



**UNIVERSIDADE LUTERANA DO BRASIL**  
**PRÓ-REITORIA DE GRADUAÇÃO**  
**DEPARTAMENTO DE ENGENHARIA ELÉTRICA**



**Filipe Cirtoli**

**CONTROLADOR PID COM SINTONIA AUTOMÁTICA**  
**IMPLEMENTADO EM PROCESSADOR DE 32 BITS**

Canoas, Julho de 2007

**FILIFE CIRTOLI**

**CONTROLADOR PID COM SINTONIA AUTOMÁTICA  
IMPLEMENTADO EM PROCESSADOR DE 32 BITS**

Trabalho de Conclusão de Curso apresentado  
ao Departamento de Engenharia Elétrica da  
ULBRA como um dos requisitos obrigatórios  
para a obtenção do grau de Engenheiro  
Eletricista

Orientador: Prof. Me. Eng°. Augusto Alexandre Durgante de Mattos

Canoas, Julho de 2007

**FILIPPE CIRTOLI**

**CONTROLADOR PID COM SINTONIA AUTOMÁTICA  
IMPLEMENTADO EM PROCESSADOR DE 32 BITS**

BANCA EXAMINADORA

---

Prof. ME. Eng<sup>a</sup>. Carla Diniz Lopes

---

Prof. ME. Eng<sup>o</sup>. Dalton Vidor

---

Orientador: Prof. Me. Eng<sup>o</sup>. Augusto Alexandre Durgante de Mattos

Agradeço a minha família pelo apoio e compreensão. Muito obrigado.

Agradeço ao meu orientador, professor Augusto de Mattos pelo apoio e ensinamentos.

Agradeço a minha noiva, Ana Paula, pela paciência, carinho e compreensão durante esta trajetória de pesquisa.

## **RESUMO**

O presente trabalho tem como objetivo o desenvolvimento de um controlador PID, implementado através de um processador de 32 bits (ARM7), utilizando o método de Ziegler-Nichols em malha-fechada. Este controlador foi testado em uma planta de segunda ordem com três fatores de amortecimento distintos, obtendo resultado satisfatório nos testes realizados.

**Palavras chave:** PID. ARM7. Controle Digital. LPC. Ziegler-Nichols.

## **ABSTRACT**

The present work has the purpose the development of a PID controller, implemented using a 32 bits processor (ARM7), using ZieglerNichols Method in closed loop. This controller was tested in a second order system with three different damping factors, obtaining a satisfactory result in all realized tests.

**Keywords:** PID. ARM7. Digital Control. LPC. Ziegler-Nichols.

## LISTA DE SÍMBOLOS

P	– Ação Proporcional
I	– Ação Integral
D	– Ação Derivativa
K <sub>p</sub>	– Ganho proporcional
T <sub>i</sub>	– Constante de tempo integrativa
T <sub>d</sub>	– Constante de tempo derivativa
L	– Tempo de retardo do método de ziegler-nichols (resposta ao salto)
T	– Constante de tempo do método de ziegler-nichols (resposta ao salto)
K <sub>cr</sub>	– Ganho crítico do método de ziegler-nichols (período crítico)
P <sub>cr</sub>	– Período crítico do método de ziegler-nichols (período crítico)
u <sub>1</sub>	– Valor máximo do valor de atuação do relé (ziegler-nichols método da realimentação com relé)
u <sub>2</sub>	– Valor mínimo do valor de atuação do relé (ziegler-nichols método da realimentação com relé)
d	– Diferença entre u <sub>1</sub> e u <sub>2</sub> (ziegler-nichols método da realimentação com relé)
A	– Amplitude de pico-a-pico da oscilação observada na variável controlada (ziegler-nichols método da realimentação com relé)
s	– Segundos
ms	– Milisegundos
Mo	– Overshoot
t <sub>s</sub>	– Tempo de estabilização
ω <sub>n</sub>	– Frequência de corte
ξ	– Fator de amortecimento
A/D	– Conversor analógico para digital
D/A	– Conversor digital para analógico

# SUMÁRIO

<b>1. INTRODUÇÃO .....</b>	<b>13</b>
<b>2. REFERENCIAL TEÓRICO.....</b>	<b>15</b>
2.1. Controladores PID industriais .....	15
2.1.1. Conceitos Gerais.....	15
2.1.2. Ação Proporcional.....	17
2.1.3. Ação Integral.....	17
2.1.4. Ação Derivativa.....	18
2.1.5. Escolha das Ações de Controle.....	19
2.1.6. Controlador P (somente ação proporcional).....	19
2.1.7. Controlador PI (ações proporcional e integral) .....	19
2.1.8. Controlador PID (ações proporcional, integral e derivativa).....	20
2.2. Controle Digital .....	20
2.3. Métodos de Ziegler-Nichols.....	23
2.3.1. Método da Resposta ao Salto.....	23
2.3.2. Método do Período Crítico.....	25
<b>3. IMPLEMENTAÇÃO DO HARDWARE.....</b>	<b>29</b>
3.1. O Microcontrolador LPC 2136 .....	29
3.1.1. Apresentação do núcleo ARM7.....	29
3.1.2. Características do LPC 2136.....	30
3.2. Hardware Básico de Trabalho.....	32
3.3. Plantas .....	34
3.4. Definição do Tempo de Amostragem .....	40
<b>4. IMPLEMENTAÇÃO DO SOFTWARE E RESULTADOS .....</b>	<b>43</b>
4.1. Interface com o usuário .....	45
4.2. Descrição da Função Principal do Software.....	45
4.3. Funções do Software.....	47
4.3.1. Função ADC.....	47
4.3.2. Função DAC.....	48
4.3.3. Função Wait.....	48
4.3.4. Função Zerar_Valores.....	48
4.3.5. Função Aquisição da Planta.....	48
4.3.6. Função Achar Máximo e Mínimo .....	48
4.3.7. Função Ganho Crítico.....	50
4.3.8. Função Período Crítico.....	50
4.3.9. Função Ziegler Nichols .....	50
4.2.10. Função PID.....	51
4.4. Resultados .....	51
<b>5. CONCLUSÃO.....</b>	<b>59</b>



<b>REFERÊNCIAS BIBLIOGRÁFICAS.....</b>	<b>60</b>
<b>ANEXOS .....</b>	<b>61</b>
<b>ANEXO A - PINAGEM DO LPC 2136.....</b>	<b>62</b>
<b>ANEXO B - KIT DE DESENVOLVIMENTO DA KEIL.....</b>	<b>66</b>
<b>ANEXO C - CÓDIGO FONTE DO SOFTWARE.....</b>	<b>67</b>

## LISTA DE FIGURAS

Figura 2.1 – Diagrama em blocos de um sistema de controle .....	16
Figura 2.2 – (a)Diagrama em bloco da ação proporcional [5];	
(b)Gráfico da atuação que a ação proporcional gera, para um erro do tipo salto .....	17
Figura 2.3 – (a)Diagrama em bloco da ação integral [5];	
(b)Gráfico da atuação que a ação integral gera, para um erro do tipo salto .....	18
Figura 2.4 – Diagramas em blocos do controlador digital .....	20
Figura 2.5 – Características da resposta ao salto do processo relevantes para o	
ajuste de Ziegler -Nichols .....	24
Figura 2.6 – Oscilações estáveis com períodos $P_{cr}$ .....	26
Figura 2.7 – (a)Diagrama em bloco do controle com relé [7];	
(b)Gráfico da resposta típica do controle de realimentação com relé .....	27
Figura 3.1 – Esquema elétrico do <i>hardware</i> .....	32
Figura 3.2 – Diagrama em blocos do kit de desenvolvimento .....	33
Figura 3.3 – Diagrama em blocos do kit de desenvolvimento ligado na planta .....	33
Figura 3.4 – Foto do kit montado.....	34
Figura 3.5 – Circuito operacional na forma de filtro passa-baixa .....	35
Figura 3.6 – Dois circuitos integrador em série .....	35
Figura 3.7 – Planta em malha fechada utilizada para teste do PID .....	37
Figura 3.8 – Foto das plantas montadas .....	37
Figura 3.9 – Gráfico da equação com $\xi = 0,2$ para entrada do tipo salto unitário .....	38
Figura 3.10 – Gráfico da equação com $\xi = 0,5$ para entrada do tipo salto unitário ...	39
Figura 3.11 – Gráfico da equação com $\xi = 0,8$ para entrada do tipo salto unitário ...	39
Figura 3.12 – Foto da planta e do <i>hardware</i> do microprocessador montado .....	40
Figura 3.13 – Gráfico da equação com $\xi = 0,5$ com tempo de amostragem de	
100 ms .....	41
Figura 3.14 – Gráfico da equação com $\xi = 0,5$ com tempo de amostragem de	
10 ms .....	41
Figura 3.15 – Gráfico da equação com $\xi = 0,5$ com tempo de amostragem de	
5 ms .....	42
Figura 3.16 – Gráfico da equação com $\xi = 0,5$ com tempo de amostragem de	
1 ms .....	42
Figura 4.1 – Ambiente IDE uVision.....	43
Figura 4.2 – Ambiente de gravação LPC2000 Flash Utility .....	44
Figura 4.3 – Programa funcionando no Hyper Terminal .....	45
Figura 4.4 – Diagrama em blocos do <i>software</i> .....	46
Figura 4.5 – Trecho de código onde habilita o A/D e o D/A.....	47
Figura 4.6 – Trecho de código da aquisição da planta .....	48
Figura 4.7 – Diagrama em blocos da função que encontra o máximo e o mínimo.....	49
Figura 4.8 – Código fonte da função que calcula o ganho crítico.....	50
Figura 4.9 – Código fonte da função que calcula os coeficientes $K_p$ , $T_i$ , $T_d$ através	
do método de Ziegler-Nichols.....	51
Figura 4.10 – Planta de $\xi = 0,2$ adquirida pelo microcontrolador em malha	
fechada com entrada do tipo salto .....	52

Figura 4.11 – Teste 1 da planta de $\xi = 0,5$ adquirida com ensaio de realimentação com relé.....	53
Figura 4.12 – Teste 1 da planta de $\xi = 0,5$ controlada e não controlada.....	53
Figura 4.13 – Teste 2 da planta de $\xi = 0,5$ adquirida com ensaio de realimentação com relé.....	54
Figura 4.14 – Teste 2 da planta de $\xi = 0,5$ controlada e não controlada.....	54
Figura 4.15 – Teste 3 da planta de $\xi = 0,5$ adquirida com ensaio de realimentação com relé.....	55
Figura 4.16 – Teste 3 da planta de $\xi = 0,5$ controlada e não controlada.....	55
Figura 4.17 – Teste 1 da planta de $\xi = 0,8$ adquirida com ensaio de realimentação com relé.....	56
Figura 4.18 – Teste 1 da planta de $\xi = 0,8$ controlada e não controlada.....	56
Figura 4.19 – Teste 2 da planta de $\xi = 0,8$ adquirida com ensaio de realimentação com relé.....	57
Figura 4.20 – Teste 2 da planta de $\xi = 0,8$ controlada e não controlada.....	57
Figura 4.21 – Teste 3 da planta de $\xi = 0,8$ adquirida com ensaio de realimentação com relé.....	58
Figura 4.22 – Teste 3 da planta de $\xi = 0,8$ controlada e não controlada.....	58

## LISTA DE TABELAS

Tabela 2.1 – Controladores comerciais [3], [4] .....	15
Tabela 2.2 – Tabela de Ziegler e Nichols pelo método da resposta ao Salto [5] .....	25
Tabela 2.3 – Fórmulas de Ziegler e Nichols para ajuste pelo método do período crítico [5] .....	26
Tabela 3.1 – Funções dos <i>jumpers</i> do kit de desenvolvimento.....	34
Tabela 4.1 – Tabela com valores das plantas controladas.....	52

# 1. INTRODUÇÃO

Nas indústrias existe a necessidade de controlar sistemas de pressão, nível, velocidade, entre outros, para isso, na maioria dos casos, é utilizado um controlador Proporcional – Integral – Derivativo (PID).

O projeto desses controladores é classicamente realizado com a resposta de um modelo de referência de segunda ordem e, a partir do conhecimento (não necessariamente) da planta que deverá ser controlada, encontra-se os parâmetros de ganho proporcional, integral e derivativo do controlador PID. Este controlador, utilizado em cascata com esta planta, em malha fechada, possibilita que a mesma acompanhe a resposta do modelo da melhor forma possível [1].

Atualmente a maioria dos controladores industriais utilizam técnicas de controle, como o controle On-Off ou PID. Os controladores que usam como técnica o controle PID normalmente utilizam sintonia automática. Porém o método utilizado para implementar a sintonia automática não é fornecida pelo fabricante aos seus consumidores, isso pode ser comprovado ao pesquisar informações sobre controladores nas empresas que o fabricam.

O objetivo principal deste trabalho é desenvolver um controlador PID microprocessado com sintonia automática para sistemas de segunda ordem utilizando o método de Ziegler-Nichols. O processador escolhido para implementar o *software* deste trabalho é o LPC 2136 de núcleo ARM7 - 32 Bits.

No capítulo 2 trata-se do conceito de PID, tanto analógico como digital, e a explicação do método de Ziegler-Nichols. O capítulo 3 refere-se ao *hardware* implementado (microprocessador e plantas utilizadas). O *software* implementado para

execução deste PID e os resultados práticos obtidos é mostrado no capítulo 4. O capítulo 5 mostra as considerações finais e as propostas de melhorias que podem ser feitas neste projeto.

