas 125
11.3.1 Resumen de Tareas 126
11.4 Características De Las Co-Rutinas 126
11.5 Semáforo. 127
CONCLUSIONES 128
12.1 Conclusiones 128
BIBLIOGRAFÍA 128
INDICE DE FIGURAS 128
INDICE DE TABLAS 128

CAPITULO 1

I N T R O D U C I Ó N

1.1 Introducción

Hoy en día existen diversas opciones para la automatización de procesos, cada opción tiene ciertas características y cada una esta enfocada en facilitar el diseño, la implementación y reducir el tiempo de desarrollo así como facilitar las pruebas de calidad del sistema desarrollado.

La automatización es una de las áreas con más demandas en la actualidad. Conforme avanza el tiempo se demanda más de los diseñadores, los cuales deben estar al día con nuevas tecnologías, todo con el objetivo de optimizar al máximo los procedimientos de desarrollo.

Nuevas tecnologías emergen día a día nuevo software que permiten el encapsulamiento de código, la modularidad, y la reusabilidad de código. Herramientas como la programación visual, la programación orientada a objeto y la implementación de sistemas operativos de tiempo real son cada día más comunes en pequeños sistemas basados en microcontroladores.

Tecnologías como el lenguaje ensamblador y acceso directo al hardware quedan en una capa de nivel baja, y solo se accesa a ella cuando se requiere un alto desempeño en algunas rutinas de la aplicación.

Software para programar en forma Visual ha Ganado mucha popularidad debido a que el tiempo de aprendizaje para desarrollar en estos lenguajes es mucho menor comparado con lenguajes de alto nivel como lo es C/C++.

El software evoluciona a la par del hardware, cada nueva generación de microcontroladores cada vez cuenta con mas capacidad como velocidad de procesamiento, capacidad de almacenamiento permanente y temporal, así como cada vez se encapsulan mas periféricos en un solo chip.

En este estudio se pretende presentar al estudiante un panorama muy amplio sobre las arquitecturas, herramientas para el desarrollo con microcontroladores.

También se pretende acabar con el paradigma o la creencia de asociar microprocesador o microcontrolador con el termino ensamblador, ya que en la actualidad un microprocesador es una microcomputadora muy potente la cual puede tener un sistema operativo y la cual puede ser programada en lenguaje de alto nivel.

Durante la elaboración de la tesis se incluyeron un par de capítulos sobre el uso de un entorno de desarrollo llamado “AVR Studio”, un compilador profesional libre llamado GCC en su presentación conocida como WinAVR y un simulador profesional con licencia comercial. Con estas tres herramientas un estudiante tiene un laboratorio completo de diseño encapsulado en una computadora.

Se añadió también documentación sobre el uso de depuradores y programadores de bajo costo.

Toda esta información fue recopilada y esta enfocada a ampliar los conocimientos del estudiante así como abrir la visión sobre las herramientas disponibles (comerciales y libres) para desarrollo de prototipos profesionales de mayor complejidad en un menor tiempo.

1.3 Importancia del estudio

En nuestro país existen problemas de calidad educativa, y de falta de recursos en los distintos laboratorios de las áreas de sistemas digitales y control.

Se analizaran algunas de las distintas herramientas de bajo costo o disponibles en forma gratuitas. Se plantearan como no es necesario una gran inversion para poder

1.4 Historia

Por mucho tiempo el estudio de los sistemas con microprocesador en los sistemas educativos se ha visto de un forma tan detallada invirtiendo mucho tiempo en el estudio de la arquitectura interna de un microprocesador en especifico, esto mas que ventajas tiene consecuencias negativas, debido a que el estudiante en cada generación no llegaba mas allá de practicas simples debido a la limitante del tiempo.

Con el paso de tiempo han emergido herramientas sofisticadas para diseñar con microcontroladores con el enfoque o la idea de trabajar con el microcontrolador como si este fuera una PC (Con la única diferencia de la capacidad de almacenamiento y procesamiento limitado)

1.5 Limites del estudio

Esta tesis cubre diferentes áreas del diseño de un sistema de control, se habla del microcontrolador, su arquitectura y las herramientas de alto nivel para la programación de los mismos. Se documenta sobre algunos de los sistemas operativos de tiempos real; así como también se hace referencia a las diferentes herramientas de depuración. En general se cubren muchos temas de los cuales de cada uno de ellos se podría escribir un libro. Se asume que el estudiante esta familiarizado con algún lenguaje de programación y que cuenta con conceptos básicos de electrónica y sistemas digitales. Esta tesis presenta cada muchos de los temas como

información y asume que el estudiante la usara como guía soportándose en textos especializados en cada unos de los temas.

1.6 Justificación de la tesis

Con la introducción de nuevas y mas potente herramientas para el desarrollo con microcontroladores y presentado la metodología del desarrollo de sistemas de control basados en microcontroladores desde un punto de vista mas simple y similar al usado en la programación de computadoras personales se incitara y motivara al alumno de las clases de programación de microcontroladores a ver los sistemas dedicados como computadoras personales, mostrando que herramientas como las usadas en la programación de computadoras personales pueden ser usadas para la programación de microcontroladores.

Se mostrara como tecnologías como la programación orientada a objetos, la programación visual y diversas tecnologías y sistemas usados en el desarrollo de grandes proyectos de software para computadoras personales también pueden ser usados para desarrollo proyectos complejos basados en microcontroladores.

Se pretende también mostrar al estudiante las diferentes herramientas profesionales de bajo costo disponibles y como puede tener en una computadora personal un sistema de desarrollo profesional con una inversión muy baja.

1.7 Metodología

- Encuesta a alumnos participantes de concursos de creatividad
- Reuniones y sondeo a maestros de las clases de microprocesadores y microcontroladores
- Evaluación de software de alto nivel, visual y generadores de código
- Desarrollo practico de prototipos
- Análisis de tiempo contra desarrollo de prototipo usando herramientas antiguas contra el uso de herramientas de diseño modernas.
- Investigación en la tendencia de las herramientas de diseño para sistemas digitales y sistemas basados en microcontroladores

CAPITULO 2

MICROPROCESADORES Y MICROCONTROLADORES

2.1 Programas de apoyo para mejorar la Educación Básica

El microprocesador es un circuito integrado que contiene la Unidad Central de Proceso, también llamada procesador, de un computador. La Unidad Central de Proceso está formada por la Unidad de Control, que interpreta las instrucciones, y el Camino de Datos, que las ejecuta.

Hacia el exterior del dispositivo salen las líneas de los tres buses principales, que son: bus de direcciones, bus datos y bus control, estos buses nos permiten la interconexión con la memoria (RAM, ROM,PROM, etc) y puertos de entrada y salida. Podemos definir un microprocesador como un sistema abierto en el cual se tiene completa libertad sobre la configuración final de los dispositivos básicos con los que interactuara.

En base a lo anterior podemos decir que un microcontrolador es un microprocesador encapsulado en un chip con los componentes básico necesario para funcionar con un sistema. Estos componentes básicos serian:

Oscilador

Memoria

Puertos De Entrada y Salida

En los siguientes diagramas se aprecia un diagrama a bloques de un sistema con microcontrolador y un sistema con microprocesador

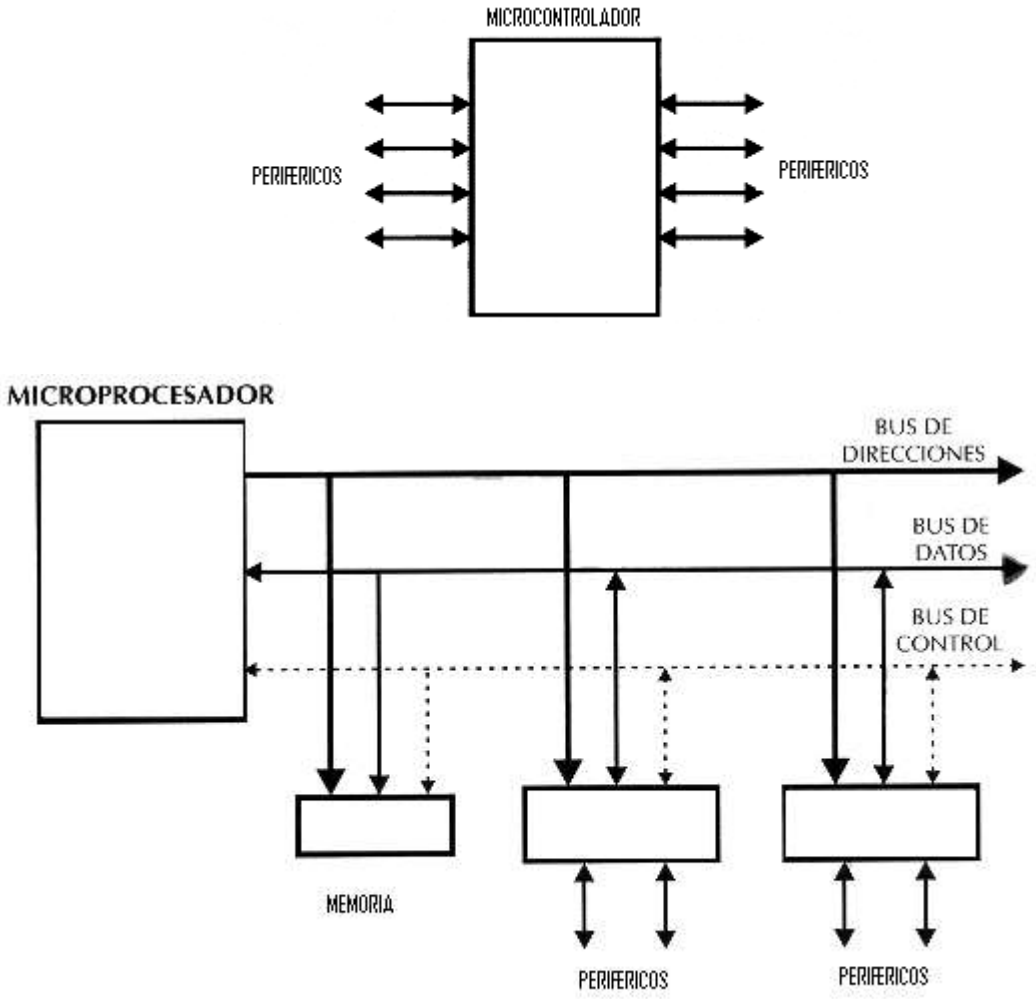


Figura 2.1

2.2 ¿Porque Elegir Un Microcontrolador?

El microprocesador es un circuito potente y de aplicaciones múltiples, una de sus ventajas es que puede ser configurado a las necesidades del diseño añadiendo solo los periféricos necesarios para el funcionamiento del sistema.

La ventaja mencionada en la sección anterior es una gran desventaja para un estudiante con acceso limitado a un laboratorio profesional en el que se podría acceder a equipo para armar circuitos impresos complejos. Comparando un sistema con

microprocesador contra un sistema con microcontrolador encontraremos las siguientes necesidades al trabajar con el microprocesador.

- Microprocesador
- Circuito De Memoria RAM
- Circuito De Memoria ROM
- Circuito ADC
- Circuito DAC
- Circuito Oscilador
- Circuito Controlador De Interrupciones
- Circuito Controlador De Puerto Serie
- Circuito De Reloj De Tiempo Real

Ahora asumiendo que cada circuito tiene de 20 a 40 terminales y para armar un circuito con microprocesador usáramos los componentes listados, estaríamos hablando de interconectar un circuito de 200 a 400 terminales… Armar este circuito en una tablilla de experimentos es una tarea titánica.

Ahora listaremos los componentes mínimos para armar un circuito con microcontrolador:

- Microcontrolador

Los microcontroladores los encontramos en diferentes encapsulados con una cantidad de patillas que van desde 8 hasta mas de 100, con características y periféricos encapsulados muy diversos, por lo que seguramente encontraríamos alguno que se adapte a nuestras necesidades.

2.3 Diferencia Entre Microcontrolador Y Microprocesador

Un microcontrolador es un sistema completo con unas prestaciones limitadas que no pueden modificarse y que puede llevar a cabo las tareas para las que ha sido programado de forma autónoma, también cabe decir que es un sistema autónomo e independiente. Un microprocesador, en cambio, es simplemente un componente que conforma el microcontrolador, que lleva a cabo ciertas tareas que analizaremos más adelante y que, en conjunto con otros componentes, forman un microcontrolador.

Un microcontrolador es un solo circuito integrado que contiene todos los elementos electrónicos que se utilizaban para hacer funcionar un sistema basado con un microprocesador; es decir contiene en un solo integrado la Unidad de Proceso, la memoria RAM, memoria ROM, puertos de entrada, salidas y otros periféricos, con la consiguiente reducción de espacio.

2.4 Arquitecturas

65xx

ARM

Altera Nios, Nios II

AVR (puramente microcontroladores)

EISC

DEC Alpha

Intel

Intel 8970, 8085,

Zilog Z80

Intel Itanium

MIPS

Motorola 68xxx, ColdFire

Motorola 68000

OpenRISC

PA-RISC

SPARC

INMOS Transputer

x86

Cambridge Consultants XA

2.5 MEMORIA

Dispositivo ó conjunto de dispositivos capaces de almacenar información. Debe disponer de un sistema de recuperación de información y de un sistema de direccionamiento de la misma. La capacidad de memoria es la cantidad de información que puede almacenar en una memoria en concreto. Hay dos tipos de capacidad:

Útil: LA que puede almacenar el usuario.

Bruta: Número total de unidades de información (bits ó bytes) que pueden ser almacenadas en un dispositivo.

2.6 Tipos de Memoria.

La memoria se puede dividir en dos tipos:

- Memoria RAM
- Memoria ROM

Memoria RAM (Memoria de Acceso Aleatorio)

Es la memoria de almacenamiento principal en donde la PC guarda los datos que está utilizando en ese momento.

Los chips RAM contienen circuitos que sirven para almacenar temporalmente instrucciones de programas y datos. El computador divide un chip de RAM en varias localidades de tamaño igual, estas localidades tienen una dirección única, de manera que la computadora puede distinguirlas cuando se le ordena que escriba o lea información. Dicha información almacenada en la RAM no es más que un patrón de corriente eléctrica que fluye por los circuitos del chip. Esto quiere decir que si se interrumpe la energía eléctrica, por cualquier razón, la computadora olvida de inmediato todo lo que había almacenado. En términos técnicos se dice que la memoria RAM es volátil, ya que la información que contiene no se conserva de manera permanente.
La memoria RAM no es un mero almacén de datos, es también un lugar de trabajo: en

esta memoria RAM se puede leer y escribir información, aunque se borra en cuanto se apaga el ordenador.

DRAM (Dynamic RAM)

Este es el tipo de RAM más comúnmente usado. Internamente está compuesto por condensadores de pequeña capacidad, que almacena la información mediante la carga y la descarga equivalen a 1 y 0 lógicos, respectivamente.

Son de bajo costo, pero tiene el inconveniente de que pierden su carga (es decir son volátiles), y por tanto la información, demasiado rápido por lo que deben ser constantemente "refrescados" con una nueva carga.

La DRAM tiene sus contras: la transferencia de la información que va desde la memoria hasta el procesador "es más lenta"; requiere de Caché para mejorar su desempeño; usa más energía, lo cual implica una menor duración de la batería para los usuarios de computadoras portatiles.

Existen varios tipos de DRAM, cada placa madre esta preparada para uso de varios de estos tipos, así que antes de comprar consulta el manual de la placa madre.

SRAM (RAM estatica)

Su diseño interno está hecho en base a transistores que almacenan la información cuando son polarizados en corte o saturación, correspondientes a los estados lógicos 1 y 0, respectivamente permaneciendo en esta condición hasta que se cambie la información. No necesitan ser “refrescados”, son muy veloces pero más costosos que los DRAM.

Memoria ROM (Memoria de Solo Lectura)

Esta memoria guarda información de manera permanente, es decir no es volátil como la memoria RAM. La computadora puede leer la información almacenada en ella, pero no escribir sobre esta memoria.

Todas las computadoras cuentan con dispositivos de ROM que contienen las instrucciones de arranque y otra información crítica. La información en la ROM se graba

permanentemente cuando se fabrica el computador, de modo que siempre este disponible cuando este opere.

2.7 UNIDAD ARITMETICO LOGICA (ALU)

Es una de las partes de la UPC. La **Unidad Aritmético Lógica** (**UAL**), o **Arithmetic ogia Unit** (**ALU**), es un circuito digital que calcula operaciones aritméticas (como adición, substracción, etc.) y operaciones lógicas (como OR, NOT, XOR, etc.), entre dos números y lo que entrega es el resultado de las operaciones y además algunas particularidades del resultado (desbordamiento, signo, valor=0,…).

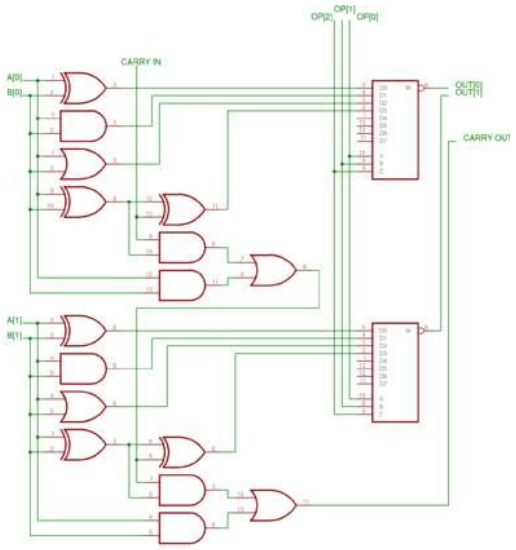


Figura 2.2

Una ALU simple de 2 bits que hace operaciones de AND, OR, XOR y adición.

La ALU se compone básicamente de: Circuito Operacional, Registros de Entradas, Registro Acumulador y un Registro de Estados, conjunto de registros que hacen posible la realización de cada una de las operaciones.

La mayoría de las acciones de la computadora son realizadas por la ALU. La ALU toma datos de los registros del procesador. Estos datos son procesados y los resultados de esta operación se almacenan en los registros de salida de la ALU. Otros mecanismos mueven datos entre estos registros y la memoria.

Una unidad de control controla a la ALU, al ajustar los circuitos que le dicen a la ALU qué operaciones realizar. Cuanto más compleja sea la operación, tanto más costosa será la ALU, más espacio usará en el procesador, y más energía disipará, etc...

Los datos sobre los que se realizan las operaciones se denominan operandos. Al elemento encargado de ejecutar las operaciones se le denomina operador, y esta formado por una serie de circuitos electrónicos que son capaces de sumar dos números binarios o hacer las operaciones lógicas elementales.

La mayoría de las ALU pueden realizar las siguientes operaciones:

- Operaciones aritméticas de números enteros (adición, sustracción, multiplicación y división)
- Operaciones lógicas de bits (AND, NOT, OR, XOR)
- Operaciones de desplazamiento de bits (Desplazan o rotan una palabra en un número específico de bits hacia la izquierda o la derecha, con o sin extensión de signo). Los desplazamientos pueden ser interpretados como multiplicaciones o divisiones por 2.

Para que el operador realice la operación, los operandos se llevan a la UAL y se guardan en unos registros denominados registros de trabajo. El resultado de la operación se guarda también en un registro antes de ser llevado a la memoria o a la Unidad de Entradas y Salidas. Frecuentemente se utiliza un mismo registro para guardar uno de los operandos y, también, el resultado, denominado registro Acumulador.

El operador, además de calcular el valor de la operación, modifica el registro de estado según el resultado de la operación.

2.8 RELOJ (CLOCK)

La mayoría de los microcontroladores requiere dos patas para utilizarlas en una función llamada “clock”. La traducción literal de esta palabra es “Reloj” pero los informáticos suelen usar directamente el término en inglés. La necesidad del clock es evidente. Si un micro es un dispositivo que sigue una serie de pasos de programa debemos decirle a que ritmo debe leer esos pasos. Así pues, llamamos al control del ritmo en que correrá nuestra programación del microprocesador reloj. Ese ritmo se le puede dar con un circuito LC, y cuya frecuencia de trabajo es de 1 MHz (ritmo de 4 uS por paso de programa) a condición de que el circuito interno del micro se encargue de generar una señal excitando al circuito LC externo. Pero existe un componente electrónico llamado “cristal” que suplanta al circuito LC con una enorme precisión y que se utiliza junto con dos capacitores de pequeño valor para generar el reloj de los micros.

La mayoría de los dispositivos de lógica secuencial, son de naturaleza síncrona. Es decir, están diseñados y operan en función de una señal de sincronización. Esta señal, conocida como señal de reloj, usualmente toma la forma de una onda cuadrada periódica. Calculando el tiempo máximo en que las señales eléctricas pueden moverse en las varias bifurcaciones de los muchos circuitos de un microprocesador podemos seleccionar un periodo apropiado para la señal del reloj.

Los osciladores más comunes son los de cristal externo y los de oscilador interno. Los de Cristal Externo son mucho más veloces que los de oscilador interno, pero también son más costosos.

2.9 UNIDADES DE ENTRADAS Y SALIDAS

Las entradas y/o salidas están interconectadas con la CPU y con el exterior mediante líneas dedicadas a este propósito ó mediante líneas de propósito general. Por lo general nos sirven para comunicar nuestro sistema con el exterior, mediante el uso de dichas entradas y salidas. Algunos ejemplos de ello son el teclado, el ratón y la impresora que sirven de interconexión con el CPU de una computadora.

Los dispositivos de entrada nos sirven para introducir datos (información) a la computadora para su proceso. Los datos se leen de los dispositivos de entrada y se almacenan en la memoria central o interna del microprocesador.

Los dispositivos de salida nos permiten representar los resultados (salida) del proceso de datos. El dispositivo de salida típico es la pantalla o monitor de la computadora. Aunque podemos aprovechar estas salidas para alimentar un motor, para controlar la electricidad, etc.

CAPITULO 3

MICROCONTROLADOR AVR

3.1 Microcontrolador AVR

Esta familia está basada en una nueva arquitectura RISC que incorpora memoria flash para el programa y memoria EEPROM para los datos. Además esta arquitectura fue diseñada para ser totalmente compatible con lenguaje C permitiendo trabajar en alto nivel dejando la tarea de optimización del código objeto al ensamblador, aumentando la eficiencia en la programación junto con un juego de instrucciones mucho más poderoso que los existentes en el mercado, con instrucciones que se ejecutan en un solo ciclo de reloj. Además todos los dispositivos de la familia traen puerto serial que permite reprogramar la memoria flash dentro de la aplicación

3.2 Arquitectura

Los dispositivos AVR usan arquitectura RISC (por sus siglas en ingles "Reduced Intruction Set Computing"), esta arquitectura fue diseñada para ejecutar los programas muy rápido con el uso de un juego reducido de instrucciones. La mayor parte del juego de instrucciones se ejecutan en un solo ciclo de maquina, un microcontrolador con esta arquitectura con un oscilador de * Mhz puede ejecutar cerca de 8 millones de instrucciones por segundo.

