ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

TESINA DE SEMINARIO

"DISEÑO DE PLATAFORMA DE TRABAJO PARA MICROCONTROLADORES ATMEL CON VISUALIZACIÓN DE SALIDAS EN TERCER ESTADO (TRI-STATE) PARA PROBAR SU ARQUITECTURA CON ELEMENTOS DE ENTRADA Y SALIDA"

Previo a la obtención del título de:

INGENIERO EN ELECTRICIDAD

ESPECIALIZACIÓN ELECTRÓNICA Y AUTOMATIZACIÓN INDUSTRIAL

Presentado por:

Marlon German Contreras Urgiles

Arturo Mateo Ayala Rocafuerte

GUAYAQUIL – ECUADOR

AÑO 2012

AGRADECIMIENTO

Primero a Dios por que me a dado la oportunidad de vivir, estudiar y compartir todos los buenos momentos de la vida con quienes queremos y amamos.

A mis padres quienes con su esfuerzo y amor me han ayudado y apoyado siempre.

Y a todas aquellas personas que han sido amigos, compañeros, profesores que me han brindado su amistad desinteresada y sincera.

Marlon German Contreras Urgiles

AGRADECIMIENTO

A Dios por haber sido mi inspiración en todo el transcurso de mi carrera dándome fuerzas cuando lo necesitaba y bendiciéndome en el momento oportuno. A mis Padres y hermanos quienes siempre estuvieron a mi lado con su ayuda valida y desinteresada. A esta prestigiosa institución ESPOL por haberme dado unos ejemplares profesores a quienes admiro mucho por su dedicación y entrega a su profesión, y porque también me dio valiosos compañeros de aulas que han sido deseosos y perseverantes de lograr sus metas.

Arturo Mateo Ayala Rocafuerte

DEDICATORIA

A mis Padres, mi hermano, mi hermana por ser juntos una familia maravillosa, unida, leal y ejemplar que me han enseñado todo aquello que no se enseña en las aulas.

Marlon German Contreras Urgiles

A mi Madre, a mi hermano Leonardo Ayala, por su gran apoyo siempre que lo necesitaba

A mi Esposa e hijo por ser las razones que me motivan a lograr alcanzar mis metas.

Arturo Mateo Ayala Rocafuerte

TRIBUNAL DE SUSTENTACION

_____

Ing. Carlos Valdivieso.
Profesor del Seminario.

_____

Ing. Hugo Villavicencio.
Profesor Delegado del Decano.

DECLARACIÓN EXPRESA

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

(Reglamento de exámenes y títulos profesionales de la ESPOL)

_____

Marlon German Contreras Urgiles

_____

Arturo Mateo Ayala Rocafuerte

RESUMEN

El presente proyecto tiene como finalidad la elaboración e implementación de una plataforma de trabajo con el KIT AVR Butterfly de Atmel, para que en ella podamos montar, probar y corregir una amplia variedad de ejercicios que cubran todas o la mayoría de las capacidades que poseen los microcontroladores de Atmel, en este caso el Atmega169 que es el microcontrolador en el cual se sustenta el AVR Butterfly.

Nuestro proyecto se basa en los Puerto Entrada – Salida y Tri-state del Atmega169 por lo que se desarrollaron cinco ejercicios que muestran como podemos configurar y utilizar de diversas formas las capacidades de los Puertos I/O que posee Atmel con sus microcontroladores AVR.

Además en los primeros capítulos realizamos una revisión de dos de las más importantes empresas fabricantes de microcontroladores como son Intel y Microchip, en lo concerniente a la arquitectura y manejo de sus puertos de Entrada – Salida.

Posteriormente se presenta una completa guía de características y programación del Kit AVR Butterfly, así como también el manejo y utilización de las herramientas de software proporcionadas por Atmel en esta caso el IDE AVR Studio 4 y para la simulación de la circuitería electrónica y microcontroladores el software independiente de simulación PROTEUS.

INDICE DE TABLAS

INDICE DE FIGURAS

INDICE GENERAL

CAPITULO 2

HERRAMIENTAS DE HARDWARE Y SOFTWARE

CAPITULO 3

DESCRIPCION DE LA PLATAFORMA DE TRABAJO

CAPITULO 4

DESARROLLO DE EJERCICIOS EN LA PLATAFORMA DE TRABAJO

ABREVIATURAS

Bit	Unidad de medida de información equivalente a la elección entre dos posibilidades igualmente probables.
CPU	Unidad central de proceso.
PC	Computador personal.
USB	Bus serial universal.
ADC	Convertidor Analógico Digital
IDE	Integrated Development Environment
I/O	Input – Output
LCD	Liquid Cristal Display
PWM	Pulse-width modulation
HZ	High Impedance
LED	Light emitting Diode
RISC	Reduced Instruction Set Computer
JTAG	Joint Test Action Group

INTRODUCCION

Una parte fundamental que caracteriza a los microcontroladores son los Puertos de Entrada – Salida que poseen los mismos, ya que esta es una de las características que los diferencian de los microprocesadores que para utilizarlos deben implementar circuitería adicional que comunique su bus de datos y control a dichos puertos externos. Además la gran importancia que tiene la característica de tener un estado en alta impedancia de los puertos de entrada y salida que permite que la comunicación entre diversos dispositivos y el microcontrolador sea más eficiente. Durante el desarrollo de los microcontroladores esta característica se a mejorado desde los primeros modelos de Intel como son la serie MCS-51, que poseía una arquitectura básica y simple de manejo de Puertos de Entrada – Salida, con únicamente un Latch para manejar un pin de un puerto, hasta los modernos microcontroladores de Microchip y Atmel que poseen de dos a tres registros independientes para configurar dichos puertos.

Además que no solamente los Puertos de Entrada – Salida sirven para ese fin, sino que también poseen diversos funciones multiplexadas con la función principal de Puerto I/O lo que les da mayor importancia y funcionabilidad además esta característica varia de fabricante y modelo brindándonos una amplia gama de posibilidades para realizar nuestros diseños y proyectos.

CAPITULO 1

PUERTOS DE ENTRADA / SALIDA EN UN MICROCONTROLADOR

1.1 Generalidades

Todo microcontrolador posee dentro de su arquitectura una parte fundamental para su correcto funcionamiento, esta es la encargada de comunicar al microcontrolador con el exterior, y se denominan puertos de ENTRADA / SALIDA llamados más comúnmente como puertos I/O, mediante ellos podemos ingresar o transferir información digital o analógica (en el caso de convertidor ADC) hacia o desde cualquier circuitería o periférico externo que deseemos.

Esta es una de las diferencias que existen entre un microcontrolador y un microprocesador, ya que este último no posee dichos puertos y deben ser implementados, ya sea de manera sencilla con flip-flops para situaciones de poca complejidad o con integrados especializados de ser necesarias mayores prestaciones.

Además de funcionar como simple nexo entre el CPU del microcontrolador y las señales externas a manejar, los puertos I/O pueden multiplexarse para desempeñar otras funciones más complejas de entrada/salida como por ejemplo los convertidores Analógico Digital, Manejadores LCD, PWM, Puertos de comunicación Serial, en fin

la cantidad de funciones a desempeñar depende del fabricante y la familia de microcontrolador.

1.2 Estructura Genérica de un puerto I/O (Ref. 1)

Podemos decir que un pin de un puerto I/O está constituido principalmente por lo que se conoce como LACTCH que simplemente es un flip-flop, más generalmente de tipo D el cual se encarga de mantener la señal en caso de que esta sea de salida en uno de los pines que conforma un puerto, que en su mayoría está formado por ocho pines o terminales de puerto, formando así lo que se conoce como registro de puerto, y que según el fabricante se puede tener más de un registro por puerto para manejarlos, además del Latch son fundamentales los buffers de tres estados o triestado, los cuales se encargan de aislar o habilitar las señales de entrada como lectura del pin o el Latch cuando se necesite ser leído.

1.3 Tri – State (Tercer Estado)

En la electrónica digital el sistema utilizado para la comunicación, almacenamiento y transferencia de datos entre dispositivos es el sistema binario que como su nombre lo indica únicamente consta de dos señales, nivel lógico alto o “1” y nivel lógico bajo o “0” que son en realidad niveles medibles de voltaje que se definen de acuerdo al

diseño del sistema al que pertenecen, pero siempre gobernados por la lógica binaria que nos implementar toda esa gran variedad sistemas digitales que existen.

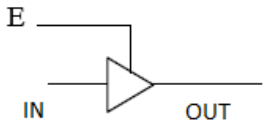
Sin embargo se utiliza también un tercer estado o como se también se lo denomina Tri - State, decimos que un puerto o pin de un puerto se encuentra en Tri-state cuando el mismo no presenta ni estado lógico alto ni estado lógico bajo. Sino que se encuentra en un estado de alta impedancia (HZ). En este estado el puerto o pin no influye sobre ningún dispositivo conectado en él y tampoco sobre él, es decir se encuentra aislado del resto de la circuitería o sistema al que este conectado.

Este estado es utilizado cuando se quiere comunicar varios dispositivos por medio de un bus de datos común, ahorrándose así líneas de comunicación entre ellos ya que todos usaran el mismo bus en determinados tiempos. Un puerto puede ponerse en Tri-State únicamente si anteriormente se configuro como entrada.

1.4 Buffer Tri-estado

Un buffer de tres estados se encargan de aislar el puerto con el bus de datos del sistema cuando el pin se lo utiliza como entrada, esta situación es necesaria ya que el bus de datos del microcontrolador no solamente lo utilizan los puertos de I/O del mismo, sino también la memoria de datos del microcontrolador, de no existir este aislamiento la CPU del microcontrolador no podría saber cuándo está presente un dato de la memoria o del puerto I/O, además de que esta situación podría provocar un

corto circuito interno ya que podrían darse niveles de voltaje diferentes conectados a la misma línea del bus, en la figura 1.1 podemos ver el esquema de un buffer de tres estados típico.



IN	E	OUT
X	0	Z
0	1	0
1	1	1

Figura 1.1. Esquema y tabla de verdad de un buffer triestado.

Como podemos ver cuando Enable representada por E se encuentra en bajo la salida OUT del buffers estará en alta impedancia Z, en esta situación no importa en qué estado se encuentre la entrada IN, mientras que para cuando E se encuentre en alto la salida OUT reflejara lo que se encuentre en la entrada IN como un seguidor normal. También existen puertas con la entrada E inversa, sin embargo esta situación la maneja directamente la CPU del microcontrolador ya que para el programador esto es transparente. En la figura 1.2 vemos el esquema general de un pin de un puerto de I/O.

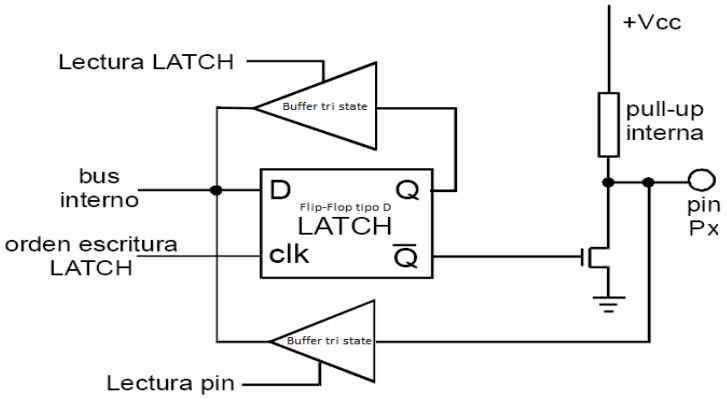


Figura 1.2. Esquema general de un pin en un puerto I/O.

1.5 Funcionamiento del Puerto I/O

El esquema mostrado en la figura 1.2 corresponde a los primeros diseños de puertos de los microcontroladores, en particular corresponde al diseño de un puerto general de la familia 8051 de microcontroladores de Intel, por ser estos los primeros microcontroladores que se diseñaron representan la base en la cual se basaron el resto de microcontroladores posteriores, ya que los demás fabricantes tomaron esta idea básica para sus propios diseños, los cuales al transcurrir el tiempo fueron cambiando, dándole mayores funciones y una arquitectura más compleja, sin embargo, para comprender el esquema de un puerto moderno y más complejo es necesario entender los primeros diseños básicos de un puerto I/O.

En la figura 1.3 veremos el flujo de la información cuando escribimos un “1” o un “0” en el pin correspondiente a un puerto general.

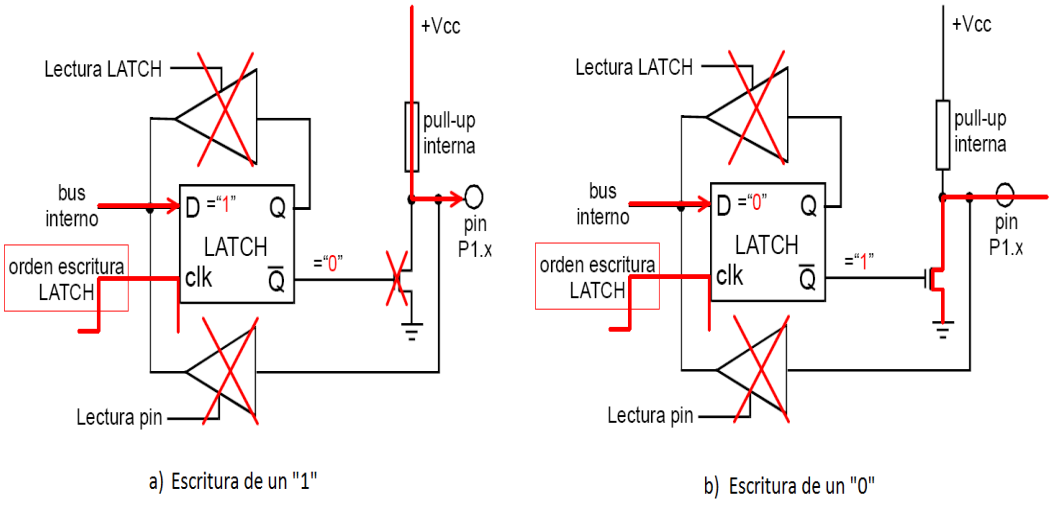


Figura 1.3. Flujo de Información en la escritura de un pin de un puerto I/O

Como podemos ver en la figura 1.3 al momento de escribir información en el pin los buffers de tres estados no intervienen, ya que no vamos a ingresar información al bus de datos sino al contrario, vamos mediante el Latch a colocar un nivel lógico en la salida del pin y por ende en la salida hacia el exterior, esto lo realiza el CPU del microcontrolador colocando una señal en alto en la entrada clk del Latch o flip-flop en el momento en que se ejecute la instrucción correspondiente de escritura en el pin, una vez realizada la escritura del Latch, este permanecerá como fue seteado y el CPU podrá seguir realizado el resto de instrucciones según su programación.

Cuando lo que se desea es leer un dato del pin, ahora si intervendrá el buffer triestado ya que únicamente mediante el buffer se realiza la lectura del nivel lógico que se encuentre en el pin del puerto, la activación del buffer la realiza el CPU del microcontrolador en el momento en que se ejecuta la instrucción de lectura colocando una señal en alto o bajo dependiendo de la lógica del buffers en la entrada Enable del mismo. Sin embargo como vemos en la Figura 1.4 a. antes de realizar la lectura en el pin debemos colocar un “1” en el Latch correspondiente ya que de los contrario podría dañarse el pin si es conectado directamente a +Vcc en la entrada, puesto que no habría un resistencia pull-up entre +Vcc y el transistor manejador del pin, si esto no se hace se debe tener en cuenta que no debemos realizar la conexión sin una resistencia entre el pin y la señal en alto, esto también es válido cuando escribimos un “0” lógico como salida.

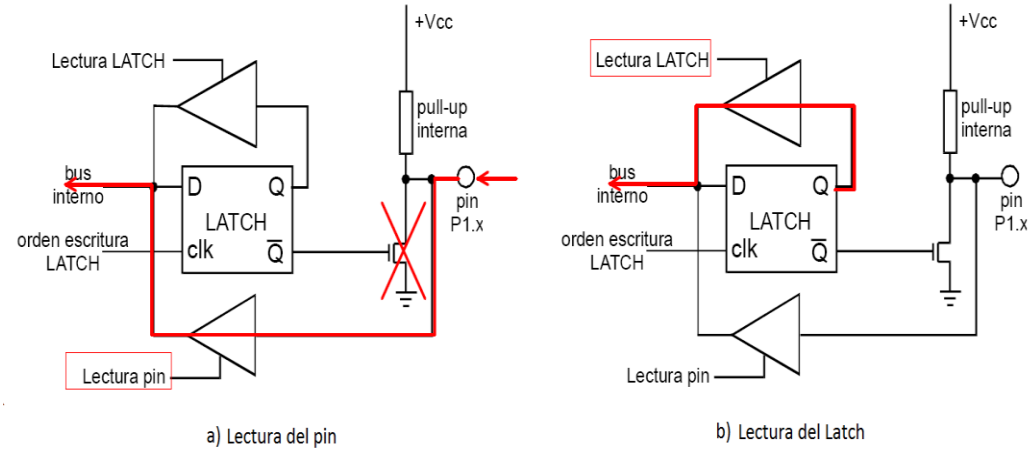


Figura 1.4. Flujo de datos para lectura del pin y del Latch del puerto.

Como podemos ver en la figura 1.4 b. también es posible leer el estado en el que se encuentre el Latch de salida. Más adelante detallaremos la arquitectura de puertos I/O de tres de los fabricantes con mayor posicionamiento en el mercado de microcontroladores como son Atmel, Microchip e Intel esta última por ser la compañía que dio vida a estos dispositivos inteligentes.

1.6 Microcontroladores Intel (Ref. 1)

Intel ha sido una de las empresas pioneras en el desarrollo de microcontroladores compatibles casi en un 100% con la arquitectura de las PC.

A comienzos de los 80 Intel propuso el modelo de microcontrolador denominado 8051 este constituye la segunda generación de microcontroladores de Intel después de la serie MCS-48, todos los modelos basados en el 8051 constituyen la familia MCS-51 DE Intel, Fue tan importante su penetración en el mercado que continúa siendo el núcleo de los microcontroladores actuales. Podrán disponer de mayor memoria de datos o de programas o que se le agreguen más instrucciones o registros, temporizadores con mayores funciones, pero siempre bajo la estructura básica del 8051. Este sirvió de base para muchos otros fabricantes de circuito integrados ya que Intel realizo contratos con otras empresas (Siemens, MHS, OKI, Philips, etc.) para que puedan usar su diseño y hacer mejoras al mismo incorporándoles más periféricos. En la actualidad es un diseño que ha tenido bastantes cambios por parte de otras empresas, y además por parte de la propia Intel que posee microcontroladores más potentes, en definitiva como sabemos Intel se ha inclinado mayoritariamente al diseño y fabricación de microprocesadores, pero sin embargo se sigue utilizando y su uso más común es como driver de teclados básicos. Se ha tomado el modelo de microcontrolador 8051 por ser el microcontrolador más representativo de Intel.

En la figura 1.5 podemos ver la distribución de pines del 8051.

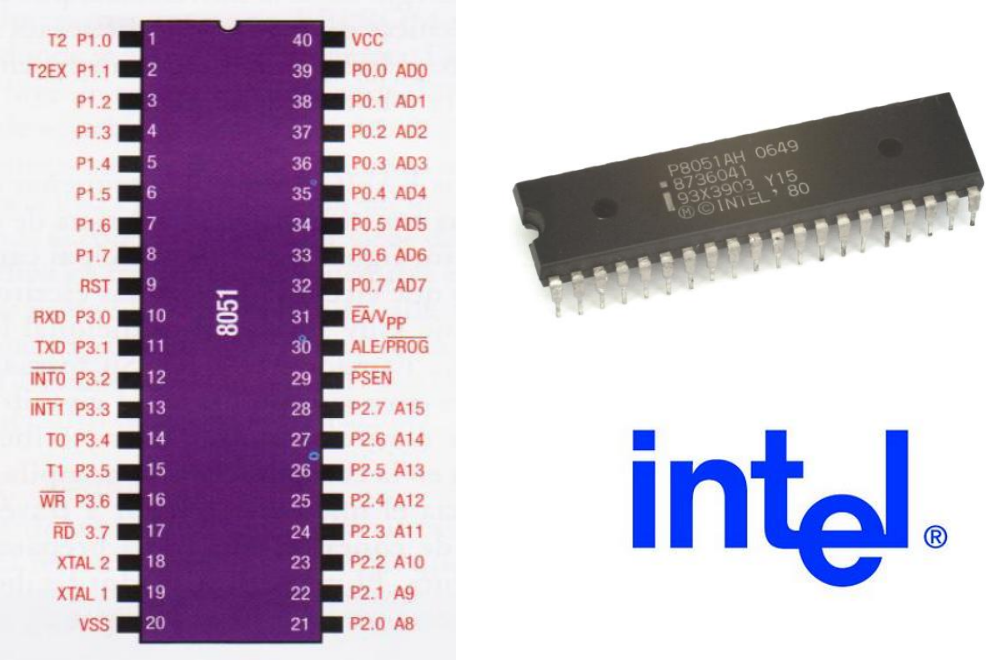


Figura 1.5. Distribución de pines del microcontrolador 8051 de Intel

1.6.1 Puertos del Microcontrolador 8051

El 8051 posee cuatro puertos I/O de 8 bits bidireccionales es decir permiten la lectura y escritura en el periférico correspondiente. Como vimos anteriormente en la Figura 1.2 sus salidas están Latcheadas, lo que posibilita mantener el dato indefinidamente hasta que se sobrescriba la información original. Otra característica importante es que los puertos pueden ser utilizados como buses de direcciones, datos y control, por lo que en estas circunstancias se dice que el microcontrolador trabaja como microprocesador. Estos Latch forman los puertos del microcontrolador los mismos que se encuentran ubicados entre los registros de funciones especiales donde se almacena la información enviada a los mismos mediante programas.

Para el Puerto 0 el nombre de este registro es P0 y cuyos pines de salida son de colector abierto, para el puerto 1 P1, para el puerto 2 P2, y para el puerto 3 P3, por lo que en las instrucciones cuando se quiere referir a uno de estos Puertos para efectuar una operación de escritura o de lectura se debe hacer utilizando como operando en la instrucción el nombre de registro asociado a dicho puerto.

En la figura 1.6 observamos el esquema de los 4 diferentes puertos que existen en 8051. Los mismos tienes diferencias de acuerdo a su función y con estos esquemas podemos darnos cuenta de su funcionamiento.

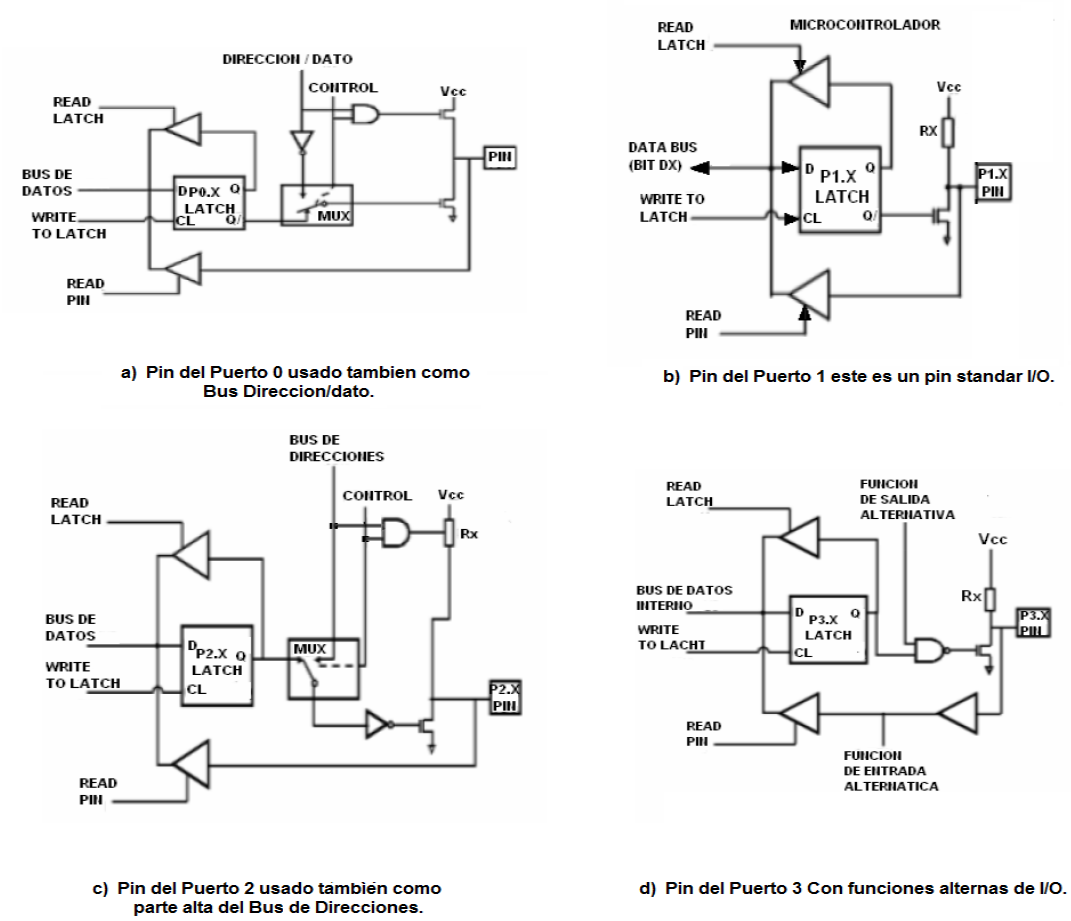


Figura 1.6. Esquema de los 4 Puertos I/O del 8051.

El Puerto P0 multiplexa en el tiempo por sus 8 líneas la parte baja del bus de direcciones durante el acceso a la memoria externa de programa y datos, y el bus de datos. También recibe los bytes de código durante la programación de la memoria EPROM integrada y salen a través del él los bytes de código durante la verificación de la memoria EPROM o ROM. Este puerto puede adoptar el estado de alta impedancia (triestado) por la disposición de los transistores cuando funciona como entrada normal digital (colector abierto) ya que no posee resistencia Pull-up como el resto de puertos en este estado.

En el Puerto P1 los bits P1.0 y P1.1 tiene otra función especial pero sola en el modelo 8052, que son T2 (Timer/Contador 2 Entrada externa) y T2EX (Timer/Contador 2 Captura e impulso de recarga).

El Puerto P2 emite la parte alta del bus de direcciones en los accesos de memoria externa (memoria de programa) cuando utilizan 16 bits de dirección y en los accesos a la memoria de datos que usa también 16 bits de dirección. También recibe la parte alta de la dirección, durante la programación y verificación de la memoria EPROM.

El Puerto P3 tiene las siguientes funciones especiales:

- P3.0 RXD (Entrada puerto serie)
- P3.1 TXD (Salida puerto serie)
- P3.2 INT0 (Interrupción 0 externa)
- P3.3 INT1 (Interrupción 1 externa)
- P3.4 TO (Entrada externa Timer 0)

- P3.5 T1 (Entrada externa Timer 1)
- P3.6 WR (Autorización escritura memoria datos externa)
- P3.7 RD (negado, Autorización lectura memoria datos externa)

ALE/PROG (PROG negado) es un pulso que emite el microcontrolador para indicar a dispositivo de memoria externa que está presente la parte baja del bus de direcciones en su respectivo puerto. PROG es el pin de entrada de pulsos de programación de la memoria EPROM.

PSEN (Negada) es la señal strobe para leer en la memoria de programa externo. La memoria externa tiene 2 modalidades, de programa y de datos. Para diferenciarlas, utiliza la señal PSEN.

EA/VPP cuando se mantiene un nivel alto, se ejecuta solo el programa de EPROM interna, a menos que el contador de programa exceda de FFF (4K) para el 8051. Si se mantiene un nivel bajo, se ejecuta el programa de memoria externa siempre, independiente de la dirección de memoria. Es decir si:

EA=1 actúa como microcontrolador

EA=0 actúa como microprocesador

XTAL 1 y XTAL 2 son la entrada y la salida, respectivamente, de un amplificador inversor que puede ser configurado para su uso como oscilador. Se pude utilizar

indistintamente un cristal de cuarzo o un resonador cerámico. O sino solamente se utilizaría XTAL 2 como entrada de una señal de oscilador externa.

