UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

SISTEMA DE CONTROL PID PARA UNA TINA DE BAÑO DE COBRE POR ELECTRÓLISIS

INFORME DE SUFICIENCIA

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRONICO

PRESENTADO POR:

JORGE VIVANCO ALVAREZ

PROMOCIÓN
1982 – I

LIMA – PERÚ
2006

SISTEMA DE CONTROL PID PARA UNA TINA DE BAÑO DE COBRE POR ELECTRÓLISIS

Dedico este trabajo a:

Mis padres, inspiración plena de lucha y sacrificio,

Mis Hermanos, por el apoyo incondicional en mi carrera,

Y mis sobrinos esperanza de superación.

SUMARIO

El presente trabajo describe la tecnología del Control PID aplicado al baño de cobre por electrólisis , aplicando el control a la corriente que circula entre los electrodos de un proceso electrolítico, considerando la regulación de temperatura de la celda electrolítica, siendo estas variables (corriente y temperatura) las que tienen que controlar para obtener un acabado brillante del baño efectuado.

Se presenta una visión general de los procesos electroquímicos para conseguir el baño de cobre y el motivo de la elección del método a mostrar en este informe, considerando sus ventajas sobre los otros métodos

También se hace un análisis mediante diagramas de bloques del sistema , la demostración y simulación del proceso aplicando el programa MATLAB. La explicación del proceso de control y la programación de microcontroladores para el control de temperatura y de corriente.

INDICE

PROLOGO 1

CAPITULO I 2
FORMULACION DEL PROBLEMA

1.1 Breve referencia a los procesos de baño de cobre 2

1.2 Fundamento teórico 4

CAPITULO II 7
MODELO MATEMATICO DEL PROCESO

2.1 Descripción del sistema 7

2.2 Estructura modelada de temperatura 13

CAPITULO III 17
LINEALIZACION Y DISCRETIZACION

3.1 Linealización de la dinámica de la tina 17

3.1.1 Controlabilidad y observabilidad 18

3.1.2 Ecuación de estado discreta 18

3.2 Simulación de la respuesta del modelo discreto 18

CAPITULO IV 21
CONTROL PID EN TIEMPO DISCRETO

4.1 Introducción al control PID en tiempo discreto 21

4.2 Diseño del sistema de control PID del baño de cobre 23

4.3 Simulación del sistema de control PID 24

CAPITULO V 61
PROPUESTA DE IMPLEMENTACION

5.1 El hardware del sistema de control 61

5.2 El software del sistema de control 63

5.3 costo aproximado 63

CONCLUSIONES Y RECOMENDACIONES 64

ANEXOS

A- Descripción del PIC 16F877

B- Datasheet LM335 National Semiconductor

C- Datasheet PIC 18F452 Microchip

D- AN937 Microchip

BIBLIOGRAFIA.

PROLOGO

Este informe sintetiza el diseño de un sistema de control PID para el proceso de baño de cobre por electrólisis, usando para la simulación el MATLAB. Se usa un microcontrolador programado para el control PID y para generar la señal PWM usando un circuito analógico asociado para regular el voltaje del proceso de electrólisis en forma proporcional al procesamiento PID del decremento o incremento de la corriente.

Para detectar la variación de corriente usamos como sensor una resistencia en serie con el circuito del proceso, cuyo voltaje variable será restado por un voltaje setpoint promedio y tratada esta diferencia por el conversor análogo digital del microcontrolador.

Cuando es complicado obtener un modelo matemático como el caso de este proceso electroquímico, se recomienda usar un modelo matemático de primer orden.

También sería factible, si deseamos controlar el espesor de la capa de cobre, usar las ecuaciones de la Ley de Faraday.

CAPITULO I

FORMULACION DEL PROBLEMA

1.1 Breve referencia a los procesos de baño de cobre-

1.1.1 Reseña histórica

El término electrólisis procede etimológicamente de ¨electro¨ (electricidad) y ¨lisis (desatar o romper). Es decir hace referencia al hecho de romper enlaces químicos por acción de la electricidad . Para ello se necesita hacer circular una corriente eléctrica por un electrólito .

Faraday, dió el nombre de electrólitos (de ¨litos¨ = piedra) a las piedras (sales o bases) que fundidas o disueltas conducían la corriente eléctrica. Al recipiente donde las colocaban lo denominaron cuba electrolítica . Para hacer llegar la energía eléctrica desde un generador de corriente continua (pila) utilizaron unos conductores que llamaron ¨electrodos¨ . Para asignarle el nombre a los electrodos tomaron dos nombres etimológicamente muy acertados . Los eligieron entre otros términos porque eran nombres que sonaban bien . Así eligieron ¨ánodo¨ y ¨cátodo¨.

1.1.2 Cobreado electroquímico

Operación que consiste en depositar una capa de cobre sobre una superficie. Según el proceso de cobreado se emplea la terminología siguiente:

Cobreado electrolítico: deposición electrolítica puede ser:
Monovalente: Cobreado cianurado, aminado.
Bivalente: cobreado a partir de soluciones de sulfato de cobre, pirofosfato de cobre.

Cobreado químico : deposición de cobre por desplazamiento obtenido sobre determinados metales por inmersión en una solución apropiada.

Cobreado por metalización a pistola recubrimiento por proyección de cobre fundido a pistola

Chapado de cobre: recubrimiento por aplicación sobre el metal base de una hoja de cobre o aleación de cobre, obteniéndose la adherencia por laminado del conjunto después de tratamientos especiales preparatorio de las superficies.

COBRIZADO-

Frecuentemente , el cobre forma la primera capa en un sistema de capas de recubrimiento, puesto que es fácil depositar en metales y plásticos , ya que presenta una elevada conductividad ; además, la capa de cobre es muy resistente,
económica de aplicar y forma una buena base adhesiva para otros metales. El cobrizado puede aplicarse a partir de baños alcalinos cianurados y baños ácidos con ácido sulfúrico.

Los dos procesos de cobrizado empleados con más frecuencia son el método de ácido Sulfúrico (sulfatos) y el de cianuro (baño alcalino). El cobrizado ácido con sulfatos, generalmente requiere un control más estricto del baño a fin de mantener los parámetros en el rango óptimo, sin embargo se evita el uso de cianuro. El baño ácido también puede utilizarse como primer revestimiento metalizado en plásticos por su gran ductibilidad.

En un baño ácido el sulfato de cobre ($CuSO_4$) representa la fuente de iones de cobre que se deposita en la superficie a recubrir. Para este proceso se recomienda sulfato de cobre químicamente puro. El baño de cobre típico contiene sulfato de cobre (250 g/l), ácido sulfúrico (100 g/l), iones de cloruro (<1 g/l) y aditivos de brillo (6 g/l). El ácido sulfúrico sirve para aumentar la conductividad de la solución y para disolver el ánodo de cobre, este ánodo conduce la corriente eléctrica y proporciona los iones de cobre para formar sulfato de cobre. El proceso de cobre ácido se realiza a una temperatura entre 20° y 30°C. Los electrólitos cúpricos de ácido sulfúrico contienen generalmente altas concentraciones de químicos orgánicos auxiliares, pues requieren un mayor control de los parámetros de operación del baño a fin de obtener ciertas características como dureza, nivelación y brillo.

Los baños alcalinos de cobre cianurado operan a una temperatura elevada , de 40 a

60°C, y contienen el cobre aglutinado en forma de complejos cianurados. Este tipo de baños generalmente contienen cianuro de cobre (60 g/l) , cianuro libre (aproximadamente 20 g/l), hidróxido de sodio (20 g/l) y aditivos de brillo (10 g/l).
Por el peligro que representa a la salud humana y al ambiente , al usar baños cianurados, deben respetarse normas especiales referentes a la salud ocupacional y seguridad en el trabajo, y la protección al ambiente, tanto durante el cobrizado como en el manejo y el tratamiento de los residuos y las aguas residuales.

1.2-Fundamento teórico-

La electrodeposición consiste en el depósito de una lámina fina de metal de unos 0.02 mm de espesor sobre una superficie conductora de la electricidad, lo que se puede conseguir mediante un proceso de electrólisis .

La electrodeposición de cobre es uno de los ejemplos más sencillos. El metal que se quiere recubrir se coloca como cátodo, y como ánodo se pone un electrodo de cobre. La disolución en que se sumergen ambos electrodos contiene una sal de Cu^{2+} ,como puede ser el $CuSO_4$.

En el ánodo se produce la semirreacción de oxidación:

$$Cu\ (s) \longrightarrow Cu^{2+}\ (ac) + 2e^-$$

En el cátodo se produce la semirreacción de reducción:

$$Cu^{2+}\ (ac) + 2e^- \longrightarrow Cu\ (s)$$

A medida que se deposita cobre sobre el cátodo, van pasando iones de Cu^{2+} a la disolución, desde el ánodo, manteniendo constante la concentración de Cu^{2+} . Por lo tanto, en este proceso no existe una reacción neta, sino que hay simplemente una transferencia de cobre desde el ánodo hasta el cátodo.

La cantidad de sustancias liberadas en los electrodos durante la electrólisis está en relación con la carga total que ha fluído en el circuito eléctrico. Si se observan las semirreacciones de los electrodos, la relación es estequiométrica, como se puede ver en los siguientes ejemplos.

En el proceso catódico, con un mol de iones Cu^{2+} reaccionan 2 moles de electrones, siendo necesarios 2 Faradays (1 faraday, $9.65*10^4$ coulombios, equivale a la carga de 1 mol de electrones) para precipitar 1 mol de cobre metal:

$$1\ mol\ Cu \longrightarrow 1\ mol\ Cu^{2+} \longrightarrow 2\ moles\ e^- \longrightarrow 2*(9.65*10^4)C$$

Para cada metal, depende de su semirreacción de reducción, como por ejemplo:

$$Ag^{+}(ac) + e^{-} \longrightarrow Ag(s)$$

Cuando un mol de electrones reacciona con un mol de iones plata, pasa un Faraday a través del circuito, y un mol de metal plata es depositado en un electrodo:

En el caso del oro:

$$Au^{3+}(ac) + 3\,e^{-} \longrightarrow Au(s)$$

Con un mol de iones Au^{3+} reaccionan 3 moles de electrones, pasando 3 Faradays a través del circuito para provocar la deposición de 1 mol de oro metal:

$$1\ mol\ Au \longrightarrow 1\ mol\ Au^{3+} \longrightarrow 3\ moles\ e^{-} \longrightarrow 3 \times (9.65*10^{4})C$$

Por tanto si se conoce la cantidad de sustancia precipitada, se puede saber la cantidad de carga eléctrica que provocó su deposición, según la relación:

gramos metal——> moles metal ——> moles e^{-}———Faradays ——>coulombs

El flujo de calor en la celda no es proporcional a la corriente que fluye en la Electrólisis por lo que requiere una relación distinta a la de transferencia de calor.

Para el flujo de corriente se puede usar la ley de Faraday

$Q = \int I\,dt$; I= Corriente uniforme (Amperios) durante dt

$Q = I(t_2 - t_1) = nF$

n = número de equivalentes oxidados o reducidos

F = Faradios

1F = 96,490 Coulombs = 1 mol de electrones ($6.023 * 10^{-23}$ electrones)

$n = m / m_{eq}$

m = peso del material oxidado o reducido

m_{eq} = peso equivalente expresado en gramos del material.

Ejemplo 1- Si se supone eficiencia de 100%, ¿Cuántos coulombs se necesitan para recubrir una pieza metálica con 50gr de Cu a partir de una solución acuosa $CuSO_4$?

$n = (m / m_{eq}) = (50gr / 31.77gr) = 1.576$

$Q = nF = 1.576 * 96490$ <Coulombs = 152,000 Coulombs.

El peso equivalente es sólo la mitad del peso atómico del cobre, debido a que cada ión de cobre requiere dos electrones para reducirse a átomo de cobre.

Ejemplo 2- ¿ Cuántos amperios se requieren durante un período de 1000 segundos para recubrir una pieza con 5gr de Ag a partir de una solución de Ag^{+} ?.

$I = [nF / (t_2 - t_1)] = [(5gr / 107.88gr)*96490 \text{ Coulombs} / 1000 \text{ seg}] = 4.45 \text{ Amp.}$

Ejemplo 3- ¿Cuánto tiempo se requiere para recubrir una pieza metálica con 10gr de oro a partir de una solución de ión áurico (Au^{3+}) con una corriente de 0.5 Amperios?

$(t_2 - t_1) = (nF / I) = [(10gr / 65.7gr)*96490 / (0.5 \text{ coulombs} / \text{seg})] = 29{,}400 \text{ seg.}$

Aquí, debido a que la reacción en el electrodo es $Au^{3+} + 3e^{-} \rightarrow Au$, el peso equivalente del oro es una tercera parte de su peso atómico.

CAPITULO II

MODELO MATEMATICO DEL PROCESO

2.1 Descripción del sistema

El sistema consta de dos procesos que son controlados por un µcontrolador 18F452, los cuales son el control de temperatura y el control de corriente del baño de cobre por electrólisis.

Estamos utilizando el µcontrolador 18F452 porque la firma Microchip nos está además proporcionando el programa del control PID para este µcontrolador, y el programa de cálculo matemático. Se usan dos puertos de entrada de señal análoga, una para la muestra de voltaje proporcional a la variación de temperatura y el otro para la muestra de la señal de corriente de la electrólisis. Los modelos matemáticos que usamos son, uno para el control de temperatura que es usado en transferencia de calor para un horno [7], y el otro para el control de corriente que es usado en procesos de transporte de materia como sucede en muchos procesos químicos que es un modelo para un sistema simple de primer orden con retardo

2.2 Estructura modelada de temperatura

2.2.1 Principios

La fase a considerarse para el modelamiento del proceso contempla el nivel de temperatura en la tina para obtener un baño de cobre óptimo.

En la figura 2.1 se muestra el esquema representativo de la tina de medición de temperatura

Se trata de diseñar un sistema de lógica programable , capaz de controlar la temperatura en las celdas entre los márgenes comprendidos desde 20°C a 30°C con las siguientes especificaciones:

- Temperatura de consigna seleccionable por programa.
- Detección de temperatura real a base de sensor.

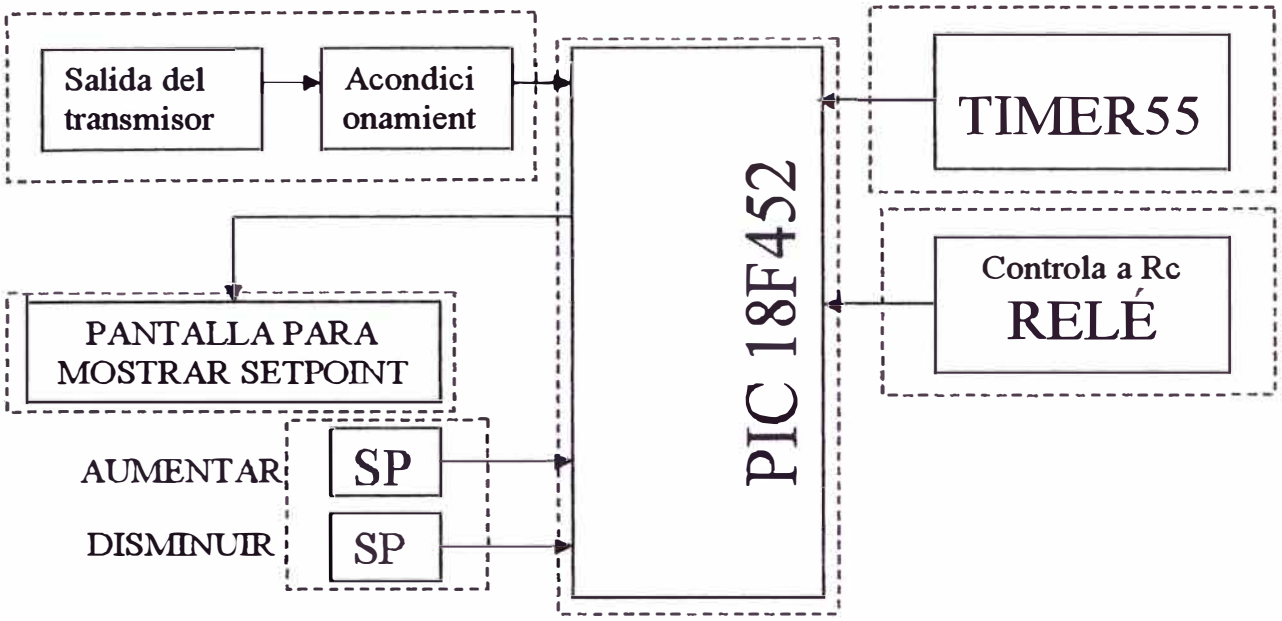


Figura 2.1 Diagrama de bloques del sistema de control

2.2.2 Funcionamiento.

Al establecer la temperatura de consigna, y si ésta es superior a la real de la tina (medida por el sensor y amplificada por la etapa DC), el relé de estado sólido se cierra permitiendo la aplicación de tensión a la resistencia calefactora Rc, lo que produce el calentamiento de la tina y la consiguiente elevación de su temperatura . Esta temperatura, convertida en tensión continua, es comparada por el µC con la temperatura de referencia (consigna), hasta que ambas coinciden. En este momento, el µC realiza la apertura del relé a través de la línea T0, de esta forma, se deja de aplicar la tensión de la fuente a la resistencia calefactora, permitiendo el enfriamiento de la tina y comenzando de nuevo la secuencia.

En definitiva, sí :

- Temperatura de consigna > Temperatura real → Rc activada (ON)
- Temperatura de consigna < Temperatura real → Rc desconectada (OFF).

2.2.3 Elección del sensor de temperatura en la tina.

Se ha elegido el sensor LM335 que es un sensor de temperatura de estado sólido que posee una sensibilidad de 10mV/°K, se usa en el caso de aplicaciones en las que se necesita medir temperaturas comprendidas entre –10 y 100°C, su diseño es similar al zener de dos terminales. Ver figura 2.2.

En la hoja de datos correspondiente al sensor LM335, se observa que la sensibilidad del dispositivo es de 10 mV/ °K, sin embargo en caso de esta aplicación lo que se desea es

medir grados Celsius. La relación que existe entre grados Kelvin y grados Celsius es la siguiente:

Una elevación de 1 grado en la escala Kelvin es igual a una elevación de 1 grado en la escala Celsius y el punto de congelación del agua es de 0°c y equivale a 273°K.

- Rango de temperatura de – 10 a 100°C

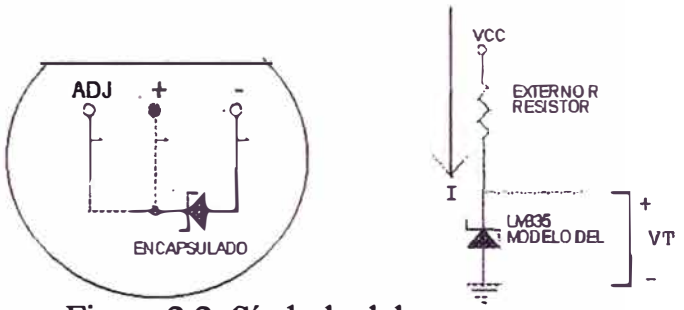


Figura 2.2 Símbolo del sensor

2.2.4 Diseño del Circuito de Acondicionamiento de la Señal (CAS)

Condiciones:

- El margen del ADC está entre 0V y 5 V .
- La salida del CAS debe ser lineal es decir cuando la salida del CAS es 0°C la entrada en el ADC es 0 V, cuando la salida del CAS mide 10°C , la entrada del ADC es 1V y así sucesivamente hasta llegar a los 50°C, en cuyo caso la salida del CAS o entrada del ADC es de 5V.

Ecuación que representa el sensor

En la figura 2.3 se muestra una gráfica de voltaje de salida del sensor LM335 en función de la temperatura.

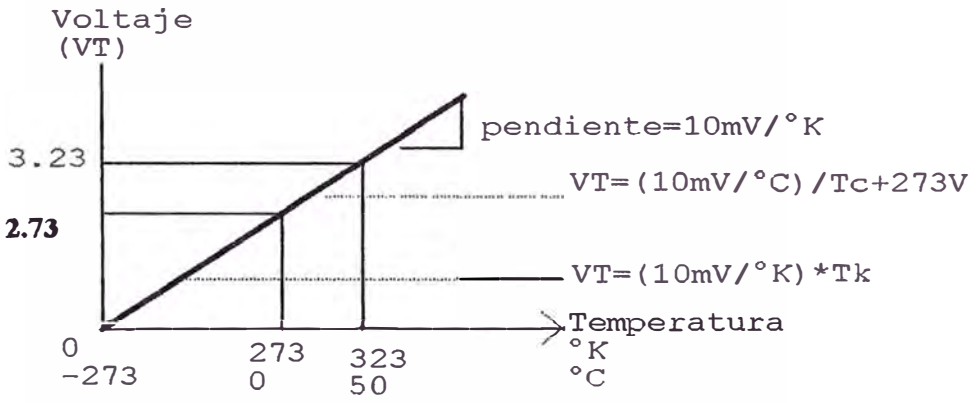


Figura. 2.3 Gráfica del voltaje en función de las características de temperatura

La pendiente de la línea equivale a la sensibilidad del dispositivo: 10 mV/°K por lo tanto, el voltaje de salida expresado en °K es el siguiente:

$$V_T = (10mV/°k) * (T_{ent°K})$$

En la cual T es la temperatura en °k a 273 °k (°0 C), el voltaje de salida del sensor es :

$$V_T = (10mV/°k) * 273°k = 2.73V$$

Como se puede observar en la figura 2.3

Ahora es posible expresar el voltaje de salida del sensor en grados Celsius como en la siguiente ecuación .

$$V_T = (10\,mV/°c) * (T_{ENT°c}) + 2.73$$

Donde T, es la entrada expresada en grados Celsius . En el caso de esta aplicación a 0°C V_T= 2.73V, a 50°C V_T=3.23 V. Este es el margen de entrada correspondiente al circuito de acondicionamiento de la señal(CAS)

El margen de salida del CAS viene a ser el margen de entrada al ADC, el cual está comprendido entre 0V y 5 V.

En la figura 2.4 siguiente se muestra un diagrama de bloques de este sistema de adquisición de datos para medición de temperatura.

Figura 2.4 Bloques que describen el comportamiento de la adquisición de datos

En base a información de que se dispone sobre el sensor y el convertidor ADC, se grafican las características de entrada / salida del CAS en la figura 2.6, se muestra esta gráfica. Los valores de salida del CAS se grafican en el eje Y, hay que tener presente que estos valores corresponden al margen de voltaje del convertidor de 0 a 5V los valores de entrada del CAS se grafican en el eje X, estos valores son los del margen del sensor – 2.73 a 3.23 V en el caso de esta aplicación.

La pendiente de la recta es :

$$m = \frac{(5-0)}{(3.23-2.73)} = 10 = \frac{\Delta V_o}{\Delta V_T} \qquad (2.1)$$

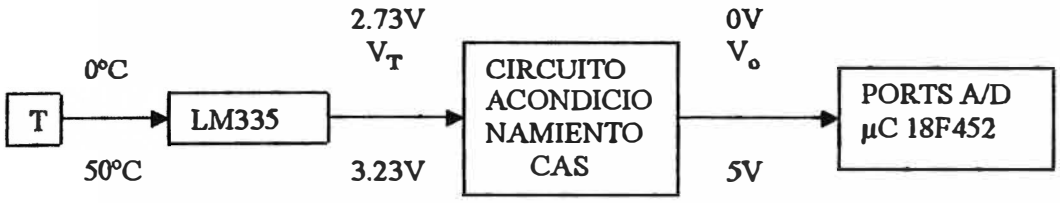


Figura 2.5 Diagrama de bloques del sistema para medida de temperatura

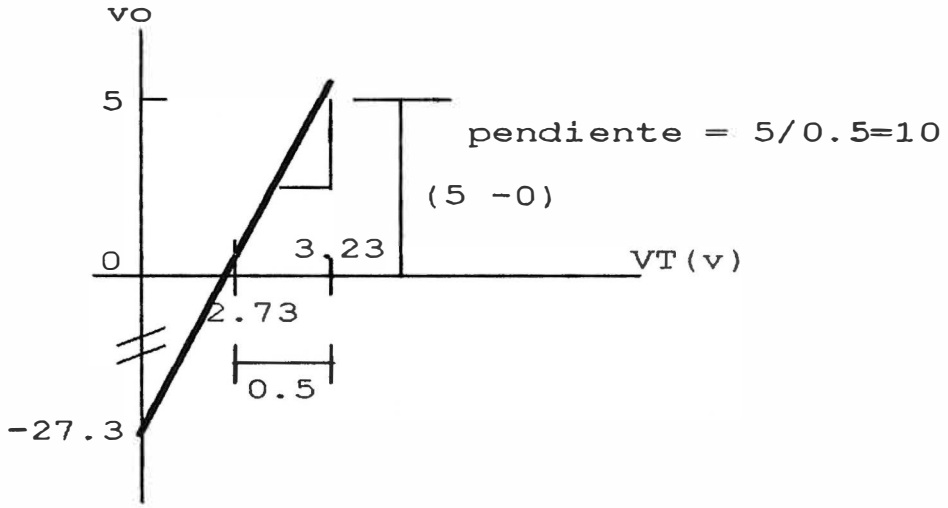


Figura 2.6 Las características de entrada – salida para el CAS

El valor de 10, es la ganancia por la que hay que multiplicar V_T, la desviación de cd se encuentra con solo elegir un punto de la línea y sustituirlo en la ecuación de una línea recta .

$$Y = mx + b \qquad (2.2)$$

Después de elegir un par de coordenadas (2.73,0) se obtiene :

$$0 = (10)(2.73) + b$$

Resolviendo la ecuación anterior para b se obtiene :

$$b = -27.3\ V$$

Por lo tanto, la ecuación de voltaje de salida del CAS es :

$$Vo = (10)*V_T - 27.3V \qquad (2.3)$$

Si bien la desviación de cd es de – 27.3 V, el voltaje de salida , Vo, no toma este valor debido que el margen de V_T esta comprendido entre 2.73 y 3.23 V, este margen de V_T limita el valor de Vo de 0V a 5 V.

c) Circuito acondicionador.

d) Una vez obtenido la ecuación del CAS expresado en la forma y= mx + b, se desea diseñar un circuito en que la ganancia de 10 y la desviación de –27.3 V se defina de manera independiente. La solución no está en un sumador no inversor puesto que en este, la ganancia y la desviación no se pueden definir de manera independiente. Lo que se necesita es un amplificador operacional como el que se muestra en al figura 2.7.

Un amplificador inversor de ganancia - 1 , seguido por un sumador inversor La ecuación de voltaje de salida del sumador es :

$$Vo = \frac{-R_F}{R4} * (-V_T) - \frac{R_F * E_{cd}}{R6} \qquad (2.4)$$

$$y = mx + b$$

Donde : $R_F/R4 = 10$ si se elige R4 = 10K entonces R_F = R5 =100K.

También correlacionando los términos correspondientes a la desviación de cd de la ecuación se tiene:(- R_F / R6)*Ecd = -27.3 (2.5)

suponiendo que se conecta Ecd a la fuente de +15V. Dado que R_F = 100k entonces R6 = 54.9K (2.6)

Observese que Ecd es un voltaje positivo y que V_T es un voltaje negativo que está a la entrada del sumador inversor. Dado que el LM335 genera un voltaje positivo, V_T, el amplificador inversor con ganancia de –1 se utiliza para generar - V_T, como se muestra en el circuito completo de la figura 2.7 .

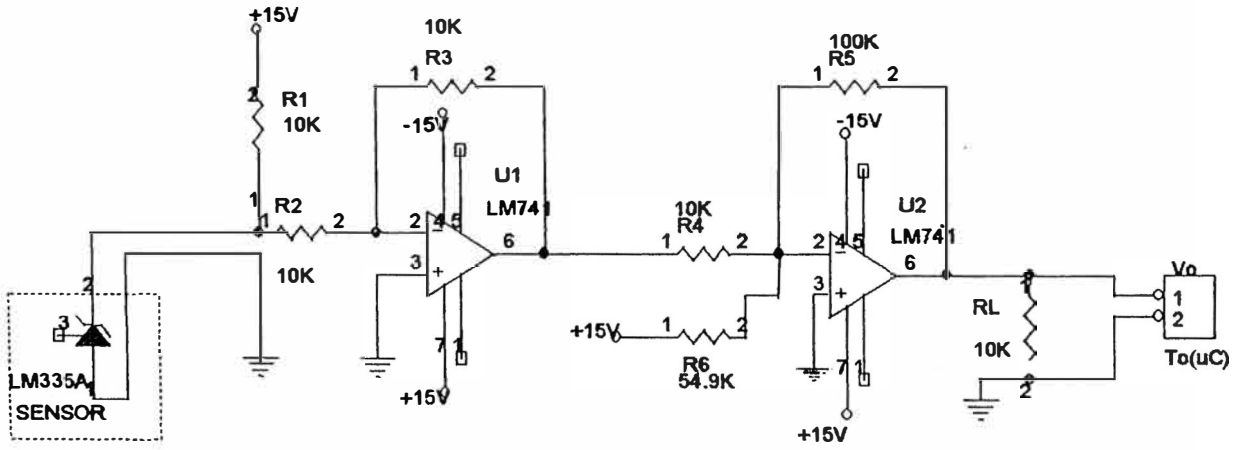


Figura 2.7 Diseño del CAS acorde al sistema de medición de temperatura

2.2.5 Construcción de la sonda de prueba

Se utiliza un tubo metálico o de vidrio sellado, como muestra en la figura 2.8

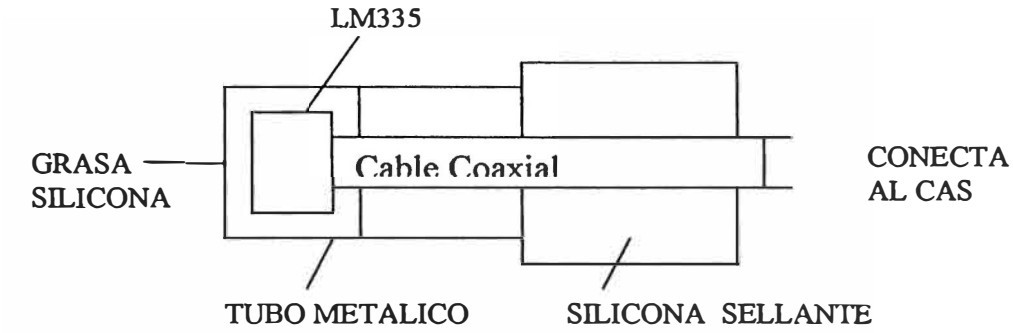


Figura 2.8 Disposición de la sonda

La grasa de silicona que rodea al sensor LM335 garantiza una adecuada transferencia de calor . El otro extremo del tubo se cierra con sellante de silicona de alta temperatura para evitar la entrada del líquido. La sonda se conecta al sistema a través de un cable coaxial.

2.2.6 Modelo del proceso de medición de temperatura

Considerando la tina mostrada en la figura 2.9 para realizar el baño de cobre, es necesario controlar la temperatura con precisión, para esto es necesario conocer las características térmicas de la tina.