El juego de instrucciones AVR está implementado físicamente y disponible en el mercado en diferentes dispositivos, que comparten el mismo núcleo AVR pero tienen distintos periféricos y cantidades de RAM y ROM: desde el microcontrolador de la familia *Tiny AVR* ATtiny11 con 1KB de memoria flash y sin y 8 patillas, hasta el microcontrolador de la familia *Mega AVR*ATmega2560 con 256KB de memoria flash, 8KB de memoria RAM, 4KB de memoria EEPROM, conversor análogo digital de 10 bits y 16 canales, temporizadores, comparador analógico, JTAG, etc. La compatibilidad entre los distintos modelos es preservada en un grado razonable.

Los microcontroladores AVR tienen un "pipeline" con dos etapas (cargar y ejecutar), que les permite ejecutar la mayoría en un ciclo de reloj, lo que los hace relativamente rápidos entre los microcontroladores de 8-bit.

3.3 Memoria

Memorias del tipo Flash, Eprom y SRAM están integradas en un solo circuito con el fin de remover la necesidad de memoria externa, aunque en ocasiones siempre existe la posibilidad de necesitar memoria externa, algunas serie de la familia AVR soporta memoria externa.

Memoria Flash:

Las instrucciones del programa son almacenadas en este tipo de memoria, aunque la memoria es de 8bits las instrucciones son de 16 bits.

El tamaño de la memoria del programa es generalmente indicado en el nombre del dispositivo, ejemplo: Atmega16 es un dispositivo de 16 KB de flash.

Memoria interna de datos

El espacio de memoria de datos consiste de registros de archivo, registros de entrada y salida y memoria SRAM, esta última es usada para almacenar variables del programa que no caben en los registros de archivo. La memoria SRAM se encuentra despues de los registros de archivo. Aunque la SRAM y los registros de archivo se encuentran en secciones separadas ambos pueden ser accesados como si ambos fueran SRAM

Registros internos

Los registros de archivo se encuentran a partir de la posición 0000 seguido de 64 posiciones llamadas Registros internos. Estos registros tienen funciones especificas, algunas de ellas pueden ser configuración de los puertos de entrada y salida, configuración del puerto RS-232, configuración de del convertidor analógico digital, et.

EEPROM

Algunos dispositivos AVR cuentan con una pequeña sección de memoria EEPROM en la cual se pueden almacenar datos de configuración para la aplicación, los cuales podrian ser mantenidos aun después de una desconexión de la energía.

Esta memoria no esta dentro del mapa de memoria del dispositivo y solamente puede ser asesada de forma similar a la que se accedan un dispositivo externo, esto es usando un registro especial para las operaciones de lectura/escritura.

3.4 Interfase JTAG

La interfase JTAG provee acceso a las funciones de depuración directamente en el dispositivo esto mientras el el circuito ejecuta su programa en el sistema diseñado. La interfase JTAG permite acceder en forma directa a memoria y registros internos, fijar puntos de interrupción en el código, y hacer una ejecución del programa paso a paso con el fin de observar el comportamiento del sistema.

Atmel provee una serie de interfase JTAG

1. La interfase JTAGICE se conecta a la computadora por medio del puerto serie. Esta interfase es una herramienta profesional y con un costo no muy accesible a para un estudiante (~ $4000 pesos). Aunque esta interfase es muy popular y aun sigue siendo comercializada pronto estará obsoleta ya que los nuevos dispositivos de Atmel ya no son compatibles con ella.
2. Interfase JTAGICE mkII esta es una nueva interfase que reemplaza al modelo JTAGICEII, esta interfase cuenta con puerto USB para conectarla al PC y esta soportada por el popular sistema de desarrollo AVR Studio. Existen versiones económicas compatibles de bajo costo (~ $500 pesos) con prestaciones similares pero con la limitante de un reducido numero de dispositivos soportados.
3. La interfase AVR Dragon es una interfase de bajo costo la cual sustituye en algunos casos a la interfase JTAGICE mkII, esta interfase soporta ISP y debugWire para dispositivos con 32KB de memoria de programa o menos.

Existen en el mercado una gran cantidad de dispositivos compatible JTAG de bajo costo así como muchos sitios en el Internet con instrucciones de cómo construir uno, la mayor partes de estos dispositivos compatibles no soportan DebugWire y soportan una serie limitada de dispositivos, entre los dispositivos mas comúnmente soportados se encuentran el Atmega 16, Atmega 32, Atmega 64 y Atmega 128.

En resumen podemos decir que tener un depurador con interfase JTAG es una herramienta necesaria y ahora accesible a que pueda ser adquirida por un estudiante.

3.5 Reloj

Proveer una señal de reloj al procesador del AVR es otro proceso importante de diseño. La frecuencia del reloj determina el ritmo con el que los programas serán ejecutados. En el AVR, la mayoría de las instrucciones son ejecutadas en un ciclo de reloj, algunas usan dos ciclos de reloj, y para algunos otros toma cuatro o cinco ciclos de reloj(Por ejemplo los PICs de Microchip).

El reloj del AVR puede ser operado con una variedad de componentes. Muchos AVR están disponibles con un oscilador interno RC. Si la aplicación no demanda mucha precisión, y si la frecuencia nominal de 1MHz a 5 v es suficiente para la aplicación, entonces esta opción puede ser utilizada. De lo contrario se puede utilizar el generador interno con otro componente como un cristal de cuarzo o un dispositivo cerámico. Si es posible incluso se puede utilizar un generador de señal TTL.

Una opción muy practica para suministrarle un reloj al AVR es usar el oscilador interno RC disponible en algunos de los procesadores (AT90S1200, 2343, Tiny22). El AT90S1200 es distribuido con el oscilador RC deshabilitado y puede ser reprogramado con la ayuda de un programador paralelo para habilitar bit RCEN así como seleccionar el oscilador RC. Sin embargo, el AT90S1200 puede ser usado teniendo el oscilador RC habilitado.

3.6 Ejecución de operaciones

El procesador del AVR es manejado por el reloj del sistema, que puede ser tomado del exterior o, si está disponible y habilitado, se puede usar un reloj interno RC. Este reloj del sistema sin ninguna división es usado directamente para todos los accesos dentro del procesador. El procesador tiene un conducto de dos etapas, una función traer/decodificar es llevada a cabo concurrentemente con la instrucción de ejecución. Esto es ilustrado en la figura 3.1.

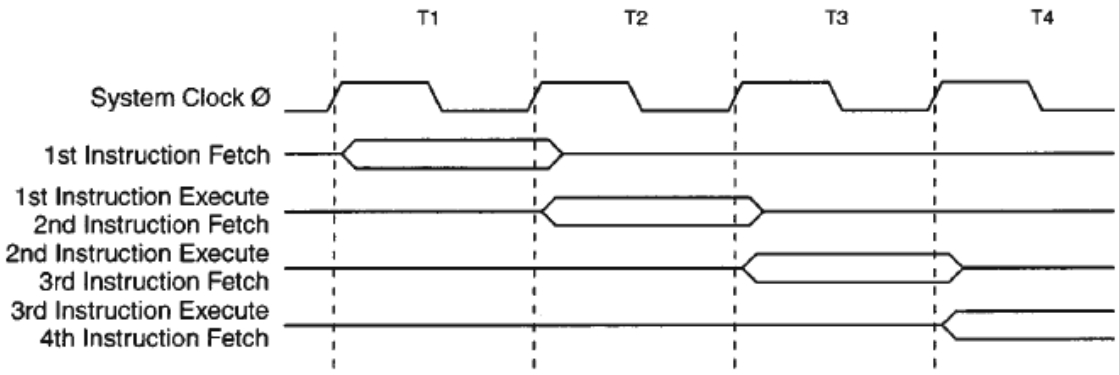


Figura 3.1

Una vez que la instrucción es traída, si es una instrucción relacionada con la ALU, entonces puede ser ejecutada por la ALU como se muestra en la figura 3.1 en un solo ciclo.

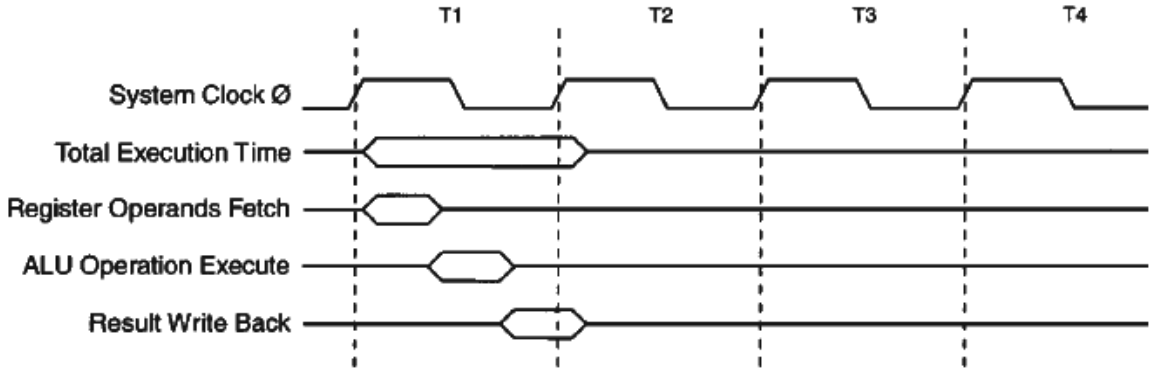


Figura 3.2

Por otro lado, la memoria SRAM toma dos ciclos, como se muestra en la figura 3.7. Esto es debido a que el acceso SRAM usa un indicador de registro para la dirección de la SRAM. El apuntador de registros es uno de los siguientes (X, Y o Z pares de registro). El primer ciclo de reloj es necesario para acceder al

archivo de registro y para operar el indicador de registro (las instrucciones de acceso de la SRAM permiten predireccionar incrementando la operación del indicador de registros). Al final del primer ciclo de reloj, el ALU ejecuta el cálculo, y entonces esta dirección es usada para acceder la locación de la SRAM y escribir en ella (o leer de ella en el registro de destino), como se ilustra en la figura 3.3.

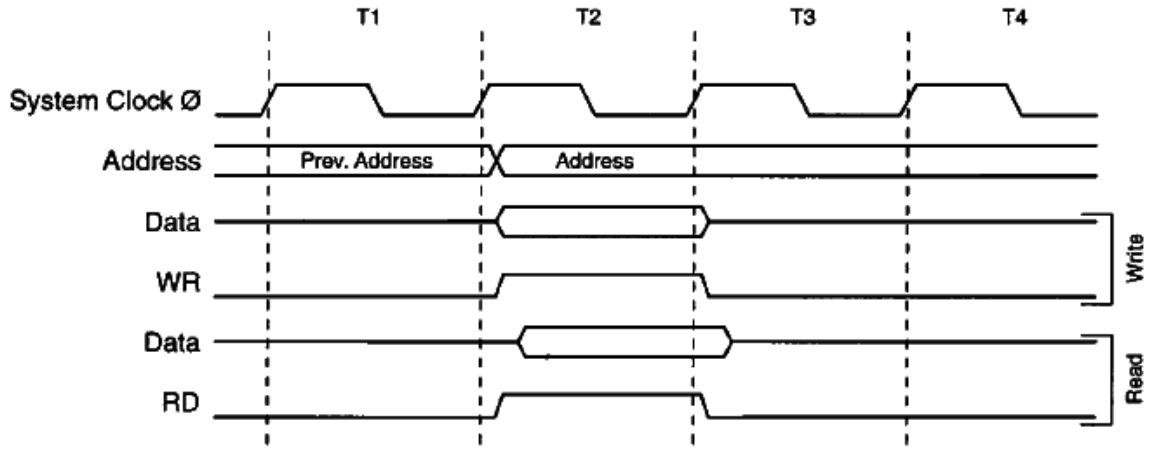


Figura 3.3

3.7 Modos de direccionamiento

Las diferentes instrucciones del AVR pueden ser catalogadas en 10 diferentes modos de direccionamiento. Cada instrucción es un código de operación que indica a la lógica de control del procesador que hacer. La otra parte de la instrucción es el operando en el que opera el código de operación.

3.7.1 Registro directo (Un Registro)

La instrucción de registro directo puede operar en cualquiera de los registros del archivo de registros. Este lee el contenido de los registros, opera en el contenido de los registros, y luego almacena el resultado de la operación en el mismo registro. La figura 3.4 ilustra el origen y destino de este tipo de instrucciones.

El formato de la instrucción es: Mnemotecnia Registro de Destino.

A continuación un ejemplo de estas instrucciones. Rd es cualquier registro del archivo de registros y es el registro de destino (también es la fuente) para la operación.

COM Rd: Complemento (invierte todos los bits) del registro Rd y es almacenado de nuevo en el registro Rd.

INC Rd: Incrementa el contenido de Rd con un uno.

DEC Rd: Decrementa el contenido de Rd de uno en uno.

TST Rd: Prueba si existe un cero o o contenido negativo en el registro Rd.

CLR Rd: Carga $00 en el registro Rd.

SER Rd: Carga $FF en el registro Rd.

LSL Rd: Recorre el contenido del registro Rd un lugar a la izquierda. Un “cero” es colocado en la posición cero, y el contenido del bit 7 es copiado a la bandera de acarreo.

LSR Rd: Recorre el contenido del registro Rd un lugar a la derecha. Un “cero” es colocado en el bit 7, y el contenido del bit 0 es copiado en la bandera de acarreo-

ROL Rd: Rota el contenido de la izquierda del registro Rd directo al acarreo. La bandera de acarreo se va al bit cero, y el bit 7 se va al acarreo.

ROR Rd: Rota el contenido de la derecha del registro directo al acarreo. La bandera de acarreo se va al bit 7, y el bit cero se va al acarreo.

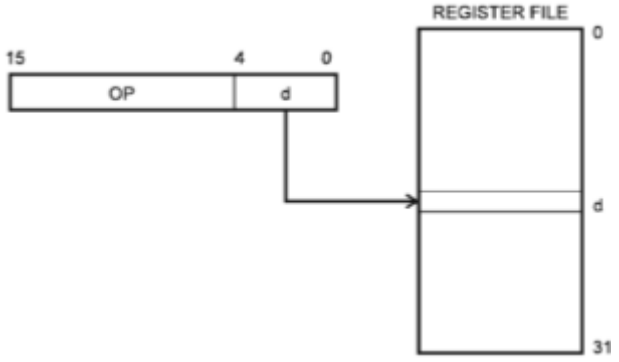


Figura 3.4

ASR Rd: Recorre de forma aritmética el contenido de Rd, manteniendo el bit 7 en el mismo lugar. Esto se lleva a cabo dividiendo entre dos para cada corrimiento.

Swap Rd: Intercambia los nibles del registro Rd.

3.7.2 Registro directo (Dos Registros)

En este tipo de instrucciones se involucra a dos registros.los registros son llamados registro de origen, Rs, y registro de destino, Rd. La instrucción lee los dos registros, opera en sus contenidos y almacena los resultados en el registro de destino. La figura 4.2 ilustra el origen y destino para este tipo de instrucciones.

Ejemplo de estas instrucciones son: ADD Rd, Rs; SUB Rd, Rs; AND Rd, Rs; MOV Rd, Rs; OR Rd, Rs;

Ejemplo de estas instrucciones son: ADD Rd, Rs; SUB Rd, Rs; AND Rd, Rs; MOV Rd, Rs; OR Rd, Rs;

Entrada/Salida Directa (I/O)

Estas instrucciones son usadas para las entradas/salidas. Los registros de entrada/salida pueden ser usados solo con las siguientes instrucciones: In Rd, PORTADDRESS; Out PORTADDRESS, Rs.
Rd, Rs pueden ser cualquiera de los 32 registros del archivo de registros, y los registros I/O (entrada/salida) puede ser cualquier registro desde $00 hasta $3F (un total de 64 registros de entrada/salida). La figura 3.5 ilustra como funciona esta instrucción.

Datos Directos

Esta es una instrucción de dos palabras. Una de las palabras es la dirección del espacio de la memoria de datos. Un máximo de 64 kbytes pueden ser usados mediante este tipo de instrucciones.

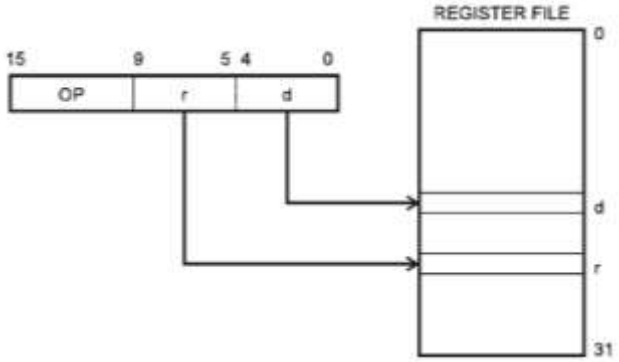


Figura 3.5 Acceso directo de dos registros.

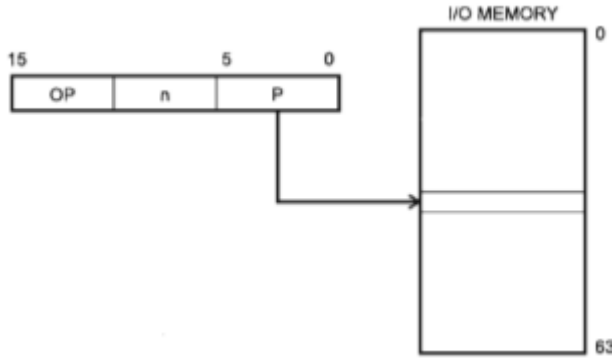


Figura 3.6 Acceso directo de entrada y salida a la memoria.

Ejemplos de estas instrucciones son: LDS RD, K; K es una dirección de16-bit. STS K, Rs;
La figura 3.6 indica como operan las instrucciones de datos directos.

Datos indirectos

Son similares a las instrucciones de datos directos, excepto que estas instrucciones son de una sola palabra, y utiliza un indicador de registro (X, Y o Z) que tiene la dirección base de la memoria de datos. Para la dirección base en el indicador de registro, se puede añadir una compensación, al igual que una operación de incremento o decremento en el indicador de contenido. Ejemplo de estas instrucciones son: LD Rd, X; X es el indicador de registro (registros R26, R27); LD Rd, X+; Rd es el registro de destino y es cargado con el contenido de la memoria de datos situada por el registro X, y después de que se ingresa a la memoria, el registro X es incrementado. ST X, Rs; ST X+, Rs; ST-Y, Rs; La figura 3.6 ilustra como opera una variante de la instrucción de datos indirectos.

Direccionamiento indirecto de programa

En este tipo de instrucciones, el registro Z es usado para indicar al programa de memoria. Hasta 64 kbytes de memoria de programa pueden ser usados con el registro Z de 16 bit. Ejemplo de estas instrucciones son: IJMP y ICALL. La figura 3.7 ilustra como operan las instrucciones de direccionamiento indirecto de programa.

Direccionamiento relativo de programa

Estas instrucciones son del tipo RJMP y RCALL, donde se hace uso de +/- 2k del contador del programa. La figura 3.8 ilustra como opera el direccionamiento relativo de programa.

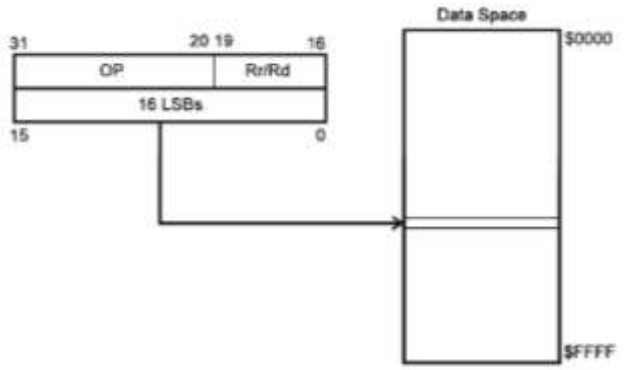


Figura 3.7 Acceso directo a la memoria de datos.

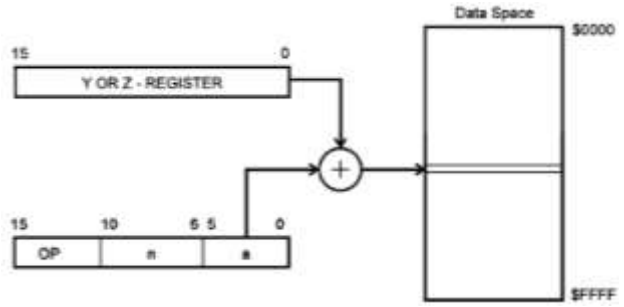


Figura 3.8 Acceso indirecto a la memoria de datos.

CAPITULO 4

PERIFÉRICOS DEL MICROCONTROLADOR

4.1 Convertidor Analógico – Digital

Un convertidor analógico digital (ADC) es un dispositivo capaz de convertir un nivel de voltaje analógico en un determinado valor binario (digital).

Un convertidor analógico-digital establece una relación entre su entrada y salida, dependiendo de la resolución con que este cuente. El valor de la resolución se obtiene, si sabemos, el valor máximo de entrada y la cantidad máxima de salida en dígitos binarios.

El convertidor del Atmega posee una resolución de 10 bits. Utiliza el método de aproximaciones sucesivas y puede ser multiplexado en 6 canales.

Consideraciones del ADC: Para utilizar de mejor manera el convertidor analógico digital del AVR ATmega, debemos tener presentes las siguientes consideraciones:

4.1.1 Tiempo de conversión:

Para realizar adecuadamente una conversión, la frecuencia del circuito de aproximaciones sucesivas debe estar entre 50 KHZ y 200 Khz, para una resolución de 10 bits. Si se requiere una resolución menor, se puede utilizar frecuencias superiores a 200Khz.

Para obtener la frecuencias aceptables, el modulo del ADC contiene un circuito de pre-escala.

Una vez que se inicia una conversión, el ADC toma 13 ciclos de reloj de ADC, llevar a cabo una conversión. Exceptuando a primera que toma 25 ciclos

4. 1.2 Referencias de voltaje.

El voltaje en la terminal V_{REF} indica el rango de voltaje de la conversión. Las referencias de voltaje pueden ser seleccionadas entre A_{VCC} , referencia interna de 2.56V o el voltaje en la Terminal externa AREF. Si se ocupa la terminal AREF, se recomienda conectar un capacitor entre la terminal y tierra para aumentar la inmunidad al ruido del ADC.

4.1.3 Definiciones.

Offset:

La desviación de la primera conversión comparada con el valor ideal.

Error de Ganancia:

Una vez ajustado el offset, el error de ganancia es la desviación del resultado obtenido en el valor más grande comparado contra su valor ideal.

No-linealidad Integral (INL, Integral non-linearity):

Una vez ajustados el offset y el error de ganancia, el error de no linealidad integral es el valor de la desviación máxima de una conversión actual comparada contra una inversión ideal. (0.5LSB)

No-linealidad diferencial (DNL, Diferencial non lineality):

La máxima desviación del ancho del código actual (el intervalo entre dos conversiones adyacentes) con respecto al código ideal.

Error de cuantización:

Debido a la cuantización del voltaje de entrada dentro de un rango finito de valores, un cierto rango de voltajes de entrada (1 LSB) se consideran con el mismo valor.

Certeza absoluta:

Es la máxima desviación de una conversión actual, comparada contra una conversión ideal de cualquier código. Este es el efecto de compuesto de offset, error de ganancia, DNL, INL y error de cuantización (2LBS).

