ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

"CONTROLADOR PID DE TEMPERATURA UTILIZANDO LA TARJETA DE DESARROLLO AVR BUTTERFLY"

TESINA DE SEMINARIO

PREVIA A LA OBTENCIÓN DEL TÍTULO DE:

INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

PRESENTADO POR:

GONZÁLEZ RUIZ JOSÉ ARMANDO

LECARO JARRÍN ANDRÉS RAÚL

GUAYAQUIL – ECUADOR

AÑO 2011

AGRADECIMIENTO

Doy gracias a Dios en primer lugar por haberme permitido culminar de manera exitosa mis estudios superiores. A mis padres que con sus sabios consejos y apoyo incondicional supieron guiarme para alcanzar este gran objetivo. A mis maestros por sus conocimientos impartidos durante la carrera. Y a mi esposa y a mi hijo que han sido el principal motivo para seguir adelante.

José González Ruiz

Agradezco a Dios por haberme permitirme culminar mis estudios superiores y a mis padres por darme ese apoyo incondicional.

Andrés Lecaro Jarrín

DEDICATORIA

A mis padres y hermanos, a mi esposa y amado hijo por estar siempre brindándome esa confianza.

José González Ruiz

A mis padres y hermanos porque siempre estuvieron inculcándome por el camino del saber.

Andrés Lecaro Jarrín

TRIBUNAL DE SUSTENTACION

Ing. Carlos Valdivieso A.
PROFESOR DEL SEMINARIO DE GRADUACIÓN

Ing. Hugo Villavicencio
PROFESOR DELEGADO DEL DECANO

DECLARACIÓN EXPRESA

“La responsabilidad del contenido de este proyecto de graduación nos corresponden exclusivamente; y el patrimonio intelectual de la misma a la ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL”

(Reglamento de exámenes y títulos profesionales de la ESPOL)

_____

José González Ruiz

_____

Andrés Lecaro Jarrín

RESUMEN

El presente proyecto de graduación expone el diseño e implementación de un controlador PID de temperatura utilizando la tarjeta AVR Butterfly demostrando una de las diversas aplicaciones que tiene la tarjeta como en el caso de los diseños de control digital.

Inicialmente se describe los antecedentes de los controladores de temperatura en especial y cómo en la actualidad la tecnología ha evolucionado para dar paso al diseño de los controladores digitales de temperatura para cualquier sistema de control.

Luego de haber realizado una breve reseña de este tipo de controladores, se detalla las herramientas a utilizar tanto de software como de hardware que se necesitará para la realización del proyecto en curso. Esto es el uso del microcontrolador ATmega169 de la tarjeta AVR Butterfly, la respectiva configuración de puertos a usar tantos de entrada como de salida; el uso de la pantalla LCD de la tarjeta para la visualización de la temperatura deseada (Set Point) y la temperatura actual de la planta; las interrupciones externas mediante el joystick de la tarjeta que permitirá ingresar la temperatura deseada y también activar el control PID para la planta. Se empleará un sensor de temperatura LM35 que estará midiendo continuamente la temperatura de la planta. La planta a controlar será una simple bombilla de 12V/50W. Además se proyectará mediante Isis Proteus las respectivas simulaciones.

Finalmente se muestran los resultados obtenidos de las pruebas de funcionamiento del sistema a lazo cerrado por lo que se procede al respectivo análisis para concretar las conclusiones y recomendaciones del uso de este tipo de controlador digital de temperatura.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

GLOSARIO

AVR Advanced Virtual RISC, RISC Virtual Avanzado. También significa Alf Vergard RISC, en honor a sus creadores Alf Egil Bogen y Vergard Wollan.

CI Circuito Integrado que contiene un conjunto de dispositivos que generan salidas en base al voltaje aplicado en sus entradas.

Código fuente Archivo de texto que es procesado por un lenguaje ensamblador o un compilador para producir un archivo de objeto intermedio, o código de máquina que pueda ejecutarse en un microcontrolador.

Hardware Partes o componentes físicos que integran una herramienta; inclusive ella misma como una unidad.

HEX Tipo de archivo hexadecimal compuesto de registros que le especifican al microcontrolador datos para la memoria del programa.

IDE Integrated Development Environment, Ambiente Integrado de Desarrollo. Software usado para el Desarrollo de Proyectos con Microcontroladores.

ISP In System Programming. Característica que permite grabar o leer un dispositivo sin tener que extraerlo de su aplicación/sistema.

JTAG	Joint Test Action Group. Estándar IEEE 1149-1190 (Standard Test Access Port and Boundary-Scan Architecture). Se define como la electrónica que puede incorporar un CI para asistir en el test, mantenimiento y soporte de placas y circuito impreso. Esta circuitería incluye una interfaz estándar a través de la cual se transfieren datos y comandos con los que el componente puede responder a un conjunto mínimo de instrucciones.
LCD	Liquid Crystal Dysplay, Pantalla de Cristal de Líquido. Están formadas por dos filtros polarizantes con filas de cristales líquidos alineados perpendicularmente; aplicando una corriente eléctrica a los filtros se consigue que la luz pase o no dependiendo de que lo permita o no el segundo filtro. Si se intercalan tres filtros adicionales de color, uno por cada color primario, se obtienen pantallas que reproducen imágenes en color.
Microcontrolador	Computador de limitadas prestaciones que está contenido en el chip de un CI programable y que destina a gobernar una sola tarea con el programa que reside en su memoria. Sus líneas de entrada/salida soportan el conexionado de los sensores y actuadores del sistema a controlar.
Programador	Dispositivo electrónico que permite cargar un programa al microcontrolador

INTRODUCCIÓN

En la actualidad el avance tecnológico ha dado paso a la introducción de nuevos sistemas de control incorporando tecnología digital para la resolución de los diversos tipos de control, hablando específicamente de los microcontroladores que en sí se han proyectado a todos los campos de aplicación. De aquí que sin menospreciar el trabajo que aún realizan los controladores de tipos analógicos, los tipos de controladores digitales han sido de gran utilidad para optimizar recursos de hardware en especial.

Dado el amplio uso de los controladores PID en diversos procesos de control, el uso de microcontroladores para el desarrollo de este tipo de aplicaciones ha tomado fuerza gracias a la incorporación de lenguajes de alto nivel que facilitan ampliamente este tipo de implementaciones, además de los bajos costos de adquisición de estos dispositivos, distribución de software de desarrollo gratuito y amplia información en la Internet.

Los controladores de temperatura constituyen una de las aplicaciones de este tipo de controlador como diseño de los sistemas de control y de mucha utilidad en diferentes campos de aplicación. Las soluciones actuales con otro tipo de microcontroladores de otra familia han optimizado en su mayoría los recursos que en un sistema de control analógico de temperatura se requería.

El presente proyecto tiene como propósito mostrar al lector un método de diseño práctico y sencillo en el desarrollo de controladores digitales PID implementados en microcontroladores AVR de la gran familia ATMEL. Una configuración mucho más integrada del diseño de un controlador de temperatura, manejable y asequible por los usuarios y sencillo de manejar. Por ende hemos trazado los siguientes objetivos:

- Diseñar e implementar un sistema de control PID de temperatura utilizando un microcontrolador avanzado como lo es el Kit AVR Butterfly que integra la mayoría de componentes de hardware a utilizar.
- Utilizar el método de sintonización de la curva de reacción de Ziegler-Nichols para el cálculo de parámetros y/o constantes del control PID.
- Diseñar un controlador PID digital basado en el microcontrolador ATmega169.
- Implementar la etapa de fuerza con un MOSFET cuyo *gate* maneje señales digitales.

CAPÍTULO 1

1. DESCRIPCIÓN GENERAL DEL PROYECTO

1.1. ANTECEDENTES

A partir de la era de la automatización, el hombre ha implementado varios controles automáticos, uno de ellos es el controlador de temperatura. En primera instancia se logró implementar controladores de tipo analógicos. Años más tarde y con la tecnología digital surgen los microcontroladores y he aquí la implementación de los controladores digitales de temperatura. A continuación se cita algunos desarrollos de la mano del hombre sobre los controladores de temperatura.

Alrededor de 1624, J.Kepler desarrolló un sistema de control automático de temperaturas para un horno, motivado por su creencia, basada en que los metales podrían transformase en oro manteniéndolos a una temperatura exactamente constante durante largos períodos de tiempo. También se usó este regulador de temperatura en una incubadora para pollos.

No fue hasta 1777, que se desarrolló un regulador conveniente de temperatura para el uso industrial por Bonnemain, quien lo utilizó para una incubadora. Su dispositivo fue instalado más adelante en el horno de una planta de calefacción de agua caliente.

Posteriormente, los controladores de temperatura cubrieron otros campos de aplicación en los que respecta a plantas de refrigeración como las que utilizan las refrigeradoras domésticas, las cámaras de refrigeración, frigoríficos, etc.

1.2. DESCRIPCIÓN DEL PROYECTO

Nuestro proyecto a realizar es un controlador PID de temperatura utilizando la tarjeta AVR Butterfly, que es un kit de desarrollo de los microcontroladores de la familia ATMEL. Su importancia radica en que nos permite diseñar un controlador digital de temperatura usando el microcontrolador Atmega169 y que incorpora todas las funciones necesarias para el diseño de control.

La mayoría de los recursos que se necesita para desarrollar nuestro proyecto ya viene incorporado en dicha tarjeta, tales como: el microcontrolador Atmega169 que es el cerebro del proyecto; el joystick

que permitirá a que el usuario pueda establecer la temperatura deseada mediante interrupciones externas; y la pantalla LCD que permitirá visualizar el comportamiento del controlador PID. Se requerirá un circuito externo de fuerza para controlar nuestra planta. La planta a controlar que utilizaremos será una bombilla eléctrica de 12V/50W.

Uno de los objetivos para el proyecto que se quiere realizar es lograr posicionar la temperatura deseada para que así, el error de estado estacionario sea cero. Otro requerimiento es que la temperatura deseada alcance muy rápidamente su posición final. En este caso, se desea disminuir el tiempo de establecimiento para que sea mínimo y tenga un sobrepaso considerable.

1.3. LIMITACIONES OPERACIONALES

Nuestro equipo como tal tiene limitaciones predeterminadas. En nuestro caso, las temperaturas a controlar serán por encima de la temperatura ambiente ya que no contamos con sistema de ventilación para descender la temperatura. Por ende, el campo de aplicación estará perfilado para sistemas de control que requieran un rango de temperaturas por encima de la temperatura ambiente.

1.4. SITUACIÓN ACTUAL

Hoy en día y con los avances tecnológicos, los controladores digitales de temperatura han reemplazado parcialmente a los controladores analógicos sin desmerecer su amplio campo de aplicación. Los

argumentos para cambiar de tecnología son el bajo costo de los circuitos digitales y el reducido espacio que estos ocupan.

Un sistema analógico tiene muchas limitaciones que los sistemas digitales permiten solventar; por ejemplo, un sistema analógico debe respetar unos criterios de calidad que afectan a la transmisión de la señal. Como la señal transmitida debe ser una réplica análoga de la señal original, es necesario que esta forma no se distorsione.

A continuación se citarán y se explicarán el funcionamiento de tres ejemplos prácticos de las múltiples aplicaciones de un controlador de temperatura ya sea en el sector industrial, doméstico, médico, etc.

Se desea mantener la temperatura de un lugar determinado utilizando un aire acondicionado en su valor de referencia. Se debe tener un dispositivo de control de la temperatura (sistema de refrigeración), y un sensor de temperatura tal como se indica en la figura 1.1. El P, PI o PID irá controlando la variable (en este caso la temperatura). En el instante que esta no sea la correcta avisará al dispositivo de control de manera que esta actúe, corrigiendo el error. De todos modos, lo más correcto es utilizar un PID.

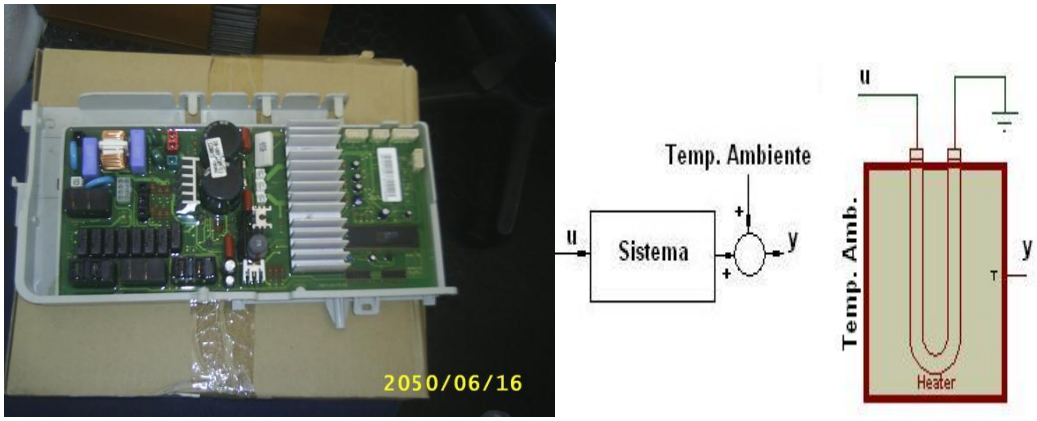


Fig. 1.1: Tarjeta controladora aire condicionado

La figura 1.2 muestra el dispositivo de controlador de temperatura para refrigeradores domésticos. Estamos familiarizados a este tipo de dispositivos ya que en la mayoría de los hogares disponemos de este artefacto doméstico y sin saber a veces que es un típico controlador de temperatura.