RESET señal de inicialización del sistema. Un reset interno al sistema se produce cuando se pone en pin de RESET a nivel alto durante un cierto tiempo.

1.6.2 Operación de escritura en los puertos del microcontrolador 8051

Para la operación de ESCRITURA en el puerto, la instrucción más habitual es la siguiente:

MOV Px,<dato> ; Px ← dato.

x toma valores 0, 1, 2 y 3 según el puerto.

Admitiendo <dato> todos los tipos de direccionamiento o algún registro

La operación de escritura, utilizando los puertos puede ser realizada por cualquiera de ellos, no obstante, el puerto *PO* es el que presenta una mayor cargabilidad, permitiendo comandar ocho cargas TTL - LS, mientras que los otros tres permiten cuatro cargas TTL – LS.

Para comandar cargas de mayor consumo energético, como relés, se recomienda utilizar, entre el puerto y la carga drivers no inversores.

Si se necesita activar o desactivar un bit de un puerto (se puede hacer extensivo a todos los registros direccionales *bit a bit* del SFR), puede hacerlo utilizando las instrucciones booleanas.

CLR <bit>

SETB <bit>

CPL <bit>

Así por ejemplo, para activar el *bit 2* del *Puerto P0.*

SETB P0.2 o también SETB 82

De manera que los registros que son direccionables bit a bit son:

Dirección de Memoria	Registro
80	**P0**
88	T0
90	**P1**
98	SCON
A0	**P2**
A8	IE
B0	**P3**
B8	IP
C0	
C8	T2CON
D0	PSW
D8	
E0	ACC
E8	
F0	B
F8	

Tabla 1.1. Registros de funciones Especiales manipulables bit a bit.

Obsérvese en la tabla que de un registro a otro existe una división de 8 bytes esto se necesario para que sea posible la identificación para la realización del direccionamiento bit a bit de manera que cuando se refiere a cada uno de los 7 bits que se tienen en un registro se pueda identificar cada uno con una dirección de memoria.

Las instrucciones que son válidas para uno, son válidas para todos. Cuando en las instrucciones vamos a trabajar con los puertos de damos en mismo tratamiento que le damos a los registros de funciones especiales, es más, PO, P1, P2 y P3, constituyen registros ubicados dentro de los registros de funciones especiales. Por lo que todas aquellas instrucciones que pueden ser utilizadas para manipular estos registros pueden ser utilizadas para manipular a los puertos. Por ejemplo si queremos sumar el contenido del registro Acumulador con el valor presente n cada uno de los terminales de Puerto 2, lo haríamos mediante la siguiente instrucción:

```
ADD   A,P2          ; A ← A+P2
```

En esta instrucción lo que se lee es el Latch de salida como lo habíamos visto en los esquemas de los puertos, es decir no tomamos un dato entrante, sino el que se encuentra configurado previamente en el Latch.

1.6.3 Operación de lectura en los puertos del microcontrolador 8051

Para poder efectuar la lectura mediante programa del valor presente en uno de sus terminales o pin es una condición imprescindible que el bit correspondiente en su registro es decir el Latch se haya puesto previamente a “1”. Esta condición por lo general resulta transparente para el programador ya que, con la activación de la señal de reset todos los bits de los registro de los puertos son cargados con el valor de “1”, con lo que quedan preparados para ser trabajados como de entrada. Como en la mayoría de aplicaciones si usted decide que un puerto será de entrada esta condición la va a mantener durante todo el tiempo pues no tendrá que preocuparse por esto ya que con la imprescindible activación de la señal de reset del microcontrolador en su aplicación al iniciar la misma no habrá inconveniente. No obstante, puede haber alguna aplicación en la cual un determinado Puerto pueda ser utilizado una parte del tiempo como entrada y otra parte de tiempo como salida, en este caso cada vez que usted vaya a conmutar el puerto de salida hacia entrada lo primero que usted debe hacer es cargar en todos los bits del registro del puerto un “1”.

La operación de lectura o de adquisición de datos no representa ningún tipo de problema; solamente se deberá cambiar el orden de los operandos en la instrucción respecto a la de escritura.

Para la operación de lectura, el formato de la instrucción más habitual es el siguiente:

MOV <registro>,PX ; registro ← PX

Al ubicar en la instrucción el registro Px en la posición que ocupa la localización fuente estamos indicando que la instrucción es de lectura sobre dicho registro.

Cuando se ejecuta esta instrucción todos los buffers de lectura de dicho puerto se activan para poder leer el valor del dato externo aplicado a cada pin del puerto. estos buffers de lectura interpretaran que en el terminal correspondiente del puerto hay un "0" lógico o un "1" lógico, los rangos de estos voltajes corresponden a los de la familia lógica TTL son:

0 Volts < Low < 0.8 Volts

2.0 Volts < High < 5.5 Volts

Nótese que este microcontrolador siempre tenemos la resistencia Pullup activada, excepto en el puerto P0 ya que este necesita de una resistencia Pullup externa cuando va a funcionar como puerto de salida. Cuando se usan como entrada estas serán entradas en alta impedancia.

1.7 Microcontroladores Microchip (Ref. 2 y 3)

Microchip es una de las mayores empresas fabricantes de microcontroladores, su familia de microcontroladores se denomina PIC (Peripheral Interface Controller)

En general existen diversos modelos de microcontroladores PIC desde los básicos como PIC16F84 que se aprecia en la figura 1.7 de 8 bits con solo dos puertos I/O hasta los modernos y potentes PIC32M de 32 bits con más de cinco Puertos I/O. Los PIC de Microchip son los que más se han comercializado y con los que más se ha trabajo en nuestro país, ya que existe amplia información de aprendizaje de los mismos ya sea por medio de libros que se encuentran en español y amplia información en internet, precio y disponibilidad, posee un reducido número de instrucciones 35 en su gama media, herramientas de desarrollo fáciles y baratas así como también debido a su utilización para decodificador de seguridad de sintonizadores de tv y consolas de videojuegos lo que los ha hecho muy populares y comerciales. Además claro de poseer muchas características que los han ubicado como uno de los mejores productos en este campo.

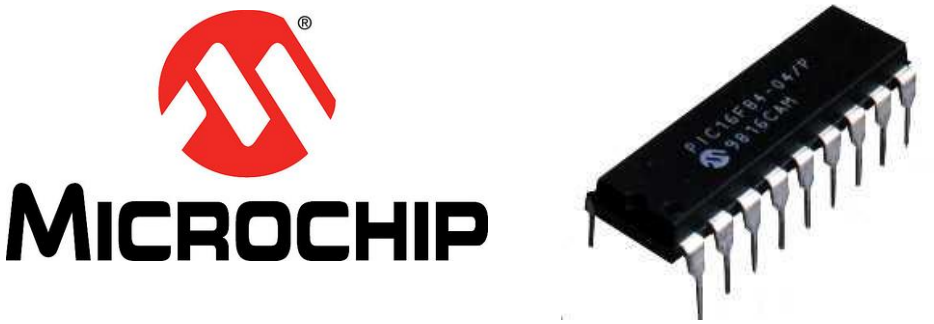


Figura 1.7. Microcontrolador PIC16F84 de MICROCHIP.

1.7.1 Puertos de los Microcontroladores PIC de Microchip

En general, las cápsulas con 18 pines, que se consideran de gama baja como las que contienen al PIC16C62X, PIC16C71 y PIC16C84, y demás, destinan 13 pines a los puertos de E/S, dejando las otras 5 pines para labores generales, como recibir la tensión de alimentación (VDD y TIERRA), conectar el cristal de cuarzo (OSC1 y OSC2) y otra para el Reset/Tensión de Programación (MCLR/VPP). Las cápsulas con 28 pines, como la que contiene le PIC16C73, poseen 22 para los puertos de E/S y, finalmente, las de 40 pines (PIC16C74), 33 se dedican a las líneas de E/S.

Mientras que los microcontroladores encapsulados con 18 pines solo suelen disponer de 2 Puertos de E/S (A y B), los de 40 pines de gama media y alta, llegan a tener 5 puertos de E/S (A, B, C, D y E).

La mayoría de los pines de los Puertos son multifunción, es decir, soportan diferentes funciones según sean programadas. Así, por ejemplo, existen pines que a veces funcionan como líneas de E/S digitales y otras como entradas o salidas de señales analógicas para un comparador.

Por lo general en la mayoría de los modelos de PIC el Puerto A solo está compuesto por 5 o 6 pines es decir es un puerto menor a 8 bits como es característica de un Puerto I/O normal.

El Puerto B es el único que tiene pull-up se las activa cuando el puerto se programa como entrada y el bit<7> (RBPU) del registro OPTION se pone a 0, en los demás puertos deben ser implementadas externamente según sea necesario.

1.7.2 Configuración de los Puertos I/O de Microchip

Los bits de cada puerto se configuran mediante dos registros especiales los cuales son:

1. El registro de datos, al que se denomina PORTx (PortA, PortB, etc. dependiendo del modelo de PIC).

2. El registro de control TRISx, con el que se programa el sentido (Entrada o Salida) de las líneas de cada puerto.

Cada uno de sus bits puede programarse como una línea de Entrada o de Salida, según se ponga un 1 ó un 0 en el bit del registro de control TRISx correspondiente.

Un 1 en el bit “n” del registro TRISx configura el pin del puerto como entrada y pone en alta impedancia el mismo. Si en el bit “n” de TRISx fuera un 0, en pin se configura como salida y el contenido del biestable de datos correspondiente del PORTx pasaría a al pin de I/O externa.

En la figura 1.8 Podemos ver un puerto general de entrada salida de los microcontroladores de Microchip, este es el diseño básico, otros puertos se basan en

este esquema, pero implementando los componentes necesarios para cada función alterna.

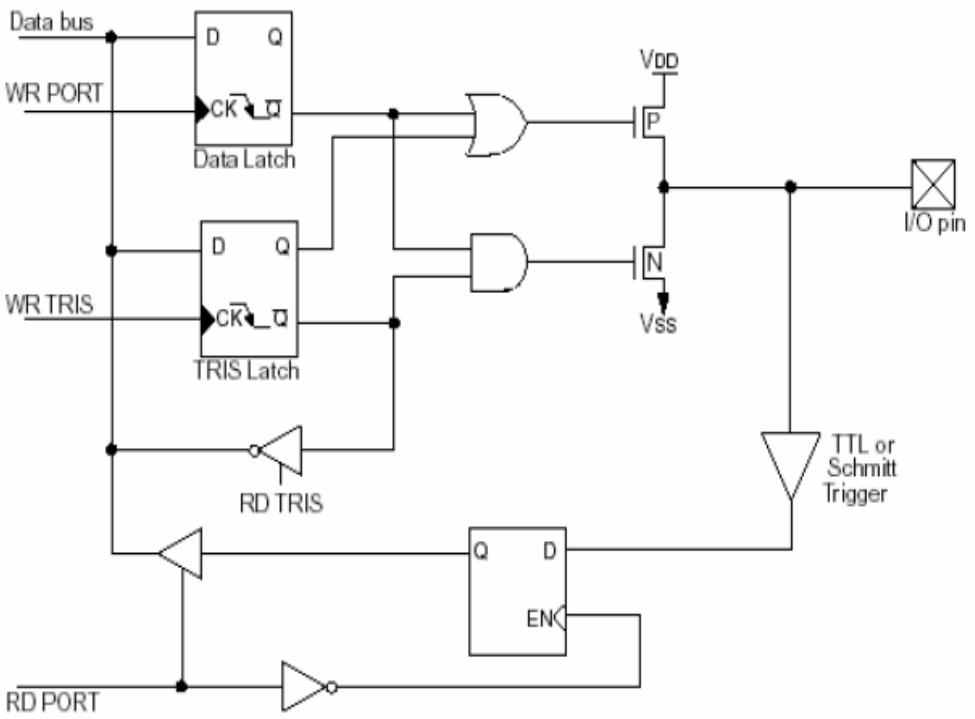


Figura 1.8. Esquema básico de pin estándar de un PIC.

Cuando un pin de I/O funciona como salida, el bit que proviene del bus de datos se guarda en el biestable del dato con lo cual la información que ofrece este pin permanece invariable hasta que se rescriba otro bit.

Cada línea de E/S de los puertos se programa de forma independiente y puede ser Entrada o Salida. Cuando se produce un reset, todos los bits de los registros TRIS pasan a tener el valor 1 y todas las líneas de E/S actúan como Entrada por evidentes motivos de seguridad para evitar daños irreparables. Sin embargo cuando una función

está multiplexada en un pin de I/O de propósito general, la funcionalidad de un pin puede cambiar para acomodarse a los requisitos del módulo periférico. Por ejemplo, si un micro tiene un conversor A/D, los pines asociados a este módulo están configurados en un RESET como entradas analógicas para evitar un consumo de corriente excesivo en el buffer de dicho pin si este estuviera configurado como entrada digital y el valor de tensión en el pin estuviera a un nivel intermedio.

Instrucciones como bsf y bcf comienzan leyendo el valor del puerto y cargándolo en el registro W; allí ejecutan la puesta a 1 ó a 0 del bit seleccionado y, luego, depositan el registro W en el puerto. También hay que tener en cuenta las modificaciones que se produzcan en los pines que son entrada y pasan a salida, pues pueden estar presentes datos antiguos en el registro de salida del puerto al ser memorizados.

Hay que prestar mucha atención a las operaciones que, tras una lectura de un puerto. Sigue una escritura de la misma. Se debe dejar pasar un tiempo determinado para que se estabilice el voltaje de los pines. Insertando entre la lectura y la escritura una instrucción NOP o cualquier otra que no implique a los puertos, se eliminan estos errores potenciales.

Las líneas del Puerto B pueden programarse para disponer de carga “pull up” interna si actúan como entradas y también pueden provocar interrupciones.

1.8 Microcontroladores Atmel (Ref. 4 y 5)

Atmel junto a Microchip es líder en diseño, fabricación y comercialización de circuitos integrados, los microcontroladores Atmel se fabrican utilizando los más avanzados procesos de integración, con tecnologías como BiCMOS, CMOS y germanio de silicio (SiGe). Además la arquitectura AVR de Atmel ha hecho que estos microcontroladores ganen una excelente aceptación, ya que poseen unas características muy destacables en comparación a otros fabricantes, una de ellas es su velocidad ya que de acuerdo a como están organizados sus registros de propósito general junto con el ALU del CPU y el diseño del conjunto de instrucciones estos microcontroladores trabajan a 1 Mips reales a una frecuencia de reloj de 1 MHz.

También por su diseño orientado a programación en lenguaje C, hace que el grado de optimización del código sea alto permitiendo así tener una mayor eficiencia en el tamaño de código en su programación.

Esta empresa maneja tres grandes grupos de microcontroladores RISC (reduced instruction set computer) cuyas CPU llegan hasta los 32 bits. El primer grupo tiene una arquitectura basada en el 8051 de Intel con memoria de programa tipo flash. El segundo grupo son los microcontroladores AT91 basados en núcleos ARM, los cuales soportan compilador en C, emulador, etc. El último grupo lo conforman los microcontroladores AVR, de arquitectura RISC y CPU de 8 bits y módulos USART, SPI, ADC, etc. Esta arquitectura es propia de Atmel basada en arquitectura Harvard.

A continuación detallaremos el funcionamiento y las principales características de los puertos I/O que posee el microcontrolador ATmega169 que se aprecia en la figura 1.9 ya que es el que se utilizará en este trabajo dentro del kit AVR Butterfly, sin embargo es también valido para otros microcontroladores de la familia AVR Atmel.

Figura 1.9. Microcontrolador Atme169.

1.8.1 Puertos I/O de Atmel

Todos los puertos AVR tienen la funcionalidad de Leer-Modificar-Escribir cuando se usan como puertos generales de I/O digitales. Esto permite cambiar la dirección de los datos en un pin del puerto, con las instrucciones SBI (Set Bit I/O Register) y CBI (Clear Bit I/O Register), sin cambiar involuntariamente la dirección de los datos de cualquier otro pin. Lo mismo se aplica al cambiar el valor del controlador (si está configurado como salida) o al habilitar/deshabilitar resistores pull-up (si está configurado como entrada).

Cada buffer de salida tiene características simétricas de controlador, con alta capacidad de sumidero y de fuente. El controlador de pin es bastante poderoso para manejar directamente las pantallas de LED. Todos los pines del puerto tienen resistores pull-up de selección individual con resistencia invariante con la fuente de

voltaje. Todos los pines de I/O tienen diodos de protección para V_{CC} y Tierra como indica la Figura 1.10.

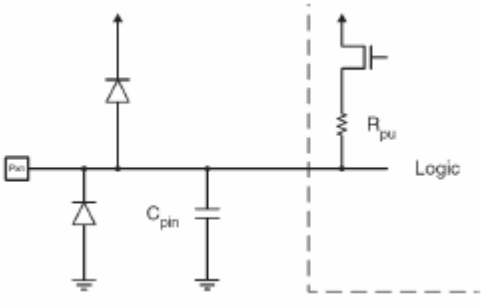


Figura 1.10. Diodos de protección para cada pin

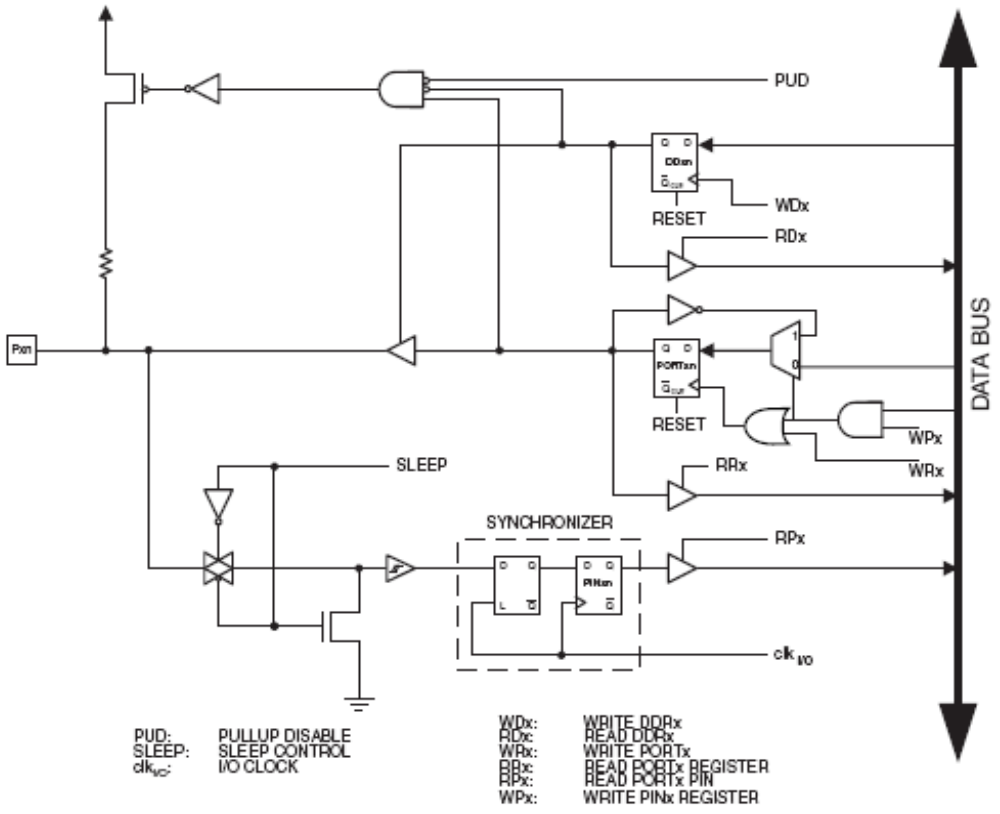
A cada puerto se le asignan tres localidades de dirección de memoria de I/O, una a cada uno de los siguientes registros: Registros de Datos – PORTx, el Registro de Dirección de Datos – DDRx, y el Registro de los Pines de Entrada del Puerto – PINx. El Registro de los Pines de Entrada del Puerto es únicamente de lectura, mientras que el Registro de Datos y el Registro de Dirección de Datos son de lectura/escritura. Sin embargo, escribiendo un uno lógico en un bit en el Registro PINx, resultará en una conmutación en el bit correspondiente en el Registro de Datos. Además, el bit PUD (Deshabilitar Pull-up) del MCUCR deshabilita la función de pull-up de todos los pines en todos los puertos cuando está seteado.

La mayoría de los pines del puerto son multiplexados con funciones alternas para las características de los periféricos en el dispositivo.

Nótese que habilitar las funciones alternas de algunos de los pines de los puertos no afectará al uso de los otros pines como I/O digitales en el puerto.

1.8.2 Los Puertos como I/O Digital General

Todos los puertos del microcontrolador ATmega169 son puertos de I/O bi-direccionales con pull-ups internas opcionales. La Figura 1.11 muestra una descripción funcional de uno de los pines del puerto de I/O, generalmente llamado Pxn (x puerto, n pin).



Nota: WRx, WPx, WDx, RRx, RPx, y RDx son comunes para todos los pines del mismo puerto. clk, SLEEP y PDU son comunes para todos los puertos

Figura 1.11. Descripción funcional de un pin.

Configurar el pin. Cada pin del puerto consiste de tres bits de tres registros: DDxn, PORTxn y PINxn.

El bit DDxn en el Registro DDRx selecciona la dirección de los datos de este pin. Si DDxn es escrito con uno lógico, Pxn se configura como un pin de salida. Si DDxn es escrito con cero lógico, Pxn se configura como un pin de entrada. Ver Tabla 1.1.

DDxn	PORTxn	PUD (in MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Tabla 1.2. Configuraciones posibles para un pin.

Si PORTxn es escrito con uno lógico cuando el pin está configurado como una entrada, se activa la resistencia pull-up correspondiente. Para desactivar la resistencia pull-up, PORTxn tiene que ser escrito con cero lógico o el pin tiene que ser configurado como pin de salida.

Los pines del puerto están en alta impedancia cuando la condición de reset se activa, incluso si ningún reloj está funcionando.

Si PORTxn es escrito con uno lógico cuando el pin está configurado como un pin de salida, el pin del puerto es conducido a alto (uno). Si PORTxn se escribe con cero lógico cuando el pin está configurado como un pin de salida, el pin del puerto es conducido a bajo (cero).

Alternar el Pin. Escribir un uno lógico en el PINxn alterna el valor del PORTxn, independientemente del valor de DDRxn.

Conmutar entre Entrada y Salida. Al conmutar entre el estado de alta impedancia ({DDxn, PORTxn}=0b00) y salida en alto ({DDxn, PORTxn}=0b11), un estado intermedio con cualquier pull-up habilitada ({DDxn, PORTxn}=0b01) o salida en bajo ({DDxn, PORTxn}=0b10) debe ocurrir. Normalmente, el estado de pull-up habilitada es completamente aceptable, puesto que un entorno de alta impedancia no notará la diferencia entre un controlador de nivel alto pronunciado y una pull-up. Si este no es el caso, el bit PUD en el Registro MCUCR puede setearse para deshabilitar todas las pull-ups en todos los puertos.

Conmutar entre entrada con pull-up y salida en bajo generará el mismo problema. El usuario debe usar o el estado de alta impedancia ({DDxn, PORTxn}=0b00) ó el estado de salidas en alto ({DDxn, PORTxn}=0b11), como un paso intermedio.

Leer el Valor del Pin. Independientemente de la configuración del bit de Dirección de Datos DDxn, el pin del puerto puede leerse a través del bit de Registro PINxn. Como se muestra en la Figura 1.11, el bit de Registro PINxn y el latch anterior constituyen un sincronizador. La Figura 1.12 muestra un diagrama de tiempo de la sincronización al leer un valor externo aplicado al pin. Los retraso de propagación máximo y mínimo se denotan como tpd,max y tpd,min respectivamente.

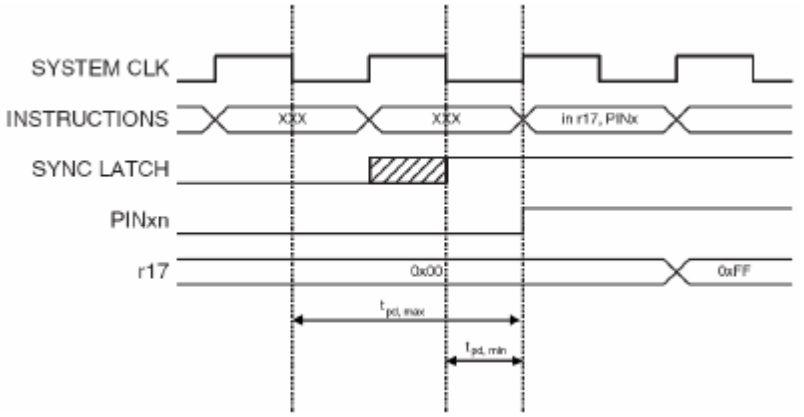
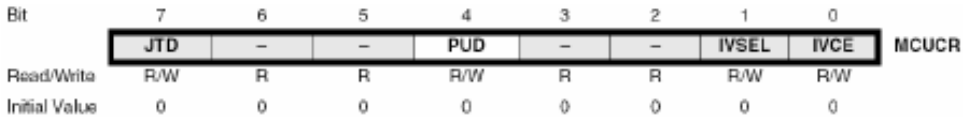


Figura 1.12. Sincronización al leer un valor de pin aplicado externamente.

Los Pines No Conectados. Si algunos pines no han sido utilizados, se recomienda asegurarse de que estos pines tienen un nivel definido. Aunque la mayoría de las entradas digitales están desactivadas en los modos de sueño profundo, entradas flotantes se debe evitar para reducir el consumo de corriente en todos los otros modos en que las entradas digitales están activadas (Reset, el modo activo y el modo de espera). El método más simple para garantizar un nivel definido en un pin no usado es habilitar la pull-up interna. En este caso, la pull-up se deshabilitará durante el reset. Si el bajo consumo de energía es importante durante el reset, es recomendable usar una pull-up o una pull-down externa. Conectar los pines no usados directamente a VCC o a GND no es recomendable, puesto que esto puede causar corrientes excesivas si el pin es configurado accidentalmente como una salida.

1.8.3 El registro de control de la MCU - MCUCR



Bit	7	6	5	4	3	2	1	0	
	JTD	–	–	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R/W	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

El Bit 4 – PUD: Deshabilitador de las Pull-up. Cuando este bit está escrito a uno, las pull-ups en los puertos de I/O están deshabilitadas incluso si los Registros DDxn y PORTxn están configurados para habilitar las pull-ups ({DDxn, PORTxn}=0b01).

1.8.4 Ejemplos de Programación de Puertos I/O

Código de ejemplo en ASM, para configuración de puertos de Atmel como salida y como entrada.

```
; Define pull-ups and set outputs high
; Define directions for port pins
ldi r16,(1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0)
ldi r17,(1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0)
out PORTB,r16
out DDRB,r17
; Insert nop for synchronization
nop
; Read port pins
in r16,PINB
```

El siguiente código de ejemplo en lenguaje C realiza la misma configuración de puertos que el ejemplo anterior.

```
unsigned char i;
...
/* Define pull-ups and set outputs high */
/* Define directions for port pins */
PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0);
DDRB = (1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0);
/* Insert nop for synchronization*/
__no_operation();
/* Read port pins */
i = PINB;
...
```

La nomenclatura (1<<PB7) es igual a desplazar 1 hacia la izquierda 7 veces, por ejemplo: 00000001 << 7 = 10000000.

Y además el símbolo "|" entre ellos significa la operación OR por lo que la instrucción:

PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0);

Es igual a:

(10000000) OR (01000000) OR (00000010) OR (00000001) = 11000011

Por lo que:

PORTB = 11000011

CAPITULO 2

HERRAMIENTAS DE HARDWARE Y SOFTWARE

El presente proyecto se trabajó con el kit de desarrollo, entrenamiento y aprendizaje de Atmel "AVR Butterfly" el cual provee de una excelente conjunto de herramientas y facilidades para que el estudiante aprenda y el evalúe el potencial que tienen los microcontroladores de Atmel.