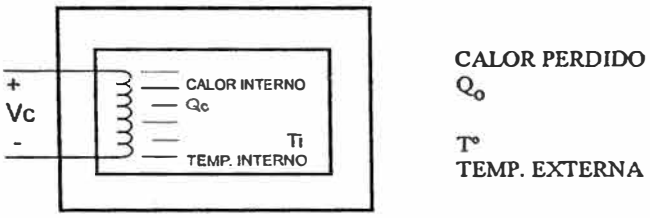


Figura 2.9 Disposición de la tina

Asumiendo que el calor interno dentro de la tina es igual en todo el recinto, el calor entregado a la tina se modela de la siguiente forma:

El flujo de energía neta a una sustancia afecta a la temperatura de la sustancia de acuerdo con

$$C \frac{d}{dt}(T_i - T_o) = Qc - Qe \qquad (2.7)$$

$$Qc = \frac{V_c^2}{Rc} \qquad (2.8)$$

$$Qe = \frac{Ti - To}{Re} \qquad (2.9)$$

Donde: Ti = Temperatura interior en °K ó en °C

To = Temperatura exterior en °K ó en °C

Qc = Flujo de calor interno en J/s o Watt

Qe = Flujo de pérdida en J/s ó Watt

C = es la capacidad térmica en J/ °K

Re = es la resistencia térmica en °K/Watt ó °K/(J/s)

Rc = resistencia eléctrica en ohmios

Vc = voltaje de calefacción

En este caso se usará la temperatura expresada en °C.

Considerando constante la temperatura externa en la tina se puede expresar la dinámica de la tina como:

$$\frac{dTi}{dt} + \frac{1}{Re\,C}Ti = \frac{To}{Re\,C} + \frac{V_c^2}{Rc\,C} \qquad (2.10)$$

y cuando Ti alcanza su estado estacionario

$$Ti = To + \frac{Re}{Rc} V_c^2 \qquad (2.11)$$

Los datos de respuesta transitoria son rápidos y relativamente fáciles de obtener. Son representativos de las señales naturales a las cuales está expuesto el sistema, y así un modelo basado sobre estos datos puede ser una base confiable para el diseño de un sistema de control.

La tabla 2.3 contiene datos experimentales de la caída de temperatura cuando se desconecta la tina y la tabla 2.4 tabula el tiempo de calentamiento versus la temperatura del interior en estado estable, para ambas tablas la temperatura ambiente es de 30 ° C.

Tabla 2.3 Datos de caída de temperatura.

Tiempo	Temperatura(°C)
14:23:10	120.0
14:31:00	108.0
14:39:30	92.0
14:48:35	80.5
15:05:00	63.0
15:34:00	40.5

Tabla 2.4 Temperatura de la tina en estado estable.

Tiempo	Temperatura(°C)
9.8	34.1
20.0	50.5
29.5	74.0
40.0	110.0

La curva exponencial 120 $e^{-t/\tau}$ obtenida con los datos de la tabla 2.3, permite calcular su constante de tiempo, $\tau = ReC = 3000$ seg.

La curva cuadrática obtenida con los datos de la tabla 2.4, permite obtener la relación

$$K_h = \frac{Re}{Rc} = 0.05 \qquad (2.12)$$

La dinámica de la tina resulta:

$$\frac{dT_i(t)}{dt} = -\frac{1}{3000}T_i(t) + \frac{1}{3000}T_O(t) + \frac{1}{60000}V_C^2 = f(T_i, T_O, V_C) \qquad (2.13)$$

$\theta_i = Ti$, $\theta_0 = T_0$, $V_c = Vc$

Luego la ecuación diferencial del proceso:

$$\frac{d\theta_i}{dt} = -\frac{1}{3000}\theta_i + \frac{1}{3000}\theta_0 + \frac{1}{60000}V_c^2 = f(\theta_i, \theta_0, V_c) \qquad (2.14)$$

definiendo las variables:

$$x = \theta_i, \quad V = \theta_0, \quad U = V_c$$

Se puede escribir la ecuación (2.14) como:

$$x = -\frac{1}{3000}x + \frac{1}{3000}V + \frac{1}{60000}U^2 = f(x, U; , V) \tag{2.15}$$

la salida es:

$$y = \theta_i = x = h(x, u, w) \tag{2.16}$$

La ecuación linealizada del proceso se encuentra aplicando el jacobiano para el punto de operación $\overline{\theta}_i = 50\,^{\circ}C$, $\overline{\theta}_i = 30\,^{\circ}C$, $\overline{V_c} = 20\ Volt.$**y considerando las variables residuales:**

$$x = \theta_i - \overline{\theta}_i\,, \; u = V_c - \overline{V}_c, \; V = \theta_0 - \overline{\theta_0}$$

Así: la ecuación de estado:

$$\dot{x} = \left(\frac{\partial f}{\partial x}\right)_{\bar{x}} x + \left(\frac{\partial f}{\partial y}\right)_{\bar{u}} u + \left(\frac{\partial f}{\partial V}\right)_{\bar{V}} v$$

$$x = -\frac{1}{3000}x + \frac{2}{60000}U\,\Big|_{\overline{U}=\overline{V}_c=20} u + \frac{1}{3000}v$$

$$\dot{x} = -\frac{1}{3000}x + \frac{1}{1500}\,|u + \frac{1}{3000}v$$

A = -(1/3000), B = (1/1500), E = (1/3000)

$$x = Ax + Bu + Ev \tag{2.17}$$

La ecuación de salida:

$$\dot{x} = \left(\frac{\partial h}{\partial x}\right)_{\bar{x}} x + \left(\frac{\partial h}{\partial y}\right)_{\bar{u}} u + \left(\frac{\partial h}{\partial w}\right)_{\bar{w}} w$$

$$y = 1x + 0u + 0w$$

$$C_c = 1, \; D_c = 0, \; F_c = 0$$

$$y = C_c\, x \tag{2.18}$$

CAPITULO III

LINEALIZACION Y DISCRETIZACION

3.1 Linealización de la Dinámica de la Tina

Para efectos de linealización se definen las siguientes variables residuales :

$$\theta_i(t) = T_i - T_i^o \qquad (3.1)$$

$$\theta_o(t) = T_0 - T_0^o \qquad (3.2)$$

$$v_c = V_c - V_c^o \qquad (3.3)$$

Luego, la ecuación no lineal 2.15 puede ser linealizada empleando el jacobiano para el punto de operación:

$$T_i^o = 50°C \quad , \quad T_o^o = 30°C \quad , \quad V_c^o = 20V \qquad (3.4)$$

resultando:

$$\frac{d\theta_i(t)}{dt} = \left[\frac{\partial f}{\partial T_i}\right]_{T_i^o} \theta_i + \left[\frac{\partial f}{\partial T_o}\right]_{T_o^o} \theta_o + \left[\frac{\partial f}{\partial V_c}\right]_{V_c^0} v_c \qquad (3.5)$$

$$\frac{d\theta_i(t)}{dt} = \frac{1}{3000}\theta_i(t) + \frac{1}{3000}\theta_0 + \frac{1}{1500}V_c \qquad (3.6)$$

El sistema resultante es de primer orden:

- La representación linealizada en el espacio de estado, sin la presencia de parámetros variantes en el tiempo, es el siguiente:

  $x = Ax + Bu + Ew$ (3.7)

  $y = Cx + Du + Fv$ (3.8)

donde

A = matriz de estado, B = matriz de control , E = matriz de disturbios de estado .

C = matriz de salida de los estados, D = matriz de salida de las entradas.

F = matriz de disturbios en las salidas.

Sin la presencia de disturbios se puede hacer: v = w = 0

3.1.1 Controlabilidad y Observabilidad

Para que un proceso sea completamente controlable, y la matriz de controlabilidad sólo contenga filas y columnas independientes, se debe cumplir que :

Rango(M) = rango ([B AB.....A^{N-1} B]) = n

Donde n es el orden del proceso.

Para que el proceso dinámico sea observable, la matriz de observabilidad N debe cumplir:

$$\text{Rango } [N] = n$$

$$N = [C\ CA\ .....CA^{n-1}]^T \qquad (3.9)$$

3.1.2 Ecuación de Estado Discreta

Para un tiempo discreto K = t /T y asumiendo que entre muestras la señal de control u es constante, luego para una señal de entrada u(t) = u(KT), KT< (K+ 1)T se obtienen las ecuaciones de estado discretas y de salida:

$$X(K+1) = G\,x(K) + H\,u(K) \qquad (3.10)$$

$$Y(K) = C\,x(K) + D\,u(K) \qquad (3.11)$$

Donde:

$$G = \phi(T) = e^{AT} \qquad H = \left[\int_0^T \phi(\lambda)d\lambda\right]B \qquad (3.12)$$

3.2 Simulación de la respuesta del modelo discreto

Las matrices del proceso en tiempo discreto, así como la respuesta del proceso se han determinado usando el programa Matlab, según puede apreciarse en el archivo de nombre TINA.m

```
% TINA.m

A= [-1/3000];

B= [1/1500];

C= [1];

D= [0];

E= [1/3000];

%       CONTROLABILIDAD Y OBSERVABILIDAD DEL PROCESO
```

```
r AB = rank(ctrb(A,B));
%  r AB = 1
%  Completamente controlable
rAC = rank(obsv(A,C)) ; % rAC = 1 => Completamente observable
eigA = eig(A);              % COMPUTA EIGENVALORES
% eigA =A= -1/3000;
%       CONVERSION AL ESPACIO DISCRETO
 T=10;                      % TIEMPO DE MUESTREO
[G,H,C,D]=c2dm(A,B,C,D,T,'zoh');
[G,F,C,D]=c2dm(A,E,C,D,T,'zoh');
%       ECUACION DE ESTADO DISCRETA
%        Teta(K+1)=G*Teta(K)+H*vc(K)+F*Teta_o
eigG = eig(G);
eigH = eig(H);
eigF = eig(F);
% eigG = G = 0.9967;      % POR SER PROCESO DE PRIMER ORDEN
% G = 0.9967; H = 0.0067; F = 0.0333;
%       RESPUESTAS AL ESCALON
[Y , X ,t] = step(A,B,C,D);
[YY,XX] = dstep(G,H,C,D);
tt = linspace(0,size(YY,1)*T,size(YY,1));
subplot(211)
plot(t,Y); grid
ylabel('Y(t)')
```

```
xlabel('tiempo en segundos')

subplot(212)

plot(tt,YY); grid

xlabel('tiempo en segundos')

ylabel('Y(K) discreta')
```

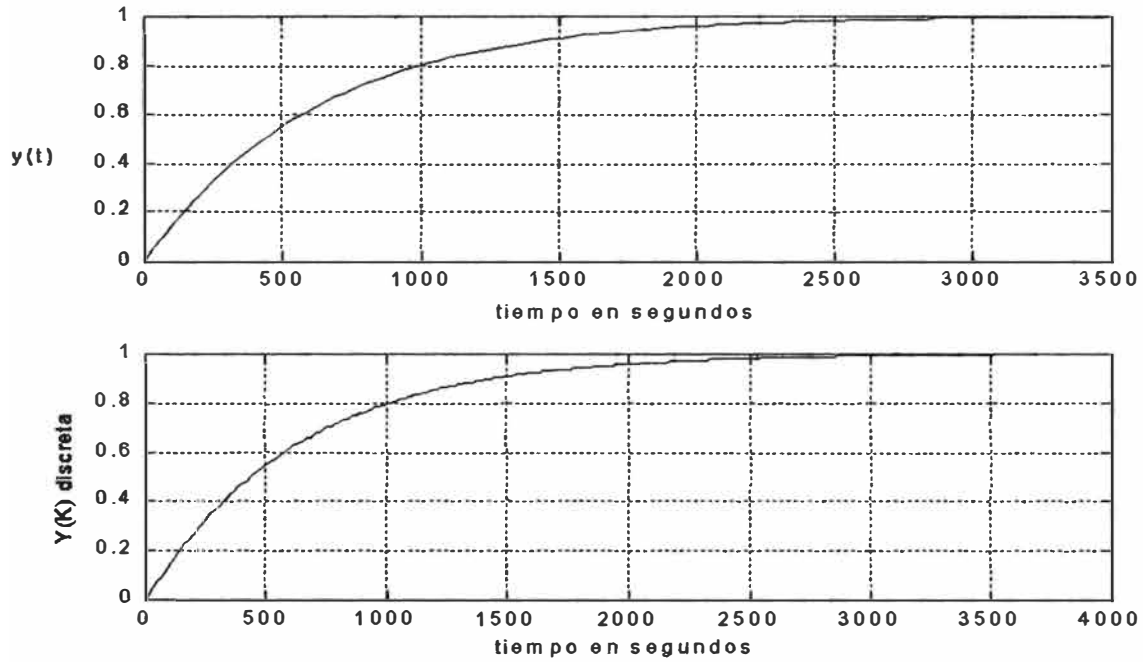


Figura 3.1. Respuesta al escalón de la temperatura interior de la tina de baño de cobre.

CAPITULO IV

CONTROL PID EN TIEMPO DISCRETO

4.1 Introducción al control PID en tiempo discreto

El algoritmo de control PID es bastante usado para mantener las variables del proceso (mediciones) en un valor predeterminado (setpoint).

El control de temperatura es la forma más común de un control de lazo cerrado. Por ejemplo , en un sistema simple de control de temperatura de una caldera se desea mantener la temperatura en un valor determinado (setpoint), s(t). La salida de control fija el estado de la válvula del actuador para aplicar menos calor a la caldera, si es que la temperatura momentánea dentro de la caldera es mayor que la del setpoint, y más calor si la temperatura es menor.

Mediante el algoritmo PID se calcula la salida con la adición de tres términos. Un término es proporcional al error (el error está definido como el setpoint menos el valor momentáneo de la medición). El segundo término es proporcional a la integral del error sobre el tiempo, y el tercer término es proporcional a la velocidad del cambio (primera derivada) del error. La forma general de la ecuación de un controlador PID en forma análoga es:

$$M(t) = Kp * (e(t) + Ki \int e(t)\,dt + Kd\, de(t)/dt)$$

Donde :

M(t) : salida de control

Kp : ganancia proporcional

Ki : constante de integración

Kd : constante de derivación

s(t) : setpoint momentáneo

x(t) : valor momentáneo de la variable medida

e(t) : error en función del tiempo = s(t) – x(t)

Los valores de las constantes Kp, Ki y Kd se deben definir según el comportamiento de control que se desee tener. La constante Ki es conocida en muchos libros como (1/Ki) . Sólo es cuestión de fijar adecuadamente las unidades . La versión Ki presentada aquí , se prefiere debido a que un aumento de Ki origina también un aumento de la acción de la ganancia integrativa, así como lo hacen también Kp y Kd con la ganancia proporcional y la ganancia derivativa respectivamente.

El término proporcional de la ecuación del PID contribuye con un monto de la salida de control, proporcional al error momentáneo del proceso. Mientras más grande sea el valor de Kp, mas fuerte reaccionará el sistema a la diferencia entre el setpoint y la temperatura momentánea . Un controlador PID puede ser usado como un simple controlador proporcional haciendo los valores Ki y Kd igual a cero.

Un control proporcional simple no toma en cuenta los cambios de carga en el proceso bajo control.

La parte integral del controlador adiciona a la salida de control un término que es proporcional a la suma de todos los errores previos del proceso en el sistema. Mientras haya un error en el proceso, el término integral incrementará más y más la salida de control hasta que la suma de todos los errores previos sea cero. El término "reset rate" es usado para describir la acción integrativa de un controlador PID.

La parte derivativa del controlador PID adiciona a la salida de control un valor proporcional a la velocidad de cambio del error del proceso , esto da como resultado que la respuesta al control sea más rápida.

El algoritmo de un controlador PID empleado en una computadora (µprocesador), esta basado en la teoría de control en tiempo discreto. La ecuación equivalente del PID en tiempo discreto es:

$$m(i) = Kp * [e(i) + T * Ki \Sigma e(k) + (kd /T)* (e(i) - e(i-1))]$$

donde:

T : intervalo de muestreo

e(i) : error en el i-avo intervalo de muestreo

e(i-1): error en el intervalo de muestreo anterior

m(i) : salida de control

Kp : cte. Proporcional

Ki : cte. de integración

Kd : cte. de derivación

Vemos que el término integral ha sido reemplazado por una sumatoria y el término derivativo por una aproximación de ecuaciones de diferencia de primer orden. En la práctica actual, los términos de diferencia de primer orden, [e(i)-e(i-1)], son muy susceptibles a los problemas de ruido.

Un problema común en los sistemas de control discretos se origina por la sumatoria de los términos de error debido a la acción integral de la ecuación de control. Si un proceso mantiene un error por un largo periodo de tiempo, es posible que la sumatoria forme un valor numérico muy grande que saturará la salida del D/A, y deberá pasar un tiempo bien grande hasta que este valor esté otra vez dentro de los límites de saturación del D/A.

Como se puede apreciar, el algoritmo matemático para implementar en un procesador un controlador PID discreto es bien sencillo. Sólo hay que tomar en cuenta adicionalmente, la velocidad de procesamiento del procesador, la longitud de los datos, cálculo matemático (punto fijo o punto flotante) y la cantidad de memoria a emplear.

4.2 Diseño del sistema de control PID del baño de cobre

El Microcontrolador 18F452 con la salida del sensor de temperatura acondicionada compara con el setpoint de temperatura y controla un relé de comando del calefactor.

El Microcontrolador 18F452 con la rutina PID_main.asm recibe la señal análoga de la muestra de voltaje que es proporcional a la corriente de la electrólisis y la compara con el setpoint adecuado convirtiendo la diferencia en un dato digital que actualiza la variable error0:error1, tanto como el pid_stad1 signo del error, entonces realiza el control PID discreto y el resultado es puesto en la variable pid_out0:pid_out2, con el bit de signo de pid_stad1. El valor en pid_out0:pid_out2 es convertido al correcto valor (PWM) que puede ser aplicado a la planta que para este caso es una planta de primer orden $G(s) = [k / (1 + \tau s)]*e^{-sL}$.

Experimentalmente aplicando una señal de escalón se puede ver la respuesta del proceso y con los parámetros que se obtienen L y T por intermedio de las tablas de Ziegler-Nichols para la sintonización de un controlador PID, el método de la curva de reacción se obtienen los parámetros Kp, Ki y Kd que requiere el controlador PID discreto del µC 18F452.

4.3 Simulación del sistema de control PID

El sistema a controlar

Para comenzar abordaremos el problema del control de un sistema simple de primer orden con retardo, definido por la función de transferencia.

$$G(s) = \frac{k}{1+\tau s} e^{-sL}$$

donde k representa la ganancia estática del sistema,τ es su constante de tiempo y L es el retardo del mismo.

Este tipo de sistemas, a pesar de su sencillez, modelan bastante bien una amplia clase de sistemas dinámicos que involucran generalmente fenómenos de transporte de materia como sucede en muchos procesos químicos, térmicos y muchos otros muy comunes en la industria de procesos.

Comenzaremos viendo la respuesta de este sistema a lazo abierto ante una entrada en escalón, y para ello construiremos el siguiente sistema en SIMULINK.

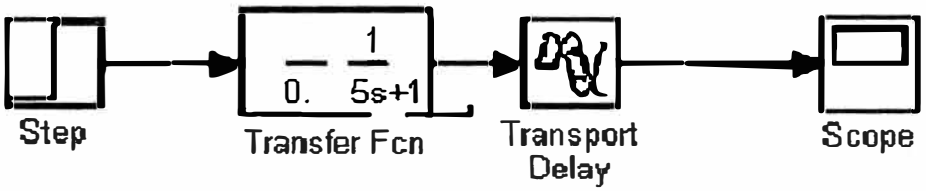


Figura 4.1 Respuesta al escalón de Sistema de primer orden a lazo abierto con SIMULINK

para un sistema con los siguiente parámetros: $2 = k$, $\tau = 0.5$s y $L=0.8$ s.

En la simulación observaremos la respuesta esperada, es decir la respuesta de un sistema de primer orden con un retardo de 0.8 segundos respecto a la entrada en escalón marcada.

Controlador PID

El controlador PID es una estructura de control en la que la señal de control del proceso se expresa en función del error, $e(t)=y_{ref}(t) - y(t)$, según la expresión estandar:

$$u(t) = K_p\left(e(t) + \int_0^t K_i e(\tau)d\tau + K_d \frac{de(t)}{dt} \right)$$

donde *Kp* , *Ki* y *Kd* corresponden respectivamente a las constantes Proporcional, Integral y Derivativa del controlador.

La expresión anterior puede igualmente expresarse como la siguiente función de transferencia del controlador PID.

$$K(s) = \frac{U(s)}{E(s)} = K_p\left(1 + \frac{K_i}{s} + K_d s \right)$$

Esta función de transferencia puede implementarse en SIMULINK

Es posible construir la estructura del PID partiendo de bloques elementales de SIMULINK del siguiente modo:

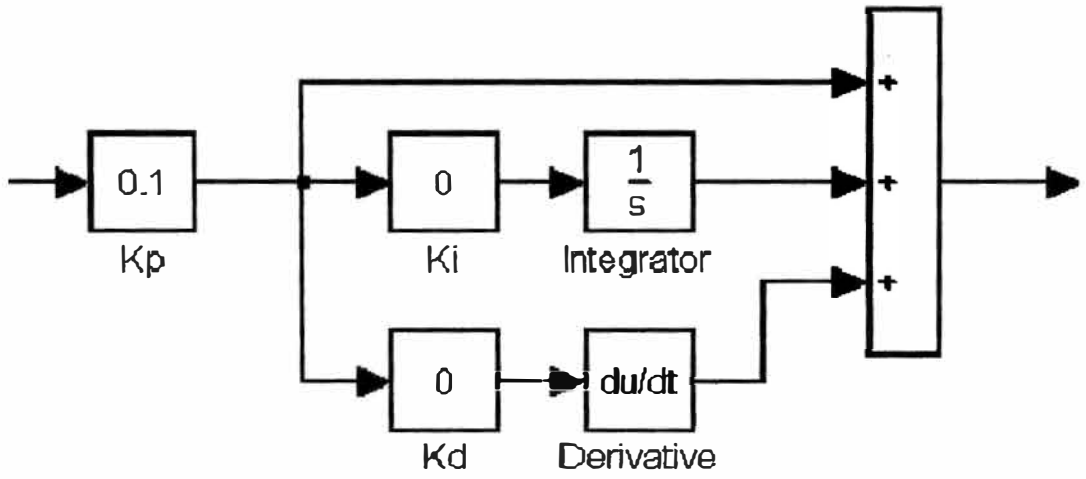


Figura 4.2 Estructura PID con bloques elementales de SIMULINK

Esta estructura será la que emplearemos en lo sucesivo.

Control a lazo cerrado.

Para comprobar la influencia del controlador PID en el sistema propuesto construiremos la siguiente estructura de control realimentada.

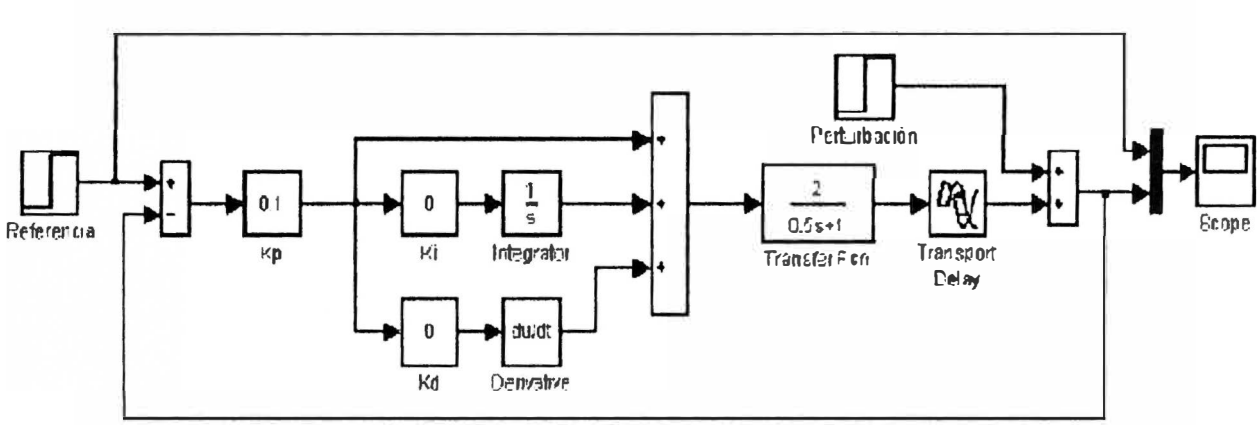


Figura 4.3 Respuesta al escalón de sistema de primer orden a lazo cerrado con SIMULINK

Esta estructura representa un control PID clásico que incluye el controlador en la cadena directa del sistema.

Ajuste del PID. Reglas del Ziegler-Nichols.

Para un ajuste inicial del controlador anterior, emplearemos las conocidas reglas de Ziegler-Nichols.

Primer método de Ziegler-Nichols

Las características del sistema estudiado permite emplear el método de respuesta a escalón de Ziegler-Nichols que caracteriza un sistema mediante dos parámetros, L y T, obtenidos a partir de la respuesta a lazo abierto del mismo como representa la figura 4.4.

Según este procedimiento de sintonización los parámetros del controlador pueden obtenerse de acuerdo con las expresiones de la siguiente tabla.

Controlador	K_p	K_i	K_d
P	$\frac{T}{L}$	0	0
PI	$0.9\frac{T}{L}$	$\frac{0.3}{L}$	0
PID	$1.2\frac{T}{L}$	$\frac{1}{2L}$	$0.5L$

Tabla 4.1: Parámetros del PID según el método de respuesta a Escalón de Ziegler-Nichols

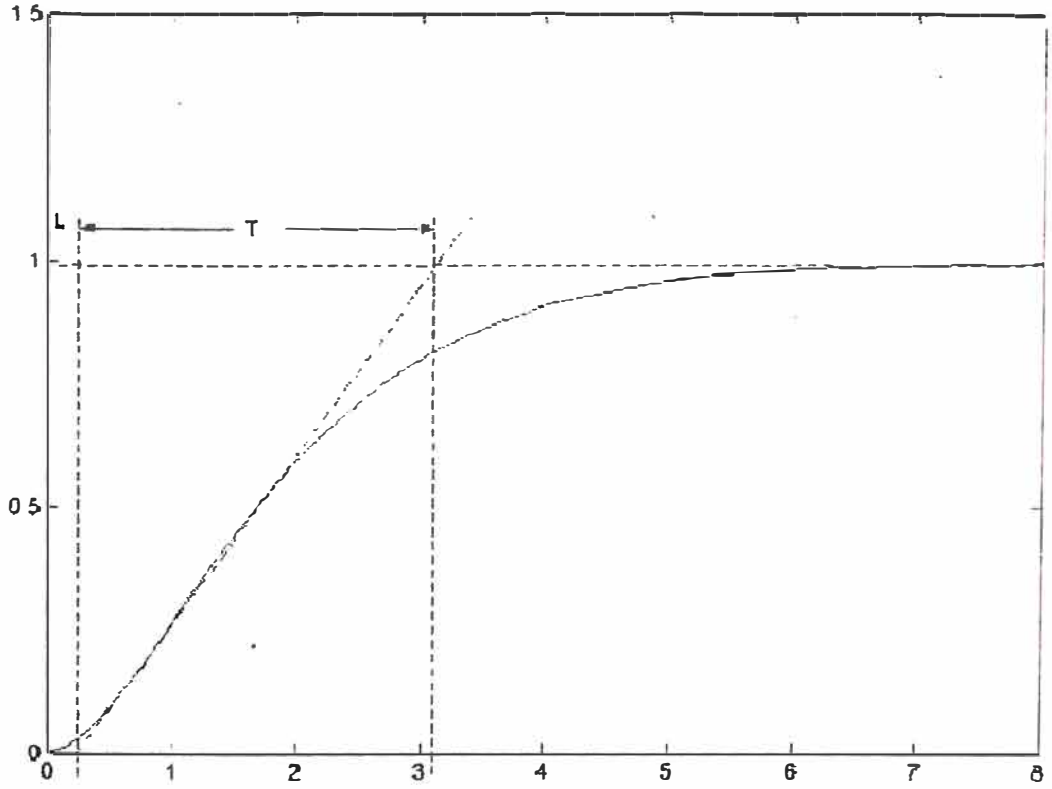


Figura 4.4 Parámetros L y T de la Respuesta al escalón del Sistema de primer orden a lazo abierto obtenidos con SIMULINK

De este modo a partir de la respuesta a lazo abierto del sistema, calcularemos los controladores P, PI y PID apropiados para nuestro sistema.