Fig. 1.2: Controladores para refrigeradores domésticos

En el sector médico, ciertos profesionales en la rama de la medicina hacen uso equipos especiales como las termocunas que sirve para mantener la temperatura que necesita un niño prematuro tal como si estuviere en el vientre de la madre. La figura 1.3 muestra un tipo de termocuna que es de vital utilidad en estos casos de riesgo.

Fig. 1.3: Incubadora Rarosch2 utiliza un controlador de temperatura

Podemos mencionar otras aplicaciones como por ejemplo: en procesos de plástico y máquinas de empaque, en procesos de fabricación de hornos y de alimentos, en la industria láctea para diferentes procesos de producción y sello, en tecnología de ambiente y climas como sistema de invernadero, en los calentadores de agua para duchas y en las piscinas térmicas, entre otras.

CAPÍTULO 2

2. BASES TÉCNICAS DEL PROYECTO

En el presente capítulo se indica la base teórica y el soporte técnico de hardware y software para la realización del proyecto en curso. Antes de mencionar dichos recursos de apoyo, es necesario conocer de manera básica el funcionamiento y las características del sistema de control PID.

2.1. CONTROL PID

El controlador PID (Proporcional, Integral y Derivativo) es un controlador realimentado cuyo propósito es hacer que el error en estado estacionario, entre la señal de referencia y la señal de salida de la planta, sea cero de manera asintótica en el tiempo, lo que se logra mediante el uso de la acción integral. Además el controlador tiene la capacidad de anticipar el futuro en muy cortos intervalos de tiempo a través de la acción derivativa que tiene un efecto predictivo sobre la salida del proceso.

Es interesante señalar que más de la mitad de los controladores industriales que se usan hoy en día utilizan esquemas de control PID o PID modificado. Los controladores PID analógicos, son principalmente de tipo hidráulico, neumático, electrónico, eléctrico o sus

combinaciones. En la actualidad, muchos de estos se transforman en formas digitales mediante el uso de microcontroladores.

A continuación se realizará una breve descripción de las características generales de cada una de las herramientas a utilizar.

2.2. HERRAMIENTAS DE SOFTWARE

Para desarrollar el proyecto hicimos uso de dos tipos de software: AVR Studio 4, cuyo fin es la programación del ATmega169 y Proteus que nos servirá para la simulación completa del proyecto.

2.2.1. AVR STUDIO 4

AVR Studio 4 es un Entorno de Desarrollo Integrado (IDE) para escribir y depurar aplicaciones AVR en el entorno de Windows 9x/Me/NT/2000/XP. Soporta varias de las fases por las cuales se atraviesa al crear un nuevo producto basado en un microcontrolador AVR.

Este software apoya al diseñador en el diseño, desarrollo, depuración y parte de la comprobación del proceso. Es actualizado continuamente y está disponible para descargarlo desde la siguiente página web "www.atmel.com". Además 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.

2.2.2. WINAVR

WinAVR es un conjunto de herramientas de desarrollo para microcontroladores RISC AVR de Atmel, basado en software de código abierto y compilado para funcionar en la plataforma Microsoft Windows. WinAVR incluye las siguientes herramientas:

− avr-gcc, el compilador de línea de comandos para C y C++.

− avr-libc, la librería del compilador que es indispensable para avr-gcc.

− avr-as, el ensamblador.

− avrdude, la interfaz para programación.

− avarice, la interfaz para JTAG ICE.

− avr-gdb, el depurador.

− Programmers Notepad, el editor.

− MFile, generador de archivo makefile.

Para obtener gratuitamente WinAVR, es necesario visitar la página Web http://sourceforge.net/projects/winavr y descargar la última versión de este software.

Para instalar WinAVR es necesario ejecutar el archivo descargado y aceptar las opciones predeterminadas durante la instalación.

2.2.3. PROTEUS

Proteus es un software de diseño electrónico desarrollado por Labcenter Electrónicas que consta de dos módulos: Ares e Isis y que incluye un tercer módulo opcional denominado Electra.

2.2.3.1. ISIS

Mediante este programa podemos diseñar el circuito que deseemos con componentes muy variados, desde una simple resistencia hasta algún que otro microprocesador o microcontrolador, incluyendo fuentes de alimentación, generadores de señales y muchas otras prestaciones. Los diseños realizados en Isis pueden ser simulados en tiempo real. Una de estas prestaciones es VSM, una extensión de la aplicación con la cual podremos simular, en tiempo real, todas las características de varias familias de microcontroladores, introduciendo nosotros mismos el programa que queramos que lleven a cabo. Se pueden simular circuitos con microcontroladores conectados a distintos dispositivos, como motores, lcd´s, displays, etc. El módulo VSM incluye, entre otras, las familias PIC10, PIC12, PIC16, PIC18, PIC24 y dsPIC33. ISIS es el corazón del entorno integrado PROTEUS. Es mucho más que un simple programa de dibujo de esquemas electrónicos. Combina un entorno de diseño de una potencia

excepcional con una enorme capacidad de controlar la apariencia final de los dibujos.

ISIS es la herramienta ideal para una rápida realización de complejos diseños de esquemas electrónicos destinados tanto a la construcción de equipos electrónicos como a la realización de tareas de simulación y prueba.

Además, encontrará en ISIS una herramienta excepcional para la realización de atractivos esquemas electrónicos destinados a su publicación en libros, manuales o documentos técnicos.

2.2.3.2. ARES

Ares es la herramienta de rutado de Proteus, se utiliza para la fabricación de placas de circuito impreso, esta herramienta puede ser utilizada de manera manual o dejar que el propio programa trace las pistas, aunque aquí podemos también utilizar el tercer módulo, Electra (Electra Auto Router), el cual, una vez colocados los componentes trazará automáticamente las pistas realizando varias pasadas para optimizar el resultado. Con el módulo Ares también podemos tener una visualización en 3D del PCB que se ha diseñado.

2.3. HERRAMIENTAS DE HARDWARE

Los componentes físicos que utilizamos para la realización del proyecto fueron: el Kit AVR Butterfly, el sensor de temperatura LM35, bombilla eléctrica de 12V/50W, transistor MOSFET, fuente de poder de 12V y cuatro pilas recargables AA.

A continuación mencionaremos las principales características del Kit AVR Butterfly y del sensor de temperatura.

2.3.1. KIT AVR BUTTERFLY

El Kit AVR Butterfly se diseñó para demostrar los beneficios y las características importantes de los microcontroladores ATMEL.

El AVR Butterfly utiliza el microcontrolador AVR ATmega169V, que combina la Tecnología Flash con el más avanzado y versátil microcontrolador de 8 bits disponible. En la siguiente figura se puede apreciar el Kit AVR Butterfly.

Fig. 2.1: Hardware disponible del Kit de desarrollo AVR Butterfly

2.3.1.1 CARACTERÍSTICAS DE LA TARJETA AVR BUTTERFLY

El Kit AVR Butterfly expone las siguientes características principales:

• La arquitectura AVR en general y la ATmega169 en particular.

• Diseño de bajo consumo de energía.

• El encapsulado tipo MLF.

• Periféricos:

- Controlador LCD.

- Memorias:

  Flash, EEPROM, SRAM.

  DataFlash externa.

• Interfaces de comunicación:

- UART, SPI, USI.

• Métodos de programación

- Self-Programming/Bootloader, SPI, Paralelo, JTAG.

• Convertidor Analógico Digital (ADC).

• Timers/Counters:

- Contador de Tiempo Real (RTC).

- Modulación de Ancho de Pulso (PWM).

El AVR Butterfly está proyectado para el desarrollo de aplicaciones con el ATmega169 y además puede usarse como un módulo dentro de otros productos.

Los siguientes recursos están disponibles en el Kit AVR Butterfly:

• Microcontrolador ATmega169V (en 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 la presión en el centro.

• Altavoz piezoeléctrico, para reproducir sonidos.

• Cristal de 32 KHz para el RTC.

• Memoria DataFlash de 4 Mbit, para el almacenar datos.

• Convertidor de nivel RS-232 e interfaz USART, para comunicarse con unidades fuera del Kit sin la necesidad de hardware adicional.

• Termistor de Coeficiente de Temperatura Negativo (NTC), para sensar y medir temperatura.

• Resistencia Dependiente de Luz (LDR), para sensar y medir intensidad luminosa.

• Acceso externo al canal 1 del ADC del ATmega169, para lectura de voltaje en el rango de 0 a 5 V.

• Emulación JTAG, para depuración.

• Interfaz USI, para una interfaz adicional de comunicación.

• Terminales externas para conectores tipo Header, para el acceso a periféricos.

• Batería de 3 V tipo botón (600mAh), para proveer de energía y permitir el funcionamiento del AVR Butterfly.

• Bootloader, para programación mediante la PC sin hardware especial.

• Aplicación demostrativa preprogramada.

• Compatibilidad con el Entorno de Desarrollo AVR Studio 4.

2.3.1.1. MICROCONTROLADOR ATMEGA169

El ATmega169 es un Microcontrolador de 8 bits con arquitectura AVR RISC, este 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 a 16 MHz.

• Memoria no volátil para Programa y Datos.

- Flash de 16 K bytes, auto-programable en el sistema.

Resistencia: 10 000 ciclos de 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.

- Operación Real de Lectura Durante la Escritura.

- EEPROM de 512 bytes.

- Resistencia: 100 000 ciclos de Escritura/Borrado.

- SRAM Interna de 1 Kbyte.

- Bloqueo de Programación para Seguridad del Software.

• Interfaz JTAG (conforme el Standard IEEE 1149.1)

− Capacidad de Boundary-Scan Acorde al Standard 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 1 de 5-bits.

− Controlador de LCD de 4 x 25 segmentos.

− Dos Temporizadores/Contadores de 8 bits con Preajustador (Prescaler) Separado y Modo de Comparación.

− Un Temporizador/Contador de 16 bits con Preajustador (Prescaler) Separado, 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 en el chip.

− Comparador Analógico.

− Interrupción y Salida del Modo de Sleep por Cambio en Pin.

• Características especiales del microcontrolador.

− Reset al Encendido y Detección 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 de Encapsulado

− 53 Líneas 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, – 8 MHz a 2.7 – 5.5 V.

− ATmega169: 0 – 8 MHz a 2. – 5.5 V, 0– 16 MHz a 4.5 – 5.5 V.

• Rango de Temperatura

− Desde -40 º C a 85 ºC.

• Consumo de Energía

− En el Modo Activo:

1 MHz, 1.8 V: 350 uA.

32 KHz, 1.8 V: 20 uA (incluyendo Oscilador).

32 KHz, 1.8 V: 40 uA (incluyendo Oscilador y LCD).

− En el Modo Power-Down:

0.1 uA a 1.8 V.

El AVR ATmega169 es compatible con un completo conjunto de programas y Herramientas de Desarrollo que incluye: Compiladores C, Ensambladores de Macro, Depurador/Simuladores de Programa, Emuladores de Circuito, Kits de Iniciación y Kits Evaluación.

En la figura 2.2 se observa el Diagrama de Bloques del Microcontrolador ATmega169.