Figura 2.1. Kit AVR Butterfly

En el presente capitulo describiremos el kit "AVR Butterfly", el software de programación y depuración "AVR Studio 4" que provee Atmel de manera gratuita en su página web y que es un IDE con excelentes prestaciones, ambiente y herramientas de trabajo para desarrollo de software tanto en leguaje ASM como el Lenguaje C y el software ISIS Profesional o Proteus como se lo conoce más comúnmente, el cual es un simular de circuitos eléctricos e integrados muy potente que nos permite visualizar y evaluar el correcto funcionamiento de nuestros diseños.

2.1 Descripción del AVR Butterfly (Ref. 6)

El Kit AVR Butterfly se diseñó para demostrar los beneficios y las características importantes que tienen los microcontroladores de Atmel. Además de poder utilizarlos como una herramienta solida de aprendizaje y evaluación de proyectos tanto para estudiantes como personas particulares que quieran entrar en el mundo de los microcontroladores. Ya que posee un buen numero de periféricos con los que podemos realizar una gran variedad de ejercicios.

Este Kit utiliza el microcontrolador ATmega169, que es un potente microcontrolador de 8 bits el cual posee las siguientes características:

- Arquitectura RISC avanzada.
- Conjunto de 130 instrucciones ejecutables en un solo ciclo de reloj.
- 32 x 8 registros de trabajo de propósito general.
- Rendimiento de hasta 16 MIPS reales a 16 MHz.
- Memoria no volátil para programa de altas prestaciones.
- Flash de 16 Kbyte con 10000 ciclos escritura-borrado.
- Sección opcional para código de arranque con Lock Bits independientes. Programación en el sistema a través del programa de arranque residente en el chip.
- EEPROM de 512 bytes con 100000 ciclos escritura-borrado.
- SRAM interna de 1 Kbyte.
- Bloque de programación para seguridad del software.

- Interfaz JTAG (conforme el Estándar IEEE 1149.1)
- Capacidad de Boundary - Scan conforme al estándar JTAG.
- Soporta depuración On - Chip.
- Programación de la Flash, EEPROM, fusibles y Lock bits a través de la interfaz JTAG.
- Características de los periféricos.
- 6 puertos de I/O de 8 bits y uno de 5 bits.
- Controlador de LCD de 4 x 25 segmentos.
- Dos Timers/Counter de 8 bits con pre ajustador separado, modo de comparación y modo de captura.
- Un Timer/Counter de 16 bits con pre ajustador, modo de comparación y modo de captura.
- Contador de tiempo real con oscilador separado.
- Cuatro canales PWM.
- Ocho canales ADC de 8 bits cada uno.
- Interfaz serial USART programable.
- Interfaz serial SPI maestro/esclavo.
- Interfaz serial universal con detector de condición de inicio.
- WatchDog Timer programable con oscilador separado incluido.
- Comparador analógico.
- Interrupción y salida del modo Sleep por cambio de pin.
- Características especiales de microcontrolador.

- Reset al encendido y detección de Brown-Out programable.
- Oscilador interno calibrable.
- Fuentes de interrupción internas y externas.
- Cinco modos de Sleep: Idle, ADC Noise Reduction, Power Save, Power-Down y Standby.

• Entradas/Salidas y tipo encapsulado.

- 53 lineas de I/O programables.
- 64 patillas en el encapsulado TQFP y 64 pads (para montaje superficial) en el encapsulado QFN/MLF.

• Rangos de velocidad.

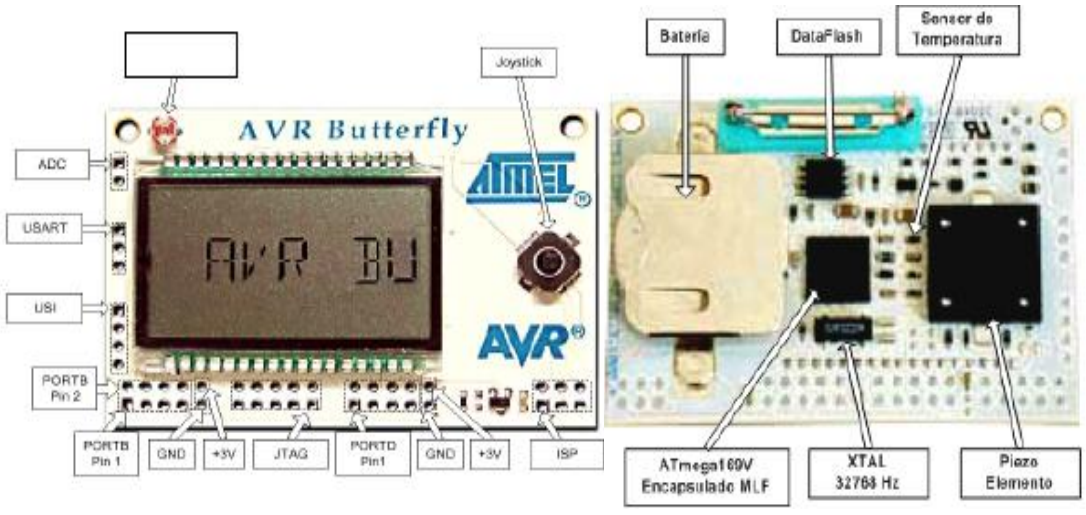
- ATmega169V: 0 - 4 MHz a 1.8 - 5.5 V, 0 - 8 MHz a 2.7 - 5.5 V.
- ATmega169: 0 - 8 MHz a 2.7 - 5.5 V, 0 - 16 MHz a 4.5 - 5.5 V.

Los siguientes recursos están disponibles en el Kit AVR Butterfly:

- Microcontrolador ATmega169v (encapsulado tipo MLF).
- Pantalla tipo vidrio LCD de 120 segmentos, para demostrar las capacidades del controlador de LCD incluido dentro del ATmega169.
- Joystick de cinco direcciones, incluida presión en el centro.
- Altavoz piezoeléctrico, para reproducir sonidos.
- Cristal de 32 KHz para el RTC.
- Memoria data flash de 4 Mbit, para almacenamiento de datos.

- Convertidor de nivel RS-232 e interfaz USART, para comunicarse con unidades fuera del Kit sin necesitar hardware extra.
- Termistor de coeficiente de temperatura negativo (NTC), para medición de temperatura.
- Acceso externo al canal 1 del ADC del ATmega169, para lectura de voltaje en el rango de 0 a 5 V.
- Interfaz USI, para comunicación.
- Terminales externas para conector tipo headers.
- Batería de 3 V tipo botón (600 mAh), para el funcionamiento del Kit.
- Bootloader, para programación mediante la PC sin hardware especial.
- Aplicación demostrativa preprogramada.
- Compatibilidad con el entorno de desarrollo AVR Studio 4.

En la figura 2.2. Se observa el hardware disponible.



a) Parte frontal **b) Parte Posterior**

Figura 2.2. Partes del Kit AVR Butterfly

2.1.1 Firmware Incluido

El AVR Butterfly viene con una aplicación preprogramada, la misma que tiene como interfaz de usuario el joystick del kit. Esta aplicación sirve como un ejemplo, para visualizar las prestaciones del kit.

Las siguientes opciones vienen en la programación estándar.

- Código cargador de arranque (Bootloader Code).
- Código de la Aplicación.
- Máquina de Estados.
- Funciones incluidas:
  - Nombre – Etiqueta.
  - Reloj (Fecha).
  - Medición de temperatura.
  - Medición de voltaje.
  - Reproducción de sonidos.
  - Ahorro de energía.
  - Ajuste de contraste del LCD.

La aplicación puede ser actualizada sin necesitar hardware externo o adicional, ya que el kit posee una circuitería con un convertidor de nivel RS-232.

2.1.2 Joystick

El joystick representa uno de los periféricos que nos sirven como interfaz de usuario en cualquier aplicación que desarrollemos a no ser que implementemos un teclado externo o algún otro tipo de interfaz, pero es la interfaz por defecto para el firmware incluido en el Butterfly.

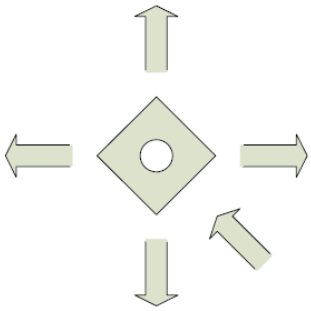


Figura. 2.3. Representación de las entradas del joystick.

Con este periférico podemos movernos a través del menú que tiene el firmware.

Para moverse entre las alternativas del menú, se debe presionar el joystick hacia ARRIBA o hacia ABAJO. Para entrar en un submenú, se debe presionar el joystick hacia la DERECHA. Para salir de un submenú, se debe presionar hacia la IZQUIERDA. Para ingresar o ajustar un valor, debemos presionar la tecla central ENTER.

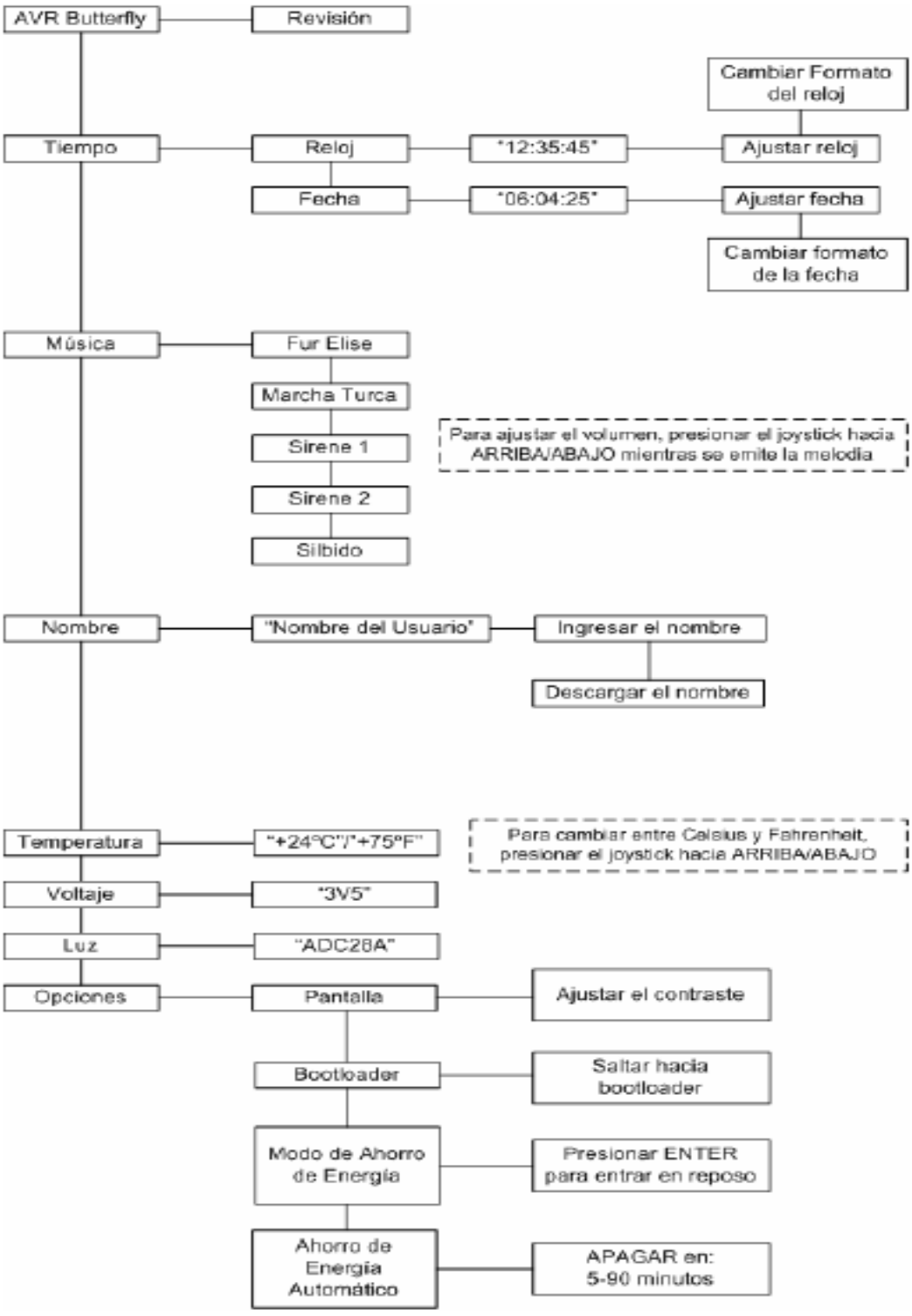


Figura. 2.4. Representación del menú del firmware incluido en el AVR Butterfly.

2.2 Actualización del AVR Butterfly

El AVR Butterfly viene con un Bootloader que usa la característica Self-programming del microcontrolador ATmega169. El bootloader combinado con el circuito convertidor de nivel RS-232, integrado en el AVR Butterfly, hace posible actualizar la aplicación sin ningún hardware externo adicional.

El AVR Prog, es una herramienta incluida en el AVR Studio 4, es usado como interfaz entre la PC y el usuario, como se muestra en figura 2.5.

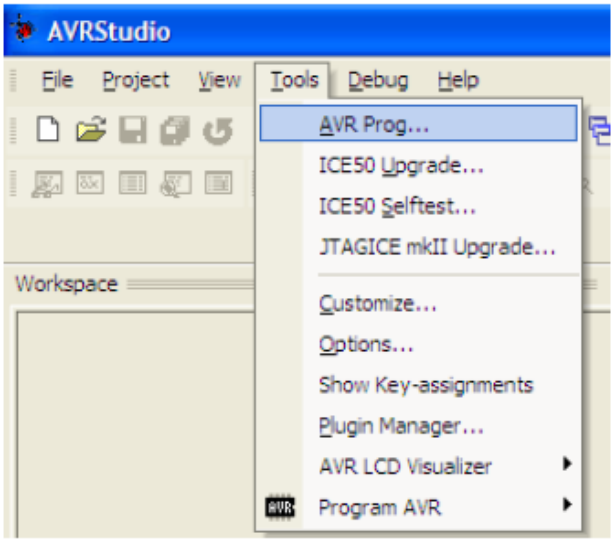


Figura 2.5. AVR Prog en el menú herramientas del AVR Studio 4.

Los datos son transmitidos hacia el microcontrolador a través de la interfaz serial RS-232, para lo cual se debe conectar un cable serial desde la PC hacia el Kit como se describirá más adelante.

Desde la aplicación del AVR Butterfly se puede saltar hacia la sección de Arranque (boot section), desplazándose a través del menú: “Opciones>Bootloader>Saltar hacia Bootloader”, como se ve en la Figura 2.4; ó simplemente reseteando el microcontrolador ATmega169 al cortocircuitar los pines 5 y 6 en el conector J403, conector ISP. Ver Figura 2.6 y Figura 2.2.

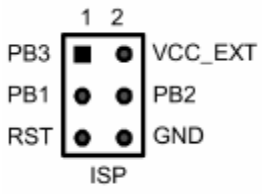


Figura 2.6. Conector para ISP

Luego de un reset, el microcontrolador ATmega169 comenzará desde la sección de Arranque. Nada se desplegará en el LCD mientras esté en la sección de Arranque. Entonces se deberá presionar ENTRAR en el joystick y mantener esa posición; mientras tanto desde la PC en el AVR Studio, iniciar el AVR Prog.

Una vez que se haya iniciado el AVR Prog, soltar el joystick del AVR Butterfly. Desde el AVR Prog, como se ve en la Figura 2.7, utilizar el botón “Browse” para buscar el archivo con la extensión *.hex con el que desea actualizar al AVR Butterfly. Una vez localizado el archivo *.hex, presionar el botón “Program”. Se notará que “Erasing Device”, “Programming” y “Verifying” se ponen en “OK”, de manera automática. Luego de actualizar la aplicación, presionar el botón “Exit” en el AVR Prog para salir del modo de programación del ATmega169.

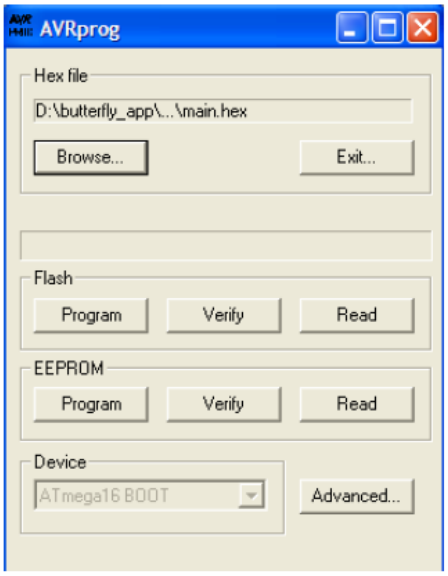


Figura 2.7. AVR Prog.

Para saltar de la Sección de Arranque hacia la de Aplicación presione el joystick hacia ARRIBA.

2.2.1 Fuses y Lock bits

Para que el firmware funcione correctamente, los siguientes Fuses y Lock bits del microcontrolador ATmega169 son los únicos que se debe programar.

Extended Fuse Byte (0xFF)

− Nada

Fuse High Byte (0x98)

− JTAGEN (Interfaz JTAG Activada)

− SPIEN (Programación Serial Activada)

− BOOTSZ1 (Tamaño del Boot 1024 palabras)

− BOOTSZ0

− BOOTRST (Vector Boot Reset Activado)

Fuse Low Byte (0xE2)

− SUT0 (Tiempo de inicio 65 ms)

− CKSEL3 (Oscilador RC Interno)

− CKSEL2

− CKSEL0

Lock Bit Byte (0xEF)

− BLB11 (SPM no está autorizado para escribir a la sección del Boot Loader)

Aplicado al AVR Butterfly, para todos los Fuses y Lock bits, “1” significa sin programar y “0” significa programado.

2.2.2 Programación mediante conexión serial con la PC

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

2.2.3 Distribución de pines para la comunicación serial entre el AVR y la PC

La comunicación con la PC requiere de tres líneas: TXD, RXD y GND. TXD es la línea para transmitir datos desde la PC hacia el AVR Butterfly, RXD es la línea para recepción de datos enviados desde el AVR Butterfly hacia la PC y GND es la tierra común. En la Tabla 2.1 se observa la distribución de los pines para la comunicación serial, a la izquierda los pines del AVR Butterfly y a la derecha los pines del conector DB9 de la PC.

Tabla 2.1. Distribución de pines entre Butterfly y la PC

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

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

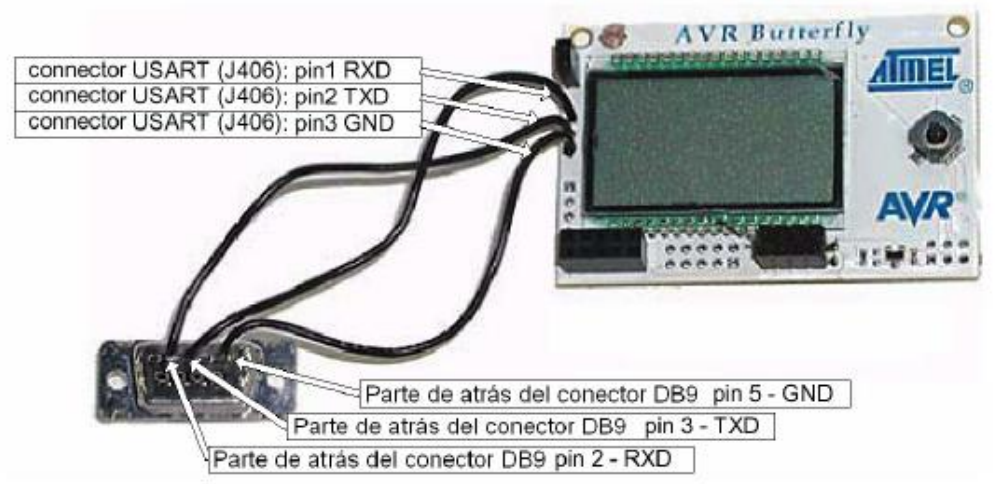


Figura 2.8. Conexiones de interfaz USART del AVR Butterfly

2.2.4 Programación del AVR Butterfly

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

- En la PC, localizar y ejecutar el AVR Studio.

- En el menú “File” del AVR Studio seleccionar “Open File“, luego seleccionar el archivo con el que se desea programar al AVR Butterfly, tal como se puede observar en la Figura 2.9.

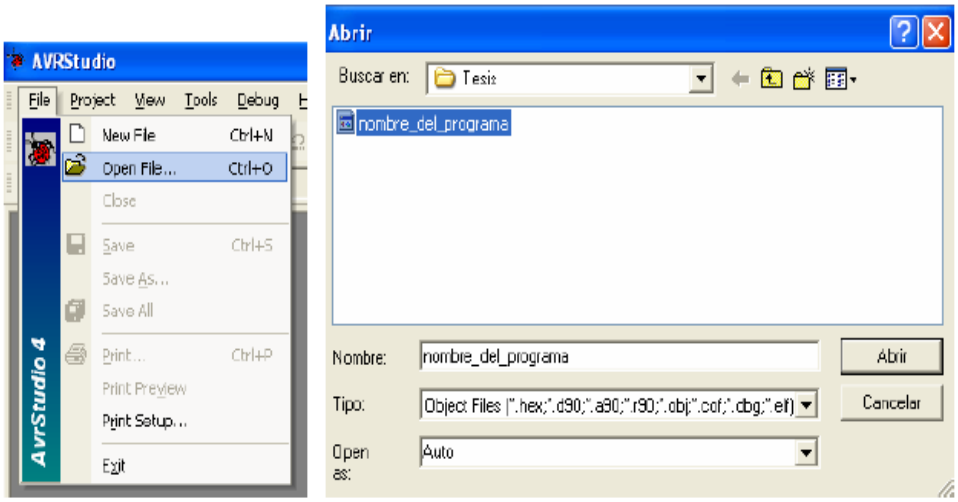


Figura 2.9. AVR Studio, selección del archivo COF para depuración.

- Seleccionar el AVR Simulator y luego el ATmega169 como en la Figura 2.10.

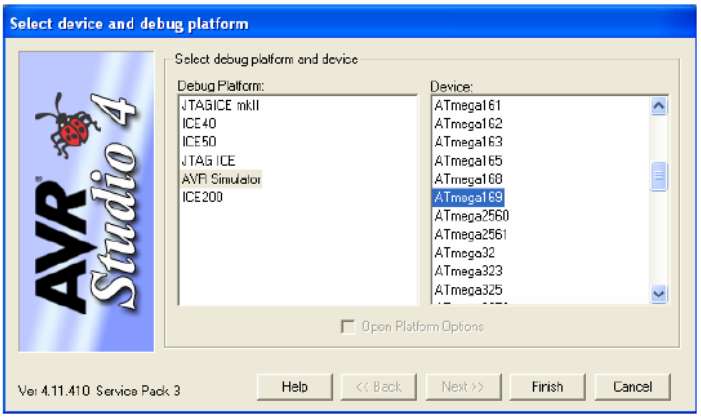


Figura 2.10. Selección del AVR Simulator y dispositivo ATmega169.

- Presionar “Finish”.

- Conectar el cable serial entre la PC y el AVR Butterfly como se puede observar en la Figura 2.8.

- Resetear el AVR Butterfly cortocircuitando los pines 5 y 6 en el conector J403, conector ISP, o quitando y aplicando nuevamente la fuente de alimentación. Ver Figura 2.6 y Figura 2.2.

- Luego de un reset, el microcontrolador ATmega169 comenzará desde la sección de Arranque. Nada se desplegará en el LCD mientras esté en la sección de Arranque. Entonces se deberá presionar ENTRAR en el joystick y mantener esa posición; mientras tanto desde la PC en el AVR Studio, iniciar el AVR Prog.

- Una vez que se haya iniciado el AVR Prog, soltar el joystick del AVR Butterfly. Desde el AVR Prog, como se ve en la Figura 2.7, utilizar el botón "Browse" para buscar el archivo con la extensión *.hex con el que desea actualizar al AVR Butterfly. Una vez localizado el archivo *.hex, presionar el botón "Program". Se notará que "Erasing Device", "Programming" y "Verifying" se ponen en "OK", de manera automática. Luego de actualizar la aplicación, presionar el botón "Exit" en el AVR Prog para salir del modo de programación del ATmega169.

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

2.3 AVR Studio 4

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

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

• La definición del producto. El producto que debe crearse se define basándose en el conocimiento de la tarea que se quiere resolver y la entrada que tendrá en el mercado.

• La especificación formal. Se define una especificación formal para el producto.

• Asignación de la tarea a un equipo. A un equipo del proyecto, que consiste de una o mas personas, se le asigna la tarea de crear el producto basándose en la especificación formal.

• El equipo del proyecto pasa por la secuencia normal de diseño, desarrollo, depuración, comprobación, planificación de producción, producción, prueba y embarque.

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

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

• Windows 2000/XP o Posterior

• Hardware Recomendado:

- Procesador Intel Pentium de 200MHz o equivalente.

- Resolución de Pantalla de 1024x768 (Resolución mínima de 800x600).

- Memoria RAM de 64Mb.

- Disco Duro con 50 Mb de espacio disponible.

2.3.1 Descripción General del IDE

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

AVR Studio 4 proporciona herramientas para la administración de proyectos, edición de archivo fuente, simulación del chip e interfaz para emulación In-circuit para la poderosa familia RISC de microcontroladores AVR de 8 bits.

AVR Studio 4 consiste de muchas ventanas y sub-módulos. Cada ventana apoya a las partes del trabajo que se intenta emprender. En la Figura 2.11 se puede apreciar las ventanas principales del IDE.

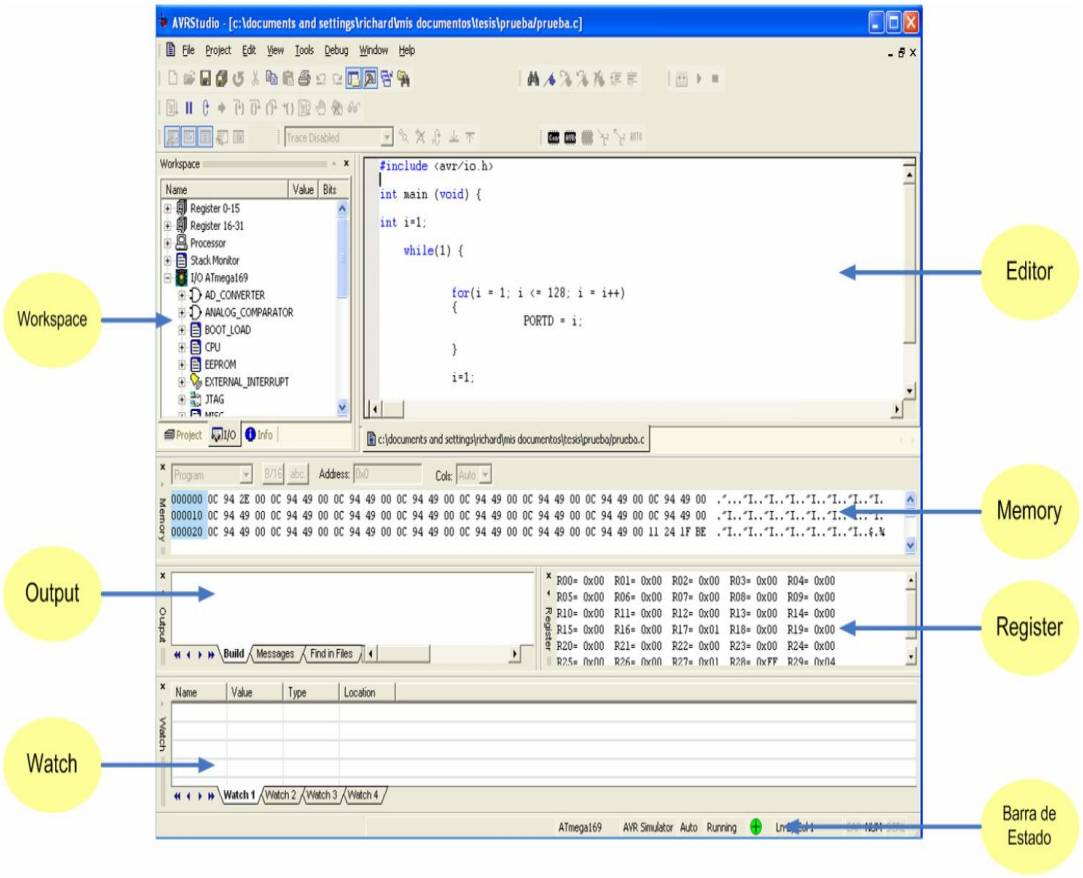


Figura 2.11. Entorno de desarrollo integrado AVR Studio 4.