CONVERSION ANALOGA /DIGITAL CON PIC 18F452

Para sensar la señal análoga, diferenciarla con un setpoint y tenerla en forma digital y luego esa señal tratarla con controlador PID (PIDInt).

```
LIST  P=18F8452
 INCLUDE <P18F452.INC>

VARXEQU   0X40
VARY EQU   0X41
VARZ EQU   0X42

 ORG  0X2007
        DW  0X3B32
ORG  0X0000
INICIO        BSF STATUS,RP0
        BCFSTATUS,RP1
        BSF TRISB,0; PULSADOR
        BCF ADCON1,PCFG0      ;Configura siete bits como i/o digital
        BSF ADCON1,PCFG1      ;y como única entrada análoga
        BSF ADCON1,PCFG2      ;el pin RA0 cuyo voltaje de
        BSF ADCON1,PCFG3      ;referencia es 5v (+) y 0v (-)
        BSF ADCON1,ADFM       ;justificación a la derecha del resultado
        CLRF TRISC            ;salidas para mostrar los
        CLRF TRISD            ;10 bits de conversión
        BSF TRISA,0           ;entrada para la señal análoga
        BCF STATUS,RP0
BCF ADCON0,CHS0               ;selección del
        BCF ADCON0,CHS1       ;canal
        BCF ADCON0,CHS2       ;a utilizar (ra0)

        BCF ADCON0,ADCS0      ;selección de f/32
        BSF ADCON0,ADCS1      ;para el reloj de conversión

        BSF ADCON0,ADON       ;activación del módulo conversor

 STOP BTFSS PORTB,0
        GOTO STOP
 NEW  BSF ADCON0,GO_DONE ;inicia la conversión

 WAIT BTFSC ADCON0,GO_DONE      ;salta si ya culminó la conversión
        GOTO WAIT
        MOVFW ADRESH ;w = ADRESH
        MOVWF error0:error1     ;error0:error1H = W
        BSF STATUS,RP0
        MOVFW ADRESL  ;W = ADRESL
        BCF STATUS,RP0
        MOVWF error0:error1     ;error0:error1L = W
```

```
        CALL DELAY
        GOTO NEW
DELAY   MOVLW   0X2
        MOVWF   VARZ
ICY     MOVLW   0XFF
        MOVWF   VARY
ICX     MOVLW   0XFF
        MOVWF   VARX
DCX     DECFSZ  VARX
        GOTO    DCX
        DECFSZ  VARY
        GOTO    ICX
        DECFSZ  VARZ
        GOTO    ICY
        RETURN
 END
```

CONVERSION DE DATOS DIGITALES A PWM

```
LIST P=18F452
INCLUDE <P18F452.INC>

ORG 0X2007
        DW 0X3B32

ORG 0X0000

INICIO      BSF STATUS,RP0
        BCF STATUS,RP1
        BCF TRISC,2 ; PORTC,2 OUT
        BSF TRISD,3 ; PORTD,3 IN
        MOVLW .124 ; PERIODO = 0.5ms
        MOVWF PR2 ;   (F=4MHZ)

BCF STATUS,RP0
BCF T2CON,TMR2ON ;TIMER2 OFF
BCF T2CON,T2CKPS1
BSF T2CON,T2CKPS0 ;PREESCALA =4
BSF CCP1CON,5 ;1
BCF CCP1CON,4 ; 0
MOVLW .62
MOVWF CCPR1L ; CICLO UTIL 50%
BSF CCP1CON,CCP1M3
BSF CCP1CON,CCP1M2 ; MODO PWM
WAIT BTFSS PORTD,3
        GOTO WAIT

        BSF T2CON,TMR2ON ; TIMER2 ON
STOP GOTO STOP
        END
```

PIDInt

```
;*****************************************************************************
;* This file contains PID functions with interrupt.
;*****************************************************************************
;*File name:     PIDInt.asm
;*Dependencies: p18f452.inc (change to specific application requirements)
;*Processors:    PIC18
;*Assembler:     MPASMWIN 02.70.02 or higher
;*Linker:        MPLINK 2.33.00 or Higher
;*Company:       Microchip Technology, Inc.
;*
;* Software License Agreement
;*
;* The software supplied herewith by Microchip Technology Incorporated
;* (the "Company") for its PICmicro® Microcontroller is intended and
;* supplied to you, the Company's customer, for use solely and
;* exclusively on Microchip PICmicro Microcontroller products. The
;* software is owned by the Company and/or its supplier, and is
;* protected under applicable copyright laws. All rights are reserved.
;* Any use in violation of the foregoing restrictions may subject the
;* user to criminal sanctions under applicable laws, as well as to
;* civil liability for the breach of the terms and conditions of this
;* license.
;*
;* THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
;* WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
;* TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
;* PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
;* IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
;* CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;*
;*
;*
;* Author                Date            Comment
;*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;* C.Valenti              June 29, 2004    Initial Release (V1.0)
;*
;* Revisions:
```

```
                                        PIDInt
;*      7/8/04  -Removed unused variables a_Err1Lim & a_Err2Lim
;*                      Modified code after the "restore_limit" label to reflect
using
;*                      aErr1Lim & aErr2Lim defined constants.
;*                      -Changed constant: #define derivCount to #define
derivCountVal
;*                      -pidStat1 bit comments were corrected to the correct bit
#
;*                      -In the PidInterrupt routine, the " movlw derivCountVal
" was added
;*                      for loading the derivCount variable.
;*
;*      10/20/04 -Added bra statment to the Derivative routine
;*                       Amended code for checking the a_Error2 limits.
;*
;*****************************************************************

;PID Notes:
;       PROPORTIONAL = (system error * Pgain )
;       System error = error0:error1
;
;       INTEGRAL = (ACUMULATED ERROR  * Igain)
;       Accumulated error (a_error) = error0:error1 + a_Error0:a_Error2
;
;       DERIVATIVE = ((CURRENT ERROR  - PREVIOUS ERROR) * Dgain)
;       delta error(d_error) = errro0:error1 - p_error0:p_error1
;
;       Integral & Derivative control will be based off sample periods of "x"
time.
;       The above sample period should be based off the PLANT response
;       to control inputs.
;               SLOW Plant response = LONGER sample periods
;               FAST Plant response = SHORTER sample periods
;
;       If the error is equal to zero then no PID calculations are completed.
;
;       The PID routine is passed the 16- bit errror data by the main
application
;       code through the error0:error1 variables.
;       The sign of this error is passed through the error sign bit:
```

```
                                        PIDInt
;                       pidStat1,err_sign

;       The PID outputs a 24-bit vaule in pidOut0:pidOut2 and the sign of this

;       result is the pid_sign bit in the pidStat1 register.

;-----------------------------------------------------------------------


        list            p=18F452
        #include        <p18f452.inc>

;***** SYSTEM CONSTANTS
#define aErr1Lim                0x0F    ;accumulative error limits (4000d)
#define aErr2Lim                0xA0

#define timer1Hi                0x3E    ;Timer1 timeout defined by timer1Lo &
timer1Hi
#define timer1Lo                0x0D    ;this timout is based on Fosc/4

#define derivCountVal   .10             ;determies how often the derivative term
will be executed.

;#define        pid_100                 ;comment out if not using a 0 - 100%
scale


        EXTERN  FXM1616U,FXD2416U,_24_BitAdd,_24_bit_sub
        EXTERN  AARGB0,AARGB1,AARGB2,AARGB3
        EXTERN  BARGB0,BARGB1,BARGB2,BARGB3

        GLOBAL  error0, error1, pidStat1

;***** VARIABLE DEFINITIONS

pid_data        UDATA
#ifdef  pid_100
percent_err     RES     1                       ;8-bit error input, 0 - 100% (0
- 100d)
percent_out     RES     1                       ;8-bit output, 0 - 100% (0 -
100d)
#endif

derivCount      RES 1                   ;This value determins how many times the
Derivative term is
                                                ;calculated based on
each Integral term.
pidOut0         RES     1                       ;24-bit Final Result of PID for
the "Plant"
pidOut1         RES     1
pidOut2         RES     1
error0          RES     1                       ;16-bit error, passed to the PID
error1          RES     1
a_Error0        RES     1                       ;24-bit accumulated error, used
for Integral term
a_Error1        RES     1
a_Error2        RES     1
p_Error0        RES     1                       ;16-bit previous error, used for
Derivative term
p_Error1        RES     1
d_Error0        RES     1                       ;16-bit delta error (error -
previous error)
d_Error1        RES     1

prop0           RES     1                       ;24-bit proportional value
prop1           RES     1
prop2           RES     1
integ0          RES 1                   ;24-bit Integral value
```

```
                                        PIDInt
integ1          RES     1
integ2          RES     1
deriv0          RES 1                   ;24-bit Derivative value
deriv1          RES     1
deriv2          RES     1

kp                      RES     1                       ;8-bit proportional Gain
ki                      RES     1                       ;8-bit integral Gain
kd                      RES     1                       ;8-bit derivative Gain

pidStat1        RES     1                       ;PID bit-status register
pidStat2        RES     1                       ;PID bit-status register2
tempReg         RES     1                       ;temporary register

;                                       pidStat1 register

;
```

```
;       |  bit 7   |   bit 6    |  bit 5 |    bit 4   |   bit 3    |  bit 2   |  bit 1   |  bit 0   |
;       | pid_sign | d_err_sign |   mag  | p_err_sign | a_err_sign | err_sign | a_err_z  |  err_z   |
;
```

```
err_z           equ     0                       ;error zero flag, Zero = set
a_err_z         equ     1                       ;a_error zero flag, Zero = set
err_sign        equ     2                       ;error sign flag, Pos = set/ Neg = clear
a_err_sign      equ     3                       ;a_error sign flag, Pos = set/ Neg = clear
p_err_sign      equ     4                       ;a_error sign flag, Pos = set/ Neg = clear
mag                     equ     5                       ;set = AARGB magnitude, clear = BARGB magnitude
d_err_sign      equ     6                       ;d_error sign flag, Pos = set/ Neg = clear
pid_sign        equ     7                       ;PID result sign flag, Pos = set/ Neg = clear

;       _____ pidStat2 register
;       | bit 7 |  bit 6  |  bit 5  |   bit 4   |   bit 3   | bit 2 |  bit 1  |  bit 0  |
;       |       |         |         |           |           |       |         | d_err_z |
;
```

```
d_err_z         equ     0                       ;d_error zero flag, Zero = set

_PIDCODE        CODE                    ;start PID code here
;*****************************************************************;
; Function: PidInit
;
;                                                                 ;
; PreCondition: Called by the application code for PID initalization    ;
;                                                                 ;
; Overview: PID variables are cleared, PID gains are given values,
;
;                       flags are initialized.
;
;                                                                 ;
```

```
                                        PIDInt
; Input:
                                                ;
;                                                                       ;
; Output: none
;                                                                       ;
;                                                                       ;
; Side Effects: W register is changed
        ;
;                                                                       ;
; Stack requirement: 1 levels deep                                      ;
;                                                                       ;
;***********************************************************************;
PidInitalize:
        GLOBAL  PidInitalize
        clrf    error0
        clrf    error1
        clrf    a_Error0
        clrf    a_Error1
        clrf    a_Error2
        clrf    p_Error0
        clrf    p_Error1
        clrf    d_Error0
        clrf    d_Error1

        clrf    prop0
        clrf    prop1
        clrf    prop2
        clrf    integ0
        clrf    integ1
        clrf    integ2
        clrf    deriv0
        clrf    deriv1
        clrf    deriv2

        clrf    kp
        clrf    ki
        clrf    kd

        clrf    pidOut0
        clrf    pidOut1
        clrf    pidOut2

        clrf    AARGB0
        clrf    AARGB1
        clrf    AARGB2
        clrf    BARGB0
        clrf    BARGB1
        clrf    BARGB2

        movlw   .160                                    ;10 x 16, Kp, Ki & Kd
are 8-bit vlaues that cannot exceed 255
        movwf   kp                                              ;Enter the PID
gains scaled by a factor of 16, max = 255

        movlw   .160                                    ;10 x 16
        movwf   ki

        movlw   .160                                    ;10 x 16
        movwf   kd

        movlw   .10
        movwf   derivCount                              ;derivative action =
TMR1H:TMR1L * derivCount
        bcf             pidStat1,err_z                  ;start w/error not equal
to zero
        bsf             pidStat1,a_err_z                ;start w/a_error equal
to zero
        bsf             pidStat2,d_err_z                ;start w/d_error equal
```

```
                                        PIDInt
to zero
        bsf             pidStat1,p_err_sign             ;start w/ previous error
= positive
        bsf             pidStat1,a_err_sign             ;start w/ accumulated
error = positive

        bcf             PIR1,TMR1IF                             ;clear T1 flag
        bsf             PIE1,TMR1IE                             ;enable T1
interrupt

        movlw   b'00000001'                             ;configure T1 for Timer
operation from Fosc/4
        movwf   T1CON
        movlw   timer1Hi                                ;load T1 registers with
5ms count
        movwf   TMR1H
        movlw   timer1Lo
        movwf   TMR1L

        return                                                  ;return back to
the main application code

;*************************************************************************;
; Function: PidMain
;
;                                                                         ;
; PreCondition: error0:erro1 are loaded with the latest system error      ;
;                                                                         ;
; Overview: This is the routine that the application code will call       ;
;                       to get a PID correction value. First, the error is
checked        ;
;                       to determine if it is zero, if this is true, then the
PID     ;
;                       code is complete.
                                ;
;
                                                        ;
; Input: error0:error1, sign of the error: pidStat1,err_sign              ;
;                                                                         ;
; Output: prop0:prop2                                                     ;
;                                                                         ;
; Side Effects: W register is changed
        ;
;                                                                         ;
; Stack requirement: 5 levels deep                                        ;
;                                                                         ;
;*************************************************************************;
PidMain:
        GLOBAL  PidMain
        bcf             PIE1,TMR1IE                             ;disable T1 interrupt
#ifdef          pid_100                         ;if using % scale then scale up
PLANT error
        movlw   .40                                             ; 0 - 100% == 0 - 4000d

        mulwf   percent_err,1           ;40 * percent_err --> PRODH:PRODL
        movff   PRODH,error0
        movff   PRODL,error1            ;percentage has been scaled and
available in error0:error1
#endif
        movlw   0
        cpfseq  error0                          ;Is error0 = 00 ?
        bra             call_pid_terms          ;NO, done checking

        cpfseq  error1                          ;YES, Is error1 = 00 ?
        bra             call_pid_terms          ;NO, start proportional term
        bsf             pidStat1,err_z          ;YES, set error zero flag
        bsf             PIE1,TMR1IE                     ;enable T1 interrupt
```

```
                                        PIDInt
        return                                          ;return back to the main
application code

call_pid_terms
        call    Proportional            ;NO, start with proportional term
        call    Integral                        ;get Integral term
        call    Derivative                      ;get Derivative term

        call    GetPidResult            ;get the final PID result that will go
to the system
        bsf             PIE1,TMR1IE                     ;enable T1 interrupt
        return                                          ;return back to the main
application code

;*************************************************************************
; Function: Proportional                                                 ;
;                                                                        ;
; PreCondition: error0:erro1 are loaded with the latest system error     ;
;                                                                        ;
; Overview: This routine will multiply the system's 16-bit error by the ;
;           proportional gain(Kp) --> error0:error1 * Kp                 ;
;                                                                        ;
;                                                                        ;
; Input: error0:error1, sign of the error: pidStat1,err_sign             ;
;                                                                        ;
; Output: prop0:prop2                                                    ;
;                                                                        ;
; Side Effects: W register is changed                                    ;
;                                                                        ;
; Stack requirement: 2 levels deep                                       ;
;                                                                        ;
;*************************************************************************
Proportional:
        clrf    BARGB0
        movff   kp,BARGB1
        movff   error0,AARGB0
        movff   error1,AARGB1
        call    FXM1616U                        ;proportional gain * error

        movff   AARGB1,prop0            ;AARGB2 --> prop0
        movff   AARGB2,prop1            ;AARGB3 --> prop1
        movff   AARGB3,prop2            ;AARGB4 --> prop2
        return                                          ;return to mainline code

;*************************************************************************
; Function: Integral                                                     ;
;                                                                        ;
; PreCondition:  error0:erro1 are loaded with the latest system error    ;
;                                                                        ;
; Overview: This routine will multiply the system's 16-bit accumulated   ;
;           error by the integral gain(Ki)--> a_Error0:a_Error1 * Ki     ;
;                                                                        ;
;                                                                        ;
; Input: a_Error0:a_Error1, sign of a_Error: pidStat1,a_err_sign         ;
;                                                                        ;
; Output: integ0:integ2                                                  ;
;                                                                        ;
; Side Effects: W register is changed                                    ;
;                                                                        ;
; Stack requirement: 2 levels deep                                       ;
;                                                                        ;
;*************************************************************************
Integral:
```

```
                                        PIDInt
        btfsc   pidStat1,a_err_z        ;Is a_error = 0
        bra             integral_zero           ;Yes

        clrf    BARGB0                          ;No
        movff   ki,BARGB1                       ;move the integral gain into
BARGB1
        movff   a_Error1,AARGB0
        movff   a_Error2,AARGB1
        call    FXM1616U                        ;Integral gain * accumulated
error

        movff   AARGB1,integ0           ;AARGB1 --> integ0
        movff   AARGB2,integ1           ;AARGB2 --> integ1
        movff   AARGB3,integ2           ;AARGB3 --> integ2
        return                                          ;return
integral_zero
        clrf    integ0                          ;a_error = 0, clear Integral
term
        clrf    integ1
        clrf    integ2
        return


;*************************************************************************;
; Function: Derivative                                                    ;
;                                                                         ;
; PreCondition:  error0:erro1 are loaded with the latest system error     ;
;                                                                         ;
; Overview: This routine will multiply the system's 16-bit delta
;
;           error by the derivative gain(Kd) --> d_Error0:d_Error1 * Kd ;
;                       d_Error0:d_Error1 = error0:error1 - p_Error0:p_Error1
;
;
                                                                ;
; Input: d_Error0:d_Error1, pidStat2,d_err_z
   ;
;                                                                         ;
; Output: deriv0:deriv2                                                   ;
;                                                                         ;
; Side Effects: W register is changed
        ;
;                                                                         ;
; Stack requirement: 2 levels deep                                        ;
;                                                                         ;
;*************************************************************************;
Derivative:
        btfsc   pidStat2,d_err_z        ;Is d_error = 0?
        bra             derivative_zero         ;YES

        movff   d_Error1,BARGB1         ;result ---> BARGB1
        movff   d_Error0,BARGB0         ;result ---> BARGB0
        movff   kd,AARGB1
        clrf    AARGB0
        call    FXM1616U                        ;Derivative gain * (error_1 -
prv_error1)

        movff   AARGB1,deriv0           ;AARGB1 --> deriv0
        movff   AARGB2,deriv1           ;AARGB2 --> deriv1
        movff   AARGB3,deriv2           ;AARGB3 --> deriv2
        return                                          ;return
derivative_zero
        clrf    deriv0                          ;d_error = 0, clear Derivative
term
        clrf    deriv1
        clrf    deriv2
        return
```

PIDInt

```
;*****************************************************************;
; Function: GetPidResult                                          ;
;                                                                 ;
; PreCondition: Proportional, Integral & Derivative terms have been
;                                 calculated. The Timer1 interrupt is disabled
within          ;
;                                 this routine to avoid corruption of the PID
result.         ;
;                                                                 ;
; Overview: This routine will add the PID terms and then scale down     ;
;                       the result by 16. This will be the final result that is
;
;                       calcualted by the PID code.
                                ;
;                                                                 ;
; Input: prop0:prop2, integ0:integ2, deriv0:deriv2
   ;
;                                                                 ;
; Output: pidOut0:pidOut2                                         ;
;                                                                 ;
; Side Effects: W register is changed
        ;
;                                                                 ;
; Stack requirement: 4 levels deep max.
                ;
;                                                                 ;
;*****************************************************************;
GetPidResult:
        movff   prop0,AARGB0                    ;load Prop term & Integral term
        movff   prop1,AARGB1
        movff   prop2,AARGB2
        movff   integ0,BARGB0
        movff   integ1,BARGB1
        movff   integ2,BARGB2

        call    SpecSign                                ;YES, call routine for
add/sub sign numbers
        btfss   pidStat1,mag                    ;which is greater in magnitude ?
        bra             integ_mag                               ;BARGB is
greater in magnitude
        bra             prop_mag                                ;AARGB is
greater in magnitude

integ_mag                                               ;integ > prop
        bcf             pidStat1,pid_sign               ;PID result is negative
        btfsc   pidStat1,a_err_sign
        bsf             pidStat1,pid_sign               ;PID result is positive
        bra     add_derivative                  ;(Prop + Integ) + derivative

prop_mag                                                ;integ < prop
        bcf             pidStat1,pid_sign               ;PID result is negative
        btfsc   pidStat1,err_sign
        bsf             pidStat1,pid_sign               ;PID result is positive

add_derivative
        movff   deriv0,BARGB0                   ;YES, AARGB0:AARGB2 has result
of Prop + Integ
        movff   deriv1,BARGB1                   ;load derivative term
        movff   deriv2,BARGB2

        movff   pidStat1,tempReg                ;pidStat1 ---> tempReg
        movlw   b'11000000'                             ;prepare for sign check
of bits 7 & 6
        andwf   tempReg,f
```

PIDInt

```
        movf    tempReg,w                               ;check error sign &
a_error sign bits
        sublw   0x00
        btfsc   STATUS,Z
        bra             add_neg_d                               ;bits 7 & 6 (00)
are NEGATIVE, add them
        bra             other_combo_d                   ;bits 7 & 6 not equal to
00
add_neg_d
        call    _24_BitAdd                              ;add negative sign
values
        bra             scale_down                              ;scale result

other_combo_d
        movf    tempReg,w
        sublw   0xC0
        btfsc   STATUS,Z
        bra             add_pos_d                               ;bits 7 & 6 (11)
are POSITIVE, add them
        bra             find_mag_sub_d                  ;bits 7 & 6 (xx) are
different signs , subtract them
add_pos_d
        call    _24_BitAdd                              ;add positive sign
values
        bra             scale_down                              ;scale result
find_mag_sub_d
        call    MagAndSub                               ;subtract unlike sign
numbers
        btfss   pidStat1,mag                    ;which is greater in magnitude ?
        bra             deriv_mag                               ;BARGB is
greater in magnitude
        bra             scale_down                              ;derivative term
< part pid term, leave pid_sign as is

deriv_mag                                                       ;derivative term
> part pid term
        bcf             pidStat1,pid_sign               ;PID result is negative
        btfsc   pidStat1,d_err_sign
        bsf             pidStat1,pid_sign               ;PID result is positive

scale_down
        clrf    BARGB0                                  ;(Prop + Integ + Deriv)
/ 16 = FINAL PID RESULT to plant
        movlw   0x10
        movwf   BARGB1
        call    FXD2416U
        movff   AARGB2,pidOut2                  ;final result ---> pidOut2
        movff   AARGB1,pidOut1                  ;final result ---> pidOut1
        movff   AARGB0,pidOut0                  ;final result ---> pidOut0

#ifdef          pid_100                                 ;Final result needs to
be scaled down to  0 - 100%
        movlw   0x06                                    ;% ratio for propotional
& integral & derivative
        movwf   BARGB0
        movlw   0x40
        movwf   BARGB1

        call    FXD2416U                                ;pidOut0:pidOut2 / %
ratio = 0 - 100% value
        movf    AARGB2,W                                ;AARGB2 --> percent_out
        movwf   percent_out                             ;error has been scaled
down and is now available in a 0 -100% range
#endif

        return                                                  ;return to
mainline code
```