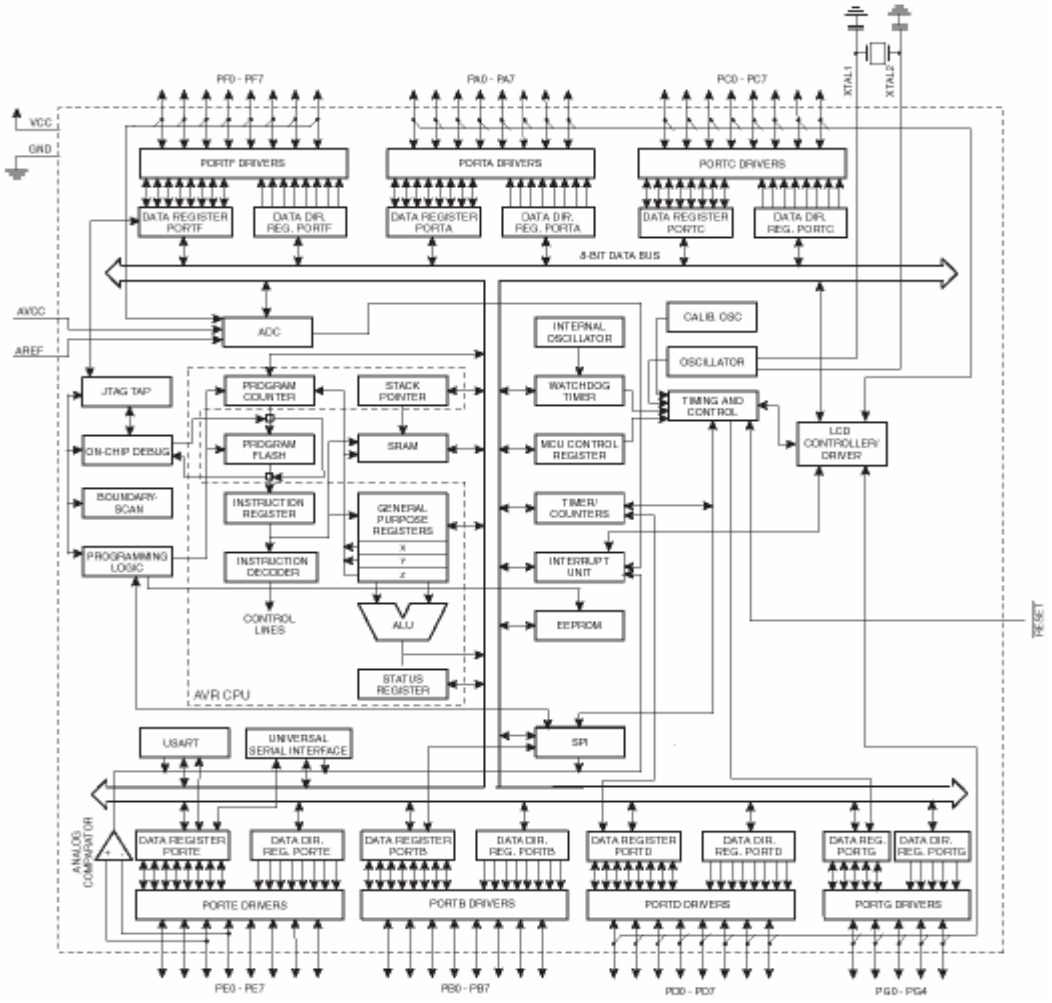


Fig. 2.2: Digrama de bloques del microcontrolador ATmega169

EL núcleo AVR combina un rico conjunto de instrucciones con 32 Registros de Trabajo de Propósito General (General Purpose Working Registers). Todos los 32 registros están conectados directamente a la Unidad de Lógica Aritmética (ALU), permitiendo que se acceda a dos registros independientes en una sola instrucción ejecutada en un ciclo de reloj. La arquitectura da como resultado código más eficiente mientras que el rendimiento alcanzado es diez veces más rápido que los convencionales microcontroladores CISC.

Este dispositivo se fabrica empleando tecnología Atmel de memoria no volátil de alta densidad. La Flash ISP en el Chip permite que la memoria de programa se reprograme en el sistema a través de una interfaz serial SPI, por un programador convencional de memoria no volátil ó por un programa de Arranque (Boot Program) residente en el Chip ejecutándose en el núcleo del AVR. El programa de Arranque puede usar cualquier interfaz (SPI, USART, UART) para descargar el programa de la aplicación en la memoria Flash de Aplicación. El Software en la sección Flash de Arranque continuará ejecutándose mientras la sección Flash de Aplicación esté actualizándose, proporcionando la operación real de Lectura durante la Escritura.

Al combinar una CPU RISC de 8 bits con una Flash Auto programable en el Sistema sobre un chip monolítico, el ATmega169 de Atmel se convierte en un microcontrolador poderoso que provee una solución altamente flexible y de efectividad de costo para muchas aplicaciones de control integrado.

2.3.1.2.1 CONFIGURACIÓN DE LOS PINES DEL MICROCONTROLADOR ATMEGA169

En la figura 2.3 se ilustra la distribución de pines del microcontrolador ATmega169V, en éste se aprecia el encapsulado MLF de 64 pines.

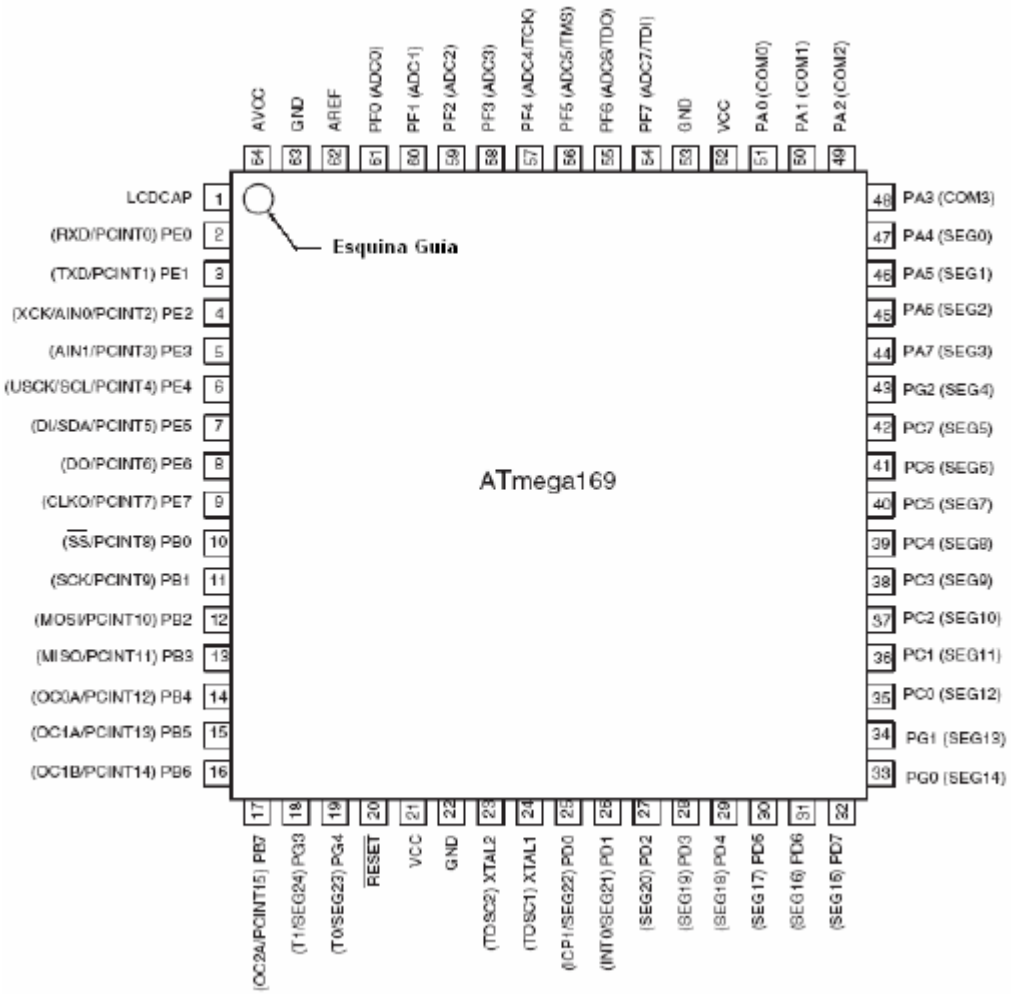


Fig. 2.3: Distribución de pines del microcontrolador ATmega169V

2.3.1.3 JOYSTICK

Para operar el AVR Butterfly se emplea el joystick como una entrada para el usuario. Este opera en cinco direcciones, incluyendo presión en el centro; tal como se puede ver en la figura 2.4.

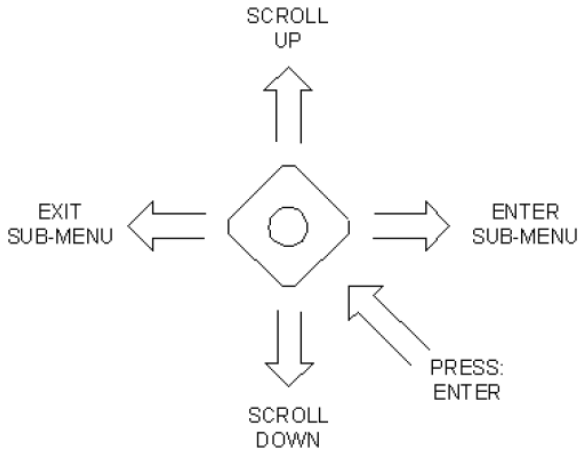


Fig. 2.4: Entrada tipo Joystick

2.3.1.4. CONECTORES

Algunos de los pines de I/O del microcontrolador ATmega169 están disponibles en los conectores del AVR Butterfly. Estos conectores son para comunicación, programación y entrada al ADC del ATmega169. En la figura siguiente se puede apreciar los conectores del AVR Butterfly.

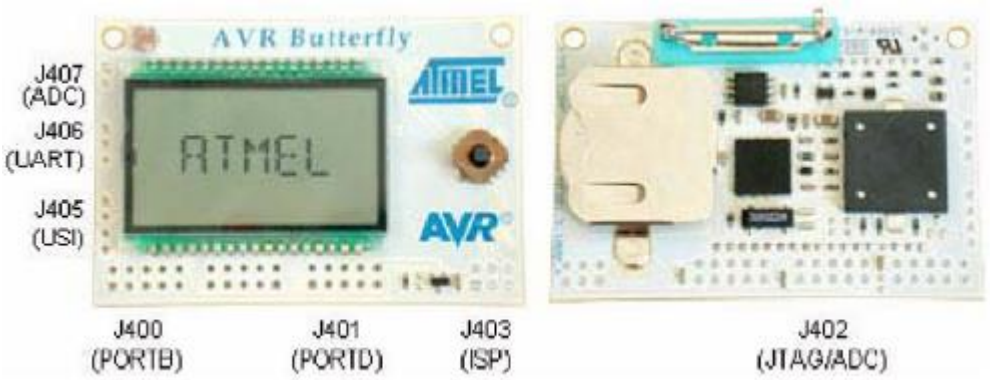


Fig. 2.5: Conectores del AVR Butterfly para acceso a periféricos.

2.3.1.5. EL LCD

En las aplicaciones donde es necesaria la interacción con el usuario es muy útil poder mostrar información para el usuario. Una interfaz muy simple para mostrar información podría ser el estado de unos LEDs; mientras que la interacción más compleja puede beneficiarse de una pantalla capaz de desplegar letras, números, palabras o incluso oraciones. Las Pantallas de Cristal Líquido (LCD) son frecuentemente usadas para desplegar mensajes. Los módulos LCD pueden ser gráficos y se los puede usar para desplegar gráficos y texto, ó pueden ser alfanuméricos capaces de visualizar entre 10 y 80 caracteres. Los módulos LCD alfanuméricos estándar son fáciles de conectar, pero son bastante costosos debido a que tienen incorporado drivers/controladores que se ocupan de la generación de los caracteres/gráficos sobre el vidrio LCD.

El vidrio LCD es la placa de vidrio en la cual el cristal líquido está contenido. Para reducir el costo de una aplicación donde

se requiere una pantalla se puede elegir usar una MCU que tenga un driver incorporado para LCD. La MCU puede entonces manejar directamente el vidrio LCD, eliminando la necesidad del driver integrado en el módulo LCD. El costo de la pantalla puede reducirse tanto como para un factor de 10, puesto que un vidrio LCD (LCD sin Driver) tiene un costo mucho más bajo que un módulo LCD (LCD con Driver).

El microcontrolador ATmega169 tiene un controlador LCD (LCD Driver) integrado capaz de controlar hasta 100 segmentos. El núcleo altamente eficiente y el consumo de corriente muy bajo de este dispositivo lo hace ideal para aplicaciones energizadas por batería que requieren de una interfaz humana.

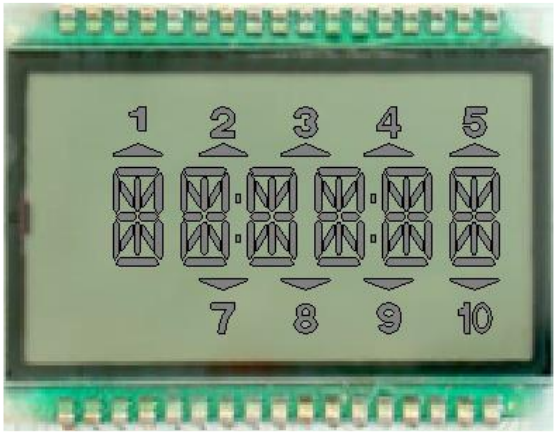


Fig. 2.6: Pantalla LCD

2.3.2. PROGRAMADOR DEL POLOLU

La programación de la tarjeta AVR Butterfly se lo hará por medio del ISP de la tarjeta (Fig. 3.7), utilizando su respectivo conector y el programador del POLOLU (Fig. 3.8). Cabe

recalcar que para la instalación del USB se necesita del driver *pgm03A* disponible en la web.

