UNIVERSIDAD DE BELGRANO

Las tesinas de Belgrano

Facultad de Ingeniería y Tecnología Informática
Carrera de Ingeniería Electrónica

Termómetro Digital con Control de Temperatura

Nº 137 **Carlos Girola**

Tutor: Aníbal Intino

Departamento de Investigación
Febrero 2005

Indice

1. Introducción 5
1.1. Selección del tema del trabajo final 5
1.2. Objetivos 5
2. Descripción general del proyecto 5
2.1. Diagrama en bloques 5
2.2. Funcionamiento básico del dispositivo 6
3. Selección de componentes 6
4. Descripción detallada de los bloques 6
4.1. Fuente regulada de tension 6
4.2. Microcontrolador 9
4.3. Display 12
4.4. Sensor de temperatura 14
4.5. Control externo 16
5. Funcionamiento del software 16
5.1. Rutina principal (Main) 16
5.2. Display 17
5.3. Sensor de temperatura (1-Wire) 17
5.4. Parser (Diagrama de estados) 17
6. Aplicaciones 20
7. Posibilidades futuras 20
8. Esquemas de hardware 21
9. Conclusiones 23
10. Contenido del CD 23
11. Bibliografia 24
Anexo: Funciones principales del código fuente 25

1. Introducción:

1.1. Selección del tema del trabajo final:

En la elección del tema del trabajo final se tuvieron en cuenta varios factores. El más importante fue el interés por profundizar los conocimientos adquiridos en la materia Microprocesadores, del 5º año de la carrera, en la cual se desarrollaron diversos aspectos del funcionamiento del microcontrolador 8051 de la familia Intel. Así fue que en este proyecto se decidió utilizar un microcontrolador de la misma familia pero con mayores prestaciones.

En el Instituto Nacional de Tecnología Industrial (INTI) surgió la posibilidad de implementar una nueva tecnología, con la cual todavía no se había trabajado hasta ese momento, denominada 1-Wire.

Uno de los dispositivos de la tecnología 1-Wire que se encontraba disponible era un sensor de temperatura que posee un conversor A/D incorporado, y resultó muy interesante poder implementar esa tecnología nueva para el instituto.

La tecnología 1-Wire está basada en que con solo un cable se puede comunicar el dispositivo con el microcontrolador. A su vez, la alimentación también puede ser tomada de ese cable, por lo que además sólo se requiere conectar el dispositivo a masa para su funcionamiento.

Para comprender e implementar la tecnología 1-Wire, se utilizó el sensor de temperatura digital que se encontraba disponible. Digital, porque el mismo se encarga de realizar la conversión de temperatura y de entregar el valor discreto por el cable, listo para ser usado en el microcontrolador.

Con estos componentes en mente, se decidió realizar un termómetro digital con control de temperatura, para darle una utilidad práctica al proyecto además de solamente medir temperatura.

1.2. Objetivos:

Como fue mencionado con anterioridad, el objetivo principal del proyecto es profundizar los conocimientos sobre los microcontroladores Intel de la familia 8051, como así también su conexión con periféricos externos. Por este motivo, y por la necesidad de mostrar la temperatura medida y los limites seleccionados para realizar el control externo, el proyecto necesitará disponer de un display, y una botonera para poder navegar los menues y realizar las selecciones de los valores.

Otra meta importante es lograr comprender el funcionamiento de la tecnología 1-Wire, para esto se cuenta con diversas notas de aplicación, que seguramente facilitarán la implementación del sensor y su interconexión con el resto de los componentes.

En resumen, lo que se pretende lograr es realizar un proyecto con un microcontrolador que abarque desde la fuente de alimentación, hasta la interfaz con el usuario, pasando por la realización de una plaqueta prototipo, la conexión de todos los periféricos necesarios y el software para cargar en el microcontrolador.

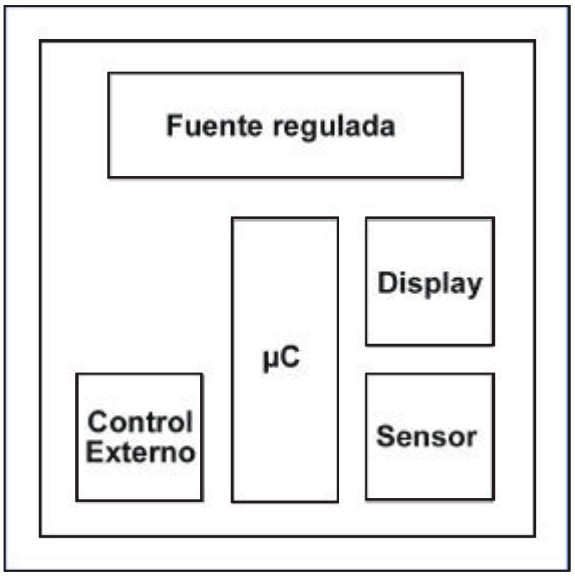
2. Descripción general del proyecto:

2.1. Diagrama en bloques

El dispositivo está compuesto por cinco grandes bloques, que pueden verse en la figura siguiente. Estos son:

- Fuente de tensión
- Microcontrolador
- Display
- Sensor de temperatura

  Control externo



2.2. Funcionamiento básico del dispositivo:

El termómetro digital cuenta con una fuente de tensión regulada de 5V, la cual se necesita para alimentar el microcontrolador, el display, y el sensor de temperatura. Esta fuente dispone de un regulador de tensión tipo 7805, equipado con un disipador hecho de aluminio ya que el transformador de entrada es de 220V – 9V y el integrado ha alcanzado temperaturas de 40 - 50°C. El funcionamiento detallado de la fuente, como así también de todos los componentes será explicado mas adelante.

El microcontrolador central se encarga de manejar todos los periféricos, realiza un barrido del teclado cada 20 mseg para ver si fue presionada alguna tecla, se encarga de pedirle al sensor que comience la conversión de temperatura, y una vez finalizada la misma, la muestra en el display y controla que no se haya producido ninguna alarma en los límites establecidos, en cuyo caso activa o desactiva el relé que se encarga del control externo.

Tal vez la función más compleja del microcontrolador, aunque no la más importante, es la de recorrer los menues a pedido del usuario. Esto requiere la realización de un parser, el cual recuerda el menú que esta recorriendo el usuario, y en función de eso, y de la tecla que fue presionada, accede al menú siguiente.

El funcionamiento del sensor de temperatura es bastante simple de explicar. Este recibe del microcontrolador un pedido de conversión de temperatura, y al cabo de un tiempo de aproximadamente 750 mseg avisa que la conversión fue completada y guarda el valor en memoria para que el micro pueda leerla cuando lo desee.

La única limitación que se puede encontrar, es que el rango de medición de temperatura del sensor no es muy amplio, sino que es de –55°C a +125°C, por lo que muchas aplicaciones industriales no podrán llevarse a cabo con este dispositivo.

El display debe mostrar la temperatura actual, y también los limites seleccionados por el usuario para realizar el control externo. A su vez, debe mostrar los menues mientras el usuario los recorre. Pero la parte más compleja del display es su conexión, ya que requiere la utilización de 11 pines del microcontrolador, además de la alimentación, ground, y un pequeño circuito para ajustar el contraste.

3. Selección de componentes:

Microcontrolador:

La idea incial del proyecto era utilizar un micro chico para tratar de aprovecharlo al máximo. Las primeras pruebas comenzaron con un AT89C2151. Este micro de Atmel posee 2k de memoria de código y 128 bytes de memoria de datos. La decisión por utilizar este micro, en lugar de uno mas grande, fue porque los pines que posee, que son quince, resultaron suficientes para conectarse con los periféricos. Pero luego de comenzar la realización del software, rápidamente se consumieron los 2k de memoria, y hubo que migrar a un micro un poco mas grande, el AT89S8252. La ventaja de este micro es que posee 8k de memoria de código y además 256 bytes de RAM, los que fueron necesarios más adelante. Además, posee la gran ventaja de poder ser programado «on-circuit», esto quiere decir que no necesita ser retirado del zócalo para ser programado, sino que se conecta directamente a la PC a través del puerto paralelo y es programado en cuestión de segundos. Como contrapartida, requiere la realización del circuito de programación en la placa, pero afortunadamente alcanza con un conector de tres pines y su respectivo cableado hacia el microcontrolador.

Sensor de temperatura:

En este caso la elección fue hecha de antemano, ya que fue utilizado el sensor 1-Wire disponible para el proyecto. Este sensor es un DS18S20, de la marca Maxim, el cual cuenta con un rango de medición de temperaturas de –55°C a +125°C, y una resolución de 9 bits (+- 0.5°C entre –10°C y +85°C).

Display:

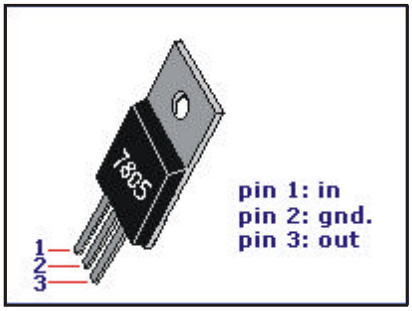
Analizando las necesidades de que se muestren en pantalla al mismo tiempo la temperatura que está siendo medida, y los límites que están configurados por el usuario, se requiere un display con una cantidad relativamente grande de caracteres, y 2 líneas. Por esto es que el display elegido es un Winstar WH2002A, con 2 líneas y 20 caracteres por línea, esto es suficiente para mostrar los datos necesarios y también para recorrer claramente los menues.

4. Descripción detallada de los componentes:

4.1. Fuente regulada de tensión:

Partes de una fuente de tensión:

La figura 1 muestra un diagrama en bloques de la fuente de tensión implementada en el proyecto. Esta convierte 220V AC a 5V regulados DC. A continuación se explicarán los componentes utilizados en cada bloque de la fuente.



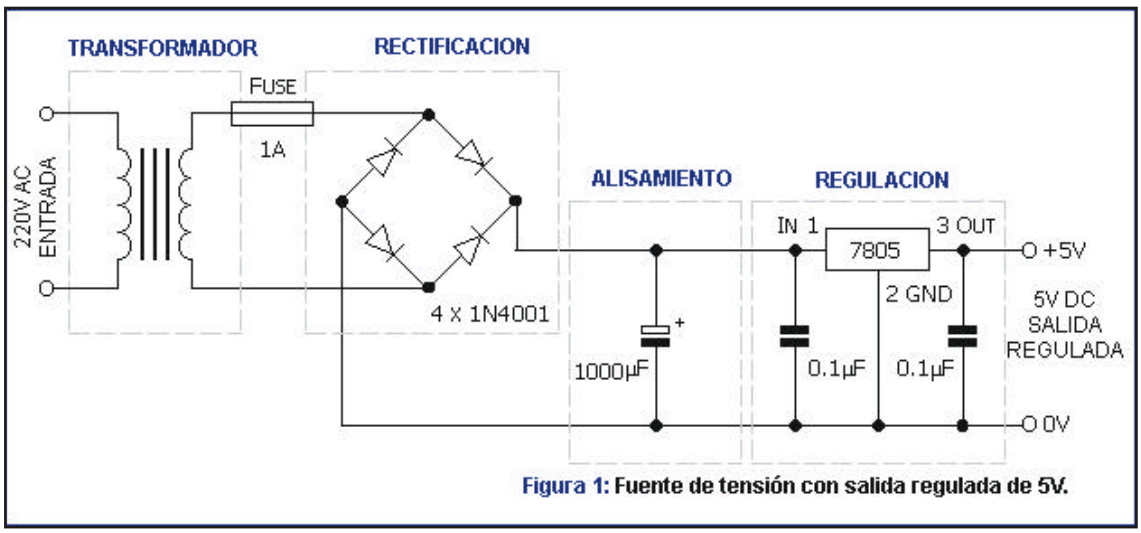


Figura 1: Fuente de tensión con salida regulada de 5V.