PIDInt

```
;*****************************************************************;
; Function: GetA_Error                                            ;
;                                                                 ;
; PreCondition: Proportional term has been calculated             ;
;                                                                 ;
; Overview: This routine will add the current error with all of the
;                       previous errors. The sign of the accumulated error will
;                       also be determined.     After the accumulated error is
;                       calculated then it is checked if it = 00 or as exceeded
;                       the defined limits.
;
; Input: a_Error0:a_Error1, error0:error1
;
; Output: a_Error0:a_Error1 (updated value)
;
; Side Effects: W register is changed
;
; Stack requirement: 4 levels deep max.
;
;*****************************************************************;
GetA_Error:
        movff   a_Error0,BARGB0                 ;load error & a_error
        movff   a_Error1,BARGB1
        movff   a_Error2,BARGB2
        clrf    AARGB0
        movff   error0,AARGB1
        movff   error1,AARGB2

        call    SpecSign                                ;call routine for
add/sub sign numbers
        btfss   pidStat1,mag                    ;which is greater in magnitude ?
        bra             a_err_zero                              ;bargb, keep
sign as is or both are same sign

        bcf             pidStat1,a_err_sign             ;aargb, make sign same
as error, a_error is negative
        btfsc   pidStat1,err_sign
        bsf             pidStat1,a_err_sign             ;a_error is positive

a_err_zero
        bcf             pidStat1,a_err_z                ;clear a_error zero flag
        movlw   0
        cpfseq  AARGB0                                  ;is byte 0 = 00
        bra             chk_a_err_limit                 ;NO, done checking

        cpfseq  AARGB1                                  ;is byte 1 = 00
        bra             chk_a_err_limit                 ;NO, done checking

        cpfseq  AARGB2                                  ;is byte 2 = 00
        bra             chk_a_err_limit                 ;NO, done checking
        bsf             pidStat1,a_err_z                ;YES, set zero flag

        movff   AARGB0,a_Error0                 ;store the a_error
        movff   AARGB1,a_Error1
        movff   AARGB2,a_Error2
        return                                                  ;a_error = 00,
```

```
return

chk_a_err_limit
        movff   AARGB0,a_Error0                 ;store the a_error
        movff   AARGB1,a_Error1
        movff   AARGB2,a_Error2

        movlw   0                                               ;a_error reached
limits?
        cpfseq  a_Error0                        ;Is a_Error0 > 0 ??, if
yes limit has been exceeded
        bra             restore_limit           ;YES, restore limit
value

        cpfseq  a_Error1                        ;Is a_Error1 = 0 ??, if
yes, limit not exceeded
        bra             chk_a_Error1            ;NO
        return                                          ;YES
chk_a_Error1
        movlw   aErr1Lim
        cpfsgt  a_Error1                        ;Is a_Error1 >
aErr1Lim??
        bra             equal_value                     ;NO, check for
a_Error1 = aErr1Lim ?
        bra             restore_limit           ;YES, restore limit
value
equal_value
        cpfseq  a_Error1                        ;a_Error1 = aErr1Lim?
        return                                          ;no, done
checking a_error
chk_a_Error2
        movlw   aErr2Lim                        ;Yes, a_Error1 =
aErr1Lim
        cpfsgt  a_Error2                        ;Is a_Error2 > aErr2Lim
??
        return                                          ;NO, return to
mainline code

restore_limit
        clrf    a_Error0                        ;YES, a_error limit has
been exceeded
        movlw   aErr1Lim
        movwf   a_Error1
        movlw   aErr2Lim
        movwf   a_Error2
        return                                          ;return to
mainline code


;*****************************************************************;
; Function: GetDeltaError                                        ;
;                                                                 ;
; PreCondition: The derivative routine has been called to calculate the ;
;                               derivative term.
;                                                                 ;
;                                                                 ;
; Overview: This routine subtracts the previous error from the current  ;
;                       error.
;                                                                 ;
;                                                                 ;
; Input: P_Error0:p_Error1, error0:error1
;                                                                 ;
;                                                                 ;
; Output: d_Error0:d_Error1, d_Error sign
;                                                                 ;
;                                                                 ;
; Side Effects: W register is changed
```

```
                                        PIDInt
        ;
;                                                                               ;
; Stack requirement: 3 levels deep max.
                ;
;                                                                               ;
;*******************************************************************************;
GetDeltaError:
        clrf    AARGB0                                  ;load error and p_error
        movff   error0,AARGB1
        movff   error1,AARGB2
        clrf    BARGB0
        movff   p_Error0,BARGB1
        movff   p_Error1,BARGB2

        movf    pidStat1,w                              ;pidStat1 ---> tempReg
        movwf   tempReg                                 ;prepare for sign check
of bits 4 & 2
        movlw   b'00010100'
        andwf   tempReg,f

        movf    tempReg,w                               ;check error sign &
a_error sign bits
        sublw   0x00
        btfsc   STATUS,Z
        bra             p_err_neg                               ;bits 4 & 2 (00)
are NEGATIVE,
        bra             other_combo2                    ;bits 4 & 2 not equal to
00
p_err_neg
        call    MagAndSub
        bcf             pidStat1,d_err_sign             ;d_error is negative
        btfsc   pidStat1,p_err_sign             ;make d_error sign same as
p_error sign
        bsf             pidStat1,d_err_sign             ;d_error is positive
        bra             d_error_zero_chk                ;check if d_error = 0

other_combo2
        movf    tempReg,w
        sublw   0x14
        btfsc   STATUS,Z
        bra             p_err_pos                               ;bits 4 & 2 (11)
are POSITIVE
        bra     p_err_add                               ;bits 4 & 2 (xx) are
different signs

p_err_pos
        call    MagAndSub
        bcf             pidStat1,d_err_sign             ;d_error is negative
        btfsc   pidStat1,p_err_sign             ;make d_error sign same as
p_error sign
        bsf             pidStat1,d_err_sign             ;d_error is positive
        bra             d_error_zero_chk                ;check if d_error = 0
p_err_add
        call    _24_BitAdd                              ;errors are different
sign
        bcf             pidStat1,d_err_sign             ;d_error is negative
        btfsc   pidStat1,err_sign               ;make d_error sign same as error
sign
        bsf             pidStat1,d_err_sign             ;d_error is positive

d_error_zero_chk
        movff   AARGB1,d_Error0
        movff   AARGB2,d_Error1

        movff   error0,p_Error0                 ;load current error into
previous for next deriavtive term
        movff   error1,p_Error1                 ;load current error into
previous for next deriavtive term
```

```
                                        PIDInt
        bcf             pidStat1,p_err_sign             ;make p_error negative
        btfsc   pidStat1,err_sign               ;make p_error the same sign as
error
        bsf             pidStat1,p_err_sign             ;make p_error positive

        bcf             pidStat2,d_err_z                ;clear delta error zero
bit
        movlw   0
        cpfseq  d_Error0                                ;is d_error0 = 00
        return                                                  ;NO, done
checking

        cpfseq  d_Error1                                ;YES, is d_error1 = 00
        return                                                  ;NO, done
checking
        bsf             pidStat2,d_err_z                ;set delta error zero
bit
        return                                                  ;YES, return to
ISR

;*****************************************************************************
; Function: SpecSign                                                         ;
;                                                                            ;
; PreCondition: The sign bits in pidStat1 have been set or cleared
;                                 depending on the variables they represent.
;               ;
;                                                                            ;
; Overview: This routine takes the numbers loaded into the math
;                                 variables (AARGB, BARGB) and determines whether
they    ;
;                                 need to be added or subtracted based on their
sign            ;
;                                 which is located in the pidStat1 register.
;               ;
;                                                       ;
; Input: pidStat1
                                        ;
;                                                                            ;
; Output: add/sub results in math variables (AARGB, BARGB)                   ;
;                                                                            ;
; Side Effects: W register is changed
        ;
;                                                                            ;
; Stack requirement: 2 levels deep max.
                ;
;                                                                           ;
;*****************************************************************************
SpecSign
        movff   pidStat1,tempReg                ;pidStat1 ---> tempReg
        movlw   b'00001100'                             ;prepare for sign check
of bits 3 & 2
        andwf   tempReg,f

        movf    tempReg,w                               ;check error sign &
a_error sign bits
        sublw   0x00
        btfsc   STATUS,Z
        bra             add_neg                                 ;bits 3 & 2 are
NEGATIVE (00), add them
        bra             other_combo                             ;bits 3 & 2 not
equal to 00
add_neg
        call    _24_BitAdd                              ;add negative sign
values
        return
```

```
other_combo
        movf    tempReg,w
        sublw   0x0C
        btfsc   STATUS,Z
        bra             add_pos                                 ;bits 3 & 2 are
POSITIVE (11), add them
        bra             find_mag_sub                    ;bits 3 & 2 are
different signs (xx), subtract them
add_pos
        call    _24_BitAdd                              ;add positive sign
values
        return
find_mag_sub
        call    MagAndSub                               ;subtract unlike sign
numbers
        return

;*************************************************************************;
; Function: MagAndSub                                                    ;
;                                                                        ;
; PreCondition: This routine has been called by SpecSign because the     ;
;                                 numbers being worked on are different in sign.
        ;
;                                                                        ;
; Overview: This routine will detemine which math variable
;
;                       (AARGB or BARGB) is greater in number manitude and then
;
;                       subtract them, the sign of the result will be determined
by      ;
;                       the values in the math variables and their signs.
        ;
;
                                                        ;
; Input: pidStat1
                                        ;
;                                                                        ;
; Output: add/sub results in math variables (AARGB, BARGB)               ;
;                                                                        ;
; Side Effects: W register is changed
        ;
;                                                                        ;
; Stack requirement: 2 levels deep max.
                ;
;                                                                        ;
;*************************************************************************;
MagAndSub:
        movf    BARGB0,w
        subwf   AARGB0,w                                ;AARGB0 - BARGB0 --> W
        btfsc   STATUS,Z                                ;= zero ?
        bra             check_1                                 ;YES
        btfsc   STATUS,C                                ;borrow ?
        bra             aargb_big                               ;AARGB0 >
BARGB0, no borrow
        bra             bargb_big                               ;BARGB0 >
AARGB0, borrow
check_1
        movf    BARGB1,w
        subwf   AARGB1,w                                ;AARGB1 - BARGB1 --> W
        btfsc   STATUS,Z                                ;= zero ?
        bra             check_2                                 ;YES
        btfsc   STATUS,C                                ;borrow ?
        bra             aargb_big                               ;AARGB1 >
BARGB1, no borrow
        bra             bargb_big                               ;BARGB1 >
AARGB1, borrow
```

```
check_2
        movf    BARGB2,w                                ;AARGB2 - BARGB2 --> W
        subwf   AARGB2,w
        btfsc   STATUS,C                                ;borrow ?
        bra             aargb_big                               ;AARGB2 >
BARGB2, no borrow
        bra             bargb_big                               ;BARGB2 >
AARGB2, borrow

aargb_big
        call    _24_bit_sub
        bsf             pidStat1,mag                    ;AARGB is greater in
magnitude
        return

bargb_big
        movff   BARGB0,tempReg                  ;swap AARGB0 with BARGB0
        movff   AARGB0,BARGB0
        movff   tempReg,AARGB0

        movff   BARGB1,tempReg                  ;swap AARGB1 with BARGB1
        movff   AARGB1,BARGB1
        movff   tempReg,AARGB1

        movff   BARGB2,tempReg                  ;swap AARGB2 with BARGB2
        movff   AARGB2,BARGB2
        movff   tempReg,AARGB2

        call    _24_bit_sub                             ;BARGB > AARGB
        bcf             pidStat1,mag                    ;BARGB is greater in
magnitude
        return

;*****************************************************************;
; Function: PidInterrupt
        ;
;                                                                 ;
; PreCondition: This Routine will be called by the application's main   ;
;                               code.
                                        ;
;                                                                 ;
; Overview: When Timer 1 overlfows, an updated value for the Integral   ;
;                       term will be calculated. An updated value for the
derivative;
;                       term will be calculated if derivCount = 0. This routine
;
;                       will check for error = 0, if this is true,then the
routine        ;
;                       will return back to the main line code.
                ;
;
                                                        ;
; Input: pidStat1, a_error
                                ;
;                                                                 ;
; Output: Integral & Derivative terms, Timer1 registers reloaded        ;
;                                                                 ;
; Side Effects: W register is changed
        ;
;                                                                 ;
; Stack requirement: 4 levels deep max.
                ;
;                                                                 ;
;*****************************************************************;
PidInterrupt:
        GLOBAL  PidInterrupt

        btfsc   pidStat1,err_z                          ;Is error = 00 ?
```

```
                                        PIDInt
        return                                                          ;YES,
done.
        call    GetA_Error                                      ;get a_error, is
a_error = 00? reached limits?

derivative_ready?
        decfsz  derivCount,f                            ;is it time to calculate
d_error ?
        bra             skip_deriv                                      ;NO,
finish ISR
        call    GetDeltaError                           ;error - p_error

        movlw   derivCountVal                           ;prepare for next delta
error
        movwf   derivCount                                      ;delta error =
TMR1H:TMR1L * derivCount

skip_deriv
        movlw   timer1Hi                                        ;reload T1
registers with constant time count (user defined)
        movwf   TMR1H
        movlw   timer1Lo
        movwf   TMR1L
        return                                                          ;return
back to the application's ISR

        END                                                     ;directive 'end
of program'
```

```
                                        PIDMath
;*****************************************************
;This file contains the following math routines:

;24-bit addittion
;24-bit subtraction

;16*16  Unsigned Multiply

;24/16  Unsigned Divide

        list            p=18F452
        #include        <p18F452.inc>

        #define _Z              STATUS,2
        #define _C              STATUS,0

        GLOBAL  AARGB0,AARGB1,AARGB2,AARGB3
        GLOBAL  BARGB0,BARGB1,BARGB2,BARGB3
        GLOBAL  ZARGB0,ZARGB1,ZARGB2
        GLOBAL  REMB0,REMB1
        GLOBAL  TEMP,TEMPB0,TEMPB1,TEMPB2,TEMPB3
        GLOBAL  LOOPCOUNT,AEXP,CARGB2


LSB                     equ     0
MSB                     equ     7

math_data       UDATA
AARGB0          RES 1
AARGB1          RES 1
AARGB2          RES 1
AARGB3          RES 1
BARGB0          RES 1
BARGB1          RES 1
BARGB2          RES 1
BARGB3          RES 1
REMB0           RES 1
REMB1           RES 1
REMB2           RES 1
REMB3           RES 1
TEMP            RES 1
TEMPB0          RES 1
TEMPB1          RES 1
TEMPB2          RES 1
TEMPB3          RES 1
ZARGB0          RES 1
ZARGB1          RES 1
ZARGB2          RES 1
CARGB2          RES     1
AEXP            RES 1
LOOPCOUNT       RES 1


math_code       CODE
;-----------------------------------------------------------------------
;               24-BIT ADDITION
_24_BitAdd
        GLOBAL  _24_BitAdd
        movf    BARGB2,w
        addwf   AARGB2,f

        movf    BARGB1,w
        btfsc   _C
        incfsz  BARGB1,w
        addwf   AARGB1,f

        movf    BARGB0,w
        btfsc   _C
```

```
        incfsz  BARGB0,w
        addwf   AARGB0,f
        return

;-----------------------------------------------------------------
;               24-BIT SUBTRACTION
_24_bit_sub
        GLOBAL  _24_bit_sub
        movf    BARGB2,w
        subwf   AARGB2,f

        movf    BARGB1,w
        btfss   STATUS,C
        incfsz  BARGB1,w
        subwf   AARGB1,f

        movf    BARGB0,w
        btfss   STATUS,C
        incfsz  BARGB0,w
        subwf   AARGB0,f
        return

;-----------------------------------------------------------------------
;       16x16 Bit Unsigned Fixed Point Multiply 16 x 16 -> 32

FXM1616U
                GLOBAL  FXM1616U

                MOVFF   AARGB1,TEMPB1

                MOVF    AARGB1,W
                MULWF   BARGB1
                MOVFF   PRODH,AARGB2
                MOVFF   PRODL,AARGB3

                MOVF    AARGB0,W
                MULWF   BARGB0
                MOVFF   PRODH,AARGB0
                MOVFF   PRODL,AARGB1

                MULWF   BARGB1
                MOVF    PRODL,W
                ADDWF   AARGB2,F
                MOVF    PRODH,W
                ADDWFC  AARGB1,F
                CLRF    WREG
                ADDWFC  AARGB0,F

                MOVF    TEMPB1,W
                MULWF   BARGB0
                MOVF    PRODL,W
                ADDWF   AARGB2,F
                MOVF    PRODH,W
                ADDWFC  AARGB1,F
                CLRF    WREG
                ADDWFC  AARGB0,F

                RETLW   0x00

;-----------------------------------------------------------------
FXD2416U
                GLOBAL          FXD2416U
                CLRF            REMB0
                CLRF            REMB1
                CLRF            WREG
                TSTFSZ          BARGB0
                GOTO            D2416BGT1
                MOVFF           BARGB1,BARGB0
```

```
                CALL            FXD2408U
                MOVFF           REMB0,REMB1
                CLRF            REMB0
                RETLW           0x00

D2416BGT1
                CPFSEQ          AARGB0
                GOTO            D2416AGTB
                MOVFF           AARGB1,AARGB0
                MOVFF           AARGB2,AARGB1
                CALL            FXD1616U

                MOVFF           AARGB1,AARGB2
                MOVFF           AARGB0,AARGB1
                CLRF            AARGB0
                RETLW           0x00
D2416AGTB
                MOVFF           AARGB2,AARGB3
                MOVFF           AARGB1,AARGB2
                MOVFF           AARGB0,AARGB1
                CLRF            AARGB0

                MOVFF           AARGB0,TEMPB0
                MOVFF           AARGB1,TEMPB1
                MOVFF           AARGB2,TEMPB2
                MOVFF           AARGB3,TEMPB3

                MOVLW           0x02                    ; set loop count
                MOVWF           AEXP

                MOVLW           0x01
                MOVWF           ZARGB0

                BTFSC           BARGB0,MSB
                GOTO            D2416UNRMOK

                CALL            DGETNRMD                ; get normalization
factor
                MOVWF           ZARGB0

                MULWF           BARGB1
                MOVF            BARGB0,W
                MOVFF           PRODL,BARGB1
                MOVFF           PRODH,BARGB0
                MULWF           ZARGB0
                MOVF            PRODL,W
                ADDWF           BARGB0,F

                MOVF            ZARGB0,W
                MULWF           AARGB3
                MOVFF           PRODL,TEMPB3
                MOVFF           PRODH,TEMPB2
                MULWF           AARGB1
                MOVFF           PRODL,TEMPB1
                MOVFF           PRODH,TEMPB0
                MULWF           AARGB2
                MOVF            PRODL,W
                ADDWF           TEMPB2,F
                MOVF            PRODH,W
                ADDWF           TEMPB1,F

D2416UNRMOK
                BCF             _C
                CLRF            TBLPTRH
                RLCF            BARGB0,W
                RLCF            TBLPTRH,F
                ADDLW           LOW (IBXTBL256+1)       ; access reciprocal
table
```

```
                MOVWF           TBLPTRL
                MOVLW           HIGH (IBXTBL256)
                ADDWFC          TBLPTRH,F
                TBLRD           *-

D2416ULOOP
                MOVFF           TEMPB0,AARGB0
                MOVFF           TEMPB1,AARGB1

                CALL            FXD1608U2               estimate quotient
digit
                BTFSS           AARGB0,LSB
                GOTO            D2416UQTEST

                SETF            AARGB1
                MOVFF           TEMPB1,REMB0
                MOVF            BARGB0,W
                ADDWF           REMB0,F

                BTFSC           _C
                GOTO            D2416UQOK

D2416UQTEST
                MOVF            AARGB1,W                test
                MULWF           BARGB1

                MOVF            PRODL,W
                SUBWF           TEMPB2,W
                MOVF            PRODH,W
                SUBWFB          REMB0,W

                BTFSC           _C
                GOTO            D2416UQOK

                DECF            AARGB1,F

                MOVF            BARGB0,W
                ADDWF           REMB0,F

                BTFSC           _C
                GOTO            D2416UQOK

                MOVF            AARGB1,W
                MULWF           BARGB1

                MOVF            PRODL,W
                SUBWF           TEMPB2,W
                MOVF            PRODH,W
                SUBWFB          REMB0,W

                BTFSS           _C
                DECF            AARGB1,F

D2416UQOK
                MOVFF           AARGB1,ZARGB1

                MOVF            AARGB1,W
                MULWF           BARGB1
                MOVF            PRODL,W
                SUBWF           TEMPB2,F
                MOVF            PRODH,W
                SUBWFB          TEMPB1,F

                MOVF            AARGB1,W
                MULWF           BARGB0
                MOVF            PRODL,W
                SUBWF           TEMPB1,F
```

```
                MOVF            PRODH,W
                SUBWFB          TEMPB0,F

                BTFSS           TEMPB0,MSB              ; test
                GOTO            D2416QOK
                DECF            ZARGB1,F

                MOVF            BARGB1,W
                ADDWF           TEMPB2,F
                MOVF            BARGB0,W
                ADDWFC          TEMPB1,F

D2416QOK
                DCFSNZ          AEXP,F                  ; is loop done?
                GOTO            D2416FIXREM

                MOVFF           ZARGB1,ZARGB2

                MOVFF           TEMPB1,TEMPB0
                MOVFF           TEMPB2,TEMPB1
                MOVFF           TEMPB3,TEMPB2

                GOTO            D2416ULOOP

D2416FIXREM
                MOVFF           TEMPB1,REMB0
                MOVFF           TEMPB2,REMB1

                MOVLW           0x01
                CPFSGT          ZARGB0
                GOTO            D2416REMOK
                RRNCF           ZARGB0,W
                MOVWF           BARGB0
                CALL            DGETNRMD

                MULWF           TEMPB2
                MOVFF           PRODH,REMB1
                MULWF           TEMPB1
                MOVF            PRODL,W
                ADDWF           REMB1,F
                MOVFF           PRODH,REMB0

D2416REMOK
                CLRF            AARGB0
                MOVFF           ZARGB1,AARGB2
                MOVFF           ZARGB2,AARGB1

                RETLW           0x00

;-----------------------------------------------------
FXD2408U
                MOVFF           AARGB0,TEMPB0
                MOVFF           AARGB1,TEMPB1
                MOVFF           AARGB2,TEMPB2

                CALL            FXD1608U

                MOVFF           AARGB0,TEMPB0
                MOVFF           AARGB1,TEMPB1

                MOVFF           TEMPB2,AARGB1
                MOVFF           REMB0,AARGB0

                CALL            FXD1608U

                MOVFF           AARGB1,AARGB2
                MOVFF           TEMPB1,AARGB1
                MOVFF           TEMPB0,AARGB0
```

```
                RETLW           0x00

;-----------------------------------------------------
FXD1608U
                GLOBAL          FXD1608U

                MOVLW           0x01
                CPFSGT          BARGB0
                GOTO            DREMZERO8

FXD1608U1
                GLOBAL          FXD1608U1

                BCF             _C
                CLRF            TBLPTRH
                RLCF            BARGB0,W
                RLCF            TBLPTRH,F
                ADDLW           LOW (IBXTBL256+1)       ; access reciprocal
table
                MOVWF           TBLPTRL
                MOVLW           HIGH (IBXTBL256)
                ADDWFC          TBLPTRH,F
                TBLRD           *-

FXD1608U2
                GLOBAL          FXD1608U2

                MOVFF           AARGB0,REMB1
                MOVFF           AARGB1,REMB0

                MOVF            TABLAT,W                ; estimate quotient
                MULWF           REMB1
                MOVFF           PRODH,AARGB0
                MOVFF           PRODL,AARGB1

                TBLRD           *+
                MOVF            TABLAT,W
                MULWF           REMB0
                MOVFF           PRODH,AARGB2

                MULWF           REMB1
                MOVF            PRODL,W
                ADDWF           AARGB2,F
                MOVF            PRODH,W
                ADDWFC          AARGB1,F
                CLRF            WREG
                ADDWFC          AARGB0,F

                TBLRD           *-
                MOVF            TABLAT,W
                MULWF           REMB0
                MOVF            PRODL,W
                ADDWF           AARGB2,F
                MOVF            PRODH,W
                ADDWFC          AARGB1,F
                CLRF            WREG
                ADDWFC          AARGB0,F

                MOVF            BARGB0,W
                MULWF           AARGB1
                MOVFF           PRODL,AARGB3
                MOVFF           PRODH,AARGB2
                MULWF           AARGB0
                MOVF            PRODL,W
                ADDWF           AARGB2,F

                MOVF            AARGB3,W                ; estimate remainder
```

```
                SUBWF           REMB0,F
                MOVF            AARGB2,W
                SUBWFB          REMB1,F

                BTFSS           REMB1,MSB               ; test remainder
                RETLW           0x00

                DECF            AARGB1,F
                CLRF            WREG
                SUBWFB          AARGB0,F

                MOVF            BARGB0,W
                ADDWF           REMB0,F

        RETLW       0x00

;-----------------------------------------------------------
FXD1616U
                TSTFSZ          BARGB0
                GOTO            D1616B0GT0
                MOVFF           BARGB1,BARGB0
                CALL            FXD1608U
                MOVFF           REMB0,REMB1
                CLRF            REMB0

                RETLW           0x00

D1616B0GT0
                MOVF            BARGB0,W
                SUBWF           AARGB0,W
                BTFSS           _C
                GOTO            D1616QZERO
                BTFSS           _Z
                GOTO            D1616AGEB

                MOVF            BARGB1,W
                SUBWF           AARGB1,W
                BTFSS           _C
                GOTO            D1616QZERO

D1616AGEB
                MOVFF           AARGB0,TEMPB0
                MOVFF           AARGB1,TEMPB1

                MOVFF           AARGB1,CARGB2
                MOVFF           AARGB0,AARGB1
                CLRF            AARGB0

                MOVFF           BARGB0,BARGB2
                MOVFF           BARGB1,BARGB3

                BTFSC           BARGB0,MSB
                GOTO            D1616UNRMOK

                MOVF            BARGB0,W
                RLNCF           WREG,F
                ADDLW           LOW (IBXTBL256+3)       ; access reciprocal
table
                MOVWF           TBLPTRL
                MOVLW           HIGH (IBXTBL256)
                CLRF            TBLPTRH
                ADDWFC          TBLPTRH,F
                TBLRD           *

                MOVF            TABLAT,W                ; normalize
                MULWF           BARGB3
                MOVFF           PRODL,BARGB1
```

```
                MOVFF           PRODH,BARGB0
                MULWF           BARGB2
                MOVF            PRODL,W
                ADDWF           BARGB0,F

                MOVF            TABLAT,W
                MULWF           TEMPB1
                MOVFF           PRODL,CARGB2
                MOVFF           PRODH,AARGB1
                MULWF           TEMPB0
                MOVF            PRODL,W
                ADDWF           AARGB1,F
                CLRF            AARGB0
                MOVF            PRODH,W
                ADDWFC          AARGB0,F

D1616UNRMOK
                CALL            FXD1608U1               ; estimate quotient
digit

                MOVF            AARGB1,W
                MULWF           BARGB1

                MOVF            PRODL,W
                SUBWF           CARGB2,W
                MOVF            PRODH,W
                SUBWFB          REMB0,W

                BTFSS           _C                      ; test
                DECF            AARGB1,F

D1616UQOK
                MOVF            AARGB1,W                ; calculate remainder
                MULWF           BARGB3
                MOVF            PRODL,W
                SUBWF           TEMPB1,F
                MOVF            PRODH,W
                SUBWFB          TEMPB0,F

                MOVF            AARGB1,W
                MULWF           BARGB2
                MOVF            PRODL,W
                SUBWF           TEMPB0,F

;       This test does not appear to be necessary in the 16 bit case, but
;       is included here in the event that a case appears after testing.

;               BTFSS           TEMPB0,MSB              ; test
;               GOTO            D1616QOK
;               DECF            AARGB1

;               MOVF            BARGB3,W
;               ADDWF           TEMPB1
;               MOVF            BARGB2,W
;               ADDWFC          TEMPB0

D1616QOK
                MOVFF           TEMPB0,REMB0
                MOVFF           TEMPB1,REMB1

                RETLW       0x00
;-----------------------------------------------------------
DGETNRMD
                MOVLW           0x10
                CPFSLT          BARGB0
                GOTO            DGETNRMDH
DGETNRMDL
                BTFSC           BARGB0,3
```

```
                RETLW           0x10
                BTFSC           BARGB0,2
                RETLW           0x20
                BTFSC           BARGB0,1
                RETLW           0x40
                BTFSC           BARGB0,0
                RETLW           0x80
DGETNRMDH
                BTFSC           BARGB0,6
                RETLW           0x02
                BTFSC           BARGB0,5
                RETLW           0x04
                BTFSC           BARGB0,4
                RETLW           0x08

;-------------------------------------------------------------------------
---------------
;       Routines for the trivial cases when the quotient is zero.
;       Timing: 9,7,5   clks
;   PM: 9,7,5               DM: 8,6,4

;D3232QZERO
;               MOVFF           AARGB3,REMB3
;               CLRF            AARGB3

;D2424QZERO
;               MOVFF           AARGB2,REMB2
;               CLRF            AARGB2

D1616QZERO
                MOVFF           AARGB1,REMB1
                CLRF            AARGB1
                MOVFF           AARGB0,REMB0
                CLRF            AARGB0
                RETLW           0x00

DREMZERO8
                CLRF            REMB0
                RETLW           0x00

;-------------------------------------------------------------------------
---------------
;       The table IBXTBL256 is used by all routines and consists of 16-bit
;       upper bound approximations to the reciprocal of BARGB0.

IBXTBL256
                GLOBAL  IBXTBL256

                DATA    0x0000
                DATA    0x0001
                DATA    0x8001
                DATA    0x5556
                DATA    0x4001
                DATA    0x3334
                DATA    0x2AAB
                DATA    0x2493
                DATA    0x2001
                DATA    0x1C72
                DATA    0x199A
                DATA    0x1746
                DATA    0x1556
                DATA    0x13B2
                DATA    0x124A
                DATA    0x1112
                DATA    0x1001
                DATA    0x0F10
                DATA    0x0E39
                DATA    0x0D7A
```

```
DATA    0x0CCD
DATA    0x0C31
DATA    0x0BA3
DATA    0x0B22
DATA    0x0AAB
DATA    0x0A3E
DATA    0x09D9
DATA    0x097C
DATA    0x0925
DATA    0x08D4
DATA    0x0889
DATA    0x0843
DATA    0x0801
DATA    0x07C2
DATA    0x0788
DATA    0x0751
DATA    0x071D
DATA    0x06EC
DATA    0x06BD
DATA    0x0691
DATA    0x0667
DATA    0x063F
DATA    0x0619
DATA    0x05F5
DATA    0x05D2
DATA    0x05B1
DATA    0x0591
DATA    0x0573
DATA    0x0556
DATA    0x053A
DATA    0x051F
DATA    0x0506
DATA    0x04ED
DATA    0x04D5
DATA    0x04BE
DATA    0x04A8
DATA    0x0493
DATA    0x047E
DATA    0x046A
DATA    0x0457
DATA    0x0445
DATA    0x0433
DATA    0x0422
DATA    0x0411
DATA    0x0401
DATA    0x03F1
DATA    0x03E1
DATA    0x03D3
DATA    0x03C4
DATA    0x03B6
DATA    0x03A9
DATA    0x039C
DATA    0x038F
DATA    0x0382
DATA    0x0376
DATA    0x036A
DATA    0x035F
DATA    0x0354
DATA    0x0349
DATA    0x033E
DATA    0x0334
DATA    0x032A
DATA    0x0320
DATA    0x0316
DATA    0x030D
DATA    0x0304
DATA    0x02FB
DATA    0x02F2
```

```
DATA    0x02E9
DATA    0x02E1
DATA    0x02D9
DATA    0x02D1
DATA    0x02C9
DATA    0x02C1
DATA    0x02BA
DATA    0x02B2
DATA    0x02AB
DATA    0x02A4
DATA    0x029D
DATA    0x0296
DATA    0x0290
DATA    0x0289
DATA    0x0283
DATA    0x027D
DATA    0x0277
DATA    0x0271
DATA    0x026B
DATA    0x0265
DATA    0x025F
DATA    0x025A
DATA    0x0254
DATA    0x024F
DATA    0x024A
DATA    0x0244
DATA    0x023F
DATA    0x023A
DATA    0x0235
DATA    0x0231
DATA    0x022C
DATA    0x0227
DATA    0x0223
DATA    0x021E
DATA    0x021A
DATA    0x0215
DATA    0x0211
DATA    0x020D
DATA    0x0209
DATA    0x0205
DATA    0x0201
DATA    0x01FD
DATA    0x01F9
DATA    0x01F5
DATA    0x01F1
DATA    0x01ED
DATA    0x01EA
DATA    0x01E6
DATA    0x01E2
DATA    0x01DF
DATA    0x01DB
DATA    0x01D8
DATA    0x01D5
DATA    0x01D1
DATA    0x01CE
DATA    0x01CB
DATA    0x01C8
DATA    0x01C4
DATA    0x01C1
DATA    0x01BE
DATA    0x01BB
DATA    0x01B8
DATA    0x01B5
DATA    0x01B3
DATA    0x01B0
DATA    0x01AD
DATA    0x01AA
DATA    0x01A7
```

```
DATA    0x01A5
DATA    0x01A2
DATA    0x019F
DATA    0x019D
DATA    0x019A
DATA    0x0198
DATA    0x0195
DATA    0x0193
DATA    0x0190
DATA    0x018E
DATA    0x018B
DATA    0x0189
DATA    0x0187
DATA    0x0184
DATA    0x0182
DATA    0x0180
DATA    0x017E
DATA    0x017B
DATA    0x0179
DATA    0x0177
DATA    0x0175
DATA    0x0173
DATA    0x0171
DATA    0x016F
DATA    0x016D
DATA    0x016B
DATA    0x0169
DATA    0x0167
DATA    0x0165
DATA    0x0163
DATA    0x0161
DATA    0x015F
DATA    0x015D
DATA    0x015B
DATA    0x0159
DATA    0x0158
DATA    0x0156
DATA    0x0154
DATA    0x0152
DATA    0x0151
DATA    0x014F
DATA    0x014D
DATA    0x014B
DATA    0x014A
DATA    0x0148
DATA    0x0147
DATA    0x0145
DATA    0x0143
DATA    0x0142
DATA    0x0140
DATA    0x013F
DATA    0x013D
DATA    0x013C
DATA    0x013A
DATA    0x0139
DATA    0x0137
DATA    0x0136
DATA    0x0134
DATA    0x0133
DATA    0x0131
DATA    0x0130
DATA    0x012F
DATA    0x012D
DATA    0x012C
DATA    0x012A
DATA    0x0129
DATA    0x0128
DATA    0x0126
```

PIDMath

```
DATA    0x0125
DATA    0x0124
DATA    0x0122
DATA    0x0121
DATA    0x0120
DATA    0x011F
DATA    0x011D
DATA    0x011C
DATA    0x011B
DATA    0x011A
DATA    0x0119
DATA    0x0117
DATA    0x0116
DATA    0x0115
DATA    0x0114
DATA    0x0113
DATA    0x0112
DATA    0x0110
DATA    0x010F
DATA    0x010E
DATA    0x010D
DATA    0x010C
DATA    0x010B
DATA    0x010A
DATA    0x0109
DATA    0x0108
DATA    0x0107
DATA    0x0106
DATA    0x0105
DATA    0x0104
DATA    0x0103
DATA    0x0102
DATA    0x0101

end
```

CAPITULO V

PROPUESTA DE IMPLEMENTACION DEL SISTEMA

En este capítulo se aborda la propuesta de implementación del sistema de control, tanto del hardware y del software que involucra a la planta (baño de cobre por electrólisis y al controlador PID.

5.1 Implementación del Hardware-

. Una tarjeta interfaz de potencia (Amplificador de potencia)

. Una tarjeta de interfaz para el sensor, (esquema en fig.2.7)

. Una tarjeta de interfaz del actuador final (Filtro Paso Bajo)

. Una computadora Pentium.

. Una tarjeta con el Microcontrolador.

. En cuanto al control de potencia se pueden citar:

. Conmutación electrónica , mediante puente de transistores T o H.

. Modulación por ancho de pulsos (PWM). A mayor ancho de pulso se consigue una mayor tensión de salida, alcanzando un máximo de +Vcc.

Como se puede apreciar, en este informe se propone el uso del método PWM, por su alta fiabilidad y por existir en la actualidad transistores MOSFET de alta velocidad de conmutación, así como el soporte de altos niveles de corriente .

En la fig. 5.1 se muestra un tren de pulsos de anchura variable y de período constante (señales PWM).

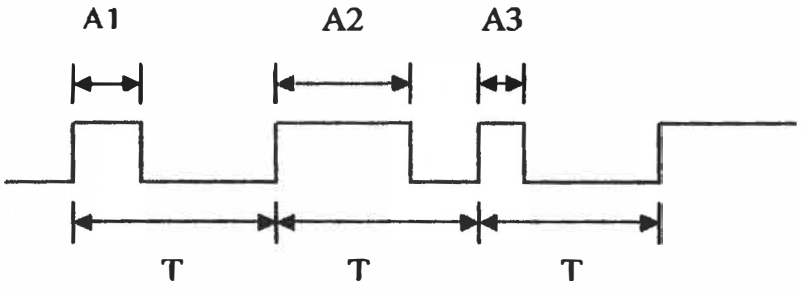


Fig.5.1 Señal PWM

Tarjeta interfaz de potencia-

Esta tarjeta está conformada por el amplificador de potencia.

Tarjeta sensora-

El puerto A del microcontrolador captura los datos correspondientes a la tensión generada por la corriente de electrólisis a través de una resistencia en serie.

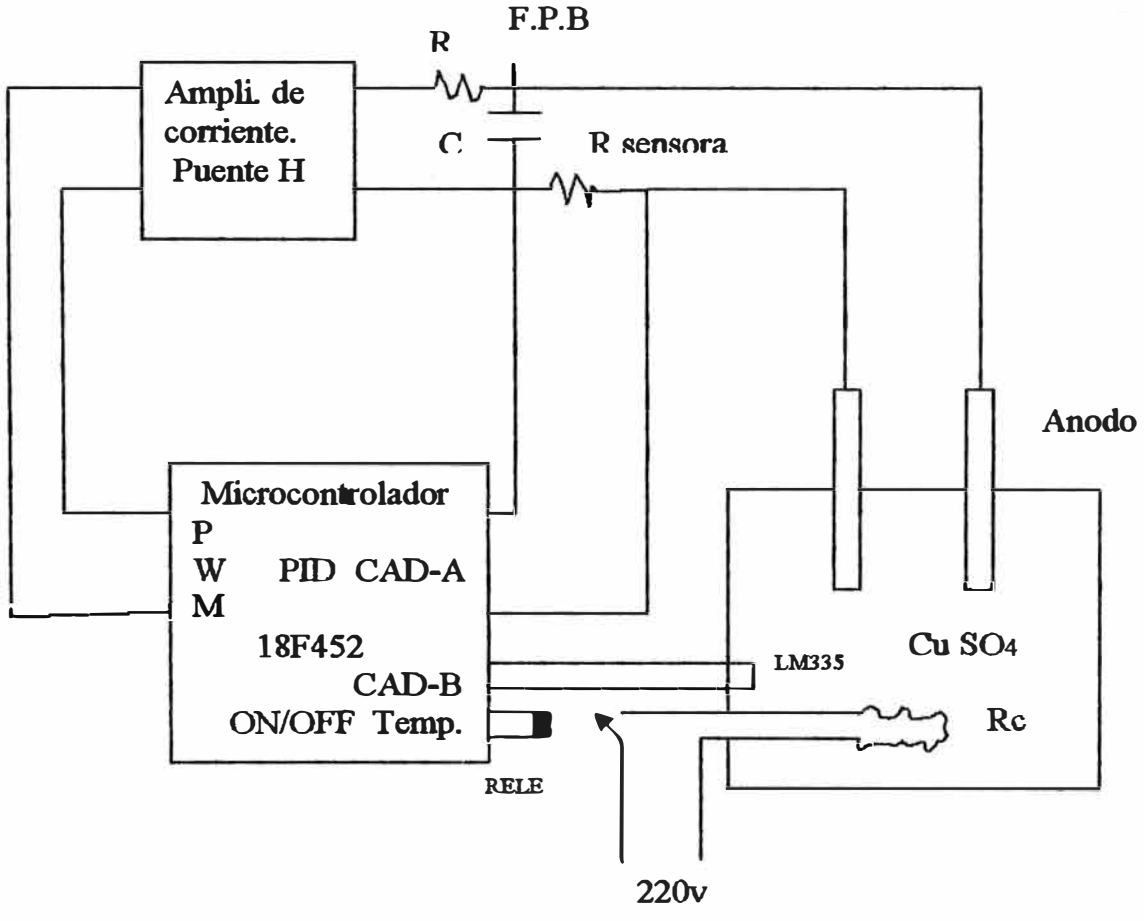


Fig.5.2 Circuito básico del Hardware

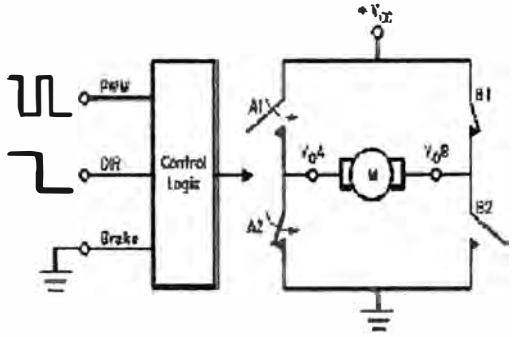


Fig.5.3 Esquema básico puente H

5.2 El software del sistema de control

Como ya lo habíamos mencionado en la descripción del sistema del capitulo II, estamos aprovechando el software proporcionado por microchip para la ejecución del proceso de control PID.

Para efectos de simulación, estamos utilizando el programa matemático MATLAB versión 6.5.

5.3 Costo aproximado

Si consideramos que la computadora puede ser incluso prestada, el costo sería producido por la confección de las tarjetas acondicionadoras que no son muy complicadas, El Microcontrolador de usarse el 16F877 es de un costo aproximado de 50 soles. El costo del sensor LM335, circuitos impresos , operacionales, la resistencia térmica, el ácido,el recipiente la fuente de alimentación y los electrodos nos dan un estimado de 1500 soles por el total del proyecto.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

1.- Es factible la construcción de este módulo de control de corriente del baño de cobre.
2.- Para encontrar los parámetros L y T verdaderos de la F.T recomendada para procesos químicos es necesario efectuar el proceso electrolítico.
3.- En este caso se ha usado un microcontrolador, pero así trabajando en assembler todos estos programas duran menos de 200 μseg. en respuesta como está explicado en la pagina 12 del AN937.
4.- Con la implementación del algoritmo PID en forma discreta se consiguen resultados de respuesta rápida y precisa.