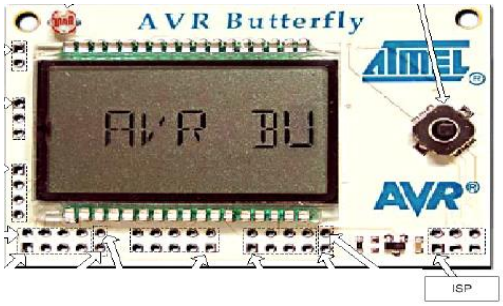


Fig. 2.7: Conector ISP para programación

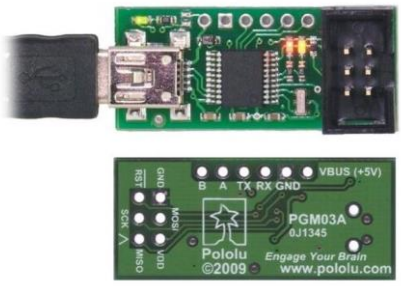


Fig. 2.8 Programador del POLOLU

2.3.3. SENSOR DE TEMPERATURA LM35

Para poder sensar la temperatura de la planta haremos uso del integrado LM35, el cual posee un factor de escala lineal de $10.0\,{}^{mV}\!/_{^\circ C}$.

Ahora, para obtener una relación temperatura – voltaje del sensor, de la siguiente manera:

$$1000mV \quad a \quad 100°C$$
$$600mV \quad a \quad 60°C$$
$$300mV \quad a \quad 30°C$$

El fabricante propone la siguiente configuración para calibrar el sensor:

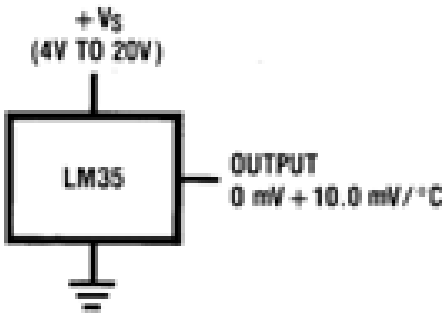


Fig. 2.9: Configuración del sensor de temperatura (+2ºC a +150ºC)

Como se puede ver en la fig. 3.9, esta configuración está calibrada para que el sensor de temperatura mida rangos de temperatura de +2ºC a 150ºC por lo que son temperaturas que están dentro del rango de operación del proyecto, pues decíamos que el sistema está diseñado para controlar temperaturas por encima de la temperatura ambiente de un clima tropical (para temperaturas negativas el fabricante propone otro tipo de configuración de conexión del sensor de temperatura).

CAPÍTULO 3

3. DISEÑO DE LA SOLUCION

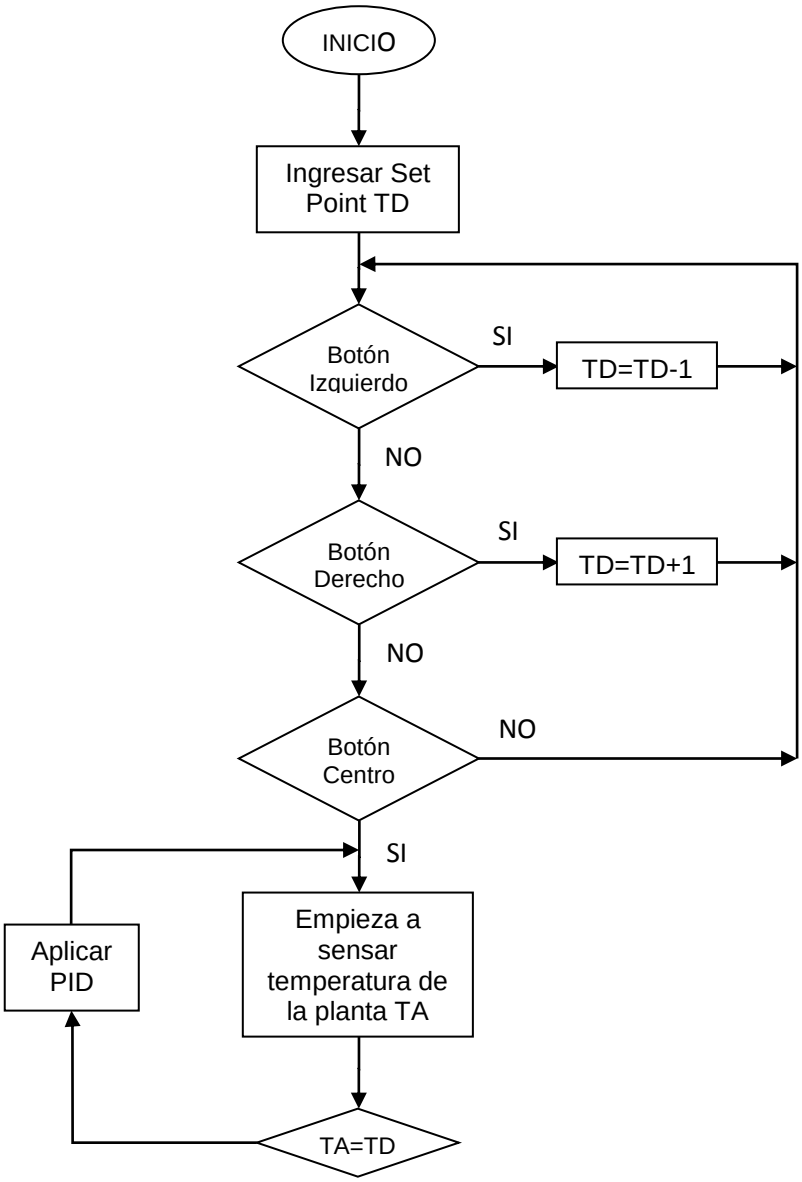


Fig. 3.1: Diagrama de flujo del sistema de control PID de temperatura.

A continuación se presenta el respectivo diagrama de bloques del proyecto por la que nos guiaremos para el diseño del mismo.

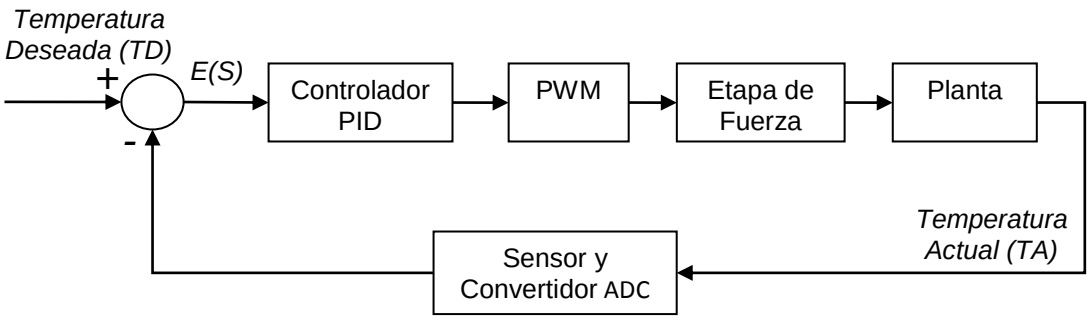


Fig. 3.2: Diagrama de bloques general del proyecto

El proyecto funcionará de la siguiente manera:

1. Al inicio, luego de encender la tarjeta por medio de un switch externo, se desplegará en pantalla la temperatura deseada a la cual se desea controlar la planta. Se ha programado una temperatura deseada inicial de $30°C$ para comodidad del usuario.
2. Después de haber fijado la temperatura deseada por medio del joystick, el controlador analiza la señal de error y envía una señal de tipo PWM al circuito de fuerza, el mismo que regulará la corriente que circula por la bombilla.
3. Luego de un tiempo determinado el sistema logrará estabilizarse con un error de estado estable de $\pm 1°C$.

3.1 ETAPAS DEL DISEÑO

Para el diseño del proyecto tomamos como consideración dividirlo por etapas y así ir diseñando y simulando paso a paso las diferentes partes del mismo.

3.1.1. CONFIGURACIÓN DE LA PANTALLA LCD

La primera etapa consiste en realizar un programa en el cual nos permita mostrar por pantalla los datos como ingresar la temperatura deseada y la temperatura actual de la planta, utilizando la pantalla LCD de la tarjeta AVR Butterfly. Este módulo se llamará LCD.h. Esta pantalla nos permite mostrar sólo seis dígitos, por lo tanto se mostrará la temperatura deseada (Fig. 3.3) y la temperatura actual (Fig. 3.4) de la siguiente forma:

Fig. 3.3: Temperatura Deseada

Fig. 3.4: Temperatura Actual

3.1.2. CONFIGURACIÓN DEL JOYSTICK

La segunda etapa consiste en ingresar mediante pulsadores la temperatura deseada, lo cual nos basamos en la utilización del joystick de la tarjeta. Si presionamos el pulsador izquierda la

temperatura deseada decrementará 1 unidad y si presionamos derecha incrementará 1 unidad. Los pulsadores arriba y abajo no serán de utilidad para nuestro proyecto. Para empezar a sensar la temperatura se procederá a presionar el pulsador centro. Este módulo se llamará Joystick.h

3.1.3. CONFIGURACIÓN DEL ADC PARA LA LECTURA DEL SENSOR DE TEMPERATURA LM35

La tercera etapa consiste en la conexión de un sensor externo a la tarjeta, el sensor que vamos a utilizar es el LM35 (Fig. 3.5), el cual entrega una señal análoga por lo que se tiene que usar Conversión Analógica Digital. El sensor se lo conectará a las terminales ADC de la tarjeta (Fig. 3.6). Este módulo se llamará ADC.h.

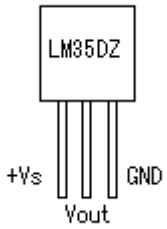


Fig. 3.5: Sensor de temperatura LM35

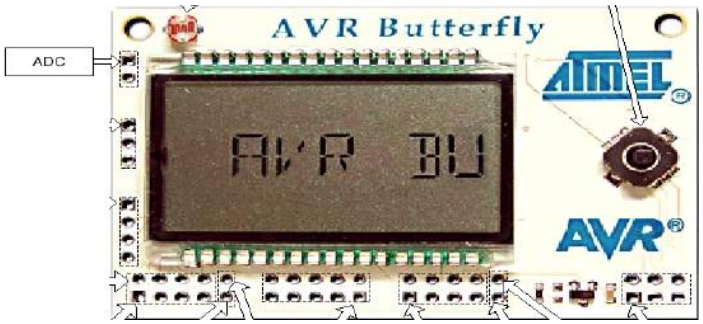


Fig. 3.6: Conexión del sensor

3.1.4. DISEÑO DE LA PLANTA

La cuarta etapa consiste en el diseño de la planta, para la simulación utilizaremos el dispositivo llamado OVEN de Proteus (Fig. 3.7).

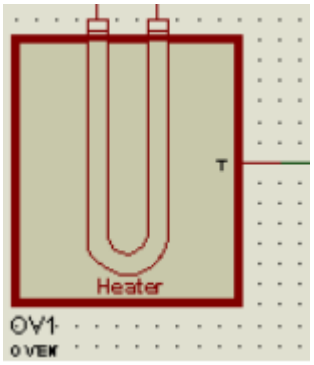


Fig. 3.7: Dispositivo OVEN de Proteus

3.1.5. DISEÑO DEL ALGORITMO PID

La quinta etapa consiste en diseñar el algoritmo PID para controlar la temperatura utilizando PWM para la señal de control, en esta etapa se necesita utilizar las variables del controlador PID que en el capítulo anterior fueron ya calculadas. Este módulo se llamará PID.h y para hallar los parámetros de este tipo de controlador se detallará paso a paso el diseño del algoritmo PID.

3.1.5.1. DISEÑO DEL CONTROLADOR PID

Se puede indicar que un controlador PID responde a la siguiente ecuación:

$$u(t) = K_p e(t) + \frac{K_p}{T_i}\int_0^t e(t)\partial t + K_p T_d \frac{\partial e(t)}{\partial t}$$

Donde $e(t)$ es el error de la señal y $u(t)$ es la entrada de control del proceso. K_p es la ganancia proporcional, T_i es la constante de tiempo integral y T_d es la constante de tiempo derivativa.

En el dominio de la frecuencia, el controlador PID se puede escribir como:

$$U(S) = K_p\left(1 + \frac{1}{T_i S} + T_d S\right)E(S)$$

3.1.5.2. SINTONIZACIÓN DE CONTROLADOR MEDIANTE ZIEGLER-NICHOLS

En lazo abierto, muchos procesos pueden definirse según la siguiente función de transferencia:

$$G(s) = \frac{K_0 e^{-s\tau_0}}{1 + \gamma_0 s}$$

Donde los coeficientes K_0, τ_0 y γ_0 se obtienen de la respuesta del sistema en lazo abierto a una entrada escalón. Se parte del sistema estabilizado en $y(t) = y_0$ para $u(t) = u_0$. Se aplica una entrada escalón de u_0 a u_1 (el salto debe estar entre un 10% y un 20% del valor nominal) y se registra la respuesta de la salida hasta que se estabilice en el nuevo punto de operación. Los parámetros se pueden obtener de la respuesta mostrada en la figura siguiente.