El Transformador:

El transformador utilizado en este proyecto es un reductor de voltaje step-down de 220V AC a 9V AC.

El rectificador:

El objetivo de un rectificador es convertir una forma de onda alterna en una continua. Hay dos circuitos diferentes de rectificación, conocidos como rectificador de «media onda» y de «onda completa». Ambos usan diodos para realizar la conversión.

En este proyecto el método usado es el de «onda completa», utilizando un puente de diodos de un Ampere.

Filtro de eliminación de ripple (smoothing):

Este circuito requerirá la eliminación del ripple de la salida de CC del rectificador, esto es bastante simple ya que sólo requiere un capacitor, como muestra la figura 2.

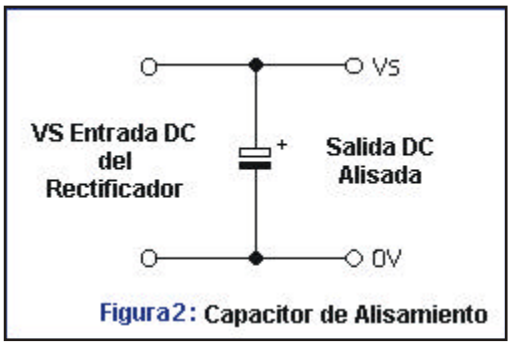


Figura 2: Capacitor de Alisamiento

Cálculo del capacitor de eliminación de ripple

El objetivo de calcular el capacitor de eliminación de ripple es encontrar su valor para lograr que la salida se mantenga lo más constante posible, y dentro del rango requerido por el regulador de tensión 7805.

$$V_P \; ? \; 9 \; ? \sqrt{2} \; ? \; 12{,}72 V_{PP}$$

Sabiendo que el transformador es de $9V_{EF}$

$$V_P \; ? \; 9 \; ? \sqrt{2} \; ? \; 12{,}72 V_{PP}$$

Se supone una variación de tensión de línea de ± 10%:

$$11{,}44V_P \; ? \; 12{,}72V_P \; ? \; 13{,}99V_P$$

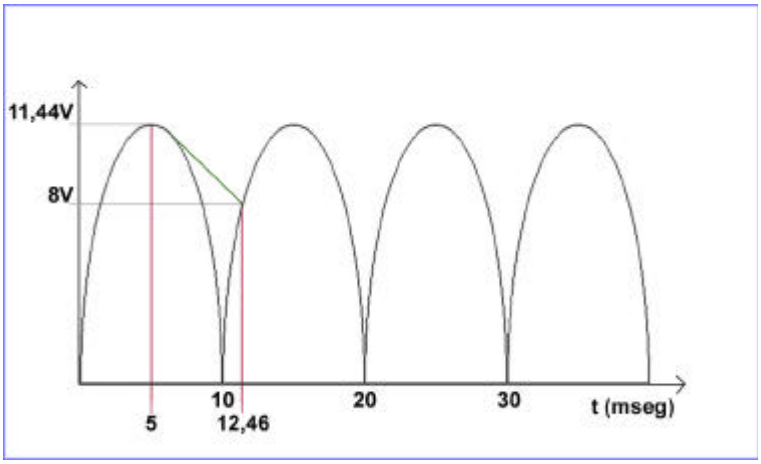
La peor condición sería que la tensión de línea fuera $11{,}44V_P$, entonces éste es el valor que usaré para los cálculos.

Según la hoja de datos del regulador 7805, éste puede funcionar con tensiones de entrada en el rango de 7V a 20V, pero como de costumbre se dejará un margen de trabajo por cualquier eventualidad, y el mínimo voltaje permitido será de 8V.

Se parte de la ecuación de la onda senoidal

$$V \; ? \; V_P \; ?\text{sen}?2?ft?? \quad t \; ? \; \frac{1}{2?f} \; ?\text{arcsen}?\frac{V}{V_P}??$$

$$t \; ? \; \frac{1}{2?\,50Hz} \; ?\text{arcsen}?\frac{8V}{11{,}94V}?? \; 2{,}46mseg$$



Ahora debe encontrarse la constante de tiempo t = RC, para que el capacitor mantenga su carga por encima de los 8V. Para esto se debe saber qué corriente requerirá el circuito, incluyendo todos sus componentes. Después de realizadas varias mediciones, puede suponerse una corriente máxima de 70mA

La resistencia mínima que se verá es:

$$R \; ? \; \frac{8V}{70mA} \; ? \; 114{,}3?$$

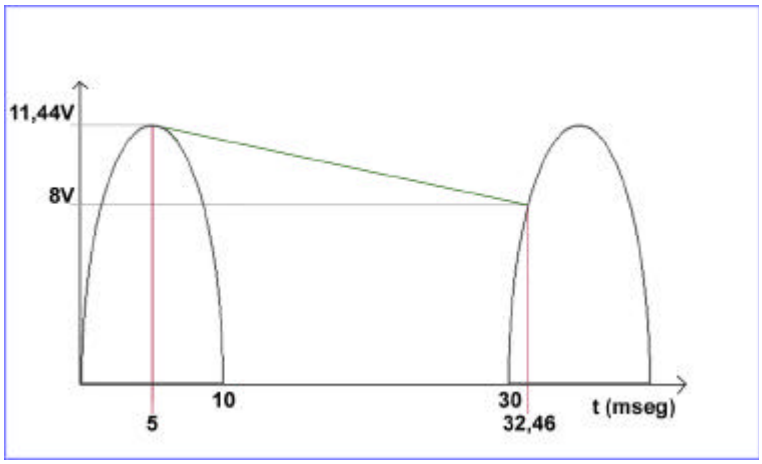
Ahora es posible hallar el valor del capacitor:

$$V_C \; ? \; V \; ?e^{?\frac{t}{?}} \; ? \; V \; ?e^{?\frac{t}{RC}}$$

$$C_{MIN} \; ? \; ? \; \frac{1}{R} \; ?\frac{t}{\ln?\frac{V_C}{V}??}$$

$$C_{MIN} \; ? \; ? \; \frac{1}{114?} \; ?\frac{7{,}46mseg}{\ln?\frac{8V}{11{,}44V}??} \; ? \; 182{,}95?F$$

Ahora bien, el valor de 182,95mF debería ser suficiente, pero la realidad es que en la línea pueden producirse micro cortes, es decir, que se pierdan semiciclos, y si esto llegara a suceder, el microcontrolador se resetearía o entraría en un estado desconocido. Para que esto no suceda, se realizará el cálculo nuevamente, pero esta vez tomando un margen de un ciclo (dos semiciclos) que equivale a 20mseg. Entonces, el capacitor debe mantener el voltaje de 8V durante 27,46mseg como mínimo.



$$C_{MIN} \; ? \; ? \frac{1}{114?} ? \frac{27{,}46 mseg}{\ln ? \frac{8V}{?11{,}44V ?} ?} \; ? \; 673{,}45 ? F$$

Ahora sí, a partir de este valor se selecciona un capacitor con un valor comercial mayor, por ejemplo de 1000mF. Esto debería ser más que suficiente para mantener el nivel de tensión por encima de los 8V, aunque se produzcan micro cortes, o si por algún error de medición de corriente, ésta crece por encima de los 70mA.

Regulación:

El circuito de este proyecto requiere una fuente regulada de 5V CC sin ripple, por lo que será necesario utilizar un regulador de tensión. Uno de los más comunes es el 7805, que pertenece a la serie de reguladores 78XX y es el que se explicará a continuación.

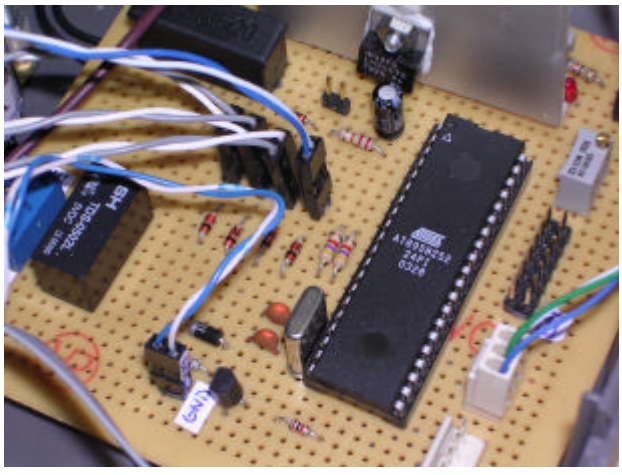
Reguladores de tensión 78XX:

Los reguladores de tensión 78XX entregan a la salida diferentes voltajes fijos, en el rango de 5V a 24V. Con un apropiado disipador de calor, el regulador puede entregar 100 mA de corriente de salida.

Disipadores:

Durante su uso, el regulador puede calentarse demasiado, por lo que es necesario añadir un disipador de calor. El tipo de disipador depende de varios factores, entre ellos el tipo de encapsulado del regulador, y la cantidad de calor que es necesario disipar. En este caso no se realizó ningún cálculo para decidir el tipo de disipador, pero basándose en la hoja de datos, no hubiera sido necesario ninguno. La razón por la cual se incorporó es porque durante la prueba del funcionamiento se registraron temperaturas de 40 – 50°C en una habitación con aire acondicionado, y se supuso que las temperaturas serían bastante mayores una vez dentro del gabinete, y con una temperatura ambiente mayor. El disipador consiste en un trozo de una barra de aluminio, atornillada al regulador.

4.2. Microcontrolador



Descripción:

El microcontrolador usado en el proyecto es un Atmel AT89S8252, compatible con MCS-51, que posee las siguientes características: 8 Kbytes de memoria flash que puede ser programada directamente desde la PC, con el cable adecuado, 2 Kbytes de memoria EEPROM, 256 Bytes de memoria RAM, 32 líneas de entrada/salida, Watchdog programable, 3 contadores de 16 Bits, 9 fuentes de interrupción, puerto serie full-duplex y oscilador on-chip.