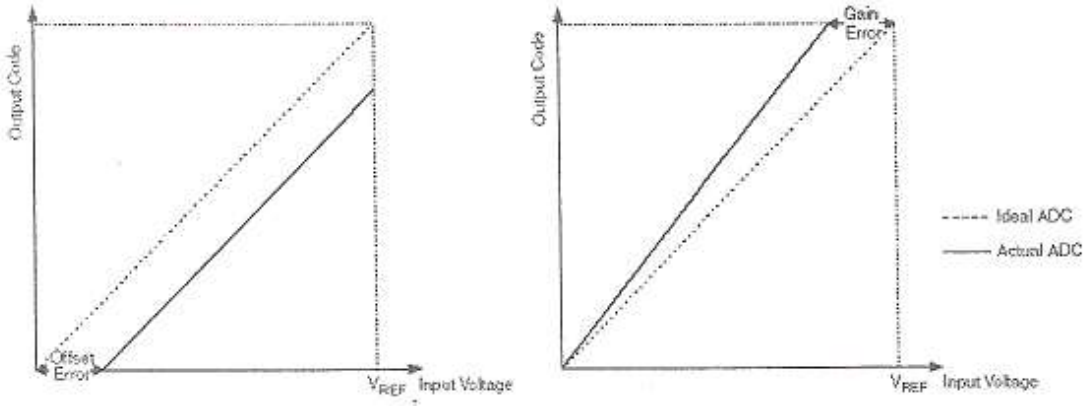


Figura 4.1 Offset y error de ganancia

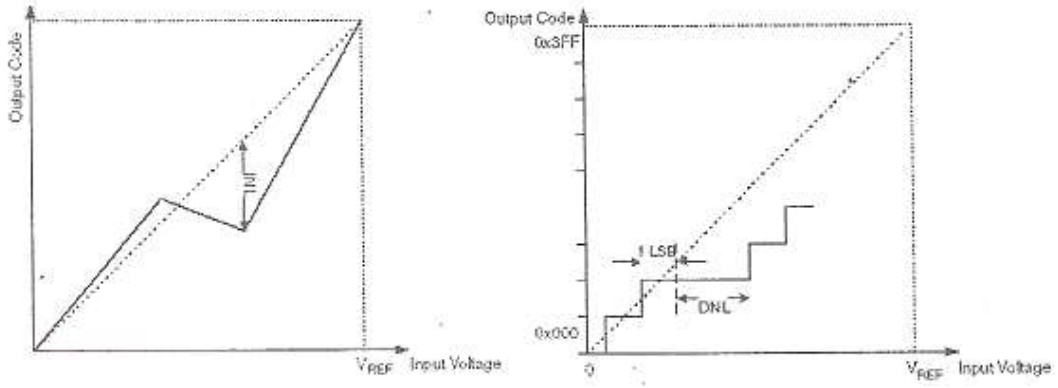


Figura 4.2 INL y DNL

4.1.4 Configuración del ADC

Cuando una conversión analógica-digital se a complementado, el resultado es colocado en este par de registros. Cuando se lee ADCL, el registro no se actualiza hasta leer ADCH, si el resultado es ajustado a la izquierda y no se requieren ms de 8 bits, se puede leer solamente ADCH. De otra forma, se debe leer primero el registro ADCL, después de ADCH.

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

Figura 4.3

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

Figura 7.4 Registro ADC, con ADLAR = 1.

Figura 4.4 Registro ADC con ADLAR

Registro de selección de multiplexor, ADMUX

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 4.5 Registro ADMUX

Bit 7:6 – REFSO bit de selección de referencia. Seleccionan la referencia de voltaje para el ADC. La opinión de referencia de voltaje intera no debe utilizarse si se tiene conectado el pin AREF. Ver tabla 7.1

Bit 5- ADLAR: Resultado alineado a la izquierda del ADC.

Bit 4- Reservado

Bit 3:0 – MUX3:MUXO, Selección de canal analógico. Selecciona que canal de ADC se utilizara. Ver tabla 7.2

REFS1	REFS0	Selección de referencia de voltaje
0	0	AREF, Referencia interna V_{ref} apagada
0	1	A_{VCC} con capacitor externo en AREF
1	0	Reservado
1	1	Referencia interna 2.56 V con capacitor externo en AREF

Tabla 7.1 Selección de referencia de voltaje.

MUX3:0	Entrada *single-ended*
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5

Tabla 4.1 Selección De Canal Analogico

Registro de control y estado A, ADCSRA.

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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 4.7

Bit 7- ADEN: Habilitación de ADC. Escribir un uno a este bit, activa ADC.

Bit 6-ADSC: Inicia conversión. En modo de conversión simple, escribir este bit a uno, inicia cada conversión. Permanece en uno mientras dure la conversión.

Bit 5- ADFR: Habilitar modo "free-runing". Si se active este bit, el ADC realiza conversión y actualizaciones al registro ADC continuamente.

Bit 4-ADIF: Bandera de interrupción por conversión del ADC. Se activa esta bandera cuando una conversión del ADC se completo.

Bit 3-ADIE: Habilitación de interrupción por ADC. Si este se activa y también esta activado al bit l del SREG.

Bit 2:0- ADPS2:ADPS): Pre-escala del reloj del ADC. Ver tabla 7.3

ADPS2	ADPS1	ADPS0	Factor de división
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Tabla 4.2

4.2 PWM

Una señal analógica tiene un rango variable de valores con una resolución infinita ya que puede tomar cualquier valor real; mientras que una señal digital solamente puede tomar valores dentro de un rango predeterminado y finito de posibilidades.

El control analógico no es siempre la mejor opción en la industria pues es mas propenso a alteraciones por ruido y con el tiempo suele presentar derivas lo cual va

complicando la sintonización de los mismos. Los controles analógicos de alta precisión suelen ser muy caros, espaciosos y pesados.

La modulación por ancho del pulso, o PWM (Pulse Width Moulation) por sus siglas en ingles, es una poderosa técnica para el control de circuitos analógicos utilizando una señal digita. En este caso la salida digital de un micro controlador.

Visto de una forma sencilla la PWM es una forma de codificar señales analógicas de una forma digital. A través del uso de corladores una señal cuadrada es modulada para obtener un determinado nivel analógico

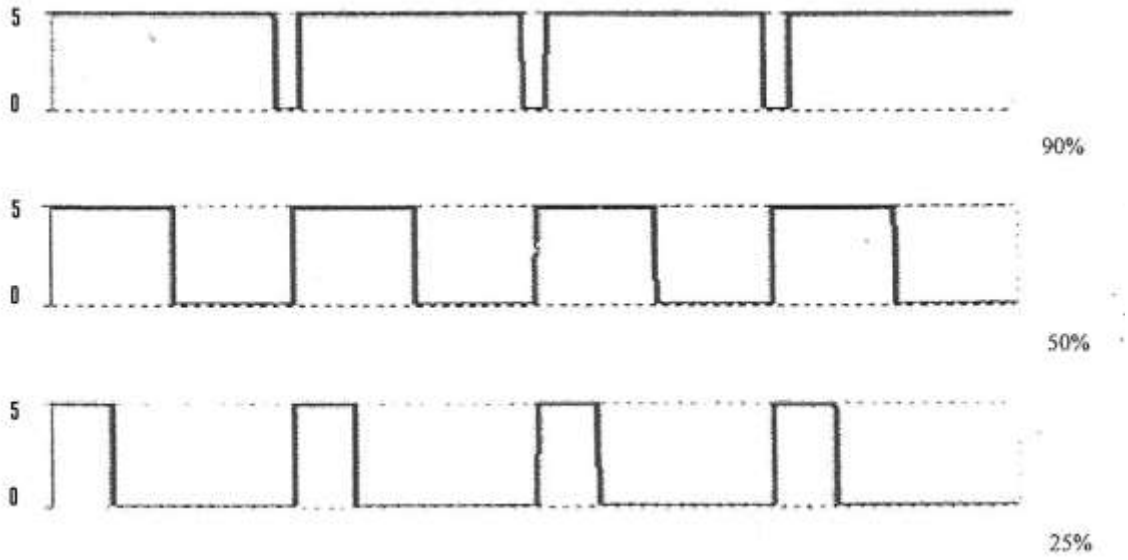


Figura 4.8 Diferentes Ciclos De Trabajo De Un PWM

4.2.1 Tipos de PWM

El micro controlador AT mega posee 3 canales de PWN (OC1A, OC1B y OC2), las dos primeras son canales de 16 bits mientras que OC2 es de 8 bits.

Características de PWM rápida:

. PWM de la alta frecuencia

. Modo de operación “single-slope” (cuesta simple)

. Modo de operación invertido y no-invertido.

Resolución de 8,9 y 10 bits o bien definida por ICR1 o OCR1A (resolución mínima de 2 bits o 16 bits).

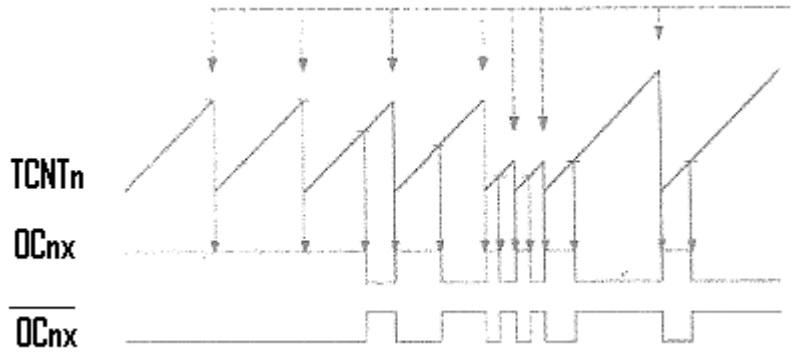


Figura 4.9 Diagrama de tiempos de modo de PWM rápido.

4.2.2 Características de PWM con corrección de fase

- Alta resolución
- Modo de operación “dual slope” (doble cuesta)
- Modo de operación invertido y no- invertido

Resolución de 8,9 y 10 bits o bien definida por ICR1 o OCR1A (resolución mínima de 2 bits o 16 bits).

Como se ve en la figura siguiente la salida generada es, en contraste con la PWM de corrección de fase, simétrica en todos los periodos. Ya que los registros OCR1x son actualizados en la parte baja del conteo, la longitud tanto de la cuesta de subida como la

cuesta de bajada son siempre iguales. Lo que nos da una salida simetría en pulsos y por lo tanto la frecuencia es correcta.

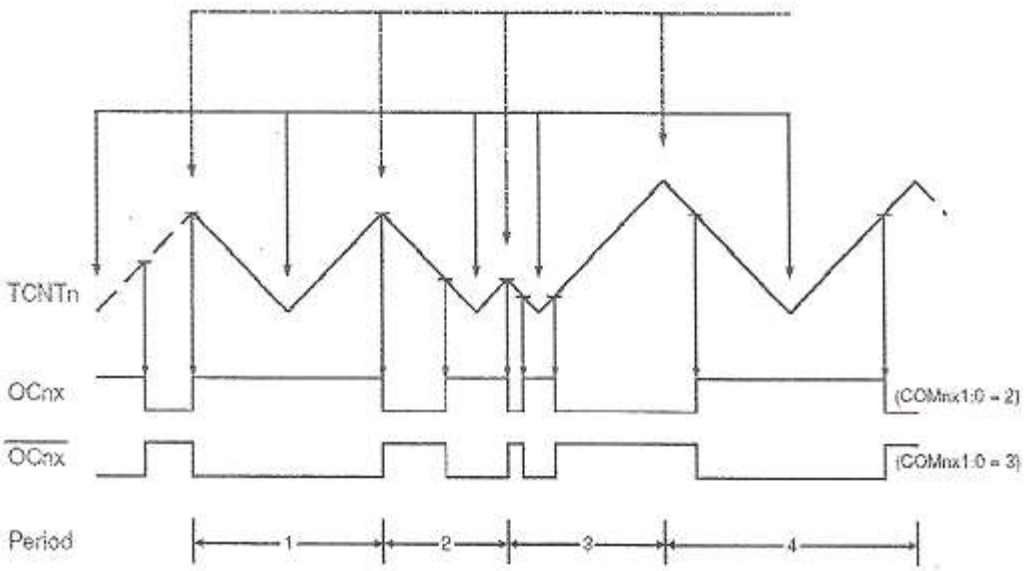


Figura 4.10
Diagrama De tiempos del modo de PWM con corrección de fase

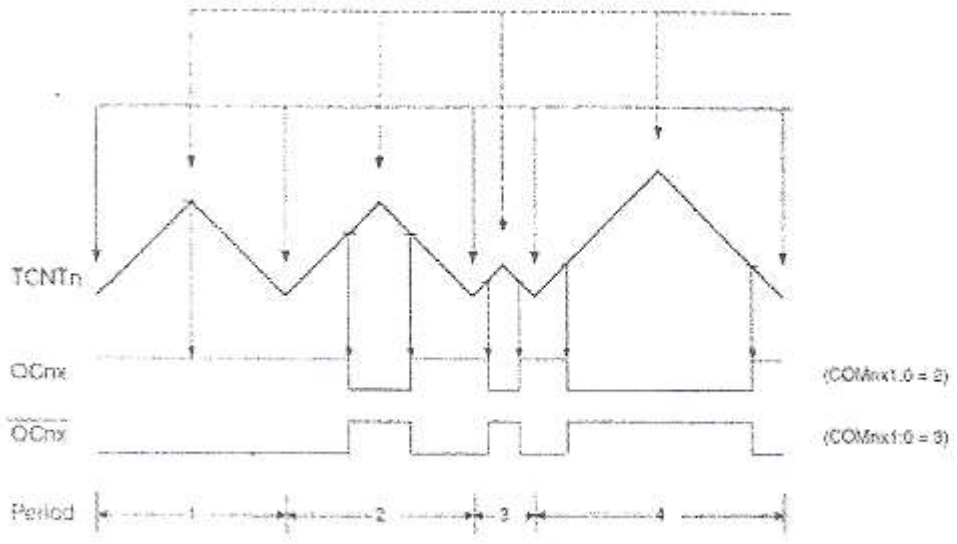


Figura 4.11
Diagrama De tiempos del modo de PWM con corrección de fase

4.4.3 Configuración de la PWM.

Los registros que intervienen en la configuración de la PWM son los mismos que intervienen en la configuración de los temporizadores.

A continuación solo veremos los bits referentes al PWM.

Registro de control A, TCCR1A.

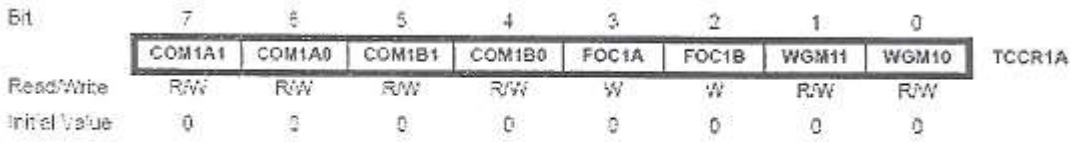


Figura 9.5 Registro TCCR1A

Figura 4.12

- Bit 7:6 – COM1A1:0 Modo de salida de comparación para canal A.
- Bit 5:4 – COM1A1:0 Modo de salida de comparación para canal B.
- Bit 1:0 – WGM11:10 Modo de generación de formas de ondas.

Registro de control B, TCCR1B

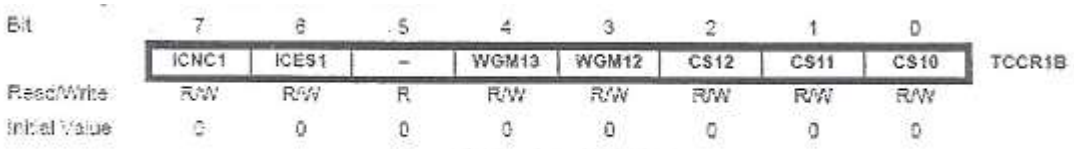


Figura 9.6 Registro TCCR1B

Figura 4.13

- Bit 4:3 – WGM13:12 Modo de generación de forma de onda.

- Bit 2:0 – Cs12:10 Fuente de reloj para el temporizador. Estos últimos tres bits, configuran la fuente de reloj que utilizara el temporizador, de acuerdo a la siguiente tabla.

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	–	–	–
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

Figura 9.7 Bits de selección de WGM.

Tabla 4.3

4.3 USART

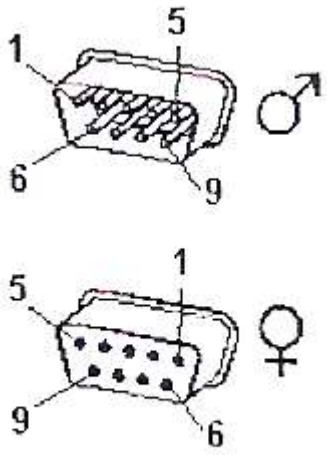
El Transmisor y Receptor Serial Asíncrono y Síncrono Universal o USART (Universal Synchronous and Asynchronous Serial Receiver and Transmitter) es una unidad de comunicación periférica muy flexible que en el microcontrolador Atmega , nos permite, entre otras funciones:

- Operación full-duplex (se puede enviar y recibir datos simultáneamente)
- Operación Sincrónica y Asincrónica
- Operación en modo maestro y esclavo con reloj síncrono.
- Soporta paquetes de 5, 6, 7,8, y 9 bits d datos y 1 o 2 de parada
- Generador de paridad par o impar
- Detección de errores
- Filtrado de ruido (inicio falso, filtro digital)
- Generación de interrupciones por transmisión completa, por recepción completa o por registro de datos de transmisión vacía.
- Comunicación entre multiprocesadores
- Doblador de velocidades modo de comunicación asíncrona

4.3.1 NORMA RS-232

Es un protocolo de comunicación serial ya establecido que define la transmisión física entre una terminal DTE (Equipo Terminal De Datos) y un modem DCE (Equipo De Comunicación De Datos).

El conector normalizado para este protocolo es el conector V24 (25 pines); sin embargo en equipos de cómputo es más utilizado el conector DB9 de 9 pines, los pines de este conector son de la siguiente manera:



No.	Nombre	Función
1	CD	Carrier Detect
2	RXD	Recepcion de Dato
3	TXD	Transmision de Dato
4	DTR	Data Terminal Ready
5	GND	Tierra del Sistema
6	DSR	Data Set Ready
7	RTS	Request to Send
8	CTS	Clear to Send
9	RI	Ring Indicator

Figura 4.14

4.3.2 CONFIGURACION DE LA USART

Registro de entrada y salida de datos de la USART, UDR.

Es el registro en donde se escriben los datos a transmitir (TXB) y de donde se leen los datos recibidos (RXB), Solo se puede escribir en este registro cuando el bit UDR (USART Registro De Datos Vacio) los datos escritos en este registro mientras el UDR este en cero, serán ignorados.

Bit	7	6	5	4	3	2	1	0	
	RXB(7:0)								UDR (Read)
	TXB(7:0)								UDR (Write)

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

Registro de Control y estado A de la USART, UCSRA

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- Bit 7- RXC, recepción completa: Este bit se activa cuando existen datos en el buffer de recepción
- Bit 6- TXC, transmisión completa: Este bit se activa cuando se han enviado todos los datos del buffer de transmisión.
- Bit 5- UDRE, Registro de datos USART vació; Indica que el buffer UDR esta listo para recibir nuevos datos.
- Bit 4- FE, error en el frame, se activa cuando existe un error en el frame de datos recibidos.
- Bit 3- DOR, data overrun, si el buffer de recepción esta lleno y se detecta un nuevo inicio.
- Bit 2- PE, error de paridad: si se activo el chequeo de paridad (UPM1=1) existe un error de paridad en la transmisión, este bit se activa.

- Bit 1- U2X, Doblador de velocidad USART, solo tiene efecto en modo asíncrono, reduciendo el divisor de baud rate de 16 a 8, lo que ocasiona que se duplique la velocidad de transmisión.

- Bit 0-MPCM, Modo de comunicación multiprocesador. Activa el modo de comunicación multi-procesador, todos los frames recibidos que no contengan información de dirección serán ignorados.

Registro de Control y estado B de la USART, UCSRB

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDREIE	RXEN	TXEN	UCSZ2	RX88	TXBS	UCS
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7- RXCIE: Habilitación de interrupción por recepción completa.
- Bit 6- TXCIE: Habilitación de interrupción por transmisión completa.
- Bit 5- UDRIE: Habilitación de interrupción por registro de datos de USART vacio.
- Bit 4- RXEN: Habilita en receptor de la USART.
- Bit 3- TXEN: Habilita en transmisor de la USART.
- Bit 2-UCSZ2: En conjunto con UCSZ1:0, establecen el tamaño de bits de datos a utilizar
- Bit1- RXB8: Si se escogió un frame de 9 bits de datos, aquí se ubica el noveno bit recibido, este debe ser leído antes de los 8 restantes.
- Bit 0- TXB8: Si se escogió un frame de 9 bits de datos, aquí se debe de escribir el noveno bit, este debe escribirse antes de los 8 restantes.

Registro de Control y estado C de la USART, UCSRC

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Initial Value	1	0	0	0	0	1	1	0

- Bit 7- URSEL: Este bit selecciona si se accede el registro UCSRC (URCEL=1) al registro UBRRH (URSEL=0)

- Bit 6-UMSEL: Modo de operación de la USART, cero en este bit la USART opera en modo asíncrono, con un uno en este bit, la USART operara de forma síncrona.

- Bit 5:4- Modo de paridad: las posibles combinaciones para el modo de operación con partida se muestra de la siguiente manera:

UPM1	UPM0	Modo de Paridad
0	0	Desactivada
0	1	Reservado
1	0	Activada, paridad par
1	1	Activada, paridad impar

- Bit 3- USBS, selección de bits de parada: 0=1 bit de parada, 1=2 bits de parada.

- Bit 2:1 –UCSZ1: UCSZ0, en conjunto con UCSZ2, establecen el tamaño de bits de datos de acuerdo a la siguiente tabla

- Bit 0- UCPOL: Polaridad de reloj. Solo aplica en modo síncrono y determina la relación entre el cambio en la salida de datos y la entrada de datos y el reloj sincronía (XCK); de acuerdo a la siguiente tabla

UCPOL	Cambio de Transmision (TXD)	Muestreo de Recepcion
0	Flanco de subida en XCK	Flanco de bajada en XCK
1	Flanco de bajada en XCK	Flanco de subida en XCK

4.4 SPI

Es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. El bus de interface de periféricos serie o bus SPI es un estándar para controlar casi cualquier electrónica digital que acepte un flujo de bits serie regulado por un reloj

Incluye una línea de reloj, dato entrante, dato saliente y una terminal de selección de dispositivo, que conecta o desconecta la operación del dispositivo con el que uno desea comunicarse. De esta forma, este estándar permite multiplexar las líneas de reloj.

Muchos sistemas digitales tienen periféricos que necesitan existir pero no ser rápidos.

La ventajas de un bus serie es que minimiza el numero de conductores, pines y el tamaño del circuito integrado. Esto reduce el costo de fabricar montar y probar el circuito. Un bus de periféricos serie es la opción más flexible cuando muchos tipos diferentes de periféricos serie están presentes.