RECOMENDACIONES

1.- Usar un solo puerto del microcontrolador para el control de la corriente del proceso, con un comparador ventana sería suficiente para la temperatura.
2.- En este caso estamos usando el μControlador 18F452 compatible en sofware con el 16F877 que es mas acequible; pero tenemos el software para el 18F452.
3.- Este trabajo se podría implementar en un tema de tesis con un desarrollo completo del proceso.

ANEXO A

Descripción del PIC 16F877

1. PIC 16f877
2. Características
3. Diagrama de bloques
4. Descripción de pines
5. Aplicación

PIC 16F877

Se denomina microcontrolador a un dispositivo programable capaz de realizar diferentes actividades que requieran del procesamiento de datos digitales y del control y comunicación digital de diferentes dispositivos.

Los microcontroladores poseen una memoria interna que almacena dos tipos de datos; las instrucciones, que corresponden al programa que se ejecuta, y los registros, es decir, los datos que el usuario maneja, así como registros especiales para el control de las diferentes funciones del microcontrolador.

Los microcontroladores se programan en Assembler y cada microcontrolador varía su conjunto de instrucciones de acuerdo a su fabricante y modelo. De acuerdo al número de instrucciones que el microcontrolador maneja se le denomina de arquitectura RISC (reducido) o CISC (complejo).

Los microcontroladores poseen principalmente una ALU (Unidad Lógico Aritmética), memoria del programa, memoria de registros, y pines I/O (entrada y/0 salida). La ALU es la encargada de procesar los datos dependiendo de las instrucciones que se ejecuten (ADD, OR, AND), mientras que los pines son los que se encargan de comunicar al microcontrolador con el medio externo; la función de los pines puede ser de transmisión de datos, alimentación de corriente para el funcionamiento de este o pines de control especifico.

En este proyecto se utilizó el PIC 16F877. Este microcontrolador es fabricado por MicroChip familia a la cual se le denomina PIC. El modelo 16F877 posee varias características que hacen a este microcontrolador un dispositivo muy versátil, eficiente y práctico para ser empleado en la aplicación que posteriormente será detallada.

Algunas de estas características se muestran a continuación:

Soporta modo de comunicación serial, posee dos pines para ello.

Amplia memoria para datos y programa.

Memoria reprogramable: La memoria en este PIC es la que se denomina FLASH; este tipo de memoria se puede borrar electrónicamente (esto corresponde a la "F" en el modelo).

Set de instrucciones reducido (tipo RISC), pero con las instrucciones necesarias para facilitar su manejo.

CARACTERISTICAS

En la siguiente tabla de pueden observar las características más relevantes del dispositivo:

CARACTERÍSTICAS	16F877
Frecuencia máxima	DX-20MHz
Memoria de programa flash palabra de 14 bits	8KB
Posiciones RAM de datos	368
Posiciones EEPROM de datos	256
Puertos E/S	A,B,C,D,E
Número de pines	40
Interrupciones	14
Timers	3
Módulos CCP	2
Comunicaciones Serie	MSSP, USART
Comunicaciones paralelo	PSP
Líneas de entrada de CAD de 10 bits	8
Juego de instrucciones	35 Instrucciones
Longitud de la instrucción	14 bits
Arquitectura	Harvard
CPU	Risc
Canales Pwm	2
Pila Harware	-
Ejecución En 1 Ciclo Máquina	-

Descripción de los puertos:

Puerto A:

- **Puerto de e/s de 6 pines**
- **RA0 è RA0 y AN0**
- **RA1 è RA1 y AN1**
- **RA2 è RA2, AN2 y Vref-**

- RA3 è RA3, AN3 y Vref+
- RA4 è RA4 (Salida en colector abierto) y T0CKI(Entrada de reloj del módulo Timer0)
- RA5 è RA5, AN4 y SS (Selección esclavo para el puerto serie síncrono)

Puerto B:

- Puerto e/s 8 pines
- Resistencias pull-up programables
- RB0 è Interrupción externa
- RB4-7 è Interrupción por cambio de flanco
- RB5-RB7 y RB3 è programación y debugger en circuito

Puerto C:

- Puerto e/s de 8 pines
- RC0 è RC0, T1OSO (Timer1 salida oscilador) y T1CKI (Entrada de reloj del modulo Timer1).
- RC1-RC2 è PWM/COMP/CAPT
- RC1 è T1OSI (entrada osc timer1)
- RC3-4 è IIC
- RC3-5 è SPI
- RC6-7 è USART

Puerto D:

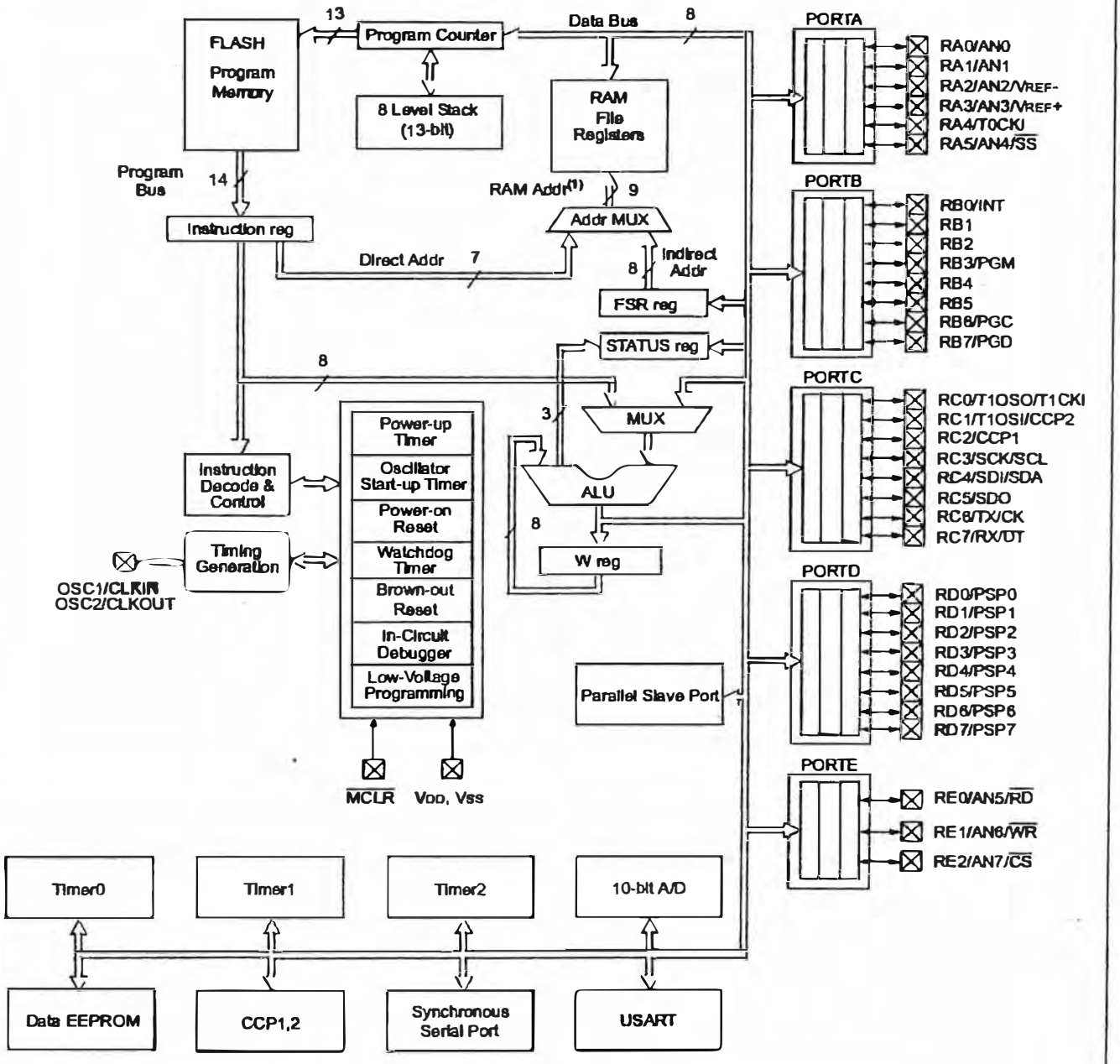
- Puerto e/s de 8 pines
- Bus de datos en PPS (Puerto paralelo esclavo)
- Puerto E:
- Puerto de e/s de 3 pines
- RE0 è RE0 y AN5 y Read de PPS
- RE1 è RE1 y AN6 y Write de PPS
- RE2 è RE2 y AN7 y CS de PPS

Dispositivos periféricos:

- Timer0: Temporizador-contador de 8 bits con preescaler de 8 bits
- Timer1: Temporizador-contador de 16 bits con preescaler que puede incrementarse en modo sleep de forma externa por un cristal/clock.
- Timer2: Temporizador-contador de 8 bits con preescaler y postescaler.
- Dos módulos de Captura, Comparación, PWM (Modulación de Anchura de pulsos).
- Conversor A/D de 10 bits.
- Puerto Serie Síncrono Master (MSSP) con SPI e I^2C (Master/Slave).
- USART/SCI (Universal Synchronous Asynchronous Receiver Transmitter) con 9 bit.
- Puerta Paralela Esclava (PSP) sólo en encapsulados con 40 pines

FIGURE 1-2: PIC16F874 AND PIC16F877 BLOCK DIAGRAM

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874	4K	192 Bytes	128 Bytes
PIC16F877	8K	368 Bytes	256 Bytes



Note 1: Higher order bits are from the STATUS register.

PIC16F87X

28/40-Pin 8-Bit CMOS FLASH Microcontrollers

Devices Included in this Data Sheet:

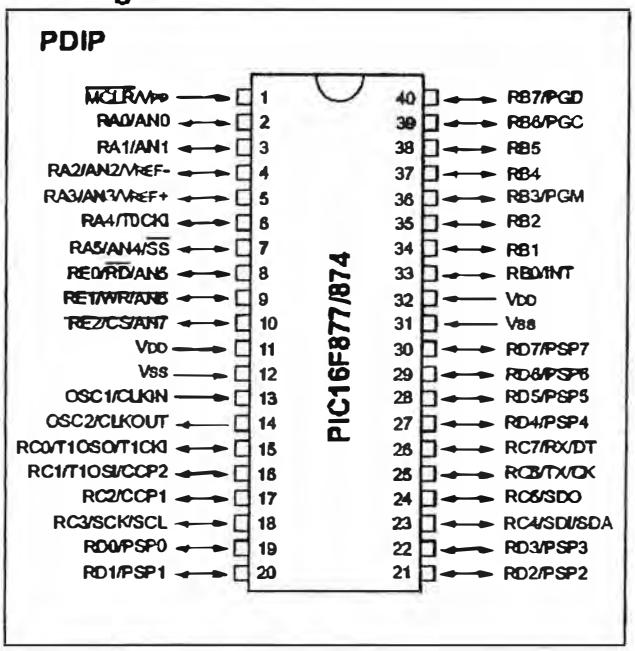
- PIC16F873
- PIC16F874
- PIC16F876
- PIC16F877

Microcontroller Core Features:

- High performance RISC CPU
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches which are two cycle
- Operating speed: DC - 20 MHz clock input
  DC - 200 ns instruction cycle
- Up to 8K x 14 words of FLASH Program Memory,
  Up to 368 x 8 bytes of Data Memory (RAM)
  Up to 256 x 8 bytes of EEPROM Data Memory
- Pinout compatible to the PIC16C73B/74B/76/77
- Interrupt capability (up to 14 sources)
- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- Power-on Reset (POR)
- Power-up Timer (PWRT) and
  Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection
- Power saving SLEEP mode
- Selectable oscillator options
- Low power, high speed CMOS FLASH/EEPROM technology
- Fully static design
- In-Circuit Serial Programming™ (ICSP) via two pins
- Single 5V In-Circuit Serial Programming capability
- In-Circuit Debugging via two pins
- Processor read/write access to program memory
- Wide operating voltage range: 2.0V to 5.5V
- High Sink/Source Current: 25 mA
- Commercial, Industrial and Extended temperature ranges
- Low-power consumption:
  - < 0.6 mA typical @ 3V, 4 MHz
  - 20 µA typical @ 3V, 32 kHz
  - < 1 µA typical standby current

Pin Diagram

PDIP



Peripheral Features:

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during SLEEP via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- 10-bit multi-channel Analog-to-Digital converter
- Synchronous Serial Port (SSP) with SPI™ (Master mode) and I^2C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) 8-bits wide, with external RD, WR and CS controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

DESCRIPCIÓN DE PINES

NOMBRE DEL PIN	PIN	TIPO	TIPO DE BUFFER	DESCRIPCIÓN
OSC1/CLKIN	13	I	ST/MOS	Entrada del oscilador de cristal / Entrada de señal de reloj externa
OSC2/CLKOUT	14	O	-	Salida del oscilador de cristal
MCLR/Vpp/THV	1	I/P	ST	Entrada del Master clear (Reset) o entrada de voltaje de programación o modo de control high voltaje test
				PORTA es un puerto I/O bidireccional
RA0/AN0	2	I/O	TTL	RAO: puede ser salida analógica 0
RA1/AN1	3	I/O	TTL	RA1: puede ser salida analógica 1
RA2/AN2/ Vref-	4	I/O	TTL	RA2: puede ser salida analógica 2 o referencia negativa de voltaje
RA3/AN3/Vref+	5	I/O	TTL	RA3: puede ser salida analógica 3 o referencia positiva de voltaje
RA4/T0CKI	6	I/O	ST	RA4: puede ser entrada de reloj el timer0.
RA5/SS/AN4	7	I/O	TTL	RA5: puede ser salida analógica 4 o el esclavo seleccionado por el puerto serial síncrono.
				PORTB es un puerto I/O bidireccional. Puede ser programado todo como entradas
RBO/INT	33	I/O	TTL/ST	RB0 pude ser pin de interrupción externo.
RB1	34	I/O	TTL	
RB2	35	I/O	TTL	RB3: puede ser la entada de programación de bajo voltaje
RB3/PGM	36	I/O	TTL	Pin de interrupción Pin de interrupción

RB4 **RB5** **RB6/PGC** **RB7/PGD**	37 38 39 40	I/O I/O I/O I/O	TTL TTL TTL/ST TTL/ST	Pin de interrupción. Reloj de programación serial
				PORTC es un puerto I/O bidireccional
RCO/T1OSO/T1CKI	15	I/O	ST	RCO puede ser la salida del oscilador timer1 o la entrada de reloj del timer1
RC1/T1OS1/CCP2	16	I/O	ST	RC1 puede ser la entrada del oscilador timer1 o salida PMW 2
RC2/CCP1	17	I/O	ST	RC2 puede ser una entrada de captura y comparación o salida PWN
RC3/SCK/SCL	18	I/O	ST	RC3 puede ser la entrada o salida serial de reloj síncrono para modos SPI e I2C
RC4/SD1/SDA	23	I/O	ST	RC4 puede ser la entrada de datos SPI y modo I2C
RC5/SD0				RC5 puede ser la salida de datos SPI
RC6/Tx/CK	24	I/O	ST	RC6 puede ser el transmisor asíncrono
	25	I/O	ST	USART o el reloj síncrono.
RC7/RX/DT	26	I/O	ST	RC7 puede ser el receptor asíncrono USART o datos síncronos
				PORTD es un puerto bidireccional paralelo
RD0/PSP0	19	I/O	ST/TTL	
RD1/PSP1	20	I/O I/O	ST/TTL	
RD2/PSP2	21	I/O I/O	ST/TTL	
RD3/PSP3	22	I/O I/O	ST/TTL	
RD4/PSP4	27	I/O	ST/TTL	
RD5/PSP5	28		ST/TTL	
RD6/PSP6	29		ST/TTL	
RD7/PSP7	30		ST/TTL	

REO/RD/AN5 **RE1/WR/AN** **RE2/CS/AN7**	8 9 10	I/O I/O I/O	ST/TTL ST/TTL ST/TTL	PORTE es un puerto I/O bidireccional REO: puede ser control de lectura para el puerto esclavo paralelo o entrada analógica 5 RE1: puede ser escritura de control para el puerto paralelo esclavo o entrada analógica 6 RE2: puede ser el selector de control para el puerto paralelo esclavo o la entrada analógica 7.
Vss	12.31	P	-	Referencia de tierra para los pines lógicos y de I/O
Vdd	11.32	P	-	Fuente positiva para los pines lógicos y de I/O
NC	-	-	-	No está conectado internamente

APLICACIÓN

El proyecto presentado tendrá como objetivo principal, diseñar un controlador de temperatura usando un microcontrolador.

Se parte del hecho de que para realizar el control, hay que sensar la variable de proceso en primer lugar, posteriormente se debe enviar las señales e instrucciones respectivas al elemento de control (microcontrolador) para que este actúe en consecuencias realizando la acción de control.

Se tiene como elemento principal un microcontrolador PIC16F877, el cual recibirá a través de pulsadores, el valor de Setpoint que desee el usuario.

Se utilizará una pantalla de LCD, donde se visualizarán los valores de Setpoint. El manejo de dicha pantalla se realizará a través de los puertos de salida del microcontrolador.

PLANTEAMIENTO DEL PROBLEMA

En los procesos industriales es necesario tener un registro y control eficiente sobre todas las variables que intervienen en el proceso, con el fin de conocer el comportamiento de la misma durante cada una de las fases del proceso, de manera tal que con esta información se realizarán las acciones necesarias para un control seguro y eficiente. Basándonos en esto se desea diseñar un controlador de temperatura.

FACTIBILIDAD

Puede decirse que el presente proyecto es factible puesto que todos los dispositivos que intervienen él, están disponibles en el mercado al igual que la información referente a su funcionamiento y los costos de los mismos son accesibles. También podemos mencionar entre otras razones que se cuenta con los equipos y accesoria técnica para la programación del PIC (dispositivo principal) así como también para la manipulación de los de más dispositivos que intervienen en el proyecto. Se ha realizados proyectos similares anteriormente obteniéndose buenos resultados.

ANEXO B

National Semiconductor

November 2000

LM135/LM235/LM335, LM135A/LM235A/LM335A Precision Temperature Sensors

General Description

The LM135 series are precision, easily-celibrated, integrated circuit temperature sensors. Operating as a 2-terminal zener, the LM135 has a breakdown voltage directly proportional to absolute temperature at +10 mV/°K. With less than 1Ω dynamic impedance the device operates over a current range of 400 µA to 5 mA with virtually no change in performance. When calibrated at 25°C the LM135 has typically less than 1°C error over a 100°C temperature range. Unlike other sensors the LM135 has a linear output.

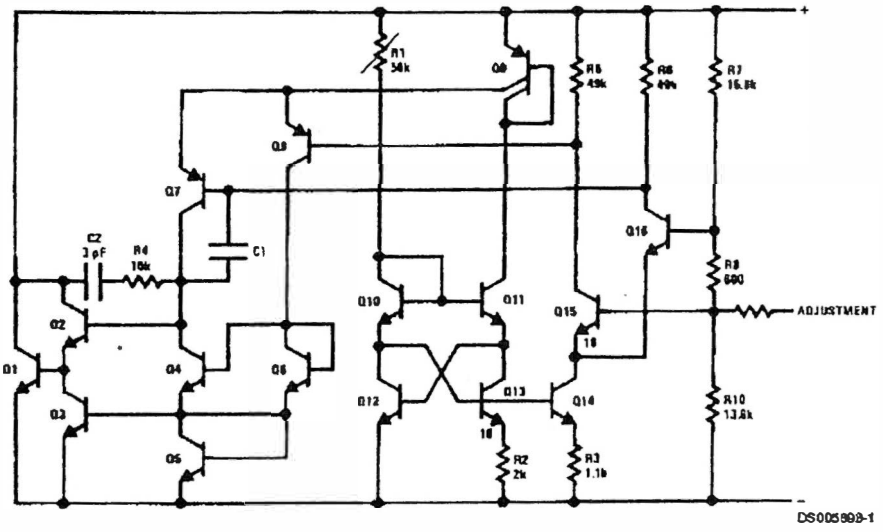
Applications for the LM135 include almost any type of temperature sensing over a −55°C to +150°C temperature range. The low impedance and linear output make interfacing to readout or control circuitry especially easy.

The LM135 operates over a −55°C to +150°C temperature range while the LM235 operates over a −40°C to +125°C temperature range. The LM335 operates from −40°C to +100°C. The LM135/LM235/LM335 are available packaged in hermetic TO-46 transistor packages while the LM335 is also available in plastic TO-92 packages.

Features

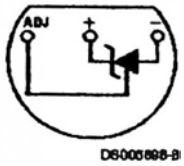
- Directly calibrated in °Kelvin
- 1°C initial accuracy available
- Operates from 400 µA to 5 mA
- Less than 1Ω dynamic impedance
- Easily calibrated
- Wide operating temperature range
- 200°C overrange
- Low cost

Schematic Diagram



Connection Diagrams

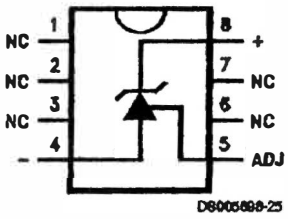
TO-92
Plastic Package



D6005698-8

Bottom View
Order Number LM335Z
or LM335AZ
See NS Package
Number Z03A

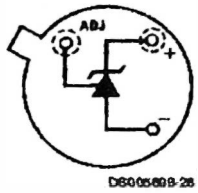
SO-8
Surface Mount Package



D6005698-25

Order Number LM335M
See NS Package
Number M08A

TO-46
Metal Can Package*



D6005698-26

*Case is connected to negative pin

Bottom View
Order Number LM135H,
LM135H-MIL, LM235H,
LM335H, LM135AH,
LM235AH or LM335AH
See NS Package
Number H03H

Absolute Maximum Ratings (Note 4)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Reverse Current	15 mA
Forward Current	10 mA
Storage Temperature	
TO-46 Package	−60°C to +180°C
TO-92 Package	−60°C to +150°C
SO-8 Package	−65°C to +150°C

Specified Operating Temp. Range

	Continuous	Intermittent (Note 2)
LM135, LM135A	−55°C to +150°C	150°C to 200°C
LM235, LM235A	−40°C to +125°C	125°C to 150°C
LM335, LM335A	−40°C to +100°C	100°C to 125°C

Lead Temp. (Soldering, 10 seconds)	
TO-92 Package:	260°C
TO-46 Package:	300°C
SO-8 Package:	300°C
Vapor Phase (60 seconds):	215°C
Infrared (15 seconds):	220°C

Temperature Accuracy (Note 1)

LM135/LM235, LM135A/LM235A

Parameter	Conditions	LM135A/LM235A			LM135/LM235			Units
		Min	Typ	Max	Min	Typ	Max	
Operating Output Voltage	T_C = 25°C, I_R = 1 mA	2.97	2.98	2.99	2.95	2.98	3.01	V
Uncalibrated Temperature Error	T_C = 25°C, I_R = 1 mA		0.5	1		1	3	°C
Uncalibrated Temperature Error	$T_{MIN} \le T_C \le T_{MAX}$, I_R = 1 mA		1.3	2.7		2	5	°C
Temperature Error with 25°C Calibration	$T_{MIN} \le T_C \le T_{MAX}$, I_R = 1 mA		0.3	1		0.5	1.5	°C
Calibrated Error at Extended Temperatures	$T_C = T_{MAX}$ (Intermittent)		2			2		°C
Non-Linearity	I_R = 1 mA		0.3	0.5		0.3	1	°C

Temperature Accuracy (Note 1)

LM335, LM335A

Parameter	Conditions	LM335A			LM335			Units
		Min	Typ	Max	Min	Typ	Max	
Operating Output Voltage	T_C = 25°C, I_R = 1 mA	2.95	2.98	3.01	2.92	2.98	3.04	V
Uncalibrated Temperature Error	T_C = 25°C, I_R = 1 mA		1	3		2	6	°C
Uncalibrated Temperature Error	$T_{MIN} \le T_C \le T_{MAX}$, I_R = 1 mA		2	5		4	9	°C
Temperature Error with 25°C Calibration	$T_{MIN} \le T_C \le T_{MAX}$, I_R = 1 mA		0.5	1		1	2	°C
Calibrated Error at Extended Temperatures	$T_C = T_{MAX}$ (Intermittent)		2			2		°C
Non-Linearity	I_R = 1 mA		0.3	1.5		0.3	1.5	°C

Electrical Characteristics (Note 1)

Parameter	Conditions	LM135/LM235 LM135A/LM235A			LM335 LM335A			Units
		Min	Typ	Max	Min	Typ	Max	
Operating Output Voltage Change with Current	400 µA≤I_R≤5 mA At Constant Temperature		2.5	10		3	14	mV
Dynamic Impedance	I_R=1 mA		0.5			0.6		Ω
Output Voltage Temperature Coefficient			+10			+10		mV/°C
Time Constant	Still Air		80			80		sec
	100 ft/Min Air		10			10		sec
	Stirred Oil		1			1		sec
Time Stability	T_C=125°C		0.2			0.2		°C/khr

Electrical Characteristics (Note 1) (Continued)

Note 1: Accuracy measurements are made in a well-stirred oil bath. For other conditions, self heating must be considered.

Note 2: Continuous operation at these temperatures for 10,000 hours for H package and 5,000 hours for Z package may decrease life expectancy of the device.

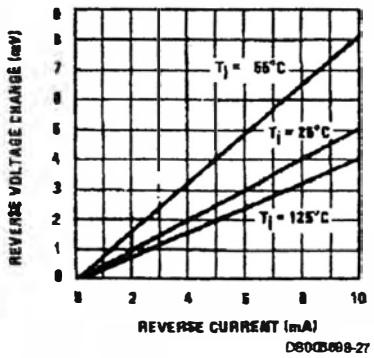
Note 3:

Thermal Resistance	TO-92	TO-46	SO-8
θ_{JA} (junction to ambient)	202°C/W	400°C/W	165°C/W
θ_{JC} (junction to case)	170°C/W	N/A	N/A

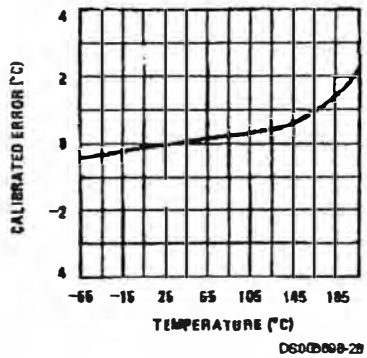
Note 4: Refer to RETS135H for military specifications.

Typical Performance Characteristics

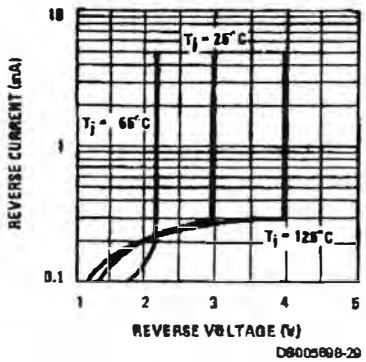
Reverse Voltage Change



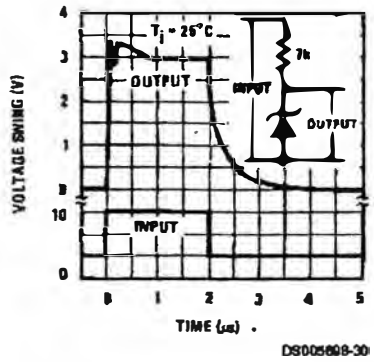
Calibrated Error



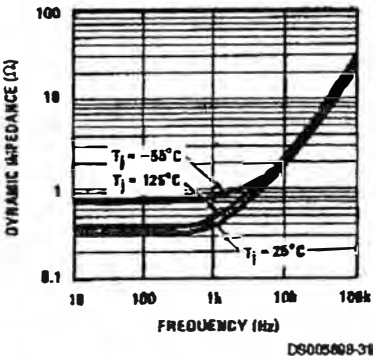
Reverse Characteristics



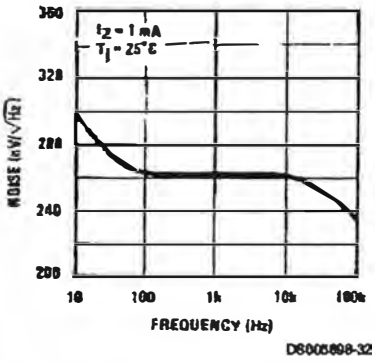
Response Time



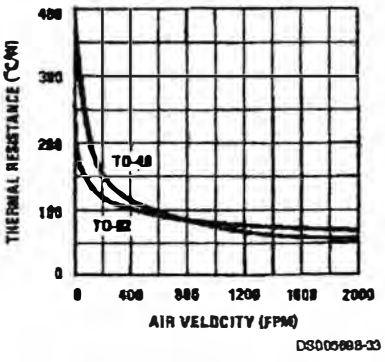
Dynamic Impedance



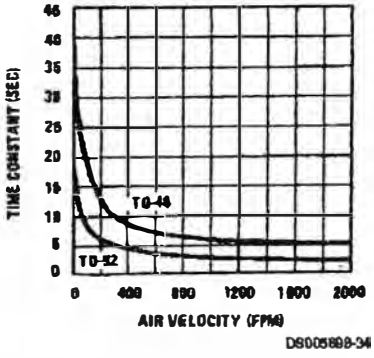
Noise Voltage



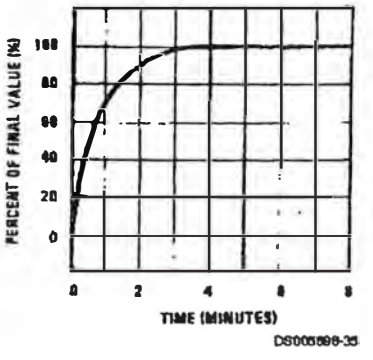
Thermal Resistance Junction to Air



Thermal Time Constant

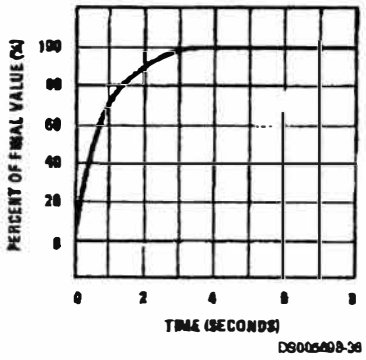


Thermal Response in Still Air

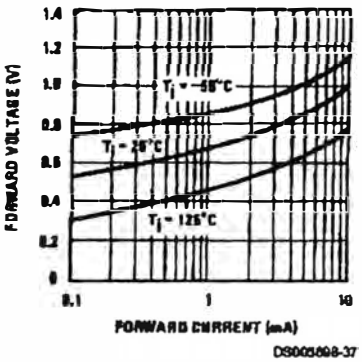


Typical Performance Characteristics (Continued)

Thermal Response in Stirred Oil Bath



Forward Characteristics



Application Hints

CALIBRATING THE LM135

Included on the LM135 chip is an easy method of calibrating the device for higher accuracies. A pot connected across the LM135 with the arm tied to the adjustment terminal allows a 1-point calibration of the sensor that corrects for inaccuracy over the full temperature range.

This single point calibration works because the output of the LM135 is proportional to absolute temperature with the extrapolated output of sensor going to 0V output at 0°K (−273.15°C). Errors in output voltage versus temperature are only slope (or scale factor) errors so a slope calibration at one temperature corrects at all temperatures.

The output of the device (calibrated or uncalibrated) can be expressed as:

$$V_{OUT_T} = V_{OUT_{T_0}} \times \frac{T}{T_0}$$

where T is the unknown temperature and T_0 is a reference temperature, both expressed in degrees Kelvin. By calibrating the output to read correctly at one temperature the output at all temperatures is correct. Nominally the output is calibrated at 10 mV/°K.

To insure good sensing accuracy several precautions must be taken. Like any temperature sensing device, self heating can reduce accuracy. The LM135 should be operated at the lowest current suitable for the application. Sufficient current, of course, must be available to drive both the sensor and the calibration pot at the maximum operating temperature as well as any external loads.