Memoria de datos – EEPROM y RAM:

El AT89S8252 implementa 2Kbytes de EEPROM on-chip para guardar datos mientras la alimentación se encuentre desconectada, y 256 Bytes de RAM. Los 128 Bytes superiores ocupan un espacio paralelo al SFR (Special Function Registers). Esto significa que los 128 Bytes superiores poseen la misma dirección pero están físicamente separados del SFR.

Se utiliza direccionamiento directo para acceder al SFR y direccionamiento indirecto para acceder a los 128 bytes superiores de la memoria RAM.

Timer 0 y Timer 1:

La función de Timer o Contador se selecciona a través de los bits de control C/T dentro del SFR (Special Function Registers) dentro del registro de control TMOD. Estos dos Timers/Contadores poseen 4 modos de trabajo, que son seleccionados por pares de bits (M1, M0), también dentro del registro de control TMOD.

Modo 0 - 13 bits:

La frecuencia de entrada es 1/12 de la frecuencia del oscilador trabajando en modo Temporizador, o la frecuencia de la señal de entrada al pin Tx (1) si trabaja en modo contador (counter).

El Timer funciona en este caso como un contador de 8 bits precedido por un predivisor de 32. El registro del contador se configura, pues, como un registro de 13 bits: 5 bits en la parte baja para el predivisor (TLx) y 8 bits en la parte alta para la cuenta (THx). Cuando el contador pasa de estar los 13 bits a 1, a estar todos a 0, el indicador de interrupción TRx se pone a 1. La entrada de conteo esta habilitada cuando TRx (bit de control del registro TCON) está a 1 y cuando GATE (Bit de modo de TMOD) está a 0, o bien cuando la entrada INTx se encuentra en estado 1.

Para el Timer 0 sería lo mismo pero utilizando los registros correspondientes a este Timer.

En este modo se produce el desbordamiento al contar 2^13=8.192 ciclos máquina o flancos de bajada, hecho que activa el flag de interrupción del timer utilizado.

Los 5 bits de TLx son de menor peso, y los 3 de mayor peso son indeterminados, por lo que deberán ser ignorados por el programador.

Modo 1 - 16 bits:

El modo 1 es idéntico al modo 0 excepto porque el registro del contador emplea su capacidad completa (16 bits) sin predivisor, 8 en TLx y otros 8 en THx. El desbordamiento se produce al contar 2^16=65.536 ciclos máquina o flancos de bajada, con lo que se activa el flag de interrupción.

Modo 2 - 8 bits auto-recarga:

El modo 2 de funcionamiento configura el Timer como un contador de 8 bits con recarga automática. La parte baja TLx sirve de contador. Cuando este contador llega a 0 por desbordamiento, el indicador TFx se pone a 1 y el contenido de THx se carga en el contador TLx. El valor de auto recarga lo elige el programador según sus cálculos, y el caso de que fuera 00 hexadecimal, entonces se produciría el desbordamiento al contar 2^8=256 ciclos máquina o flancos de bajada.

Modo 3:

Es un modo que proporciona dos contadores de 8 bits independientes, siendo los registros de cuenta, el TL0 y el TH0. El primero, mantiene la estructura del Timer T0, y el segundo, permite solamente la cuenta de ciclos máquina, realizándose su control directamente mediante la señal TR1 y activando el flag de interrupción del Timer T1 cuando se produce el desbordamiento.

El modo 3 está dirigido a las aplicaciones que necesitan un contador adicional de 8 bits. Con el Timer T0 configurado en modo 3, el 8051 posee un timer de 16 bits (T1) y dos timers de 8 bits.

Esta característica sólo es utilizable en aplicaciones en las que el Timer T1 no requiera su recurso de interrupción, puesto que en este caso su indicador (TF1) está unido a la salida del contador de 8 bits en que se convierte TH0.

En modo 3, el Timer 1 está bloqueado como si su bit de control TR1 estuviera a 0.

Timer 2:

Los modos de funcionamiento del timer 2 son tres:

a) Modo captura
b) Modo autor recarga
c) Modo generador de frecuencia para comunicación serie

Timer 2 en modo captura:

Este modo permite capturar el valor actual del timer 2. Si el bit 3 del registro T2CON es 1 vemos que el control inferior se cierra y por lo tanto cuando se produce una transición de entrada externa T2EX (flanco descendente), el valor que se encuentra en lo registros de cuenta TL2 Y TH2, es capturado por lo registros RCAP2H y RCAP2L. Además, el flanco descendente del pin T2Ex, pone a 1 el flag de EXF2

Timer 2 en modo autor recarga:

En este modo si EXEN2=0, cuando el timer 2 produce el desbordamiento pone TF2=1 y además recarga TL2 y TH2 con los valores de RCAP2H y RCAP2L, cargados inicialmente por software. Este modo puede ser utilizado como generador de frecuencia para comunicación serie, ya sea sólo para recepción (RCLK=1), sólo para transmisión (TCLK=0), o ambas (RCLK=1 y TCLK=1).

Si EXEN2=1, el timer 2 realiza lo anterior y además al producirse un flanco descendente en el pin T2EX, activa la recarga y el flag EXF2.

Cuando se emplea el modo autor recarga como generador de frecuencias para las comunicaciones serie (RCLK + TCLK=1), los componentes del timer 2 se organizan de tal manera que la frecuencia que actúa como contador es mas elevada (Fosc/2 en vez de Fosc/12). La frecuencia de comunicación FC, viene dada por la formula:

$$FC = \frac{F_{OSC}}{32 \times (65536 - (RCAP2H, RCAP2L))}$$

donde RCAP2H y RCAP2L representan el valor de 16 bits con el que se recarga el contador.

Interrupción del timer 2:

Una interrupción en el timer 2 se produce en base a los indicadores TF2 y EXF2. La detección de la interrupción se desarrolla a partir de la operación lógica OR entre estos dos indicadores, con lo cual basta con que cualquiera de estos se encuentre a 1 para que se reconozca una petición de interrupción. La puesta a cero de los indicadores TF2 y EXF2 se debe realizar por software antes de la vuelta al programa principal.

Interrupciones:

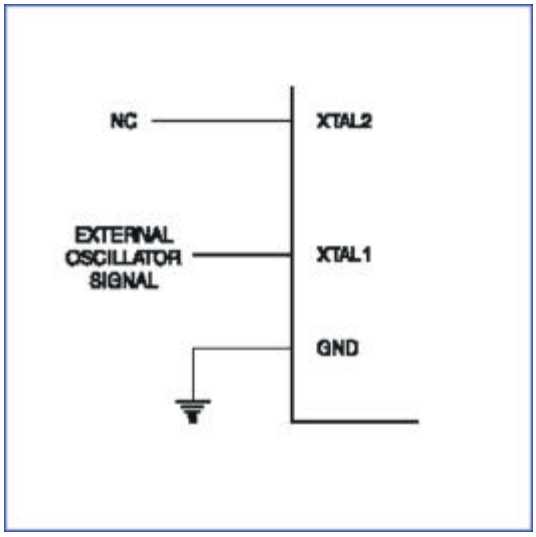
El micro posee un total de 6 vectores de interrupción: dos interrupciones externas (INT0 e INT1), tres interrupciones por timer (Timers 0, 1 y 2) y la interrupción del puerto serie.

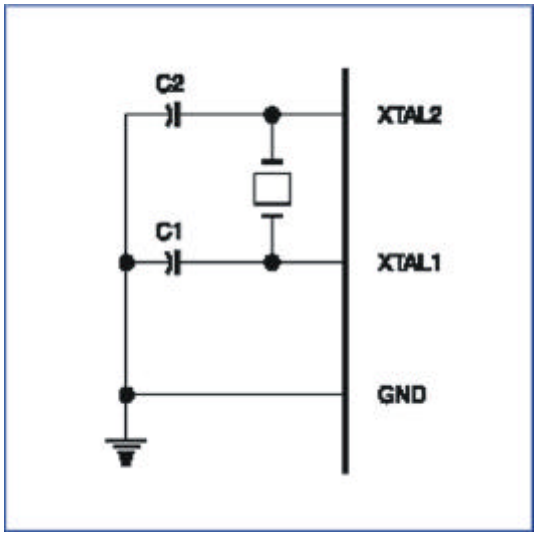
Cada una de estas interrupciones puede ser habilitada o deshabilitada individualmente mediante el seteo o borrado de un bit de control dentro del SFR (Special Function Register), en su registro de control IE. Además de esto, IE posee un bit para deshabilitar por completo todas las interrupciones.

Características del oscilador:

XTAL1 y XTAL2 son la entrada y la salida respectivamente de un amplificador inversor que puede ser configurado para usarse como un oscilador on-chip.

Para usar el dispositivo desde una fuente de clock externa, XTAL2 se debe dejar desconectado y utilizar solamente XTAL1.





Programación de la memoria Flash:

El microcontrolador posee 8kbites de memoria flash de código. Soporta dos modos diferentes para programarlos: un modo paralelo con voltaje de 12V (V_{PP}) y un modo serie con voltaje de 5V (V_{PP}).

Como se explicó resumidamente en la introducción, el método utilizado para la programación fue «in-circuit» a través del cable paralelo de la PC.

Para la programación «in-circuit» son necesarias tres cosas: el cable que va del puerto paralelo a la placa, el software para programar, y el circuito de programación en la placa. Los dos primeros ya se encontraban disponibles, por lo que solo se tuvo que aprender a utilizarlos. Y el circuito de programación resultó muy sencillo, solo tres cables, desde el microcontrolador hacia un conector polarizado de tres pines.

El software utilizado fue el SP89 Versión 0.7.1 para DOS, tal vez no sea el mejor o el más nuevo, pero es el que se está usando con éxito en el INTI desde hace tiempo.

4.3. Display

Con frecuencia, un programa en un microcontrolador debe interactuar con el mundo exterior usando dispositivos de entrada y de salida. Uno de los dispositivos más comunes que podemos conectar es un display de LCD, y dentro de estos, los más comunes son los 16x2 y 20x2. Esto significa 16 caracteres por 2 líneas y 20 caracteres por 2 líneas respectivamente.

El display elegido para el proyecto es un WH2002A, de la empresa Winstar. Este consta de dos líneas de veinte caracteres cada una (20x2), y soporta un estándar muy popular que permite comunicarse con la mayoría de los displays de LCD, sin importar el fabricante. El estándar es llamado 44780, que se refiere al chip controlador que recibe datos del microcontrolador y se comunica directamente con el display.

Funcionamiento del 44780

El estándar 44780 requiere 3 líneas de control como así también 4 u 8 líneas para el bus de datos. El usuario debe seleccionar si el LCD funcionará con un bus de 4 bits o con un bus de 8 bits. Si se usa el bus de 4 bits, el LCD requerirá de un total de 7 líneas de datos, en cambio, requerirá 11 líneas si se usa el bus de 8 bits, como en el caso de este proyecto.

Un display de LCD se maneja como si se tratara de una zona de memoria accesible por el microcontrolador, de forma que cualquier escritura de caracteres ASCII en este espacio de memoria provoca una actualización del caracter correspondiente sobre las líneas del visualizador.