El hardware consiste en señales de reloj, entrada de datos, salida de datos y una linea de selección para cada circuito integrado que tiene que ser controlado. Casi cualquier dispositivo digital puede ser controlado con esta combinación de señales. Los dispositivos se diferencian en un número predecible de formas. Unos leen el dato cuando el reloj sube otros cuando el reloj baja. Algunos lo leen en el flanco de subida del reloj y otros en el flanco de bajada. Escribir es casi siempre en la dirección opuesta de la dirección de movimiento del reloj. Algunos dispositivos tienen dos relojes. Uno para capturar o mostrar los datos y el otro para el dispositivo interno.

Ventajas

- comunicación full-duplex
- mayor velocidad de transmisión que con I²C o SMBus

- protocolo flexible en que se puede tener un control absoluto sobre los bits transmitidos
- no está limitado a la transferencia de bloques de 8 bits
- elección del tamaño de la trama de bits, de su significado y propósito
- su implementación en hardware es extremadamente simple
- consume menos energía que I²C o que SMBus debido que posee menos circuitos (incluyendo las resistencias *pull-up*) y estos son más simples
- no es necesario arbitraje o mecanismo de respuesta ante fallos
- los dispositivos *esclavo* usan el reloj que envía el *master*, no necesitan por tanto su propio reloj
- no es obligatorio implementar un transceptor (emisor y receptor), un dispositivo conectado puede configurarse para que solo envíe, sólo reciba o ambas cosas a la vez
- usa muchos menos terminales en cada chip/conector que una interfaz paralelo equivalente
- como mucho una única señal específica para cada *esclavo* (señal SS), las demás señales pueden ser compartidas

Desventajas

- consume más terminales en cada chip que I²C, incluso en la variante de 3 hilos
- el direccionamiento se hace mediante líneas específicas (señalización fuera de banda) a diferencia de lo que ocurre en I²C que se selecciona cada chip mediante una dirección de 7 bits que se envía por las mismas líneas del bus
- no hay control de flujo por hardware

- no hay señal de asentimiento. El *master* podría estar enviando información sin que estuviese conectado ningún *esclavo* y no se daría cuenta de nada
- no permite fácilmente tener varios *maestro* conectados al bus
- sólo funciona en las distancias cortas a diferencia de, por ejemplo, RS-232, RS-485, o CAN

CAPITULO 5

HERRAMIENTAS DE DISEÑO

5.1 ENSAMBLADOR

El término ensamblador se refiere a un tipo de programa informático que se encarga de traducir un fichero fuente escrito en un lenguaje ensamblador, a un fichero objeto que contiene código máquina, ejecutable directamente por la máquina para la que se ha generado.

El propósito para el que se crearon este tipo de aplicaciones es la de facilitar la escritura de programas, ya que escribir directamente en código binario, que es el único código entendible por la computadora, es en la práctica imposible. La evolución de los lenguajes de programación a partir del lenguaje ensamblador originó también la evolución de este programa ensamblador hacia lo que se conoce como programa compilador.

5.1.1 Funcionamiento

El programa lee el archivo escrito en lenguaje ensamblador y sustituye cada uno de los códigos nemotécnicos que aparecen por su código de operación correspondiente en sistema binario para la plataforma que se eligió como destino en las opciones específicas del ensamblador.

5.1.2 Tipos de ensambladores

Podemos listar tres tipos de ensambladores:

Ensambladores básicos:

Son de muy bajo nivel, y su tarea consiste básicamente en ofrecer nombres simbólicos a las distintas instrucciones, parámetros y cosas tales como los modos de direccionamiento. Además, reconoce una serie de directivas (o meta instrucciones) que indican ciertos parámetros de funcionamiento del ensamblador.

Ensambladores modulares, o macro ensambladores:

Descendientes de los ensambladores básicos, fueron muy populares en las décadas de los 50 y los 60, antes de la generalización de los lenguajes de alto nivel. Hacen todo lo que puede hacer un ensamblador, y además proporcionan una serie de directivas para definir e invocar macroinstrucciones (o simplemente, macros). Véase X86.

Ensambladores modulares 32-bits o de alto nivel:

Son ensambladores que aparecieron como respuesta a una nueva arquitectura de procesadores de 32 bits, muchos de ellos teniendo compatibilidad hacia atrás pudiendo trabajar con programas con estructuras de 16 bits. Además de realizar la misma tarea que los anteriores, permitiendo también el uso de macros, permiten utilizar estructuras de programación más complejas propias de los lenguajes de alto nivel.

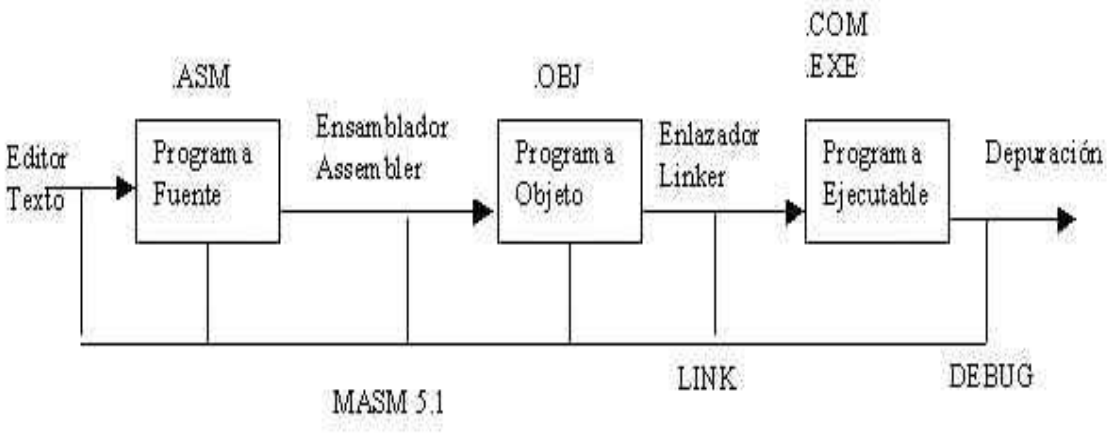


Figura 5.1

5.1.3 ventajas de los lenguajes de ensamblador.

Un programa escrito en el lenguaje ensamblador requiere considerablemente menos memoria y tiempo de ejecución que un programa escrito en los conocidos lenguajes de alto nivel, como Pascal y C.

El lenguaje ensamblador da a un programador la capacidad de realizar tareas muy técnicas que serian difíciles, si no es que imposibles de realizar en un lenguaje de alto nivel.

El conocimiento del lenguaje ensamblador permite una comprensión de la arquitectura de la maquina que ningún lenguaje de alto nivel puede ofrecer.

Aunque la mayoría de los especialistas en Software desarrolla aplicaciones en lenguajes de alto nivel, que son más fáciles de escribir y de dar mantenimiento, una práctica común es re-codificar en lenguaje ensamblador aquellas rutinas que han causado cuellos de botella en el procesamiento.

Los programas residentes y rutinas de servicio de interrupción casi siempre son desarrollados en el lenguaje ensamblador.

5.2 COMPILADOR

Un compilador es un programa que lee un programa escrito en un lenguaje *fuente* y lo traduce a un programa equivalente en otro lenguaje, el lenguaje objeto. Como parte importante de este proceso de traducción, el compilador informa al usuario de la presencia de errores en el programa fuente.

En la compilación hay dos partes análisis y síntesis .

Durante el análisis se determinan las operaciones que implica el programa fuente y se registran en una estructura jerárquica llamada árbol. A menudo se usa una clase especial de árbol llamado árbol sintáctico, donde cada nodo representa una operación y los hijos del nodo son los argumentos de la operación.

Fases de un Compilador

Un compilador típicamente opera en fases, cada una lleva a cabo una tarea sobre el programa fuente.

Las primeras tres fases suelen agruparse en una sola fase llamada fase de análisis y las últimas tres en una llamada fase de síntesis. La fase de análisis y el modulo de manejo de errores se describen posteriormente en este mismo capítulo. La fase de síntesis no es relevante en el contexto de un lenguaje multi-base de datos, ya que este sigue un enfoque diferente que el de los lenguajes tradicionales, por esta razón solo se menciona.

Muchas herramientas de software que manipulan programas fuente realizan primero algún tipo de análisis, entre estas se encuentran los editores de estructuras, impresoras estéticas, verificadores estáticos y los interpretes

5.2.1 Tipos de compiladores

Compilador cruzado:

Es el que genera un código objeto ejecutable en un ordenador distinto de aquél en el que se realiza la compilación.

Compilación-Montaje-Ejecución:

En las aplicaciones grandes es conveniente fragmentar el programa a realizar en módulos que se compilan por separado, y una vez que estos estén compilados unirlos mediante un programa denominado *montador* para formar el módulo ejecutable. El montador se encarga, a su vez, de incluir las librerías donde están guardadas las funciones predefinidas de uso común.

Compilación en una o varias pasadas:

Se llama *pasada* a cada lectura que hace el compilador del texto fuente.

Compilación incremental:

Este compilador actúa de la siguiente manera. Compila un programa fuente. Caso de detectar errores al volver a compilar el programa corregido sólo compila las modificaciones que se han hecho respecto al primero.

Autocompilador: Es aquél que está escrito en el mismo lenguaje que se pretende compilar.

Metacompilador: Es un traductor que tiene como entrada la definición de un lenguaje y como salida el compilador para dicho lenguaje.

Decompilador: Es el que traduce código máquina a lenguaje de alto nivel. Los decompiladores más usuales son los desensambladotes, que traducen un programa en lenguaje máquina a otro en ensamblador.

Bootstrapping: Es una técnica muy usada actualmente para el desarrollo de compiladores de lenguajes de alto nivel, en especial si se quiere obtener un auto-compilador, o sea, un compilador que se compile a sí mismo.

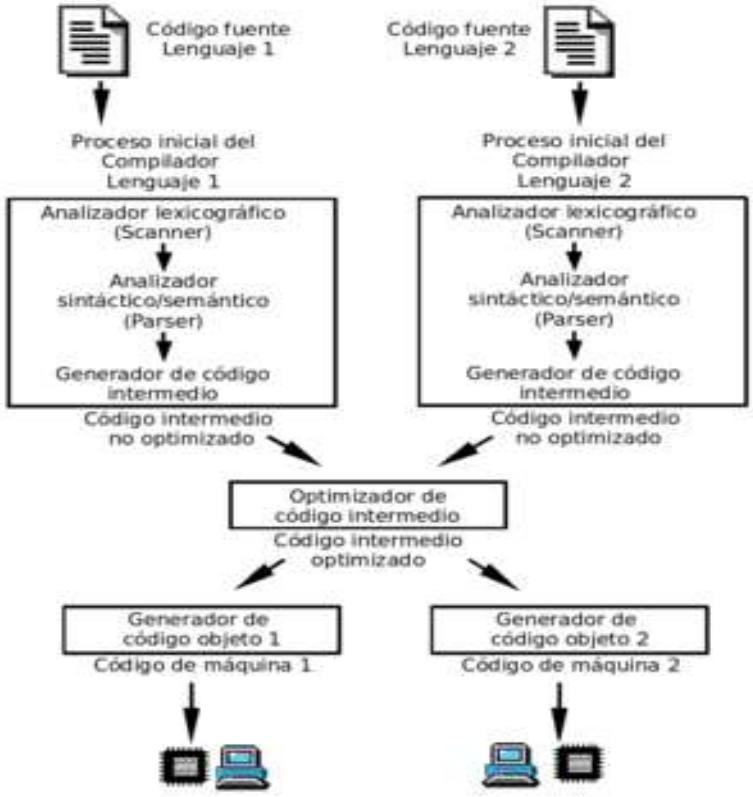


Figura 5.2 Diagrama a bloques del funcionamiento de un compilador profesional

5.3 DEPURADOR

Un depurador es un programa que permite depurar o limpiar los errores de otro programa informático.

5.3.1 Usos

Al Iniciarse la depuración, el depurador lanza el programa a depurar. Éste se ejecuta normalmente hasta que el depurador detiene su ejecución, permitiendo al usuario examinar la situación.

El depurador permite detener el programa en:

- Un punto determinado mediante un punto de ruptura.
- Un punto determinado bajo ciertas condiciones mediante un punto de ruptura condicional.
- Un momento determinado cuando se cumplan ciertas condiciones.
- Un momento determinado a petición del usuario.

Durante esa interrupción, el usuario puede:

- Examinar y modificar la memoria y las variables del programa.
- Examinar el contenido de los registros del procesador.
- Examinar la pila de llamadas que han desembocado en la situación actual.
- Cambiar el punto de ejecución, de manera que el programa continúe su ejecución en un punto diferente al punto en el que fue detenido.
- Ejecutar instrucción a instrucción.
- Ejecutar partes determinadas del código, como el interior de una función, o el resto de código antes de salir de una función.

El depurador depende de la arquitectura y sistema en el que se ejecute, por lo que sus funcionalidades cambian de un sistema a otro. Aquí se han mostrado las más comunes.

Información de depuración

Para poder aprovechar todas las posibilidades de depuración es necesario que, al compilar el programa a depurar, se indique al compilador que debe incluir instrucciones e información extra para la depuración del código. Dicha información extra consiste básicamente en la correspondencia entre las instrucciones del código ejecutable y las instrucciones del código fuente que las originan, así como información sobre nombres de variables y funciones.

Aún si no se incluye esta información de depuración, sigue siendo posible monitorizar la ejecución del programa. Sin embargo, resultará más difícil y compleja debido a esa falta de información del contexto en el que se ejecuta el programa.

5.4 EMULADOR

Un emulador es un software que permite ejecutar programas de computadora en una plataforma (arquitectura hardware o sistema operativo) diferente de la cual fueron escritos originalmente. A diferencia de un simulador, que sólo trata de reproducir el comportamiento del programa, un emulador trata de modelar de forma precisa el dispositivo que se está emulando.

Existen también emuladores por hardware estos dispositivos son comúnmente caros. Los emuladores por hardware tiene la misma función que el emulador por software con la diferencia que este realiza la emulación directamente en los circuitos generando las señales que normalmente generaría el circuito a emular.

En las siguientes figuras se muestran dos ejemplos de emuladores comerciales:

Figura 5.3 Emulador REAL ICE para PICS

Figura 5.4 Emulador para circuitos Atmega

CAPITULO 6

SISTEMAS OPERATIVOS

6.1 Que Es Un Sistema Operativo

El sistema operativo es el programa (o software) más importante de un ordenador. Para que funcionen los otros programas, cada ordenador de uso general debe tener un sistema operativo. Los sistemas operativos realizan tareas básicas, tales como reconocimiento de la conexión del teclado, enviar la información a la pantalla, no perder de vista archivos y directorios en el disco, y controlar los dispositivos periféricos tales como impresoras, escáner, etc.

En sistemas grandes, el sistema operativo tiene incluso mayor responsabilidad y poder, es como un policía de tráfico, se asegura de que los programas y usuarios que están funcionando al mismo tiempo no interfieran entre ellos. El sistema operativo también es responsable de la seguridad, asegurándose de que los usuarios no autorizados no tengan acceso al sistema.

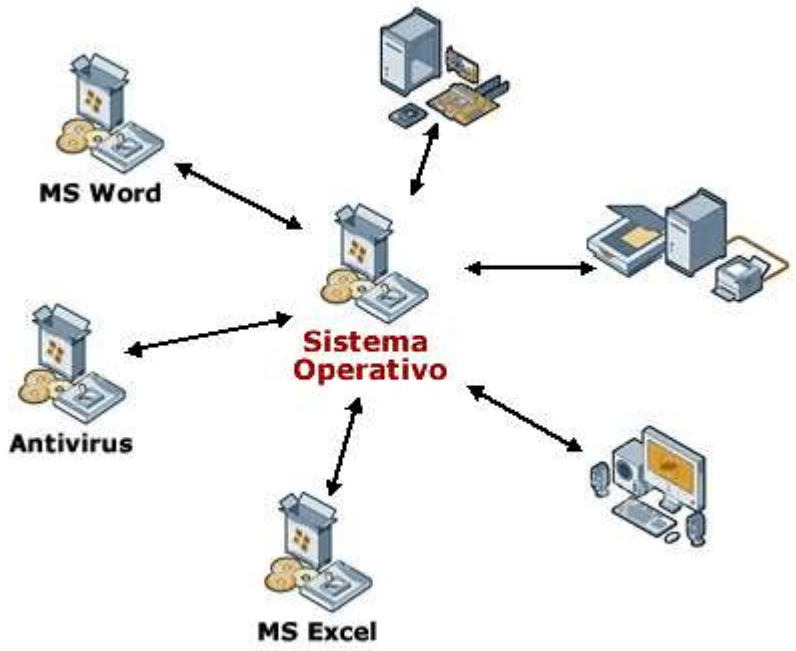


Figura 6.1

Diagrama representa la comunicación de las aplicaciones con el sistema operativo

6.2 Tipos de los Sistemas Operativos

Los sistemas operativos pueden ser clasificados de la siguiente forma:

- **Multiusuario**: Permite que dos o más usuarios utilicen sus programas al mismo tiempo. Algunos sistemas operativos permiten a centenares o millares de usuarios al mismo tiempo.
- **Multiprocesador**: soporta el abrir un mismo programa en más de una CPU.
- **Multitarea**: Permite que varios programas se ejecuten al mismo tiempo.
- **Multitramo**: Permite que diversas partes de un solo programa funcionen al mismo tiempo.
- **Tiempo Real**: Responde a las entradas inmediatamente. Los sistemas operativos como DOS y UNIX, no funcionan en tiempo real.

6.3 Principio de funcionamiento de un Sistema Operativo

Los sistemas operativos proporcionan una plataforma de software encima de la cual otros programas, llamados aplicaciones, puedan funcionar. Las aplicaciones se programan para que funcionen encima de un sistema operativo particular, por tanto, la elección del sistema operativo determina en gran medida las aplicaciones que puedes utilizar.

6.4 Sistemas operativos de tiempo real

Un sistema operativo de tiempo real (SOTR o RTOS -*Real Time Operating System* en inglés), es un sistema operativo que ha sido desarrollado para aplicaciones de tiempo real. Como tal, se le exige corrección en sus respuestas bajo ciertas restricciones de tiempo. Si no las respeta, se dirá que el sistema ha fallado. Para garantizar el comportamiento correcto en el tiempo requerido se necesita que el sistema sea predecible (determinista).

6.4.1 Características generales

Usado típicamente para aplicaciones integradas, normalmente tiene las siguientes características:

- No utiliza mucha memoria
- Cualquier evento en el soporte físico puede hacer que se ejecute una tarea
- Multi-arquitectura (puertos de código para otro tipo de UCP)
- Muchos tienen tiempos de respuesta predecibles para eventos electrónicos

En la actualidad hay un debate sobre qué es tiempo real. Muchos sistemas operativos de tiempo real tienen un programador y diseños de controladores que minimizan los periodos en los que las interrupciones están deshabilitadas, un número llamado a veces duración de interrupción. Muchos incluyen también formas especiales de gestión de

memoria que limitan la posibilidad de fragmentación de la memoria y aseguran un límite superior mínimo para los tiempos de asignación y retirada de la memoria.

6.5 RTOS mas comunes

- QNX
- LynxOS
- RedHat Embedded Linux (GNU/linux)
- FreeRTOS /SafeRTOS
- VxWorks
- Windows CE
- Micrium UCos II

CAPITULO 7

LENGUAJES DE PROGRAMACIÓN

7.1 LENGUAJE DE PROGRAMACIÓN.

Un lenguaje de programación es un conjunto de herramientas con una sintaxis y semántica propia, que permite describir un algoritmo para ser ejecutado por un sistema con microprocesador. Es muy común encontrar que un gran número de personas suele utilizar los lenguajes de alto nivel ya que son más simples de utilizar, ofrecen un lenguaje natural a la lógica de los seres humanos y brindan la posibilidad de ejecutar el algoritmo en diversos diseños de UCPs. Por otro lado, los lenguajes de bajo nivel son bastante cercanos a la lógica de los sistemas digitales, dificultándose su programación además que los algoritmos se hacen más extensos si se desea hacer lo mismo usando un HLL (high level language). Sin embargo, este tipo de lenguajes permite tener un control más directo del sistema, siendo bastante útiles cuando se desea entender realmente el funcionamiento de una CPU y por ende explotar al máximo sus capacidades, incluso cuando se emplean lenguajes de alto nivel.

7.2 Lenguajes de bajo nivel.

Su función es reducir la complejidad de programar en lenguaje de máquina (0s y 1s), para esto incluye una serie de mnemónicos legibles por los usuarios que reemplazan cadenas de ceros y unos. En este tipo de lenguajes es necesario indicarle a la CPU cada paso a realizar en la ejecución de una orden general y para esto es necesario saber como funciona todo un sistema con microprocesador. Suponga que usted quiere hacer la siguiente operación R = A + B.

Aparentemente es una simple suma entre dos variables definidas previamente, sin embargo, desde el punto de vista de CPU esto no es así ya que, y ¿qué es A?, ¿qué es

B?, ¿qué es R? Antes de indicarle al CPU que lleve a cabo la suma, habrá que indicarle como tomar las variables y donde almacenar el resultado. Una CPU no entiende sino ceros y unos y por ende cada instrucción, y cada dato será una cadena de ellos dados en un formato particular, codificados normalmente por un lenguaje como el ensamblador (bajo nivel) desde un conjunto de mnemotécnicos referidos a la operación que se desea llevar a cabo. Una desventaja de este tipo de lenguaje es que no permite portar la aplicación a diferentes CPUs, es por eso que existe un lenguaje ensamblador para cada filosofía de diseño de CPU. La siguiente gráfica muestra el flujo de construcción de una aplicación usando lenguaje ensamblador.

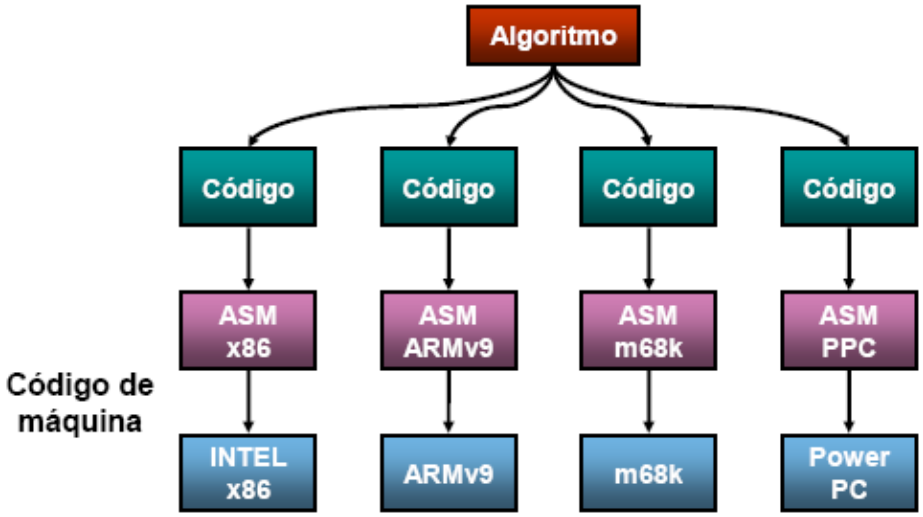


Figura 7.1

7.3 Lenguajes de alto nivel,

Lenguajes fáciles de aprender ya que están formados por un conjunto de elementos naturales para los seres humanos y por ende podemos indicarle a un sistema microprocesador que lleve a cabo una serie de tareas a partir de un conjunto lógico de órdenes muy similares a nuestro comportamiento. Debido a la naturaleza de un sistema

con microprocesador, una de las principales funciones de un lenguaje de alto nivel (HLL) será traducir el código escrito por el programador a un conjunto de ‘0’ y ‘1’ que la CPU puede entender y ejecutar. Esto permite tener códigos portables, ya que será función del HLL traducir el código escrito en alto nivel al correspondiente formato de cada CPU para que se pueda ejecutar allí. A continuación se puede observar una gráfica que muestra el flujo de diseño usando lenguajes de alto nivel.