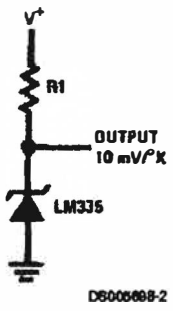
If the sensor is used in an ambient where the thermal resistance is constant, self heating errors can be calibrated out. This is possible if the device is run with a temperature stable current. Heating will then be proportional to zener voltage and therefore temperature. This makes the self heating error proportional to absolute temperature the same as scale factor errors.

WATERPROOFING SENSORS

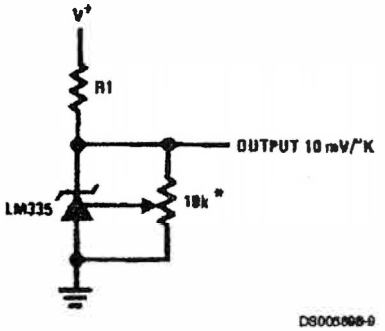
Meltable inner core heat shrinkable tubing such as manufactured by Raychem can be used to make low-cost waterproof sensors. The LM335 is inserted into the tubing about ½" from the end and the tubing heated above the melting point of the core. The unfilled ½" end melts and provides a seal over the device.

Typical Applications

Basic Temperature Sensor

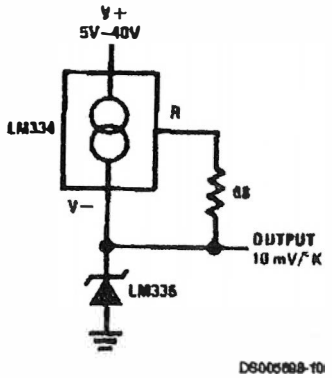


Calibrated Sensor



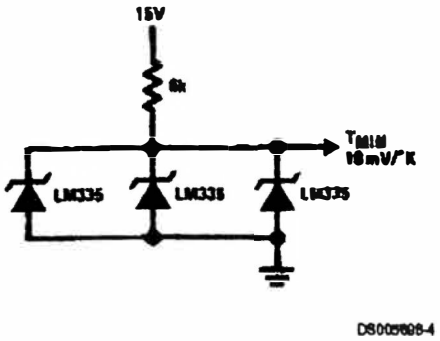
*Calibrate for 2.982V at 25°C

Wide Operating Supply



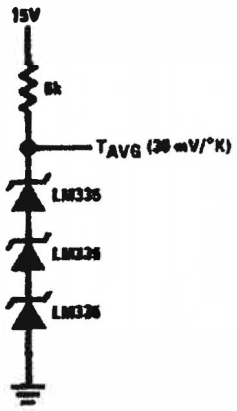
Typical Applications (Continued)

Minimum Temperature Sensing



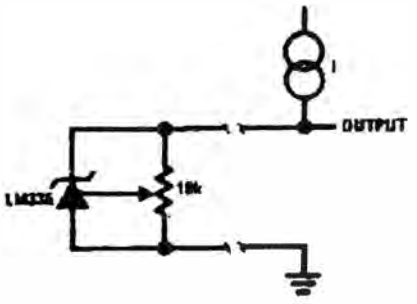
DS005698-4

Average Temperature Sensing



DS005698-18

Remote Temperature Sensing



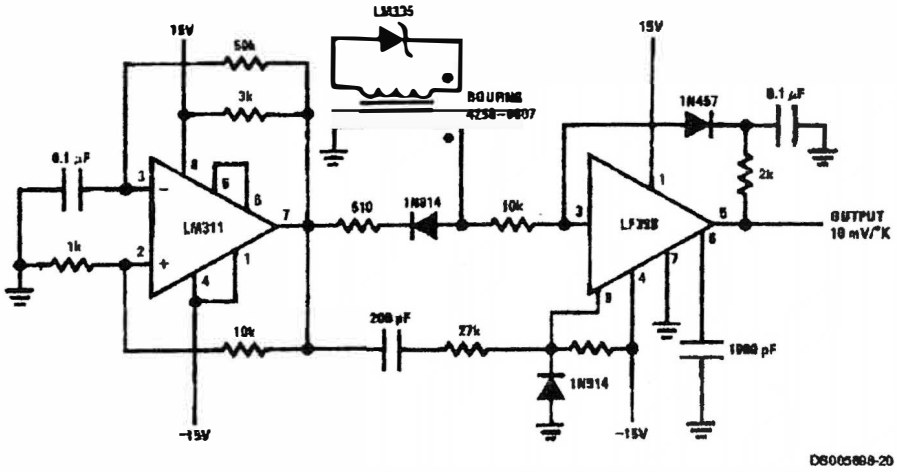
DS005698-19

Wire length for 1°C error due to wire drop

	I_R = 1 mA	I_R = 0.5 mA*
AWG	FEET	FEET
14	4000	8000
16	2500	5000
18	1600	3200
20	1000	2000
22	625	1250
24	400	800

*For I_R = 0.5 mA, the trim pot must be deleted.

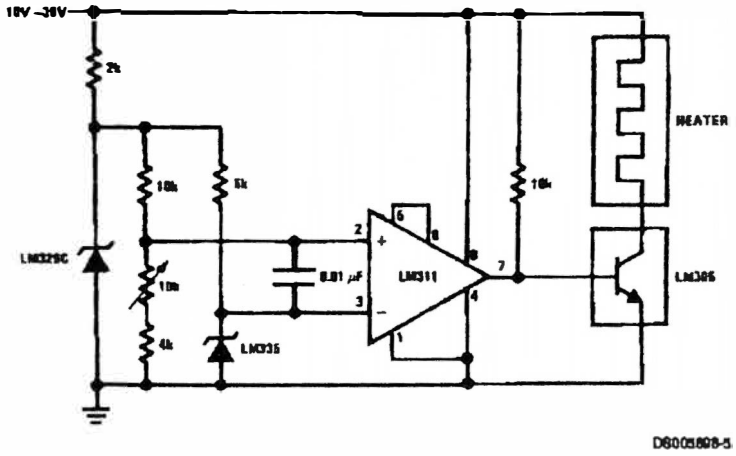
Isolated Temperature Sensor



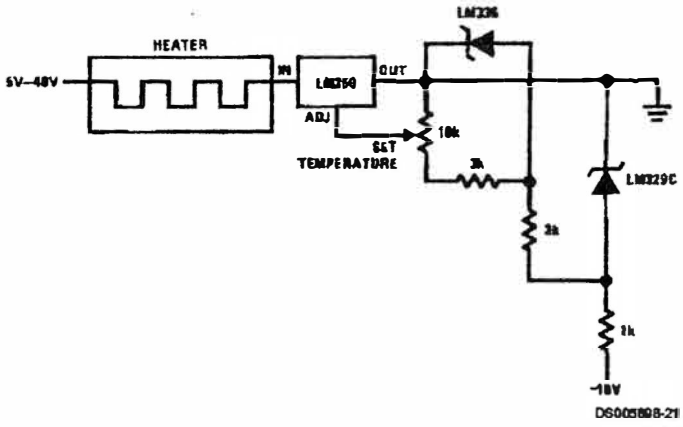
DS005698-20

Typical Applications (Continued)

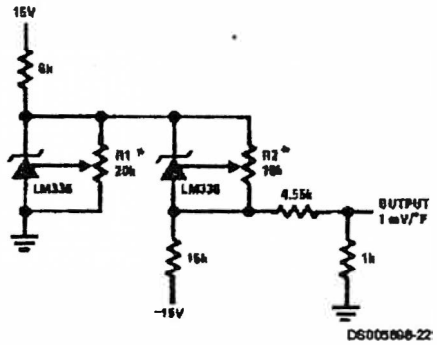
Simple Temperature Controller



Simple Temperature Control

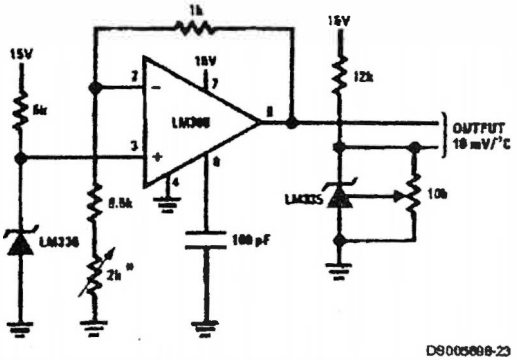


Ground Referred Fahrenheit Thermometer



*Adjust R2 for 2.554V across LM336.
Adjust R1 for correct output.

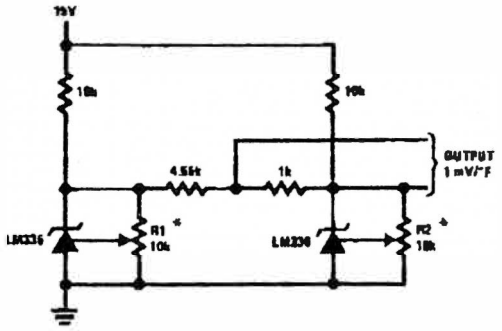
Centigrade Thermometer



*Adjust for 2.7315V at output of LM308

Typical Applications (Continued)

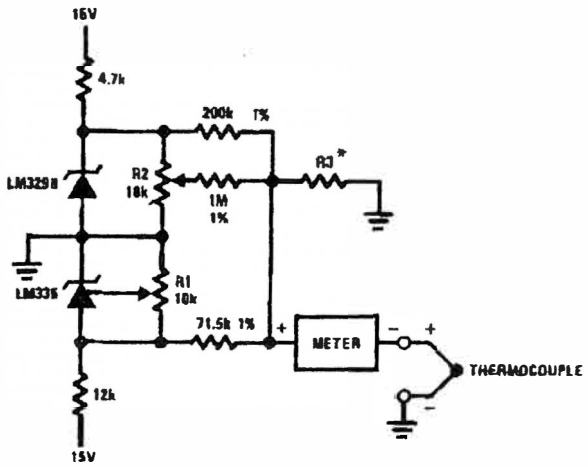
Fahrenheit Thermometer



D6005698-24

*To calibrate adjust R2 for 2.554V across LM336.
Adjust R1 for correct output.

THERMOCOUPLE COLD JUNCTION COMPENSATION
Compensation for Grounded Thermocouple



D6005698-8

*Select R3 for proper thermocouple type

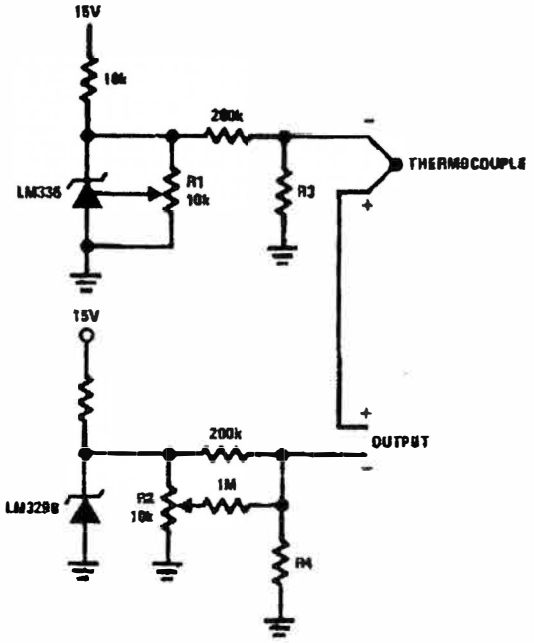
THERMO-COUPLE	R3 (±1%)	SEEBECK COEFFICIENT
J	377Ω	52.3 µV/°C
T	308Ω	42.8 µV/°C
K	293Ω	40.8 µV/°C
S	45.8Ω	6.4 µV/°C

Adjustments: Compensates for both sensor and resistor tolerances
1. Short LM329B
2. Adjust R1 for Seebeck Coefficient times ambient temperature (in degrees K) across R3.
3. Short LM335 and adjust R2 for voltage across R3 corresponding to thermocouple type

J	14.32 mV	K	11.17 mV
T	11.79 mV	S	1.768 mV

Typical Applications (Continued)

Single Power Supply Cold Junction Compensation



DS009598-11

*Select R3 and R4 for thermocouple type

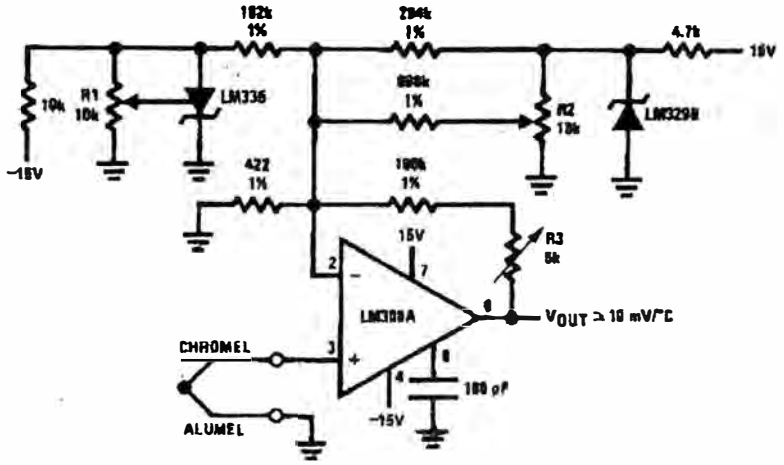
THERMO-COUPLE	R3	R4	SEEBECK COEFFICIENT
J	1.05K	385Ω	52.3 μV/°C
T	856Ω	315Ω	42.8 μV/°C
K	816Ω	300Ω	40.8 μV/°C
S	128Ω	46.3Ω	6.4 μV/°C

Adjustments:
1. Adjust R1 for the voltage across R3 equal to the Seebeck Coefficient times ambient temperature in degrees Kelvin.
2. Adjust R2 for voltage across R4 corresponding to thermocouple

J	14.32 mV
T	11.79 mV
K	11.17 mV
S	1.768 mV

Typical Applications (Continued)

Centigrade Calibrated Thermocouple Thermometer



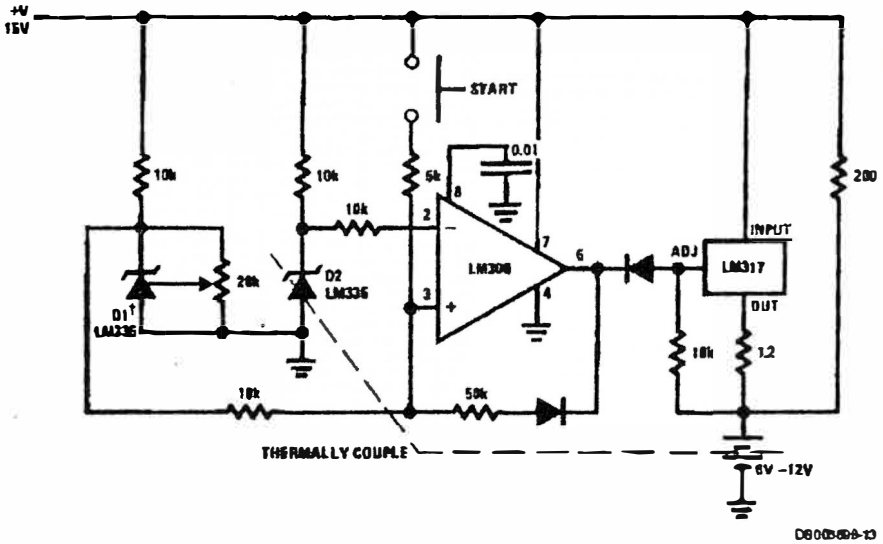
DS005698-12

Terminate thermocouple reference junction in close proximity to LM335.

Adjustments:

1. Apply signal in place of thermocouple and adjust R3 for a gain of 245.7.
2. Short non-inverting input of LM308A and output of LM329B to ground.
3. Adjust R1 so that V_{OUT} = 2.982V @ 25°C.
4. Remove short across LM329B and adjust R2 so that V_{OUT} = 246 mV @ 25°C.
5. Remove short across thermocouple.

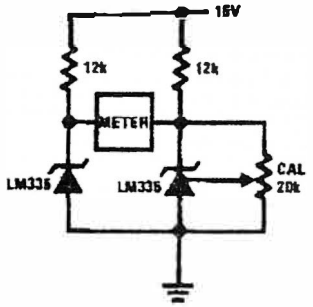
Fast Charger for Nickel-Cadmium Batteries



DS005698-13

1Adjust D1 to 50 mV greater V_Z than D2.
Charge terminates on 5°C temperature rise. Couple D2 to battery.

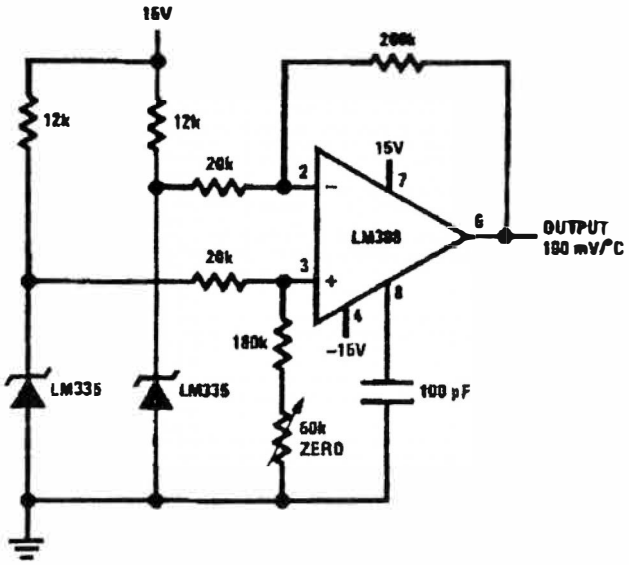
Differential Temperature Sensor



DS005698-7

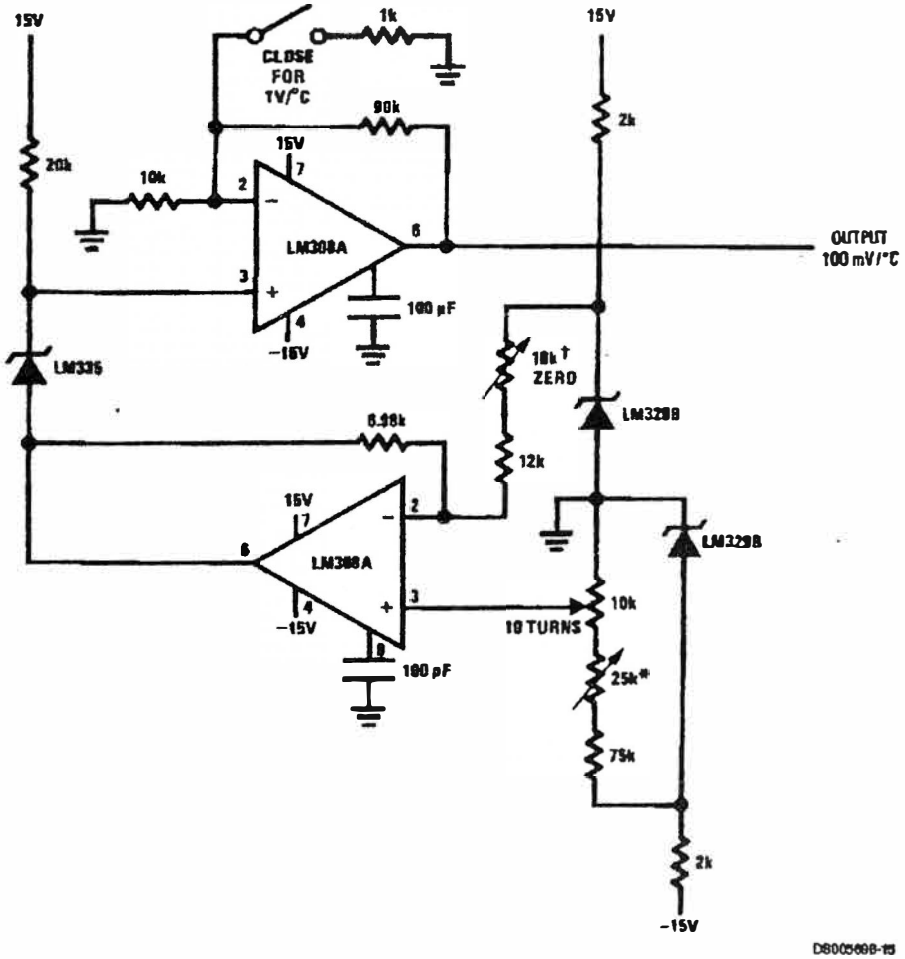
Typical Applications (Continued)

Differential Temperature Sensor



DS005698-14

Variable Offset Thermometer

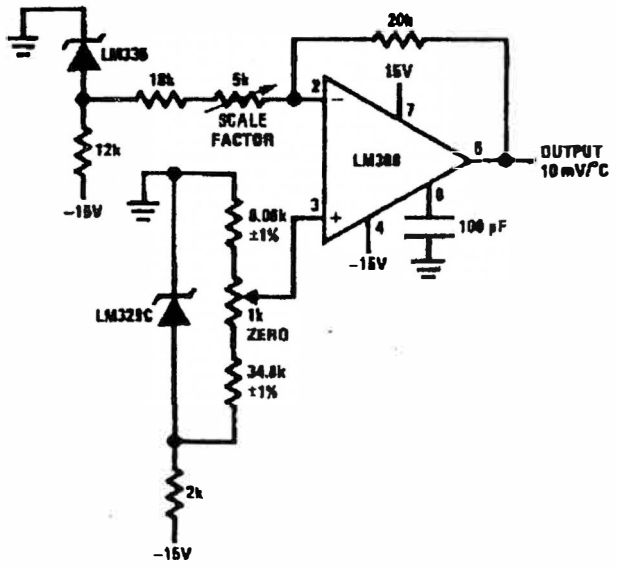


DS005698-15

†Adjust for zero with sensor at 0°C and 10T pot set at 0°C
*Adjust for zero output with 10T pot set at 100°C and sensor at 100°C
Output reads difference between temperature and dial setting of 10T pot

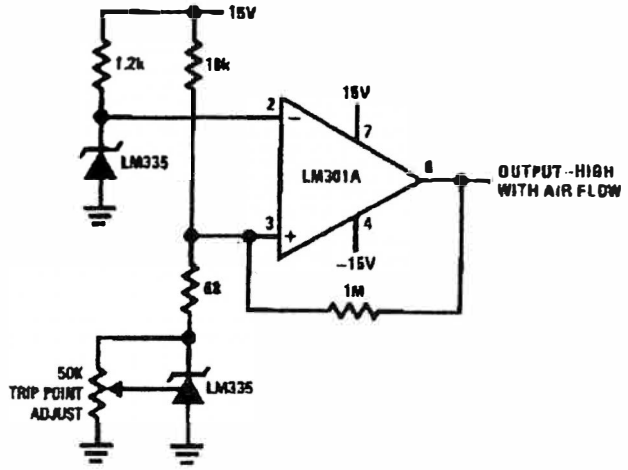
Typical Applications (Continued)

Ground Referred Centigrade Thermometer



DS005698-16

Air Flow Detector*



DS005698-17

*Self heating is used to detect air flow

Definition of Terms

Operating Output Voltage: The voltage appearing across the positive and negative terminals of the device at specified conditions of operating temperature and current.

Uncalibrated Temperature Error: The error between the operating output voltage at 10 mV/°K and case temperature at specified conditions of current and case temperature.

Calibrated Temperature Error: The error between operating output voltage and case temperature at 10 mV/°K over a temperature range at a specified operating current with the 25°C error adjusted to zero.

ANEXO C

MICROCHIP

PIC18FXX2

28/40-pin High Performance, Enhanced FLASH Microcontrollers with 10-Bit A/D

High Performance RISC CPU:

- C compiler optimized architecture/instruction set
  - Source code compatible with the PIC16 and PIC17 instruction sets
- Linear program memory addressing to 32 Kbytes
- Linear data memory addressing to 1.5 Kbytes

Device	On-Chip Program Memory		On-Chip RAM (bytes)	Data EEPROM (bytes)
	FLASH (bytes)	# Single Word Instructions		
PIC18F242	16K	8192	768	256
PIC18F252	32K	16384	1536	256
PIC18F442	16K	8192	768	256
PIC18F452	32K	16384	1536	256

- Up to 10 MIPs operation:
  - DC - 40 MHz osc./clock input
  - 4 MHz - 10 MHz osc./clock input with PLL active
- 16-bit wide instructions, 8-bit wide data path
- Priority levels for interrupts
- 8 x 8 Single Cycle Hardware Multiplier

Peripheral Features:

- High current sink/source 25 mA/25 mA
- Three external interrupt pins
- Timer0 module: 8-bit/16-bit timer/counter with 8-bit programmable prescaler
- Timer1 module: 16-bit timer/counter
- Timer2 module: 8-bit timer/counter with 8-bit period register (time-base for PWM)
- Timer3 module: 16-bit timer/counter
- Secondary oscillator clock option - Timer1/Timer3
- Two Capture/Compare/PWM (CCP) modules. CCP pins that can be configured as:
  - Capture input: capture is 16-bit, max. resolution 6.25 ns (TCY/16)
  - Compare is 16-bit, max. resolution 100 ns (TCY)
  - PWM output: PWM resolution is 1- to 10-bit, max. PWM freq. @: 8-bit resolution = 156 kHz, 10-bit resolution = 39 kHz
- Master Synchronous Serial Port (MSSP) module, Two modes of operation:
  - 3-wire SPI™ (supports all 4 SPI modes)
  - I^2C™ Master and Slave mode

Peripheral Features (Continued):

- Addressable USART module:
  - Supports RS-485 and RS-232
- Parallel Slave Port (PSP) module

Analog Features:

- Compatible 10-bit Analog-to-Digital Converter module (A/D) with:
  - Fast sampling rate
  - Conversion available during SLEEP
  - Linearity ≤ 1 LSb
- Programmable Low Voltage Detection (PLVD)
  - Supports interrupt on-Low Voltage Detection
- Programmable Brown-out Reset (BOR)

Special Microcontroller Features:

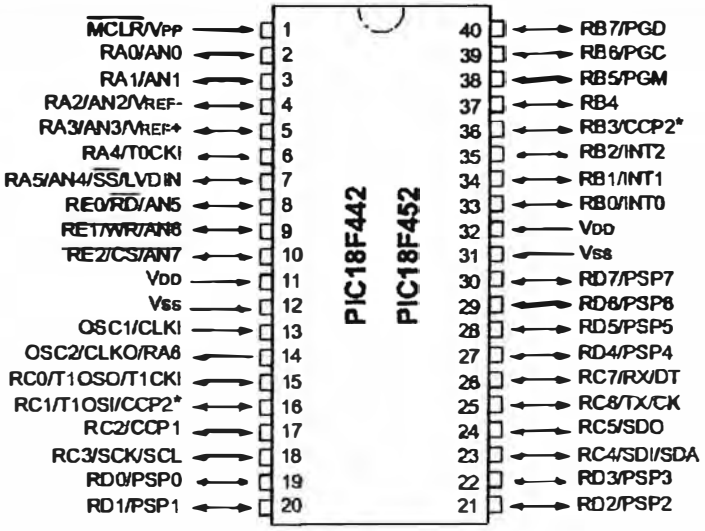
- 100,000 erase/write cycle Enhanced FLASH program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory
- FLASH/Data EEPROM Retention: > 40 years
- Self-reprogrammable under software control
- Power-on Reset (POR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own On-Chip RC Oscillator for reliable operation
- Programmable code protection
- Power saving SLEEP mode
- Selectable oscillator options including:
  - 4X Phase Lock Loop (of primary oscillator)
  - Secondary Oscillator (32 kHz) clock input
- Single supply 5V In-Circuit Serial Programming™ (ICSP™) via two pins
- In-Circuit Debug (ICD) via two pins

CMOS Technology:

- Low power, high speed FLASH/EEPROM technology
- Fully static design
- Wide operating voltage range (2.0V to 5.5V)
- Industrial and Extended temperature ranges
- Low power consumption:
  - < 1.6 mA typical @ 5V, 4 MHz
  - 25 µA typical @ 3V, 32 kHz
  - < 0.2 µA typical standby current

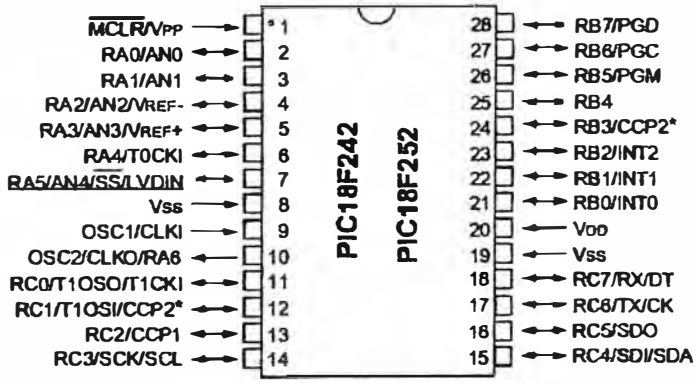
Pin Diagrams (Cont.'d)

DIP



Note: Pin compatible with 40-pin PIC16C7X devices.

DIP, SOIC



* RB3 is the alternate pin for the CCP2 pin multiplexing.

1.0 DEVICE OVERVIEW

This document contains device specific information for the following devices:

- PIC18F242
- PIC18F442
- PIC18F252
- PIC18F452

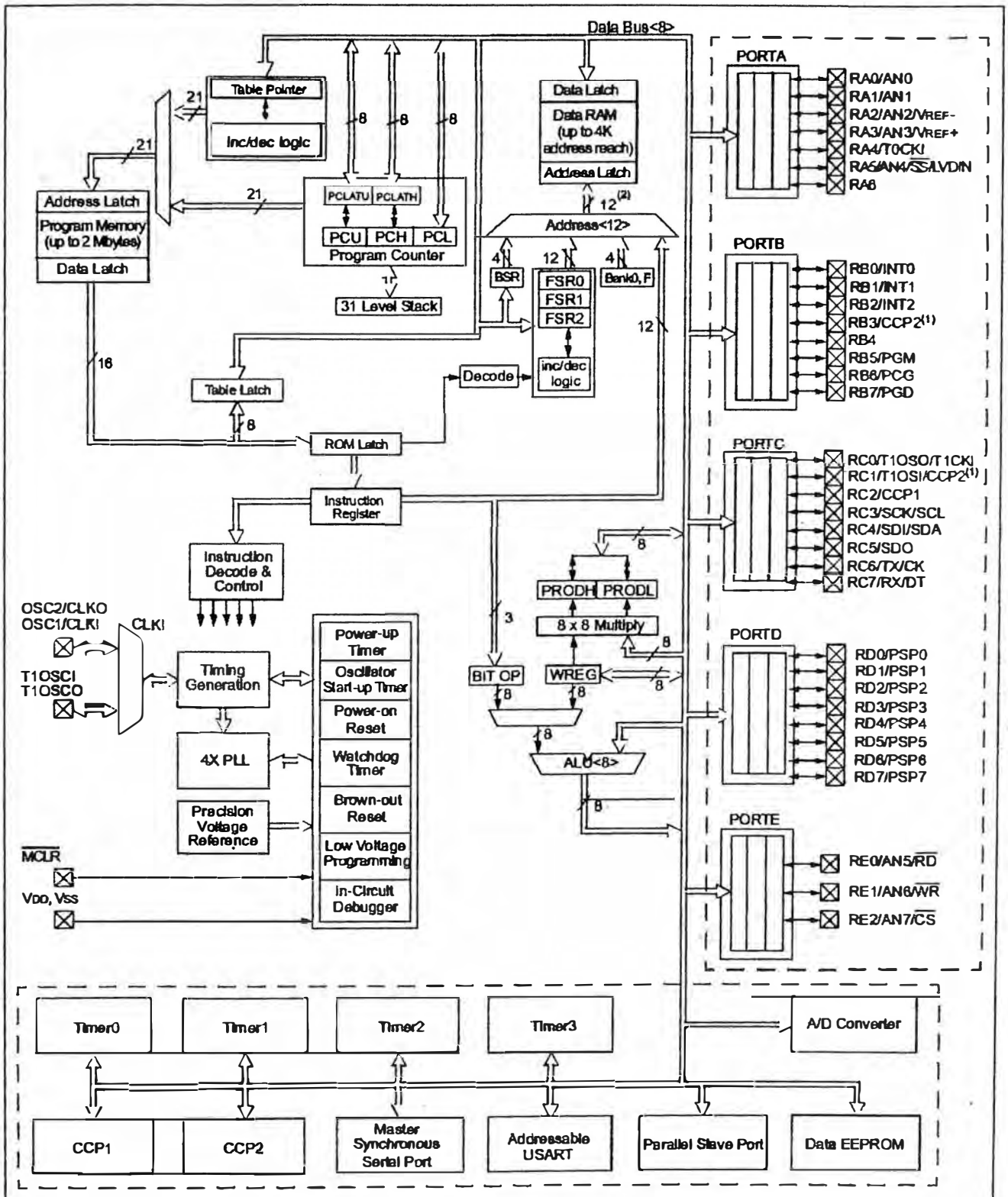
These devices come in 28-pin and 40/44-pin packages. The 28-pin devices do not have a Parallel Slave Port (PSP) implemented and the number of Analog-to-Digital (A/D) converter input channels is reduced to 5. An overview of features is shown in Table 1-1.