Administración de Proyectos. Toda creación de código dentro del AVR Studio se la realiza como proyectos de programación. Todos los proyectos de código tienen un archivo project que mantiene la información acerca del proyecto, de qué archivos consta, la estructuración del ensamblador, vistas personalizadas y así sucesivamente. El archivo project asegura que el proyecto sea el mismo cada vez que regrese a él, que todo esté adecuadamente organizado y que ensamble / compile.

2.3.2 Las Ventanas principales del AVR Studio 4

La Ventana Editor

Este es el lugar donde se realiza el trabajo de codificar. El AVR Studio usa el editor Stringray de la corporación Bsquare. Este es un editor completo para programación con toda la funcionalidad que el usuario empleará, incluso codificación de color para la sintaxis (que puede ser alterada y extendida por el usuario). La ventana del editor también se usa cuando al depurar código. En la Figura 2.12 se aprecia la ventana del Editor.

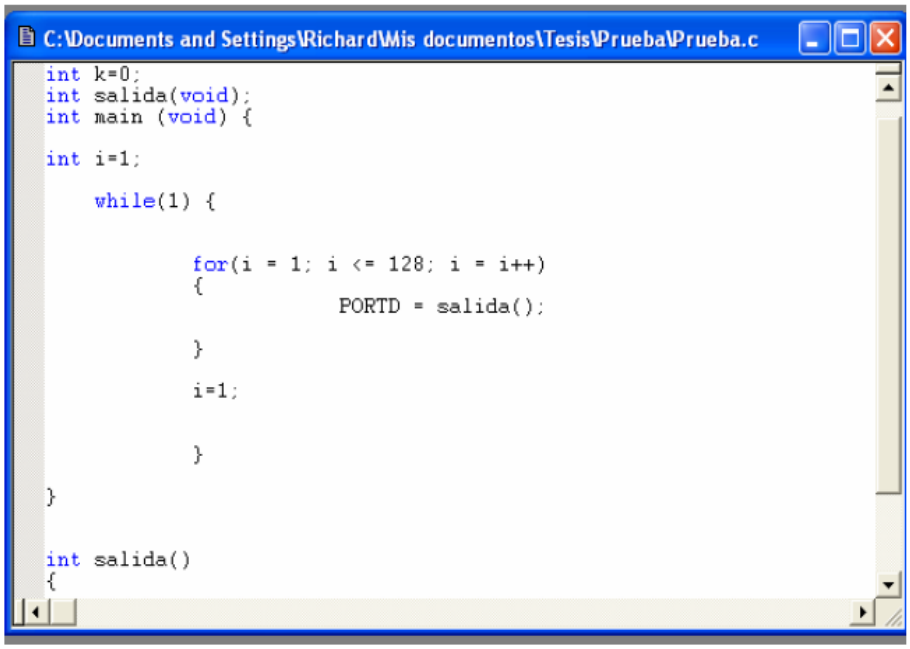


Figura 2.12. Ventana del Editor.

La Ventana Output (Ventana de Resultados)

Es una colección de varias ventanas (Build, Messages,…) integradas en un marco etiquetado. Con las etiquetas sobre el marco se selecciona la ventana que se desea observar. En la Figura 2.13 se observa la ventana de Resultados (Output) y en la misma se aprecia las distintas etiquetas correspondientes a las ventanas que la conforman.

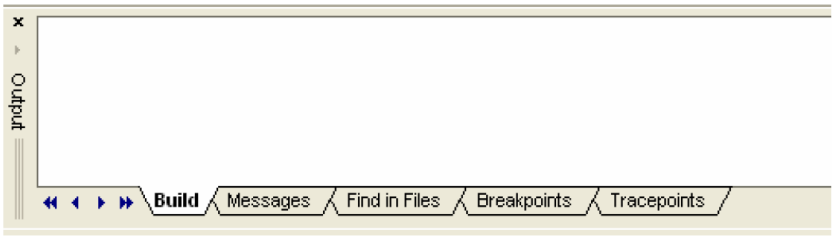


Figura 2.13. Ventana Output.

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

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

Messages. AVR Studio está hecho con muchos objetos SW encapsulados con tecnología Microsoft DCOM. Muchos de los objetos del software no tienen su propia interfaz de usuario. La ventaja Messages es la ventana común que todos los módulos de software emplean para presentar mensajes al usuario. Los mensajes están codificados por color. La mayoría de mensajes son mensajes simples sin prioridad significante que no tienen color. Las advertencias que se presentan en amarillo

señalan problemas potenciales. Los errores están en rojo. Todos los mensajes pueden mostrarse acompañados del tiempo, haciendo clic derecho con mouse sobre la ventana y seleccionando la opción mostrar el tiempo (time stamp option).

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

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

La Ventana Workspace (Área de Trabajo)

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

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

La ventana Project

Si se ha creado un proyecto de código, la ventana Project lista los archivos que están incluidos en dicho proyecto. Si se tiene abierto un archivo objeto para depuración, la ventana Project muestra el nombre del archivo objeto actualmente cargado así como también los archivos fuente a los cuales el archivo objeto se refiere. En la Figura 2.14 se puede apreciar la vista Project.

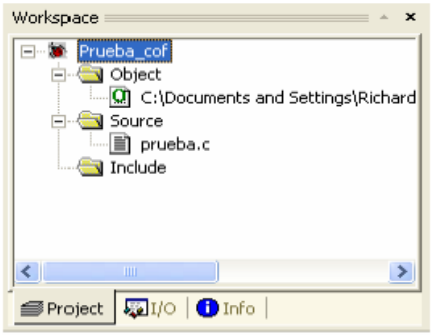


Figura 2.14. Ventana Project.

La ventana I/O

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

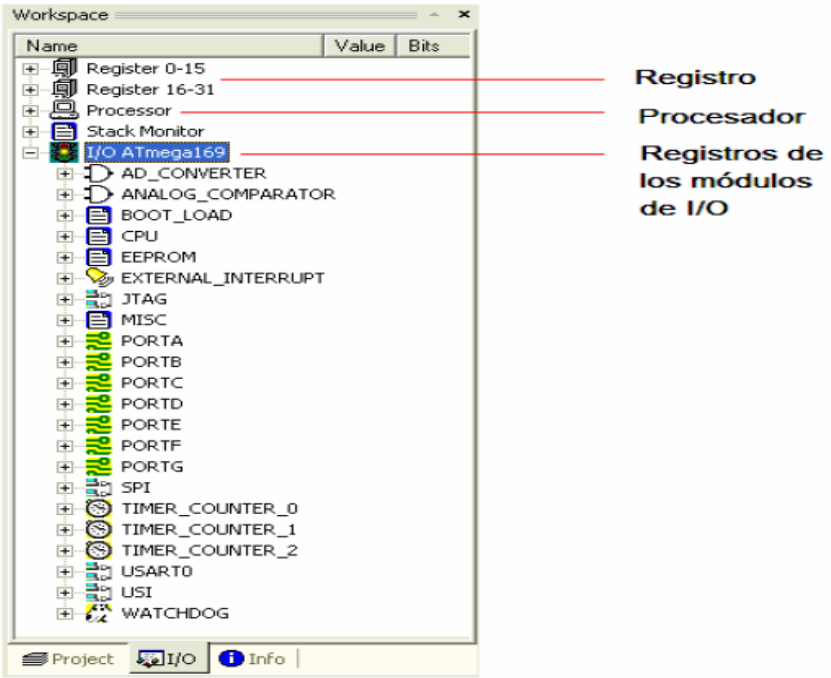


Figura 2.15. Ventana I/O.

Como se aprecia en la Figura 2.15, la ventana I/O permite visualizar los siguientes campos:

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

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

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

La ventana Info

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

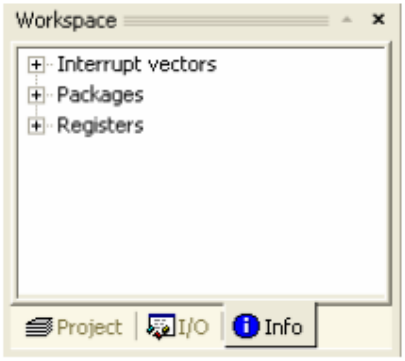


Figura 2.16. Ventana Info.

Como se aprecia en la Figura 2.16, la ventana Info se subdivide en los siguientes campos:

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

Packages. Enlista los encapsulados disponibles para el dispositivo, y la correspondiente distribución de pines para cada encapsulado. La ayuda informativa sobre las herramientas tendrá información sobre el uso de la mayoría de los pines. Así

mismo, es una característica útil si se está depurando HW y SW, se tiene la información necesaria de pines en todo momento en este campo.

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

La Ventana Watch

En la Figura 2.17 se aprecia la ventana para vigilancia de variables, esta es la ventana Watch.

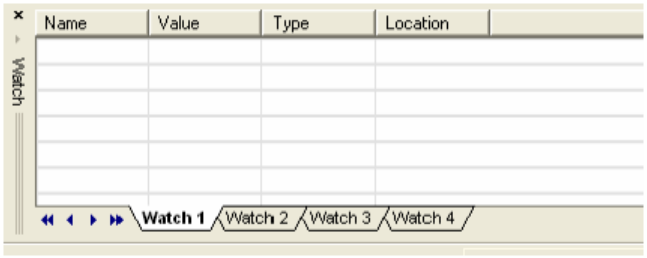


Figura 2.17. Vista Watch.

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

Diferentes lenguajes definen diferentemente su alcance, pero estar dentro del alcance significa que la variable actualmente existe en la memoria en la localidad donde el programa detiene su ejecución.

La Ventana Memory

Un microcontrolador como el AVR no puede hacer nada sin memoria. El código ejecutado está en la memoria de programa, las variables se ubican en la SRAM (principalmente), los registros de I/O se distribuyen dentro del área de memoria de I/O y la EEPROM es otra área de memoria. La ventana Memory está disponible para visualizar todos los diferentes tipos de memoria asociados a los dispositivos AVR. Y en relación a la ventana Watch y la I/O, el área expuesta en la pantalla se actualiza automáticamente cuando ocurre una detención temporal (break) de la ejecución. En la Figura 2.18 se aprecia la ventana Memory.

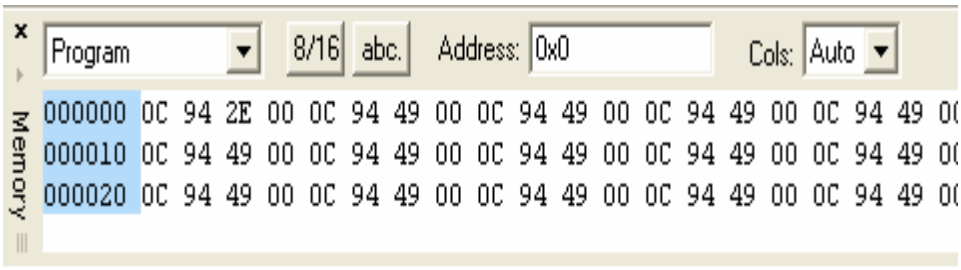


Figura 2.18. Vista Memory.

La Barra de Estado

La barra de estado siempre indica el nombre del dispositivo actual, la plataforma de depuración, el estado de ejecución y la ubicación del cursor dentro de la ventana Editor; esto se puede verificar en la Figura 2.19.

Figura 2.19. Barra de Estado.

2.3.3 Generación de Proyectos

Al iniciar la aplicación ya sea desde el icono de AVR Studio 4 en el escritorio o en el manu inicio se desplegara la siguiente ventana que se muestra en la figura 2.20.

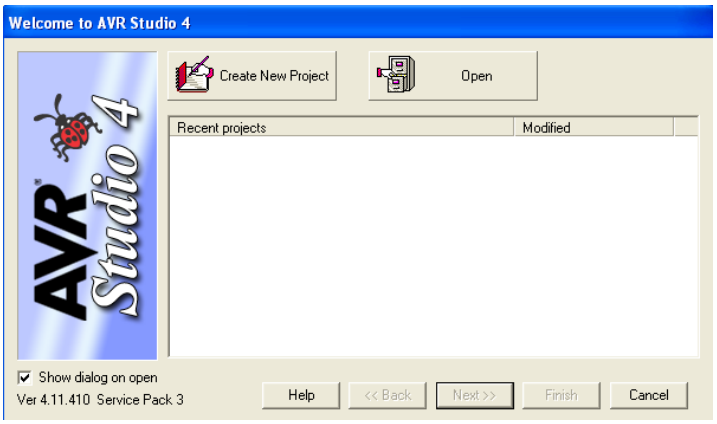


Figura 2.20. Creación de nuevo proyecto.

Para crear un nuevo proyecto el usuario debe hacer clic en el botón Create New Project que aparece en la Figura 2.20.

Luego, el usuario debe configurar el tipo de proyecto que desea crear, el nombre del archivo y el lugar donde será guardado.

Esto se ve en la figura 2.21.

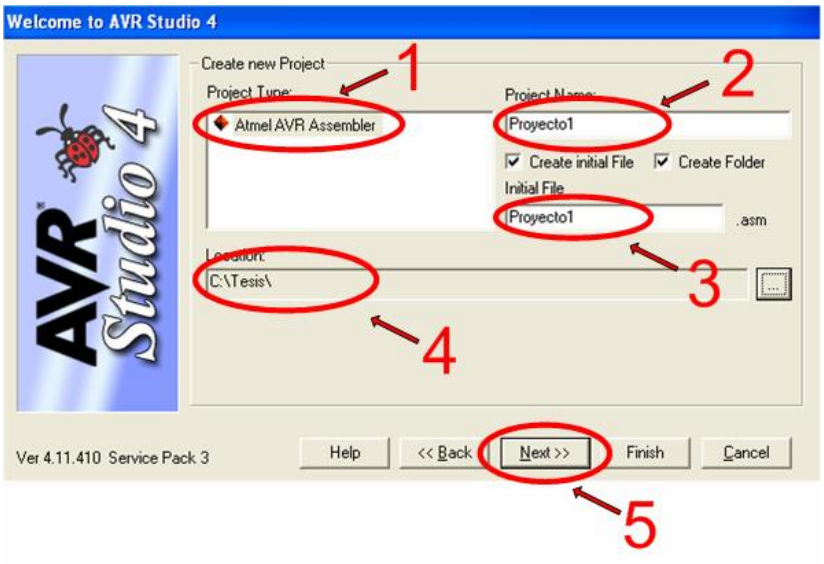


Figura 2.21. Configuración de nuevo proyecto.

1. Todos los tipos de proyecto disponibles se listan en el cuadro Project type list. El usuario debe hacer clic aquí (en Atmel AVR Assembler) para dejar saber al AVR Studio que se quiere crear un programa en ensamblador. Si se instala el compilar winavr se podrá crear proyecto en lenguaje C. Pero la configuración es la misma.

2. Aquí se coloca el nombre del proyecto, puede ser cualquiera.

3. Aquí el usuario puede especificar si el AVR Studio debe crear automáticamente un archivo ensamblador. El nombre del archivo puede ser cualquiera.

4. Seleccionar la ruta donde se quiere guardar el proyecto y los archivos.

5. Verificar siempre una vez más y asegurarse de que las casillas de verificación estén seleccionadas. Una vez seguro, presionar el botón “Next>>”.

A continuación, siguiendo el orden que aparece en la Figura 2.22, el usuario debe escoger la plataforma de depuración.

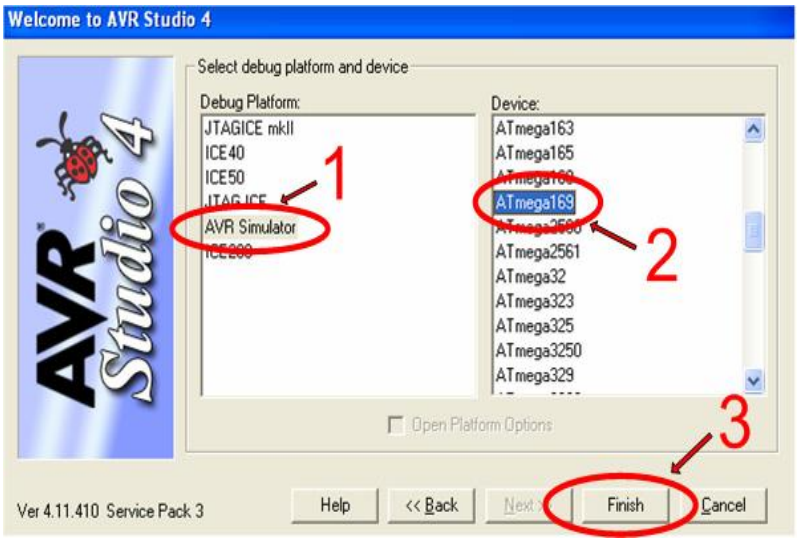


Figura 2.22. Selección de plataforma para depuración y del dispositivo a usar.

1. El AVR Studio 4 soporta un amplio rango de herramientas para emulación y depuración. Por conveniencia el usuario debe seleccionar la funcionalidad de simulación incluida, el AVR Simulator; ya que, el resto de opciones requieren de H/W especial.

2. Aquí el usuario debe seleccionar el microcontrolador que utilizará en su aplicación, en este caso ATmega169.

3. Verificar una vez más, luego presionar “Finish” para crear el proyecto e ir a editar archivo ensamblador (.asm).

El AVR Studio iniciará y abrirá un archivo .asm.

En la vista Editor el usuario debe escribir el código fuente, al hacerlo el usuario notará que el código fuente cambia de color. Esto se conoce como resaltamiento de la sintaxis y es muy útil para hacer código más legible.

Si el usuario desarrolla para un dispositivo específico debe tener en mente incluir el archivo *def.inc para el dispositivo. Cada dispositivo tiene su propio archivo *.inc que define todos los registros internos, bits y un poco de otras cosas que hace más simple escribir código para el dispositivo. Adicionalmente el archivo *.inc establece la directiva del dispositivo para el ensamblador, dejando saber al ensamblador para qué dispositivo está desarrollando.

Los archivos para el dispositivo se encuentran en la carpeta Files\Atmel\AVRTools\AVRAssembler\Appnotes de la PC. Un archivo include para el ATmega169 típicamente aparecerá con el nombre “m169def.inc”. No se debe dar una ruta con el archivo *.inc, con tal de que se encuentre en el directorio predefinido.

Una vez que el usuario haya ingresado correctamente y verificado el código fuente, debe presionar CTRL + F7 ó seleccionar [Build and Run] en el menú [Project], para compilar y ejecutar el código en el depurador.

En la ventana Output el usuario debe obtener el mensaje: Assembly complete, 0 errors, 0 warnings, que indica que el código se ha escrito correctamente; de no ser así debe revisar el archivo ensamblador (.asm) ayudándose de los mensajes de error.

Si deseamos adicionar un fichero fuente existente debemos dar clic derecho a source files en la ventana projects y elegir add existing source file y escoger el archivo fuente deseado.

2.4 PROTEUS

Proteus es una aplicación CAD (Diseño Asistido por Computador), compuesta de tres módulos:

1. ISIS (Intelligent Schematic Input System): es el módulo de captura de esquemas.
2. VSM (Virtual System Modelling): es el módulo de simulación, incluyendo PROSPICE.
3. ARES (Advanced Routing Modelling): es el módulo para la realización de circuitos impresos (PCB).

La herramienta de Proteus que se a usado en este proyecto se denomina ISIS, esta aplicación nos permite colocar en el área de trabajo todos los elementos que conformaran nuestro circuito, ya que posee una extensa librería de la mayoría de elementos que se utilizan para la creación de un circuito electrónico y eléctrico además de que esta librería esta en constante actualización, este circuito posteriormente lo podremos emular para poder así comprobar el correcto funcionamiento del mismo y poder hacer así las correcciones o ajustes necesarios, la herramienta ARES de PROTEUS no lo hemos usado en este trabajo ya que la implementación se ha realizado sobre un Protoboard y no es necesaria el PCB de cada ejercicio.

A continuación vemos el área de trabajo de ISIS en la figura 2.23. donde podemos apreciar las partes mas importantes del mismo.

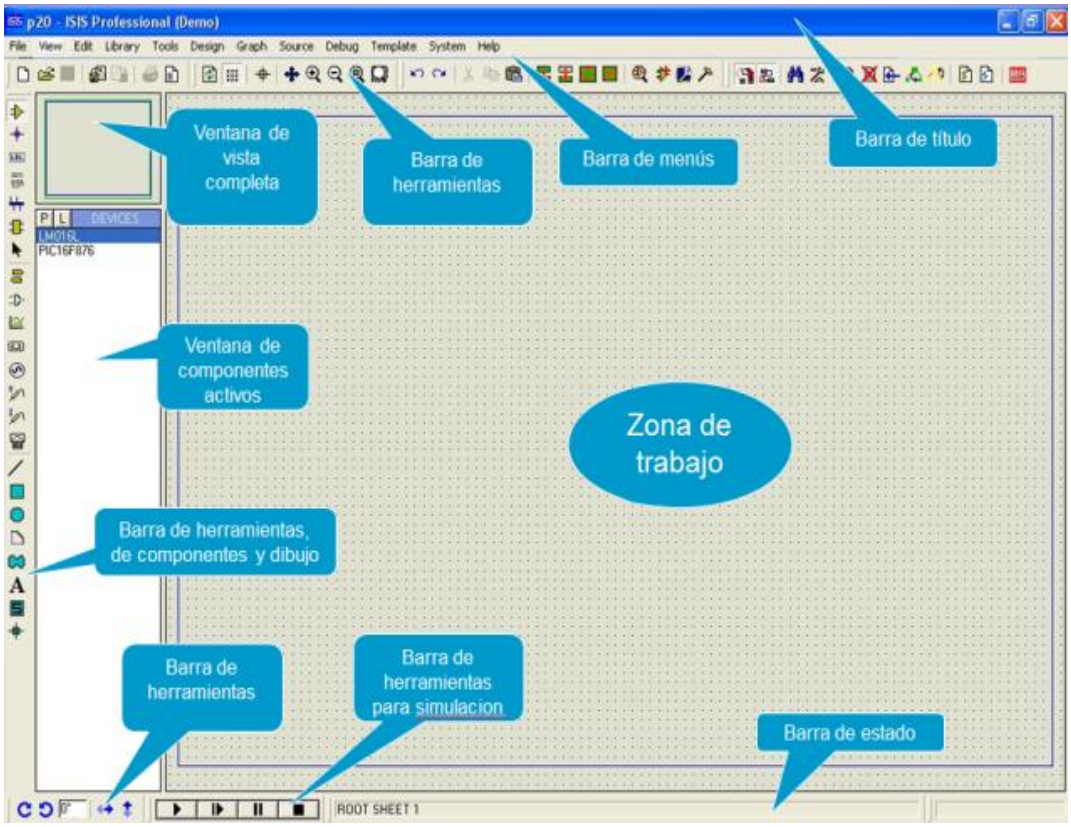


Figura 2.23. Área de trabajo de ISIS en PROTEUS.

Al abrir PROTEUS, lo primero que necesitamos es extraer los componentes que se van a utilizar en el circuito, para lo que debemos utilizar la barra de herramientas de componentes. Dando clic en el icono parecido a un OPAMP, luego dar clic en el icono P como se muestra en la figura 2.24.

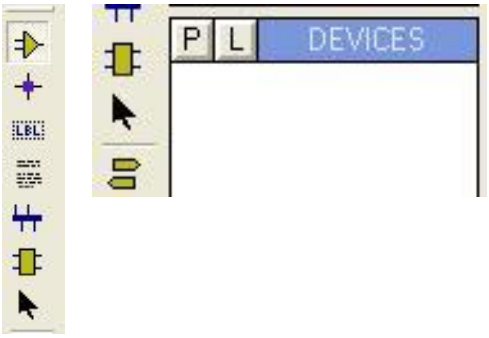


Figura 2.24. Barra de herramientas de componentes.

Al realizar esto se abrirá la ventana de librerías en donde podremos escoger los componentes dando doble clic en el componente uno a uno que vamos a usar para el circuito a simular, esto se muestra en la figura 2.25.

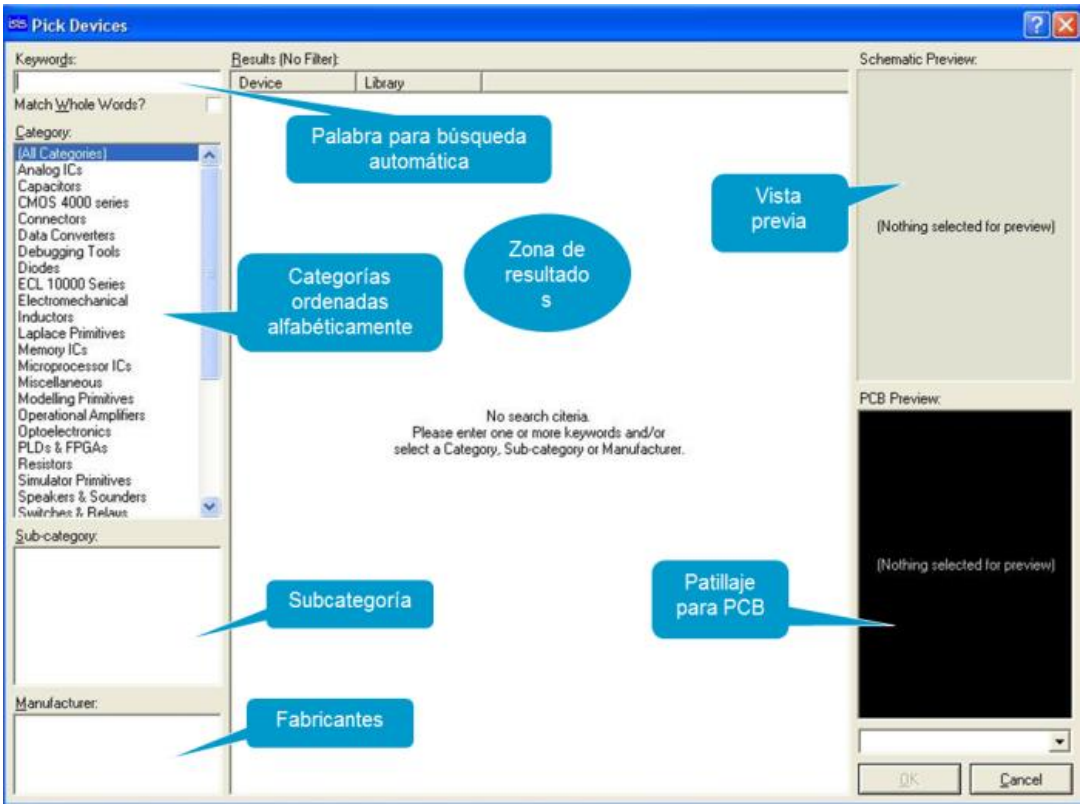


Figura 2.25. Ventana de Librerías de componentes.

A partir de aquí el programa es muy intuitivo y se maneja como lo un programa de dibujo, utilizando la mayoría de opciones comunes como copiar, mover, girar, etc.