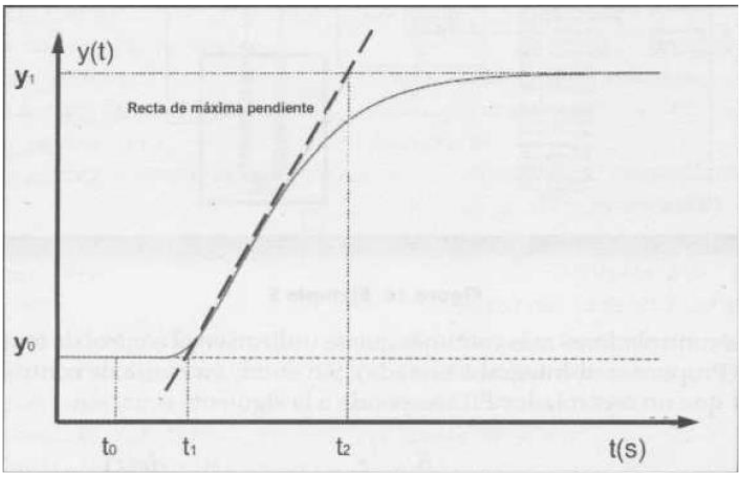


Fig. 3.8: Respuesta de salida ante una entrada escalón

Donde:

$$\tau_0 = t_1 - t_0$$

$$\gamma_0 = t_2 - t_1$$

$$k_0 = \frac{y_1 - y_0}{u_1 - u_0}$$

Según Ziegler-Nichols, la relación de estos coeficientes con los parámetros del controlador es la siguiente:

$$K_p = 1.2\frac{\gamma_0}{k_0\tau_0} \qquad T_i = 2\tau_0 \qquad T_d = 0.5\tau_0$$

3.1.5.3. CONTROLADOR DIGITAL PID

La función de transferencia para el controlador PID digital se convierte en:

$$U(z) = K_p\left[1 + \frac{T}{T_i(1 - z^{-1})} + T_d\frac{(1 - z^{-1})}{T}\right]E(z)$$

La función de transferencia discreta, también puede ser representada como:

$$\frac{U(z)}{E(z)} = a + \frac{b}{1 - z^{-1}} + c(1 - z^{-1})$$

Donde:

$$a = K_p \qquad b = \frac{K_p T}{T_i} \qquad c = \frac{K_p T_d}{T}$$

Existen distintas posibilidades de la realización práctica de un controlador PID, una de las más habituales es la realización en paralelo:

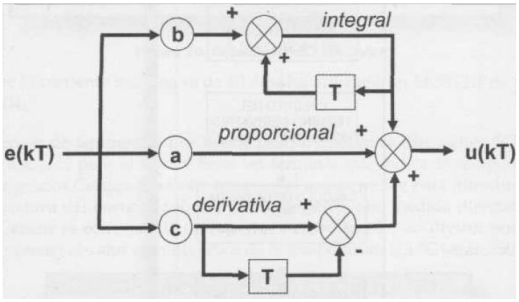


Fig.3.9: Diseño paralelo de controlador PID

3.1.5.4. ALGORITMO DE PROGRAMACIÓN EN EL MICROCONTROLADOR

El algoritmo utilizado para programar el microcontrolador se muestra en la siguiente figura. El muestreo (T) debe ser mayor que el tiempo de establecimiento del sistema en lazo abierto. En el modelo Ziegler-Nichols se toma un valor $T < \tau_0/4$.

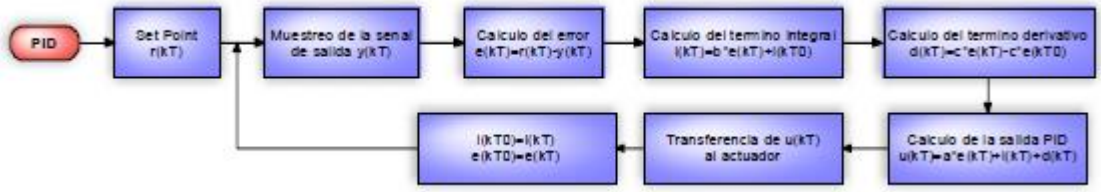


Fig. 3.10: Algoritmo de programación de PID digital en microcontrolador

Para la implementación y simulación del controlador PID se utilizara el software ISIS Proteus. Este software ofrece el modelo esquemático de un sistema de calefacción denominado OVEN, al cual se le pueden variar sus características funcionales tales como: Temperatura Ambiente de trabajo, resistencia térmica, constante de tiempo de establecimiento, constante de tiempo de calentamiento, coeficiente de temperatura y Potencia de calentamiento.

3.1.5.5. MODELO DEL SISTEMA DE CALEFACCIÓN

Para facilidades de simulación se establecerán los siguientes valores de parámetros funcionales del modelo OVEN:

- *Temperature Ambient (°C)*= 25
- *Thermal Resistence to Ambient (°C/W)*= 0.7
- *Oven Time Constant (sec)*= 10
- *Heater Time Constant (sec)*= 1
- *Temperature coefficient (V/°C)*= 1
- *Heating Power (W)*= 120

El modelo esquemático OVEN contiene un terminal sensor **T** que entrega un voltaje proporcional a la temperatura del sistema. De acuerdo a los parámetros establecidos anteriormente, este terminal entregara 1V/°C, es decir, que para una temperatura de 100°C, el terminal **T** entregara 100V.

Para obtener la respuesta del sistema en lazo abierto ante una entrada escalón (curva de reacción), se utiliza el sistema de análisis interactivo de ISIS Proteus *Interactive Analysis* (Graph Mode Tool), el cual interactúa con el sistema OVEN mediante el uso de un *Voltage Probe*1 OV1(T), según se muestra en la Figura 3.11. Observar que para realizar la entrada escalon de 0V a 2V se utiliza un interruptor SW1.

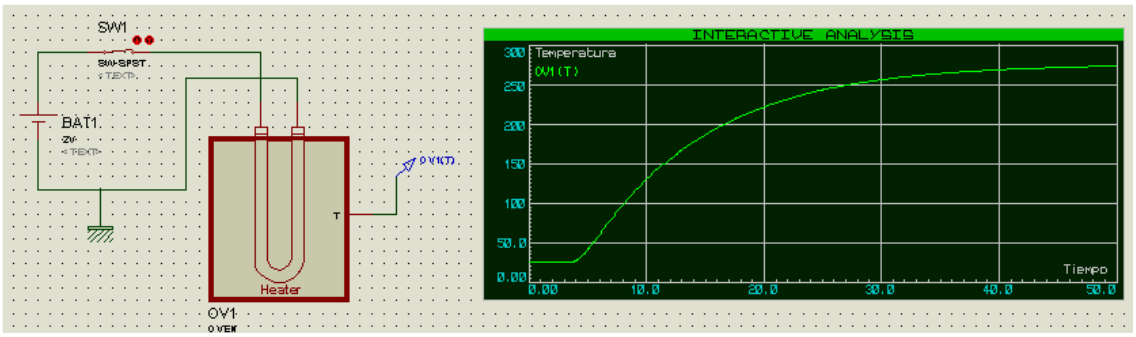


Fig. 3.11: Esquemático para análisis de respuesta ante entrada escalón

De la recta de máxima pendiente se deducen los parámetros τ_0, γ_0 y k_0 definidos por el análisis en lazo abierto de Ziegler-Nichols.

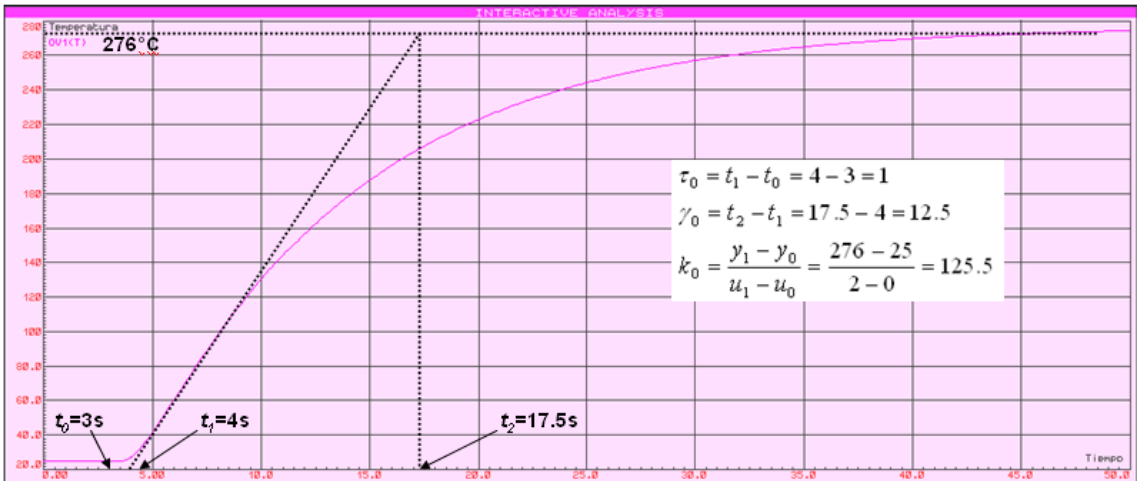


Fig. 3.12: Determinación de los parámetros por método de curva de reacción

Por tanto el modelo del sistema de calefacción queda definido así:

$$G(s) = \frac{K_0 e^{-s\tau_0}}{1 + \gamma_0 s} = 125.5 \frac{e^{-s}}{1 + 12.5s}$$

Los parámetros K_p, T_i y T_d se calculan según la Regla de Sintonización de Ziegler- Nichols basada en la respuesta al escalón:

$$K_p = 1.2 \frac{\gamma_0}{k_0 \tau_0} = 0.1195 \qquad T_i = 2\tau_0 = 2 \qquad T_d = 0.5\tau_0 = 0.5$$

Reemplazando los valores de K_p, T_i y T_d en las ecuaciones dadas anteriormente, y estableciendo un periodo de muestreo $T = 0.1s$ según criterio $T < \tau_0/4$, los parámetros del controlador discreto son:

$$a = K_p = 0.1195 \qquad b = \frac{K_p T}{T_i} = 0.0062 \qquad c = \frac{K_p T_d}{T} = 0.6215$$

Estos son los valores que utilizaremos cuando programemos el PID en el siguiente capítulo.

CAPÍTULO 4

4. IMPLEMENTACIÓN Y SIMULACIÓN

4.1. IMPLEMENTACIÓN

En el presente capítulo ilustraremos la codificación de los respectivos módulos descritos en el capítulo anterior tales como LCD.H, JOYSTIK.H, ADC.H, PID.H y el código principal CONTROLADOR_TEMPERATURA.C.