The following two figures are device block diagrams sorted by pin count: 28-pin for Figure 1-1 and 40/44-pin for Figure 1-2. The 28-pin and 40/44-pin pinouts are listed in Table 1-2 and Table 1-3, respectively.

TABLE 1-1: DEVICE FEATURES

Features	PIC18F242	PIC18F252	PIC18F442	PIC18F452
Operating Frequency	DC - 40 MHz	DC - 40 MHz	DC - 40 MHz	DC - 40 MHz
Program Memory (Bytes)	16K	32K	16K	32K
Program Memory (Instructions)	8192	16384	8192	16384
Data Memory (Bytes)	768	1536	768	1536
Data EEPROM Memory (Bytes)	256	256	256	256
Interrupt Sources	17	17	18	18
I/O Ports	Ports A, B, C	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C, D, E
Timers	4	4	4	4
Capture/Compare/PWM Modules	2	2	2	2
Serial Communications	MSSP, Addressable USART	MSSP, Addressable USART	MSSP, Addressable USART	MSSP, Addressable USART
Parallel Communications	—	—	PSP	PSP
10-bit Analog-to-Digital Module	5 input channels	5 input channels	8 input channels	8 input channels
RESETS (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)
Programmable Low Voltage Detect	Yes	Yes	Yes	Yes
Programmable Brown-out Reset	Yes	Yes	Yes	Yes
Instruction Set	75 Instructions	75 Instructions	75 Instructions	75 Instructions
Packages	28-pin DIP 28-pin SOIC	28-pin DIP 28-pin SOIC	40-pin DIP 44-pin PLCC 44-pin TQFP	40-pin DIP 44-pin PLCC 44-pin TQFP

FIGURE 1-2: PIC18F4X2 BLOCK DIAGRAM



Note 1: Optional multiplexing of CCP2 input/output with RB3 is enabled by selection of configuration bit.

2: The high order bits of the Direct Address for the RAM are from the BSR register (except for the `MOVFF` instruction).

3: Many of the general purpose I/O pins are multiplexed with one or more peripheral module functions. The multiplexing combinations are device dependent.

TABLE 1-3: PIC18F4X2 PINOUT I/O DESCRIPTIONS

Pin Name	Pin Number			Pin Type	Buffer Type	Description
	DIP	PLCC	TQFP			
MCLR/VPP	1	2	18			Master Clear (input) or high voltage ICSP programming enable pin.
MCLR				I	ST	Master Clear (Reset) input. This pin is an active low RESET to the device.
VPP				I	ST	High voltage ICSP programming enable pin.
NC	—			—	—	These pins should be left unconnected.
OSC1/CLKI	13	14	30			Oscillator crystal or external clock input.
OSC1				I	ST	Oscillator crystal input or external clock source input. ST buffer when configured in RC mode, CMOS otherwise.
CLKI				I	CMOS	External clock source input. Always associated with pin function OSC1. (See related OSC1/CLKI, OSC2/CLKO pins.)
OSC2/CLKO/RA6	14	15	31			Oscillator crystal or clock output.
OSC2				O	—	Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode.
CLKO				O	—	In RC mode, OSC2 pin outputs CLKO, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
RA6				I/O	TTL	General Purpose I/O pin.
						PORTA is a bi-directional I/O port.
RA0/AN0	2	3	19			
RA0				I/O	TTL	Digital I/O.
AN0				I	Analog	Analog input 0.
RA1/AN1	3	4	20			
RA1				I/O	TTL	Digital I/O.
AN1				I	Analog	Analog input 1.
RA2/AN2/VREF-	4	5	21			
RA2				I/O	TTL	Digital I/O.
AN2				I	Analog	Analog input 2.
VREF-				I	Analog	A/D Reference Voltage (Low) input.
RA3/AN3/VREF+	5	6	22			
RA3				I/O	TTL	Digital I/O.
AN3				I	Analog	Analog input 3.
VREF+				I	Analog	A/D Reference Voltage (High) input.
RA4/T0CKI	6	7	23			
RA4				I/O	ST/OD	Digital I/O. Open drain when configured as output.
T0CKI				I	ST	Timer0 external clock input.
RA5/AN4/SS/LVDIN	7	8	24			
RA5				I/O	TTL	Digital I/O.
AN4				I	Analog	Analog input 4.
SS				I	ST	SPI Slave Select input.
LVDIN				I	Analog	Low Voltage Detect Input.
RA6						(See the OSC2/CLKO/RA6 pin.)

Legend: TTL = TTL compatible input
CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels
I = Input
O = Output
P = Power
OD = Open Drain (no P diode to VDD)

TABLE 1-3: PIC18F4X2 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number			Pin Type	Buffer Type	Description
	DIP	PLCC	TQFP			
						PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-ups on all inputs.
RB0/INT0	33	36	8			
RB0				I/O	TTL	Digital I/O.
INT0				I	ST	External Interrupt 0.
RB1/INT1	34	37	9			
RB1				I/O	TTL	
INT1				I	ST	External Interrupt 1.
RB2/INT2	35	38	10			
RB2				I/O	TTL	Digital I/O.
INT2				I	ST	External Interrupt 2.
RB3/CCP2	36	39	11			
RB3				I/O	TTL	Digital I/O.
CCP2				I/O	ST	Capture2 input, Compare2 output, PWM2 output.
RB4	37	41	14	I/O	TTL	Digital I/O. Interrupt-on-change pin.
RB5/PGM	38	42	15			
RB5				I/O	TTL	Digital I/O. Interrupt-on-change pin.
PGM				I/O	ST	Low Voltage ICSP programming enable pin.
RB6/PGC	39	43	16			
RB6				I/O	TTL	Digital I/O. Interrupt-on-change pin.
PGC				I/O	ST	In-Circuit Debugger and ICSP programming clock pin.
RB7/PGD	40	44	17			
RB7				I/O	TTL	Digital I/O. Interrupt-on-change pin.
PGD				I/O	ST	In-Circuit Debugger and ICSP programming data pin.

Legend: TTL = TTL compatible input
CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels
I = Input
O = Output
P = Power
OD = Open Drain (no P diode to VDD)

TABLE 1-3: PIC18F4X2 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number			Pin Type	Buffer Type	Description
	DIP	PLCC	TQFP			
						PORTC is a bi-directional I/O port.
RC0/T1OSO/T1CKI	15	16	32			
RC0				I/O	ST	Digital I/O.
T1OSO				O	—	Timer1 oscillator output.
T1CKI				I	ST	Timer1/Timer3 external clock input.
RC1/T1OSI/CCP2	16	18	35			
RC1				I/O	ST	Digital I/O.
T1OSI				I	CMOS	Timer1 oscillator input.
CCP2				I/O	ST	Capture2 input, Compare2 output, PWM2 output.
RC2/CCP1	17	19	36			
RC2				I/O	ST	Digital I/O.
CCP1				I/O	ST	Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	18	20	37			
RC3				I/O	ST	Digital I/O.
SCK				I/O	ST	Synchronous serial clock input/output for SPI mode.
SCL				I/O	ST	Synchronous serial clock input/output for I^2C mode.
RC4/SDI/SDA	23	25	42			
RC4				I/O	ST	Digital I/O.
SDI				I	ST	SPI Data In.
SDA				I/O	ST	I^2C Data I/O.
RC5/SDO	24	26	43			
RC5				I/O	ST	Digital I/O.
SDO				O	—	SPI Data Out.
RC6/TX/CK	25	27	44			
RC6				I/O	ST	Digital I/O.
TX				O	—	USART Asynchronous Transmit.
CK				I/O	ST	USART Synchronous Clock (see related RX/DT).
RC7/RX/DT	26	29	1			
RC7				I/O	ST	Digital I/O.
RX				I	ST	USART Asynchronous Receive.
DT				I/O	ST	USART Synchronous Data (see related TX/CK).

Legend: TTL = TTL compatible input
CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels
I = Input
O = Output
P = Power
OD = Open Drain (no P diode to VDD)

TABLE 1-3: PIC18F4X2 PINOUT I/O DESCRIPTIONS (CONTINUED)

Pin Name	Pin Number			Pin Type	Buffer Type	Description
	DIP	PLCC	TQFP			
						PORTD is a bi-directional I/O port, or a Parallel Slave Port (PSP) for interfacing to a microprocessor port. These pins have TTL input buffers when PSP module is enabled.
RD0/PSP0	19	21	38	I/O	ST TTL	Digital I/O. Parallel Slave Port Data.
RD1/PSP1	20	22	39	I/O	ST TTL	Digital I/O. Parallel Slave Port Data.
RD2/PSP2	21	23	40	I/O	ST TTL	Digital I/O. Parallel Slave Port Data.
RD3/PSP3	22	24	41	I/O	ST TTL	Digital I/O. Parallel Slave Port Data.
RD4/PSP4	27	30	2	I/O	ST TTL	Digital I/O. Parallel Slave Port Data.
RD5/PSP5	28	31	3	I/O	ST TTL	Digital I/O. Parallel Slave Port Data.
RD6/PSP6	29	32	4	I/O	ST TTL	Digital I/O. Parallel Slave Port Data.
RD7/PSP7	30	33	5	I/O	ST TTL	Digital I/O. Parallel Slave Port Data.
						PORTE is a bi-directional I/O port.
RE0/$\overline{RD}$/AN5 RE0 $\overline{RD}$ AN5	8	9	25	I/O ST TTL Analog		 Digital I/O. Read control for parallel slave port (see also $\overline{WR}$ and $\overline{CS}$ pins). Analog input 5.
RE1/$\overline{WR}$/AN6 RE1 $\overline{WR}$ AN6	9	10	26	I/O ST TTL Analog		 Digital I/O. Write control for parallel slave port (see $\overline{CS}$ and $\overline{RD}$ pins). Analog input 6.
RE2/$\overline{CS}$/AN7 RE2 $\overline{CS}$ AN7	10	11	27	I/O ST TTL Analog		 Digital I/O. Chip Select control for parallel slave port (see related $\overline{RD}$ and $\overline{WR}$). Analog input 7.
VSS	12, 31	13, 34	6, 29	P	—	Ground reference for logic and I/O pins.
VDD	11, 32	12, 35	7, 28	P	—	Positive supply for logic and I/O pins.

Legend: TTL = TTL compatible input
CMOS = CMOS compatible input or output
ST = Schmitt Trigger input with CMOS levels
I = Input
O = Output
P = Power
OD = Open Drain (no P diode to VDD)

TABLE 20-2: PIC18FXXX INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d, a	Add WREG and f	1	0010	01da	ffff	ffff	C, DC, Z, OV, N	1, 2
ADDWFC	f, d, a	Add WREG and Carry bit to f	1	0010	0da	ffff	ffff	C, DC, Z, OV, N	1, 2
ANDWF	f, d, a	AND WREG with f	1	0001	01da	ffff	ffff	Z, N	1,2
CLRF	f, a	Clear f	1	0110	101a	ffff	ffff	Z	2
COMF	f, d, a	Complement f	1	0001	11da	ffff	ffff	Z, N	1, 2
CPFSEQ	f, a	Compare f with WREG, skip =	1 (2 or 3)	0110	001a	ffff	ffff	None	4
CPFSGT	f, a	Compare f with WREG, skip >	1 (2 or 3)	0110	010a	ffff	ffff	None	4
CPFSLT	f, a	Compare f with WREG, skip <	1 (2 or 3)	0110	000a	ffff	ffff	None	1, 2
DECF	f, d, a	Decrement f	1	0000	01da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
DECFSZ	f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010	11da	ffff	ffff	None	1, 2, 3, 4
DCFSNZ	f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100	11da	ffff	ffff	None	1, 2
INCF	f, d, a	Increment f	1	0010	10da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4
INCFSZ	f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011	11da	ffff	ffff	None	4
INFSNZ	f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100	10da	ffff	ffff	None	1, 2
IORWF	f, d, a	Inclusive OR WREG with f	1	0001	00da	ffff	ffff	Z, N	1, 2
MOVF	f, d, a	Move f	1	0101	00da	ffff	ffff	Z, N	1
MOVFF	f_s, f_d	Move f_s (source) to 1st word f_d (destination) 2nd word	2	1100 1111	ffff ffff	ffff ffff	ffff ffff	None	
MOVWF	f, a	Move WREG to f	1	0110	111a	ffff	ffff	None	
MULWF	f, a	Multiply WREG with f	1	0000	001a	ffff	ffff	None	
NEGF	f, a	Negate f	1	0110	110a	ffff	ffff	C, DC, Z, OV, N	1, 2
RLCF	f, d, a	Rotate Left f through Carry	1	0011	01da	ffff	ffff	C, Z, N	
RLNCF	f, d, a	Rotate Left f (No Carry)	1	0100	01da	ffff	ffff	Z, N	1, 2
RRCF	f, d, a	Rotate Right f through Carry	1	0011	00da	ffff	ffff	C, Z, N	
RRNCF	f, d, a	Rotate Right f (No Carry)	1	0100	00da	ffff	ffff	Z, N	
SETF	f, a	Set f	1	0110	100a	ffff	ffff	None	
SUBFWB	f, d, a	Subtract f from WREG with borrow	1	0101	01da	ffff	ffff	C, DC, Z, OV, N	1, 2
SUBWF	f, d, a	Subtract WREG from f	1	0101	11da	ffff	ffff	C, DC, Z, OV, N	
SUBWFB	f, d, a	Subtract WREG from f with borrow	1	0101	10da	ffff	ffff	C, DC, Z, OV, N	1, 2
SWAPF	f, d, a	Swap nibbles in f	1	0011	10da	ffff	ffff	None	4
TSTFSZ	f, a	Test f, skip if 0	1 (2 or 3)	0110	011a	ffff	ffff	None	1, 2
XORWF	f, d, a	Exclusive OR WREG with f	1	0001	10da	ffff	ffff	Z, N	
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b, a	Bit Clear f	1	1001	bbba	ffff	ffff	None	1, 2
BSF	f, b, a	Bit Set f	1	1000	bbba	ffff	ffff	None	1, 2
BTFSC	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011	bbba	ffff	ffff	None	3, 4
BTFSS	f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010	bbba	ffff	ffff	None	3, 4
BTG	f, d, a	Bit Toggle f	1	0111	bbba	ffff	ffff	None	1, 2

Note 1: When a PORT register is modified as a function of itself (e.g., `MOVF PORTB, 1, 0`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this Instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

4: Some instructions are 2-word instructions. The second word of these instructions will be executed as a NOP, unless the first word of the instruction retrieves the information embedded in these 16-bits. This ensures that all program memory locations have a valid instruction.

5: If the Table Write starts the write cycle to internal memory, the write will continue until terminated.

TABLE 20-2: PIC18FXXX INSTRUCTION SET (CONTINUED)

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
CONTROL OPERATIONS									
BC	n	Branch if Carry	1 (2)	1110	0010	nnnn	nnnn	None	
BN	n	Branch if Negative	1 (2)	1110	0110	nnnn	nnnn	None	
BNC	n	Branch if Not Carry	1 (2)	1110	0011	nnnn	nnnn	None	
BNN	n	Branch if Not Negative	1 (2)	1110	0111	nnnn	nnnn	None	
BNOV	n	Branch if Not Overflow	1 (2)	1110	0101	nnnn	nnnn	None	
BNZ	n	Branch if Not Zero	2	1110	0001	nnnn	nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110	0100	nnnn	nnnn	None	
BRA	n	Branch Unconditionally	1 (2)	1101	0nnn	nnnn	nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110	0000	nnnn	nnnn	None	
CALL	n, s	Call subroutine1st word	2	1110	110s	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
CLRWDT	—	Clear Watchdog Timer	1	0000	0000	0000	0100	$\overline{TO}$, $\overline{PD}$	
DAW	—	Decimal Adjust WREG	1	0000	0000	0000	0111	C	
GOTO	n	Go to address1st word	2	1110	1111	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
NOP	—	No Operation	1	0000	0000	0000	0000	None	
NOP	—	No Operation	1	1111	xxxx	xxxx	xxxx	None	4
POP	—	Pop top of return stack (TOS)	1	0000	0000	0000	0110	None	
PUSH	—	Push top of return stack (TOS)	1	0000	0000	0000	0101	None	
RCALL	n	Relative Call	2	1101	1nnn	nnnn	nnnn	None	
RESET		Software device RESET	1	0000	0000	1111	1111	All	
RETFIE	s	Return from interrupt enable	2	0000	0000	0001	000s	GIE/GIEH, PEIE/GIEL	
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
RETURN	s	Return from Subroutine	2	0000	0000	0001	001s	None	
SLEEP	—	Go into Standby mode	1	0000	0000	0000	0011	$\overline{TO}$, $\overline{PD}$	

Note 1: When a PORT register is modified as a function of itself (e.g., `MOVF PORTB, 1, 0`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

4: Some instructions are 2-word instructions. The second word of these instructions will be executed as a NOP, unless the first word of the instruction retrieves the information embedded in these 16-bits. This ensures that all program memory locations have a valid instruction.

5: If the Table Write starts the write cycle to internal memory, the write will continue until terminated.

TABLE 20-2: PIC18FXXX INSTRUCTION SET (CONTINUED)

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word MSb			LSb	Status Affected	Notes
LITERAL OPERATIONS									
ADDLW	k	Add literal and WREG	1	0000	1111	kkkk	kkkk	C, DC, Z, OV, N	
ANDLW	k	AND literal with WREG	1	0000	1011	kkkk	kkkk	Z, N	
IORLW	k	Inclusive OR literal with WREG	1	0000	1001	kkkk	kkkk	Z, N	
LFSR	f, k	Move literal (12-bit) 2nd word to FSRx 1st word	2	1110 1111	1110 0000	00ff kkkk	kkkk kkkk	None	
MOVLB	k	Move literal to BSR<3:0>	1	0000	0001	0000	kkkk	None	
MOVLW	k	Move literal to WREG	1	0000	1110	kkkk	kkkk	None	
MULLW	k	Multiply literal with WREG	1	0000	1101	kkkk	kkkk	None	
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
SUBLW	k	Subtract WREG from literal	1	0000	1000	kkkk	kkkk	C, DC, Z, OV, N	
XORLW	k	Exclusive OR literal with WREG	1	0000	1010	kkkk	kkkk	Z, N	
DATA MEMORY ↔ PROGRAM MEMORY OPERATIONS									
TBLRD*		Table Read	2	0000	0000	0000	1000	None	
TBLRD*+		Table Read with post-increment		0000	0000	0000	1001	None	
TBLRD*-		Table Read with post-decrement		0000	0000	0000	1010	None	
TBLRD+*		Table Read with pre-increment		0000	0000	0000	1011	None	
TBLWT*		Table Write	2 (5)	0000	0000	0000	1100	None	
TBLWT*+		Table Write with post-increment		0000	0000	0000	1101	None	
TBLWT*-		Table Write with post-decrement		0000	0000	0000	1110	None	
TBLWT+*		Table Write with pre-increment		0000	0000	0000	1111	None	

Note 1: When a PORT register is modified as a function of itself (e.g., `MOVF PORTB, 1, 0`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned.

3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

4: Some instructions are 2-word instructions. The second word of these instructions will be executed as a NOP, unless the first word of the instruction retrieves the information embedded in these 16-bits. This ensures that all program memory locations have a valid instruction.

5: If the Table Write starts the write cycle to internal memory, the write will continue until terminated.

ANEXO D

MICROCHIP

AN937

Implementing a PID Controller Using a PIC18 MCU

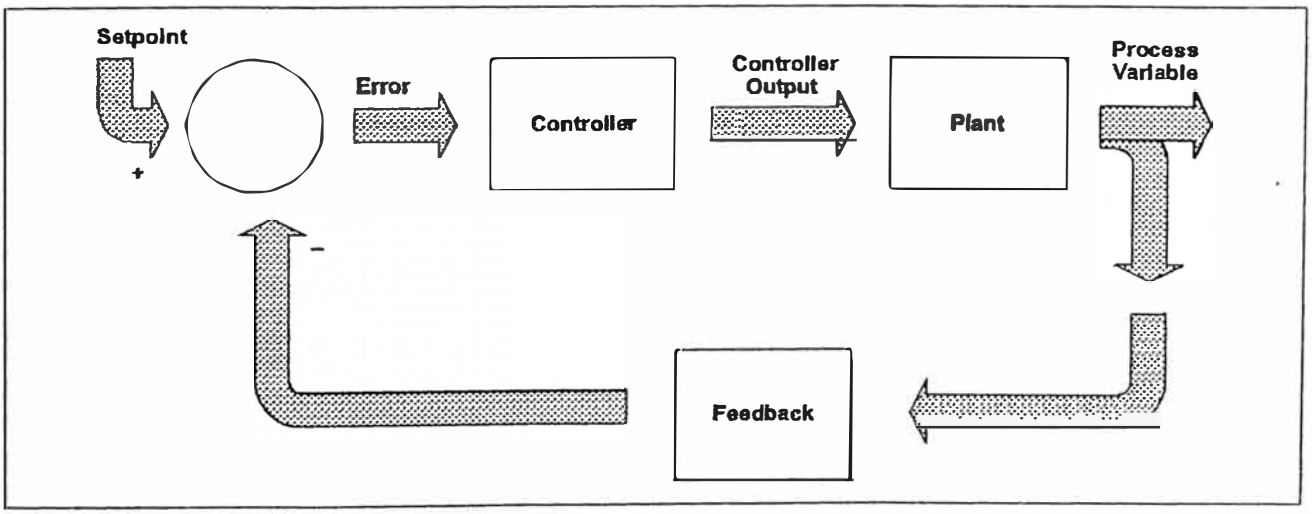
Author: *Chris Valenti*
Microchip Technology Inc.

INTRODUCTION

Continuous processes have been controlled by feedback loops since the late 1700's. In 1788, James Watt used a flyball governor on his steam engine to regulate its speed. The Taylor Instrument Company implemented the first fully functional Proportional, Integral and Derivative (PID) controller in 1940. Although feedback control has come a long way since James Watt, the basic approach and system elements have not changed. There are several elements within a feedback system; for discussion purposes, we will use a home heating temperature control system as our model in the descriptions below.

- **Plant** – The physical heating and cooling parts of the system.
- **Sensors** – The devices (thermistors measuring temperature) that measure the variables within the Plant.
- **Setpoint** – This is a value (i.e., 70 degrees), which is converted to a voltage that the process drives towards.
- **Error Signal** – This is the difference between the response of the Plant and the desired response (Setpoint). In a house, the thermostat may be set to 70 degrees, but the temperature is actually 65 degrees, therefore resulting in an error of 5 degrees (Error = Setpoint – Measured).
- **Disturbances** – These are unwanted inputs to the Plant, which can be common. A disturbance would be an open entry door allowing a gust of cold air to blow in, quickly dropping the temperature and causing the heat to come on.
- **Controller** – Intentionally left for last, this is the most significant element of a control system. The Controller is responsible for several tasks and is the link that connects together all of the physical and nonphysical elements. It measures the output signal of the Plant's Sensors, processes the signal and then derives an error based on the signal measurement and the Setpoint. Once the sensor data has been collected and processed, the result must be used to find PID values, which then must be sent out to the Plant for error correction. The rate at which all of this happens is dependent upon the Controller's processing power. This may or may not be an issue depending on the response characteristic of the Plant. A temperature control system is much more forgiving on a Controller's processing capabilities than a motor control system. Figure 1 shows a basic block diagram of a feedback control system.

FIGURE 1: FEEDBACK CONTROL LOOP



OBJECTIVES

The objectives for this application note are to:

- discuss in detail the three elements of a PID Controller: Proportional, Integral and Derivative
- discuss a firmware PID routine on a PIC18 device
- discuss the implementation of a firmware-based PID that has the flexibility of adapting to different systems, but is capable of being specifically tuned later on
- discuss the details of tuning a PID once implementation has been completed

SOURCE CODE OVERVIEW

Before going further, let's discuss how the PID source code is configured. There is no specific way a PID should be implemented in firmware; the methods discussed in this application note only touch upon a few of the many possibilities.

The PID routine is configured in a manner that makes it modular. It is intended to be plugged into en existing piece of firmware, where the PID routine is passed the 8-bit or 16-bit error value (Desired Plant Response – Measured Plant Response). Therefore, the actual error value is calculated outside of the PID routine. If necessary, the code could be easily modified to do this calculation within the PID routine. The PID can be configured to receive the error in one of two ways, either as a percentage with a range of 0 to 100% (8-bit), or a range of 0 to 4000 (16-bit). This option is configured by a `#define` statement at the top of the PID source code with the PID's variable declarations. The gains for proportional, integral and derivative all have a range of 0 to 15. For resolution purposes, the gains are scaled by a factor of 16 with an 8-bit maximum of 255. A general flow showing how the PID routine would be implemented in the main application code is presented in Figure 2.

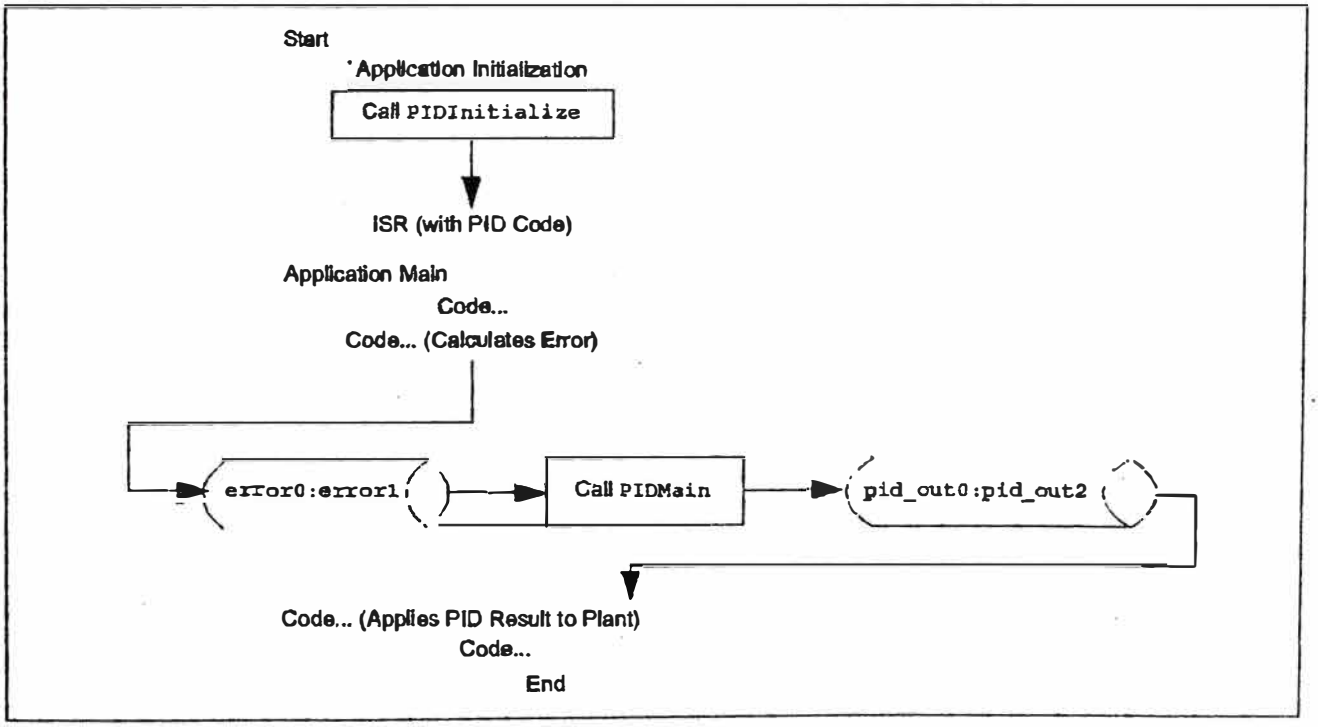
There were two methods considered for handling the signed numbers. The first method was to use signed math routines to handle all of the PID calculations. The second was to use unsigned math routines and maintain a sign bit in a status register. The latter method was implemented. There are five variables that require a sign bit to be maintained:

- `error`
- `a_error`
- `p_error`
- `d_error`
- `pid_error`

All of these sign bits are maintained in the pid_stat1 register (see Register 1).

Although all of the PID status bits are shown in Register 1 and Register 2, the user needs only to be concerned with the error sign bit (err_sign) and the PID final result sign bit (pid_sign). The err_sign bit is inserted into the PID routine along with the error. The user will check the pid_sign bit to determine which direction the Plant must be driven.

FIGURE 2: PID FIRMWARE IMPLEMENTATION



Firmware Variables and Constants

The list of firmware variables and constants and their definitions that are discussed in this application note are shown in Table 1.

TABLE 1: FIRMWARE VARIABLES AND CONSTANTS

Variable/Constant	Type	Definition
error0:error1	Error Variable	16-bit variable, difference between the Setpoint and measured output of the Plant
a_error0:a_error1	Error Variable	16-bit variable, accumulative error which is the sum of all past errors
d_error0:d_error1	Error Variable	16-bit variable, difference between error0:error1 and p_error0:p_error1
p_error0:p_error1	Error Variable	16-bit variable, value of the last error
a_err_1_lim	Error Variable	8-bit constant defining the accumulative error limits
a_err_2_lim	Error Variable	8-bit constant defining the accumulative error limits
kd	Gains	8-bit variable, derivative gain, max. = 15 (16 levels)
ki	Gains	8-bit variable, integral gain, max. = 15 (16 levels)
kp	Gains	8-bit variable, proportional gain, max. = 15 (16 levels)
pid_stat1	Status Register	8-bit variable, status bit register (see Register 1)
pid_stat2	Status Register	8-bit variable, status bit register (see Register 2)
deriv0:deriv2	Terms	24-bit variable, value of the derivative term
integ0:integ2	Terms	24-bit variable, value of the integral term
pid_out0:pid_out2	Terms	24-bit variable, final PID results
prop0:prop2	Terms	24-bit variable, value of the proportional term
timer1_hi	Time Base	8-bit constant loaded into the TMR1H register
timer1_lo	Time Base	8-bit constant loaded into the TMR1L register

Note: In 16-bit variables, the first variable is the Most Significant Byte (MSB), whereas the second variable is the Least Significant Byte (LSB). For example, in the variable error0:error1, error0 = MSB 8-bit and error1 = LSB 8-bit.