Una vez realizado el esquema del circuito, la simulación se la realiza mediante la barra de simulación situada en la parte inferior izquierda de nuestra área de trabajo, en la figura 2.26. Podemos ver la barra de simulación la cual es bastante intuitiva ya que posee típicos botones de Inicio, Pausa y Stop.

Figura 2.26. Barra de Simulación de PROTEUS.

En la barra de menú en System podemos ingresar en la opción set animation options, donde encontraremos una ventana en la cual podemos configurar aspectos de animación nuestra simulación como mostrar el estado lógico de pines y nodos, también que muestre el voltaje en las líneas de esquema mediante colores y también la dirección de corrientes en las mismas lo que es de ayuda para evaluar el funcionamiento de nuestro circuito en la figura 2.27. Podemos observar la ventana de configuración de simulación.

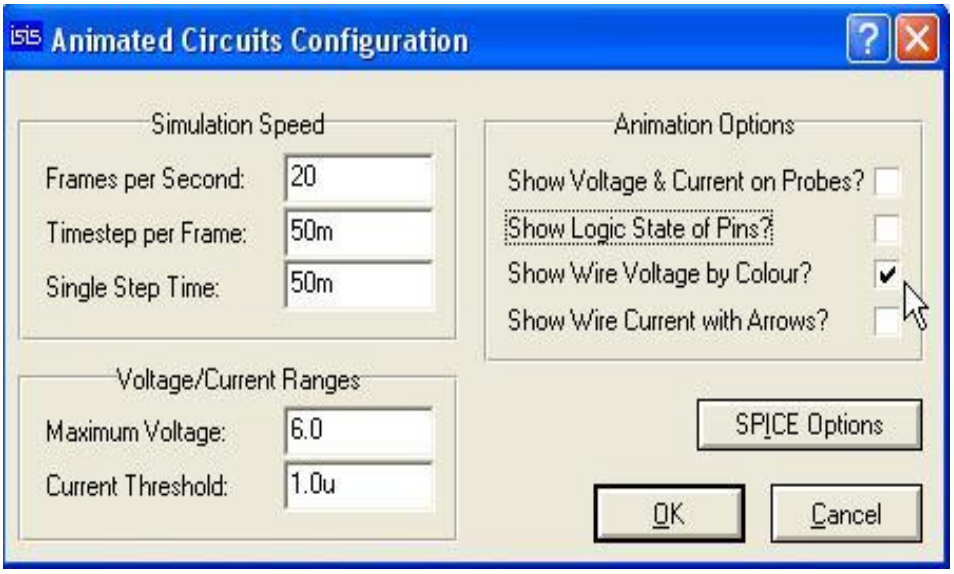


Figura 2.27. Ventana de configuración de animación de simulación.

CAPITULO 3

DESCRIPCION DE LA PLATAFORMA DE TRABAJO

3.1 Descripción de la Plataforma de trabajo desarrollada

En este proyecto se ha propuesto desarrollar una plataforma de trabajo en la cual podamos implementar diversos ejercicios que nos permitan ejecutar el aprendizaje de microcontroladores en este caso de Atmel.

La Plataforma de trabajo se ha implementado con el Kit AVR Butterfly de Aprendizaje de Atmel montado sobre un Protoboard, con su debida fuente de alimentación la cual está constituida por dos pares de pilas en sus respectivos porta pilas, ya que como Atmel utiliza tecnología CMOS podemos trabajar con voltajes bajos en este caso de 3.0 V y además que las pilas proveen de un nivel de voltaje estable, también se construyeron diversos cables de conexión entre el AVR Butterfly y el Protoboard y con la programación directa de podemos realizar entre el AVR Butterfly y una PC por medio de un simple cable serial hace de esta sencilla plataforma un herramienta muy versátil y útil ya que podemos montar sobre ella un sin número de ejercicios que nos permiten el correcto y practico aprendizaje de los Microcontroladores Atmel.

En este caso nuestro tema trata sobre los puertos de entrada y salida de un microcontrolador.

3.2 Partes de la Plataforma de Trabajo

La plataforma desarrollada consta de las siguientes partes:

- Kit AVR Butterfly de aprendizaje de Atmel. Ver Figura 3.1.
- Protoboard de tamaño medio modificada. Ver Figura 3.2. a).
- Dos porta pilas de tamaño doble AA. Ver Figura 3.2. b).
- Cable USB Serial en caso de que la PC no tenga puerto serial. Ver Figura 3.3 a).
- Cable serial para conexión con PC y AVR Butterfly. Ver Figura 3.3 b).
- Conjunto de varios cable para conexionado entre el Kit AVR Butterfly y la Protoboard. Ver Figura 3.4.
- Circuito de encendido y reset del Kit AVR Butterfly. Ver Figura 3.5.

En la Figura 3.6 podemos observar la plataforma implementada.

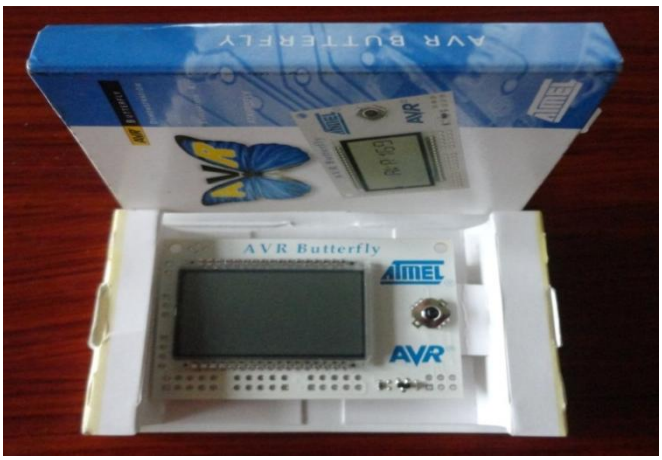
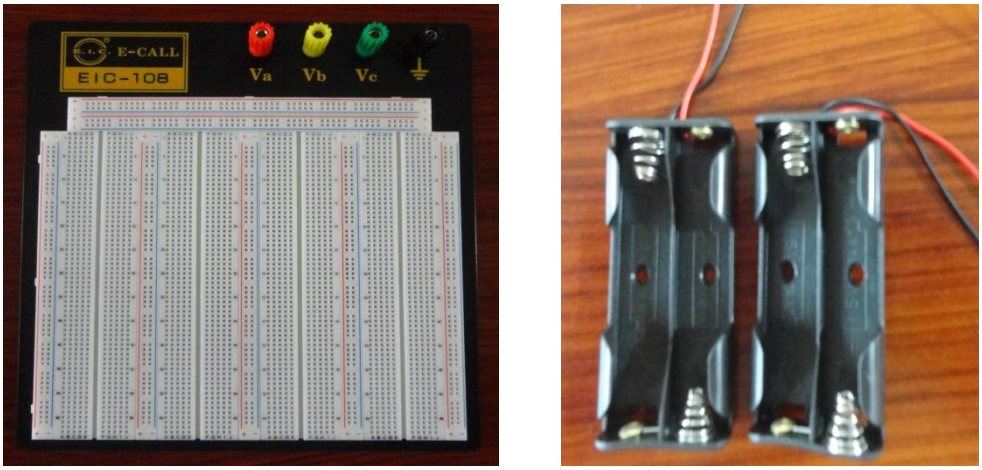


Figura 3.1. Kit AVR Butterfly.



a) Protoboard **b) Porta pilas**

Figura 3.2. Protoboard y porta pilas tamaño doble AA.

a) Cable USB Serial para PC **b) Cable serial para conexión con AVR Butterfly**

Figura 3.3. Cable USB Serial y cable serial para conexión con AVR Butterfly.

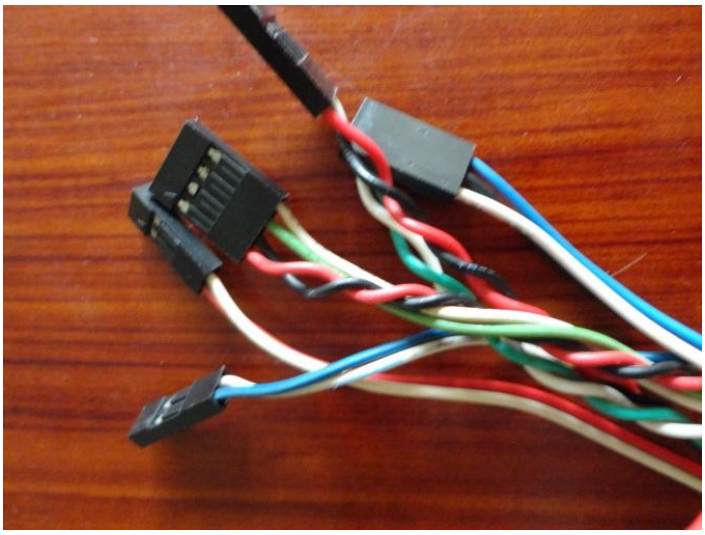


Figura 3.4. Cable para conexionado de AVR Butterfly con el Protoboard.

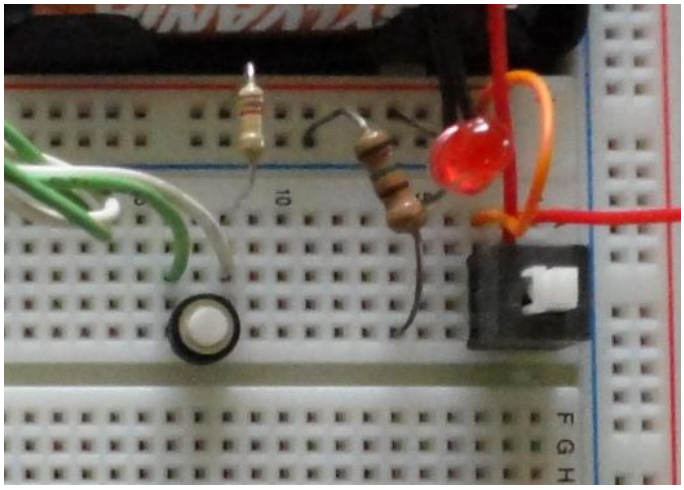


Figura 3.5. Circuito de encendido y reset para AVR Butterfly.

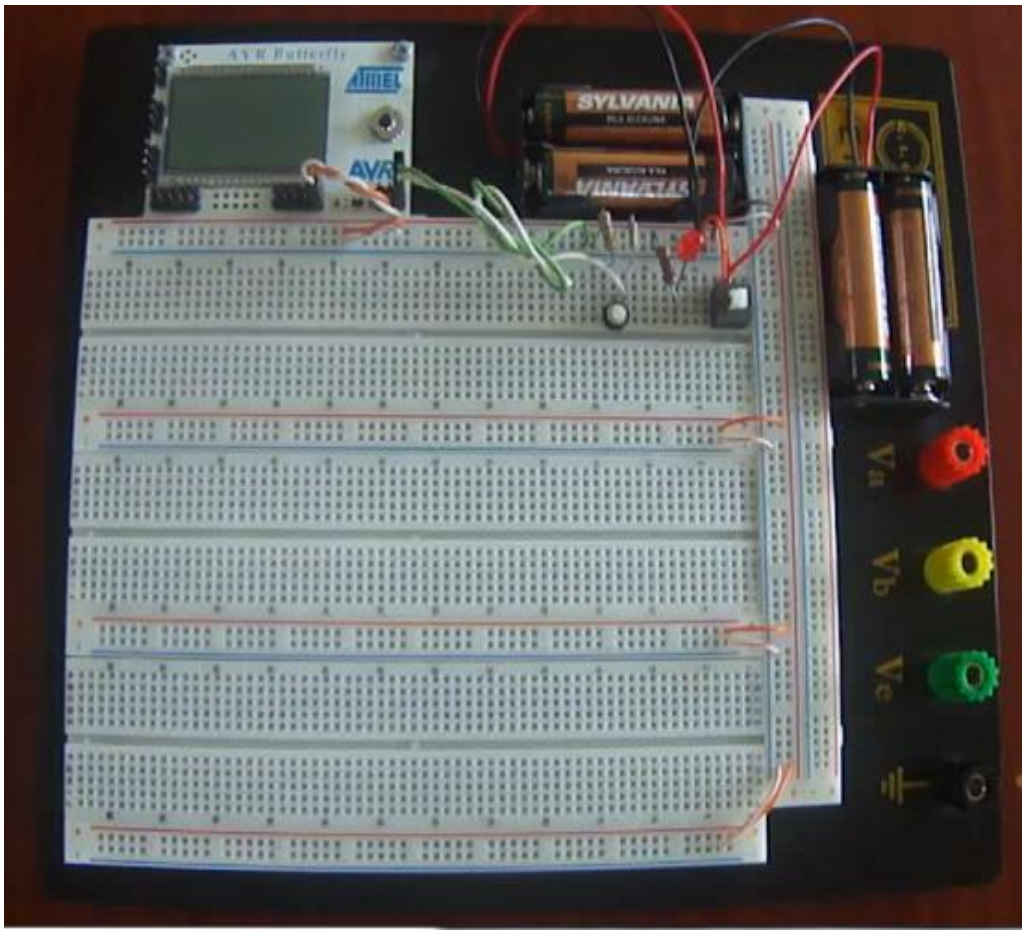


Figura 3.6.Plataforma implementada lista para montar ejercicios.

3.3 Descripción de los Ejercicios a Desarrollar

A continuación se hará una breve descripción y explicación de cada uno de los ejercicios que se desarrollaran e implementaran en nuestra plataforma de trabajo.

Se incluye la lista de materiales e imágenes para poder montar los ejercicios y así probar su funcionamiento, en el capitulo 4 de se detallaran los diagramas de flujo y la programación de cada uno.

3.4 Descripción de Ejercicio 1

3.4.1 Tema

"**Lectura, escritura de Puertos I/O y Visualización de Alta Impedancia**".

El presente ejercicio tiene como finalidad enseñar un programa básico de entradas y salidas para que el estudiante observe la programación más sencilla en este sentido como inicio para posteriores ejercicios.

Aquí lo se hace es mediante un Dip-switch prender y apagar leds correspondientes al switch operado teniendo mas prioridad los pines menos significativos, solo se mantendrá encendido uno de los led, el menos significativo si hay mas de un switch activado.

Además que en el primer switch podemos visualizar cuando uno de los pines cambia a estado de alta impedancia desde el estado de salida el cual mantenía encendido dicho led que posteriormente se apaga al cambiar a estado de alta impedancia.

En la figura 3.7 podemos observar el circuito implementado.

3.4.2 Lista de Materiales

Los materiales a utilizar en este ejercicio son los siguientes:

- Un Dip switch de 8 o más pines
- 9 Leds (8 rojos y 1 verde).
- 9 Resistencias de 330 ohm o 1K
- Plataforma de Trabajo con AVR Butterfly

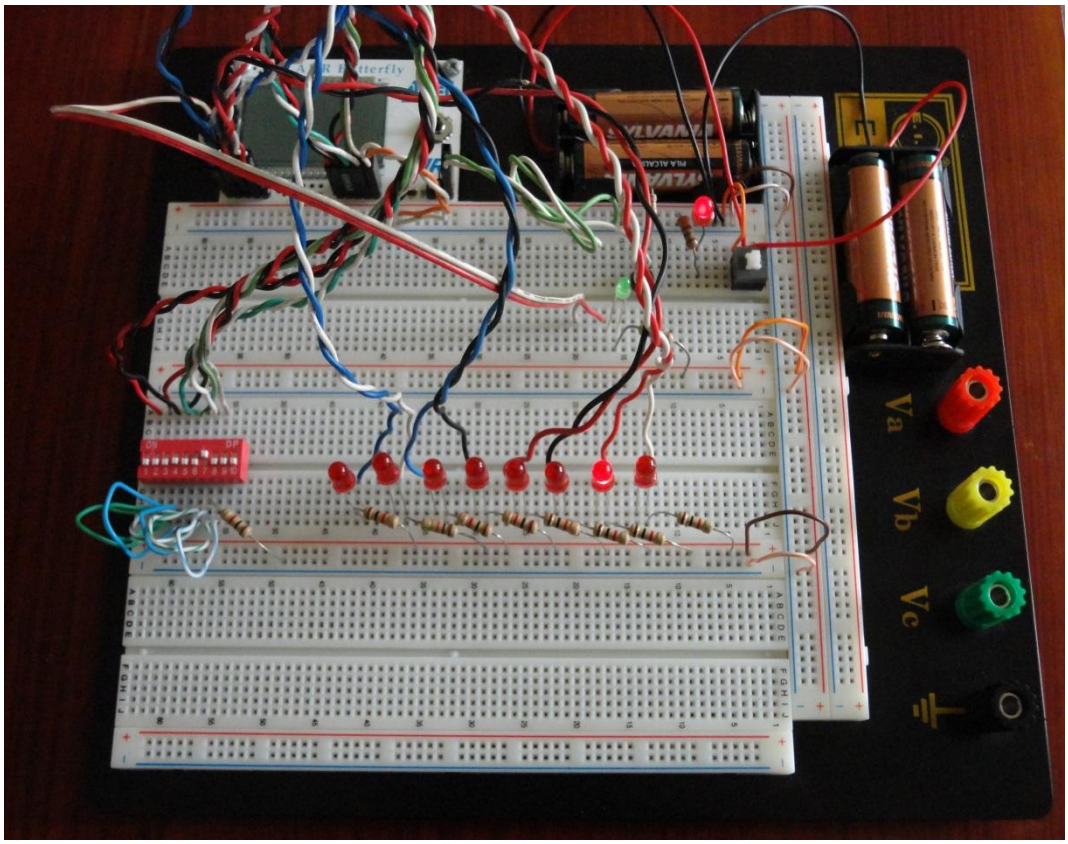


Figura 3.7. Circuito implementado del Ejercicio 1 "Lectura, escritura de Puertos I/O y Visualización de Alta Impedancia"

3.5 Descripción de Ejercicio 2

3.5.1 Tema

"**Visualización en display desde 2 puertos alternados entre salida y alta impedancia**".

Para este ejercicio se usaron los puertos D y B como salidas de los números y el puerto E como entrada, para generar estos números en un display, el puerto B genera los números pares y el puerto D los números impares pero lo harán en un mismo bus de datos para lograr esto sin que haya conflicto en el envío de datos se coloca a uno de los puertos en alta impedancia mientras el otro puerto está generando los números, luego se hace el proceso inverso, el puerto D estará generando los números impares mientras el puerto B se pondrá en alta impedancia.

En la figura 3.8 se puede observar el circuito implementado de este ejercicio.

3.5.2 Lista de Materiales

Los materiales a utilizar en este ejercicio son los siguientes:

- 2 Push Botton.
- 1 Display 7 segmentos cátodo común.
- 9 Resistencias de 330 ohm o 1K
- Plataforma de Trabajo con AVR Butterfly

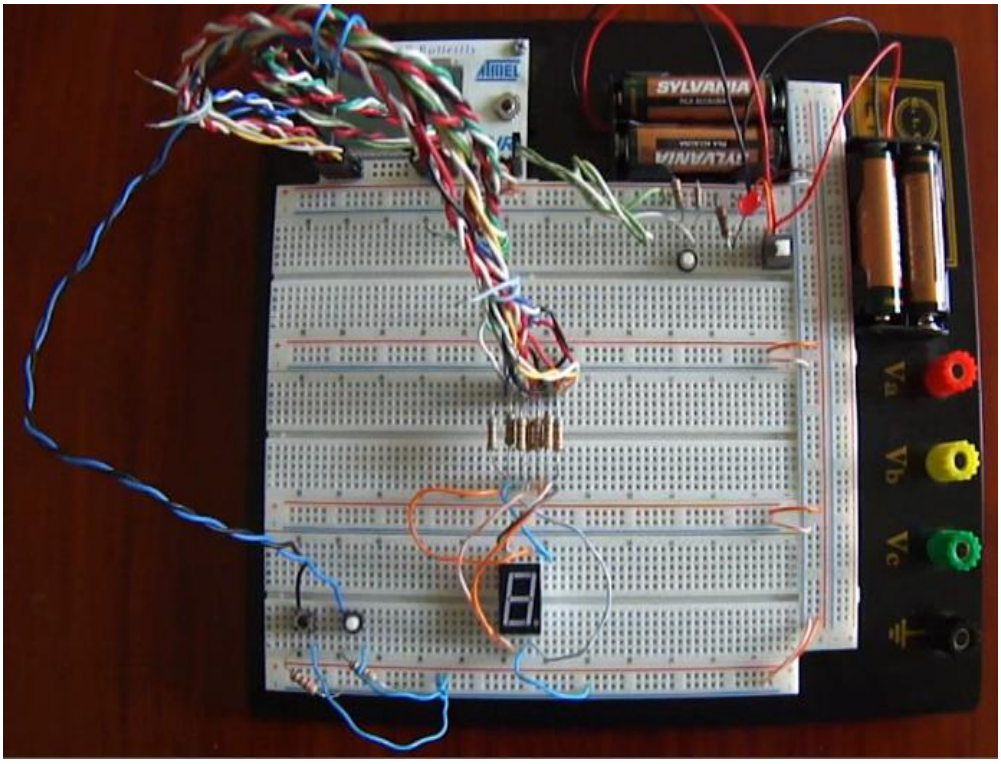


Figura 3.8. Circuito implementación del ejercicio 2 "Visualización en display desde 2 puertos alternados entre salida y alta impedancia".

3.6 Descripción de Ejercicio 3

3.6.1 Tema

"**Punta Lógica**".

La Punta Lógica es una herramienta de mucha utilidad para cuando se trabaja con circuitos digitales y se quiere hacer un seguimiento de cómo están las señales en el mismo, así podemos saber si en algún punto del circuito se encuentra un estado lógico

alto, bajo o en alta impedancia. El circuito trabaja de mediante dos pines los cuales están conectados entre sí por medio de una resistencia de 47 Kohm, el extremo del pin configurado como entrada es la que debemos utilizar como la punta lógica la cual está definida, luego de colocar la punta en lugar a testear presionamos en centro del joystick del AVR Butterfly, para luego visualizar en el LCD el estado presente en la punta. Así cuando queramos realizar una prueba siempre lo haremos presionando el centro del joystick.

3.6.2 Lista de Materiales

Los materiales a utilizar en este ejercicio son los siguientes:

- 1 Resistencias de 56K (se puede utilizar 2 resistencias de 120K en paralelo).
- 2 Resistencias de 330 ohm o 1K
- Plataforma de Trabajo con AVR Butterfly.

En la Figura 3.9. Podemos observar la implementación de este ejercicio.

Como se podrá observar la implementación es bastante sencilla ya que prácticamente todo el proceso se encuentra en la programación del microcontrolador, en posteriores revisiones de este ejercicio se podría implementar alguna función adicional como medición de frecuencia.

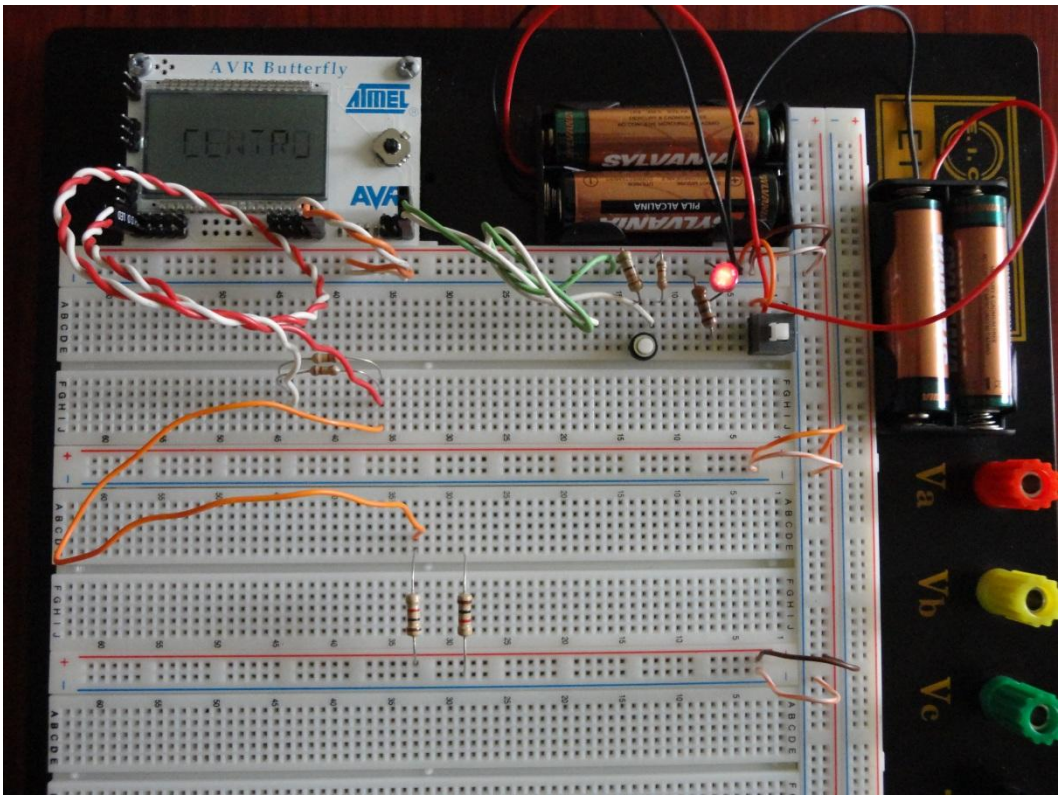


Figura 3.9. Circuito implementado del ejercicio 3 “Punta Lógica”.

3.7 Descripción de Ejercicio 4

3.7.1 Tema

“**Menú de Entradas, Salidas y Alta Impedancia**”.

En este ejercicio de realizo un pequeño menú en LCD y por medio del joystick podemos elegir una de las opciones que son: Joystick hacia arriba configuramos los pines BO y B1 como salidas y podemos encender y apagar dos leds conectados a los mismos cada vez que presionamos hacia arriba.

Joystick hacia la derecha configuramos los mismos pines B0 y B1 ahora como entradas y se conectan a dos interruptores los cuales podemos manipular para visualizar en el LCD el estado lógico en el que se encuentran dichos interruptores.

Joystick hacia la izquierda opción visualizar Alta Impedancia (HZ), aquí observamos que se enciende un led que está conectado en un pin el cual después de 1 segundo pasara a estar en alta impedancia por lo que dicho led se apagara y mostraremos en el LCD que ahora se encuentra en Alta Impedancia.

3.7.2 Lista de Materiales

Los materiales a utilizar en este ejercicio son los siguientes:

- 1 IC 74HC126 (4 Buffers de 3 estados).
- 3 Leds.
- 5 Resistencias de 330 ohm o 1K.
- 2 Interruptores.
- Plataforma de Trabajo con AVR Butterfly.

En la figura 3.10 podemos observar la implementación de este ejercicio.

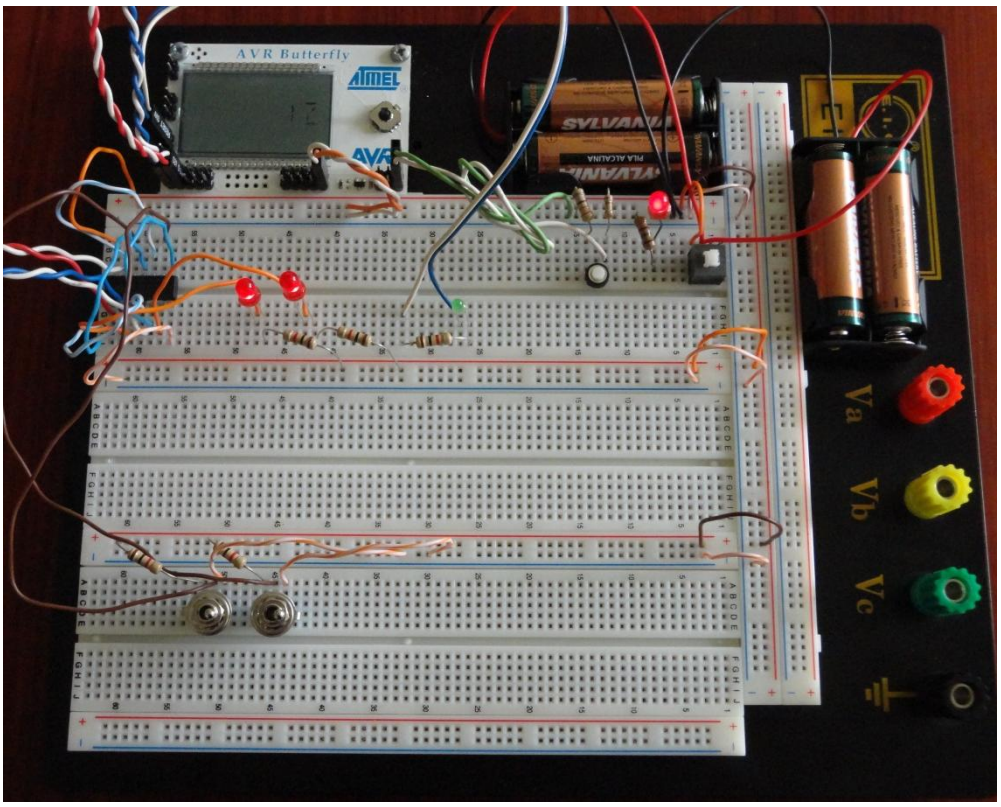


Figura 3.10. Circuito implementado del ejercicio 4 "Menú de Entradas, Salidas y Alta Impedancia".