## 2. REFERENCIAL TEÓRICO

### 2.1. Controladores PID industriais

#### 2.1.1. Conceitos Gerais

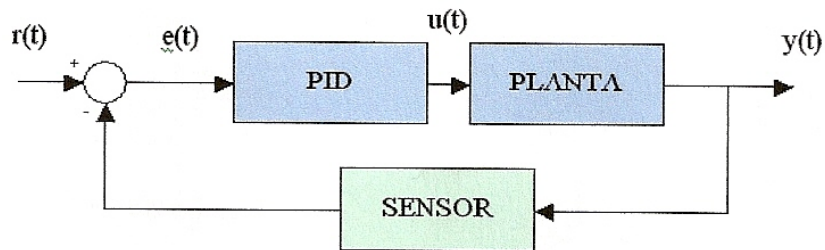
Os controladores PID (Proporcional - Integral - Derivativa) são bastante utilizados nas indústrias para controle de sistemas em geral, como dito anteriormente. De fato, na indústria, 95% das malhas de controle utilizam controladores assim, em sua maioria apenas a parte proporcional e integral são utilizadas [2].

Em geral as empresas que fabricam controladores utilizam o método On-Off ou PID, sendo que quando implementados com PID, normalmente o método On-Off também pode ser habilitado. Alguns modelos de controladores está apresentado na tabela 2.1.

Fabricante	Modelo de Controlador	On-Off	PID com Sintonia Automática	PD	P
NOVUS	N440	SIM	NÃO	NÃO	NÃO
	N480D	SIM	SIM	NÃO	NÃO
	N960	SIM	SIM	NÃO	NÃO
	N1100	SIM	SIM	NÃO	NÃO
SIEMENS	3RS01	SIM	NÃO	SIM	NÃO
	3RS02 01-1BF00	SIM	NÃO	NÃO	SIM
	3RS02 10-1BF00	SIM	SIM	NÃO	NÃO

**Tabela 2.1** – Controladores comerciais [3], [4]

O controlador PID é muito utilizado em controle de processos em função de ser facilmente programável, necessitar de poucos parâmetros de ajuste e pouco suporte de recursos para operação. Este controlador possui um conjunto de equações que são aplicadas a fim de se produzir ações de controle através de um atuador sobre um processo. A figura 2.1 apresenta o diagrama em blocos de um sistema de controle [2].



**Figura 2.1** – Diagrama em blocos de um sistema de controle

Uma versão clássica do controlador PID pode ser apresentada como na equação 2.1.

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

(Equação 2.1)

Onde “**e**” representa o erro entre a variável medida e a variável controlada e “**u**” é a variável de controle que é a soma dos termos: proporcional, integral do erro e derivada do erro. Os parâmetros de controle considerados para este tipo de controlador são o ganho do controlador  $K_p$ , a constante de tempo integrativa  $T_i$  e a constante de tempo derivativa  $T_d$ .

Esta equação correlaciona os termos os quais executam as ditas ações de controle, proporcional, integral e derivativa.



### 2.1.2. Ação Proporcional

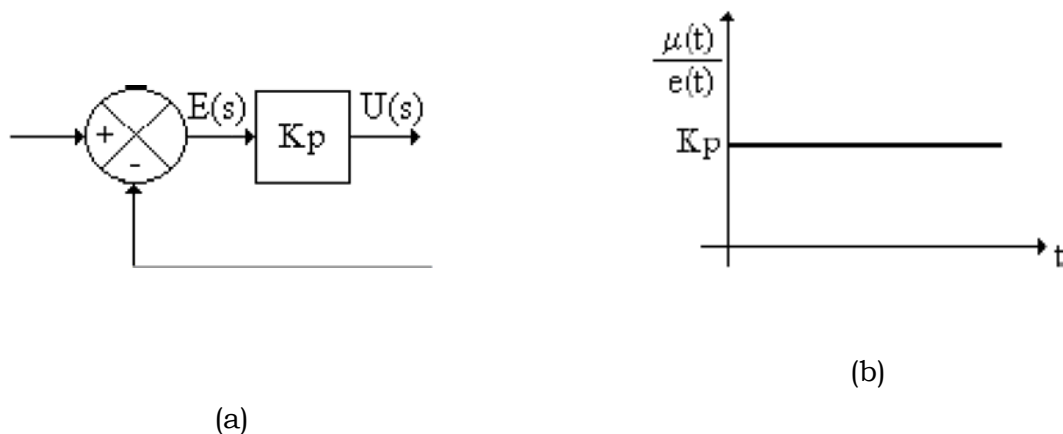
A ação proporcional é representada pelo termo  $K_p \cdot e(t)$  que implementa a ação básica de realimentação (negativa) proporcional ao erro (Figura 2.2). Desta forma, só é ativa enquanto o erro for diferente de zero. Consequentemente, para reduzir o erro de regime é preciso um valor alto do ganho  $K_p$ . Entretanto, na prática não é possível aumentar arbitrariamente o ganho tendo em vista a saturação dos atuadores, a amplificação do ruído e a estabilidade do sistema em malha fechada.

$$u(t) = K_p \cdot e(t)$$

(Equação 2.2)

$$K_p = \frac{U(s)}{E(s)}$$

(Equação 2.3)



**Figura 2.2** – (a)Diagrama em bloco da ação proporcional [5];  
(b)Gráfico da atuação que a ação proporcional gera, para um erro do tipo salto.

### 2.1.3. Ação Integral

Para o caso em que se quer rastrear um sinal tipo degrau, a função principal da ação integral é assegurar que o erro estacionário seja sempre zero (Figura 2.4). Com este termo, um valor de  $e$  diferente de zero, mesmo sendo muito pequeno, sempre

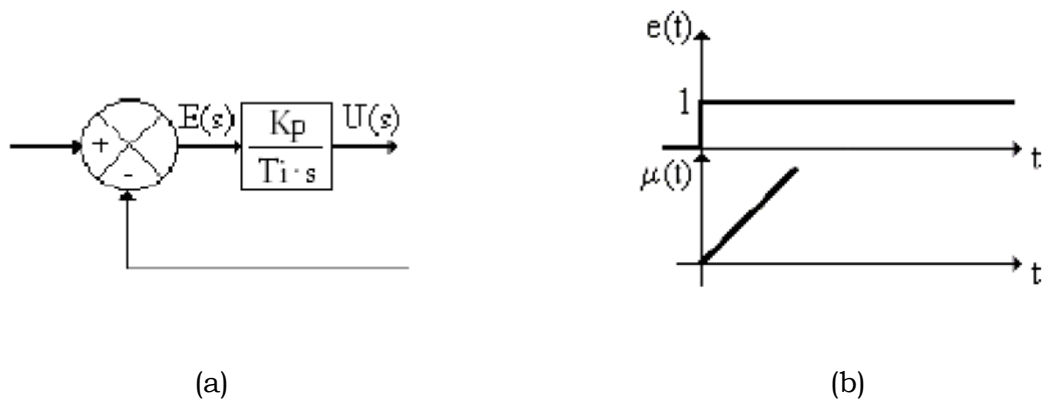
levará a uma mudança na variável de controle  $u$  que se manterá constante se o erro for nulo. Assim, se o sistema for estável, o erro em regime a um comando em degrau será necessariamente zero.

$$\frac{du(t)}{dt} = \frac{K_p}{T_i} \cdot \int_0^t e(t) \cdot dt$$

(Equação 2.4)

$$\frac{U(s)}{E(s)} = \frac{K_p}{T_i \cdot s}$$

(Equação 2.5)



**Figura 2.3** – (a)Diagrama em bloco da ação integral [5];  
(b)Gráfico da atuação que a ação integral gera, para um erro do tipo salto

#### 2.1.4. Ação Derivativa

O propósito usual da ação derivativa é melhorar a estabilidade transitória do sistema em malha fechada, agindo como um termo de amortecimento. A derivada do erro pode ser interpretada como uma predição do erro no tempo  $(t + T_d)$  observado no tempo  $t$ , como pode-se notar na expansão em série de Taylor de  $(e+T_d)$  mostrado na equação 2.6.

$$e(t + Td) = e(t) + Td \frac{de(t)}{dt} + \dots$$

(Equação 2.6)

### 2.1.5. Escolha das Ações de Controle

Embora a escolha das ações do controlador PID pareça ser trivial, na análise das malhas de controle industrial é encontrada uma série de equívocos que acabam diminuindo o desempenho do sistema de controle. Para reduzir este problema, a seguir serão apresentadas algumas estratégias relacionadas com os seguintes tipos de controladores: P, PI e PID, que são classificados conforme as ações de controle que eles possuem.

### 2.1.6. Controlador P (somente ação proporcional)

O uso da ação proporcional (somente) não consegue eliminar o *offset* permanente, devendo ser usado, portanto, apenas quando este inconveniente não ocasiona maiores problemas podendo ser tolerado. As vantagens são que a ação proporcional normalmente não produz *overshoot* e nem adiciona oscilações ao sistema, e além disso, possui apenas um parâmetro de ajuste (geralmente o mais fácil de se ajustar). Pode-se então elevar o ganho do controlador de modo que se reduza o *offset*, porém de tal forma que este ganho não passe a ampliar excessivamente o ruído de medição.

### 2.1.7. Controlador PI (ações proporcional e integral)

A inclusão da ação integral no controlador consegue eliminar o *offset* permanente, porém, torna a resposta mais oscilatória podendo até levar o processo à instabilidade, como no caso do uso excessivo do ganho proporcional ou da ação integral. A sintonia do controlador PI é uma tarefa um pouco mais complexa, pois existem dois parâmetros de ajuste ( $K_p$  e  $T_i$ ) ao invés de um ( $K_p$ ). Contudo, este tipo de controlador quando bem sintonizado, geralmente é capaz de prover um desempenho adequado para a grande maioria dos processos. De qualquer forma, o ajuste de um

controlador PI continua sendo mais simples que o ajuste de um controlador PID, e por este motivo é o mais utilizado em processos industriais.

### **2.1.8. Controlador PID (ações proporcional, integral e derivativa)**

A inclusão da ação derivativa no controlador tende a reduzir o caráter oscilatório da resposta. Contudo, ela possui duas desvantagens que diminuem o seu uso. A primeira está relacionada com a maior dificuldade para sintonizar o controlador, já que ele conta com três parâmetros ( $K_p$ ,  $T_i$  e  $T_d$ ) e geralmente, o parâmetro relacionado com a ação derivativa ( $T_d$ ) é o mais difícil de ajustar. A segunda está relacionada com a sensibilidade à presença de ruído, que pode ser excessivamente acentuado pela ação derivativa, podendo até instabilizar o processo. Devido a estes problemas, o uso da ação derivativa geralmente só é aconselhado em malhas que apresentem dinâmicas lentas (com uma inércia inicial elevada), como malhas de temperatura.

## **2.2. Controle Digital**

O controle digital caracteriza-se pelo uso de um computador, microcontrolador, ou microprocessador, que gera a lei de controle e exerce a função de controlador. Controladores digitais são flexíveis e as funções de controle podem ser facilmente modificadas. Leis de controle mais complexas também podem ser implementadas sem dificuldade. O diagrama em blocos do sistema de controle é mostrado na Figura 2.4.



**Figura 2.4** – Diagramas em blocos do controlador digital

Em um controle digital o sinal de saída é amostrado e convertido em uma sequência de pulsos expressos em um código numérico (código binário, por exemplo). A função de transferência do controlador é convertida em uma equação diferença implementada como um programa no computador. A saída do computador por sua vez,

que é expressa também no mesmo código binário, é convertida para um sinal contínuo. Esta saída é o sinal de atuação.

Sistemas de controle amostrados são usados quando um elevado grau de precisão é requerido. As vantagens com relação ao controle analógico são:

1. Implementação simples de controles complexos.
2. Flexibilidade no caso de mudança de leis de controle.
3. Superioridade com relação a controladores analógicos do ponto de vista de ruídos internos e efeitos de *drift*.

Por outro lado algumas desvantagens também se apresentam:

1. Erros são introduzidos pelos processos de amostragem e quantização, e podem degradar o desempenho do sistema.
2. O projeto pode se tornar mais complexo para compensar esta degradação.