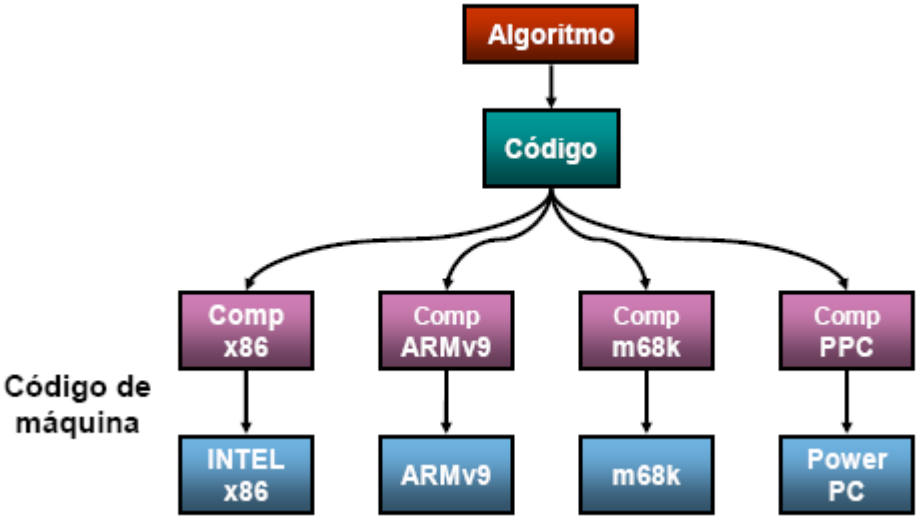


Figura 7.2

7.4 Lnguaje C.

C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones.

Se trata de un lenguaje débilmente tipificado de medio nivel pero con muchas características de bajo nivel. Dispone de las estructuras típicas de los lenguajes de alto nivel pero, a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. Los compiladores suelen ofrecer extensiones al lenguaje que posibilitan

mezclar código en ensamblador con código C o acceder directamente a memoria o dispositivos periféricos.

Generalizando, un programa en C consta de tres secciones. La primera sección es donde van todos los ``headers". Estos ``headers" son comúnmente los ``#define" y los ``#include". Como segunda sección se tienen las ``funciones". Al igual que Pascal, en C todas las funciones que se van a ocupar en el programa deben ir antes que la función principal (main()). Declarando las funciones a ocupar al principio del programa, se logra que la función principal esté antes que el resto de las funciones. Ahora, solo se habla de funciones ya que en C no existen los procedimientos.

Y como última sección se tiene a la función principal, llamada main. Cuando se ejecuta el programa, lo primero que se ejecuta es esta función, y de ahí sigue el resto del programa.

Los símbolos { y } indican ``begin" y ``end" respectivamente. Si en una función o en un ciclo while, por ejemplo, su contenido es de solamente una línea, no es necesario usar ``llaves" ({ }), en caso contrario es obligación usarlos.

La compilación de un programa C se realiza en varias fases que normalmente son automatizadas y ocultadas por los entornos de desarrollo:

7.4.1 Preprocesado:

Consistente en modificar el código fuente en C según una serie de instrucciones (denominadas directivas de preprocesado) simplificando de esta forma el trabajo del compilador. Por ejemplo, una de las acciones más importantes es la modificación de las inclusiones (#include) por las declaraciones reales existentes en el fichero indicado.

7.4.2 Compilación:

Que genera el código objeto a partir del código ya preprocesado.

7.4.3 Enlazado:

Que une los códigos objeto de los distintos módulos y bibliotecas externas (como las bibliotecas del sistema) para generar el programa ejecutable final.

7.4.4 Ventajas:

Lenguaje muy eficiente puesto que es posible utilizar sus características de bajo nivel para realizar implementaciones óptimas.

A pesar de su bajo nivel es el lenguaje más portado en existencia, habiendo compiladores para casi todos los sistemas conocidos. Este lenguaje es un de los mas populares para la programación de microcontroladores.

Proporciona facilidades para realizar programas modulares y/o utilizar código o bibliotecas existentes.

7.5 lenguaje C++.

Como todos sabemos, "C" es un lenguaje de alto nivel, basado en funciones, que permite desarrollos estructurados. Entre otras muchas características contempla la definición de estructuras de datos, recursividad o indirecciones a datos o código (punteros).
"C ++", por su parte, es un superconjunto de "C", al que recubre con una capa de soporte a la POO. Permite por tanto la definición, creación y manipulación de objetos.

Objetos.

Un objeto es una entidad que tiene unos atributos particulares (datos) y unas formas de operar sobre ellos (los métodos o función miembro). Es decir, un objeto incluye, por una parte una serie de operaciones que definen su comportamiento, y una serie de variables manipuladas por esas funciones que definen su estado. Por ejemplo, una ventana Windows contendrá operaciones como "maximizar" y variables como "ancho" y "alto" de la ventana.

Mensajes:

En C++, un mensaje se corresponde con el nombre de uno de los métodos de un objeto. Cuando se pasa un mensaje a un objeto, este responde ejecutando el código de la función asociada.

Método:

Un método (función miembro) se implementa dentro de un objeto y determina como tiene que actuar el objeto cuando se produce el mensaje asociado. En C++ un método se corresponde con la definición de la función miembro del objeto. La estructura más interna de un objeto está oculta, de tal manera que la única conexión con el exterior son los mensajes

Clases:

Una clase es la definición de un tipo de objetos. De esta manera, una clase "Empleado" representaría todos los empleados de una empresa, mientras que un objeto de esa clase (también denominado instancia) representaría a uno de esos empleados en particular.

Las principales características de la POO son: abstracción, encapsulamiento, herencia y polimorfismo:

Abstracción:

Es el mecanismo de diseño en la POO. Nos permite extraer de un conjunto de entidades datos y comportamientos comunes para almacenarlos en clases.

Encapsulamiento:

Mediante esta técnica conseguiremos que cada clase sea una caja negra, de tal manera que los objetos de esa clase se puedan manipular como unidades básicas. Los detalles de la implementación se encuentran dentro de la clase, mientras que desde el exterior, un objeto será simplemente una entidad que responde a una serie de mensajes públicos (también denominados interfaz de la clase).

Herencia:

Es el mecanismo que nos permite crear clases derivadas (especialización) a partir de clases bases (generalización). Es decir, podríamos tener la clase "Empleado" (clase base) y la clase "Vendedor" derivando de la anterior. Una librería de clases (como la MFC) no es más que un conjunto de definiciones de clases interconectadas por múltiples relaciones de herencia.

Polimorfismo:

Esta característica nos permite disponer de múltiples implementaciones de un mismo método de clase, dependiendo de la clase en la que se realice. Es decir, podemos acceder a una variedad de métodos distintos (con el mismo nombre) mediante el mismo mecanismo de acceso. En C++ el polimorfismo se consigue mediante la definición de clases derivadas, funciones virtuales y el uso de punteros a objetos.

7.5.1 Ventajas y desventajas

Una de las grandes ventajas del C++ es que soporta todos los conceptos de la programación orientada a objetos, con esto el tiempo de desarrollo de software disminuye considerablemente, pero desafortunadamente este soporte trae como consecuencia un aumento en el tamaño del código, mas uso de memoria y rutinas con un desempeño menor a las generadas por C. Las rezones previamente mencionada son las causa por la cual C++ no es tan popular para programación de microcontroladores.

7.6 PASCAL.

La sintaxis del Pascal es muy similar a la del idioma inglés y los programas son fáciles de leer e interpretar. Además, Pascal no toma en consideración el hecho que las letras estén en mayúsculas o minúsculas. En el Laboratorio de Informática se emplea el compilador de Pascal de Borland, que ofrece muchas facilidades para el aprendizaje, como el realzado de sintaxis y la ubicación de los errores en la línea de código en que ocurren.

7.6.1 Tipos de datos:

Pascal es un lenguaje donde se debe especificar a la computadora qué datos va a contener una variable. Para decirle a Pascal el tipo de una variable, se usa una de las siguientes palabras clave en el lugar indicado en el esqueleto del programa (ver más abajo):

Integer: número entero entre -32,768 y 32,767

LongInt: número entero entre -2*10^31 y 2*10^31 – 1

Real: número con coma decimal entre 2.9*10-39^y 1.7*10^38

String: cadena de caracteres (conjunto de números, letras, símbolos; palabras y frases)

Char: un carácter (un dígito o una letra o un símbolo)

Aunque Pascal es un lenguaje simple y bien estructurado no es tan popular y tan portable como lo es el lenguaje C, el porcentaje de aplicaciones para microcontroladores desarrolladas en Pascal es muy bajo comparado con las aplicaciones desarrolladas en C

7.7 BASIC.

Basic es un lenguaje de programación de ordenadores, uno de los primeros lenguajes de alto nivel que se diseñó. Se creó con la intención de que fuera adecuado para introducirse en la programación, aunque hoy en día ha sido desbancado en este campo por Pascal. Existen versiones libres y comerciales para la programación de microcontroladores en Basic, pero definitivamente será muy poco común encontrar alguna aplicación seria para un microcontrolador desarrollada en alguna versión de basic.

7.8 Lenguajes Visuales

Los lenguajes visuales en el área de los microcontroladores son mas comúnmente conocidos como generadores de código.

En este tipo de aplicaciones el programador diseña una especia de diagrama el cual podría ser un diagrama de flujo, de estado o un diagrama de bloques y la aplicación generara código en C o ensamblador basando en la descripción del lenguaje grafico.

Algunos ejemplos de este tipo de aplicaciones son las siguientes:

- LabView
- HPView
- FlowCode
- Nipple
- Partes del entorno de Code Warrior
- Partes del entorno de desarrollo MPLAB

CAPITULO 8

EJEMPLOS

8.1 Introducción

En este capitulo se desarrollaran una serie de ejemplos los cuales ilustraran el uso de diferentes compiladores, lenguajes, herramientas de desarrollo, simuladores y el uso de algunos depuradores.

Las herramientas a usar son las siguientes:

Entornos de desarrollo:

- AVR Studio
- ImageCraft
- Herramientas de Keil
- FlowCode

Simuladores:

- AVR Studio
- Proteus

- FlowCode

Emuladores

- JTAG ICE mkII
- AVR USB JTAG

Programadores:

- JTAG ICE mkII
- AVR USB TAG
- Programadote USb de Microelectrónica

Lenguajes:

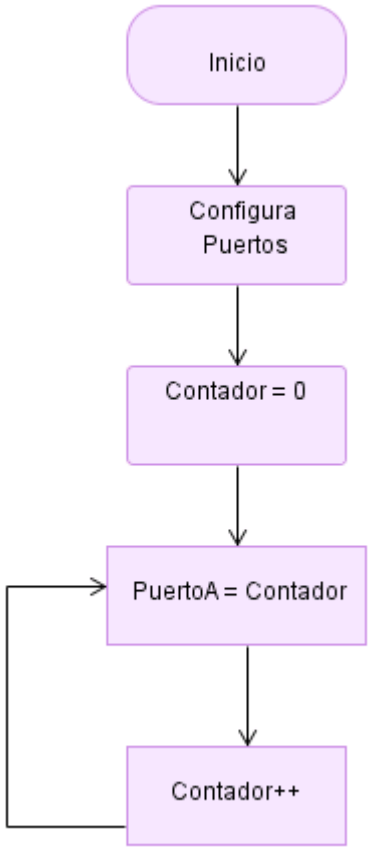
- Basic
- Pascal
- C
- Visual por diagrama de flujo

8.2 Ejemplo 1 (Led intermitente)

8.2.1 Descripción:

Este ejemplo es el clásico “Hola mundo” de cualquier libro de programación para principiantes. El programa encenderá y apagara un led conectado al línea 1 del puerto ‘A’ a una determinada frecuencia.

8.2.2 Diagrama de flujo:



8.2.3 Código ensamblador

```
.include "m32def.inc"

.ORG $0000

START:      LDI R20,$FF
            OUT DDRA,R20
            LDI R20,$00
LOOP1:      OUT PORTA,R20
            INC R20                 ;Contador
            JMP RET1
RET1R:  JMP LOOP1
RET1:   LDI R25,$FF
L1:             NOP
                NOP
                LDI R26,$FF
L2:             NOP
                DEC R26
                BRNE  L2
                DEC R25
                BRNE  L1
                JMP RET1R
```

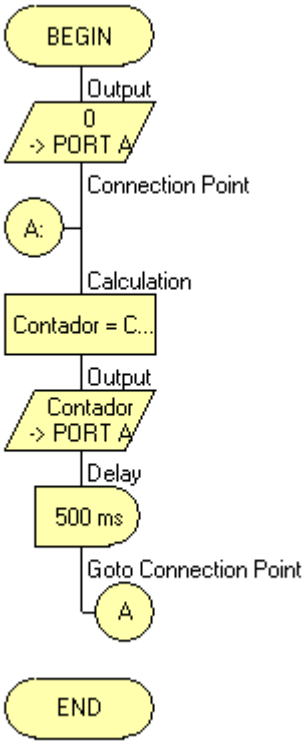
8.2.4 Código C (Avr Studio)

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

void Retardo(void);

int main()

{

        unsigned char Contador;

        Contador =0;

        DDRA = 0xFF;

        PORTA = Contador;

  for(;;)

  {

                PORTA = Contador;

                Contador++;

                Retardo();


  }


}
```

```
/*************************************************************
        Rutina de retardo.
        El retardo lo determina los ciclos controlados por j y k
*************************************************************/
void Retardo(void)
{
  unsigned int j,k;

  for(j=0;j<10000;j++)
  {
        for(k=0;k<1000;k++)
        {

        }

  }
}
```

8.2.5 Programa en FlowCode

Podrá notar que el programa en FlowCode es un diagrama de flujo.



8.2.6 Código generado por FlowCode

```
#define MX_AVR

//Defines for microcontroller

#define MX_AVR

#define MX_EE

#define MX_EE_SIZE 1024

#define MX_SPI

#define MX_SPI_B

#define MX_SPI_SDI 6

#define MX_SPI_SDO 5

#define MX_SPI_SCK 7

#define MX_UART

#define MX_UART_D

#define MX_UART_TX 1

#define MX_UART_RX 0

#define MX_MI2C

#define MX_I2C_C

#define MX_I2C_SDA 1

#define MX_I2C_SCL 0

#define MX_PWM
```

```
#define MX_PWM_PORT PORTD

#define MX_PWM_CNT 2

#define MX_PWM_TRIS1 DDRD

#define MX_PWM_1 5

#define MX_PWM_TRIS2 DDRD

#define MX_PWM_2 4


//Functions

#define F_CPU 20000000UL

#include <avr\io.h>

#include <avr\interrupt.h>

#include <avr\eeprom.h>

#include <MX_util\delay.h>

#include <MX_util\bit_cmds.h>


//Configuration data

#pragma DATA 0x0, 0xdf

#pragma DATA 0x1, 0xff
```

```
//Internal functions

#include "D:\Ext_C\Program Files\Matrix
Multimedia\Flowcode_AVR\FCD\internals.h"

//Macro function declarations

//Variable declarations

volatile char FCV_CONTADOR;

//Supplementary defines

//Macro implementations

//Supplementary implementations

int main()

{

        //Initialisation

        MCUCSR=0x00;

        WDTCR=0x10;

        //Interrupt initialisation code

        //Output

        //Output: 0 -> PORT A

        DDRA = 0xFF;

        PORTA = 0;