In 24-bit variables, the first variable is the MSB, whereas the last variable is the LSB. For example, in the variable pid_out0:pid_out2, pid_out0 = MSB 8-bit and pid_out2 = LSB 8-bit.

Data Registers

The pid_stat1 and pid_stat2 Data registers contain the individual PID status bits. The following two registers provide brief bit descriptions and their associated values.

REGISTER 1: pid_stat1 DATA REGISTER

pid_sign	d_err_sign	mag	p_err_sign	a_err_sign	err_sign	a_err_zero	err_zero
bit 7							bit 0

bit 7 **pid_sign:** Indicates sign of final PID result
1 = Result was positive
0 = Result was negative

bit 6 **d_err_sign:** Indicates sign of the derivative term
1 = Result was positive
0 = Result was negative

bit 5 **mag:** Indicates which variable is greater in magnitude (AARGB or BARGB)
1 = Result was AARGB
0 = Result was BARGB

bit 4 **p_err_sign:** Indicates sign of the previous error
1 = Result was positive
0 = Result was negative

bit 3 **a_err_sign:** Indicates sign of the accumulative error
1 = Result was positive
0 = Result was negative

bit 2 **err_sign:** Indicates sign of the error (input into the PID)
1 = Result was positive
0 = Result was negative

bit 1 **a_err_zero:** Indicates if the accumulative error is equal to zero or non-zero
1 = Result was zero
0 = Result was non-zero

bit 0 **err_zero:** Indicates if the error is equal to zero or non-zero
1 = Result was zero
0 = Result was non-zero

REGISTER 2: pid_stat2 DATA REGISTER

—	—	—	—	—	d_err_z	—	—
bit 7							bit 0

bit 7-3, 1-0 **Unimplemented**

bit 2 **d_err_z:** Indicates if the data error is equal to zero
1 = Result was zero
0 = Result was non-zero

PID Routine Flowcharts

Flowcharts for the PID main routine and the PID Interrupt Service Routine (ISR) functions are shown in Figure 3 and Figure 4 (see following pages).

The PID main routine is intended to be called from the main application code that updates the `error0:error1` variable, as well as the pid_stat1 error sign bit. Once in the PID main routine, the PID value will be calculated and put into the `pid_out0:pid_out2` variable, with its sign bit in pid_stat1. The value in `pid_out0:pid_out2` is converted by the application code to the correct value so that it can be applied to the Plant.

The PID ISR is configured for the PIC18 device's high priority interrupt at location 0x0008. The instructions within this ISR can be placed into an existing ISR, or kept as is and plugged into the application.

FIGURE 3: MAIN PID ROUTINE (`PIDMain`)

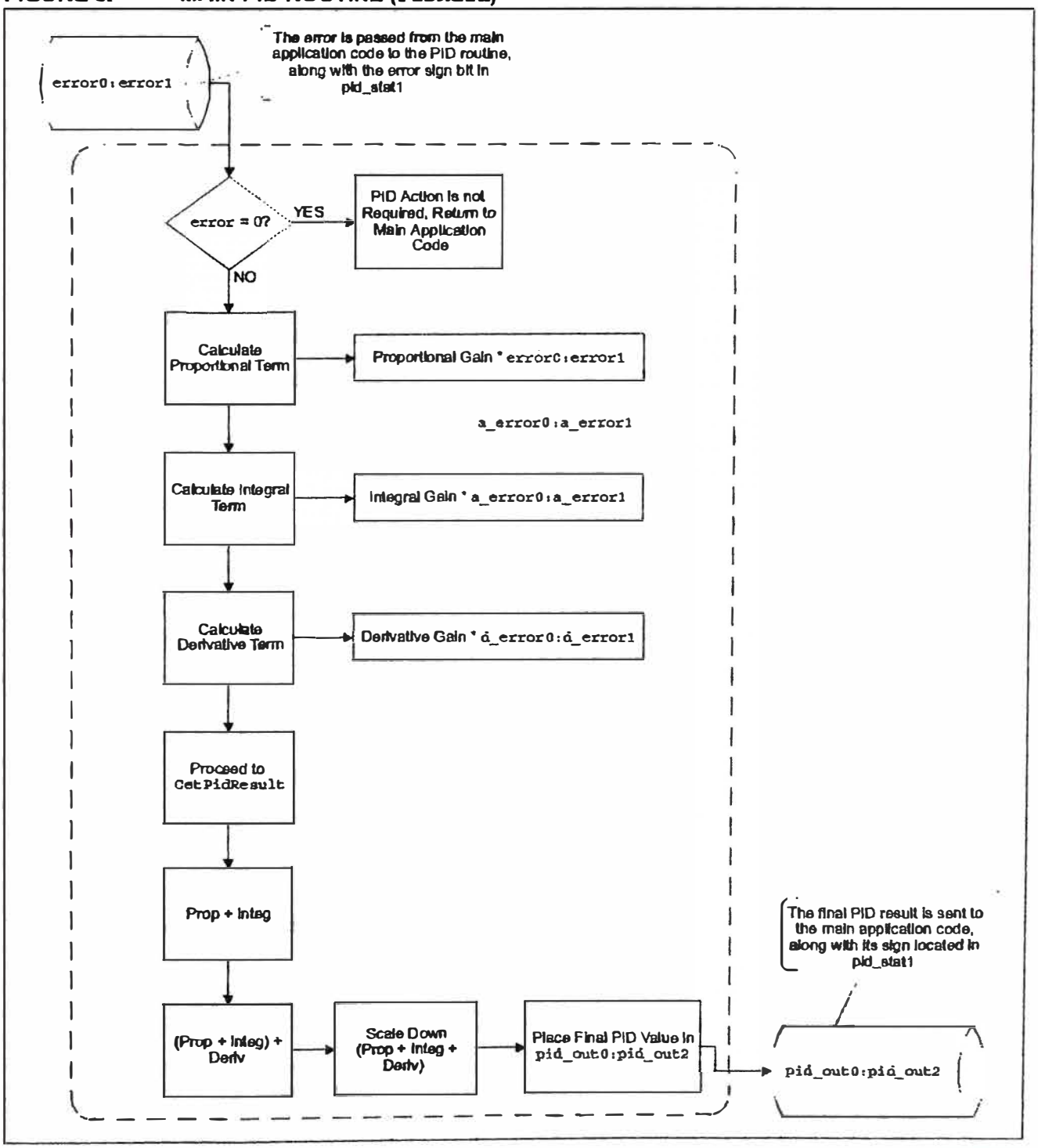
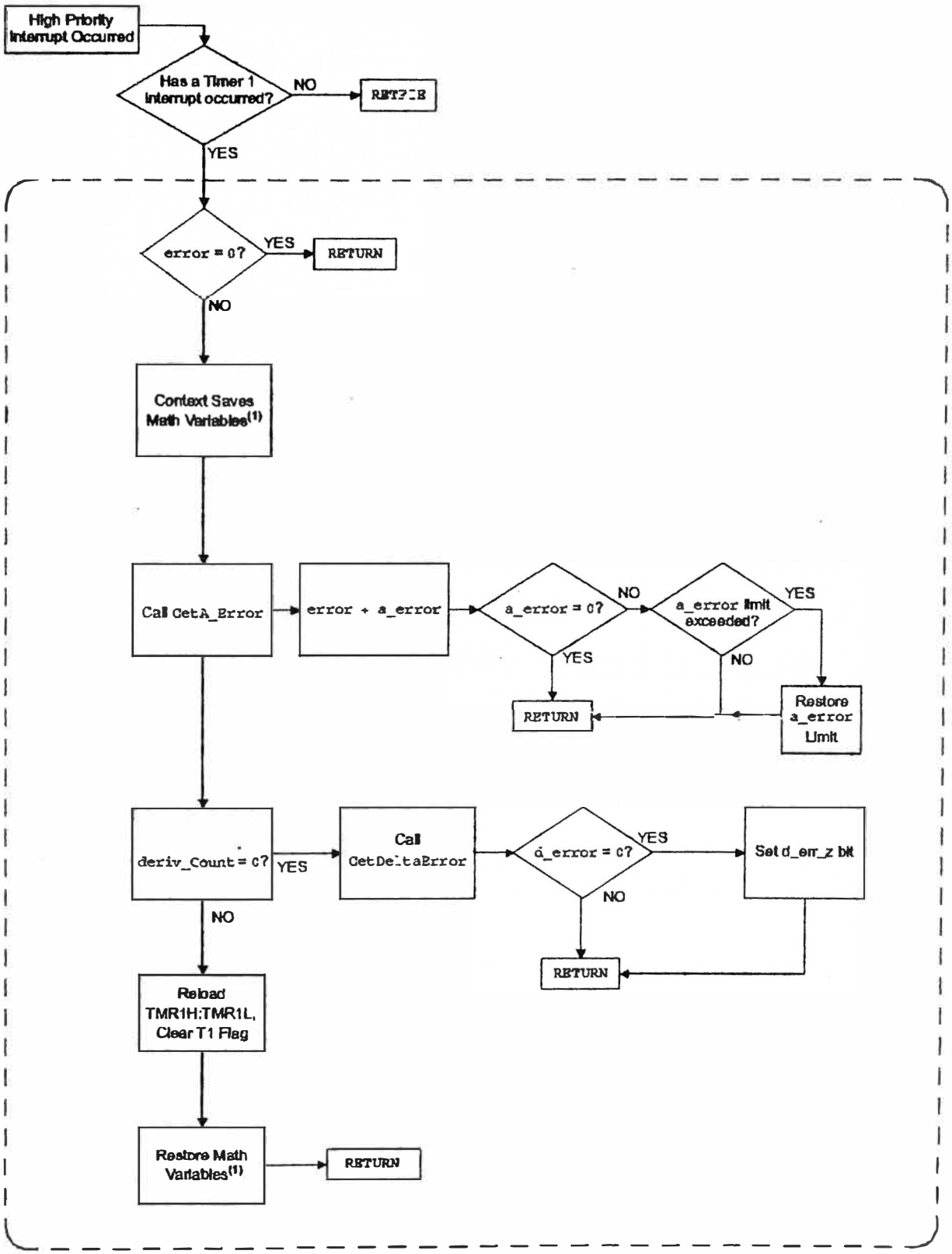


FIGURE 4: PID INTERRUPT ROUTINE (`PidInterrupt`)



Note 1: These instructions are options; they are dependant upon how the ISR is configured. The ISR code referenced in this application note is set up with context save/restore and is within the main application code ISR.

Proportional

The proportional term is the simplest of the three and is also the most commonly found control technique in a feedback system. The proportional gain (kp) is multiplied by the error. In this application note, the error is a 16-bit value, `error0:error1`. The amount of correction applied to the system is directly proportional to the error. As the gain increases, the applied correction to the Plant becomes more aggressive. This type of Controller is common for driving the error to a small, but non-zero value, leaving a steady state error. This is the reason for proportional control not being enough in some systems, thereby requiring integral and derivative control to come into play, separately or together (i.e., PI, PD or PID Controller).

IMPLEMENTATION

As mentioned earlier, the proportional is the simplest term. The error is multiplied by the proportional gain, `error0:error1` * kp. This is accomplished by the 16 * 16 multiplication routine. The result is stored in the 24-bit variable, `prop0:prop2`. This value will be used later in the code to calculate the overall value needed to go to the Plant.

EQUATION 1: PROPORTIONAL TERM

prop0:prop2 – kp * error0:error1

Integral

Unlike proportional control, which looks at the present error, integral control looks at past errors. Given this, the accumulative error (sum of all past errors) is used to calculate the integral term, but at fixed time intervals. Basically, every time the fixed interval expires, the current error at that moment is added to the `a_error` variable. A temperature system would require a longer sample period than a motor system because of the sluggish response in a temperature controlled environment. If the integral sample period was too fast in the temperatura system, the accumulative error would add too quickly to give the system a chance to respond, thereby not allowing it to ever stabilize. Another element in integral control to consider is 'wind-up'. Wind-up occurs when the accumulative error keeps increasing because the Plant output is saturated. This event can be avoided by setting limits to the accumulative error. It can also be eliminated by not executing the integral term when the Plant output is saturated. Another characteristic is excessive gain, that can create an unstable condition within the system, causing it to oscillate. The integral gain must be thoroughly tested for all possible situations to find the best overall value. In conclusion, as the accumulative error increases, the integral term has a greater effect on the Plant. In a sluggish system, this could dominate the value that is sent to the Plant.

IMPLEMENTATION

To obtain the integral term, the accumulated error must be retrieved. The accumulated error (`a_error0:a_error2`) is the sum of past errors. For this reason, the integral is known for looking at a system's history for correction. Refer to Table 2 for details on how `a_error` is accumulated.

Each time the PID routine receives an error, it may or may not be added to the accumulated error variable. This is dependant upon the Timer1 overflow rate. If Timer1 overflowed, then the error at that moment will be added to the accumulated error variable. The Timer1 overflow rate is interrupt driven and is configured as a high priority interrupt. The TMR1H:TMR1L registers are loaded with values defined by the constants, `timer1_hi` and `timer1_lo`. The values for these constants should be based on the Plant's response. The accumulated error will be multiplied by the integral gain, `a_error0:a_error2` * `ki` and the result is stored in `integ0:integ2`.

TABLE 2: `a_error` ACCUMULATION EXAMPLE

Time	Error	Timer1 Overflow	Accumulated Error
t = n	10%	No	x%
t = n + 1	8%	No	x%
t = n + 2	12%	Yes	x + 12%
t = n + 3	9%	No	(x% + 12%)
t = n + 4	6%	No	(x% + 12%)
t = n + 5	4%	Yes	(x% + 12%) + 4%
t = n + ...			

To avoid integral wind-up, accumulative error limits were installed (a_err_1_Lim:a_err_2_Lim). When the accumulative error is calculated, the result is compared against the limit variables. If the calculated value exceeds the limits, the accumulative error is made equal to the value that is determined by the user in the variable definition at the beginning of the code.

EQUATION 2: INTEGRAL TERM

integ0:integ2 – ki * a_error0:a_error1 (a_error0:a_error1 – error0:error1 + error0:error1 + ...error0:error1)

Derivative

As previously mentioned, the proportional term works on the present error, the integral term works on past errors and the derivative term works on the present and past error to forecast a future response of the system. The derivative term makes en adjustment based on the rate at which the Plant output is changing from its Setpoint. A notable characteristic in this type of control is when the error is constant, or at the maximum limit, the effect is minimal. There are some systems where proportional and/or integral do not provide enough control. In these systems, adding in the derivative term completes the control requirements.

IMPLEMENTATION

The derivative term is calculated in similar fashion to the integral term. Considering that the derivative term is based on the rate at which the system is changing, the derivative routine calculates d_error. This is the difference between the current error and the previous error. The rate at which this calculation takes place is dependant upon the Timer1 overflow. The derivative term can be extremely aggressive when it is acting on the error of the system. An alternative to this is to calculate the derivative term from the output of the system and not the error. In this application note, the error will be used. To keep the derivative term from being too aggressive, a derivative counter variable has been installed. This variable allows d_error to be calculated once for an x number of Timer1 overflows (unlike the accumulated error, which is calculated every Timer1 overflow).

To get the derivative term, the previous error is subtracted from the current error (d_errro0:d_error1 – error0:error – p_error0:p_error1). The difference is then multiplied by the derivative gain (kd) and this result is placed in deriv0:deriv2, which is added with the proportional and integral terms.

Tuning

There are several different ways to tune a PID Controller for system optimization. The code in this application note is loosely defined, giving it the flexibility to be tuned for a specific application (i.e., motor control, temperature, actuator, etc.).

Tuning a PID Controller can be somewhat difficult and time consuming and should be completed in a systematic fashion.

1. Run the system in an open loop and measure its response over time. Based on the measured response, you will get an idea for which PID term is needed most.
2. Determine the application requirements: Plant response time, which PID term will have the most affect and accumulative error limits.
3. Determine how often the a_error and d_error terms should be calculated; this will dictate the values loaded into the Timer1 and derivative counter registers.

In the current configuration, d_error is calculated once for every a_error calculation. Should this be less or more, or vice versa? Finally, once these variables are decided, the PID gains can be experimented with. Start with the smallest gains (i.e., kp = 1 * 16, ki = 1 * 16, kd = 1 * 16), slowly increasing these values until the desired output is reached. With a few code modifications, it is possible to make the Controller a proportional only Controller and tune this to an optimal value. Then it is possible to add the other terms one at a time, optimizing each time.

EQUATION 3: DERIVATIVE TERM

deriv0:deriv2 – kd * d_error0:d_error1 (d_error0:d_error1 – error0:error – p_error:p_error1)

The system response of a temperature controlled environment is shown in Figures 5 through 7. Figure 5 shows the graphic response for a proportional only feedback loop. As shown, none of the gain values can reach the input signal and maintain that level. All four gain values have settled at a non-zero value, as previously discussed.

FIGURE 5: PROPORTIONAL ONLY GRAPHIC RESPONSE

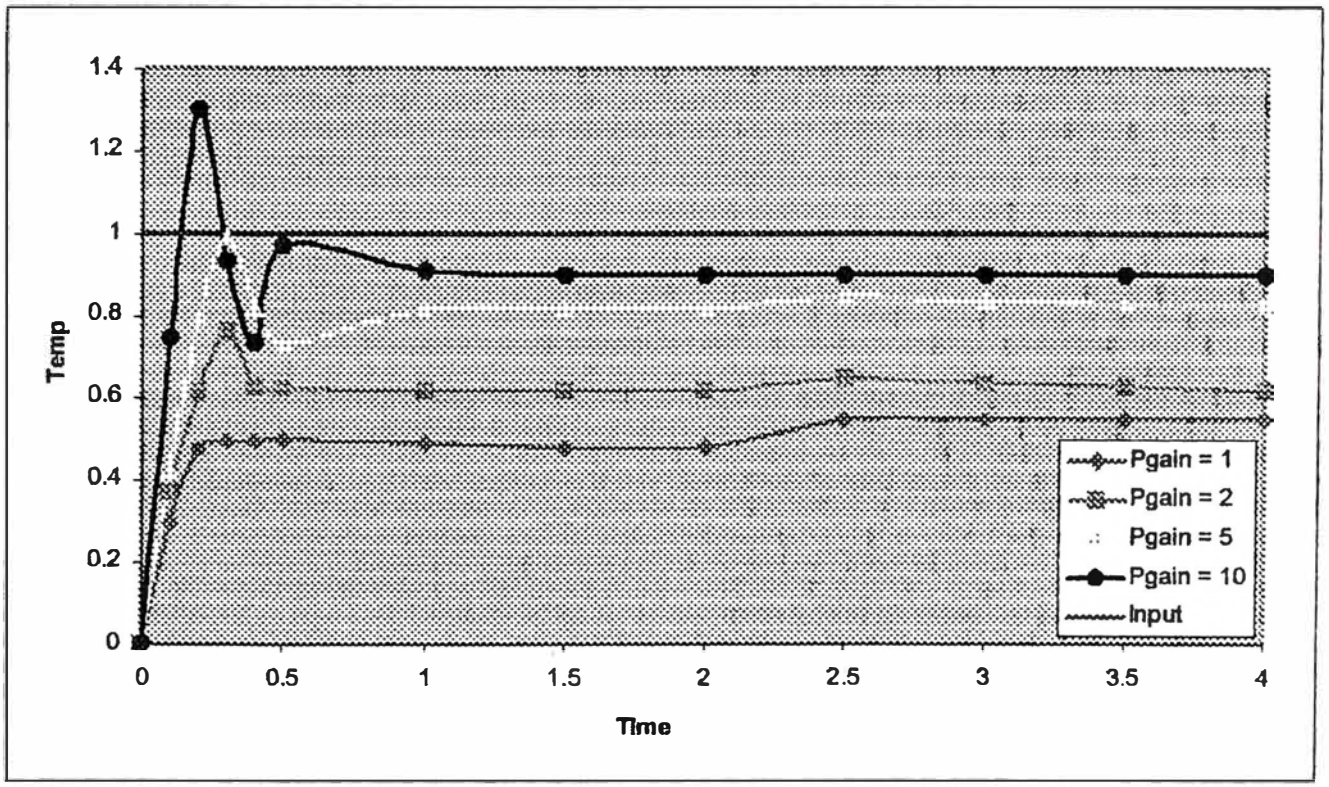


Figure 6 shows the graphic response of a Proportional/ Integral (PI) Controller. The high integral gain dominates the response (see line with diamond shapes).

With a tuned proportional and integral gain, the system does settle to its *Setpoint*, which is why PI control is adequate in many systems. The disadvantage is the time required for it to settle (t = 3), which brings us to PID control.

FIGURE 6: PROPORTIONAL/INTEGRAL (PI) CONTROLLER GRAPHIC RESPONSE

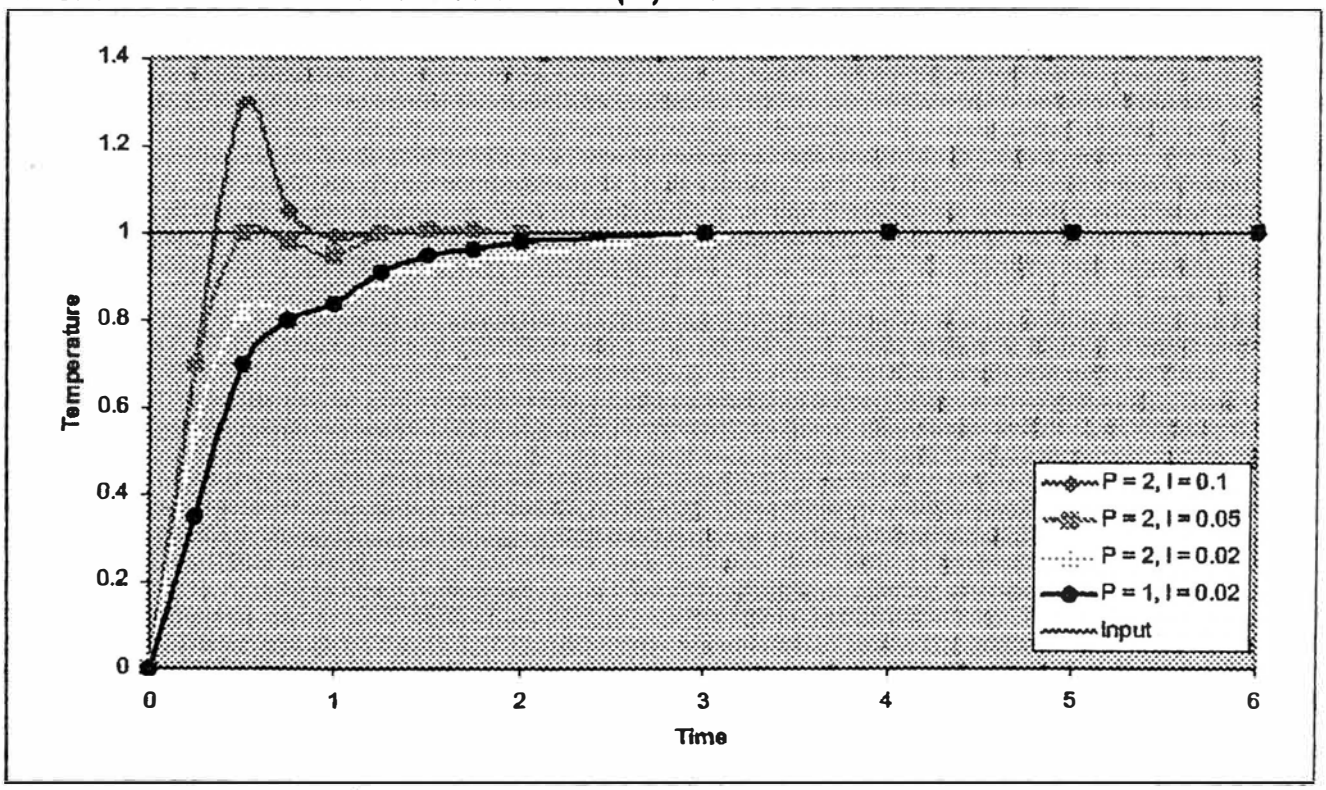
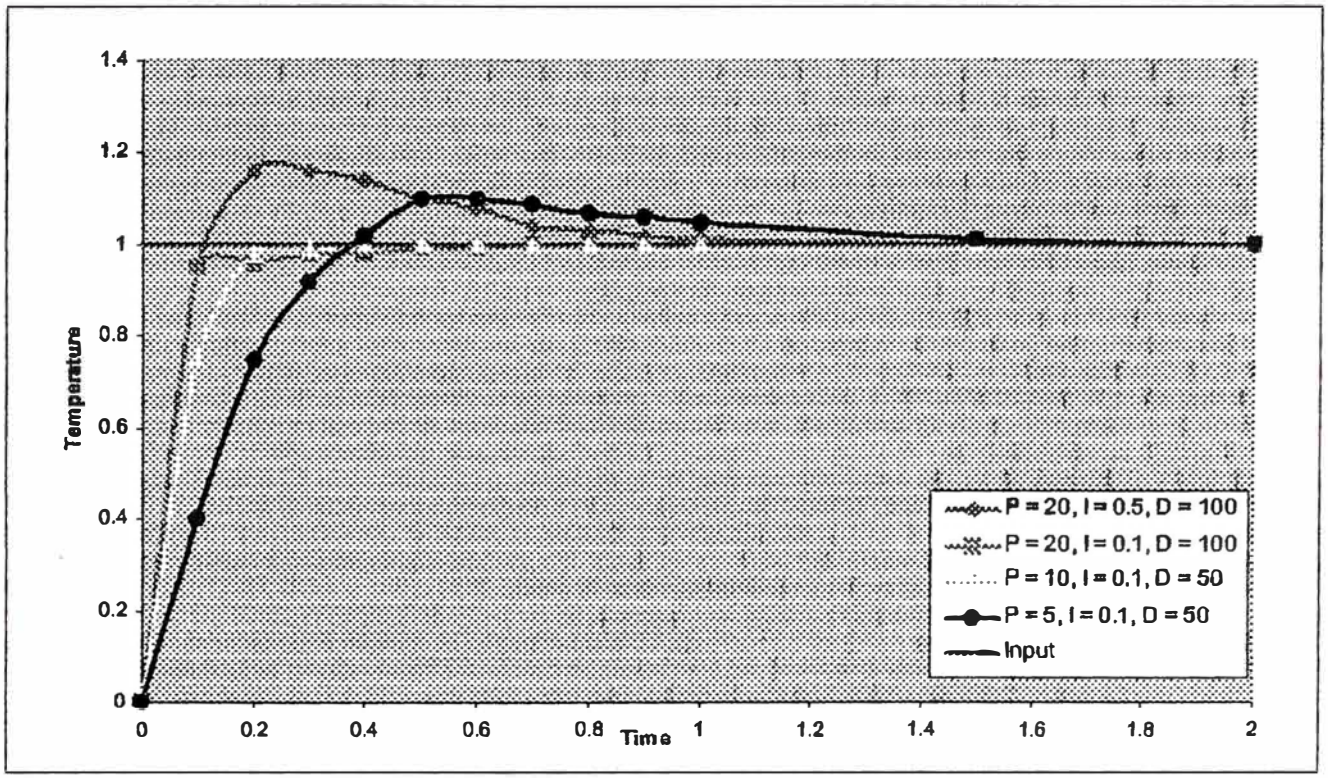


Figure 7 shows the graphic response of a PID Controller. This graph is very similar to the PI graph (Figure 6), except that the PID control takes half as long as the PI control to settle (t = 1.5) as the Setpoint.

FIGURE 7: PID CONTROLLER GRAPHIC RESPONSE



PID Output

The PID output is calculated after the proportional, integral and derivative terms have been determined. In addition to this calculation is the pid_sign bit, which the user must check to decide which direction the Plant will be driven. This bit is located in pid_stat1. The sum of all these terms is stored in `pid_out0:pid_out2`.

EQUATION 4: PID ROUTINE

PID Output = prop0:prop2 + integ0:integ2 + deriv0:deriv2

CONCLUSION

As mentioned in the introduction, the Controller's processing capabilities will dictate the system's ability to respond to the error. Table 3 shows a list of PID functions, each with the amount of instruction cycles and time required. In cases where the Plant response is sluggish, it may be possible to decrease the processor speed and save on power, but still be able to execute the PID routine in acceptable time.

TABLE 3: PID FUNCTIONS

Function	Instruction Cycles	Elapsed Time (µS) (TCY at 40 MHz)
PID Main	437	43.7
Proportional	50	5.0
Integral	52	5.2
Derivative	52	5.2
GetPidResult	270	27
GetA_Error	70	7.0
PID Interrupt	184	18.4

The measurements shown in Table 3 can vary, depending on the size of the error and how much of the math routines will be used. The measurements also reflect an error of 6% sent to the PID routine.

After the code development for this application note was completed, the PID routine was implemented on the PIC18F4431 Motor Control board (PICDEM™ MC). For the initial start of the motor, the PID gains were: kp = 96, ki = 80 and kd = 16. These were scaled values. After starting the motor and running it close to its set speed, the integral gain was changed to 144. The accumulated error was calculated every millisecond, initiated by a Timer1 overflow. The delta error (d_error) was calculated every 4 ms (derivative counter = 4).

APPENDIX A: SOURCE CODE

The complete source code, including the PID Application Maestro™ module, any demo applications and necessary support files, are available for download as a single archive file from the Microchip corporate web site at:

www.microchip.com

BIBLIOGRAFIA

[1] Manuela Martin Sanchez. " Ejemplos relacionados con electroquímica " www.etsil.upm.es/diquima/vidacotidiana/QVCParte2.pdf.

[2] Aldo Max Delgado."Guía de Laboratorio Electroquímica Industrial ". UNI-Perú 2001 .

[3] Comisión ambiental de México y GTZ. " Manual de residuos peligrosos giro galvanoplastía".1998.

[4] Hervilia Seco Lago, Julia Perez Iglesias. " Embellecimiento de objetos de la vida cotidiana ". www.etsil.upm.es/diquima/vidacotidiana/QVCParte2.pdf

[5] Wendell H. Slabaugh , Theran D. Parsons. " Química general " . EDITORIAL LIMUSA-WILEY, S:A. 1968.

[6] Raúl Benites Saravia. "CONTROL AVANZADO". UNI-Perú. 2005.

[7] Arturo Rojas Moreno. "CONTROL AVANZADO". UNI-Perú. 2001.

[8] Javier Donayre S. "Controlador Discreto PID" . Instituto de Automatización e Inteligencia Artificial. UNI-Perú .2004.

[9] Rubén Collado Hernández. "Control Motores de Corriente Contínua bajo Windows NT".www.elai.upm.es/spain/Investiga/GCII/publicaciones/PUb00/DOC002_00.pdf.

[10] Ziegler-Nichols. "Reglas de ajuste del PID." www.esi.us.es/~vivas/pra_RA_ITI/prac_5.pdf.

[11] C.Valenti. "PID_source_code.zip". 2004. www.microchip.com/stellent/idcplg?IdcService=

[12] MathWorks. " Matlab 6.5 Realase13 ". Junio 2002.

[13] Microchip Technology. " Datasheet 16F877 ". 2001, y en español : www.monografias.com/trabajos18/descripción-pic/descripcion-pic.shtml.

[14] National Semiconductor. " Datasheet LM335 ". 2000.

[15] Microchip Technology. " Datasheet 18F452 ". 2002.

[16] Microchip Technology. " Notas de Aplicación AN937a ". 2004. ww1.microchip.com/downloads/en/AppNotes/00937a.pdf.