Para converter a expressão do controlador PID no domínio tempo (equação vista no capítulo 2.1.1) para uma expressão no domínio discreto (forma digital) equivalente, deve-se seguir a aproximação por diferenças finitas, utilizando a equação 2.7, para o termo integrador e, a equação 2.8 para o termo derivativo.

$$\int_0^t e(t') dt' \approx \sum_{k=1}^n e_k \cdot \Delta t$$

(Equação 2.7)

$$\frac{de}{dt} \approx \frac{e_n - e_{n-1}}{\Delta t}$$

(Equação 2.8)

A forma digital do controlador PID pode ser descrita de duas formas: a forma da posição e da velocidade. Substituindo as aproximações por diferenças finitas na

expressão geral do controlador PID no domínio tempo, obtém-se a equação 2.9 na forma de posição.

$$u(t) = \bar{u} + Kp \cdot \left[ 1 + \frac{\Delta t}{Ti} \cdot \sum_{k=1}^n e_k + \frac{Td}{\Delta t} \cdot (e_n - e_{n-1}) \right]$$

(Equação 2.9)

Pode-se escrever esta expressão em termos da Transformada z [6]. A equação 2.10 representa o controlador PID digital em termos da transformada z conforme o algoritmo da posição, neste caso ele fornece o valor de saída do controlador diretamente.

$$\frac{U(z)}{E(z)} = Kp \cdot \left[ 1 + \frac{\Delta t}{Ti} \cdot \left( \frac{1}{1 - z^{-1}} \right) + \frac{Td}{\Delta t} \cdot (1 - z^{-1}) \right]$$

(Equação 2.10)

A forma de velocidade é uma alternativa de boa aplicação prática pois não é necessário selecionar o termo  $\bar{u}$ , nem de se comutar o somatório do erro. Desde que o estado estável nominal é constante, a mudança na saída do controlador  $\Delta p_n$  é dada por:  $\Delta p_n = p_n - p_{n-1} = p_n' - p_{n-1}'$ . Isolando  $p_{n-1}$  e por rearranjo matemático com a expressão da posição obtém-se a equação 2.11.

$$\Delta p_n = Kp \cdot \left[ (e_n - e_{n-1}) + \frac{\Delta t}{Ti} \cdot e_n + \frac{Td}{\Delta t} \cdot (e_n - 2 \cdot e_{n-1} + e_{n-2}) \right]$$

(Equação 2.11)

Substituindo  $\Delta t$  por  $T$ , onde  $T$  é o tempo de amostragem, pode-se escrever a expressão anterior na forma de transformada z como mostra na equação 2.12.

$$\frac{C(z)}{E(z)} = Kp + \frac{Kp}{Ti} \cdot \frac{T(z+1)}{2 \cdot (z-1)} + Kp \cdot Td \cdot \left(\frac{z-1}{Tz}\right)$$

(Equação 2.12)

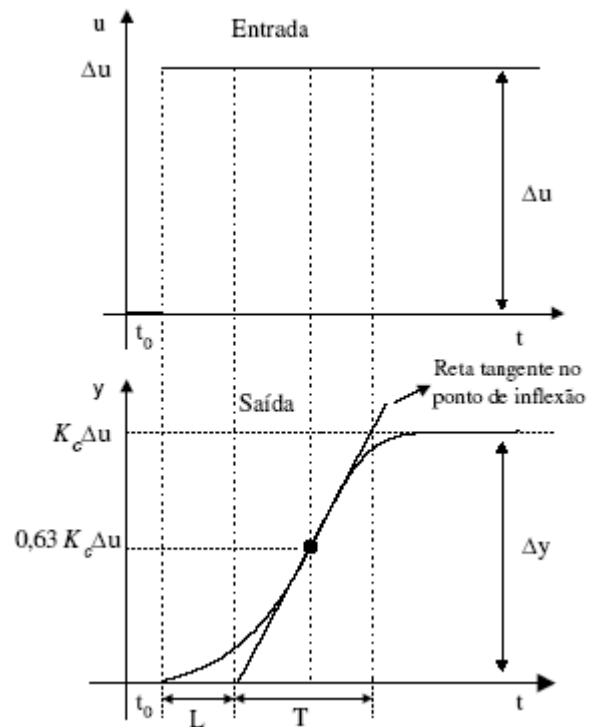
### **2.3. Métodos de Ziegler-Nichols**

Os métodos de Ziegler-Nichols foram introduzidos já em 1942 e hoje são considerados clássicos [7]. Estes métodos continuam a ser largamente aplicados até hoje, mesmo em sua forma original, mas normalmente são utilizados em alguma forma modificada. Os dois métodos básicos de ajuste de Ziegler-Nichols visam obter uma mesma resposta pré-especificada para o sistema em malha fechada, e diferem no que diz respeito à natureza da informação sobre a dinâmica do processo que é exigida por cada um deles.

O método da resposta ao salto, ou método do domínio do tempo, requer o conhecimento de duas grandezas que caracterizam a resposta ao salto de um processo. Já o método do período crítico, exige o conhecimento de duas grandezas características da resposta em frequência do processo. Uma vez obtidas estas informações, basta recorrer as equações de cada método (mais detalhes destas equações estão apresentadas nos itens 2.3.1 e 2.3.2) para calcular os ganhos do controlador. Estas equações foram determinadas de maneira empírica por meio de ensaios de processos industriais típicos. As equações originalmente propostas por Ziegler e Nichols fornecem uma resposta que foi posteriormente considerada insatisfatória. Diferentes equações foram então propostas com base nos mesmos ensaios, obtendo-se melhor desempenho.

#### **2.3.1. Método da Resposta ao Salto**

O método da resposta ao salto consiste de equações derivadas a partir dos parâmetros da curva de reação [7]. Ela é obtida do sistema, quando o mesmo está em malha aberta. Podemos ver na figura 2.5a, a excitação do tipo salto e na figura 2.5b a resposta do sistema e como são obtidos os parâmetros L e T.



**Figura 2.5** – Características da resposta ao salto do processo relevantes para o ajuste de Ziegler-Nichols

No entanto, sua aplicação é mais indicada para curva de reação na forma de um “S” e que atenda o seguinte critério:  $0,1 < \frac{L}{T} < 1$ . Para esta curva, os parâmetros, tempo de retardo  $L$ , e constante de tempo  $T$ , são determinados passando-se uma reta tangente no ponto de inflexão da curva, como ilustrado na figura 2.5.

Tendo-se obtido experimentalmente os valores de  $L$  e  $T$ , pode-se recorrer à tabela 2.2 para determinar os valores dos parâmetros do controlador PID.



Tipo de controlador	Kp	Ti	Td
P	$\frac{T}{K_c \cdot L}$	$\infty$	0
PI	$\frac{0,9 \cdot T}{K_c \cdot L}$	$3,33 \cdot L$	0
PID	$\frac{1,2 \cdot T}{K_c \cdot L}$	$2 \cdot L$	$\frac{L}{2}$

**Tabela 2.2** – Tabela de Ziegler e Nichols pelo método da resposta ao Salto [5].

Os valores da tabela 2.1 foram determinados de forma empírica com objetivo de obter uma resposta com amortecimento de 1/4 na resposta à referência para processos industriais típicos (um amortecimento de 1/4 leva a um valor máximo de *overshoot* de 25%). Enquanto a rejeição a perturbações muitas vezes apresenta um comportamento satisfatório, este amortecimento usualmente não é satisfatório na resposta à referência, causando em muitos casos um *overshoot* excessivo e baixa tolerância a variações na dinâmica do processo.

Este método limita-se a plantas de primeira ordem. Sistemas de segunda ordem, ou superior, por exemplo, não se enquadram nesta categoria. Por outro lado, este método baseia-se em identificação de formas de onda, o que pode ser problemático na prática, particularmente em aplicações com baixa relação sinal-ruído. Ainda assim, o método é adequado para um grande número de processos industriais.

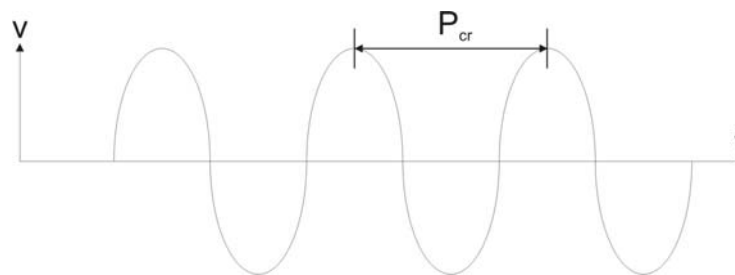
### 2.3.2. Método do Período Crítico

Se um processo é colocado em laço fechado com controle proporcional e o valor do ganho proporcional é aumentado progressivamente, a certa altura o processo iniciará a oscilar. O ganho necessário para causar esta oscilação é chamado ganho crítico (Kcr) do processo e o período da oscilação observada é dito o seu período crítico (Pcr) [5].

Para aplicação deste método primeiramente deve-se ajustar os valores de Ti igual a infinito e Td igual a zero. Utilizando somente a ação proporcional, aumenta-se o

valor de  $K_p$  de zero a um valor crítico para o qual o sinal de saída apresente oscilações mantidas (se o sinal de saída não apresenta oscilações, quaisquer que sejam os valores de  $K_p$ , então este método não pode ser aplicado) [5].

Em consequência são determinados experimentalmente os valores de ganho crítico e o período crítico correspondente (figura 2.6). De posse do ganho crítico e do período crítico basta aplicar as fórmulas propostas. A Tabela 2.3 apresenta as fórmulas originalmente apresentadas por Ziegler e Nichols quando da proposta do método.



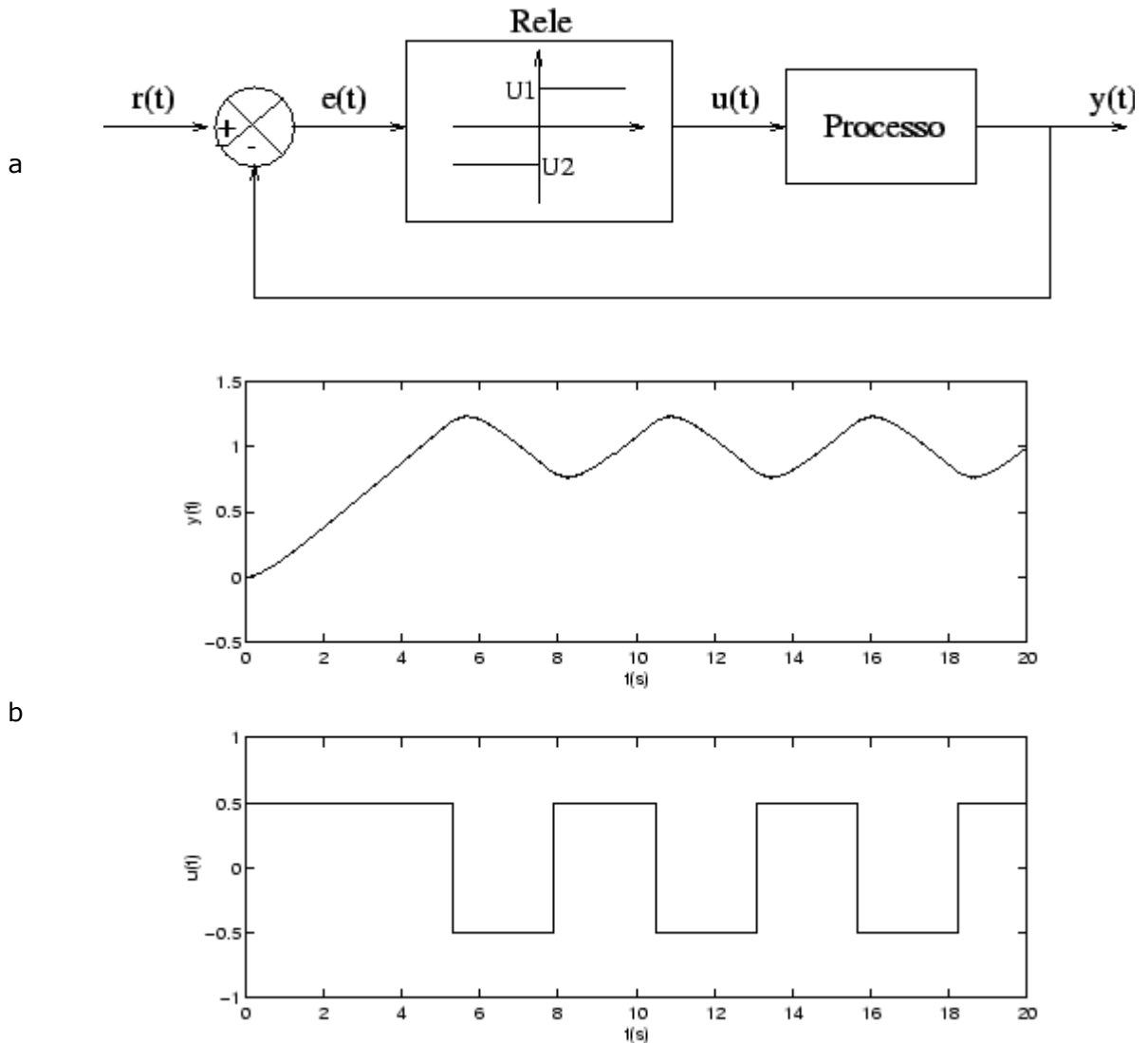
**Figura 2.6** – Oscilações estáveis com períodos  $P_{cr}$

Tipo de controlador	$K_p$	$T_i$	$T_d$
P	$0,5 \cdot K_{cr}$	$\infty$	0
PI	$0,45 \cdot K_{cr}$	$\frac{1}{1,2} \cdot P_{cr}$	0
PID	$0,6 \cdot K_{cr}$	$0,5 \cdot P_{cr}$	$0,125 \cdot P_{cr}$

**Tabela 2.3** – Fórmulas de Ziegler e Nichols para ajuste pelo método do período Crítico [5].

O procedimento para obtenção do ganho crítico e do período crítico visto até o momento é pouco eficiente por diversos motivos. Primeiramente, uma vez que o ganho deve ser aumentado de forma gradativa, o procedimento torna-se bastante demorado. Em segundo lugar, é preciso ter de antemão alguma informação sobre a dinâmica do processo a fim de determinar o valor inicial do ganho e sua taxa de variação. Finalmente, a natureza linear da oscilação faz com que ela nunca seja sustentada, mas sempre amortecida ou instável.

Uma maneira muito mais eficiente de determinar estes parâmetros é o ensaio de realimentação com relé, que não sofre de nenhum dos problemas citados acima. Imagine que o processo esteja em laço fechado com um relé na realimentação, como na Figura 2.7a. A figura 2.7b mostra a curva típica deste sistema [7].



**Figura 2.7** – (a)Diagrama em bloco do controle com relé [7];  
(b)Gráfico da resposta típica do controle de realimentação com relé.

Em um processo de controle com relé a variável controlada oscila em torno do valor de referência e a variável manipulada chaveia periodicamente entre seus dois valores possíveis. As características desta oscilação dependem das características do

processo, mas também dos valores de  $u_1$  e  $u_2$  escolhidos para o controle de realimentação com relé [8].

Uma primeira característica da oscilação a ser analisada é sua simetria. Por simetria entende-se entre os tempos em que a variável manipulada fica em seu valor máximo e mínimo: a oscilação da variável controlada será simétrica com relação ao valor de referência. Se essa oscilação é simétrica, então seu período é igual ao período crítico do processo e o ganho crítico pode ser calculado pela equação 2.13.

$$K_{cr} = \frac{4 \times d}{\pi \times A}$$

(Equação 2.13)

Onde:

- $k_{cr}$  = ganho crítico;
- $d = u_1 - u_2$  (Figura 2.7);
- $A$  = amplitude pico-a-pico da oscilação observada na variável controlada.