4.1.1. LCD.h

```
#include <avr/signal.h>
#define pLCDREG ((unsigned char *)(0xEC))
#define TAMANIO_DEL_REGISTRO_LCD 20
#define NUMERO_MAXIMO_DE_CARACTERES 36
char LCD_Data[TAMANIO_DEL_REGISTRO_LCD];
char
memo_temp_texto[NUMERO_MAXIMO_DE_CARACTERES];
unsigned char ESCRITURA_DE_CADENA_HABILITADO = 0;
unsigned char LCD_INT_contador = 0;
void inicializar_LCD(void);
```

```
void escribir_caracter_en_LCD(char , char );
void escribir_palabras_en_LCD(char *);
void borrar_LCD(void);
void actualizar_LCD(void);
unsigned int tabla_de_caracteres_LCD[] PROGMEM =
{
        0x0A51, // '*' (?)
        0x2A80, // '+'
        0x0000, // ',' (Sin definir)
        0x0A00, // '-'
        0x0A51, // '.' Signo de grados
        0x0000, // '/' (Sin definir)
        0x5559, // '0'
        0x0118, // '1'
        0x1e11, // '2
        0x1b11, // '3
        0x0b50, // '4
        0x1b41, // '5
        0x1f41, // '6
        0x0111, // '7
        0x1f51, // '8
        0x1b51, // '9'
        0x0000, // ':' (Sin definir)
        0x0000, // ';' (Sin definir)
        0x0000, // '<' (Sin definir)
        0x0000, // '=' (Sin definir)
        0x0000, // '>' (Sin definir)
        0x0000, // '?' (Sin definir)
        0x0000, // '@' (Sin definir)
        0x0f51, // 'A' (+ 'a')
```

```
0x3991, // 'B' (+ 'b')
0x1441, // 'C' (+ 'c')
0x3191, // 'D' (+ 'd')
0x1e41, // 'E' (+ 'e')
0x0e41, // 'F' (+ 'f')
0x1d41, // 'G' (+ 'g')
0x0f50, // 'H' (+ 'h')
0x2080, // 'I' (+ 'i')
0x1510, // 'J' (+ 'j')
0x8648, // 'K' (+ 'k')
0x1440, // 'L' (+ 'l')
0x0578, // 'M' (+ 'm')
0x8570, // 'N' (+ 'n')
0x1551, // 'O' (+ 'o')
0x0e51, // 'P' (+ 'p')
0x9551, // 'Q' (+ 'q')
0x8e51, // 'R' (+ 'r')
0x9021, // 'S' (+ 's')
0x2081, // 'T' (+ 't')
0x1550, // 'U' (+ 'u')
0x4448, // 'V' (+ 'v')
0xc550, // 'W' (+ 'w')
0xc028, // 'X' (+ 'x')
0x2028, // 'Y' (+ 'y')
0x5009, // 'Z' (+ 'z')
0x0000, // '[' (Sin definir)
0x0000, // '\' (Sin definir)
0x0000, // ']' (Sin definir)
0x0000, // '^' (Sin definir)
0x0000 // '_'
```

```
};
void inicializar_LCD(void)
{
        borrar_LCD();
        LCDCRA = (1<<LCDEN) | (1<<LCDAB);
        LCDCCR                                                  =
        (1<<LCDDC2)|(1<<LCDDC1)|(1<<LCDDC0)|(1<<LCDCC
3)|(1<<LCDC         C2)|(1<<LCDCC1)|(1<<LCDCC0);
        ASSR = (1<<AS2);
        LCDFRR = (0<<LCDPS0) | (1<<LCDCD1)|(1<<LCDCD0);
        LCDCRB                                                  =
        (1<<LCDCS)|(1<<LCDMUX1)|(1<<LCDMUX0)|(1<<LCDP
M2)|(1<<LC  DPM1)|(1<<LCDPM0);
        LCDCRA |= (1<<LCDIE);
}
void escribir_caracter_en_LCD(char c, char posicion)
{
        unsigned int seg = 0x0000;
        char mascara, nibble;
        char *ptr;
        char i;
        if (posicion > 5) return;
        if ((c >= '*') && (c <= 'z'))
        {
                if (c >= 'a') c &= ~0x20;
                c -= '*';
                seg=(unsigned int)
        pgm_read_word(&tabla_de_caracteres_LCD[(uint8_t)c]);
        }
        if (posicion & 0x01)
```

```
        mascara = 0x0F;
        else
        mascara = 0xF0;
        ptr = LCD_Data + (posicion >> 1);
        for (i = 0; i < 4; i++)
        {
                nibble = seg & 0x000F;
                seg >>= 4;
                if (posicion & 0x01)
                nibble <<= 4;
                *ptr = (*ptr & mascara) | nibble;
                ptr += 5;
        }
}
void escribir_palabras_en_LCD(char *palabra)
{
        unsigned char i=0;
        for( i=0;i<NUMERO_MAXIMO_DE_CARACTERES;i++)
        memo_temp_texto[i]='\0';
        LCD_INT_contador = 0;
        ESCRITURA_DE_CADENA_HABILITADO = 1;
        for(
        i=0;(i<NUMERO_MAXIMO_DE_CARACTERES)&&(*pala
bra!='\0');i++,palabra++)
        memo_temp_texto[i]=*palabra;
}
void borrar_LCD(void)
{
        unsigned char i=0;
        for( i=0;i<NUMERO_MAXIMO_DE_CARACTERES;i++)
```

```
        memo_temp_texto[i]='\0';
        for (i = 0; i < TAMANIO_DEL_REGISTRO_LCD; i++)
        {
                *(pLCDREG + i) = 0x00;
                *(LCD_Data+i) = 0x00;
        }
        actualizar_LCD();
}
void actualizar_LCD(void)
{
        ESCRITURA_DE_CADENA_HABILITADO = 0;
        for (char i = 0; i < TAMANIO_DEL_REGISTRO_LCD; i++)
        *(pLCDREG + i) = *(LCD_Data+i);
}
SIGNAL(SIG_LCD)
{
        unsigned char letra=0;
        unsigned char i=0;
        if (ESCRITURA_DE_CADENA_HABILITADO==1)
        {
                for(i=0;(i<6);i++)
                {

        if(!(memo_temp_texto[i+LCD_INT_contador]=='\0'))
                        {
                        letra                                   =
memo_temp_texto[i+LCD_INT_contador];
                        escribir_caracter_en_LCD(letra,i);
                        }
                        else
```

```
                    {
                    escribir_caracter_en_LCD(' ',i);
                    }
                    _delay_loop_2(20000);
                }
        if(LCD_INT_contador<NUMERO_MAXIMO_DE_CARAC
TERES)
        LCD_INT_contador++;
        else
        {
                LCD_INT_contador=0;
                ESCRITURA_DE_CADENA_HABILITADO = 0;
        }
}
        for (char i = 0; i < TAMANIO_DEL_REGISTRO_LCD; i++)
        *(pLCDREG + i) = *(LCD_Data+i);
}
```

4.1.2. JOYSTICK.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
   ---------------------------------------------------------
```

```
*/
#define MASCARA_PINB (1<<PINB4)
#define MASCARA_PINE ((1<<PINE3)|(1<<PINE2))
#define  IZQUIERDA  2
#define  DERECHA 3
#define  CENTRO 4
#define  NO_VALIDA 5
#define  posicion_C 2 //DERECHA
#define  posicion_D 3 //IZQUIERDA
#define  posicion_O 4 //CENTRO
#define VERDADERO  1
#define FALSO      0

volatile unsigned char SELECCION = 0;
volatile unsigned char SELECCION_VALIDA = 0;
int TD=30;
volatile int enter=0;
void inicializar_joystick(void);
void manejar_interrupcion(void);
void obtener_seleccion(void);
void inicializar_joystick(void)
{
        CLKPR = (1<<CLKPCE);
        CLKPR = (1<<CLKPS3);
        while(CLKPR & (1<<CLKPCE));
        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));
        DDRD = 0xFF;
        PORTD = 0x00;
}
void manejar_interrupcion(void)
{
        unsigned char joystick;
        unsigned char seleccion;
        joystick  = ((~PINB) & MASCARA_PINB);
         joystick |= ((~PINE) & MASCARA_PINE);
         if((joystick & (1<<posicion_C)))
                seleccion = DERECHA;
        else if((joystick & (1<<posicion_D)))
                seleccion = IZQUIERDA;
        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();
}
void obtener_seleccion(void)
{
        char temperatura_ASCII[]={'0','0','\0'};
        unsigned char seleccion;
        enter=0;
        cli();
        if(SELECCION_VALIDA)
        {
                seleccion = SELECCION;
                SELECCION_VALIDA = FALSO;
                }
                else seleccion = NO_VALIDA;
                if(seleccion != NO_VALIDA)
                {
                        switch(seleccion)
                        {
                                case CENTRO:
                                        enter=1;
                                        break;
                                case IZQUIERDA:
                                        TD++;
                                        enter=0;
                        itoa(TD,temperatura_ASCII,10);
                        escribir_caracter_en_LCD('T',0);
                        escribir_caracter_en_LCD('D',1);
```

```
        escribir_caracter_en_LCD(temperatura_ASCII[0],2);

        escribir_caracter_en_LCD(temperatura_ASCII[1],3);
                escribir_caracter_en_LCD('.',4);
                escribir_caracter_en_LCD('C',5);
                actualizar_LCD();
                break;
                        case DERECHA:
                                TD--;
                                enter=0;
                itoa(TD,temperatura_ASCII,10);
                escribir_caracter_en_LCD('T',0);
                escribir_caracter_en_LCD('D',1);

        escribir_caracter_en_LCD(temperatura_ASCII[0],2);

        escribir_caracter_en_LCD(temperatura_ASCII[1],3);
                escribir_caracter_en_LCD('.',4);
                escribir_caracter_en_LCD('C',5);
                actualizar_LCD();
                break;
                default:
        break;
        }
}
sei();
}
SIGNAL(SIG_PIN_CHANGE0)
{
```

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

4.1.3. ADC.H

```
void Inicializar_ADC(void);
int leer_ADC(void);
void Inicializar_ADC(void)
{
        CLKPR = (1<<CLKPCE);
        CLKPR = (1<<CLKPS1);
        ADMUX= (1<<REFS0)|(1<<MUX2);
        ADCSRA = (1<<ADEN)|(1<<ADPS2);
        leer_ADC();
}
int leer_ADC(void)
{
        char i;
        int ADC_temp;
        int ADCsuma = 0;
        sbi(PORTF, PF3);
        sbi(DDRF, DDF3);
        sbi(ADCSRA, ADEN);
        ADCSRA |= (1<<ADSC);
        while(!(ADCSRA & 0x10));
        for(i=0;i<8;i++)
        {
```

```
            ADCSRA |= (1<<ADSC);
            while(!(ADCSRA & 0x10));
            ADC_temp = ADCL;
            ADC_temp += (ADCH << 8);
            ADCsuma += ADC_temp;
        }
        ADCsuma = (ADCsuma >> 3);
        cbi(PORTF,PF3);
        cbi(DDRF,DDF3);
        cbi(ADCSRA, ADEN);
        return ADCsuma;
}
```

4.1.4. PID.H

```
void PID(void);
void Inicializar_PWM(void);
void leer_temperatura(void);
int valor;
float a=0.1243,b=0.0062,c=0.6215; // Constantes para parámetros de
controlador PID
float rT,eT,iT,dT,yT,uT,iT0=0,eT0=0; // Variables de controlador PID
float max=1023,min=0;   // Variables para anti-windup
extern int TD;
void PID(void)
{
        valor=leer_ADC()/3.1;   //Leer ADC
        leer_temperatura();
        yT=valor*10; // Amplificación de la señal de salida (TA)
```

```
        rT=TD*10; // Amplificación de la señal de entrada (TD)
        eT=rT-yT;  // Cálculo de la señal de error e(kT)
        iT=b*eT+iT0;  // Cálculo del término integrativo i(kT)
        dT=c*(eT-eT0);  // Cálculo del término derivativo d(kT)
        uT=iT+a*eT+dT;   // Cálculo de la señal de control u(kT)
        if (uT>max)    // Anti-windup
                uT=max;
        else if (uT<min)
                uT=min;
        OCR1B=uT;
        iT0=iT;
        eT0=eT;
        _delay_loop_2(25000);   // Periodo de muestreo T=0.1s
}
void leer_temperatura(void)
{
        char temperatura_ASCII[]={'0','0','\0'};
        itoa(valor,temperatura_ASCII,10);
        escribir_caracter_en_LCD('T',0);
        escribir_caracter_en_LCD('A',1);
        escribir_caracter_en_LCD(temperatura_ASCII[0],2);
        escribir_caracter_en_LCD(temperatura_ASCII[1],3);
        escribir_caracter_en_LCD('.',4);
        escribir_caracter_en_LCD('C',5);
        actualizar_LCD();
}
void Inicializar_PWM(void)
{
DDRB=0b01000000; // Pin 7 del Puerto B como salida
TCCR1A=0b00100001;
```

```
TCCR1B=0b00000010;
}
```

4.1.5. CONTROLADOR_TEMPERATURA.C

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <compat/deprecated.h>
#include <stdlib.h>
#define F_CPU 1000000l
#include <util/delay.h>
#include "LCD.h"
#include "joystick.h"
#include "ADC.h"
#include "PID.h"
extern volatile int enter;
int main(void)
{
        char temperatura_ASCII[]={'0','0','\0'};
        inicializar_LCD();
        sei();
        inicializar_joystick();
        itoa(TD,temperatura_ASCII,10);
        escribir_caracter_en_LCD('T',0);
        escribir_caracter_en_LCD('D',1);
        escribir_caracter_en_LCD(temperatura_ASCII[0],2);
        escribir_caracter_en_LCD(temperatura_ASCII[1],3);
        escribir_caracter_en_LCD('.',4);
        escribir_caracter_en_LCD('C',5);
        actualizar_LCD();
```

```
        while(!enter)
            SMCR = 1;
            EIMSK = ((0<<PCIE1)|(0<<PCIE0));
            Inicializar_ADC();
            Inicializar_PWM();
        while(1)
        PID();
        return 1;
}
```

4.2. SIMULACIONES

En este subcapítulo se presentarán las respectivas simulaciones en Proteus. En la gráfica posterior se muestra la temperatura deseada inicial programada de 30°Ccomo se indicó anteriormente.