La conexión más simple consiste en su conexión directa con los pines del microcontrolador, éste accede al dispositivo escribiendo y leyendo como si se tratara de un chip de memoria.

Descripción de los pines del display

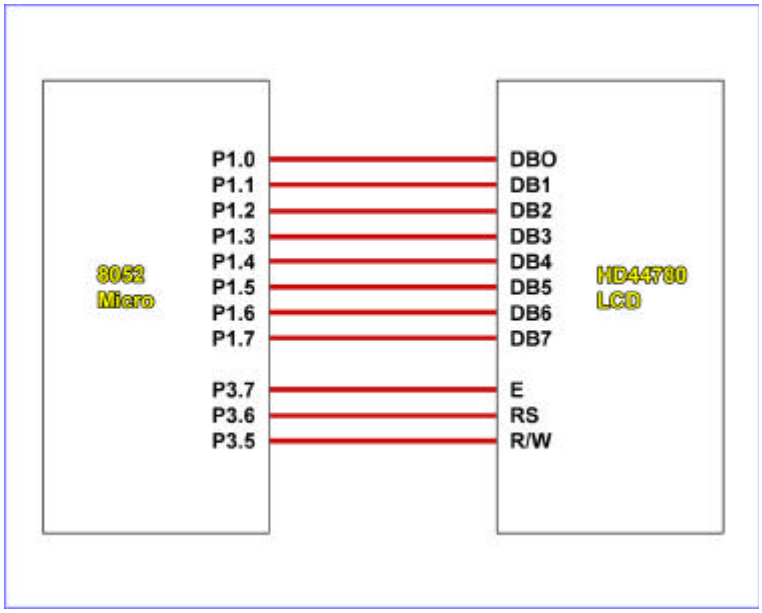
Hay 3 líneas de control (E, RS y R/W) y 8 líneas de datos (DB0 – DB7).

- E: Es llamada «Enable». Esta línea es usada para indicarle al display que se le están enviado datos. Para enviarle los datos el programa debe primero poner esta línea en alto, y luego setear las otras dos líneas de control y/o poner datos en el bus. Cuando se termina de trabajar con todas las líneas, se debe poner nuevamente en bajo la línea E. La transición le indica al display que tome los datos que se encuentran en las líneas de datos y/o control y ejecute los comandos correspondientes.
- RS: es llamada «Register Select». Cuando esta línea está en bajo los datos son tratados como comandos o instrucciones especiales, en cambio, cuando está en alto, los datos son tratados como texto para ser mostrado en el display.
- R/W: Es llamada «Read/Write». Cuando R/W esta en bajo, la información en el bus de datos esta siendo escrita en el display. Cuando la línea esta en alto, la información está siendo leída del display.

Un ejemplo de configuración de hardware

Como se mencionó anteriormente, el display requerirá 11 líneas de datos para comunicarse con el microcontrolador.

A continuación, se muestra un esquema simple de cómo sería la conexión:



Como se puede ver, hay una relación uno a uno entre los pines del microcontrolador y la línea de datos del display. Por esto, cuando se escribe el software se pueden asignar constantes para referirse a las líneas del display, como por ejemplo:

```
DBN   EQU  P1.N
E     EQU  P3.7
RS    EQU  P3.6
R/W   EQU  P3.5
DATA  EQU  P1
```

Una vez establecido esto, se pueden referir las líneas de entrada/salida con sus nombres en el estándar 44780.

Manejo de la línea de control «E»

La línea de control «E» se usa para indicarle al display que hay una instrucción en el bus de datos lista para ejecutarse. La línea «E» debe ser levantada/bajada cada vez que se envía una instrucción al LCD sin importar que la instrucción sea de lectura o escritura. En resumen, siempre se debe manipular la línea «E» cuando se comunica con el LCD para que éste sepa que se le está enviando una instrucción.

Chequeo del estado del LCD

Como se vió con anterioridad, al LCD le toma un cierto tiempo ejecutar cada instrucción. El retraso varía dependiendo de la frecuencia del oscilador del 44780 como también de la instrucción que está siendo ejecutada.

Es posible escribir código que espere una cierta cantidad de tiempo para permitir que el LCD ejecute las instrucciones, pero este método de esperar no es muy flexible. Si se cambia la frecuencia del cristal, el software necesita ser modificado. Así mismo si el LCD es cambiado por otro, aunque sea 44780 compatible, pero requiere más tiempo para realizar las operaciones, el programa también deberá ser modificado.

Un método más robusto de programación es usar el comando «Get LCD Status» para determinar si el LCD está todavía ocupado ejecutando la última instrucción recibida.

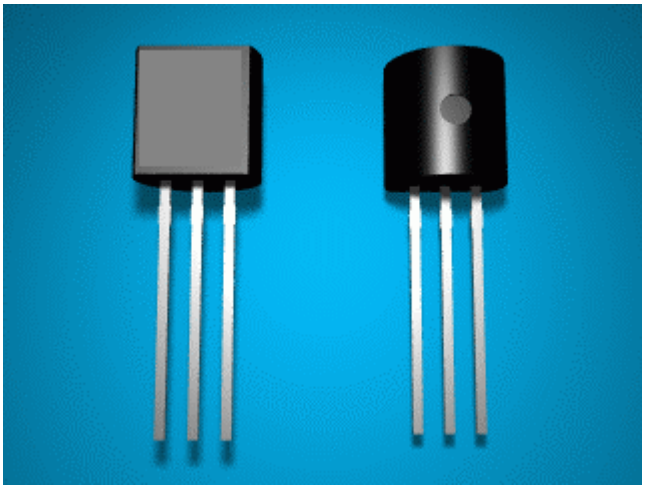
Inicialización del LCD

Antes de que se pueda realmente usar el LCD primero debe ser inicializado y configurado. Esto se hace enviando un número de instrucciones de inicialización al LCD.

Lo primero que se le debe indicar al LCD es si la comunicación se hará con un bus de 8 bits o de 4 bits. También se debe seleccionar el tipo de carácter, en este caso de 5x8 puntos.

Es una buena idea limpiar la pantalla del display como primera operación luego de que ha sido inicializado.

4.4. Sensor de temperatura



Descripción general:

El sensor es un termómetro digital Maxim DS18S20, con el que se pueden realizar mediciones de temperatura en grados centígrados con una resolución de 9 bits. También posee una función de alarma con valores no volátiles programables por el usuario. El DS18S20 se comunica a través de un bus 1-Wire, que por definición requiere solamente una línea de datos (y masa) para la comunicación con un microprocesador central. Tiene un rango de temperatura de operación desde –55°C hasta +125°C y con una precisión de 0.5°C en el rango de –10°C a +85°C. Adicionalmente, el sensor puede obtener energía directamente desde la línea de datos, eliminando la necesidad de alimentación externa.

Cada sensor posee un código de serie de 64 bits, lo que permite conectar múltiples sensores en un mismo bus 1-Wire, y por consiguiente, con un único microprocesador pueden ser controlados una gran cantidad de sensores en un área grande.

Funcionamiento – Medición de temperatura

La funcionalidad básica del DS18S20 es entregar la temperatura directamente en formato digital, es decir que no requiere un conversor analógico – digital adicional. La salida del sensor tiene una resolución de 9 bits, lo que equivale a pasos de 0.5°C.

Para realizar una medición de temperatura y su conversión, el microprocesador debe enviar un comando llamado «Convert T». Luego de la conversión, el valor medido es guardado en un registro de 2 bytes en la memoria del sensor. El microprocesador debe entonces leer la memoria del sensor para obtener los datos.

Funcionamiento – Alarmas

Luego de cada conversión de temperatura, el valor es comparado con los valores de alarma definidos por el usuario que están almacenados en los registros T_H y T_L de la memoria EEPROM del sensor.

El bit 7 de los registros indica el signo, esto deja solo 7 bits para guardar los limites de temperatura, y por consiguiente la resolución no puede ser de 0.5°C como en la medición, sino que es de 1°C.

Alimentación del DS18S20

El sensor puede ser alimentado por una fuente de alimentación externa a través del pin V_{DD}, o puede trabajar con el modo de «alimentación parásita», que le permite obtener energía directamente del bus 1-Wire, a través del pin DQ. La carga alimenta el sensor cuando el bus está en alto, y se almacena en el capacitor de alimentación parásita (C_{PP}) para proveer energía cuando el bus esta en bajo.

Código ROM de 64 bits

Cada DS18S20 contiene un código único de 64 bits almacenado en memoria ROM. Los 8 bits menos significativos contienen el código de familia 1-Wire: 10h. Los siguientes 48 bits contienen un número de serie único. Los 8 bits más significativos contienen un chequeo de redundancia cíclica (CRC) que es calculado de los primeros 56 bits del código de la ROM.

Memoria

La memoria está organizada como se muestra más abajo en el mapa de memoria. Consiste de un «scratchpad» SRAM con EEPROM no volátil para guardar los registros de alarma alto y bajo (T_H y T_L).

Los bytes 0 y 1 del «scratchpad» contienen el LSB y MSB del registro de temperatura respectivamente. Estos bytes sólo pueden leerse. Bytes 2 y 3 proveen acceso a los registros T_H y T_L. Bytes 4 y 5 están reservados para uso interno del dispositivo y no pueden ser escritos, además, cuando son leídos devuelven todos unos. Bytes 6 y 7 contienen registros que se usan para extender la resolución de medición de temperatura. El byte 8 es de solo lectura y contiene el código CRC para los bytes 0 a 7 del scratchpad.

Mapa de memoria del scratchpad

Byte 0	Temperatura LSB		
Byte 1	Temperatura MSB		
Byte 2	Registro T_H	ßà	Registro T_H
Byte 3	Registro T_L	ßà	Registro T_L
Byte 4	Reservado		
Byte 5	Reservado		
Byte 6	«Count Remain»		
Byte 7	«Count per °C»		
Byte 8	CRC		

BUS 1-Wire

El sistema de bus 1-Wire usa un único dispositivo maestro para controlar uno o más dispositivos esclavos. El DS18S20 es siempre un esclavo. Cuando hay un solo dispositivo esclavo en el bus, el sistema se denomina «single-drop», si hay múltiples dispositivos, el sistema se denomina «multi-drop».

Todos los comandos y los datos son transmitidos por sus bits menos significativos primero a través del bus 1-Wire.

La siguiente descripción del bus 1-Wire está dividida en tres partes: Configuración del hardware, secuencia de transacción y señalización del bus 1-Wire.

Configuración del hardware

El bus 1-Wire tiene por definición una sola línea de datos. Cada dispositivo (maestro o esclavo) se conecta a la línea de datos por un puerto «open drain» o «3-state». Esto permite que cada dispositivo libere

la línea de datos cuando no está transmitiendo para que el bus quede disponible para ser usado por otro dispositivo.

El bus 1-Wire requiere de un resistor de pullup externo de aproximadamente 5kW, por consiguiente el estado de reposo del bus es alto. Si el bus es mantenido en bajo durante un período de mas de 480ms, todos los componentes del bus serán reseteados.