## 3. IMPLEMENTAÇÃO DO *HARDWARE*

### 3.1. O *Microcontrolador LPC 2136*

#### 3.1.1. Apresentação do núcleo ARM7

Uma nova tendência no mercado são produtos com baixo consumo de energia e alto poder de processamento, uma qualidade desejável em todos os equipamentos eletrônicos. Para atingir este objetivo surgiram os microprocessadores ARM, em que ARM significa *Advanced RISC Machine* (máquina de RISC avançada) e RISC significa *Reduced Instruction Set Computer* (computador com set de instruções reduzido) [9].

Os microprocessadores ARM de 32 bits que são usados em celulares, PDA's, videogames portáteis, etc. O sucesso desses produtos se deve ao fato de o ARM ser uma nova tendência de mercado e pode ser mais explorado. O núcleo (core) ARM7 está disponível em vários microprocessadores de diversos fabricantes como a Philips, Analog Devices, OKI, dentre outros.

As características principais do processador ARM7TDMI-S são:

- **Modo Thumb:** o modo Thumb do ARM7TDMI-S nada mais é do que um segundo *set* de instruções com 16 bits e ajuda a economizar memória de programa.
- **Multiplicação longa:** a utilização de um *hardware* dedicado para multiplicação longa possibilita ao ARM7TDMI-S realizar operações mais complexas normalmente feitas por um DSP. Com esse *hardware* é possível realizar multiplicações de 32 bits por 32 bits, apresentando o resultado em 64 bits. Ainda esse módulo é capaz de realizar

multiplicação-acumulação (MAC) de 32 bits por 32 bits com resultado de 64 bits.

- **Depuração:** uma região do ARM7TDMI-S possui uma extensão de *hardware* capaz de realizar uma depuração dentro da aplicação através de porta JTAG.
- **Embedded ICE:** é uma extensão das funções de depuração. Esse módulo estende as funções de breakpoint (pontos de parada), visualização de registros e outros pontos do programa, o que torna muito fácil o trabalho de depuração. Esse módulo é acessado pela porta JTAG.
- **Alta capacidade de processamento:** o núcleo ARM7TDMI-S tem alta capacidade de processamento.

A família LPC 213X da Philips é baseada no núcleo ARM7TDMI-S com suporte à emulação (usando a porta J-TAG) aliada a uma memória de programa Flash de alta velocidade e uma interface de alta velocidade permite a execução a 60 MHz. Para aplicações em que o tamanho do programa é importante, podemos contar também com o modo Thumb, com o qual podemos enxugar 30% do espaço da memória de programa.

Aliada a pequenas dimensões, consumo e desempenho, é uma solução ideal para diversas aplicações em que estas características citadas são importantes. O LPC 213X pode ser utilizado para diversas aplicações, como por exemplo: equipamentos médicos, controladores industriais em geral, controle de acesso, conectividade, aplicações de uso gerais, etc.

### 3.1.2. Características do LPC 2136

As principais características do LPC 2136 são apresentadas em seguida:

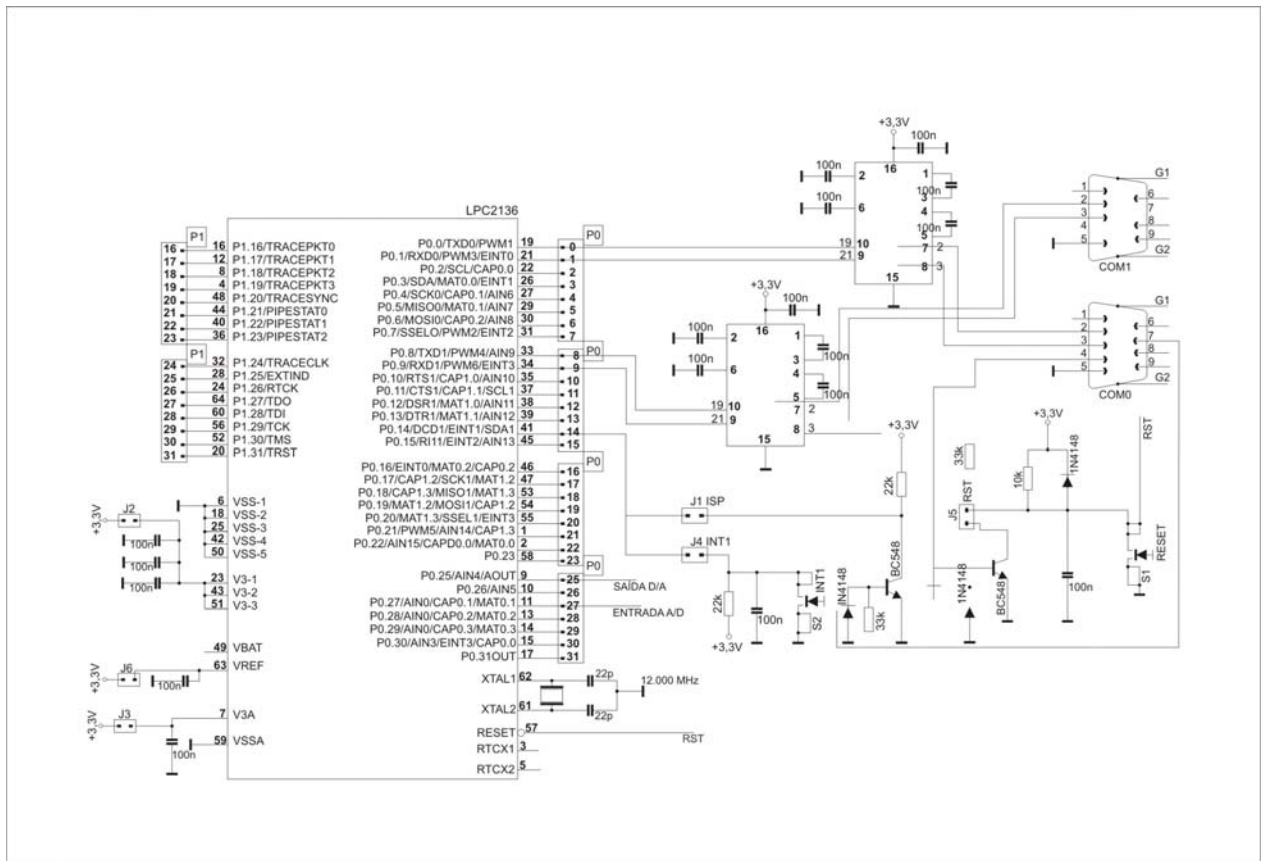
- Núcleo ARM7TDMI-S;
- Alimentação de 3,0V a 3,6V;

- 512 KBytes de memória de programa;
- 32 KBytes de memória de dados volátil (RAM);
- 22 interrupções (4 externas);
- 47 I/O's toleráveis a 5V;
- Dois *timers* ou contadores de eventos externos de 32 bits (com 4 canais captures e compares cada);
- Unidade de PWM com seis saídas;
- *Watchdog Timer*;
- Duas UART's, sendo uma UART1 com todos os pinos de controle implementados por *hardware* (CTS, DCD, DSR, RI e RTS);
- Dois barramentos I<sup>2</sup>C's;
- Um barramento SPI;
- Um módulo SSP (SPI, 4-wire ou Microware);
- RTC interno;
- Dois conversores A/D de 10 bits com 8 canais cada;
- Um canal de conversão D/A de 10 bits;
- Opera com cristal de 1 MHz até 30 MHz ou oscilador externo de 1MHz até 50 MHz;
- 60 MHz de operação máxima via PLL interno.

A pinagem do LPC 2136 e suas funções estão descritas no anexo A.

### 3.2. Hardware Básico de Trabalho

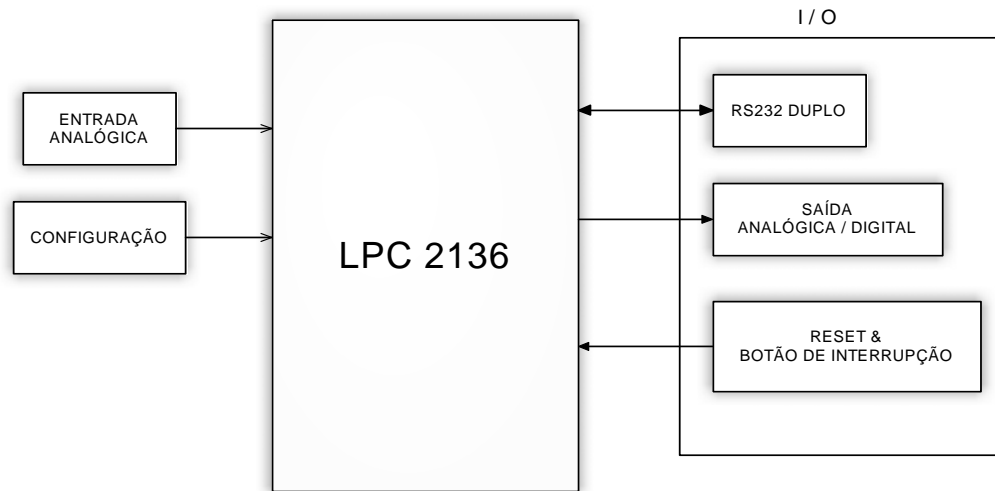
Baseado no kit de desenvolvimento da empresa Keil (o esquema elétrico pode ser visto no anexo B [10]) foi desenvolvido um *hardware*, para a utilização do LPC 2136, utilizando componentes mais fáceis de se encontrar no mercado. O esquema elétrico deste *hardware* está apresentado na figura 3.1.



**Figura 3.1** – Esquema elétrico do *hardware*

O diagrama de blocos simplificado do *hardware* utilizado para usar o microcontrolador é mostrado na figura 3.2.



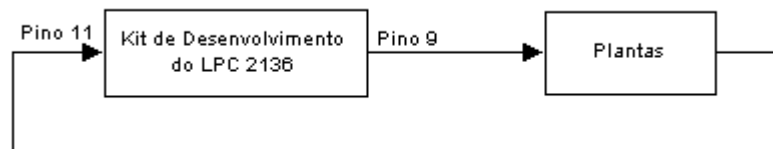


**Figura 3.2** – Diagrama em blocos do kit de desenvolvimento

No kit há duas chaves liga-desliga denominados de *reset* e de INT1. A chave de *reset* reseta o microcontrolador, e a chave INT1 habilita a interrupção externa 1 (EINT1) do microcontrolador.

O modo que o kit de desenvolvimento do trabalho é implementado junto com as plantas está apresentado no diagrama em blocos da figura 3.3.

Existe também neste kit seis *jumpers* na quais suas funções estão explicadas tabela 3.1.

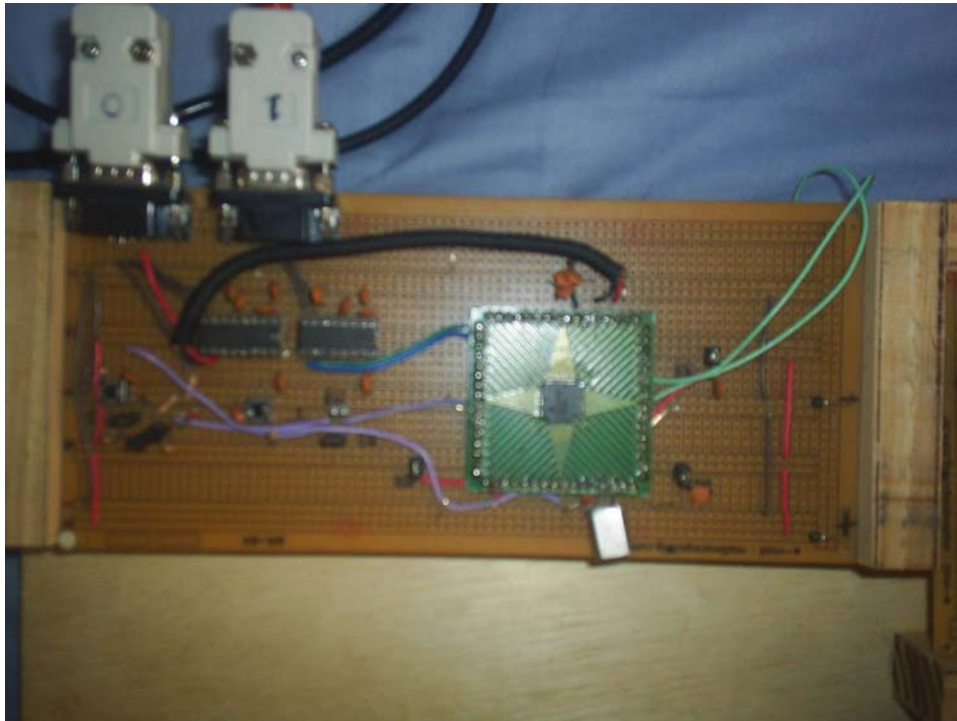


**Figura 3.3** – Diagrama em blocos do kit de desenvolvimento ligado na planta.

<i>Jumper</i>	Função
J1 - ISP	Habilita o sistema de programação. Permite que o sinal de RTS da porta COM0 gere um sinal no pino P0.14 (EINT1). Quando o RTS é ativado, o pino P0.14 é levado a nível baixo, entrando assim, no modo ISP ( <i>In-System Programming</i> ).
J2 - 3,3 V	Fornece alimentação de 3,3 Volts ao microcontrolador
J3 - V3A	Alimenta o pino de referência da entrada analógica (V3A)
J4 - INT1	Habilita a chave INT1
J5 - RST	Habilita o <i>reset</i> pela porta COM0
J6 - Vref	Alimenta o pino de referência do conversor A/D

**Tabela 3.1** – Funções dos *jumpers* do kit de desenvolvimento

Uma foto do kit montado é apresentada na figura 3.4.