        //Connection Point
```

```
        //Connection Point: A
FCC_Main_A:
        //Calculation
        //Calculation:
        //  Contador = Contador + 1
        FCV_CONTADOR =  FCV_CONTADOR + 1 ;
        //Output
        //Output: Contador -> PORT A
        DDRA = 0xFF;
        PORTA = FCV_CONTADOR;
        //Delay
        //Delay: 500 ms
        delay_ms(255);
        delay_ms(245);
        //Goto Connection Point
        //Goto Connection Point: A
        goto FCC_Main_A;
        mainendloop: goto mainendloop;
}
```

8.2.7 Simulación en Flowcode

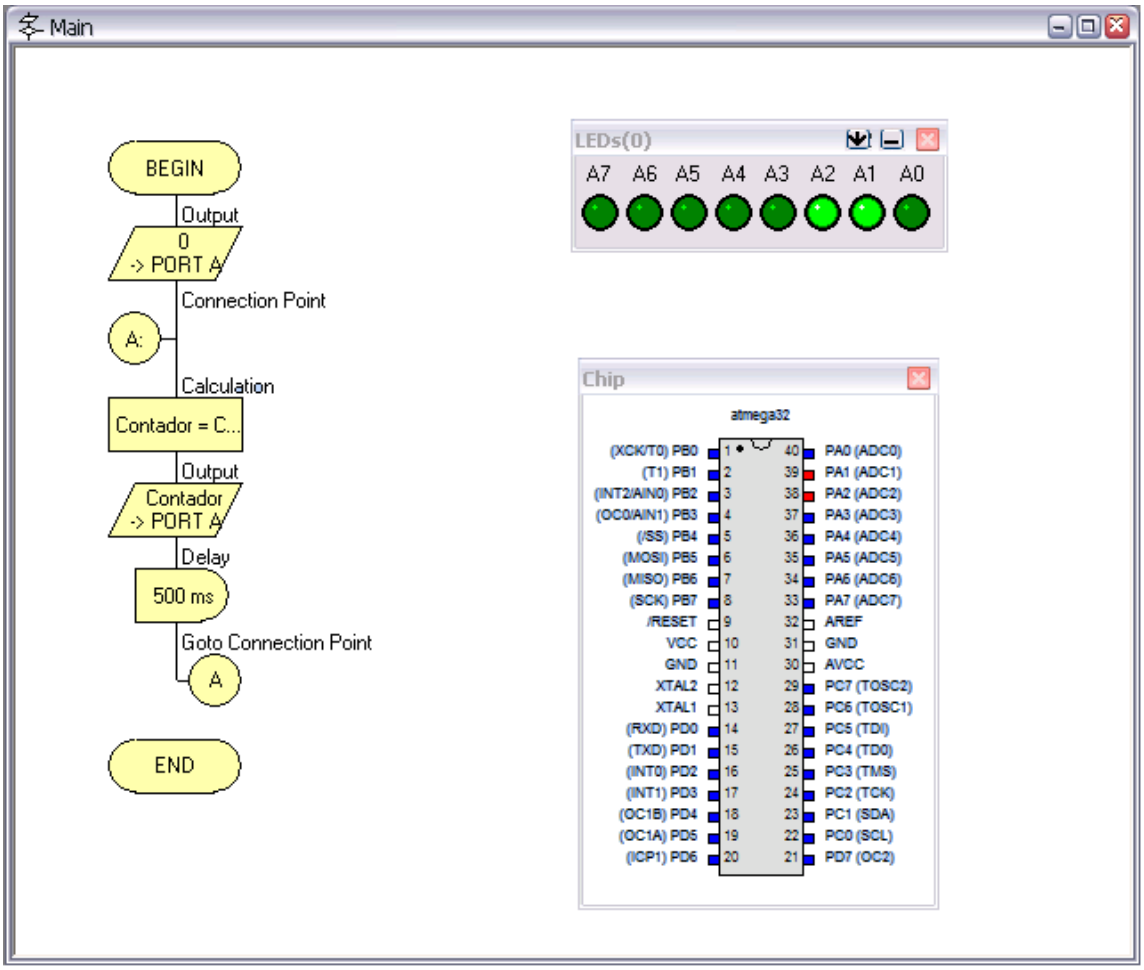
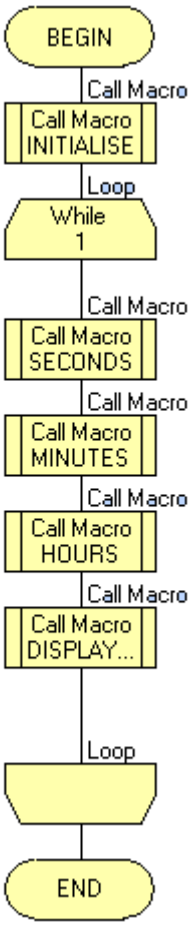


Figura 8.1 Muestra de una simulación en proceso en el programa FlowCode.

8.3 Reloj Digital

Este ejemplo implementa un reloj limitado de 24 horas, el código fue desarrollado en el lenguaje visual de FlowCode. El reloj en este ejemplo no tiene las opciones de cambiar o fijar la hora, solo inicia su cuenta a partir de ceros.

8.3.1 Diagrama De Flujo



8.3.2 Simulación

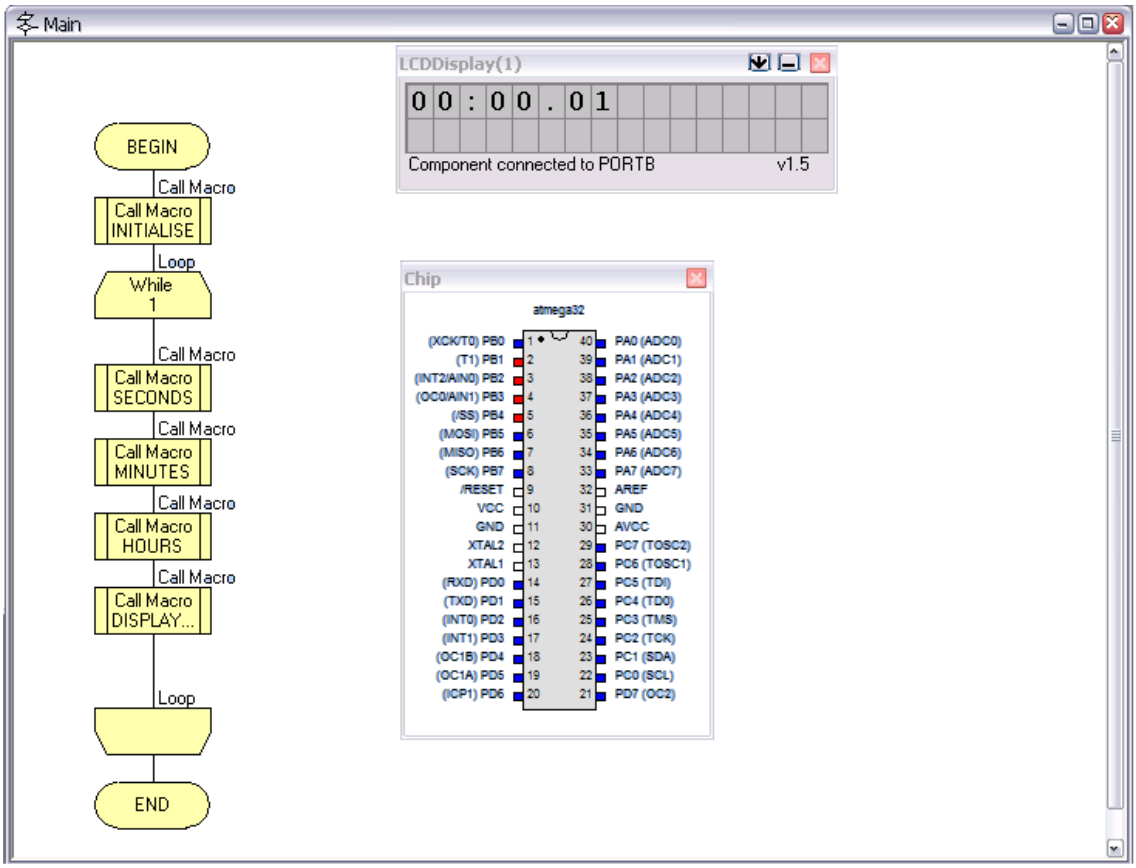


Figura 8.2 Simulación de reloj digital

CAPITULO 9

SISTEMAS DE DESARROLLO

9.1 ¿Que es un sistema de desarrollo?

Un sistema de desarrollo es un circuito sin un propósito especifico, este tipo de circuitos son diseñados para facilitar el aprendizaje de algún microcontrolador. Estas generalmente cuentan con los siguientes bloques:

- Sección de conexiones
- Sistema de desmontaje del microcontrolador (Solo en algunos modelos)
- Sensores
- Pantallas LCD para texto y gráficos
- Leds (Para el monitoreo de las salidas)
- Interfaces comunes como RS-232, RS-485, CAN, Ethernet etc.
- Sección de interruptores para la simulación de las entradas

En la siguiente figura se muestran un modelo de un sistema de desarrollo

Figura 9.1 ATSTK500 para el desarrollo con dispositivos de Atmel

9.2 Sistemas De Desarrollo Comerciales

En el mercado existen muchas opciones comerciales de sistemas de desarrollo que van desde van desde $30 dólares hasta mas de $1000 Dólares.

Unas de las opciones mas comúnmente seleccionadas es trabajar con los sistemas de desarrollo ofrecidos por el fabricante del microcontrolador, estas generalmente son económicas pero tedian pocos periféricos incluidos, en la figura anterior se puede observar como el ATSTK500 no cuenta con pantallas LCD o algún tipo de sensor, esta solo tiene lo necesario para aprender a usar los microcontroladores Atmel.

Otra opción de precio accesible para un estudiante y la cual tiene perifericos adicionales para cubrir todos los temas de las clases de microcontroladores son las de los sistemas desarrollados por la compañía Microelectrónica.

A continuación listaremos algunas opciones disponibles en algunos de los modelos disponibles para las familias de Atmel:

Figura 9.2 Sistema de desarrollo modelo EASYAVR5 de la compañía Microelectrónica.

Este tipo de sistemas cuenta con las siguientes caracteristicas:

- Puerto De Comunicación RS-232
- Soporta Microcontroladores Atmel DIP8, DIP14,DIP20 y DIP40
- Arreglos de Displays de 7 segmentos.
- Arreglos de leds e interruptores configurables
- LCD de texto y grafico
- Interfase JTAG
- Sensor de temperatura
- Programador por USB

- Ranura para memoria MMC/SD

Todo lo mencionado esta disponible para que el estudiante invierta su tiempo inmediatamente programando y pueda acceder desde sus programas algunos de los periféricos mas comúnmente usados.

9.3 Programadores Universales, ¿Son una necesidad para el estudiante?

Viendo algunos anos atrás y pensado en el equipo que era necesario para programar un microprocesador… con el avance de los sistemas de computo y la integración de tecnologías como la memoria flash en los microcontroladores muchos de esas herramientas han quedado casi obsoletas o al menos obsoletas en el diseño de prototipos.

Años atrás era imposible que un estudiante pudiera adquirir el software, programadores, borradores de memorias y otras herramientas que eran necesarias para la programación de los microcontroladores ya que adquirirlas se necesitaba invertir cientos o miles de dólares.

En los microcontroladores modernos los programadores, los borradores y en algunos casos los depuradores han quedado obsoletos, esto debido a la integración en los microcontroladores de tecnologías como ISP(In System Programming) y ICD (In System Debug).

A continuación listaremos las herramientas necesarias así como la inversión para tener un pequeño laboratorio para programación de microcontroladores:

- Entorno De Desarrollo (AVR Studio)…….Gratis
- Compilador Profesional De C (GCC)……Gratis
- Programas Para Carga (PONYPROG)……...Gratis

- Programador Hardware ……. $40
- Depurador ICD (Incluido en el programador)……Gratis
- Software Para Diagramas (Proteus Lite)…… Gratis
- Software para circuitos impresos (Varios Disponibles)……Gratis
- Sistema mínimo……. $10

Es casi imposible creer que por la cantidad de $50 dólares se tenga acceso a herramientas profesionales y sofisticadas para la programación de microcontroladores.

9.4 Construya su propio sistema de desarrollo.

Si, aun con el bajo costo es imposible para un estudiante adquirir un programador de $50 existen otras opciones para las cuales requieren menos inversión, pero un poco más de trabajo para armar el programador. En el siguiente diagrama se muestra el circuito de un sistema de desarrollo el cual se conecta a un PC por el puerto paralelo y se usan los programas AVRDUDE o PonyProg para cargar el programa al microcontrolador así como configurar los registros internos del dispositivo.

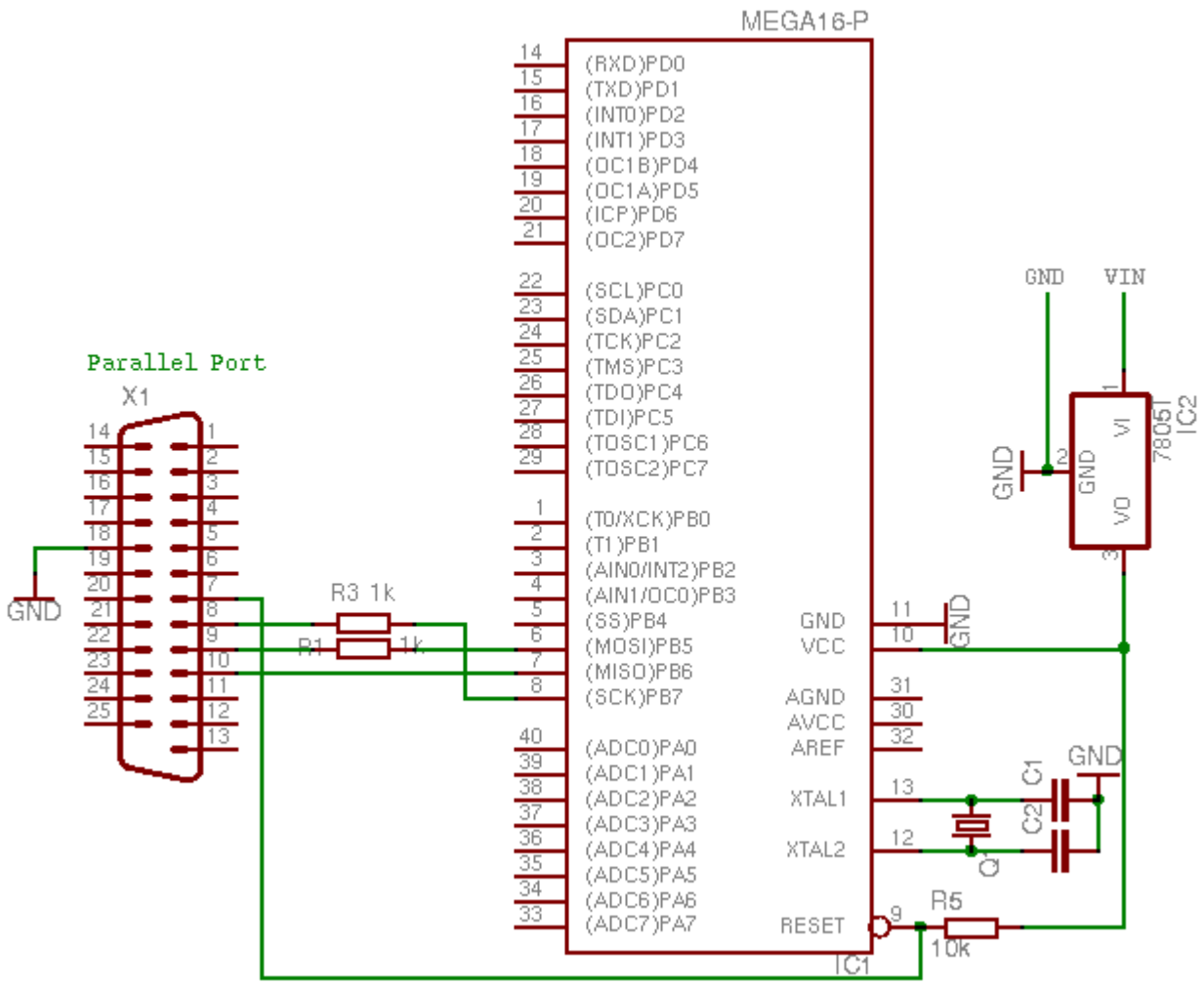


Figura 9.3 Diagrama de un programador de AVR

CAPITULO 10

APLICACIONES

10.1 Control de matriz de leds

Esta aplicación controla una matriz de leds siguiente. El programa es un contador ascendente de 0 a 9. La matriz a controlar es una matriz de 5 x 7 donde los ánodos son las columnas y los cátodos son las filas.

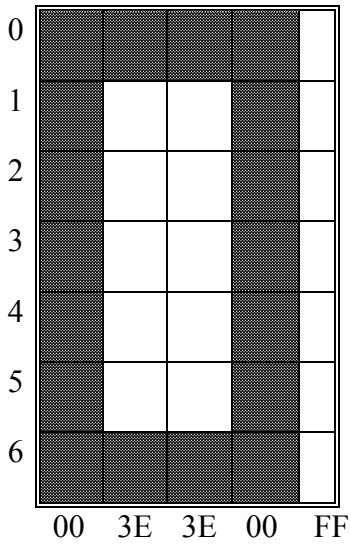
Controlar cada led en forma directa es extremadamente caro pues para una simple matriz como la que usaremos se recurriría de casi 40 líneas de salida del microcontrolador. Para resolver este problema se usa el principio de multiplezazo de líneas. Aquí todos los ánodos de cada columna están unidos y todos los cátodos de cada fila están unidos.

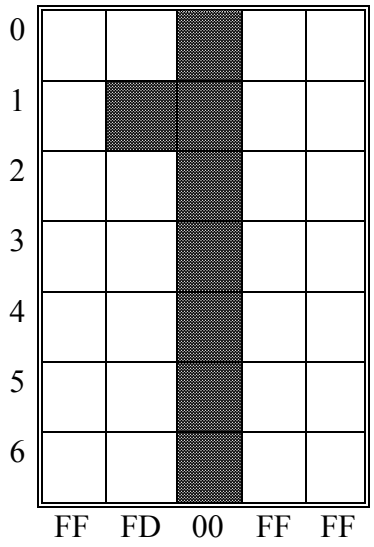
Para desplegar la un digito en la matriz de puntos se tiene que activar la información en cada una de las columnas en forma consecutiva a una velocidad lo suficientemente rápido para que de la apariencia de que la información desplegada en cada columna esta visible en todo momento.

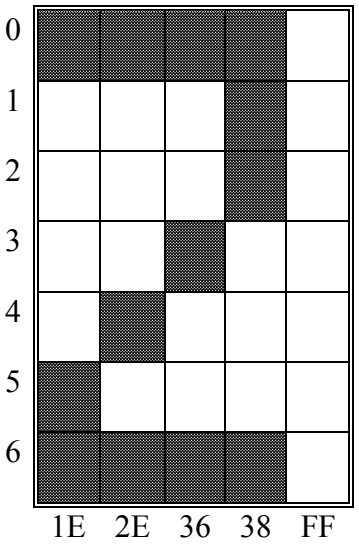
La aplicación usa el puerto C y el Puerto A, el puerto A activa los ánodos y el puerto C activa los cátodos. Como ya se menciono solo una columna de leds esta activa a la vez.

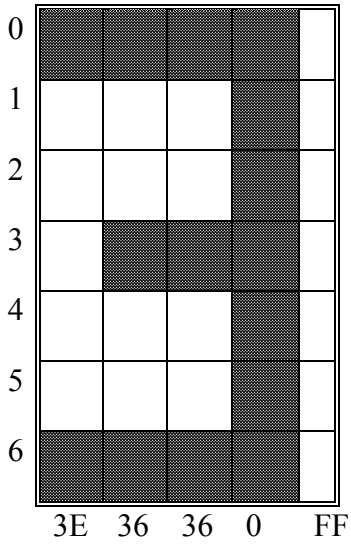
La tabla siguiente muestra como se activa cada punto de la matriz así como el número en hexadecimal que se debe de enviar por el puerto C para activar los leds correspondientes en cada columna para formar el carácter que se desea desplegar en la matriz.

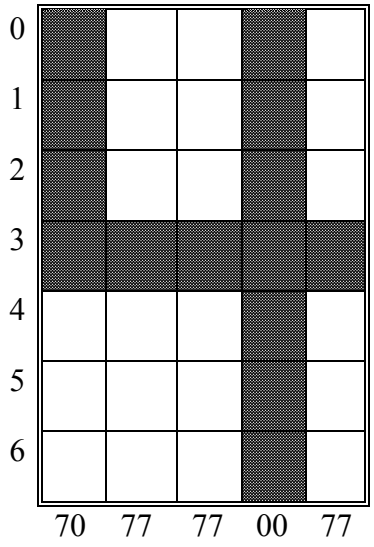
Diagrama de configuración de matriz de leds:

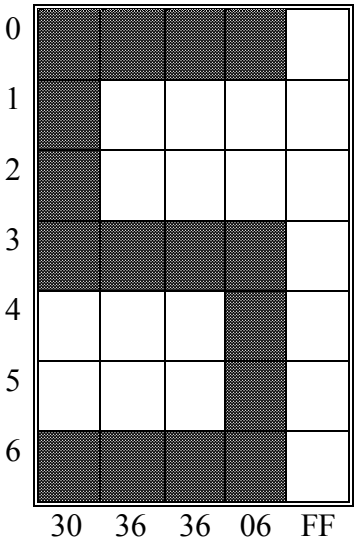


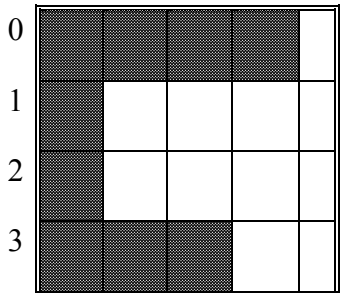


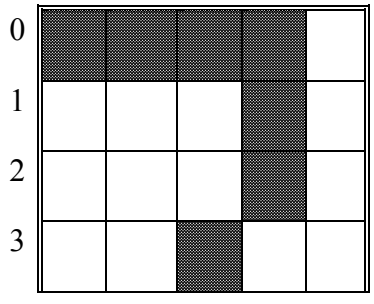


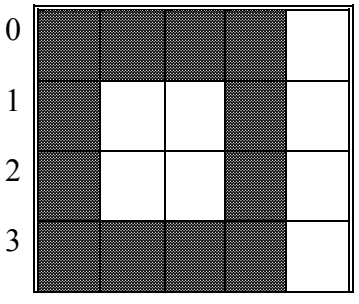


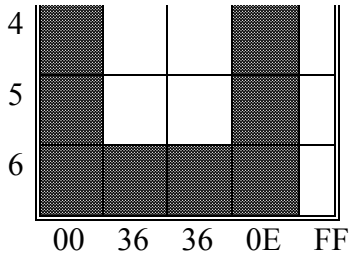


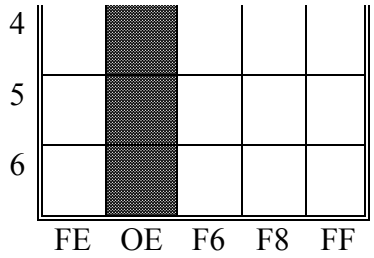


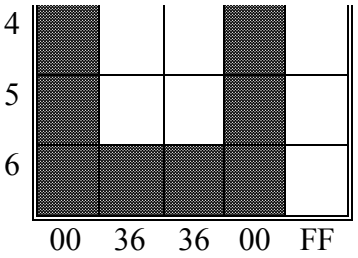


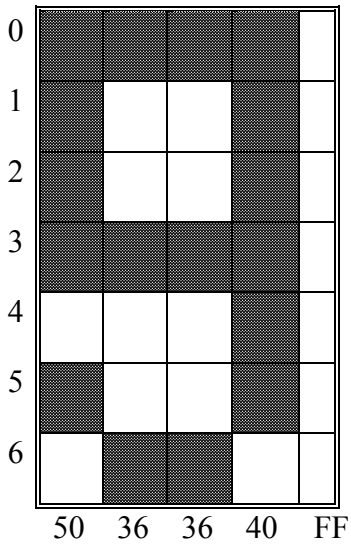


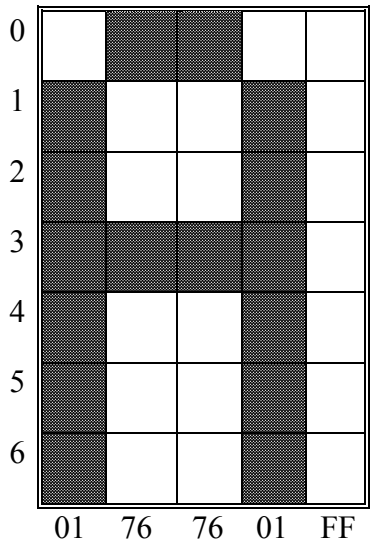


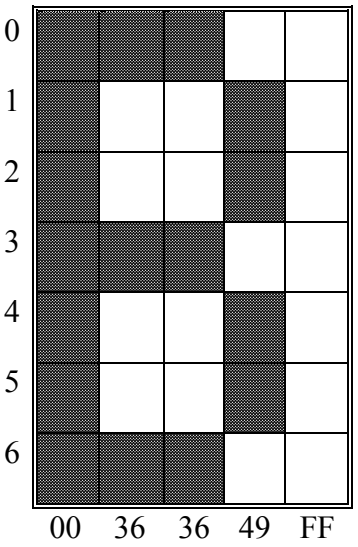

4
5
6
00 36 36 0E FF

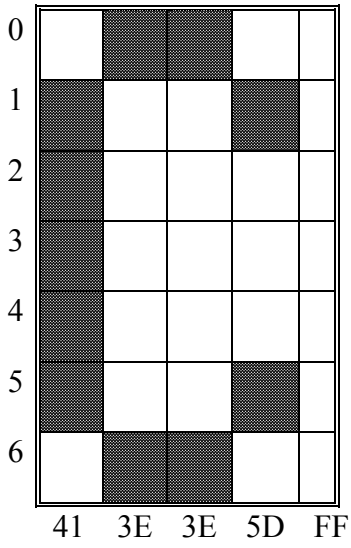

4
5
6
FE OE F6 F8 FF

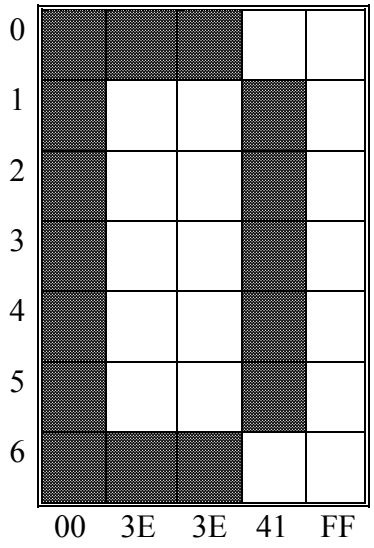

4
5
6
00 36 36 00 FF

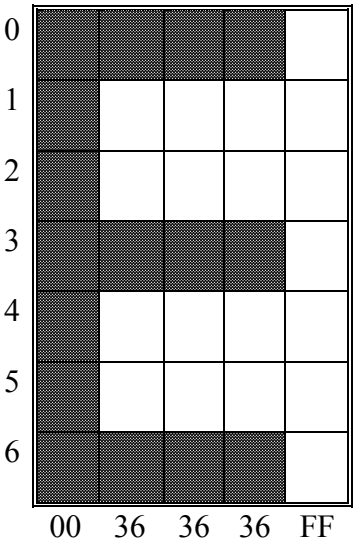

0
1
2
3
4
5
6
50 36 36 40 FF


0
1
2
3
4
5
6
01 76 76 01 FF


0
1
2
3
4
5
6
00 36 36 49 FF


0
1
2
3
4
5
6
41 3E 3E 5D FF


0
1
2
3
4
5
6
00 3E 3E 41 FF


0
1
2
3
4
5
6
00 36 36 36 FF

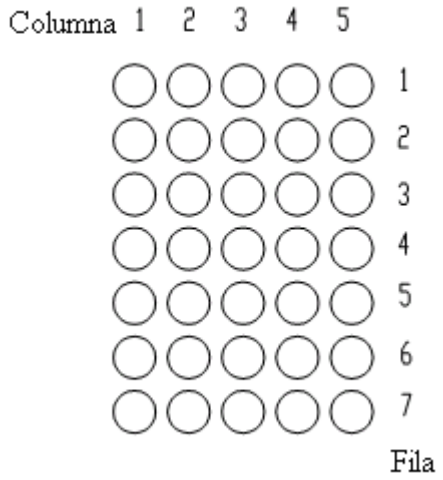


Figura 10.1 Diagrama físico de una matriz de leds.

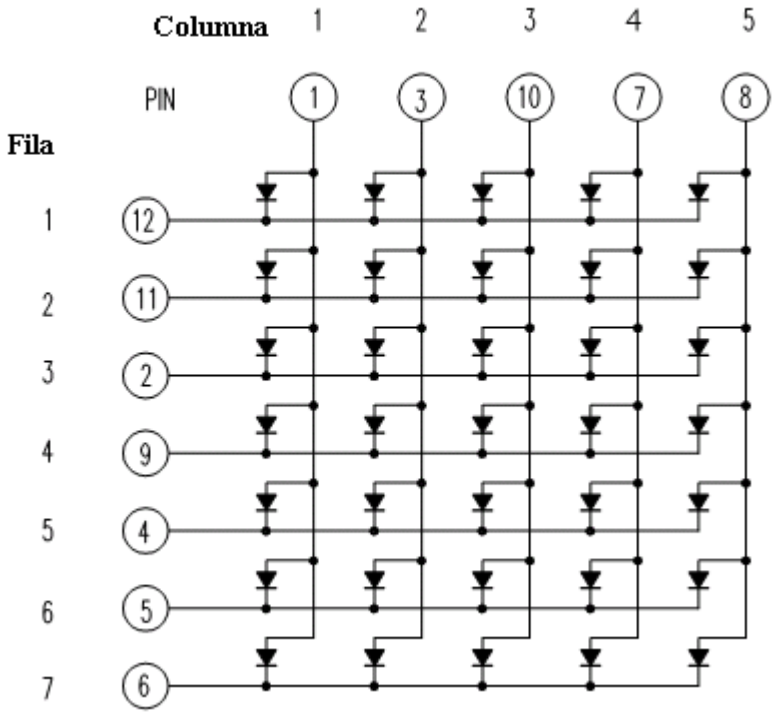


Figura 10.2 Diagrama de una matriz de leds

10.1.2 Código