3.8 Descripción de Ejercicio 5

3.8.1 Tema

"**Juego de Reacción**".

En este ejercicio podemos ver cómo utilizar tanto entradas como salidas para implementar un divertido juego.

En este ejercicio tenemos ocho Leds los cuales al iniciar el juego ya sea enciendo la plataforma o reseteando el Butterfly encenderá un Led aleatorio y deberemos antes de un tiempo presionar el Push botton correspondiente al Led que encendió si no lo hacemos se encenderá un Led Amarrillo indicando que hemos fallado, si lo hacemos bien encenderá un Led verde indicando que lo hemos hecho bien, se tienen 5 oportunidades caso contrario se pierde el juego permaneciendo el led amarillo encendido, se gana acertando 4 veces y en ese caso permanecerá encendido el led verde.

3.8.2 Lista de Materiales

Los materiales a utilizar en este ejercicio son los siguientes:

- 8 Push Botton.
- 8 Leds Rojos, 1 Led verde y 1 Led amarillo.
- 18 Resistencias de 330 ohm o 1K
- 1 Potenciómetro de 10K.
- Plataforma de Trabajo con AVR Butterfly

En la figura 3.11 Podemos observar el circuito implementado del ejercicio 5.

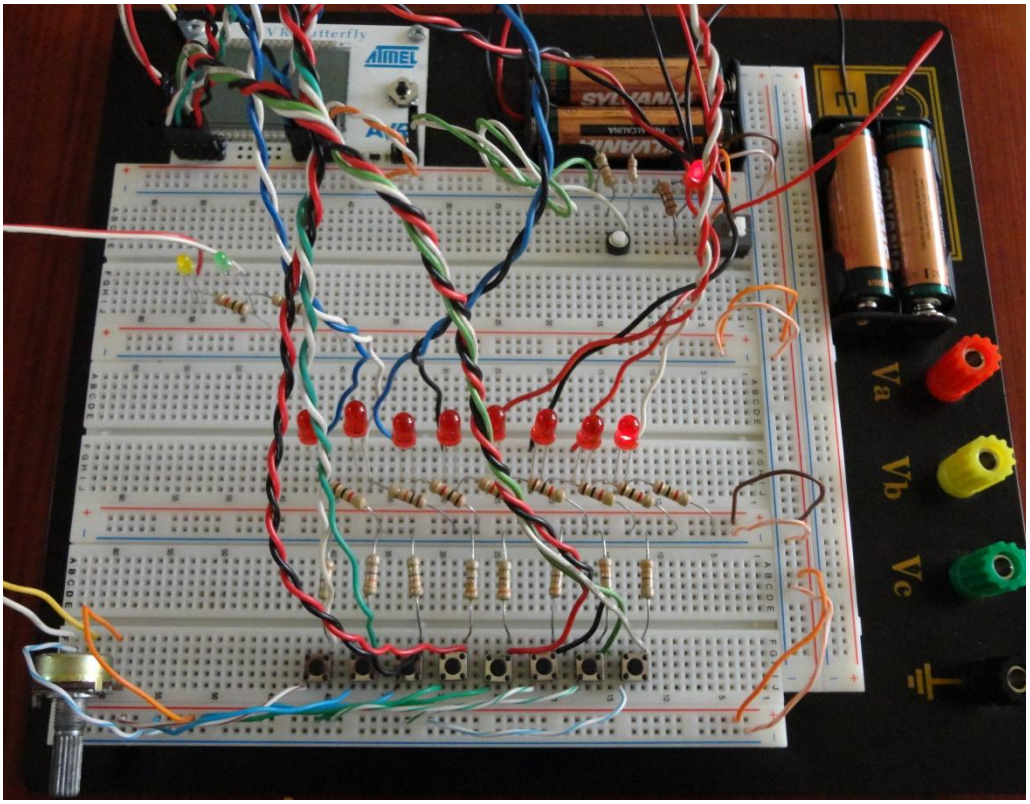


Figura 3.11. Circuito implementado del ejercicio 5 “Juego de Reacción”.

CAPITULO 4

DESARROLLO DE EJERCICIOS DE LA PLATAFORMA DE TRABAJO

En el presente capítulo se desarrollaran y analizan a profundidad cada uno de los ejercicios presentados en el capitulo 3 y que representan lo que es manejo de Puertos I/O del Atmega169 y el Kit AVR Butterfly de Atmel.

4.1 Desarrollo de Ejercicio 1

4.1.1 Tema

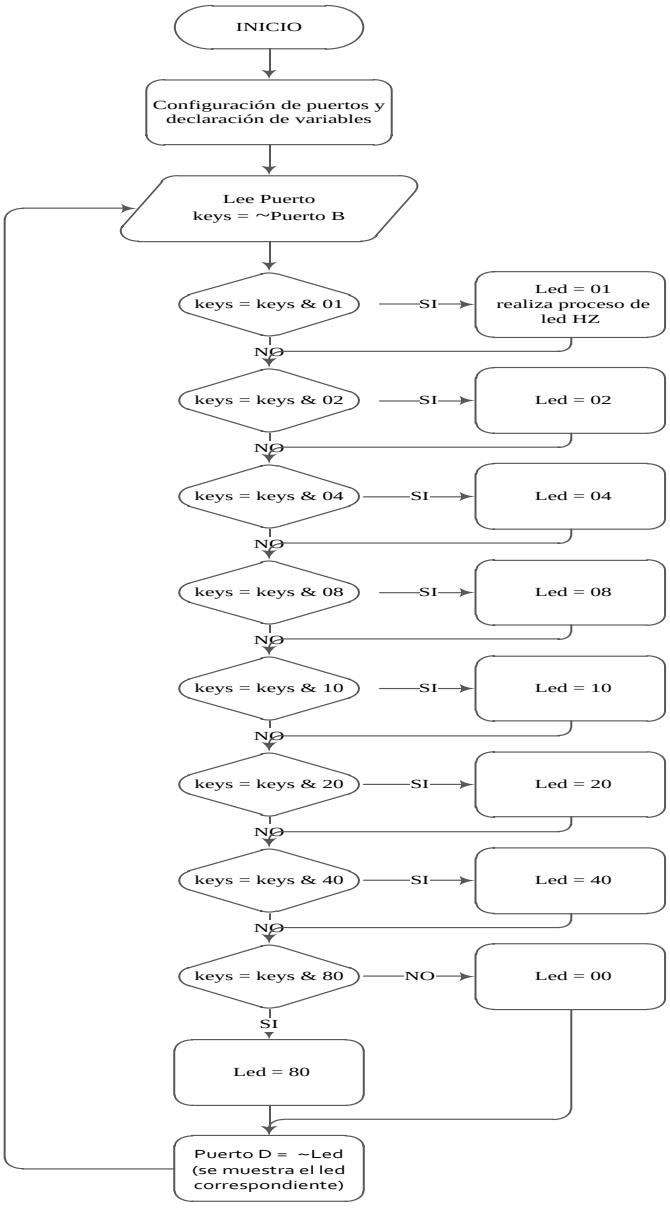
"**Lectura, escritura de Puertos I/O y Visualización de Alta Impedancia**".

El presente ejercicio tiene como finalidad enseñar un programa básico de entradas y salidas para que el estudiante observe la programación más sencilla en este sentido.

Aquí lo que se hace es mediante un Dip-switch prender y apagar leds correspondientes al switch operado teniendo mas prioridad los pines menos significativos, solo se mantendrá encendido uno de los led, el menos significativo si hay mas de un switch activado, además que en el primer switch podemos visualizar cuando uno de los pines cambia a estado de alta impedancia desde el estado de salida el cual mantenía encendido dicho led que posteriormente se apaga al cambiar a estado de alta impedancia.

4.1.2 Diagrama de Bloques

4.1.3 Diagrama de Flujo



4.1.4 Descripción del Diagrama de Flujo

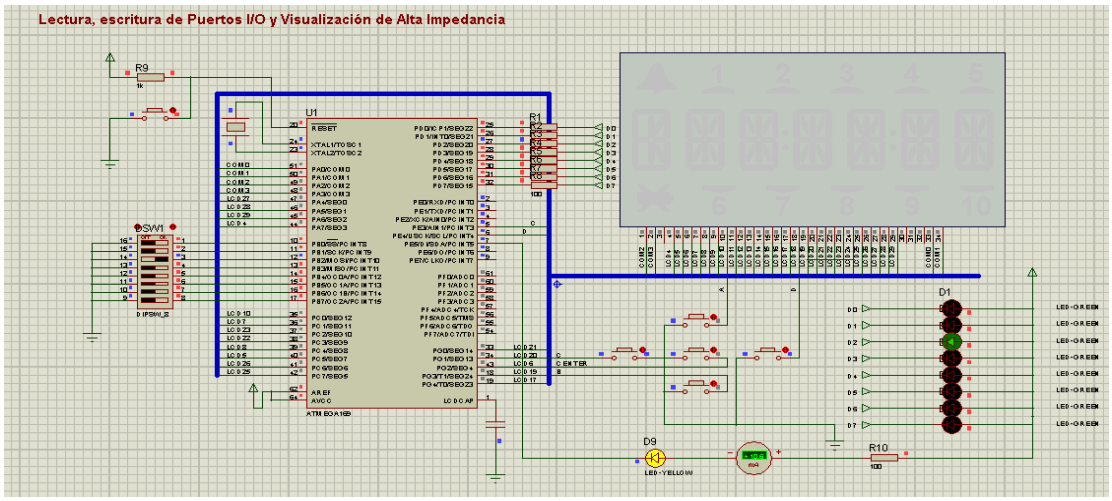
1. Se declaran las variables “keys” y “led” ambas variables son de tipo entero sin signo de 8 bits ya que no necesitamos variables de mayor tamaño, se configura el puerto D como salida y el puerto B como entrada, además se activa las resistencias de pull up en el puerto B y se activa la salida en el puerto D en alto, para la parte de alta impedancia configuramos el puerto E como salida y en estado bajo, para que el led conectado al mismo permanezca encendido.
2. Se genera un lazo infinito, la variable “keys” será igual al estado negado que tenga el puerto B, esto es por la lógica que se utiliza en la configuración de los puertos, esto lo podemos visualizar y entender mejor observando la simulación.
3. Una vez obtenido el valor en el puerto de entrada se realizan comparaciones para encontrar que switch fue accionado y por lo tanto setear la variable “led” con el valor correspondiente para luego al final del algoritmo enviar “led” negada por la lógica al puerto D y visualizar el led correspondiente.
4. Si el switch accionado corresponde al primero se realiza una proceso adicional al de setear “led” en uno, se reconfigura el puerto E a estado de entrada en alta impedancia por lo que el led conectado a dicho puerto que permanecía encendido se apagara y podemos de esta forma visualizar el estado de Alta Impedancia “HZ”.

4.1.5 Código de Programa en lenguaje C

```
#include <avr/io.h>  // librería que contiene la configuración de todos los registros de Atmel
#define F_CPU 8000000  // se define el valor del reloj para la función delay
#include <util/delay.h>  // librería que contiene las funciones delay

int main(void)
{
   unsigned char  led;  // declaración de variables
   unsigned char  keys;

  CLKPR = (1<<CLKPCE);  //Habilito configuración de reloj
  CLKPR = (0<<CLKPS1) | (0<<CLKPS0);  //configuro reloj a 8 MHz

   DDRD  = 0xff;  // seteamos Puerto D como salida
   DDRB  = 0x00; // seteamos Puerto B como entrada
   PORTB = 0xff;  // activamos resistencias Pull-up del puerto B
   PORTD = 0xff;  // puerto D en Alto y Led Apagados

   for (;;)                // lazo Infinito
  {
   keys = ~PINB;     // leemos las entrada del PINB
   DDRE  = 0XFF;  // configuramos el puerto E como salida en bajo
   PORTE = 0X00;

   if ( keys & 0x01 )         // si son presionados varios switch al mismo tiempo,
       {                      // solo el menos significativo es reconocido
          led = 0x01;        // seteamos led de acuerdo al switch accionado
          DDRE  = 0X00;  // reconfiguramos el puerto E como entrada en alta impedancia
          PORTE = 0X00;
          _delay_ms(100);
       }
     else if ( keys & 0x02 )
          led = 0x02;
      else if ( keys & 0x04 )
          led = 0x04;
      else if ( keys & 0x08 )
          led = 0x08;
      else if ( keys & 0x10 )
          led = 0x10;
      else if ( keys & 0x20 )
          led = 0x20;
      else if ( keys & 0x40 )
          led = 0x40;
      else if ( keys & 0x80 )
          led = 0x80;
      else
          led = 0;

          PORTD = ~led;          // Enciende el Led correspondiente
   }
}
```

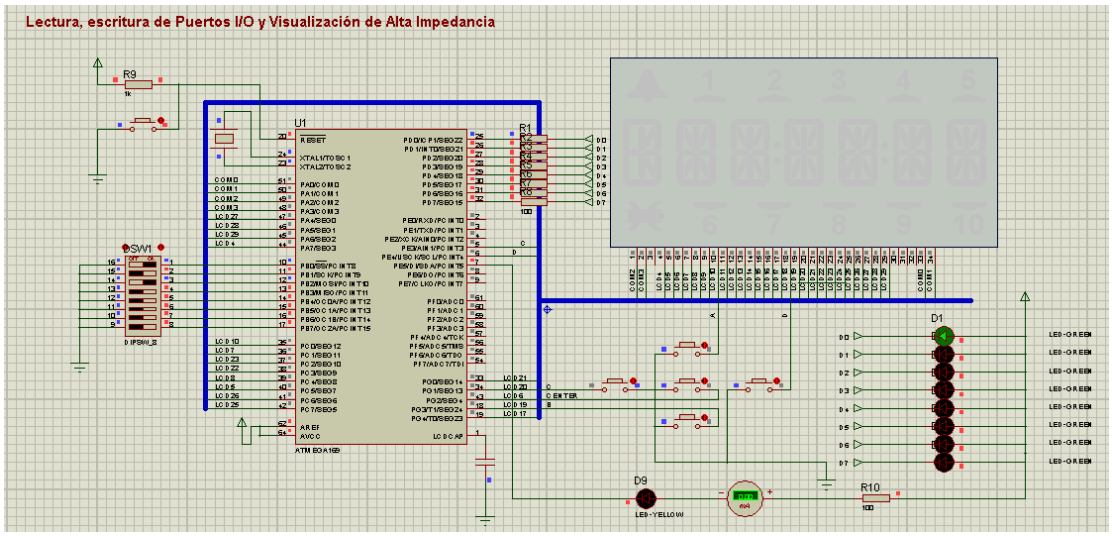
4.1.6 Simulación

Esquema 1



Presionado el switch 3 se enciende el led ubicado en la misma posición y el led HZ permanece encendido

Esquema 2



En este esquema se muestra como se le da mas prioridad al pin Bo porque es el bit menos significativo con respecto al pin B4 que es mas significativo q el anterior por lo tanto no se enciende el led correspondiente al B4 y además el puerto E donde esta cableado el LED HZ esta en alta impedancia por lo tanto no hay corriente el circuito no cierra y el LED HZ se apaga.

4.2 Desarrollo de Ejercicio 2

4.2.1 Tema

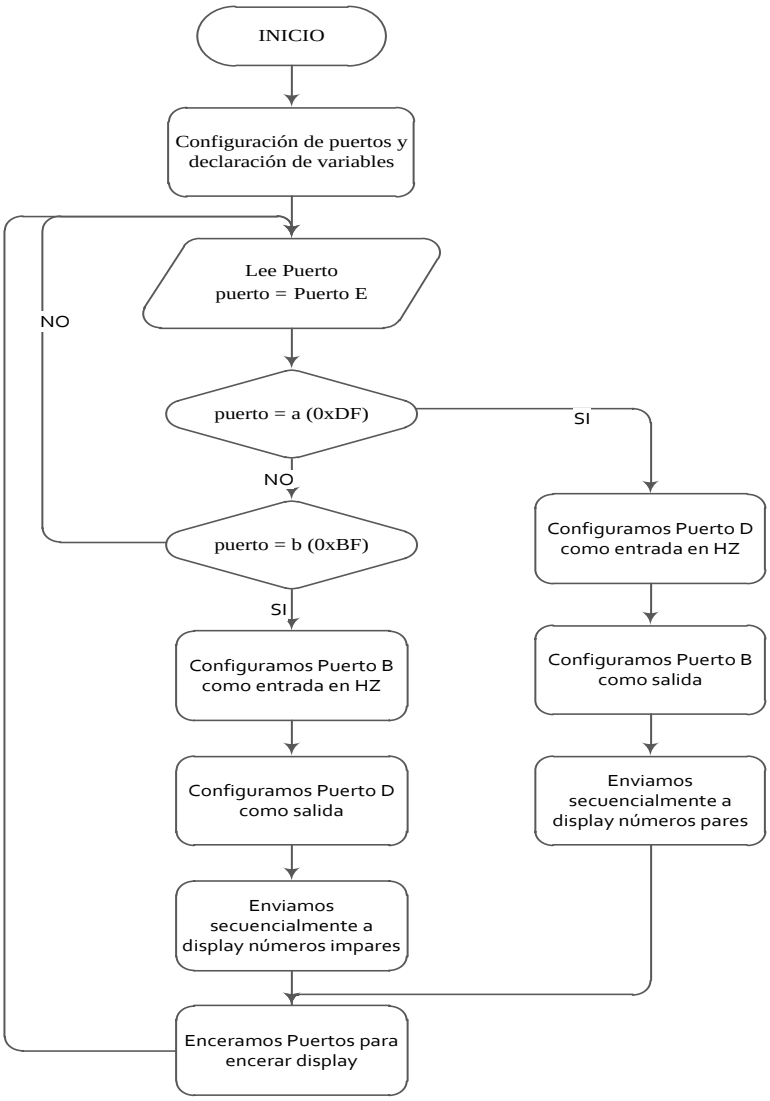
"**Visualización en display desde 2 puertos alternados entre salida y alta impedancia**".

Para este ejercicio se usaron los puertos D y B como salidas de los números y el puerto E como entrada para generar estos números en un display, el puerto B genera los números pares y el puerto D los números impares pero lo harán en un mismo bus de datos para lograr esto sin que haya conflicto en el envío de datos se coloca a uno de los puertos de salida en alta impedancia mientras el otro puerto está generando los números, luego se hace el proceso inverso, el puerto D estará generando los números impares mientras la salida puerto B se pondrá en alta impedancia, se colocaron 2 amperímetros en los pines de las salidas para mostrar la corriente que sale de los puertos al display pero cuando uno de los puertos está en alta impedancia la corriente

generada o recibida por el puerto es cero la generación de números empezara cuando alguno de los switch colocados en el puerto E sea presionado.

4.2.2 Diagrama de Bloques

4.2.3 Diagrama de Flujo



4.2.4 Descripción del Diagrama de Flujo

1. Se declaran las variables “puerto”, “a” y “b” luego se configura el puerto E como entrada con Pull Up.

2. Leemos el Puerto E y lo asignamos a la variable “puerto”.

3. Comparamos la variable “puerto” que tiene la información de entrada con la variable “a” o “b” que esta enmascarada con el correspondiente byte que indica que se a presionado el Push botton que comenzara el envió de números pares por el Puerto B o los impares por el Puerto D.

4. Si se presiona “a” configuramos el Puerto D como entrada en alta impedancia y el Puerto B como salida.

5. Enviamos por el Puerto B secuencialmente y con cierto delay los bytes correspondientes a los números pares para que se muestren en el display de 7 segmentos.

6. Si se presiono “b” configuramos el Puerto B como entrada en alta impedancia y el Puerto D como salida para enviar los números impares a ser mostrados en el mismo display y a través del mismo bus de datos que el anterior.

7. Y continua el lazo en espera de otra pulsación del Push botton.

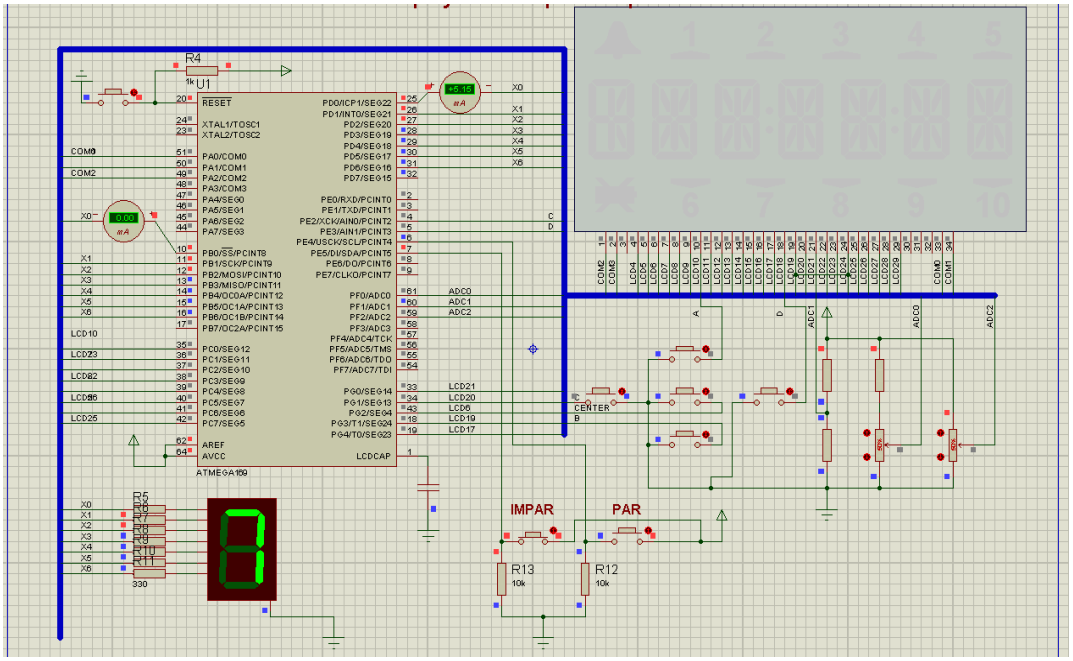
4.2.5 Código de Programa en lenguaje C.

```
#include <avr/io.h>  // librería que contiene la configuración de todos los registros de Atmel
#define F_CPU 8000000  // se define el valor del reloj para la función delay
#include <util/delay.h>  // librería que contiene las funciones delay


// Funcion Main

int main (void)
{

        unsigned char a,b;   // declaramos variables
        int puerto;

        CLKPR = (1<<CLKPCE);  //Habilito configuración de reloj
        CLKPR = (0<<CLKPS1) | (0<<CLKPS0);  //configuro reloj a 8 MHz

        DDRE = 0x00; //puertos del atmega169 como entrada con pull up
        PORTE = 0x00;

        DDRB = 0x00; //puertos del atmega169 como entrada en HZ
        PORTB = 0x00;

        DDRD = 0x00; //puertos del atmega169 como entrada en HZ
        PORTD = 0x00;

  while (1)
        {
        puerto = PINE;  // leemos el puerto E y lo asignamos a la variable puerto

        a = (puerto & 0x10);
        b = (puerto & 0x20);
        // si puerto es = a significa que se a presionado el botón inicio pares
        if (a == 0x10)
                {
                _delay_ms (1000);
                DDRD = 0x00;  //se configura Puerto D como entrada en HZ
                PORTD = 0x00;
                DDRB = 0xff;  //se configura Puerto B como salida
                PORTB = 0x3F;  //se envía la secuencia de números pares al display
                _delay_ms (1000);
                PORTB = 0x5B;
                _delay_ms (1000);
                PORTB = 0x66;
                _delay_ms (1000);
                PORTB = 0x7D;
                _delay_ms (1000);
                PORTB = 0x7F;
                _delay_ms (1000);

          }
```

```
        // si puerto es = b significa que se a presionado el botón inicio impares
          if  (b == 0x20)
           {
                    DDRB = 0x00;    //se configura Puerto B como entrada en HZ
                    PORTB = 0x00;
                    DDRD = 0xff;  //se configura Puerto D como salida
                    PORTD = 0x06; //se envía la secuencia de números impares al display
                    _delay_ms (1000);
                    PORTD = 0x4F;
                    _delay_ms (1000);
                    PORTD = 0x6D;
                    _delay_ms (1000);
                    PORTD = 0x07;
                    _delay_ms (1000);
                    PORTD = 0x67;
                    _delay_ms (1000);

                    }
          }
}
```

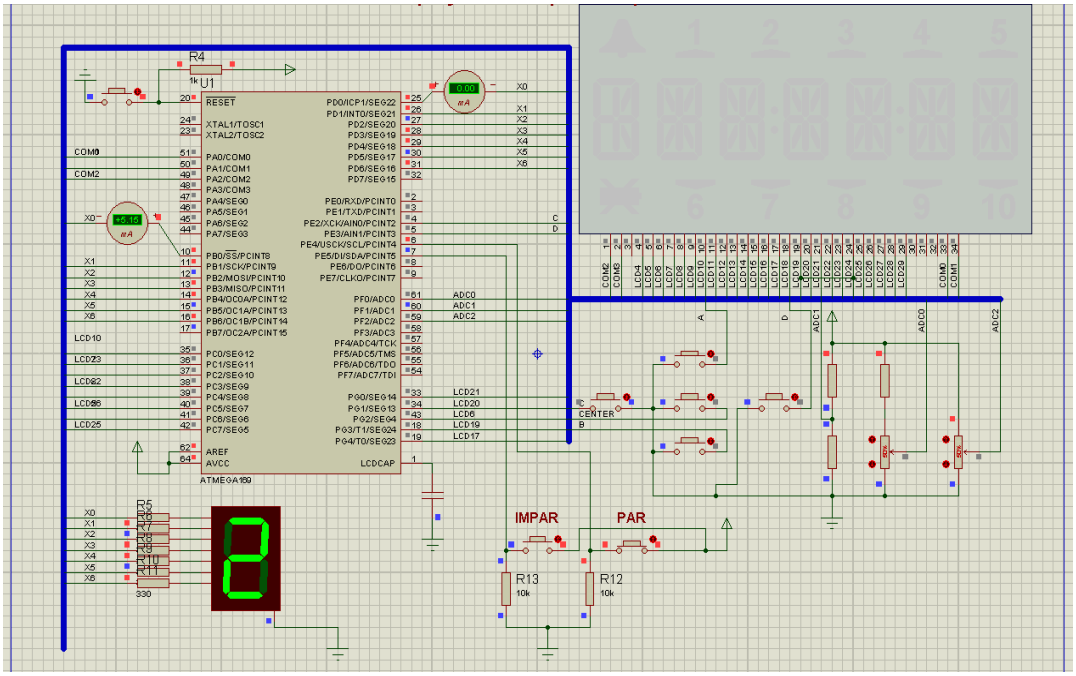
4.2.6 Simulación.