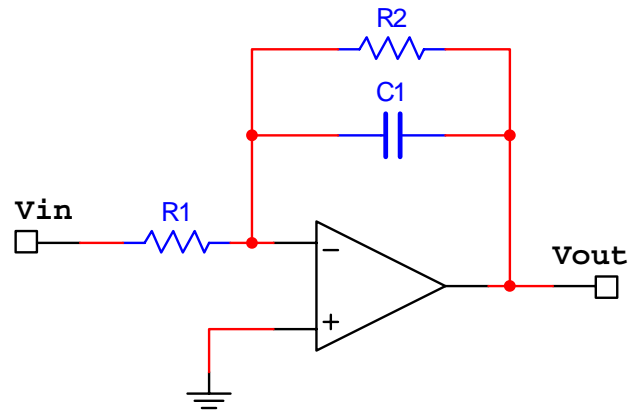


**Figura 3.4** – Foto do kit montado

### 3.3. Plantas

Para testar o controle PID proposto, foi construída uma planta eletrônica de segunda ordem, com três fatores de amortecimento distintos.

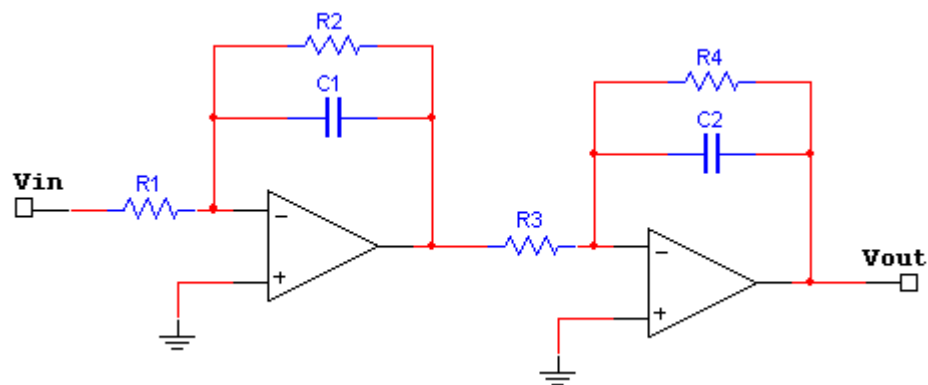
Como base do circuito foi utilizado dois amplificadores operacionais, na forma de integrador. A figura 3.5 mostra o circuito integrador, e sua função no plano S está apresentada na equação 3.1. Dois circuitos integrador (filtro passa-baixa) em série são apresentados na figura 3.6, e sua respectiva função no plano S na equação 3.2.



**Figura 3.5** – Circuito operacional na forma de filtro passa-baixa

$$G(s) = \frac{V_{out}}{V_{in}} = -\frac{R2}{R1} \cdot \frac{1}{R2.C1.s + 1}$$

(Equação 3.1)



**Figura 3.6** – Dois circuitos integrador em série

$$G(s) = \frac{\frac{1}{R1.R3.C1.C2}}{s^2 + \left( \frac{R2.C1 + R4.C2}{R2.R4.C1.C2} \right).s + \frac{1}{R2.R4.C1.C2}}$$

(Equação 3.2)

Sabendo que a equação geral de segunda ordem é descrita como:

$$G(s) = \frac{K.wn^2}{s^2 + 2.\xi.Wn.s + wn^2}$$

(Equação 3.3)

Onde:

wn - frequência de corte

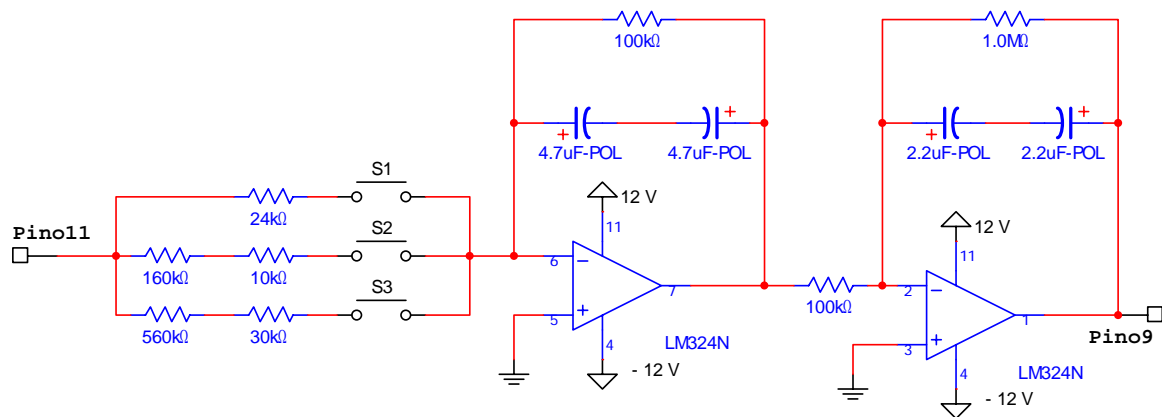
$\xi$  - Fator de amortecimento

Este circuito é baseado no circuito utilizado na disciplina de Controle Analógico. Sabemos que o circuito estudado da disciplina é da ordem de 1 à 3 segundos. Então, arbitrando  $R2 = 100 \text{ K}\Omega$ ,  $R3 = 100 \text{ K}\Omega$ ,  $R4 = 1 \text{ M}\Omega$ ,  $C1 = 2,35 \text{ }\mu\text{F}$  e  $C2 = 1,1 \text{ }\mu\text{F}$ , do circuito da figura 3.3, igualando os coeficientes a equação geral de segunda ordem e após escolhendo os valores requeridos para o fator de amortecimento, podemos então, definir o valor de  $R1$  através da equação 3.4.

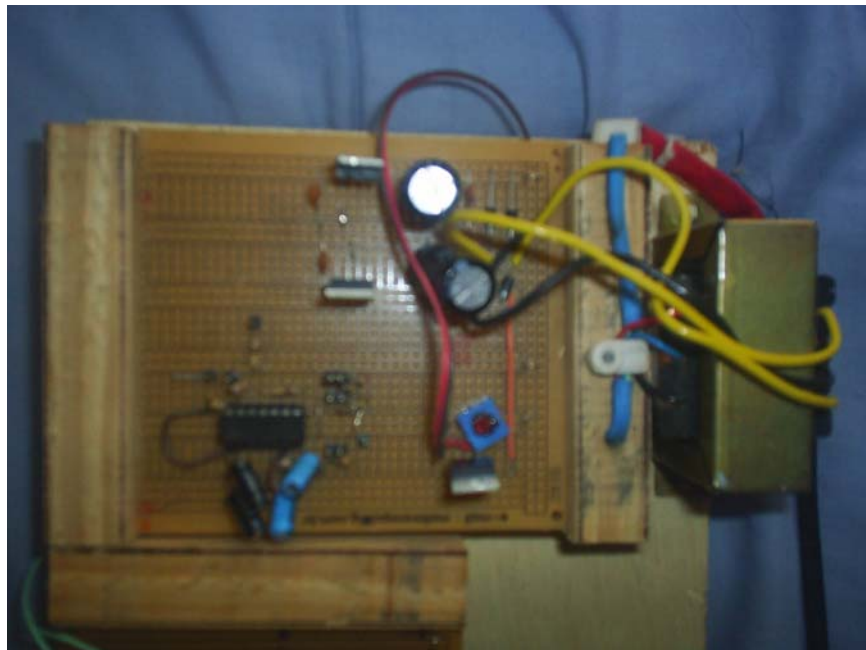
$$R1 = \frac{1 \times 10^6 . \xi^2}{1,7236 - \xi^2}$$

(Equação 3.4)

Utilizando fatores de amortecimento iguais a 0,2; 0,5 e 0,8 os valores de  $R1$  são respectivamente  $24 \text{ K}\Omega$ ,  $170 \text{ K}\Omega$ ,  $590 \text{ K}\Omega$ .



**Figura 3.7** – Planta utilizada para teste do PID



**Figura 3.8** – Foto das plantas montadas

As equações do circuito da figura 3.7 são:

Para  $\xi = 0,2$ :

$$G(s) = \frac{161,1863}{s^2 + 5,1644.s + 3,8685}$$

(Equação 3.4)

Para  $\xi = 0,5$ :

$$G(s) = \frac{22,7557}{s^2 + 5,1644.s + 3,8685}$$

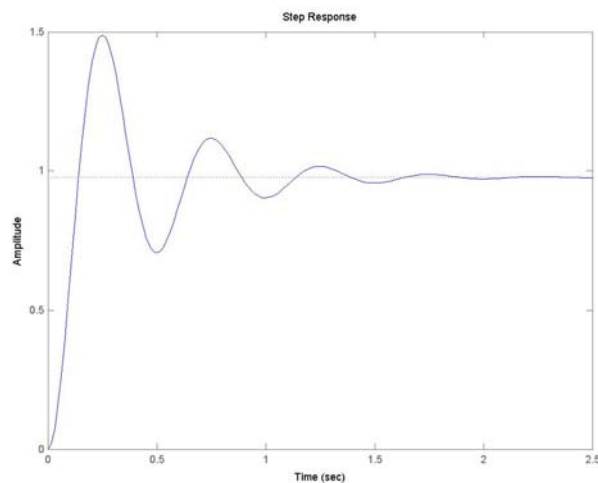
(Equação 3.4)

Para  $\xi = 0,8$ :

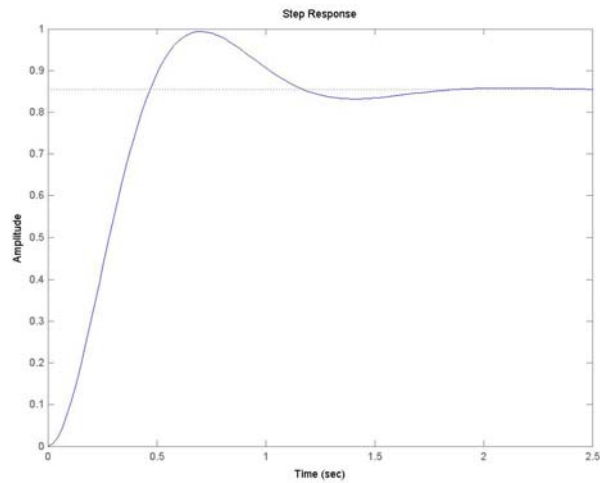
$$G(s) = \frac{6,5567}{s^2 + 5,1644.s + 3,8685}$$

(Equação 3.4)

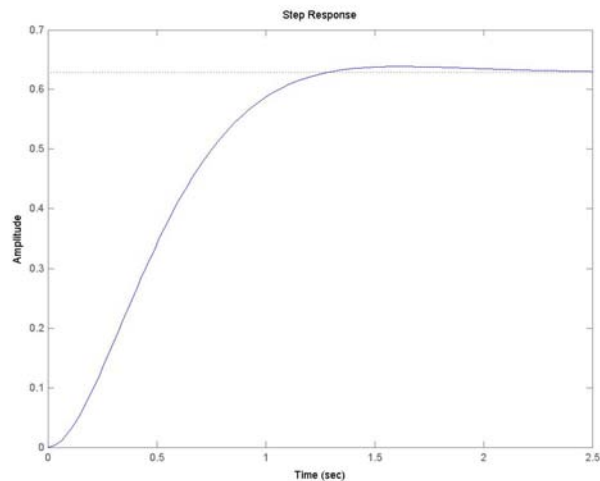
Os gráficos da simulação feito no programa MatLab® para as plantas em malha fechada com entradas do tipo salto são mostrados na figura 3.9, figura 3.10 e figura 3.11.



**Figura 3.9** – Gráfico da equação com  $\xi = 0,2$  para entrada do tipo salto unitário



**Figura 3.10** – Gráfico da equação com  $\xi = 0,5$  para entrada do tipo salto unitário

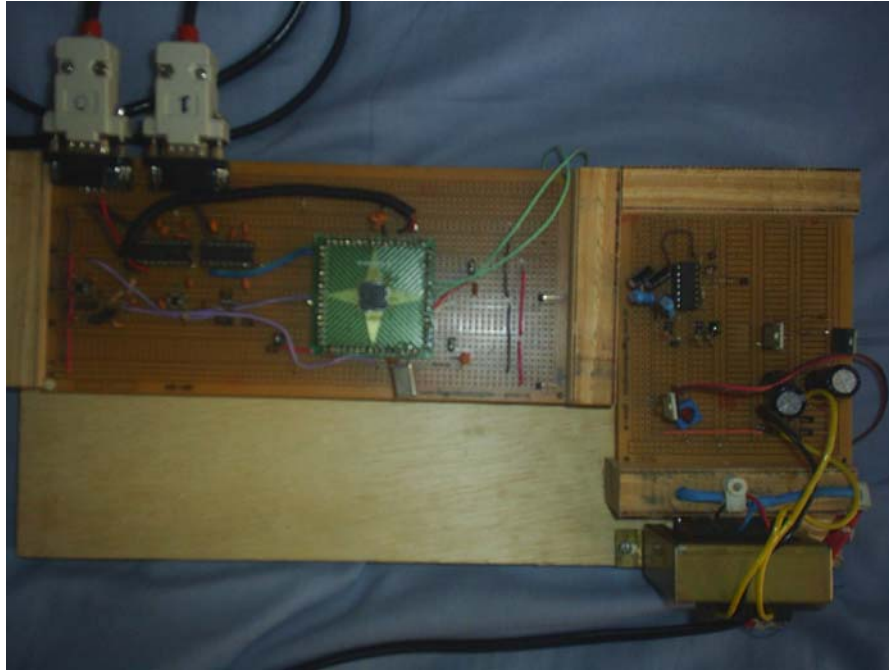


**Figura 3.11** – Gráfico da equação com  $\xi = 0,8$  para entrada do tipo salto unitário

Na figura 3.12 está mostrado a foto da planta e do *hardware* do microprocessador montados, conforme descrito na figura 3.3 e na figura 3.7.