```
#include "avr\io.h"

void MuestraChar(unsigned char mChar);
void IniPuertos(void);
void delay();

struct Display
{
  char Col[5];
};

/*
        Arreglo de estructuras que determinan que leds deben
        activarse en la matriz de leds.
*/
struct Display Dig[]=
{
0x00,0x3E,0x3E,0x00,0xFF,//0
0xFF,0xFD,0x00,0xFF,0xFF,//1
0x1E,0x2E,0x36,0x38,0xFF,//2
0x3E,0x36,0x36,0x00,0xFF,//3
0x70,0x77,0x77,0x00,0x77,//4
0x30,0x36,0x36,0x06,0xFF,//5
0x00,0x36,0x36,0x0E,0xFF,//6
```

```
0xFE,0x0E,0xF6,0xF8,0xFF,//7
0x00,0x36,0x36,0x00,0xFF,//8
0x50,0x36,0x36,0x40,0xFF,//9
0x01,0x76,0x76,0x01,0xFF,//A
0x00,0x36,0x36,0x49,0xFF,//B
0x41,0x3E,0x3E,0x5D,0xFF,//C
0x00,0x3E,0x3E,0x41,0xFF,//D
0x00,0x36,0x36,0x36,0xFF,//E
0x00,0x76,0x76,0x76,0xFF,//F
};

/*
        Rutina principal, punto d entrada al programa.
*/
int main(void)
{
        unsigned int j=0; //usado como contador
        unsigned char contador = 0;
        IniPuertos();


        PORTC =0x00;
        for(;;)
        {
```

```
                MuestraChar(contador);


        if(j>= 150)
                {
                  j=0;
                  contador++;
                }
                else
                {
                 j++;
                }

                if(contador > 0x0F)
                  contador = 0x00;

        }
}

/*
        Despliega caracteres en la matriz de leds
*/
void MuestraChar(unsigned char mChar)
{
   unsigned char Scan =1;
        unsigned char cCol = 0;
```

```
        //char col_valor;



        for(;;)
    {
                PORTA = Scan;
                PORTC = Dig[mChar].Col[cCol];
                delay();
              PORTC =0xFF;
                Scan = Scan << 1;
                cCol++;
                if(Scan == 0x20)
                   break;

        }

}

//Inicializa puertos
void IniPuertos(void)
{
        DDRA = 0xFF;          //Inicializa puerto como salida
        DDRC = 0xFF;          //Inicializa puerto como salida
        DDRD = 0xFF;          //Inicializa puerto como salida
        PORTA = 0x00;
```

```
        PORTC = 0x00;
        PORTD = 0x00;

}



/*
        Rutina de retardo
*/
void delay()
{
        static int j,k;

        for(j=0;j<100;j++)
          for(k=0;k<1;k++);
}
```

10.1.3 Diagrama esquemático

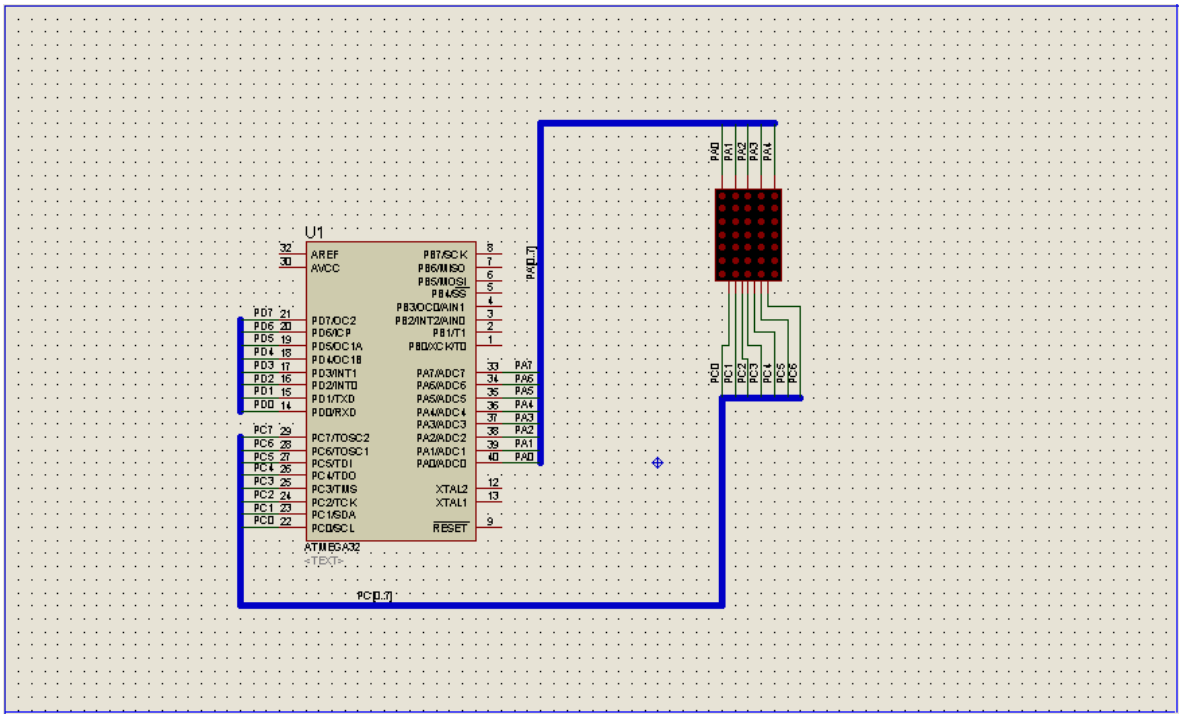


Diagrama de Atmega32 controlando una matriz de leds

10.2 Control de matriz de interruptores y matriz de leds.

10.2.1 Funcionamiento

Esta aplicación controlara una matriz de interruptores y desplegara el numero correspondiente a la tecla presionada en la matriz de leds. La operación de la matriz de leds es la misma de la aplicación 10.1 y el funcionamiento de la matriz de led se explicara en el párrafo siguiente.

En el diagrama siguiente se muestra como están interconectados los interruptores en la matriz d leds, se puede apreciar que cada una de las patillas de los interruptores están unidas en cada fila y la otra patilla esta unida a las otras patillas de los interruptores de la misma fila.

El principio de operación es muy simple, se activa una señal en una de las filas y se lee un posible resultado en las columnas. Por ejemplo se activa la fila uno y se lee el valor de las columnas, el valor en las columnas nos indica la tecla presionada en la primera fila esto en caso de que alguna tecla de la primera fila se haya presionado, posteriormente se realiza la misma operación para las filas restantes en forma consecutiva, la secuencia de activación se realiza a una determinada velocidad para que el usuario no detecte retraso en la respuesta del sistema cuando alguna tecla es activada.

El programa se encargara de decodificar el valor de las columnas en combinación con la fila activada y de esta forma podrá determinar la tecla presionada y generara la secuencia de señales correspondientes en los puertos C y A para desplegar un numero en la matriz de leds correspondiente a la tecla presionada.

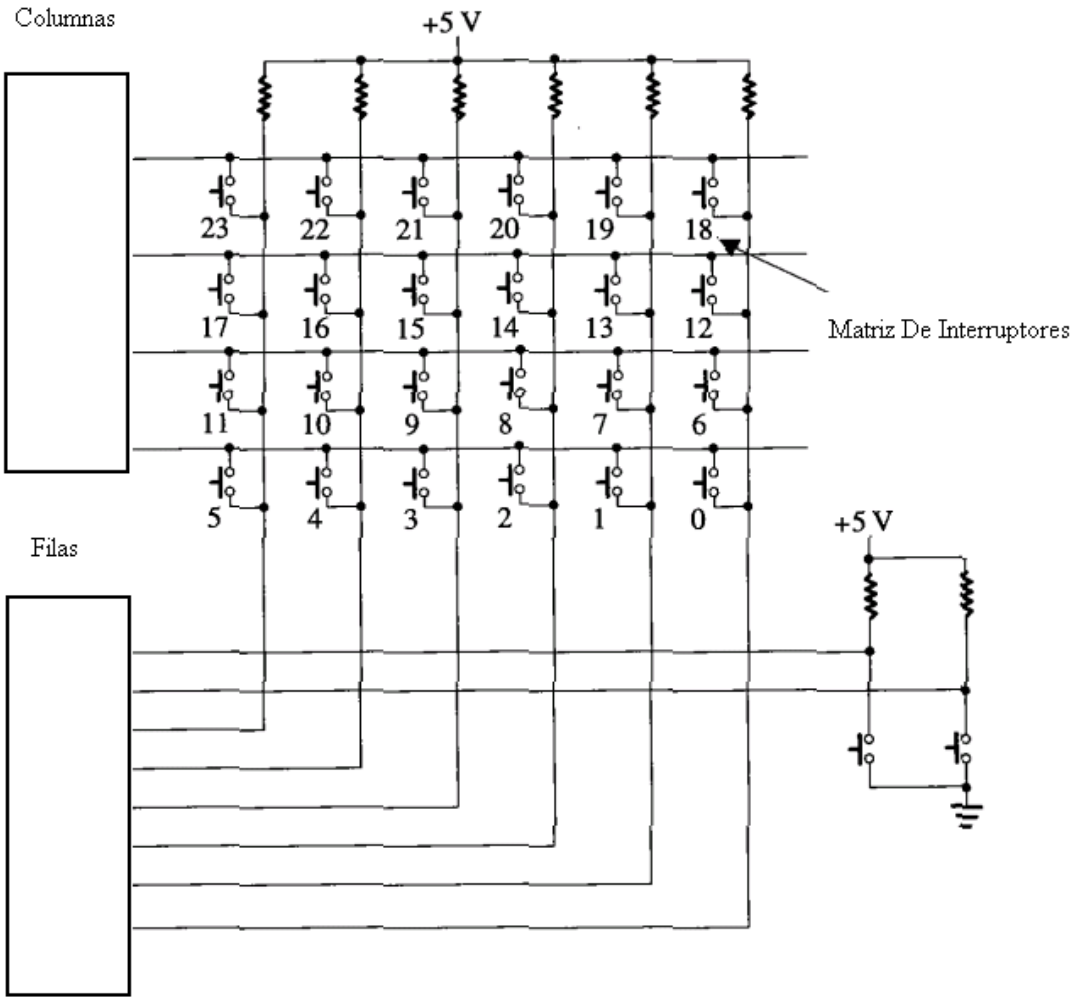


Figura 10.3 Diagrama de una matriz de interruptores 6 x 4

10.2.2 Código fuente

```
#include "avr\io.h"

unsigned char LeeTecla(void);

void MuestraChar(unsigned char mChar);

void IniPuertos(void);

void delay();


struct Display

{

  char Col[5];

};



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

 Arreglo de estruturas on informacion

  de los leds que deben activarse en el barrido

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

struct Display Dig[]=

{

0x00,0x3E,0x3E,0x00,0xFF,//0

0xFF,0xFD,0x00,0xFF,0xFF,//1

0x1E,0x2E,0x36,0x38,0xFF,//2

0x3E,0x36,0x36,0x00,0xFF,//3

0x70,0x77,0x77,0x00,0x77,//4

0x30,0x36,0x36,0x06,0xFF,//5
```

```
0x00,0x36,0x36,0x0E,0xFF,//6
0xFE,0x0E,0xF6,0xF8,0xFF,//7
0x00,0x36,0x36,0x00,0xFF,//8
0x50,0x36,0x36,0x40,0xFF,//9
0x01,0x76,0x76,0x01,0xFF,//A
0x00,0x36,0x36,0x49,0xFF,//B
0x41,0x3E,0x3E,0x5D,0xFF,//C
0x00,0x3E,0x3E,0x41,0xFF,//D
0x00,0x36,0x36,0x36,0xFF,//E
0x00,0x76,0x76,0x76,0xFF,//F
};



/*****************************************************************
        Rutina principal

*****************************************************************/
int main(void)
{
        unsigned int j=0; //usado como contador
        unsigned char tecla = 0;

        IniPuertos();
```

```
	PORTC =0x00;
	for(;;)
	{

		MuestraChar(LeeTecla());

	}
}

/*****************************************************************
	Esta funcion explora la matriz de interruptores

*****************************************************************/
unsigned char LeeTecla(void)
{
	unsigned char Scan =1;
	unsigned char valorleido;
	for(;;)
	{

		PORTD = Scan;
		valorleido = PIND;
```

```
        if((valorleido & 0xF0) != 0)
        {



                switch(Scan)
                {
                        case 1:
//Exploracion de Columna 1
                        switch((valorleido>>4) & 0x0F)
                                {
                                        case 1:
//Fila 1
                                                return 0x0A;
                                        break;
                                        case 2:
//Fila 2
                                                return 0x0B;
                                        break;
                                        case 4:
//Fila 3
                                                return 0x0C;
                                        break;
                                        case 8:
//Fila 4
                                                return 0x0D;
                                        break;
                                }
```

```
                    break;
                    break;
                case 2:
//Exploracion de Columna 2
                    switch((valorleido>>4) & 0x0F)
                    {
                        case 1:
//Fila 1
                            return 0x09;
                        break;
                        case 2:
//Fila 2
                            return 0x06;
                        break;
                        case 4:
//Fila 3
                            return 0x03;
                        break;
                        case 8:
//Fila 4
                            return 0x0E;
                        break;
                    }
                    break;
                case 4:
//Exploracion de Columna 3
                    switch((valorleido>>4) & 0x0F)
```

```
                {
                    case 1:
        //Fila 1
                        return 0x08;
                    break;
                    case 2:
        //Fila 2
                        return 0x05;
                    break;
                    case 4:
        //Fila 3
                        return 0x02;
                    break;
        //Fila 4
                    case 8:
                        return 0x00;
                    break;
                }
              break;
              break;
            case 8:

//Exploracion de Columna 4
                switch((valorleido>>4) & 0x0F)
                {
                    case 1:
        //Fila 1
```

```
                                        return 0x07;
                                    break;
                                    case 2:
//Fila 2
                                        return 0x04;
                                    break;
                                    case 4:
//Fila 3
                                        return 0x01;
                                    break;
                                    case 8:
//Fila 4
                                        return 0x0F;
                                    break;
                                }
                            break;
                            break;

                        }
                    }

                    Scan = (Scan << 1) & 0x0F;

                    if(Scan == 0x00)
                       break;
```

```
        }

 return 0x10;
}


/*****************************************************************
        Rutina que muestra el caracter en la matriz de leds

*****************************************************************/
void MuestraChar(unsigned char mChar)
{
   unsigned char Scan =1;
        unsigned char cCol = 0;

  if(mChar >= 0x10)
    return;

        for(;;)
   {
                PORTA = Scan;
                PORTC = Dig[mChar].Col[cCol];
                delay();
            PORTC =0xFF;
```

```
                Scan = Scan << 1;
                cCol++;
                if(Scan == 0x20)
                   break;
        }
}



/*****************************************************************
Inicializa puertos
*****************************************************************/
void IniPuertos(void)
{
        DDRA = 0xFF;        //Inicializa puerto como salida
        DDRC = 0xFF;        //Inicializa puerto como salida
        DDRD = 0x0F;        //Inicializa puerto como Salida/Entrada
        PORTA = 0x00;
        PORTC = 0x00;
        PORTD = 0x00;

}

/*****************************************************************
  Rutina de retardo
```

```
   No calculada solo proporciona unos cuantos mS de retardo.
**************************************************************/
void delay()
{
        static int j,k;
        for(j=0;j<100;j++)
          for(k=0;k<1;k++);


}
```

10.2.3 Diagrama esquemático

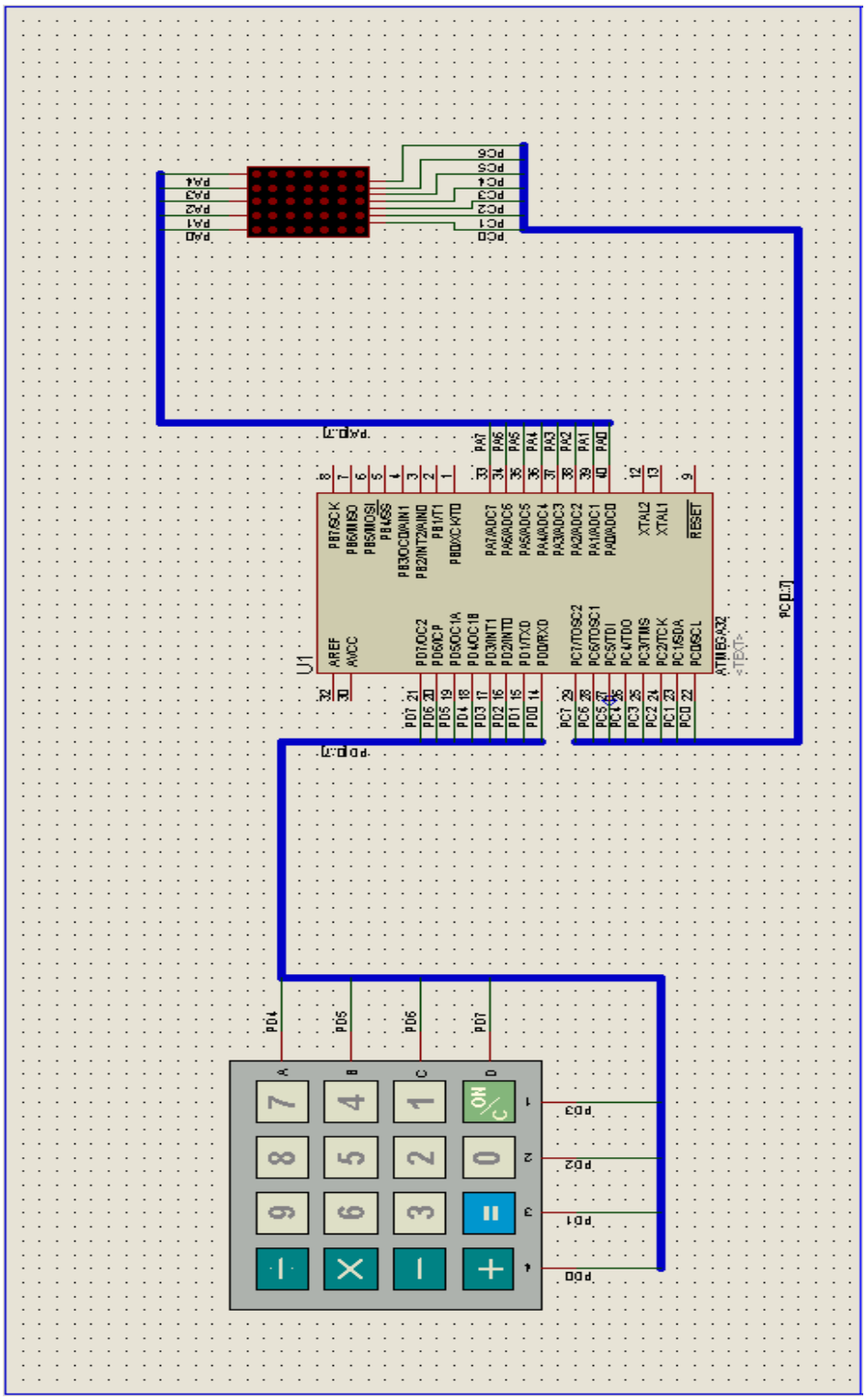


Figura 10.4

10.3 Aplicación 1 (Alarma con pantalla LCD reprográmable)

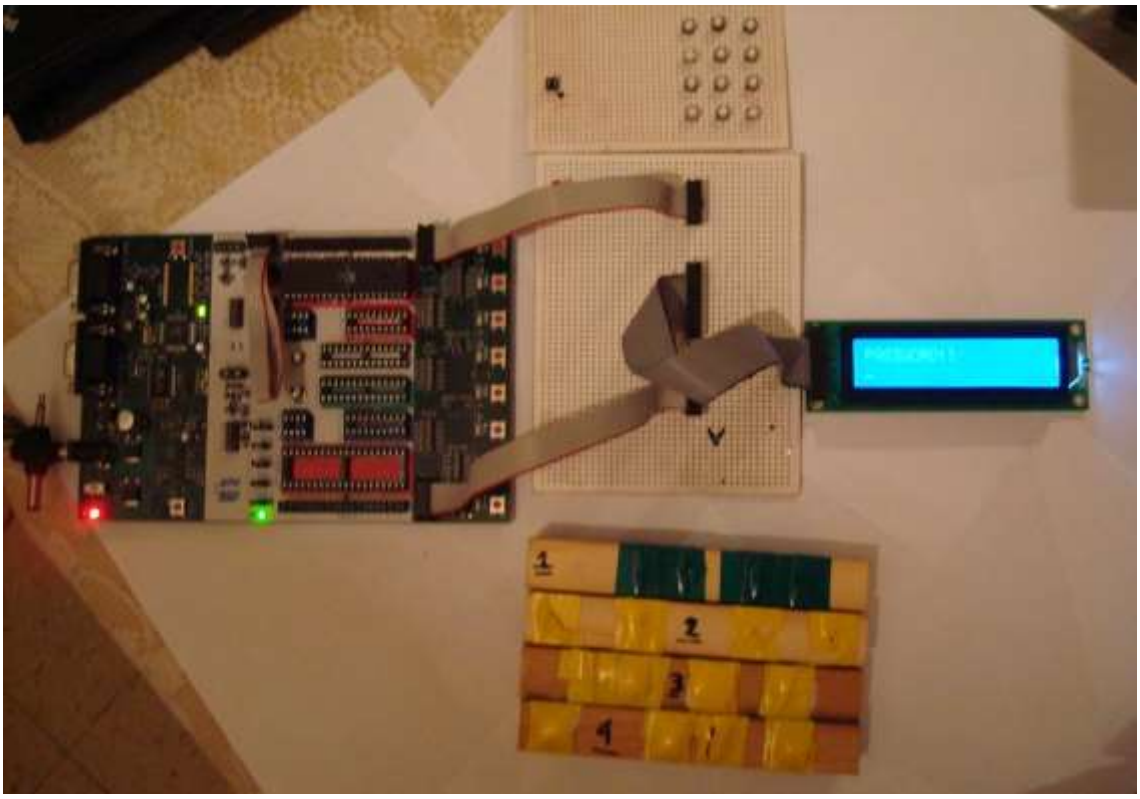


Figura 10.5

10.3.1 Descripción:

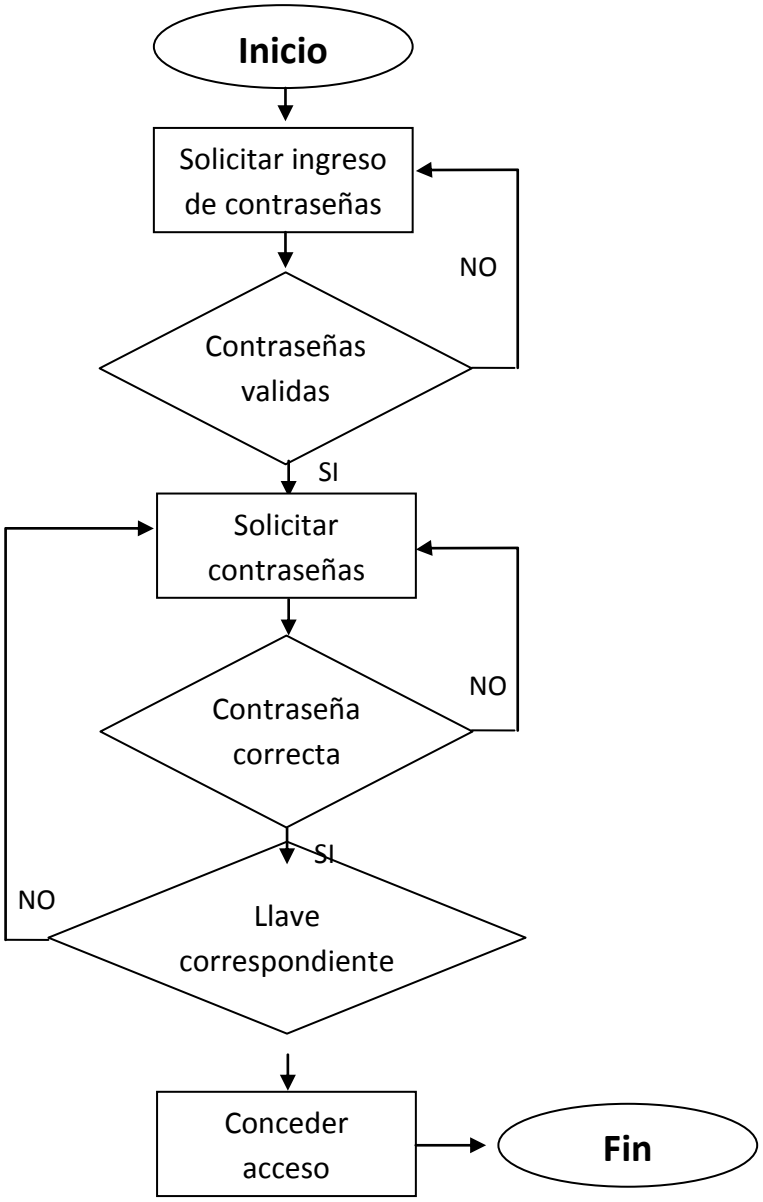
La aplicación controla un cerrojo de mediana seguridad que al ingresar una contraseña correcta y al detectar una llave mecánica, dará acceso. Puede ser utilizado para varios fines, como tal el poder abrir una puerta, poder encender un auto, una caja de seguridad, etc.

10.3.2 Funcionamiento:

Al encender el cerrojo y así iniciar el programa, en la pantalla utilizada se desplegara un mensaje “PASSWORD:1” solicitando que sea ingresada la primera de las 4 contraseñas que puede contener el dispositivo.

Una vez que se hayan ingresado las 4 contraseñas, mandara el mensaje de “ PASSWORD:”. Al ser acezada una de las posibles contraseñas, se deberá tener agregada una llave a un segundo mecanismo, que será el que al final dará acceso corroborando que la contraseña corresponde a la llave ingresada.

10.3.3 Diagrama de Flujo



10.3.4 Código Fuente:

```
#include <avr/io.h>
#include "delay.h"
unsigned char M, lectura;
unsigned char x, y;
unsigned int a, numero, pass;
unsigned int pass2, pass3, pass4, np1;
```

En esta sección estamos declarando las variables que usaremos en el transcurso del programa y también llamamos a las librerías que usaremos.