Secuencia de transacción

La secuencia de transacción para acceder al DS18S20 es la siguiente:

Paso 1: Inicialización

Paso 2: Comando ROM (seguido por un intercambio de datos)

Paso 3: Comando de función (seguido por un intercambio de datos)

Es sumamente importante seguir esta secuencia cada vez que se accede al DS18S20, ya que no responderá si cualquiera de estos pasos faltan o son cambiados de orden. Las únicas dos excepciones son los comandos «Search ROM» y «Alarm Search», luego de cualquiera de estos dos comandos ROM, el dispositivo maestro debe retornar al paso 1 de la secuencia.

Paso 1: Inicialización:

Todas las transacciones del bus 1-Wire comienzan con una secuencia de inicialización. Esta consiste en un pulso de reset transmitido por el dispositivo maestro, seguido por un pulso de presencia transmitido por el esclavo. El pulso de presencia le permite al dispositivo maestro saber que hay dispositivos (como el DS18S20) conectados al bus 1-Wire, y que están listos para operar.

Comandos ROM:

Cuando el dispositivo maestro del bus 1-Wire detectó un pulso de presencia, puede enviar un comando ROM. Estos comandos permiten seleccionar un único dispositivo dentro de una red 1-Wire, a través de su código ROM único de 64 bits. Además permiten determinar la cantidad y tipo de dispositivos que se encuentran dentro de la red.

Comandos de función:

Luego de que el dispositivo maestro envió un comando ROM para direccionar el DS18S20 al que quiere comunicarse, puede enviar uno de los comandos de función. Estos comandos permiten al dispositivo maestro leer y escribir el scratchpad, iniciar conversiones de temperatura, y determinar el modo de alimentación.

4.5. Control externo

El control externo es el bloque más sencillo que compone el dispositivo. Básicamente se trata de un relé, con capacidad para manejar pequeños dispositivos, como por ejemplo un soldador, un calentador eléctrico, o cualquier aparato similar. Debido a que la corriente que puede entregar el relé no es grande, 2 Amperes según las especificaciones, hay que evitar conectar aparatos que consuman mas de 400 Watts de potencia.

5. Funcionamiento del software

El software está realizado casi en su totalidad en lenguaje C, con el programa Keil mVision2, y consta de una rutina principal, y varias funciones para el manejo de periféricos. En los próximos puntos se explicará en detalle cada uno de ellos.

5.1. Rutina principal (Main)

La rutina principal comienza realizando la inicialización de todos los componentes y variables. Primero setea los puertos como entrada, luego inicializa los registros del timer, y del watchdog, le asigna el estado inicial a diversas variables que serán utilizadas dentro del programa, y por último llama las funciones de inicialización de los periféricos, como el display o el sensor de temperatura.

Una vez que está todo correctamente inicializado, entra en un lazo sin fin que va a realizar las siguientes funciones:

- Chequea si fue presionado alguno de los pulsadores, si así fuera, debe ejecutar las rutinas del parser.
- Chequea si esta prendido el flag de realizar la conversión y lectura de la temperatura, si es así, debe realizar los tres pasos detallados en la secuencia de transacción del bus 1-Wire, y luego mostrar la temperatura actualizada en el display.
- Atiende las interrupciones de timer que se producen cada 20 mseg.

5.2. Display

La rutina del display tiene una función de reset que se encarga de toda su configuración inicial, esta cuenta con una parte fundamental que es una seguidilla de retrasos, dependientes de la frecuencia de oscilación del cristal del microcontrolador, que terminan dejando al display listo para recibir comandos.

Dentro de su funcionamiento se encuentran tres funciones principales:

- Una de ellas se encarga de pasarle al display la información que encuentra en un buffer, el cual es llenado por las otras dos funciones.
- La primera de las dos funciones que llenan el buffer, lo hace a partir de la memoria de datos. Esta memoria es dinámica y depende de las variables del software, como por ejemplo, el valor de la temperatura a mostrar.
- La otra función toma los datos de la memoria de código y estos datos son definidos en tiempo de diseño, es decir, son estáticos y no pueden ser modificados una vez que el microcontrolador ya fue programado, como por ejemplo, el texto de los menues.

5.3. Sensor de temperatura (1-Wire)

Lo primero que se puede decir acerca del software para controlar el sensor de temperatura, es que se encuentra muy bien explicado y ejemplificado en las notas de aplicación.

Se encontraron diversas rutinas para leer y escribir bits y bytes, así como también para inicializar el dispositivo, el único problema era que estaban todos los retardos especificados para una velocidad de clock diferente a la que se utilizó, por este motivo se tuvo que estudiar con detenimiento los diagramas de tiempos, y así poder definir los niveles de retardos necesarios para cumplir los requerimientos con el clock utilizado en este proyecto.

Las funciones dentro de este módulo son:

- ow_reset: Resetea el sensor. Es necesario llamarla cada vez que el microcontrolador quiere comunicarse con el sensor, ya que es el primero de los tres pasos requeridos en la secuencia de transacción.
- read_bit: Es llamada solamente dentro de la función read_byte.
- read_byte: Es la encargada de leer todos los datos del scratchpad del sensor, se utiliza varias veces seguidas para leer secuencialmente toda la memoria.
- write_bit: Es llamada solamente dentro de la función write_byte.
- write_byte: Se utiliza para escribir en los únicos dos bytes permitidos, que son los correspondientes a los registros de alarma T_H y T_L.

5.4. Parser (Diagrama de estados)

El funcionamiento del parser es una de las partes más complejas, sino la más compleja, que hubo que hacer dentro del software. Para su realización hubo que definir primero los diferentes estados que puede tomar el parser, partiendo desde el estado de reposo en donde debe ser inicializado, y pasando por todas las opciones de configuración del sensor.

Una vez definidos los 16 estados (en este caso), el siguiente paso fue graficarlos y anotar cada una de las funciones necesarias así como también la función de los cuatro pulsadores en cada estado. Esto puede verse en los gráficos de las dos hojas siguientes.

Diagrama de Estados (Parte 1)

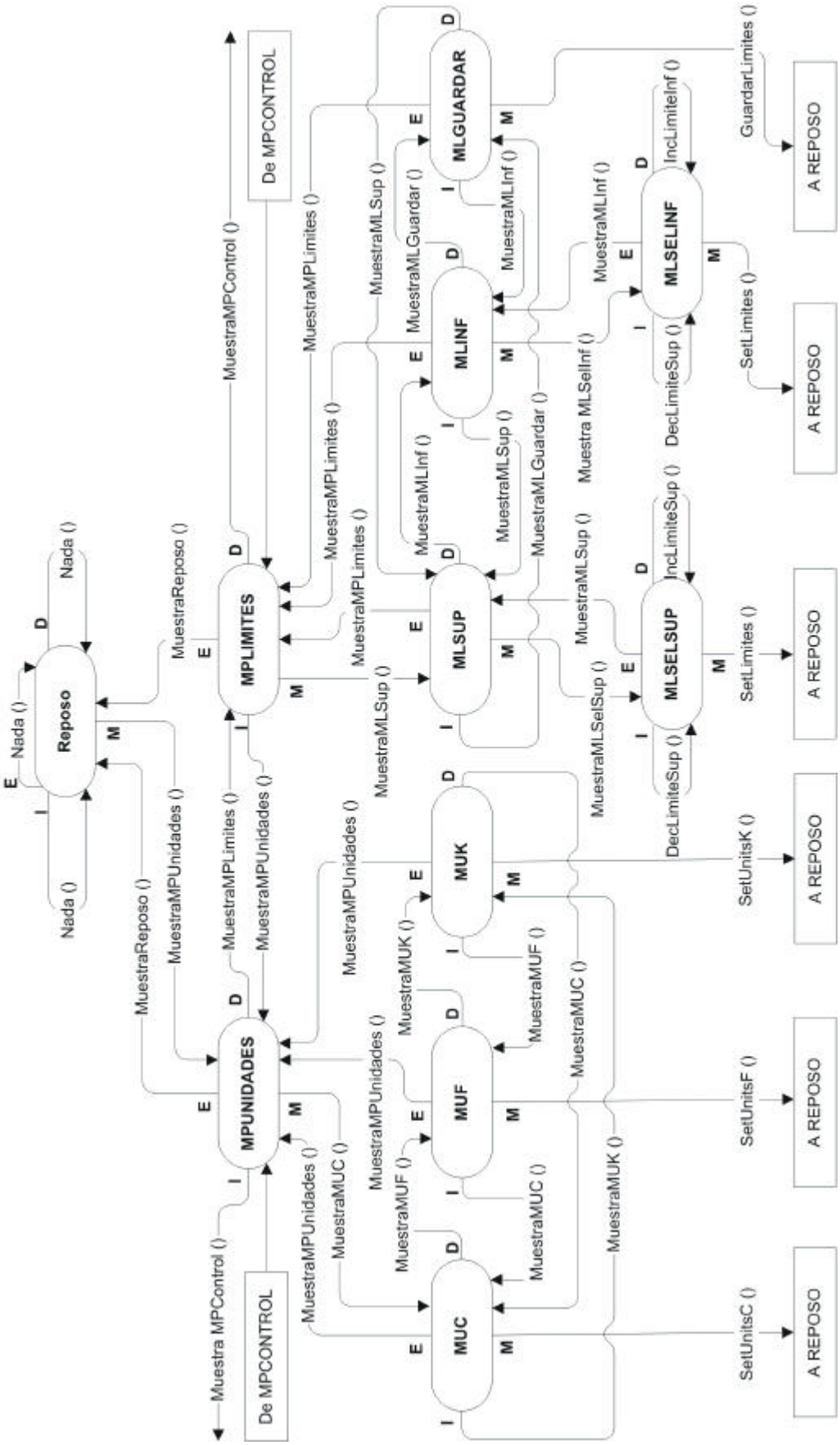
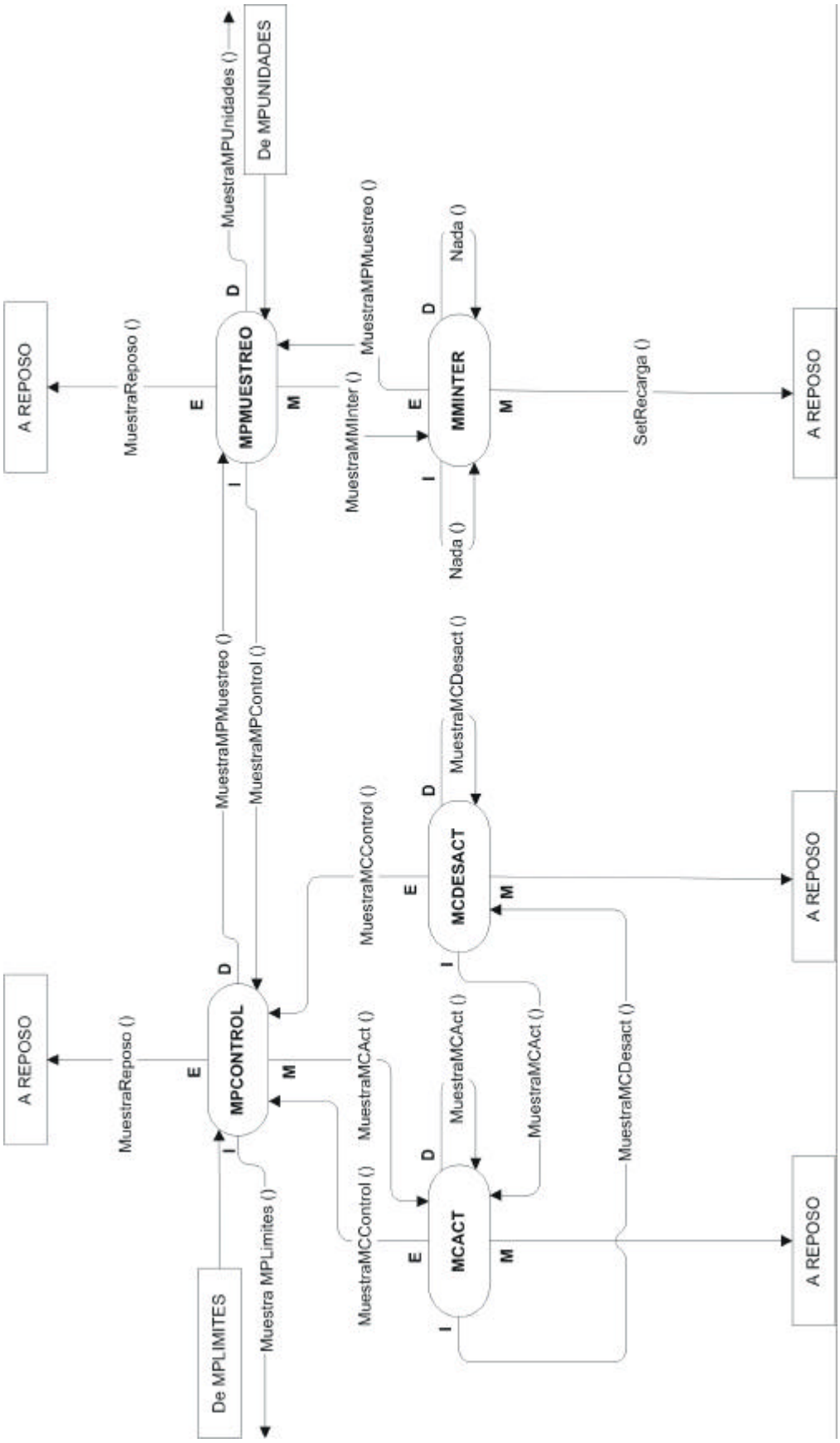