As plantas foram também testadas sem o uso do microcontrolador, para este teste foi usado realimentação do tipo unitária utilizando um amplificador subtrator. A resposta para as plantas de  $\xi = 0,5$  e  $\xi = 0,8$  são essencialmente as mesmas obtidas

por simulação. A planta de  $\xi = 0,2$  apresenta um ganho elevado a qual leva a planta a ter um valor inicial (*offset*) de 250 mV.



**Figura 3.12** – Foto da planta e do *hardware* do microcontrolador montado

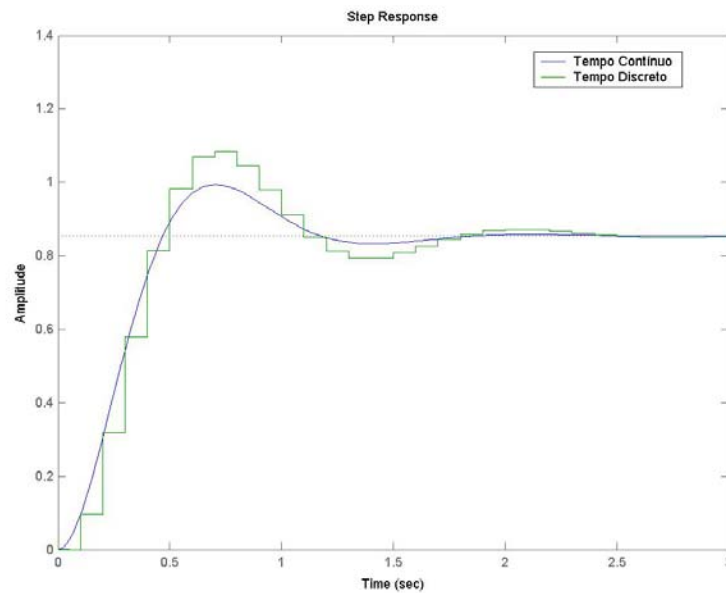
### **3.4. Definição do Tempo de Amostragem**

O tempo de amostragem foi definido através de testes realizados com as plantas no programa MatLab® (versão 6.5), utilizando uma de suas funções, mais especificamente a função (c2d) que converte uma função de transferencia, que está no modo contínuo, para o modo discreto. Os testes foram feitos com tempos de amostragem entre 100 ms e 1 ms e foi constatado que as plantas com tempo de amostragem abaixo de 5 ms reproduzem perfeitamente a planta no tempo contínuo.

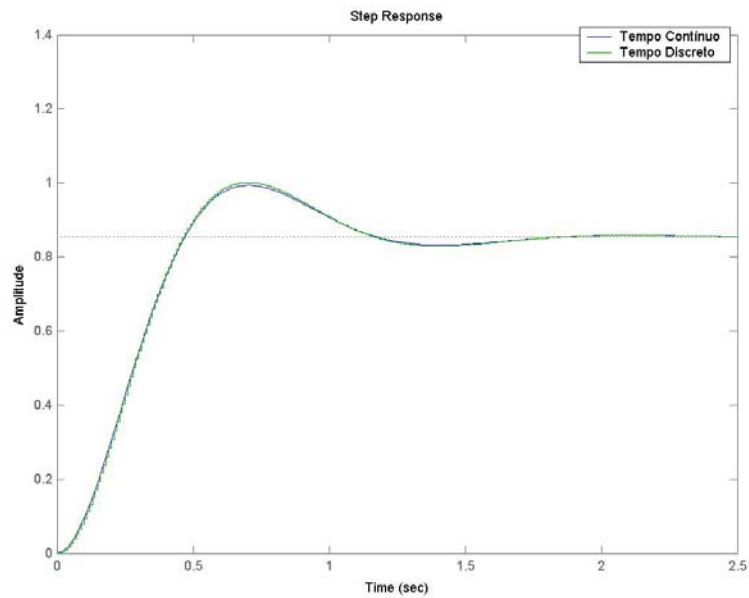
O tempo de amostragem foi definido como 5 ms por causa do *software* implementado. No software os valores de saída da planta são adquiridos pelo microprocessadores e salvos em um vetor que suporta no máximo 7500 valores. Se o tempo de amostragem for menor que 5 ms, para visualizar a planta controlada precisaria de mais que 7500 aquisições, não sendo possível pelo fato do vetor não suportar tantos valores.



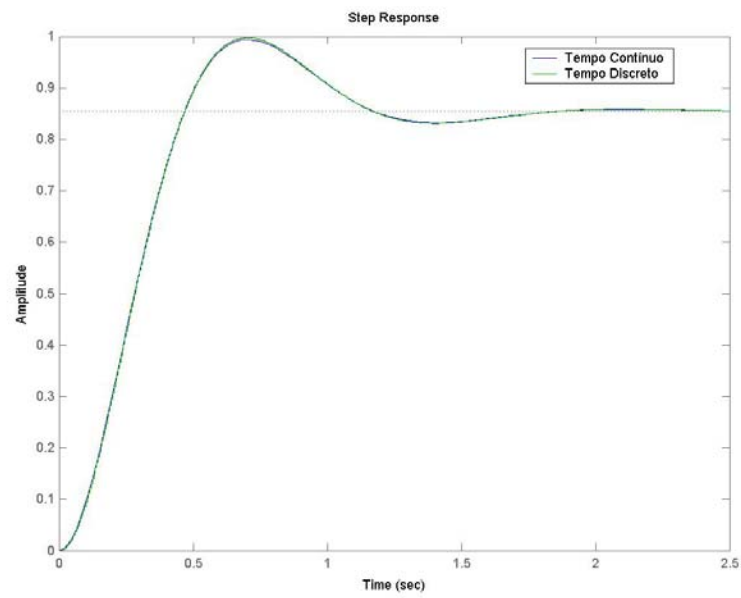
As figuras 3.12, 3.13, 3.14 e 3.15 mostram como ficou a planta de  $\xi = 0,5$  com tempos de amostragem de 100 ms, 10ms, 5 ms e 1 ms respectivamente, em relação à planta no tempo contínuo.



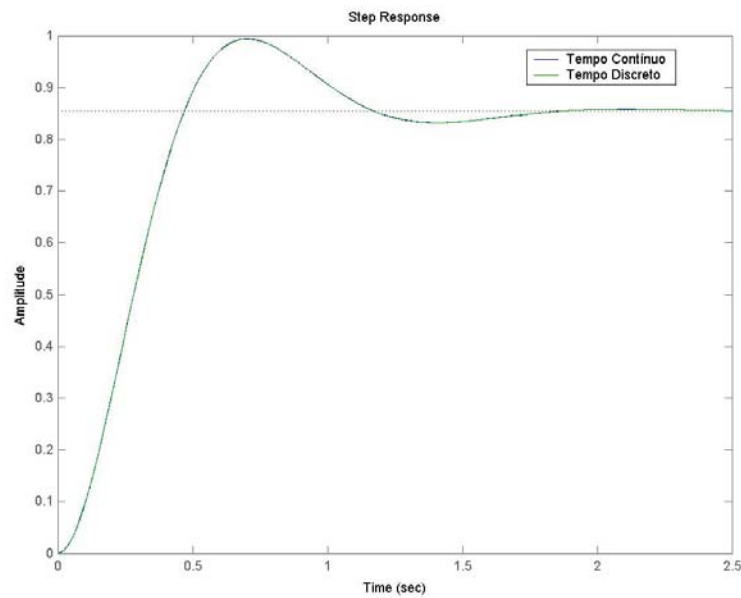
**Figura 3.12** – Gráfico da equação com  $\xi = 0,5$  com tempo de amostragem de 100 ms



**Figura 3.13** – Gráfico da equação com  $\xi = 0,5$  com tempo de amostragem de 10 ms



**Figura 3.14** – Gráfico da equação com  $\xi = 0,5$  com tempo de amostragem de 5 ms

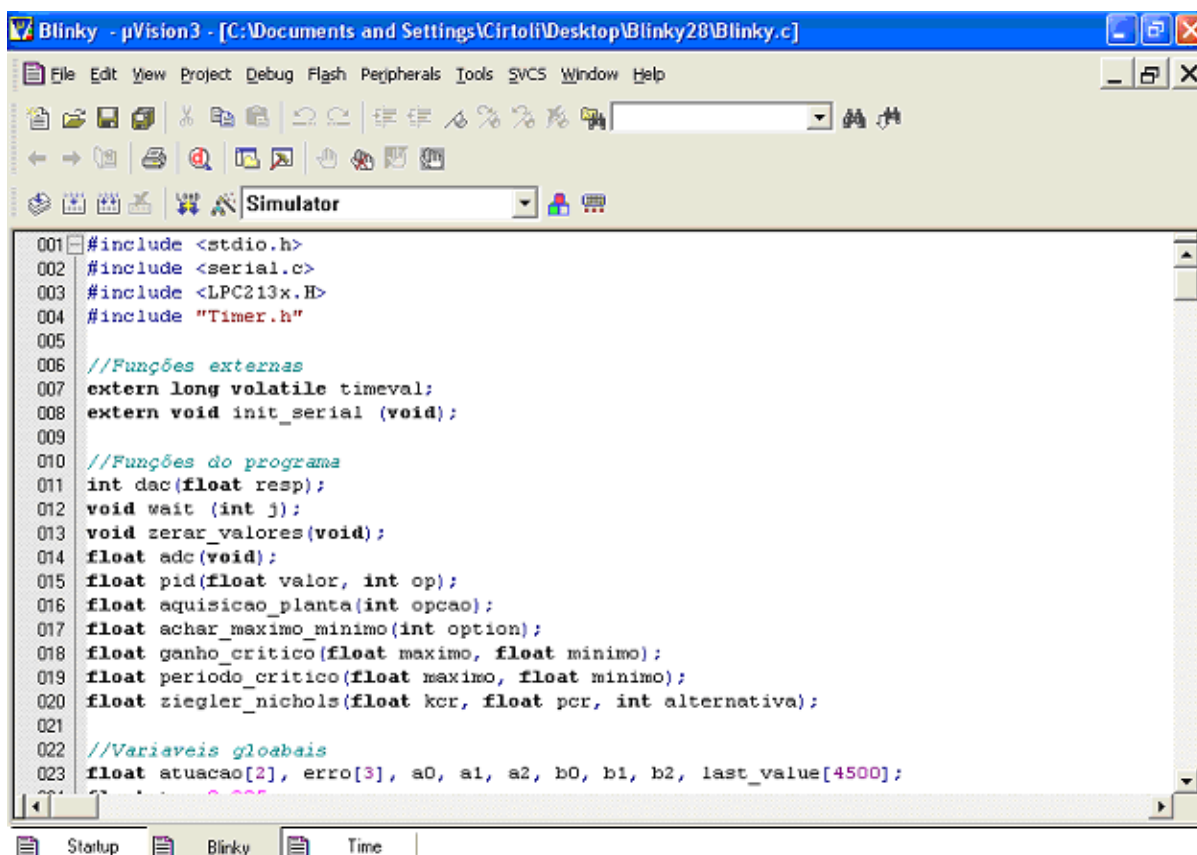


**Figura 3.15** – Gráfico da equação com  $\xi = 0,5$  com tempo de amostragem de 1 ms

## 4. IMPLEMENTAÇÃO DO SOFTWARE E RESULTADOS

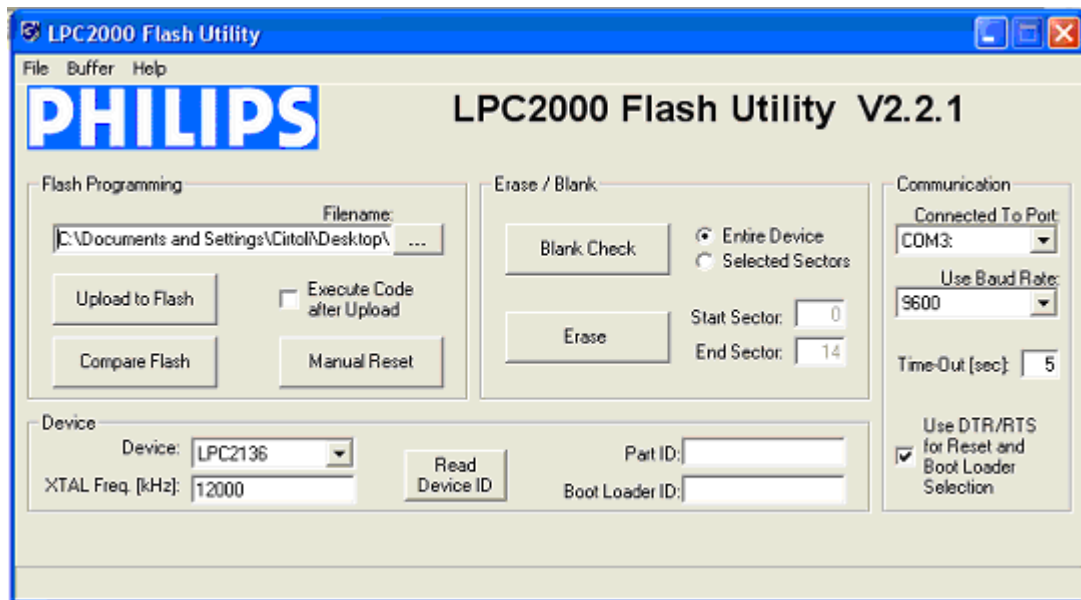
Neste capítulo será visto o funcionamento do *software* e os resultados práticos obtidos.