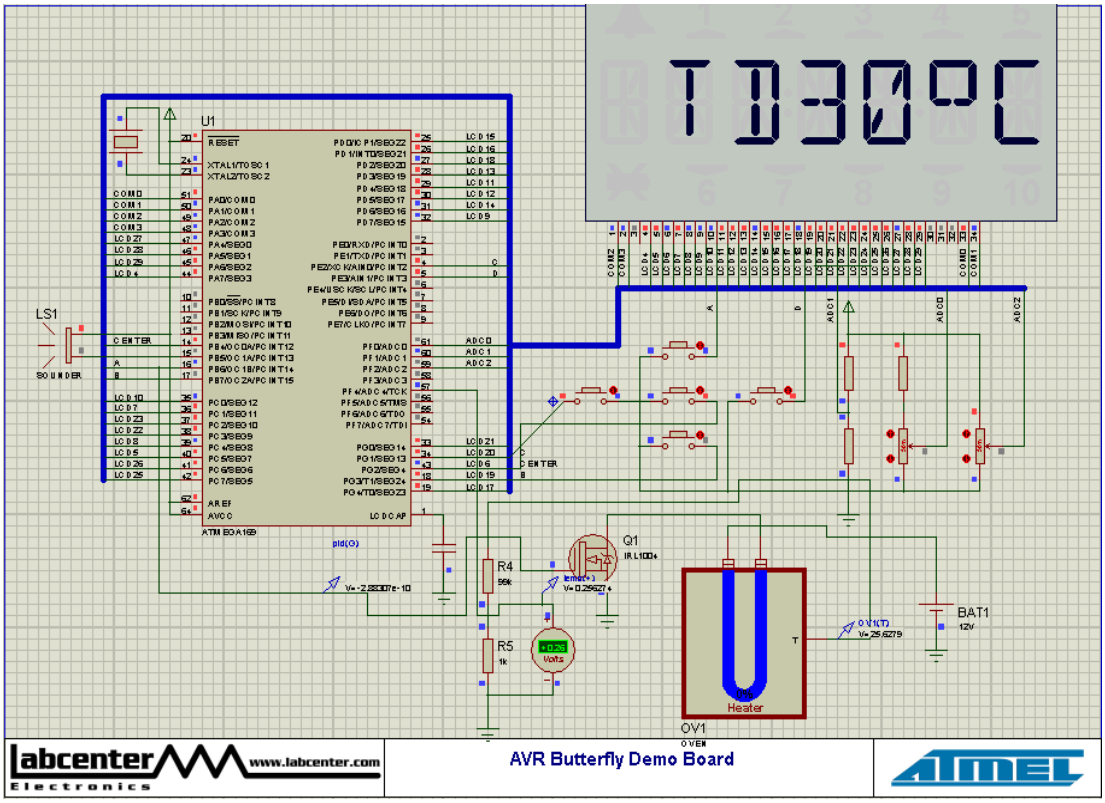


Fig. 4.1: Temperatura deseada inicial programada a 30ºC

En las siguientes gráficas se mostrarán las temperaturas deseadas de $40°C, 60°C, 80°C$ con sus respectivas respuestas de temperatura del sistema donde se puede observar el tiempo de pico, el valor de temperatura pico, el tiempo de estabilización y el error de estado estable aproximadamente de $\pm 1°C$.

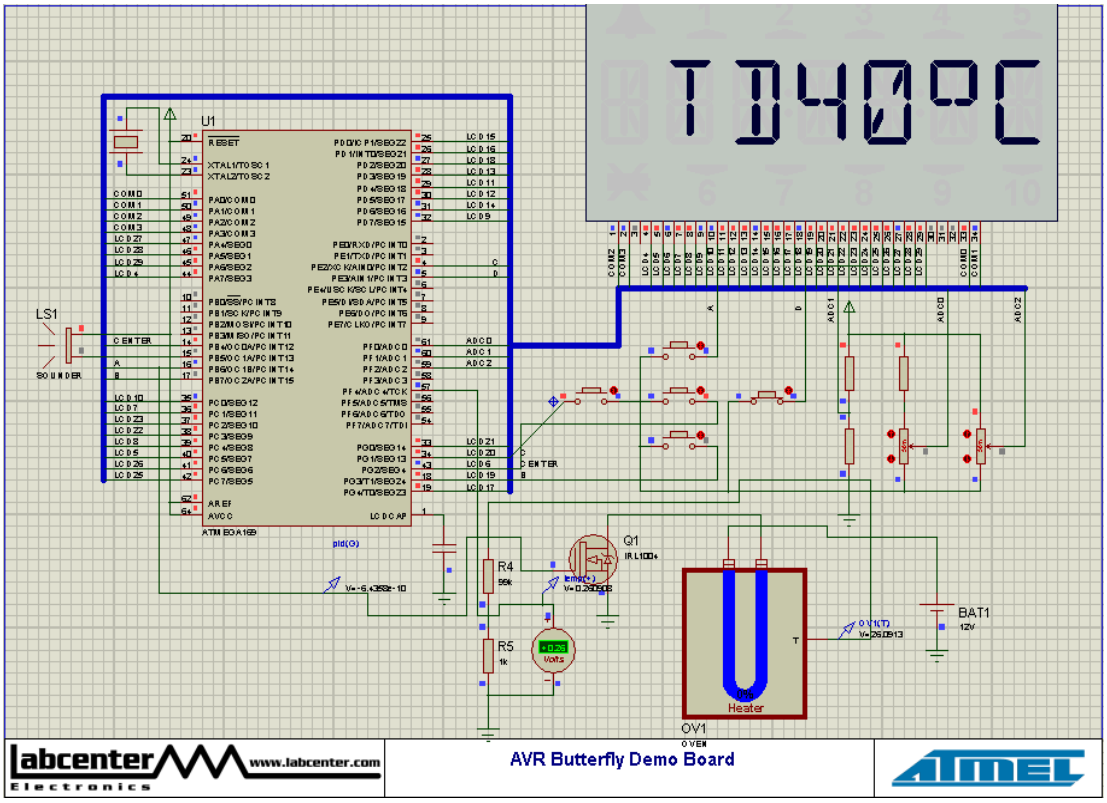
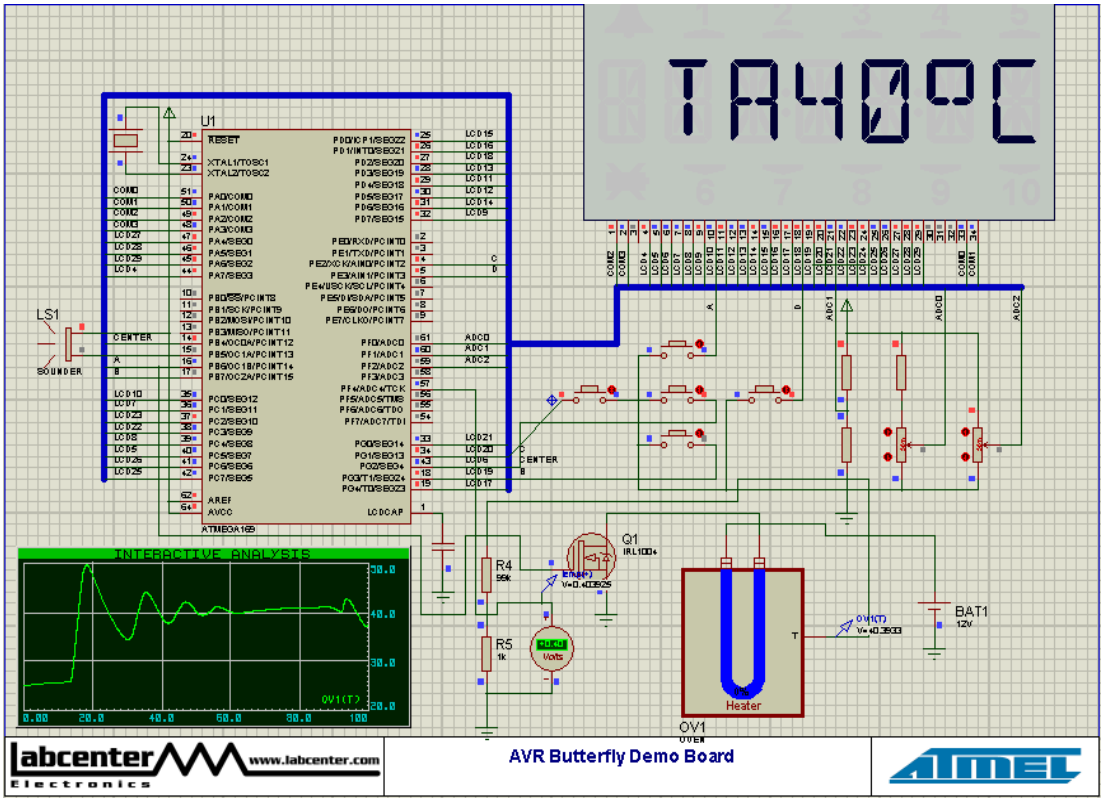


Fig. 4.2: Seteo de la temperatura deseada a 40ºC



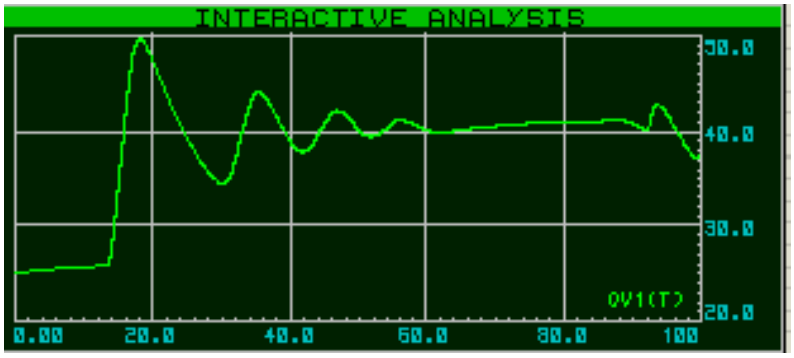


Fig. 4.3: Respuesta de temperatura del sistema a 40ºC

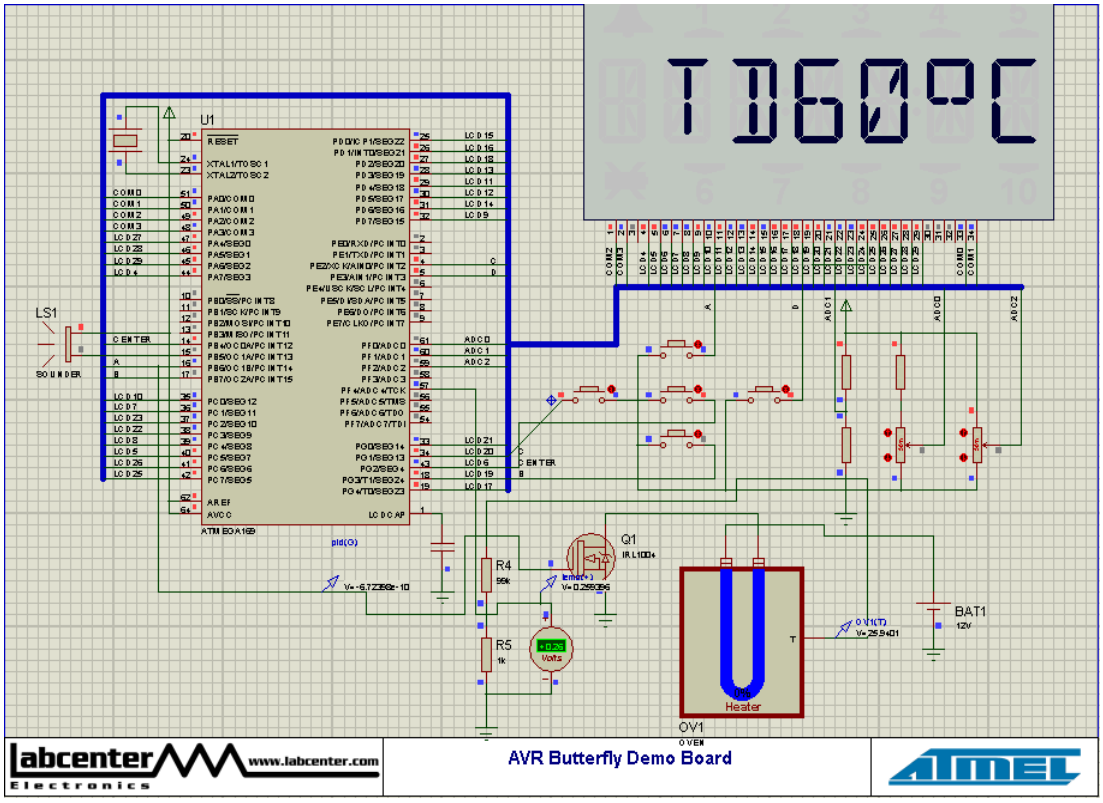
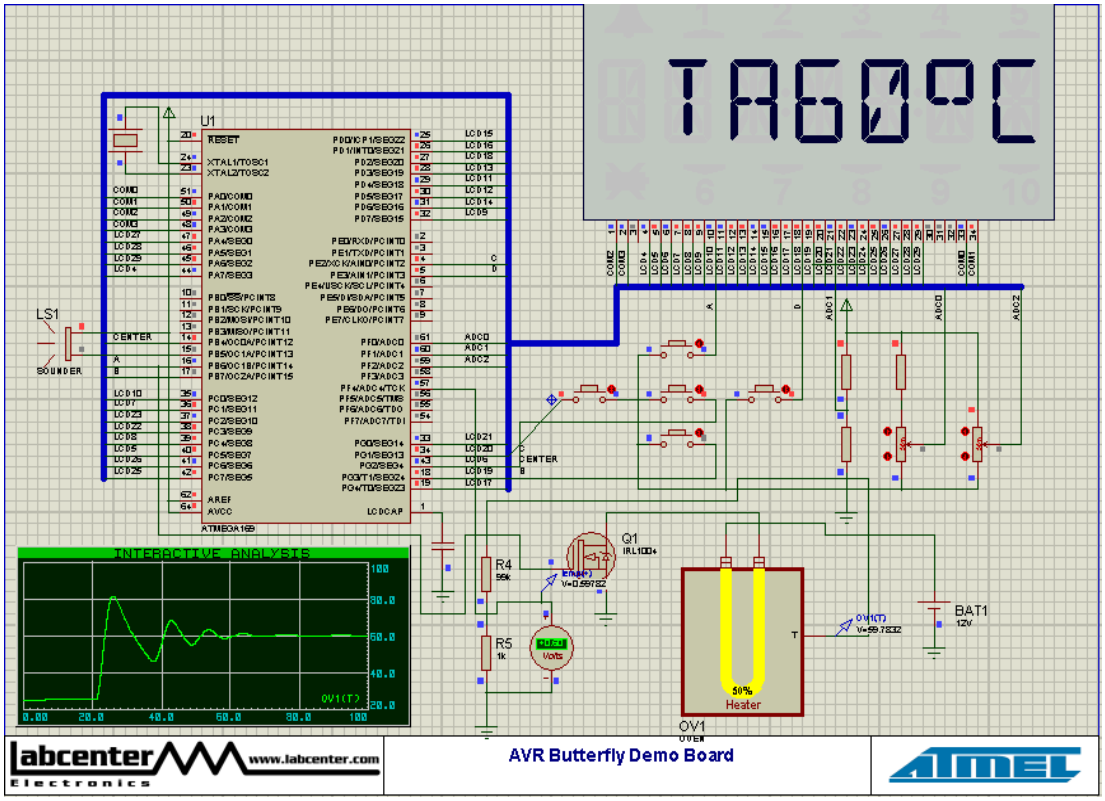