Diagrama de Estados (Parte 2)



Las funciones que se destacan en este módulo son:

- ParserIni: Es llamada en la rutina de inicialización y es la que se encarga de setear el estado inicial del parser (REPOSO) y luego de mostrarlo en la pantalla del display. Como comentario se puede agregar que obviamente el display debe estar inicializado anteriormente al parser.
- Parser: Lo primero que hace es tomar el estado actual del parser y luego se encarga de comparar la tecla que ha sido presionada con todas las teclas que tiene disponibles (y una extra llamada «DEFAULT» por si se ingresó a la función por error), para decidir cuál es el estado siguiente y qué función debe ejecutar.

6. Aplicaciones

El campo de aplicación para este controlador de temperatura es muy amplio, ya que abarca desde aplicaciones domésticas a controles industriales debido al alto rango de temperaturas que maneja que va desde los -55ºC a +125ºC.

Las distintas posibilidades que brinda este dispositivo son:

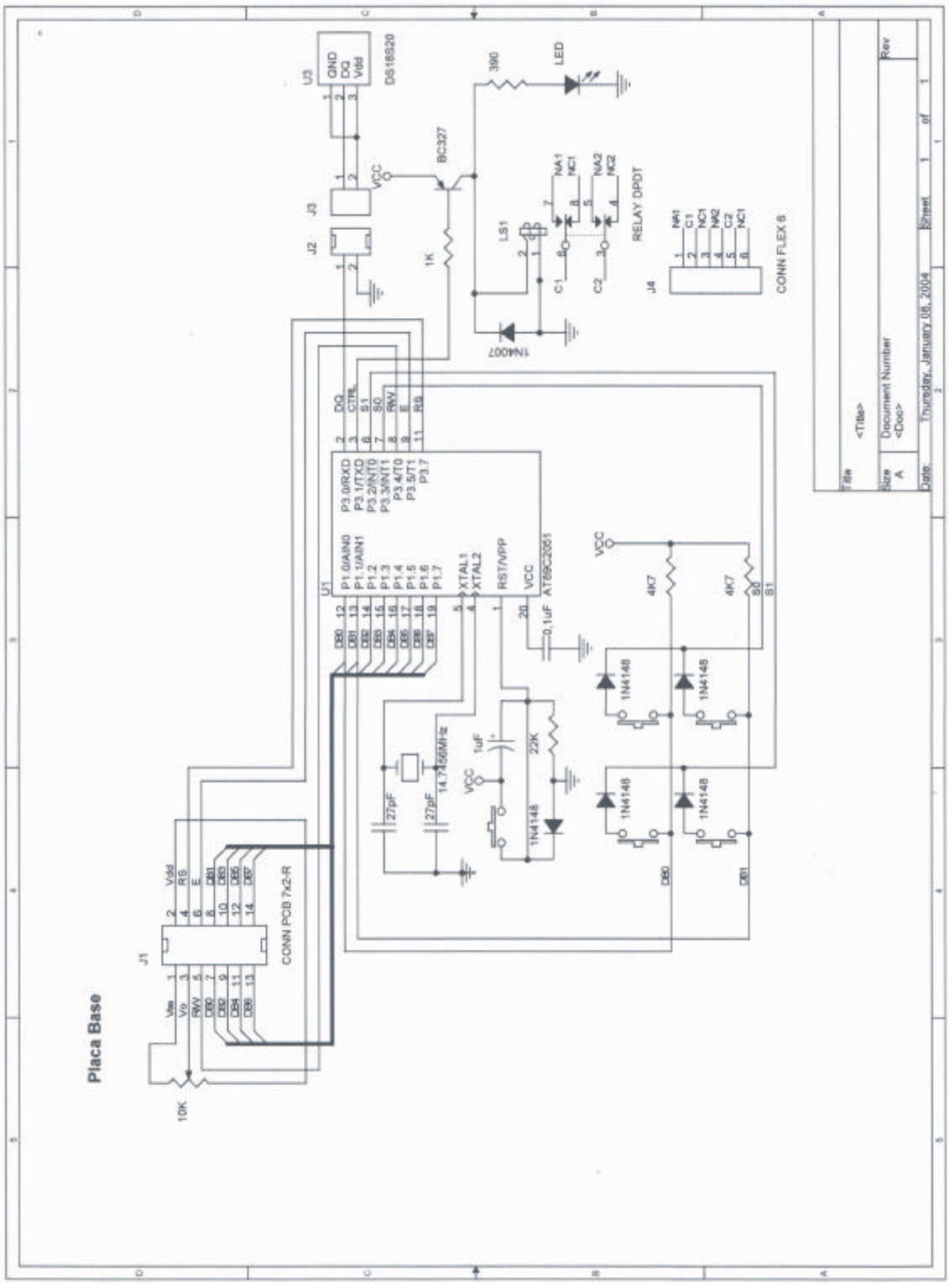
- Mantener la temperatura de una habitación dentro de los márgenes que sea necesario, como por ejemplo para una cámara frigorífica o para una bodega.
- Dado que este sensor no requiere calibración previa, puede ser utilizado (y de hecho ya ha sido utilizado) para calibrar otros sensores de temperatura no digitales.
- Regular la temperatura de líquidos, como por ejemplo para ser usado dentro un laboratorio o en el hogar.

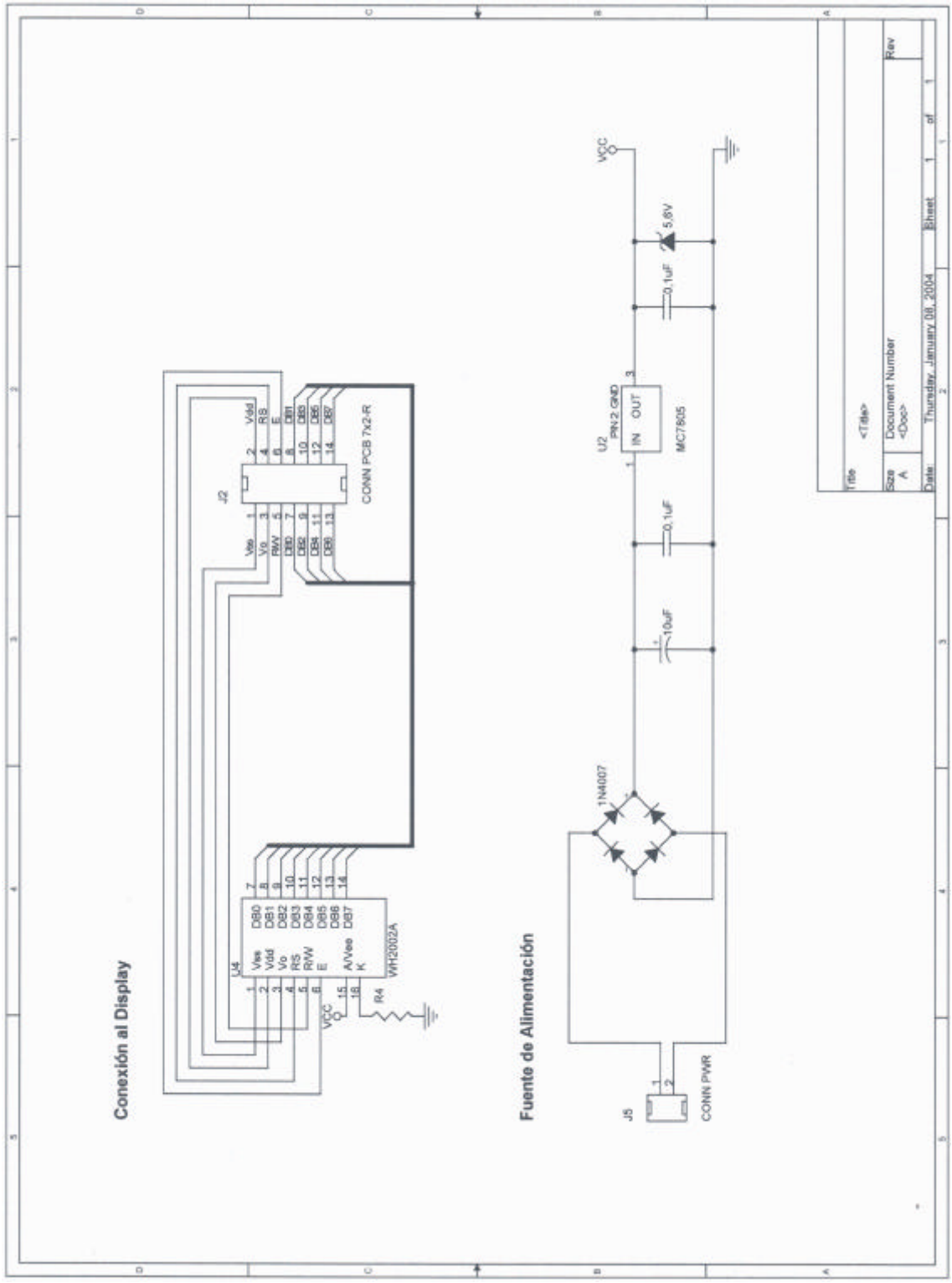
7. Posibilidades Futuras

Dentro de las posibilidades futuras de crecimiento del dispositivo se encuentran:

- El controlador de temperatura permite la actualización del software sin tener necesidad de hacer ningún cambio en el hardware, ya que puede conectarse a la PC directamente, hacer los cambios necesarios en el programa y la próxima vez que sea iniciado el dispositivo ya lo hará con la nueva configuración. Esto permite que puedan incorporarse al dispositivo funciones que antes no hayan sido previstas, o utilizarlo en aplicaciones que no habían sido tenidas en cuenta en el momento del desarrollo del dispositivo.
- A través del mismo cable en el que está conectado el sensor de temperatura actualmente, pueden conectarse todos los sensores de temperatura que se desee para poder tomar muestras de diferentes lugares al mismo tiempo. Para aplicaciones más complejas que puedan llegar a requerir sensores de otro tipo, como por ejemplo de humedad, es posible conectarlos siempre que cumplan con los requerimientos del bus 1-Wire.
- Aunque no esté contemplado por el software, la resolución del sensor entre los 0ºC y los +85ºC es de 0.5ºC. Esto no fue implementado por dos cuestiones, por un lado debido al escaso tiempo de desarrollo y por otro debido a que los límites no tienen en cuenta esta resolución, dado que al utilizar un bit menos que la medición descartan la parte decimal.
- El tipo de control de este dispositivo es de encendido-apagado, pero como el software puede modificarse es posible agregarle la opción de seleccionar el tipo de control.

8. Esquemas de Hardware




Conexión al Display
Fuente de Alimentación
J2
CONN PCB 7x2-R
U4
VHI20032A
R4
U2
MC7805
5.6V
0.1uF
10uF
1N4007
J5
CONN PWR
VCC
Title
<Title>
Size
A
Document Number
<Doc>
Rev
Date: Thursday, January 08, 2004
Sheet 1 of 1

9. Conclusiones

Al inicio del proyecto, el objetivo principal planteado fue profundizar los conocimientos sobre microcontroladores, como así también su conexión con periféricos externos, y se puede decir que fue cumplido, ya que se logró hacer funcionar el dispositivo como fue planeado, algunas cosas con mayor o menor dificultad, pero sobrepasando los obstáculos sin mayores inconvenientes. Algunas veces con ayuda, y otras por cuenta propia.

El otro objetivo principal era comprender e implementar la tecnología 1-Wire, esto también fue cumplido, y todas las funciones implementadas en el dispositivo funcionan perfectamente. Además, el código fuente está disponible para su reutilización en futuros proyectos, facilitando así el desarrollo.

Más allá de los objetivos, hay algunas cosas que pueden hacerse notar luego de haber finalizado por completo el proyecto, como por ejemplo un pequeño análisis de los tiempos de desarrollo.

Contrariamente a lo supuesto al comenzar el proyecto, la mayor parte del tiempo fue consumida por el desarrollo del software del microcontrolador (aproximadamente cuatro semanas). Y en mucho menor medida el diseño y armado de la placa prototipo del termómetro digital (entre una y dos semanas).

La ventaja que se tiene ahora para realizar un desarrollo futuro es que al estar disponible para su reutilización el código fuente, los tiempos serán mucho menores, y solo requerirá tiempo extra la implementación de opciones avanzadas de la tecnología 1-Wire.

10. Contenido del CD

- Tesina
  - Código Fuente
    - tesina.Uv2
    - (Archivos necesarios para el proyecto)
  - Esquemas
    - Esquemas de Hardware (Orcad)
    - Diagramas de Estados (Visio)
  - Fotos
    - Vista del display
    - Vista del micro
    - Vista frontal
    - Vista superior
    - Vista trasera
  - Hojas de Datos
    - AT89S8252 (Micro)
    - DS18S20 (Sensor)
    - LM78L05 (Regulador)
    - WH2002A - Driver (Display)
    - WH2002A - Info (Display)
    - WH2002A (Display)
  - Termómetro digital con control de temperatura.doc
  - Termómetro digital con control de temperatura.pdf

11. Bibliografía

Hoja de datos Microcontrolador Atmel AT89S8252 (pdf)
Hoja de datos Sensor de Temperatura 1-Wire DS18S20 (pdf)
Hoja de datos Display Winstar WH2002A (pdf)
Hoja de datos Regulador de Tensión 7805 (pdf)
Notas de Aplicación de tecnología 1-Wire
http://www.maxim-ic.com/1-Wire.cfm

Universidad del País Vasco
http://scsx01.sc.ehu.es/sbweb/webcentro/automatica/web_8051/Contenido/tutor8051_52/indicetutor_8051.htm

8052.com
http://www.8052.com/

Electronics in Meccano
http://www.eleinmec.com/article.asp?16

Introducción al 8051 - Escuela Politécnica Superior Universidad de Huelva (pdf)

Anexo: Funciones Principales del Código Fuente

```
/*****************************/
/*          main()           */
/*****************************/

void main(void)
{
    inicializacion();
    while(ON)
    {
            wdDato |= DATOMAIN;
            if (hayTeclaPres)
            {
                    inParser = teclaPres;
                    Parser();
                    hayTeclaPres = OFF;
            }
            if(leerTemperatura)
            {
                    if(!ow_reset())
{
                            write_byte(SKIP_ROM);
                            write_byte(CONVERT_T);
                            while(!read_bit())
                            {
wdDato |= DATOMAIN;                                                 delay(3);
                            }
                            ow_reset();
                            write_byte(SKIP_ROM);
                            write_byte(READ_SCRATCHPAD);
                            temperatura = read_byte();
                            signo = read_byte();
                            if (control)
                            {
                                    ow_reset();
                                    write_byte(ALARM_SEARCH);
    alarmaSet = read_bit();
                                    delay(3);
                                    alarmaSet &= read_bit();
                                    delay(3);
                                    if (!alarmaSet)
                                    {
                                            CheckAlarma();
                                    }
                            }
                            CheckYMuestra();
                    }
                    else
                    {
                            MuestraStringF1(«Error:»);
                            MuestraStringF2(«Sensor NO Conectado!»);
                    }
                    leerTemperatura = OFF;
            }
    }
}
```

```
/*************************************************/
/*              Inicialización                   */
/*************************************************/

void inicializacion(void) using 0
{

    P0 = 0xFF;   //Port 0 como entrada.
    P1 = 0xFF;   //Port 1 como entrada.
    P2 = 0xFF;   //Port 2 como entrada.
    P3 = 0xFF;  //Port 3 como entrada.

    // Inicialización del timer
    TL0 = LOW(CUENTA);              // Cuenta del Timer 0
    TH0 = HIGH(CUENTA);

    TMOD = 0x21;        // Timer 0 Modo 1. y Timer 1 modo 2 «autoreload».
    TCON = 0x50;        // (TR0 y TR1) Start ambos timers.


    // WatchDog y Memory Control Register
    WMCON |= PS1_ | PS2_ | WDTEN_;
          //Si no se usa el watchdog está bien inicializado como amanece en la inicialización.
          //para habilitar el watchdog inicializar:
          //        PS2=0; PS1= 0; PS0= 0;//Prescaler =>001=32ms Timeout.
          //        WDTEN=0;//WatchDog Enable: 0:Inhibido, 1:Habilitado.
          //Inicializaciones del watchdog.
    wdDato = 0;

    // Inicializacion de estados
    estado = ESPERAOP;      //El estado inicial es esperar la opresion de una tecla
    hayTeclaPres = OFF;     //No hay teclas presionadas al comienzo
    estadoScan = OFF;
    contador = T1REP; //Tiempo de 1 seg para la primera repeticion.
    recarga = RECARGAINI;
    intervalo = recarga;
    leerTemperatura = OFF;
    unitsCFK = 'C';
    control = OFF;

    //Funciones de inicializacion
    ParserIni();
    ResetDisplay();
    SensorIni();
    CheckYMuestra();

//—Habilitación de interrupciones————————————————————

    ET0 = ON;           // Habilita interrupción del Timer 0.
    EA = ON;            // Habilita interrupciones en general.
}
```

```
/*******************************/
/*      Interrupciones         */
/*******************************/

void timer0(void) interrupt TIMER_INT0 using 1
{

    /***********************/
    /*  Recarga del Timer  */
    /***********************/
    IniTimer0Reg( CUENTA );  // IniTimer0Reg usa banco de registros 1.

    /***********************/
    /*      WatchDog       */
    /***********************/

    wdDato |= DATOTIMER;    //Actualiza variable del WatchDog.
    if ( wdDato == DATOWD )
    {
            WMCON |= WDTRST_;//Patea el watchdog.
            wdDato = 0;             //Limpia la variable para la próxima vez.
    }

    /***********************/
    /* Barrido del teclado */
    /***********************/

    if (estadoScan) {
            SCN0 = ON;
            SCN1 = OFF;
    } else {
            SCN0 = OFF;
            SCN1 = ON;
    }
    switch (estado) {
            case ESPERAOP:
                    if ((teclaPres =(P0 & MASKRET)) == MASKRET)     //Me fijo si hay tecla presionada
                            estadoScan = ~estadoScan;       //No hay ==> prepara proximo barrido
                    else
                            estado = VALIDAOP;
                    break;

            case VALIDAOP:
                    if ((P0 & MASKRET) == teclaPres){       //Me fijo si sigue tecla presionada
                            //Se validó la tecla.
                            teclaPres |= (P2 & MASKSCAN);
                            teclaPres >>= 1;
                            teclaPres &= MASKRET;
                            hayTeclaPres = ON;          //Avisa al main que hay tecla validada.

                            contador = T1REP;
                            estado = ESPERALIB; //Cambia de estado.
                    }else {
                            estadoScan = ~estadoScan;       //No hay ==> prepara proximo barrido
```

```
                        estado = ESPERAOP;
                }
                break;

        case ESPERALIB:
                if ((P0 & MASKRET) == MASKRET)     //Me fijo si liberó las teclas
                        estado = VALIDALIB;
                else{
                        if (contador == 0)
                                estado = ESPERAREP;
                        else
                                contador—;
                }
                break;

        case VALIDALIB:
                if ((P0 & MASKRET) == MASKRET) {   //Me fijo si liberó las teclas
                        //Se validó la liberacion.
                        estado = ESPERAOP;
                        estadoScan = ~estadoScan;
                } else
                        estado = ESPERALIB;
                break;

        case ESPERAREP:
                contador = TNREP;
                hayTeclaPres = ON;
                estado = ESPERALIB;
                break;
    }

    /*********************************/
    /* Contador para la temperatura */
    /*********************************/

    if (intervalo==0){
            intervalo = recarga;
            leerTemperatura = ON;
    }
    else
    {
            intervalo—;
    }
}
```

```
/*************/
/* Display.c */
/*************/

/*************/
/* DelayDisp */
/*************/
void DelayDisp(void)
{
    for( ; d > 0 ; d— )
            ;
}

/****************/
/* ResetDisplay */
/****************/
void ResetDisplay(void)
{
    static unsigned char data c;
    RW = ON;    // Líneas en estado normal.
    E = OFF;
```