O *software* do microcontrolador foi escrito em linguagem C para a família ARM7, utilizando o IDE uVision da Keil Software (Figura 4.1). No *software* do microcontrolador estão embutidas as funções que fazem o controle das plantas mencionadas no capítulo 3.



**Figura 4.1** – Ambiente IDE uVision.

Para gravar o código compilado no microcontrolador foi utilizado o programa LPC2000 Flash Utility da Philips (Figura 4.2).

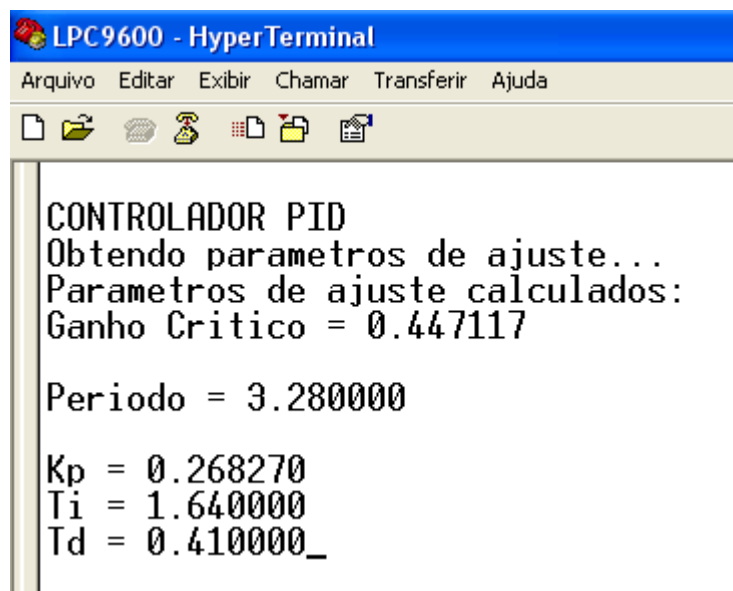


**Figura 4.2** – Ambiente de gravação LPC2000 Flash Utility

### 4.1. Interface com o usuário

Como o kit de desenvolvimento do LPC 2136 dispõe de duas portas seriais, uma foi utilizada para a programação do *software* (COM0) e a outra para a interface com o usuário (COM1). Esta interface ocorre através de mensagens de texto recebidas pelo software Hyper Terminal do Windows.

Na figura 4.3 temos um aspecto da tela com o sistema sendo utilizado.



```
LPC9600 - HyperTerminal
Arquivo  Editar  Exibir  Chamar  Transferir  Ajuda

CONTROLADOR PID
Obtendo parametros de ajuste...
Parametros de ajuste calculados:
Ganho Critico = 0.447117

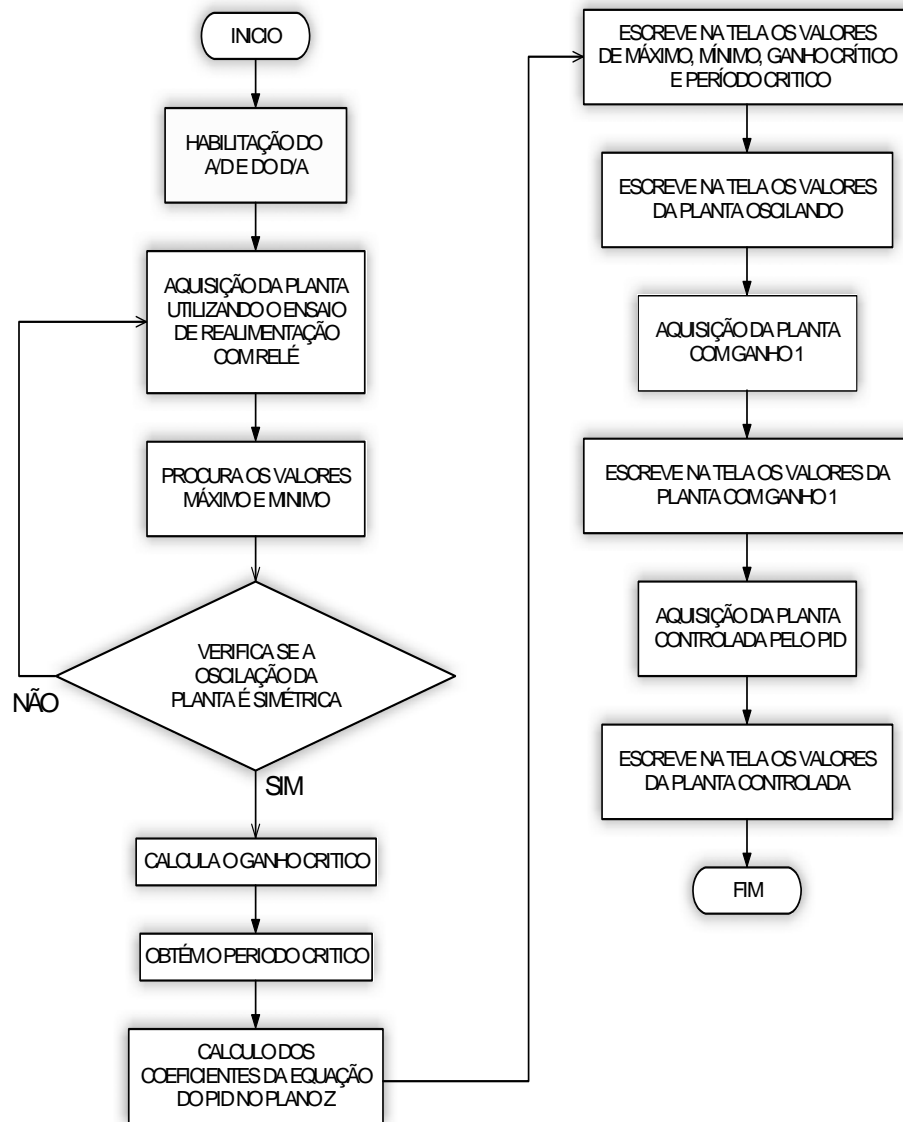
Periodo = 3.280000

Kp = 0.268270
Ti = 1.640000
Td = 0.410000_
```

**Figura 4.3** – Programa funcionando no Hyper Terminal

### 4.2. Descrição da Função Principal do Software

O diagrama em blocos do software é apresentado na figura 4.4, e seu funcionamento será explicado no decorrer deste capítulo.



**Figura 4.4** – Diagrama em blocos do software.

O primeiro bloco do programa é a habilitação do A/D e do D/A do microcontrolador, que é feito de maneira bastante simples. Para habilitá-los é necessário apenas selecionar o registrador PINSEL correspondente como mostra a figura 4.5.

Após a habilitação do A/D e do D/A, é feita a aquisição da planta utilizando o ensaio de realimentação com relé, sendo que o valor de atuação máxima do relé é aumentada até a oscilação dos sistema ficar em simetria. Após os valores adquiridos são salvos em um vetor para serem utilizados posteriormente.

```

034
035 PINSEL1 = 0x00080000;           //Habilita o D/A;
036 PINSEL1 |= 0x00400000;         //Habilita o A/D;

```

**Figura 4.5** – Trecho de código onde habilita o A/D e o D/A.

A aquisição da planta é feita com um tempo de amostragem de 5 ms, tanto no caso descrito anteriormente (realimentação por relé), como nos casos em que se adquire a planta com ganho unitário e com a planta já controlada.

O relé atua de 0 volts à um valor que faça a oscilação ser simétrica, sendo que ele atua para zero quando a diferença entre o setpoint e o valor do A/D for menor que 0,3, e quando esta diferença for maior que 0,8 ele atua para o máximo.

Encontra-se os valores de máximo e mínimo (posteriormente será explicado como foi encontrado estes valores), com essas duas informações calcula-se o ganho crítico (equação 2.12) e obtém-se o período crítico.

Com as equações vistas no capítulo 2, e com os valores do ganho e do período crítico calcula-se os coeficientes do PID no plano Z.

### **4.3. Funções do Software**

O software é composto de várias funções, cada uma com um objetivo distinto. Este capítulo mostrará o que cada uma delas executa. O código fonte completo do software está no anexo D.

#### **4.3.1. Função ADC**

Esta função faz a conversão A/D do microcontrolador. Primeiramente, a função configura e inicia a conversão A/D através do registrador ADCR, e depois, grava o valor do registrador ADDR, que está em bits.

Após gravar o valor, transforma este em um número que corresponde ao valor de tensão e o retorna para a função principal.

#### 4.3.2. Função DAC

Transforma o valor da atuação do PID, em um valor correspondente em bits, enviando este valor para o pino 11 do microcontrolador.

#### 4.3.3. Função Wait

Função que serve como um *"delay"*, o valor setado será o correspondente milissegundos.

#### 4.3.4. Função Zerar\_Valores

Apenas zera os valores do D/A, das atuações e erros, para quando houver outra aquisição da planta.

#### 4.3.5. Função Aquisição da Planta

Adquiri os valores da saída da planta. Os valores adquiridos são armazenados em vetor denominado *last\_value*. Nesta função é definido o tempo de amostragem através da função *wait*, como mostra o código fonte na figura 4.6.

```

125 float aquisicao_planta(int opcao)
127 {
128     int i;
129     float ad;
130
131     for(i=0; i<4500; i++)
132     {
133         wait(5);
134         ad = adc();
135         dac(pid(ad, opcao));
136         last_value[i] = ad;
137     }
138     return(0);
139 }
```

**Figura 4.6** – Trecho de código da aquisição da planta.

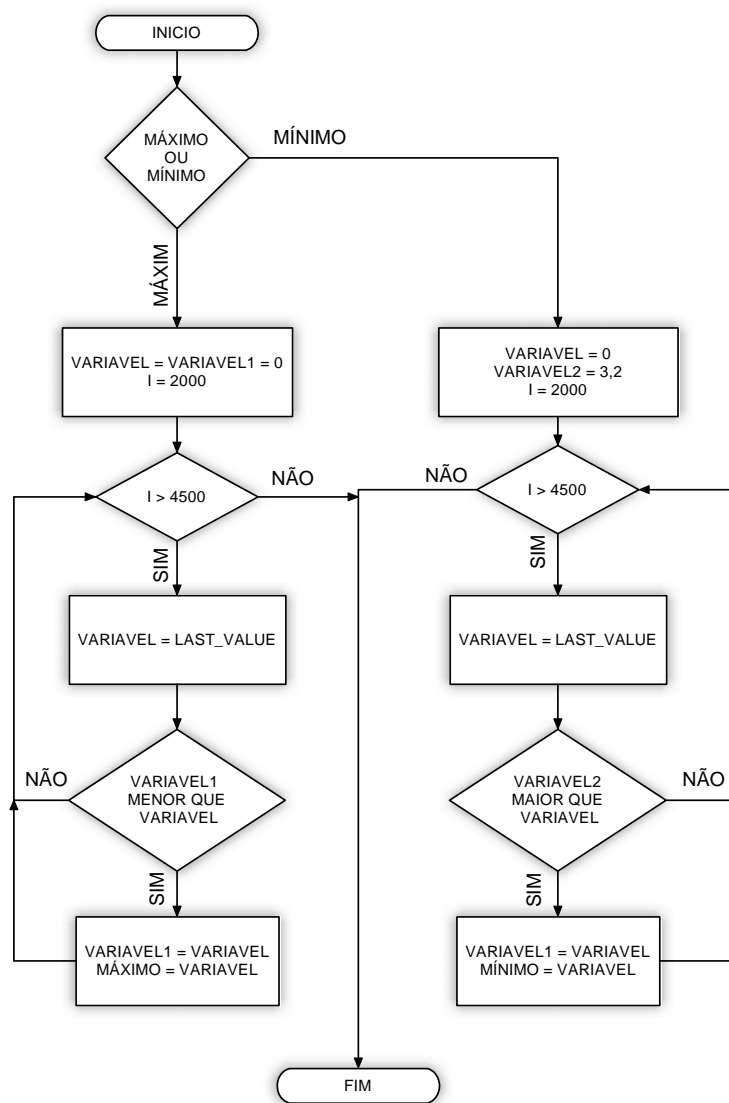
#### 4.3.6. Função Achar Máximo e Mínimo

Encontra o máximo e o mínimo da planta, quando esta foi ensaiada com realimentação com relé.



Para encontrar o valor máximo, o programa define uma variável como zero, e depois fica comparando esta variável com o valor do vetor (last\_value), salvando o valor do vetor no valor máximo e no valor desta variável. Isso fica acontecendo até chegar no final do vetor.

Para encontrar o valor mínimo é utilizado o mesmo procedimento com algumas pequenas alterações, que podem ser vistas no próprio código. O diagrama em blocos desta função é mostrado na figura 4.7.



**Figura 4.7** – Diagrama em blocos da função que encontra o máximo e o mínimo.

#### 4.3.7. Função Ganho Crítico

Esta função calcula o valor do ganho crítico utilizando a equação 2.13. Na figura 4.8 está apresentado o trecho de código que mostra essa função.

```

132 float ganho_critico(float maximo, float minimo)
133 {
134     float gcr, amplitude;
135
136     amplitude = maximo - minimo;
137     gcr = (4 * rele)/(3.1416 * amplitude);
138     return(gcr);
139 }
140

```

**Figura 4.8** – Código fonte da função que calcula o ganho crítico.

#### 4.3.8. Função Período Crítico

Encontra o período crítico, para isso, a função determina quando chega ao máximo, depois chega no ponto médio do gráfico e conta até passar duas vezes pelo mesmo ponto. Essa contagem vezes o valor do tempo de aquisição é o período crítico.