Esquema 1



El switch en el pin E4 ha sido presionado entonces el puerto D genera los números impares que pasan por el bus de datos al display mientras el puerto B permanece en alta impedancia así no podrá transmitir ni enviar datos mientras el otro puerto lo esté haciendo, la corriente que circula desde los pines al display es de aproximadamente de 5 mA.

Esquema 2



En este esquema se muestra ahora la generación de los números pares del 1 al 9 los datos son transmitidos desde el puerto B hasta el Display por medio del bus de datos mientras el puerto D permanece en alta impedancia para no generar conflictos en la transmisión de datos ya que ambos puertos están conectados al mismo bus de datos la corriente que transmite el puerto D al display es aproximadamente 5 mA mientras la

corriente transmitida por el puerto D al display es 0 esto debido a la alta impedancia que es el estado en que se encuentra el puerto D.

4.3 Desarrollo de Ejercicio 3

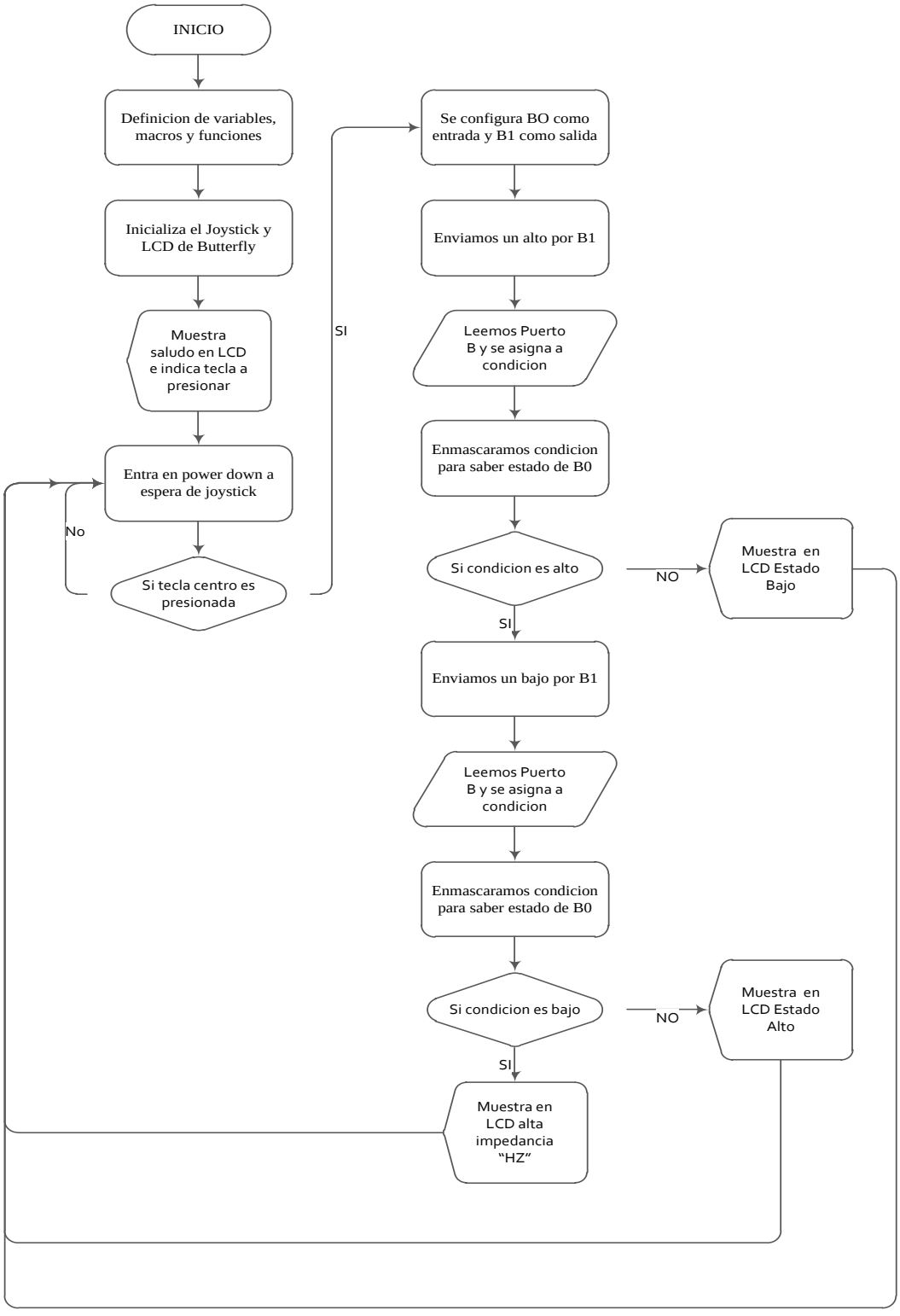
4.3.1 Tema

"**Punta Lógica**".

La Punta Lógica es una herramienta de mucha utilidad para cuando se trabaja con circuitos digitales y se quiere hacer un seguimiento de cómo están las señales en el mismo, así podemos saber si en algún punto del circuito se encuentra un estado lógico alto, bajo o en alta impedancia. El circuito trabaja de mediante dos pines los cuales están conectados entre sí por medio de una resistencia de 47 Kohm, el extremo del pin configurado como entrada es la que debemos utilizar como la punta lógica la cual está definida, luego de colocar la punta en lugar a testear presionamos en centro del joystick, para luego visualizar en el LCD el estado presente en la punta. Así cuando queramos realizar una prueba siempre lo haremos presionando el centro del joystick.

4.3.2 Diagrama de Bloques

4.3.3 Diagrama de Flujo



4.3.4 Descripción del Diagrama de Flujo.

1. Definimos variables, funciones y macros, inicializamos joystick y LCD.

2. Mostramos saludo por LCD e indicaciones de teclado.

3. Ponemos al microcontrolador en sleep power down en donde esperara hasta que se presione una tecla del joystick, que causara una interrupción y lo sacara de ese estado.

4. Cuando presiones una tecla del joystick, la evaluaremos para saber si la que fue presionada fue la tecla del centro que es la que inicia la prueba caso contrario seguirá esperando.

5. Una vez presionada la tecla del centro, comenzamos con la evaluación del estado en la punta lógica.

6. Configuramos los pines B0 como entrada y B1 como salida, estos pines están conectados entre si con una resistencia de 47 k aproximadamente, B0 sera la que sea la punta lógica ya que es la entrada, ahora colocamos un alto en B1 y leemos el estado presente en B0 si en B0 esta cero sabremos que la punta detecto un bajo si no es así, procedemos a enviar por B1 uno un bajo, y nuevamente leemos el estado de B0, si en B0 esta presente un uno confirmaríamos el estado alto en la punta, pero si esta un cero, sabremos que en la punta existe un estado de alta impedancia HZ, luego de haber analizado el ejercicio queda listo para otra prueba.

En este este ejercicio se usa las librerías para manejar LCD que vienen con el AVR Butterfly.

4.3.5 Código de Programa en lenguaje C.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 8000000UL
#include <util/delay.h>
#include "lcd_driver.h"
#include "lcd_functions.h"

/*****************************************************************
Pines del ATmega169 conectados con el Joystick:
-----------------------------------------------------------
Bit                  7 6 5 4 3 2 1 0
-----------------------------------------------------------
PORTB              B A   O
PORTE                      D C
-----------------------------------------------------------
PORTB | PORTE B A O D C => posición
-----------------------------------------------------------
*****************************************************************/


//Definiciones
//-----------------------------------------------------------
#define MASCARA_PINB ((1<<PINB7)|(1<<PINB6)|(1<<PINB4))
#define MASCARA_PINE ((1<<PINE3)|(1<<PINE2))
#define ARRIBA 0
#define ABAJO 1
#define IZQUIERDA 2
#define DERECHA 3
#define CENTRO 4
#define NO_VALIDA 5
#define posicion_A 6 //ARRIBA
#define posicion_B 7 //ABAJO
#define posicion_C 2 //IZQUIERDA
#define posicion_D 3 //DERECHA
#define posicion_O 4 //CENTRO
#define VERDADERO 1
#define FALSO 0

unsigned char SELECCION = 0;
unsigned char SELECCION_VALIDA = 0;

// Macros
#define sbi (port,bit)  (port |= (1<<bit))   //
#define cbi (port,bit)  (port &= ~(1<<bit))  //


//Funciones
//-----------------------------------------------------------
void inicializar (void);
void manejar_interrupcion (void);
```

```
void obtener_seleccion (void);
//-------------------------------------------------------


//Main
//-------------------------------------------------------
int main (void)
{
	inicializar ();
	sei ();
	LCD_Init ();
	_delay_ms (5);
	LCD_puts ("HOLA",1);
	_delay_ms (3000);
	LCD_puts ("CENTRO INICIA PRUEBA",1);
	_delay_ms (100);


	while (1)
	{
		SMCR = ((1<<SM1)|(1<<SE)); //Habilito Sleep y Power Down
	}

	return 0;
}
//-------------------------------------------------------


//Funcion Obtener seleccion
//-------------------------------------------------------
void obtener_seleccion (void)
{
	unsigned char seleccion,condicion;
	cli ();
	if (SELECCION_VALIDA)
	{
		seleccion = SELECCION;
		SELECCION_VALIDA = FALSO;
	}
	else seleccion = NO_VALIDA;
	if (seleccion != NO_VALIDA)
	{


		switch (seleccion)
		{
		//-----------------------------------------------------------------
			case ARRIBA:
				_delay_ms(5);
			break;
		//-----------------------------------------------------------------
			case ABAJO:
				_delay_ms(1);
```

```
                LCD_puts("GRUPO 1 MARLON - ARTURO",1);
                _delay_ms(15);
        break;
//-------------------------------------------------------------------
        case DERECHA:
                _delay_ms(8);
        break;

//-------------------------------------------------------------------
        case IZQUIERDA:
                _delay_ms(10);
        break;
//-------------------------------------------------------------------
        case CENTRO:
        cbi (PORTE,4);
        cbi (DDRE,4);
        sbi (PORTE,5);
        cbi (DDRE,5);
        _delay_ms(500);
        cbi (DDRB,0);
        cbi (PORTB,0);
        sbi (DDRB,1);
        sbi (PORTB,1);
        _delay_ms (500);

                sbi (PORTB,1);
                condicion = PINB;
                condicion &= 1;
                if (condicion == 1)
                        {
                        _delay_ms (10);
                        cbi (PORTB,0);
                        _delay_ms (10);
                        cbi (PORTB,1);
                        _delay_ms (10);
                        condicion = PINB;
                        condicion &= 1;
                        if (condicion == 0)
                                {
                                goto HZ;
                                }
                                goto HIGH;
                        }
                LCD_puts("BAJO",1);
                break;

        HZ:
                LCD_puts("HZ",1);
                break;

        HIGH:
                LCD_puts("ALTO",1);
                break;
```

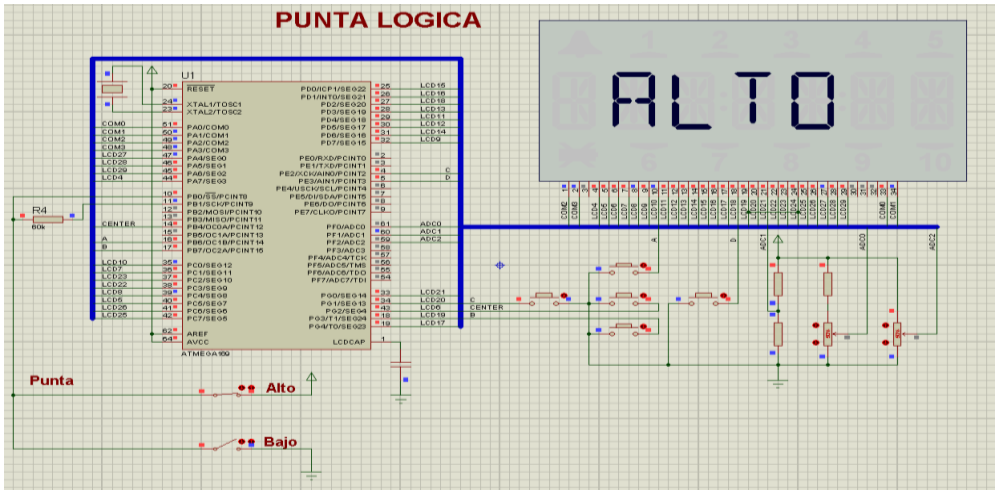
```
                        //-------------------------------------------------------------------------
                                default:
                                break;
                        }
                }
                sei();
}


//Funcion Inicializar
//-----------------------------------------------------------
void inicializar(void)
{
        CLKPR = (1<<CLKPCE);  //Habilito configuración de reloj
        CLKPR = (0<<CLKPS1) | (0<<CLKPS0);  //configuro reloj a 8 MHz
        DDRB |= 0xD0;
        PORTB |= MASCARA_PINB;
        DDRE |= 0x0C;
        PORTE |= MASCARA_PINE;
        DDRB = 0;//entrada
        PORTB = MASCARA_PINB;//habilitar PULL-UPs
        DDRE = 0;//entrada
        PORTE = MASCARA_PINE;//habilitar PULL-UPs
        PCMSK1 |= MASCARA_PINB;
        PCMSK0 |= MASCARA_PINE;
        EIFR = ((1<<PCIF1)|(1<<PCIF0));
        EIMSK = ((1<<PCIE1)|(1<<PCIE0));
}
//-------------------------------------------------------------

//Funcion Para Interrupcion
//-------------------------------------------------------------
void manejar_interrupcion(void)
{
        unsigned char joystick;
        unsigned char seleccion;
        joystick = ((~PINB) & MASCARA_PINB);
        joystick |= ((~PINE) & MASCARA_PINE);
        if((joystick & (1<<posicion_A)))
                seleccion = ARRIBA;
        else if((joystick & (1<<posicion_B)))
                seleccion = ABAJO;
        else if((joystick & (1<<posicion_C)))
                seleccion = IZQUIERDA;
        else if((joystick & (1<<posicion_D)))
                seleccion = DERECHA;
        else if((joystick & (1<<posicion_O)))
                seleccion = CENTRO;
        else seleccion = NO_VALIDA;
        if(seleccion != NO_VALIDA)
        {
                if(!SELECCION_VALIDA)
                {
```

```
                        SELECCION = seleccion;
                        SELECCION_VALIDA = VERDADERO;
                }
        }
        EIFR = ((1<<PCIF1)|(1<<PCIF0));
        obtener_seleccion();
}
//------------------------------------------------------------


SIGNAL(SIG_PIN_CHANGE0)
{
manejar_interrupcion();
}

SIGNAL(SIG_PIN_CHANGE1)
{
manejar_interrupcion();
}
```

Las librerías que nos permiten manejar el LCD del AVR Butterfly las podemos descargar de la pagina web de Atmel, en la sección de búsqueda colocaremos Butterfly nos aparecerá una lista de temas encontrados, escogeremos el primero y nos enviara a la descripción del Kit AVR Butterfly en la parte de overview y hacia el final de la pagina encontraremos la ultima versión de las aplicaciones de ejemplo del Butterfly una vez descargado, allí encontraremos 4 ficheros necesarios que son lcd_driver.h, lcd_functions.h, lcd_driver.c y lcd_functions.c debemos copiar estos ficheros en la carpeta de nuestro proyecto donde se encuentre nuestro fichero main, luego debemos adicionar los ficheros ".c" a nuestro proyecto como archivos fuente y en cambio incluir los fichero ".h" en el encabezado del programa como en el ejercicio anterior, de esta forma podremos utilizar las funciones que nos permiten enviar y visualizar mensajes en la pantalla LDC del AVR Butterfly.

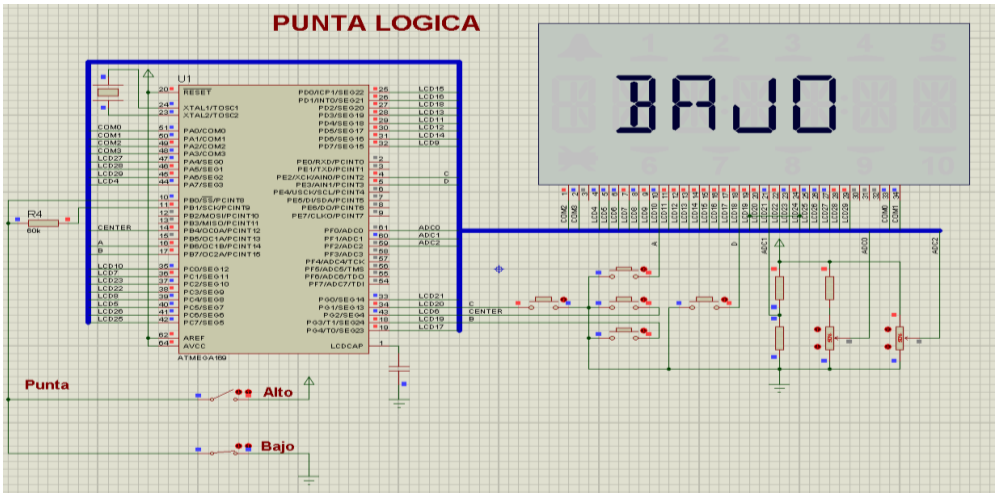
4.3.6 Simulación.

Esquema 1



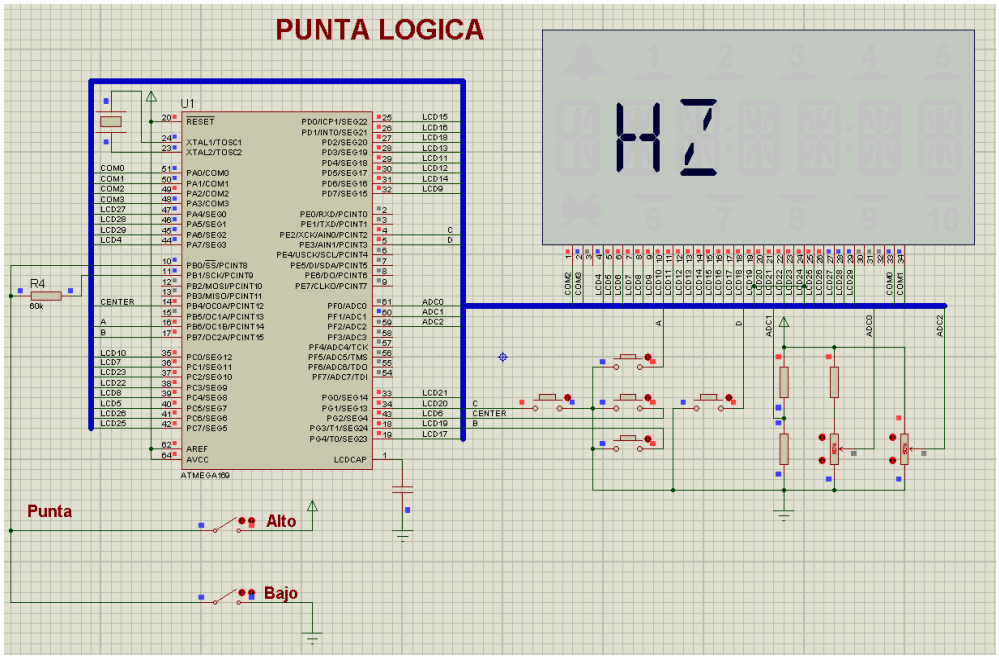
En este esquema observamos como la punta lógica detecta un estado alto en la misma ya que el interruptor hacia Vcc se encuentra cerrado confirmando dicho estado.

Esquema 2



En este esquema podemos ver como a detectado el estado bajo.

Esquema 3



Aquí podemos observar la detección de Hz en la punta a no estar conectado a nada.

4.4 Desarrollo de Ejercicio 4

4.4.1 Tema

"**Menú de Entradas, Salidas y Alta Impedancia**".

En este ejercicio de realizo un pequeño menú en LCD y por medio del joystick podemos elegir una de las opciones que son:

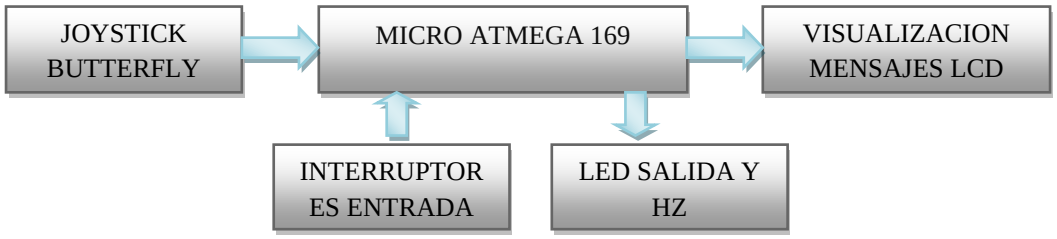
Joystick hacia arriba configuramos los pines BO y B1 como salidas y podemos encender y apagar dos leds conectados a los mismos cada vez que presionamos hacia arriba.

Joystick hacia la derecha configuramos los mismos pines B0 y B1 ahora como entradas y se conectan a dos interruptores los cuales podemos manipular para visualizar en el LCD el estado lógico en el que se encuentran dichos interruptores.

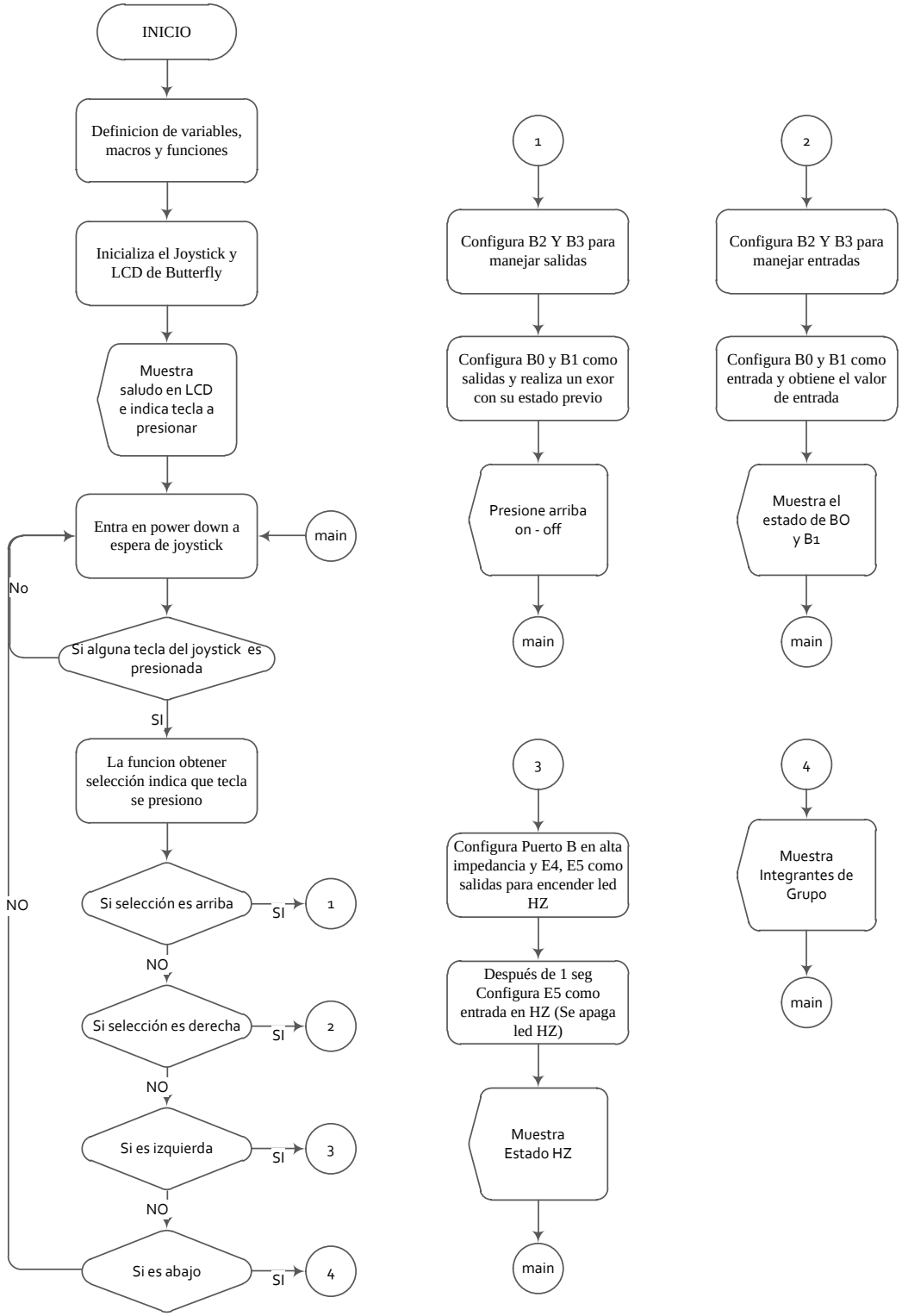
Joystick hacia la izquierda aquí observamos que se enciende un led que está conectado en un pin el cual después de 1 segundo pasara a estar en alta impedancia por lo que dicho led se apagara y mostraremos en el LCD que ahora se encuentra en Alta Impedancia.

Para la parte de configuración de entrada y salida se utilizó un buffer de tres estados conectados entre los pines B0 y B1 hacia las circuiterías de entrada (interruptores) y salida (Leds) ya que si no se hacía esto se interferían entre si por estar conectados a los mismos pines del Atmega169. Todo esto se aprecia en la simulación.

4.4.2 Diagrama de Bloques



4.4.3 Diagrama de Flujo



4.4.4 Descripción del Diagrama de Flujo.

1. Definimos variables, funciones y macros, inicializamos joystick y LCD.

2. Mostramos saludo por LCD e indicaciones de teclado.

3. Ponemos al microcontrolador en sleep power down en donde esperara hasta que se presione una tecla del joystick, que causara una interrupción y lo sacara de ese estado.

4. Cuando se presiona una tecla del joystick el microcontrolador recibirá una interrupción, la cual activara una función que verificara que tecla ha sido presionada.

5. Si se presiono la tecla arriba entrara en una subrutina en la que configuraremos los pines B0 y B1 como salidas y los pines B2 y B3 se configuran para manejar un buffer triestado externo que habilitara los led hacia B0 y B1, ahora cada vez que presiones hacia arriba en microcontrolador encenderá o apagara los led conectados hacia el.

6. Si se presiona la tecla derecha se configurara BO y B1 como entradas y además se reconfigura B2 y B3 para que el buffer externo conecte los interruptores en las entradas BO y B1, luego leeremos el estado lógico en estos dos pines y mostraremos el resultado en el LCD.

7. Si presionamos la tecla izquierda por medio de dos pines E4 como salida en alto y E5 como salida en bajo encenderemos un led conectado entre ellos, que luego de 1 segundo se apagara ya que cambiaremos el estado de E5 a entrada en Hz y mostraremos en el LCD el mensaje de “HZ”.

8. La tecla hacia abajo mostrara en el LCD el mensaje del numero de grupo y los integrantes de este proyecto.

4.4.5 Código de Programa.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 8000000UL
#include <util/delay.h>
#include  “lcd_driver.h”
#include  “lcd_functions.h”

//este ejercicio ejecuta las mismas funciones para manejo de lcd y obtención de tecla presionada
//que el ejercicio anterior
/*********************************************************************
Pines del ATmega169 conectados con el Joystick:
-------------------------------------------------------------
Bit                  7 6 5 4 3 2 1 0
-------------------------------------------------------------
PORTB             B A   O
PORTE                     D C
-------------------------------------------------------------
PORTB | PORTE B A O D C => posición
-------------------------------------------------------------
*********************************************************************/

//Definiciones
//-------------------------------------------------------------
#define MASCARA_PINB ((1<<PINB7)|(1<<PINB6)|(1<<PINB4))
#define MASCARA_PINE ((1<<PINE3)|(1<<PINE2))
#define ARRIBA 0
#define ABAJO 1
#define IZQUIERDA 2
#define DERECHA 3
#define CENTRO 4
#define NO_VALIDA 5
#define posicion_A 6 //ARRIBA
#define posicion_B 7 //ABAJO
#define posicion_C 2 //IZQUIERDA
#define posicion_D 3 //DERECHA
#define posicion_O 4 //CENTRO
#define VERDADERO 1
#define FALSO 0

unsigned char SELECCION = 0;
unsigned char SELECCION_VALIDA = 0;
```

```
// Macro definitions
#define sbi (port,bit)  (port |= (1<<bit))   //
#define cbi (port,bit)  (port &= ~(1<<bit))  //


//Funciones
//-------------------------------------------------------------
void inicializar (void);
void manejar_interrupcion (void);
void obtener_seleccion (void);
//-------------------------------------------------------------


//Main
//-------------------------------------------------------------
int main (void)
{

        inicializar ();
        sei();
        LCD_Init ();
        _delay_ms (1);
        LCD_puts ("HOLA",1);
        _delay_ms (1000);
        LCD_puts ("ARRIBA PUERTO OUT",1);
        _delay_ms (5000);
        LCD_puts ("DERECHA PUERTO IN",1);
        _delay_ms (5000);
        LCD_puts ("IZQUIERDA PUERTO HZ",1);
        _delay_ms (5000);
        LCD_puts ("ABAJO INTEGRANTES",1);
        _delay_ms (4000);
        LCD_puts ("CENTRO MAIN",1);
        _delay_ms (3000);
        LCD_puts ("MUEVA EL JOYSTICK",1);
        _delay_ms (10);

        while (1)
        {
                SMCR = ((1<<SM1)|(1<<SE)); //Habilito Sleep y Power Down
        }

        return 0;
}
//-------------------------------------------------------------


//Funcion Obtener seleccion
//-------------------------------------------------------------
void obtener_seleccion (void)
{
        unsigned char seleccion,KEY;
```