(A partir de acá se produce una seguidilla de retrasos para respetar los tiempos de inicialización del Display)

```
    actualizaFila1 = OFF;
    actualizaFila2 = OFF;
    actualizaCursor = OFF;

}
/***************/
/* WriteDispF1 */
/***************/
void WriteDispF1(void)
{
    RS = OFF;
    RW = ON;
    do      // Espera leyendo el busy flag.
    {
            EA = OFF;           // Inhibe interrupciones.
            DATAPORT = 0xFF;        // Port como entrada.
            E = ON;
            c = DATAPORT;
            E = OFF;
            EA = ON;            // Habilita Interrupciones
    }while( c & 0x80 );
    RW = OFF;                   // Escritura
    E = ON;
    EA = OFF;                   // Inhibe interrupciones.
    DATAPORT = 0x80;            // Dirección de DDRAM de la primera fila.
    E = OFF;
    EA = ON;                    // Habilita Interrupciones
    for( i = 0; i<CANTCOLDISP; i++ )
    {
            RW = ON;    // Read.
            RS = OFF;   // Instrucción.
            do      // Espera leyendo el busy flag.
            {
```

```
                EA = OFF;               // Inhibe interrupciones.
                DATAPORT = 0xFF;        // Port como entrada.
                E = ON;
                c = DATAPORT;
                E = OFF;
                EA = ON;                // Habilita Interrupciones
        }while( c & 0x80 );
        RW = OFF;                               // Vuelve a escritura.                   RS = ON;
                // Datos
        E = ON;
        EA = OFF;               // Inhibe interrupciones.
        DATAPORT = strDispF1[i];        // Dirección de DDRAM
        E = OFF;
        EA = ON;                // Habilita Interrupciones
    }
}
/***************/
/* WriteDispF2 */
/***************/
void WriteDispF2(void)
{
    static unsigned char data i;
    static unsigned char data c;
    RS = OFF;
    RW = ON;
    do      // Espera leyendo el busy flag.
    {
        EA = OFF;               // Inhibe interrupciones.
        DATAPORT = 0xFF;                // Port como entrada.
        E = ON;
        c = DATAPORT;
        E = OFF;
        EA = ON;                // Habilita Interrupciones
    }while( c & 0x80 );
    RW = OFF;                   // Escritura
    E = ON;
    EA = OFF;               // Inhibe interrupciones.
    DATAPORT = 0xC0;        // Dirección de DDRAM de la primera fila.
    E = OFF;
    EA = ON;                // Habilita Interrupciones
    for( i = 0; i<CANTCOLDISP; i++ )
    {
        RW = ON;
        RS = OFF;                               // Instrucción.
        do      // Espera leyendo el busy flag.
        {
            EA = OFF;               // Inhibe interrupciones.
            DATAPORT = 0xFF;        // Port como entrada.
            E = ON;
            c = DATAPORT;
            E = OFF;
            EA = ON;                // Habilita Interrupciones
        }while( c & 0x80 );
        RW = OFF;                               // Vuelve a escritura.
        RS = ON;                                // Datos
        E = ON;
        EA = OFF;               // Inhibe interrupciones.
```

```
            DATAPORT = strDispF2[i];        // Dirección de DDRAM
            E = OFF;
            EA = ON;                        // Habilita Interrupciones
    }
}

/******************/
/* StrCpyDispF1/2C */
/******************/
void StrCpyDispF1/2C(unsigned char code *strPtr)
{
    static unsigned char data i;
    i = 0;
    while( (strPtr[i] != 0) && (i < (unsigned char)CANTCOLDISP) )
    {
            strDispF1/2[i] = strPtr[i];
            i++;
    }
    while( i < (unsigned char)CANTCOLDISP )
            strDispF1/2[i++] = ' ';
}

/******************/
/* StrCpyDispF1/2i */
/******************/
void StrCpyDispF1/2i(unsigned char data *strPtr, unsigned int index)
{
    static unsigned char data i;
    static unsigned char data j;
    i = index;
    j = 0;
    while( (strPtr[j] != 0) && (i < CANTCOLDISP) )
    {
            strDispF1/2[i] = strPtr[j];
            i++;
            j++;
    }
    while( i < CANTCOLDISP )
            strDispF1/2[i++] = ' ';
}
```

```
/*************/
/* DS18S20.c */
/*************/

/*************/
/*   DELAY   */
/*************/
void delay(int useconds)
{
    int data s;
    for (s=0; s<useconds;s++);
}

/*************/
/* SensorIni */
/*************/
void SensorIni(void)
{
    ow_reset();
    write_byte(SKIP_ROM);  //Envía comando SKIP ROM.
    write_byte(READ_SCRATCHPAD); // Leo el scratch
    temperatura = read_byte();
    signo = read_byte();
    alarmaH = read_byte();
    alarmaL = read_byte();
}

/*************/
/* OW_RESET  */
/*************/
unsigned char ow_reset(void)
{
    unsigned char data presence;
    EA = OFF;    //Deshabilito int. para permitir tiempos exactos
    DQ = 0;      //pull DQ line low
    delay(36);   // leave it low for 480us
    DQ = 1;      // allow line to return high
    delay(4);    // wait for presence
    presence = DQ; // get presence signal
    delay(32);   // wait for end of timeslot
    EA = ON;     //Habilito interrupciones
    return(presence); // presence signal returned
} // 0=presence, 1 = no part

/*************/
/* READ_BIT  */
/*************/
bit read_bit(void)
{
    unsigned char data i;
    EA = OFF;           //Deshabilito int. para permitir tiempos exactos
    DQ = 0;                     // pull DQ low to start timeslot
    DQ = 1;                     // then return high
    for (i=0; i<4; i++);// delay 15us from start of timeslot
    EA = ON;
    return(DQ);         // return value of DQ line
}
```

```
/*************/
/* WRITE_BIT */
/*************/
void write_bit(char bitval)
{
    EA = OFF;   //Deshabilito interrupciones para permitir tiempos exactos
    DQ = 0;             // pull DQ low to start timeslot
    if(bitval==1)
        DQ =1;              // return DQ high if write 1

    delay(5);           // hold value for remainder of timeslot
    DQ = 1;
    EA = ON;
}// Delay provides 13us per loop, plus 19.5us. Therefore delay(5) = 84.5us

/*************/
/* READ_BYTE */
/*************/
unsigned char read_byte(void)
{
    unsigned char data i;
    unsigned char data value = 0;

    for (i=0;i<8;i++)
    {
        if(read_bit())
            value|=1<<i;  // reads byte in, one byte at a time and then
                          // shifts it left
        delay(3); // wait for rest of timeslot
    }
    return(value);
}

/**************/
/* WRITE_BYTE */
/**************/
void write_byte(char val)
{
    unsigned char data i;
    unsigned char data temp;
    for (i=0; i<8; i++)     // writes byte, one bit at a time
    {
        temp = val>>i;      // shifts val right 'i' spaces
        temp &= 0x01;       // copy that bit to temp
        write_bit(temp);// write bit in temp into
    }
}
```

```
/****************/
/*   Parser.c   */
/****************/

/****************/
/*  ParserIni   */
/****************/
void ParserIni(void)
{
    estadoActual = REPOSO;
    inParser = DEFAULT;
    MuestraReposo();
}
/****************/
/*    Parser    */
/****************/
void Parser(void)
{
    ptrEstadoParser = dirEstadoParser[estadoActual];
    while ( (ptrEstadoParser->entrada != inParser) &&
                    (ptrEstadoParser->entrada != DEFAULT) )
    {
            ptrEstadoParser++;// si no es la ent. buscada pasa a la otra.
    }
    estadoActual = ptrEstadoParser->proxEstado; //actualiza estado
    (*ptrEstadoParser->Accion) ();                    //realiza acción
}

/********************/
/*  CheckYMuestra   */
/********************/
void CheckYMuestra(void)
{
    if (estadoActual == REPOSO)
            MuestraReposo();
}

/**********************/
/*   MuestraReposo    */
/**********************/
void MuestraReposo(void)
{
    // Armado de la Fila 1
    StrCpyDispF1C(strReposo);
    alarmaHInt = AlarmToInt(alarmaH);
    IntToAscii(strASCII,alarmaHInt);
    strASCII[0] = '|';
    StrCpyDispF1i(strASCII,LIMITESF1);

    // Armado de la Fila 2 Parte 1
    StrCpyDispF2C(« «);
    temperaturaInt = TempToInt(temperatura,signo);
    IntToAscii(strASCII,temperaturaInt);
    if(control)
            strASCII[0] = '#';
    StrCpyDispF2i(strASCII,TEMPPOS1);

    // Armado de la Fila 2 Parte 2
```

```
    alarmaLInt = AlarmToInt(alarmaL);
    IntToAscii(strASCII,alarmaLInt);
    strASCII[0] = '|';
    StrCpyDispF2i(strASCII,LIMITESF2);
    WriteDispF2();
}

/*********************************/
/* Funciones que muestran menues */
/*********************************/
void MuestraMPUnidades(void)
void MuestraMPLimites(void)
void MuestraMPControl(void)
void MuestraMPMuestreo(void)
void MuestraMUC(void)
void MuestraMUF(void)
void MuestraMUK(void)
void MuestraMLSup(void)
void MuestraMLInf(void)
void MuestraMLSelSup(void)
void MuestraMLSelInf(void)
void MuestraMLGuardar(void)
void MuestraMCAct(void)
void MuestraMCDesact(void)
void MuestraMMInter(void)

/***********************************/
/* Funciones que ejecutan acciones */
/***********************************/
void SetLimites(void)
void IncLimiteSup(void)
void DecLimiteSup(void)
void IncLimiteInf(void)
void DecLimiteInf(void)
void SetUnitsC(void)
void SetUnitsF(void)
void SetUnitsK(void)
void SetControlAct(void)
void SetControlDesact(void)
void SetRecarga(void)
void IncRecarga(void)
void DecRecarga(void)
void GuardarLimites(void)
```