Fig. 4.4: Seteo de la temperatura deseada a 60ºC



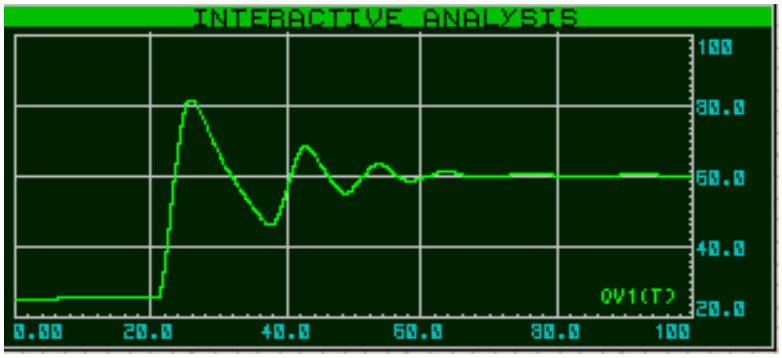


Fig. 4.5: Respuesta de temperatura del sistema a 60ºC

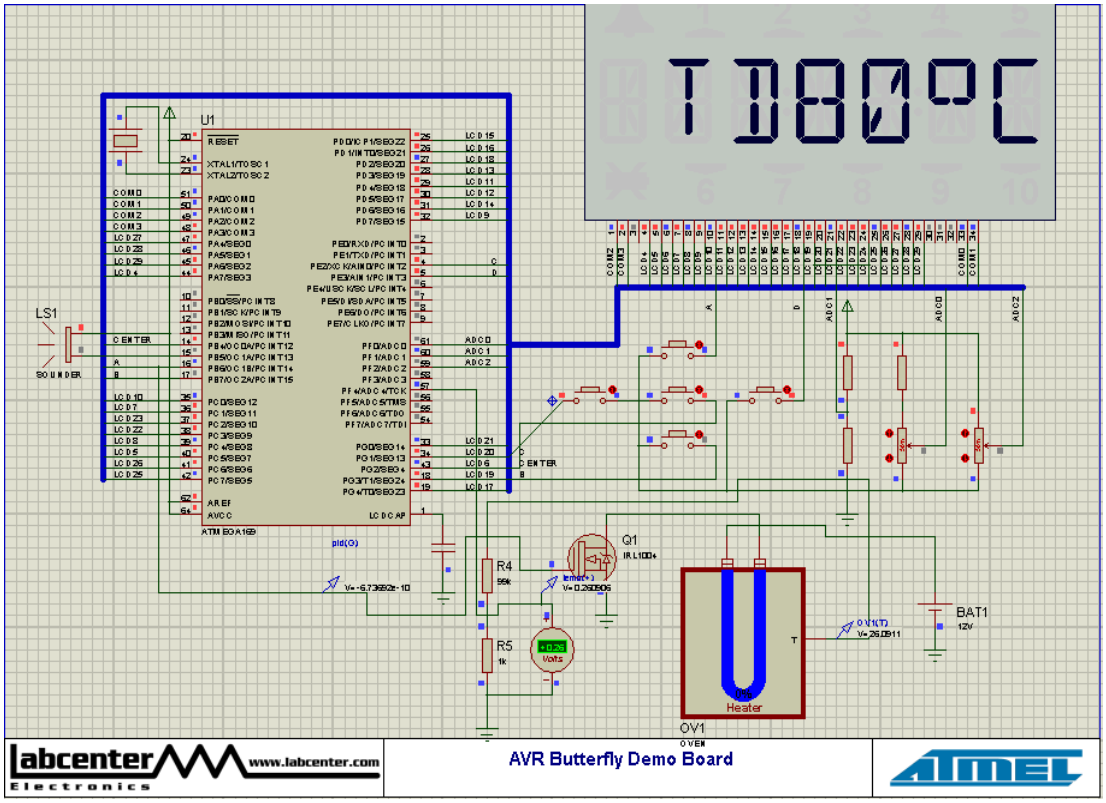
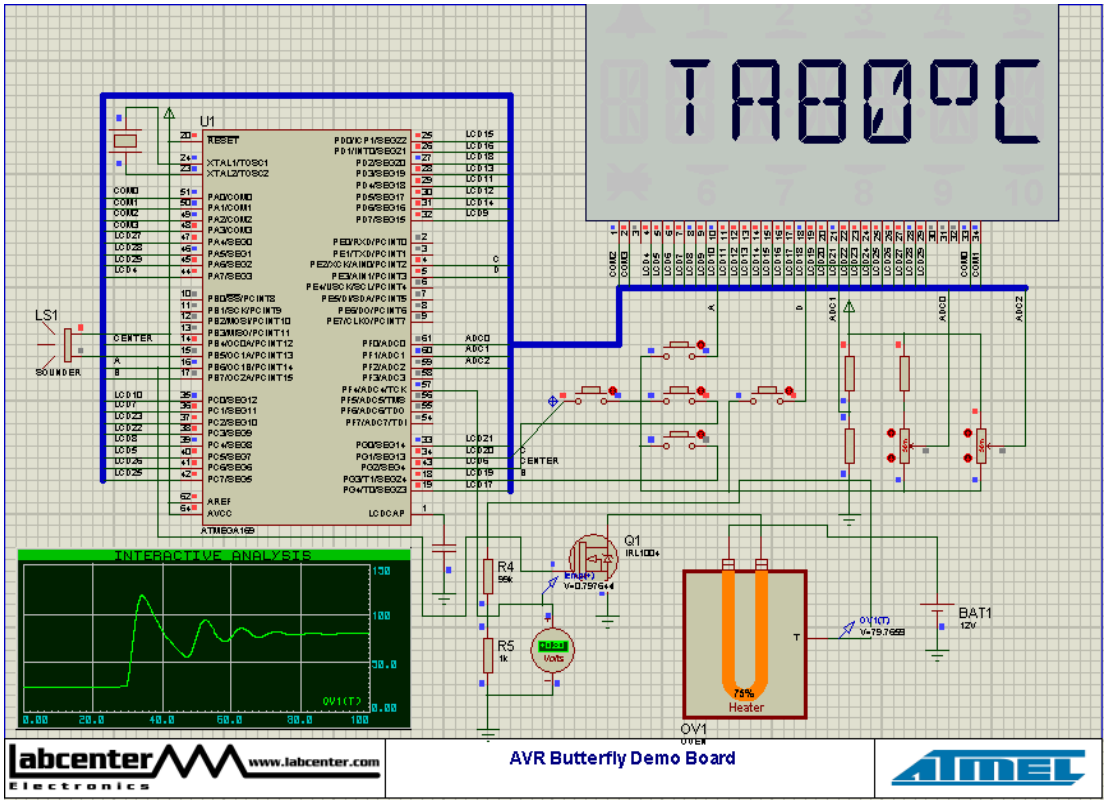


Fig. 4.6: Seteo de la temperatura deseada a 80ºC



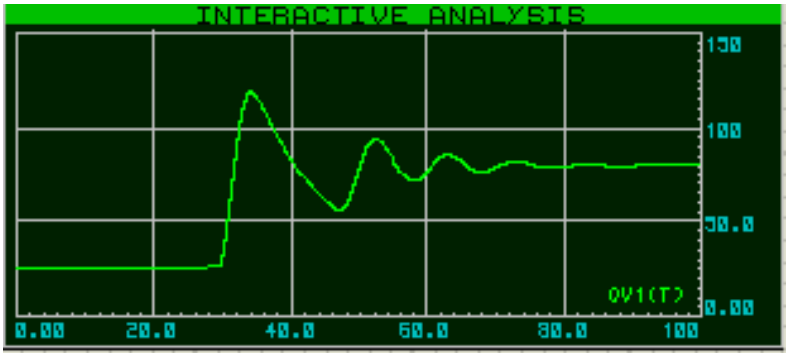


Fig. 4.7: Respuesta de temperatura del sistema a 80ºC

CONCLUSIONES

1. Se logró diseñar e implementar otra configuración de controlador PID de temperatura basado en el uso de la tarjeta AVR Butterfly cumpliendo, de esta manera, uno de los objetivos del presente proyecto.
2. Con el uso del kit AVR Butterfly conseguimos optimizar en gran proporción las dimensiones a la que un controlador de temperatura analógico puede ser implementado.
3. El proyecto ofrece un manejo sencillo de interacción usuario-planta, ya que el kit dispone de la mayoría de componentes que un controlador de temperatura requiere.
4. A temperaturas cercanas a la temperatura ambiente, el sistema tiende a estabilizarse en un tiempo menor que cuando queremos controlar temperaturas por encima de los 40ºC. Esto se debe a que el sensor se encuentra midiendo la temperatura ambiente y al controlador le toma poco en estabilizarse.
5. Luego de alcanzar la estabilidad, la planta se ve perturbada a cambios externos que producen un estado de oscilación aleatorio alrededor del punto de operación, de aproximadamente 1ºC.
6. El tiempo de respuesta del sistema, además del controlador y de la propia planta, también depende del voltaje de alimentación de la lámpara, ya que éste determinará la máxima corriente que pueda circular por ella.

7. El tiempo de estabilidad del sistema es directamente proporcional a la temperatura deseada o Set Point, es decir, el sistema se tomará más tiempo en estabilizarse cuando queremos controlar temperaturas altas, y viceversa
8. Debido a que trabajamos con una lámpara, el sistema no puede controlar cambios de temperatura de mayor a menor, para el enfriamiento hay que esperar a que ocurra la transferencia de calor con el medio ambiente.
9. Existen diversas formas de perturbar el sistema en estado estacionario. Una de ellas es simplemente acercar o alejar el sensor de la planta, así cambiaremos la temperatura actual del sistema y el controlador corregiría estos cambios.

RECOMENDACIONES

1. Se recomienda leer meticulosamente el datasheet del Kit AVR Butterfly, la configuración de los componentes de hardware y de sus pines ya que cualquier conexión errónea puede generar daños irreversibles en los componentes.
2. El transistor de potencia utilizado en la etapa de fuerza maneja corrientes del orden de los amperios por lo que se recomienda hacer uso de un disipador de calor para evitar cualquier tipo de quemaduras.
3. Para un correcto funcionamiento del sistema se recomienda utilizar una fuente de voltaje para la planta que demande corrientes por encima de los 5A y otra alimentación separada de la planta para la tarjeta que, en nuestro caso son las 4 pilas recargables AA de 1.2V cada una.
4. Antes de comenzar a utilizar el sensor de temperatura, asegurarse de su funcionamiento calibrándolo con un termómetro.
5. Si conservamos el sensor de temperatura a una distancia determinada de la bombilla (3cm aproximadamente), el sistema funcionará de manera correcta. Caso contrario, si el sensor no se encuentra fijo, el sistema presentaría perturbaciones.

BIBLIOGRAFÍA

(1) ATMEL
AVR Butterfly Evaluation Kit – User Guide
http://www.atmel.com/dyn/resources/prod_documents/doc4271.pdf
Abril 24, 2011

(2) Escuela Superior Politécnica del Ejército
Características del Kit AVR Butterfly en español
http://www.espe.edu.ec/repositorio/T-ESPE-014271.pdf
Abril 24,2011

(3) Ilber Adonayt Ruge Ruge (Universidad de Cundinamarca)
Método básico para implementar un controlador digital PID
http://www.edutecne.utn.edu.ar/microcontrol_congr/industria/MTODOB~1.PDF
Abril 26,2011

(3) Pololu Robotics & Electronics
Pololu USB AVR Programmer User's Guide
http://www.pololu.com/docs/pdf/0J36/pololu_usb_avr_programmer.pdf
Abril 30,2011

(4) Alldatasheet
Características del Mosfet k1257
http://www.alldatasheet.com/view.jsp?Searchword=K1257
Mayo 01, 2011

(5) National Semiconductor
Características del sensor de temperatura LM35
http://www.national.com/ds/LM/LM35.pdf
Mayo 02,2011