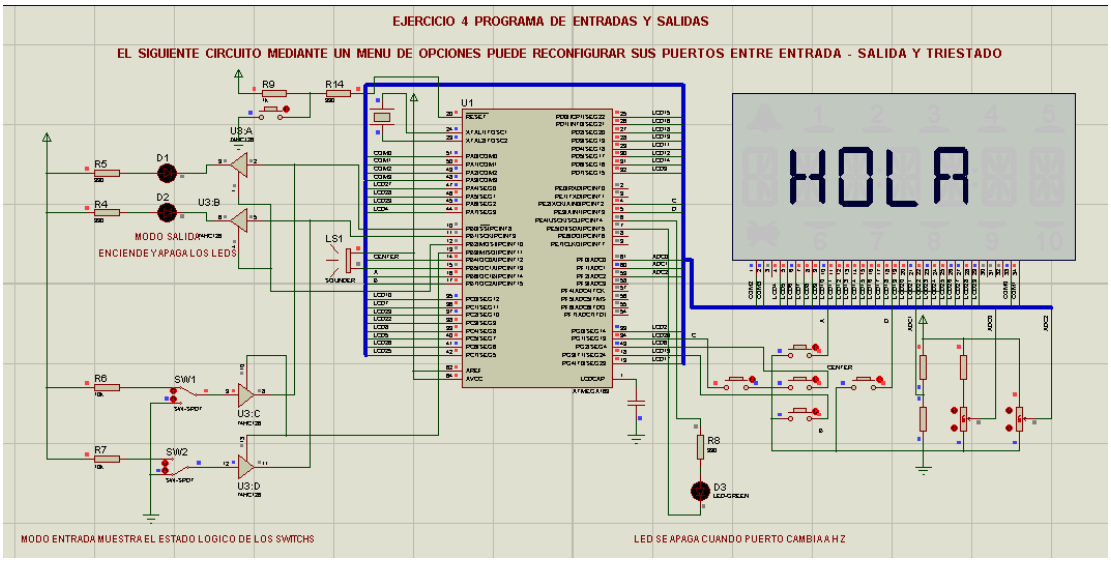
```
cli();
if (SELECCION_VALIDA)
{
        seleccion = SELECCION;
        SELECCION_VALIDA = FALSO;
}
else seleccion = NO_VALIDA;
if (seleccion != NO_VALIDA)
{
        switch (seleccion)
        {//codigo para encender y apagar leds opcion como salidas
        //-------------------------------------------------------------------
                case ARRIBA:
                        cbi (PORTB,3);
                        _delay_ms(1);
                        cbi (PORTB,2);
                        _delay_ms(1);
                        DDRB=0x0f;
                        _delay_ms(1);
                        sbi (PORTB,2);
                        _delay_ms(1);
                        PORTB ^=(1<<PB0);
                        PORTB ^=(1<<PB1);
                        LCD_puts("ARRIBA ON OFF",1);
                        _delay_ms(5);
                break;
        //-------------------------------------------------------------------
                case ABAJO: //muestra  integrantes del grupo en la LCD
                        cbi (PORTB,3);
                        _delay_ms(1);
                        cbi (PORTB,2);
                        _delay_ms(1);
                        cbi (PORTB,0);
                        _delay_ms(1);
                        cbi (PORTB,1);
                        _delay_ms(1);
                        LCD_puts("GRUPO 1 MARLON - ARTURO",1);
                        _delay_ms(15);
                break;
        //-------------------------------------------------------------------
                case DERECHA:
                        //codigo que configura Puerto como entrada y muestra el estado
                        //logico del puerto en la LCD
                        cbi (PORTB,3);
                        _delay_ms(1);
                        cbi (PORTB,2);
                        _delay_ms(1);
                        cbi (DDRB,0);
                        _delay_ms(1);
                        cbi (DDRB,1);
                        _delay_ms(1);
                        cbi (PORTB,0);
                        _delay_ms(1);
```

```
                cbi (PORTB,1);
                _delay_ms(1);
                sbi (PORTB,3);
                _delay_ms(5);

                KEY = PINB;
                _delay_ms(5);
                KEY &= 0x03;
                _delay_ms(5);
                if (KEY == 0x03)
                        {
                        LCD_puts ("ES 11",1);
                        _delay_ms(15);
                        }
                else if (KEY == 0x01)
                                {
                                LCD_puts ("ES 01",1);
                                _delay_ms(15);
                                }
                        else if (KEY == 0x02)
                                        {
                                        LCD_puts ("ES 10",1);
                                        _delay_ms(15);
                                        }
                                else if (KEY == 0x00)
                                                {
                                                LCD_puts ("ES 00",1);
                                                _delay_ms(15);
                                                }
        break;

//---------------------------------------------------------------------
        case IZQUIERDA: //rutina para apagar led por cambio en Puerto a HZ
                cbi (PORTB,3);
                _delay_ms(1);
                cbi (PORTB,2);
                _delay_ms(1);
                cbi (PORTB,0);
                _delay_ms(1);
                cbi (PORTB,1);
                _delay_ms(1);
                DDRE = 0x30;
                sbi (PORTE,4);
                _delay_ms(5);
                cbi (PORTE,5);
                _delay_ms(2000);
                cbi (DDRE,5);
                _delay_ms(1);
                cbi (PORTE,5);
                _delay_ms(1);
                LCD_puts("High Z");
                _delay_ms(10);
        break;
```

```
                //-------------------------------------------------------------------
                        case CENTRO:
                                main(); //vuelve a main

                        break;
                //-------------------------------------------------------------------
                        default:
                        break;
                }
        }
        sei();
}

//Funcion Inicializar
//-----------------------------------------------------------
void inicializar(void)
{
        CLKPR = (1<<CLKPCE);  //Habilito configuración de reloj
        CLKPR = (0<<CLKPS1) | (0<<CLKPS0);  //configuro reloj a 8 MHz
        DDRB |= 0xD0;
        PORTB |= MASCARA_PINB;
        DDRE |= 0x0C;
        PORTE |= MASCARA_PINE;
        DDRB = 0;//entrada
        PORTB = MASCARA_PINB;//habilitar PULL-UPs
        DDRE = 0;//entrada
        PORTE = MASCARA_PINE;//habilitar PULL-UPs
        PCMSK1 |= MASCARA_PINB;
        PCMSK0 |= MASCARA_PINE;
        EIFR = ((1<<PCIF1)|(1<<PCIF0));
        EIMSK = ((1<<PCIE1)|(1<<PCIE0));
}
//-----------------------------------------------------------

//Funcion Para Interrupcion
void manejar_interrupcion (void)
{
        unsigned char joystick;
        unsigned char seleccion;
        joystick = ((~PINB) & MASCARA_PINB);
        joystick |= ((~PINE) & MASCARA_PINE);
        if ((joystick & (1<<posicion_A)))
                seleccion = ARRIBA;
        else if ((joystick & (1<<posicion_B)))
                seleccion = ABAJO;
        else if ((joystick & (1<<posicion_C)))
                seleccion = IZQUIERDA;
        else if ((joystick & (1<<posicion_D)))
                seleccion = DERECHA;
        else if ((joystick & (1<<posicion_O)))
                seleccion = CENTRO;
        else seleccion = NO_VALIDA;
        if (seleccion != NO_VALIDA)
```

```
        {
                if (!SELECCION_VALIDA)
                {
                        SELECCION = seleccion;
                        SELECCION_VALIDA = VERDADERO;
                }
        }
        EIFR = ((1<<PCIF1)|(1<<PCIF0));
        obtener_seleccion();
}
//funciones que actúan cuando se presenta la interrupción ya sea en E0 o E1
//y envían el programa a identificar tecla presionada mediante manejar_interrupcion

SIGNAL(SIG_PIN_CHANGE0)
{
manejar_interrupcion();
}

SIGNAL(SIG_PIN_CHANGE1)
{
manejar_interrupcion();
}
```

4.4.6 Simulación.



En este esquema podemos apreciar el inicio del programa con su respectivo saludo, luego del cual podemos presionar una de las teclas del joystick para hacer una de la

tareas, si presionamos hacia arriba encenderemos el par de led rojos si volvemos a presionar los apagamos esta es la opción de puerto como salida, si presionamos derecha veremos en pantalla el estado lógico de los switch, esta es la opción de puerto como entrada, si presionamos izquierda se encenderá el led rojo el cual después de 2 segundos se apagara debido a que el pin al cual esta conectado cambia a HZ, esta es la opción HZ y finalmente si presionamos Abajo se mostrara en el LCD los nombres de los integrantes del grupo.

4.5 Desarrollo de Ejercicio 5

4.5.1 Tema

"**Juego de Reacción**".

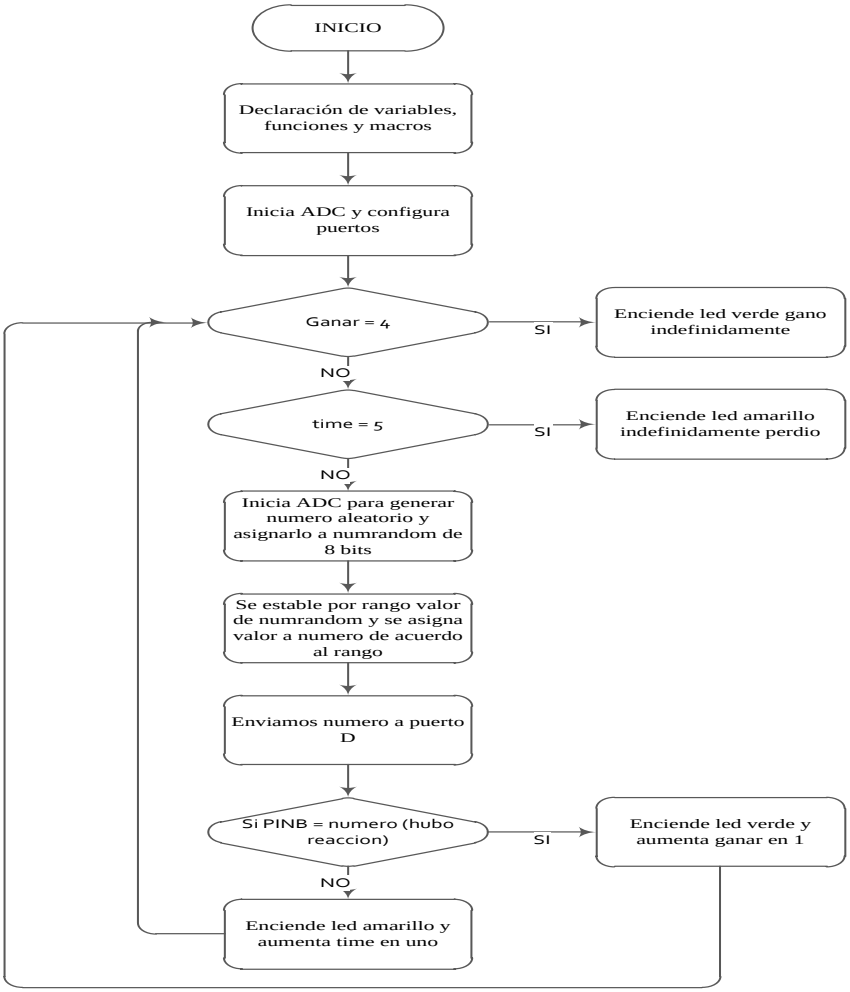
En este ejercicio podemos ver cómo utilizar tanto entradas como salidas para implementar un divertido juego.

En este ejercicio tenemos ocho Leds los cuales al iniciar el juego ya sea enciendo la plataforma o reseteando el Butterfly encenderá un Led aleatorio y deberemos antes de un tiempo presionar el Push botton correspondiente al Led que encendió si no lo hacemos se encenderá un Led Amarrillo indicando que hemos fallado, si lo hacemos bien encenderá un Led verde indicando que lo hemos hecho bien, se tienen 5 oportunidades caso contrario se pierde el juego permaneciendo el led amarillo

encendido, se gana acertando 4 veces y en ese caso permanecerá encendido el led verde.

4.5.2 Diagrama de Bloques

4.5.3 Diagrama de Flujo



4.5.4 Descripción del Diagrama de Flujo

1. Declaramos Variables, funciones y macros a utilizar.

2. Ya dentro de “main” iniciamos el ADC y configuramos el Puerto B como entrada, el Puerto D y E como salidas.

3. Lo primero que hace la aplicación es preguntar por el estado de las variable ganar y time que nos indicara si el juego continua o a terminado, ya que si se ha ganado o acertado 4 veces la variable ganar hará que el juego termine con victoria encendiendo un led verde indefinidamente hasta presionar reset para jugar nuevamente, o caso contrario si la variable time llega a 5 antes que ganar el juego terminara con perdida encendiendo un led amarillo indefinidamente hasta darle un reset pata jugar nuevamente.

4. Si las variables no están en su tope el juego sigue, y lo hace primero encontrando un número aleatorio para comenzar a jugar, lo hace por medio del ADC que coge una muestra de voltaje externo en el pin F1, con esta muestra podemos asignar un valor a la variable numero que será la que se envié al puerto D encendiendo un led.

5. El usuario deberá presionar el Push botton correspondiente al led encendido en un periodo de tiempo establecido para ganar caso contario será un error.

6. Este proceso continua hasta que una de las variables ganar en caso de acertar 4 veces o perder cuando se ha errado 5 veces alcance estos valores y veamos que hemos perdido o ganado el juego.

4.5.5 Código de Programa en lenguaje C.

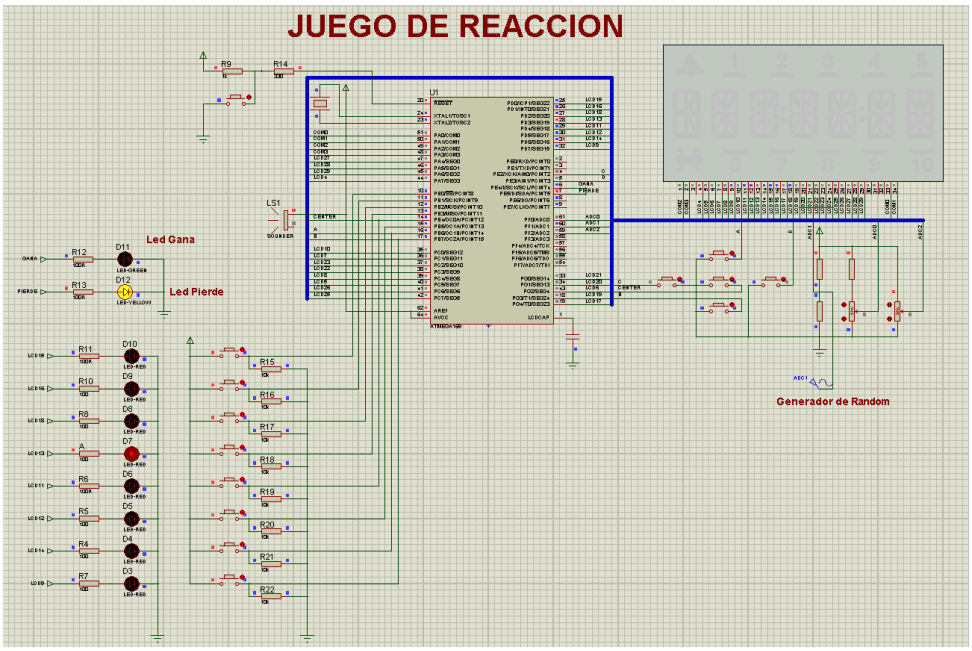
```
#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 8000000UL
#include <util/delay.h>

// Macros
#define sbi(port,bit)  (port |= (1<<bit))   //
#define cbi(port,bit)  (port &= ~(1<<bit))  //

void inicia_adc(void);

unsigned char numrandom, numero, adchigh;
int time = 0;
int ganar = 0;

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

int main (void)
{

CLKPR = (1<<CLKPCE);  //Habilito configuración de reloj
CLKPR = (0<<CLKPS1) | (0<<CLKPS0);  //configuro reloj a 8 MHz

sbi (PORTF, PORTF3); //configuro PORTF3 como salida en alto
sbi (DDRF, PORTF3);  //para alimentar en ADC

_delay_ms (3000);

inicia_adc ();

DDRB  = 0X00;  //puerto b como entrada
PORTB = 0XFF; //inicio del puerto b
DDRD  = 0XFF;  //puerto d como salida
PORTD = 0X00; //inicio del puerto b
DDRE  = 0XF0;  //puerto e como salida
PORTE = 0X00;  //inicio del puerto e

while (1)
        {
        INICIO:
        //-------------------------------------------------------------------------
        if (ganar == 4)     //si las oportunidades se acaban ?
        {
        ganar =0;           //se encera ganar
                while (1)
                {
                PORTE=0X10; //lazo infinito denota que gano y se enciende
                }
        }
        if (time ==5)       //si las oportunidades se acaban ?
```

```
{
time =0;  //se encera time
while (1)
        {
        PORTE=0X20; //lazo infinito denota que perdio y se enciende
        }                       //el led rojo indefinido

}

//-------------------------------------------------------------------------

_delay_ms(1000);

ADCSRA |= (1<<ADSC); //inicio captura ADC

if (ADSC == 1) //espero hasta que finalice la captura ADC
_delay_ms(1);

numrandom = ADCL; //asigno la parte menos significativa de la captura ADC
adchigh = ADCH;  //a numrandom

// asigno según numero capturado led a encender para el juego

if ((numrandom>=0x01)&&(numrandom <=0x20))
        {
        numero = 0x01;
        goto JUEGO;
        }
if ((numrandom>=0x21)&&(numrandom <=0x40))
        {
        numero = 0x02;
        goto JUEGO;
        }
if ((numrandom>=0x41)&&(numrandom <=0x62))
        {
        numero = 0x04;
        goto JUEGO;
        }
if ((numrandom>=0x63)&&(numrandom <=0x82))
        {
        numero = 0x08;
        goto JUEGO;
        }
if ((numrandom>=0x83)&&(numrandom <=0xA2))
        {
        numero = 0x10;
        goto JUEGO;
        }
if ((numrandom>=0xA3)&&(numrandom <=0xC4))
        {
        numero = 0x20;
        goto JUEGO;
        }
```

```
        if ((numrandom>=0xC5)&&(numrandom <=0xE4))
                {
                numero = 0x40;
                goto JUEGO;
                }
        if ((numrandom>=0xE5)&&(numrandom <=0xFF))
                {
                numero = 0x80;
                goto JUEGO;
                }
        goto INICIO;
JUEGO:

        PORTD=numero; //enciende led de reaccion
        _delay_ms (300);

        if (PINB == numero) //pregunta si hubo reaccion del usuario
        {
        PORTE=0x10; //entonces enciende led verde
        _delay_ms (500);
        PORTE=0x00;
        PORTD=0x00;
        ganar = ganar + 1;
        goto INICIO; //regresa
        }
        _delay_ms (1000);
        if(PINB==0x00)  //si el boton no alcanza a ser presionado
        {
        PORTE=0x20;    //enciende led amarillo de advertencia
        _delay_ms (500);
        PORTE=0x00;
        PORTD=0x00;
        time = time + 1; //decrece el numero de oportunidades
        goto INICIO;
        }
        //-------------------------------------------------------------------------
        }
}


//-------------------------------------------------------------
void inicia_adc (void)
{

//AVCC externo y canal 1

ADMUX|=(0<<REFS1)|(0<<REFS0)|(1<<MUX0);

//Habilitar ADC y Reloj 8Mhz/64 125 Khz

ADCSRA =(1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(0<<ADPS0);

}
```

4.5.6 Simulación.

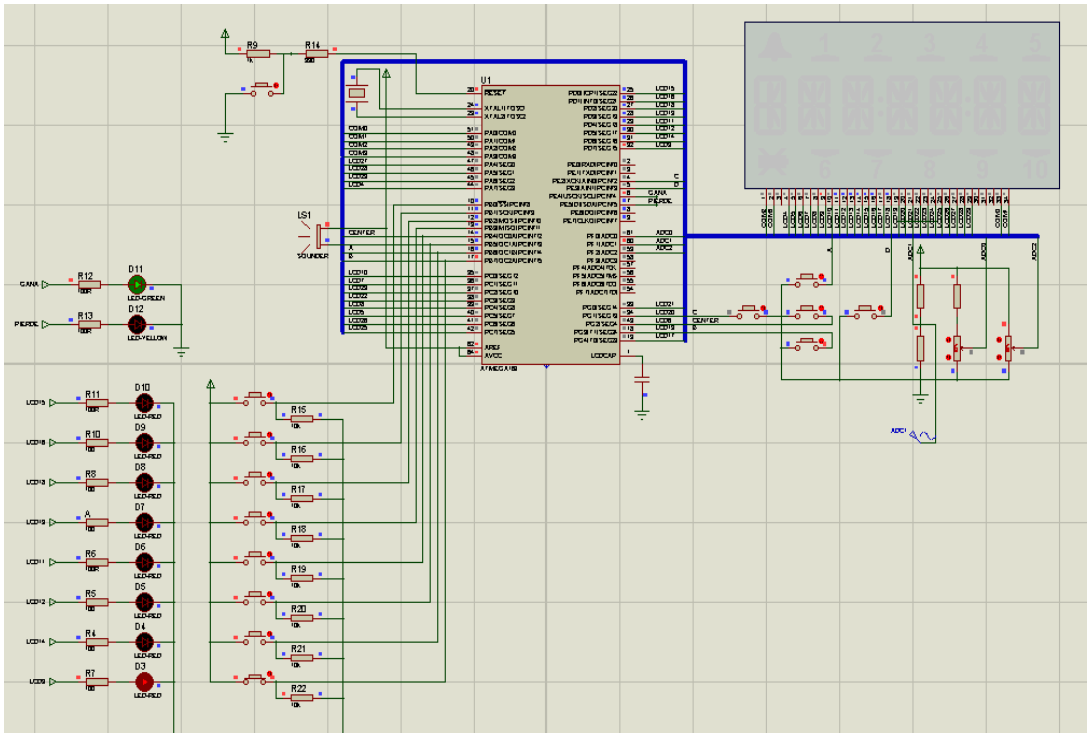
Esquema 1



Aquí vemos el esquema del juego de reacción, el juego funciona encendiendo un led cualquiera (aleatorio) el usuario debe presionar el botón correspondiente lo más rápido posible ya que de no hacerlo se encenderá un led amarillo tiene 5 oportunidades caso contrario pierde si acierta se enciende el led verde, con 4 aciertos gana, si acertaste las 4 veces se encenderá el led verde indefinidamente indicando que has ganado el juego, si en caso contrario has errado las 5 veces se encenderá el led amarillo indefinidamente indicando que has perdido el juego, el juego comenzara nuevamente presionando reset.

En este esquema vemos que el usuario a errado por lo que se enciende el Led amarrillo.

Esquema 2



Aquí vemos como se enciende el Led verde cuando el jugador acertado presionado al Push botton a tiempo en concordancia son el led rojo encendido aleatoriamente.

CONCLUSIONES

1. Con el presente proyecto hemos mostrado como de manera simple y sin necesitar mucho tiempo, se puede implementar un sistema robusto de aprendizaje y prueba de buenas prestaciones, gracias a la utilización del Kit AVR Butterfly de Atmel, ya que este posee muchas características que nos facilitan su programación y utilización y en el cual podemos implementar un sin número de ejercicios limitados únicamente por la imaginación de quien los diseñe.
2. Además hemos comprobado las excelentes características de los microcontroladores Atmel, en este caso el ATmega169 que viene en el Kit AVR Butterfly que hemos utilizado, siendo este únicamente una muestra de lo es la familia completa de microcontroladores de Atmel, permitiéndonos de esta manera expandir nuestros conocimientos hacia otras arquitecturas de microcontroladores, y así tener un mayor rango de criterios a la hora de escoger un dispositivo o integrarlos según las especificaciones que se necesiten.
3. Hemos podido analizar tres diferentes arquitectura de puertos de Entrada y Salida desde la más básica con Intel, pasando por la más comercial como Microchip, hasta otra con mayor complejidad pero más versátil como Atmel. De esta forma adquirimos mayores conocimientos para trabajar con uno u otro fabricante según las necesidades de nuestro trabajo o proyecto.

RECOMENDACIONES

1. Debemos tener cuidado cuando conectamos cargas a los puertos de salida del Atmega169, y no exceder la capacidad de corriente de cada puerto, que ocasionaría el daño de dicho puerto o el microcontrolador.
2. Debemos tener siempre en cuenta que el AVR Butterfly ocupa los puertos B y D cuando utilizamos el joystick y la LCD para no implementar circuitería en dichos pines cuando estamos utilizando dichas herramientas del Kit.
3. Cuando se programa el AVR Butterfly a través del puerto serie y con el AVR Studio 4, en ocasiones el AVRProg no se conecta, entonces lo que debemos hacer es dar clic en aceptar en el mensaje de error de no conexión y luego sin presionar el joystick en al AVR Butterfly ejecutar nuevamente el AVRProg, veremos que la aplicación comenzara con normalidad.
4. Debemos tener siempre en cuenta al programar nuestras aplicaciones que el espacio de memoria a utilizar deberá ser el total de la flash menos el espacio del Bootloader que se encuentra programado en el microcontrolador.
5. Es recomendable manipular con mucho cuidado el Kit AVR Butterfly ya que es de un tamaño reducido, por lo que es fácil que se nos pueda escapar de las manos y por lo tanto estropearse.

BIBLIOGRAFIA

1) Gonzales Vásquez José Adolfo, Introducción a los Microcontroladores 8X52 8X51, 1° Edición McGraw-Hill, 1992, paginas 1-9, 42-58, 130-149.
2) Angulo Usátegui José María, Microcontroladores PIC, 3º Edición, McGraw-Hill, 2003, paginas 15-21, 101-104.
3) PICmicro Mid-Range MCU Family Reference Manual, Data Sheet Microchip, Diciembre 1997, sección 9 I/O Ports.
4) Dhananjay V. Gadre, Programming and Customizing the AVR Microcontroller, 1°Edición MacGraw-Hill, 2001,paginas 1-93.
5) 8-bit AVR Microcontroller ATmega 169, DataSheet, revisión 2010, secciones 13 y 22.
6) Pardue Joe, C Programming for Microcontrollers, 2005, paginas 1-295.