```
void delay()
{  unsigned char a;
   for (a = 1; a; a++)
    ; ;
}

void inicio (unsigned char letra)
{
PORTC=0X01;
PORTA=letra; wait(.5);
PORTC=0X05;  wait(5);
PORTC=0X01;  wait(.5);
PORTA=0X00;  wait(5);
}
```

Se declaran dos funciones que nos ayudaran en el uso del manejo de la pantalla. Un retraso y un código que se necesita en el bus de datos de la pantalla para mandar el código que al final nos mostrara el mensaje.

```
void letras()
{
PORTC=0X00;
PORTA=0X01;  wait(.5);
PORTC=0X04;  wait(5);
PORTC=0X00;  wait(.5);
PORTA=0X00;  wait(5);
wait(.5);
inicio(0X50);  wait(.5);          (P)
inicio(0X41);  wait(.5);          (A)
inicio(0X53);  wait(.5);          (S)
inicio(0X53);  wait(.5);          (S)
inicio(0X57);  wait(.5);          (W)
inicio(0X4F);  wait(.5);          (O)
inicio(0X52);  wait(.5);          (R)
inicio(0X44);  wait(.5);          (D)
inicio(0X3A);  wait(.5);          (:)
PORTC=0X00;
PORTA=0XC0;  wait(.5);
PORTC=0X04;  wait(5);
PORTC=0X00;  wait(.5);
```

Función declarada para mandar visualizar el mensaje de “PASSWORD:” en la pantalla.

Contiene la información que necesita el bus de datos la pantalla para desplegar cada letra y realizar órdenes.

```
PORTA=0X00;  wait(5);
}
```

Función que nos ayudara a detectar que cual botón fue presionado en nuestra matriz de botones.

```
unsigned int leerpuerto()
{
wait(10);
M=0X20;
PORTC=M;
lectura=PIND;
if (lectura ==0x01)
{
for(;;)
{
if (PIND ==0x00)
{
PORTC=0X01;
PORTA=0x31; wait(.5);
PORTC=0X05; wait(5);
PORTC=0X01;
PORTA=0X00; wait(5);
break;
}
}
a=a+1;    y=y+1;
}

if(lectura ==0x02)
{
for(;;)
{
if (PIND ==0x00)
{break;}
}
a=a+1;    y=y+4;
PORTC=0X05;
PORTA=0x34;    wait(1);
PORTC=0X01;
PORTA=0X00;
}
if (lectura ==0x04)
{
for(;;)
}
lectura=PIND;
```

```
if (lectura ==0x00)
break;
}
a=a+1;    y=y+7;
PORTC=0X05;
PORTA=0x37;    wait(1);
PORTC=0X01;
PORTA=0X00;  }

if (lectura ==0x08)
{ for(;;)
{
lectura=PIND;
if (lectura ==0x00)
break;}
if (a==6)
{ numero=y; }
else
{ letras();
  numero=0; }
a=0; y=0;
return(numero);
}
wait(5);
M=0X40; PORTC=M;
lectura=PIND;
if (lectura ==0x01)
{ for(;;)
{ lectura=PIND;
if (lectura ==0x00)
{ break; }  }
a=a+1;    y=y+2;
PORTC=0X05;
PORTA=0x32;    wait(1);
PORTC=0X01;    PORTA=0X00; }


if (lectura ==0x02)
{ for(;;)
{ lectura=PIND;
if (lectura ==0x00)
{ break; } }
a=a+1;    y=y+5;
PORTC=0X05;
PORTA=0x35;    wait(1);
```

```
PORTC=0X01;    PORTA=0X00; }

if (lectura ==0x04)
{ for(;;)
{ lectura=PIND;
if (lectura ==0x00)
break; }
a=a+1;    y=y+8;
PORTC=0X05;
PORTA=0x38;    wait(1);
PORTC=0X01;    PORTA=0X00; }


if (lectura ==0x08)
{ for(;;)
{ lectura=PIND;
if (lectura ==0x00)
break; }
a=a+1;    y=y+0;
PORTC=0X05;
PORTA=0x30;   wait(1);
PORTC=0X01;   PORTA=0X00;   }

wait(5);
M=0X80;  PORTC=M;  lectura=PIND;
If (lectura ==0x01)
{ for(;;)
{ lectura=PIND;
If (lectura ==0x00)
{ break; }  }
a=a+1;    y=y+3;
PORTC=0X05;
PORTA=0x33;    wait(1);
PORTC=0X01;    PORTA=0X00;  }

if (lectura ==0x02)
{ for(;;)
{ lectura=PIND;
if (lectura ==0x00)
{ break; }  }
a=a+1;    y=y+6;
PORTC=0X05; PORTA=0x36;
wait(1);
PORTC=0X01; PORTA=0X00;  }
```

Numero 0

Numero 3

Numero 6

```
if (lectura ==0x04)
{ for(;;)
{ lectura=PIND;
if (lectura ==0x00)
break;  }
a=a+1;    y=y+9;
PORTC=0X05; PORTA=0x39;
wait(1);
PORTC=0X01; PORTA=0X00;   }
```

```
if (lectura ==0x08)
{ for(;;)
{ lectura=PIND;
if (lectura ==0x00)
break; }
a=0;    y=0;
PORTA=0xC0; PORTC=0X04;
delay();
PORTC=0X00; PORTA=0X00;  }
return(0); }
```

RESET

```
void pedirpass()
{ letrascambiarpass();
PORTC=0X01; PORTA=0x31;
wait(.5);
PORTC=0X05; wait(5);
PORTC=0X01;
PORTA=0X00; wait(.5);
PORTC=0X00;
PORTA=0XC0; wait(.5);
PORTC=0X04; wait(5);
PORTC=0X00; wait(.5);
PORTA=0X00; wait(5);
for(;;)
{ pass1=leerpuerto2();
if (pass1 != 0)
```

Mandamos el código para visualizar el numero “1” después del mensaje “Password:”

Entramos a un “for” y mandamos lo leído en la matriz de interruptores a una variable denominada pass1. Si pass1 es diferente de cero, aceptara la clave y seguirá con la siguiente

```
{break;}
}

letrascambiarpass();
PORTC=0X01; PORTA=0x32;
wait(.5);
PORTC=0X05; wait(5);
PORTC=0X01;
PORTA=0X00; wait(.5);
PORTC=0X00;
PORTA=0XC0; wait(.5);
PORTC=0X04; wait(5);
PORTC=0X00; wait(.5);
PORTA=0X00; wait(5);
for(;;)
{ pass2=leerpuerto2();
if (pass2 != 0)
break;  }
```

Como anteriormente lo hicimos, mandamos llamar el mensaje “Password:” seguido del numero “2”. Mandamos lo leido en la matriz de interruptores a una variable pass2 y si es diferente de cero, aceptara la clave.

```
letrascambiarpass();
PORTC=0X01; PORTA=0x33;
wait(.5);
PORTC=0X05; wait(5);
PORTC=0X01; PORTA=0X00;
wait(.5);
PORTC=0X00; PORTA=0XC0;
wait(.5);
PORTC=0X04; wait(5);
PORTC=0X00; wait(.5);
PORTA=0X00; wait(5);
for(;;)
{ pass3=leerpuerto2();
if (pass3 != 0)
break;
}
letrascambiarpass();
PORTC=0X01; PORTA=0x34;
wait(.5);
PORTC=0X05; wait(5);
PORTC=0X01; PORTA=0X00;
wait(.5);
PORTC=0X00; PORTA=0XC0;
```

```
wait(.5);
PORTC=0X04; wait(5);
PORTC=0X00; wait(.5);
PORTA=0X00; wait(5);
for(;;)
{ pass4=leerpuerto2();
if (pass4 != 0)
break;}
np1=0;
}

int main()
{
DDRD=0XF0;
DDRB=0X0F;
DDRA=0XFF;
DDRC=0XFF;
unsigned int valor, c;
```

Inicializamos los puertos

```
PORTC=0X00;
PORTA=0X3F;  wait(.5);
PORTC=0X04;  wait(5);
PORTC=0X00;  wait(.5);
PORTA=0X00;  wait(5);
PORTC=0X00;
PORTA=0X0F;  wait(.5);
PORTC=0X04;  wait(5);
PORTC=0X00;  wait(.5);
PORTA=0X00;  wait(5);
PORTC=0X00;
PORTA=0X01;  wait(.5);
PORTC=0X04;  wait(5);
PORTC=0X00;  wait(.5);
PORTA=0X00;  wait(5);
PORTC=0X00;
PORTA=0X06;  wait(.5);
PORTC=0X04;  wait(5);
PORTC=0X00;  wait(.5);
PORTA=0X00;  wait(5);
```

Mandamos los códigos necesarios a la pantalla para inicializarla

```
pedirpass();

letras();

for(;;)
{ c=0;
valor=leerpuerto();
if (valor != 0)
for(;;)
{ if ((valor==pass1)&&(PINB==0XC9))
{c=3; PORTC=0X88;}
if ((valor==pass2)&&(PINB==0X5A))
{c=3; PORTC=0X88;}
if ((valor==pass3)&&(PINB==0X95))
{c=3; PORTC=0X88;}
if ((valor==pass4)&&(PINB==0X65))
{c=3; PORTC=0X88;}
if (valor==2)
pedirpass(); }

if (c != 3)
{ letras();
break; }
if (PIND==0X08)
{ valor=0;
letras();
break; } } } }
```

Si alguna de las contraseñas cumple junto con la combinación en el puerto D (llave) se abrirá el cerrojo mandando un pulso a través del puerto C.

Si no se detecta la entrada de ninguna combinación correcta, iniciara el display y el programa mandando el mensaje de “PASSWORD” de nuevo en el display

10.3.5 Diagrama en PROTEUS

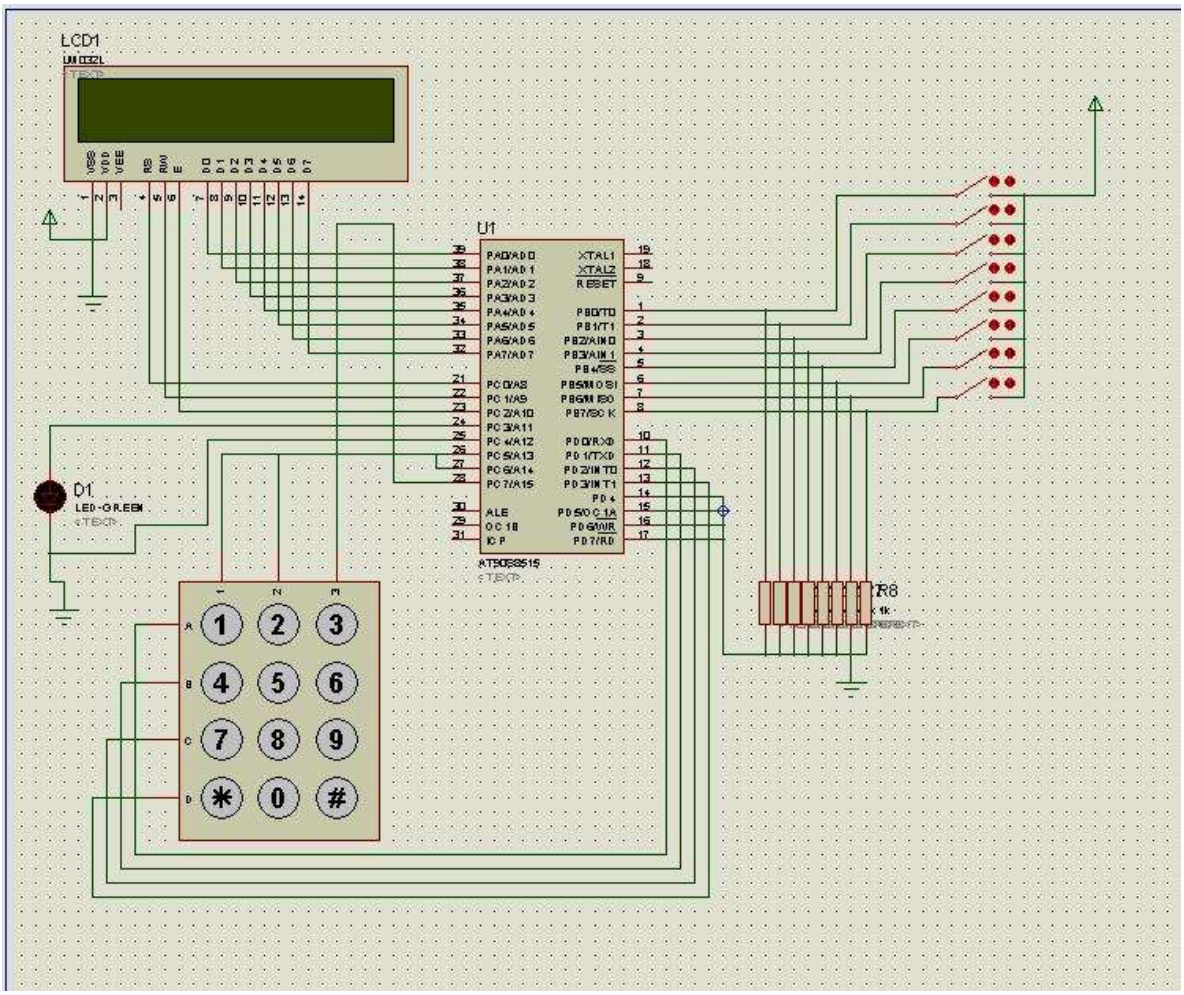


Figura 10.6

CAPITULO 11

FREERTOS

11.1 Que es FreeRTOS?

FreeRTOS es un sistema operativo de tiempo real el cual ha sido portado a varios microcontroladores entre los que se encuentran algunos de la familia ATMEL como el Atmega64, Atmega128 o Atmega256.

FreeRTOS es un Sistema operativo código abierto el cual permite al implementación del RTOS en aplicaciones comerciales sin pagar derechos de autor.

Algunas de las características que distinguen a FRERTOS son las siguientes:

- Es portable.
- Es compacto
- Es Código Abierto
- Permite su uso en aplicaciones comerciales
- Existe soporte de los autores (No es gratis)
- Existe una versión certificada llamada SAFERTOS(No es gratis)
- Es simple
- Esta bien documentado

- El paquete de instalación incluye ejemplos de como implementarlo en un proyecto

11.2 Filosofía de diseño de FreeRTOS:

- **Simplicidad**
- **Portable**
- **Conciso**

Todo el código esta escrito en C, con algunas pequeñas excepciones las cuales son inevitables y tuvieron que ser escritas en ensamblador.

11.3 Características De Las Tareas

FreeRTOS permite que una aplicación de tiempo real sea estructurada como un grupo de tareas autónomas solamente. Este es el modelo tradicional usado por un planificador RTOS.

Una aplicación de tiempo real que utiliza RTOS se puede estructurar como un grupo de tareas independientes. Cada tarea ejecuta dentro de su propio contexto sin dependencia coincidente en otras tareas dentro del grupo o del planificador mismo.

Solamente una tarea dentro de una aplicación puede ejecutarse en cualquier punto o momento en tiempo real el planificador de tareas es responsable de decidir a que tarea debe asignar tiempo. El planificador puede por lo tanto comenzar repetidamente y parar cada tarea (intercambie cada tarea adentro y hacia fuera) como la aplicación se ejecute. Ya que una tarea no tiene ningún conocimiento de la actividad del planificador, es responsabilidad del planificador en tiempo real asegurarse del proceso del contexto (valores del registro, contenido del apilado, etc) cuando una tarea se intercambia adentro es exactamente que como cuando la misma tarea fue intercambiada hacia fuera. Para

recuperar cada tarea se proporciona cada tarea tiene su propia pila. Cuando la tarea se intercambia fuera del contexto de la ejecución se guarda el apilado de esa tarea así que puede también ser restaurado exactamente cuando la misma tarea se restaure.

11.3.1 Resumen de Tareas

- Simple.
- No restringe su uso.
- Soporte completo de derecho preferente.
- Prioridad completamente dada.
- Cada tarea mantiene su propia pila dando por resultado un uso más alto de la RAM.
- La Re-entrada debe ser cuidadosamente considerado si se esta usando el derecho preferente.

11.4 Características De Las Co-Rutinas

1.- Uso del apilado

Todas las co-rutinas dentro de una aplicación comparten un solo apilado. Esto reduce grandemente la cantidad de RAM requerida comparada a un uso similar escrito usando tareas.

2.-planificador y prioridades.

Las Co-rutinas se planifican cooperativamente dando la prioridad con respecto a otras co-rutinas, pero se pueden incluir en un uso que utilice tareas con derecho preferente.

3.- Implementación Macro.

La implementación de co-rutinas es proporcionada a través de un grupo de macros.

4.- Restricción en uso.

La reducción en uso de la RAM viene del costo de algunas restricciones rigurosas en cómo las co-rutinas pueden ser estructuradas.

11.5 Semáforo.

Un semáforo es un mecanismo para sincronizar el acceso a recursos en un sistema multitarea.

En FreeRTOS la funcionalidad del semáforo binario, de cuenta y del mutex es proporcionada por un sistema de macros.

Las macros utilizan la puesta en práctica de la cola mientras que ésta proporciona todo lo necesario sin código adicional o la prueba por encima.

CAPITULO 12

CONCLUSIONES

12.1 Conclusiones

El enfoque propuesto permitirá el desarrollo de prototipos más complejos con procedimientos profesionales en menor tiempo.

Con el uso de las herramientas descritas un estudiante podrá tener su propio laboratorio de diseño en casa con una inversión muy pequeña y el uso de una computadora personal.

Las ventajas del uso de herramientas modernas son muchas entre las que se encuentran:

- Diseño de sistemas complejos profesionales.
- Programas con menos error y mas fáciles de mantener.
- Menor inversión en el diseño del prototipo, ya que el diseño puede ser simulado antes de su fabricación.

Como resumen final diremos que el paradigma de la extrema complejidad de programar microcontroladores los cuales comúnmente esta unidos al termino ensamblador queda eliminada y afirmamos que la programación de sistemas basados en microcontroladores puede ser tan fácil como la programación de computadoras personales.

BIBLIOGRAFÍA

El soporte bibliográfico usado en esta tesis se relaciona con maestros de universidades, material técnico de los fabricantes Atmel, Microchip y motorota así como también diversos sitios del Internet y el siguiente listado de libros:

- Embedded C Programming And the Atmel AVR.
  Autor: Barnett, Cox & O' Cull
  Editorial Thomson
- Microcontrller Techonology The 68HC11.
  Autor: Meter Spasov
  Editorial: Prentice Hall
- Introducción a los microcontroladores
  Autor: José Adolfo González Vázquez
  Editorial: McGraw Hill
- What is a microcontroller
  Autor: Andy Lidsay
  Editorial: Parallax inc.
- C Programming For Microcontrollers
  Autor: Joe Pardue
  Editorial: Smiley Micros
- The Microcontroller Idea Book
  Autor: Jan Axelson
- Embedded Systems Building Blocas
  Autor: Jean Labrose
  Editorial: R&D Books

INDICE DE FIGURAS

Figura 2.1 7

Figura 2.2 13

Figura 3.1 21

Figura 3.2 21

Figura 3.3 22

Figura 3.4 23

Figura 3.5 24

Figura 3.6 25

Figura 3.7 26

Figura 3.8 26

Figura 4.1 29

Figura 4.2 29

Figura 4.3 30

Figura 4.4 30

Figura 4.5 31

Figura 4.7 31

Figura 4.8 33

Figura 4.9 34

Figura 4.10 35

Figura 4.11 35

Figura 4.12 36

Figura 4.13 36

Figura 4.14 39

Figura 5.3 55

Figura 5.4 55

Figura 6.1 57

Figura 7.1 61

Figura 7.2 62

Figura 9.1 84

Figura 9.3 88

Figura 10.3 100

Figura 10.4 111

Figura 10.5 112

Figura 10.6 123

INDICE DE TABLAS

Tabla 4.1 31

Tabla 4.2..... 32

Tabla 4.3..... 37