#### 4.3.9. Função Ziegler Nichols

Calcula os coeficientes da equação do PID por meio de software, através das fórmulas vistas no capítulo 2. Pode-se escolher qual controlador (P, PI, PID) utilizar, porém somente o programador tem acesso a esta opção. A figura 4.9 demonstra esta função.

```

097 float ziegler_nichols(float kcr, float pcr, int alternativa)
098 {
099     float kp, ki, kd, ti=0, td=0;
100
101     switch(alternativa)
102     {
103         case 0: kp = 0.5 * kcr;
104                ki = 0;
105                kd = 0;
106                break;
107         case 1: kp = 0.4 * kcr;
108                ki = (0.25 * kcr * t) / pcr;
109                kd = 0;
110                ti = 0.8 * pcr;
111                break;
112         case 2: kp = 0.6 * kcr;
113                ki = (0.6 * kcr * t) / pcr;
114                kd = (0.075 * kcr * pcr) / t;
115                ti = 0.5 * pcr;
116                td = 0.125 * pcr;
117                break;
118     }
119
120     printf("\nKp = %f\nTi = %f\nTd = %f",kp,ti,td);
121     a0 = kp + ki + kd;
122     a1 = ki - kp - (2 * kd);
123     a2 = kd;

```

**Figura 4.9** – Código fonte da função que calcula os coeficientes Kp, Ti, Td através do método de Ziegler-Nichols.

#### 4.2.10. Função PID

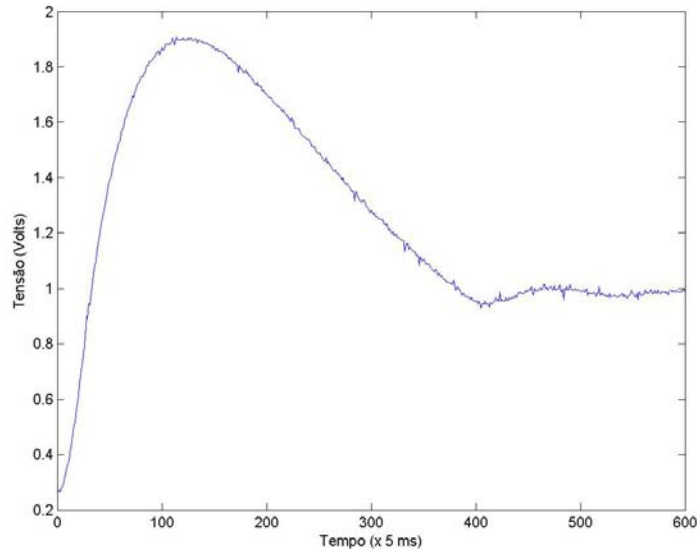
Esta função tem como finalidade fechar o laço e determinar a atuação que será aplicada na entrada da planta. Nesta parte do programa está a equação PID no plano Z e a atuação do ensaio com realimentação com relé.

### 4.4. Resultados

Nos testes práticos realizados, o projeto controlou as plantas com  $\xi = 0,5$  e  $\xi = 0,8$  com resultados satisfatórios. A planta de  $\xi = 0,2$  não obteve ajuste por causa do seu alto *offset* (250 mV).

A figura 4.10 mostra a planta de  $\xi = 0,2$  com realimentação com ganho unitário adquirida pelo microcontrolador, comparando esta figura com a figura 3.9 (simulada

pelo MatLab®) pode ver que os dois gráficos não coincidem diferentemente das plantas com  $\xi = 0,5$  e  $\xi = 0,8$ .

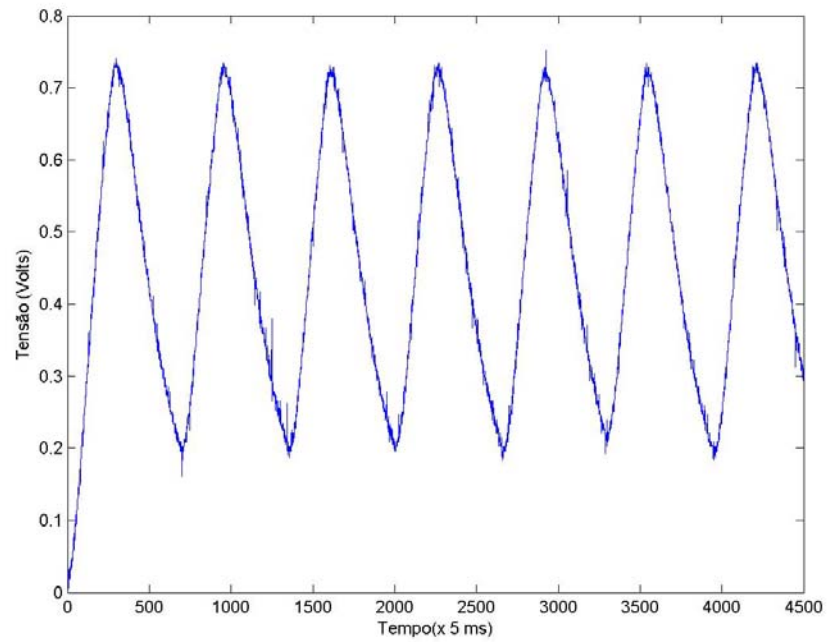


**Figura 4.10** – Planta de  $\xi = 0,2$  adquirida pelo microcontrolador em malha fechada com entrada do tipo salto.

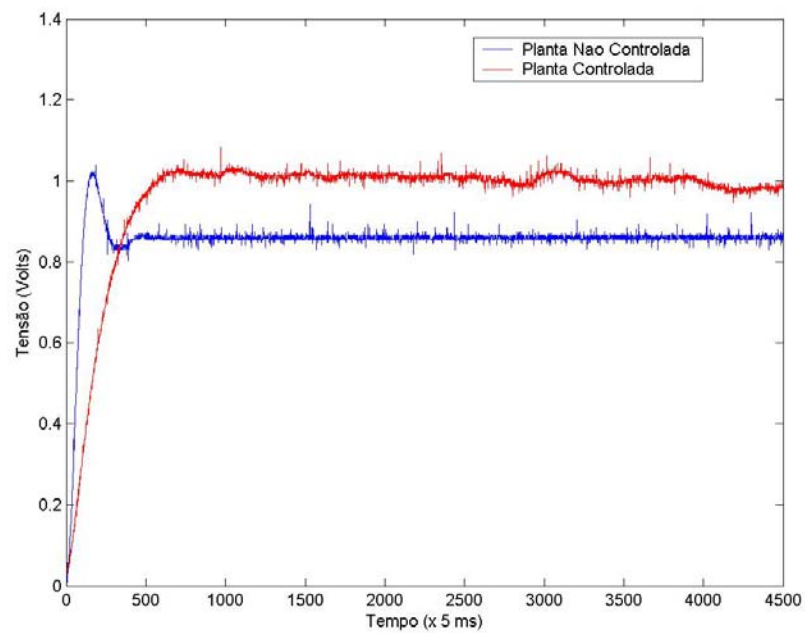
A seguir está apresentado os 3 testes realizados em cada planta ( $\xi = 0,5$  e  $\xi = 0,8$ ) com o sistema efetuando o controle. As figuras mostram os sistema quando está sendo realimentado por relé (sistema oscilando) e apresenta também a resposta das plantas controladas e não controladas. As tabelas a mostram o *overshoot* ( $M_o$ ) e o tempo de estabilização ( $t_s$ ) aproximado, os valores do ganho crítico ( $k_{cr}$ ) e período crítico ( $p_{cr}$ ) e os coeficientes  $k_p$ ,  $t_i$  e  $t_d$  do controlador PID calculados pelo software.

	$\xi = 0,5$			$\xi = 0,8$		
	Teste 1	Teste 2	Teste 3	Teste 1	Teste 2	Teste 3
$M_o$	0 %	0 %	0 %	0%	0%	0%
$t_s$	2,5 s	2,5 s	2,5 s	2,5 s	2,5 s	2,5 s
$k_{cr}$	0,447117	0,435033	0,457279	1,506331	1,556265	1,556265
$p_{cr}$	3,28 s	3,295 s	3,195 s	3,24 s	3,2 s	3,22
$K_p$	0,26827	0,26102	0,274368	0,903799	0,933759	0,933759
$t_i$	1,64 s	1,6475 s	1,5975 s	1,622 s	1,6 s	1,61 s
$t_d$	0,41 s	0,411875 s	0,399375 s	0,405 s	0,4 s	0,4025 s

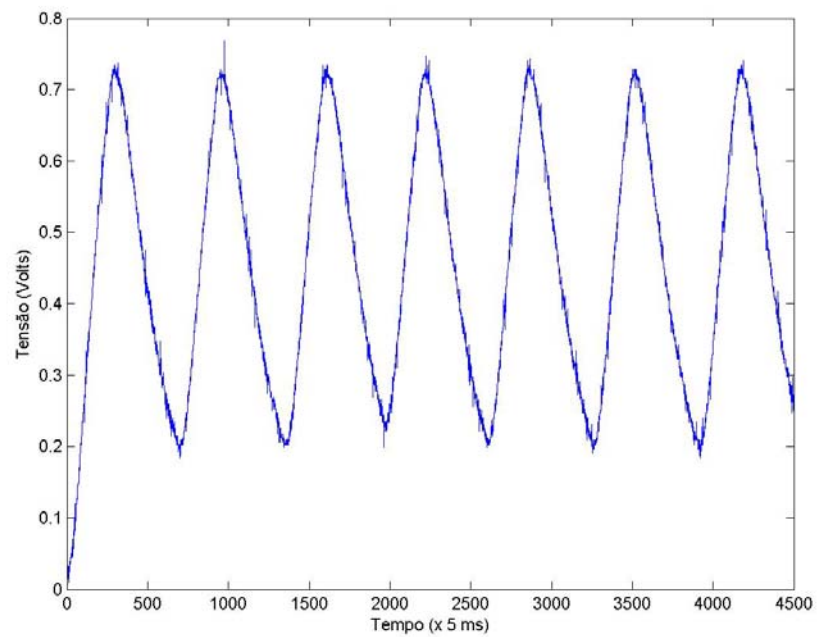
**Tabela 4.1** – Tabela com valores das plantas controladas



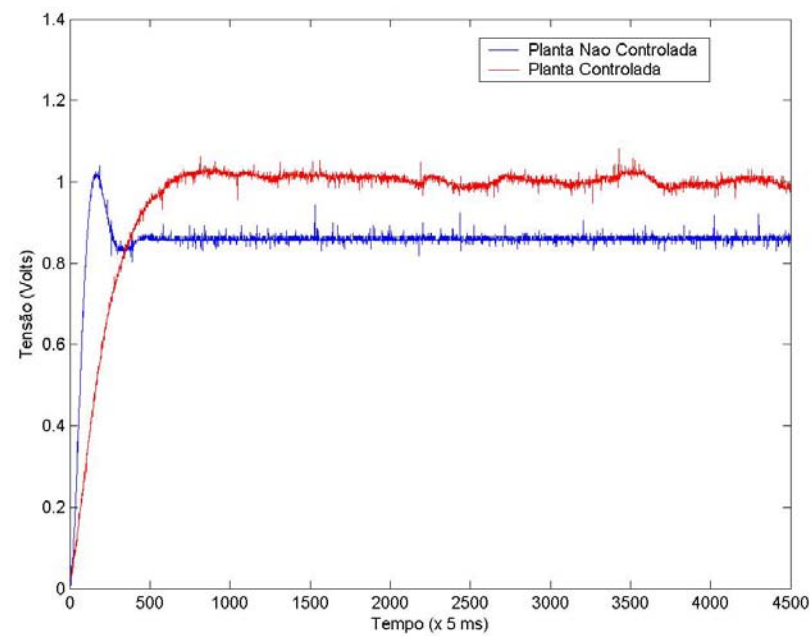
**Figura 4.11** – Teste 1 da planta de  $\xi = 0,5$  adquirida com ensaio de realimentação com relé.



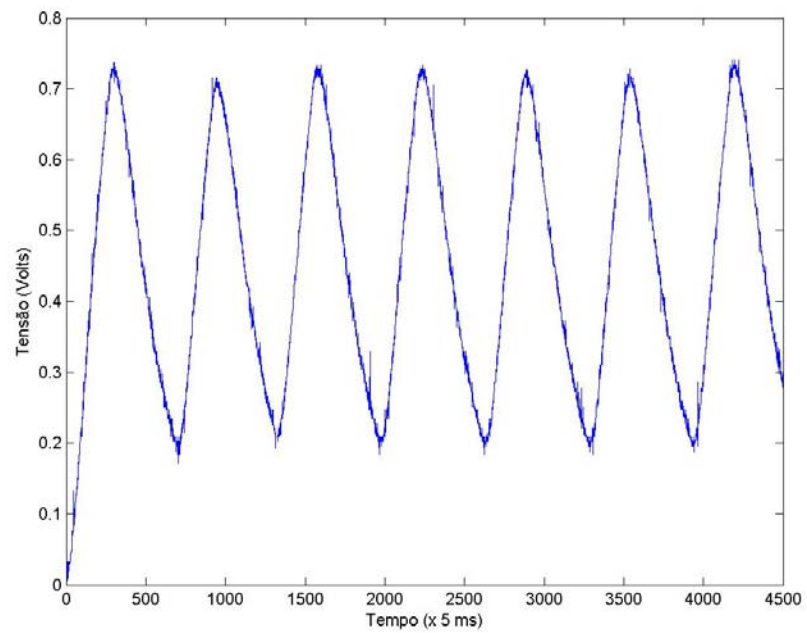
**Figura 4.12** – Teste 1 da planta de  $\xi = 0,5$  controlada e não controlada.



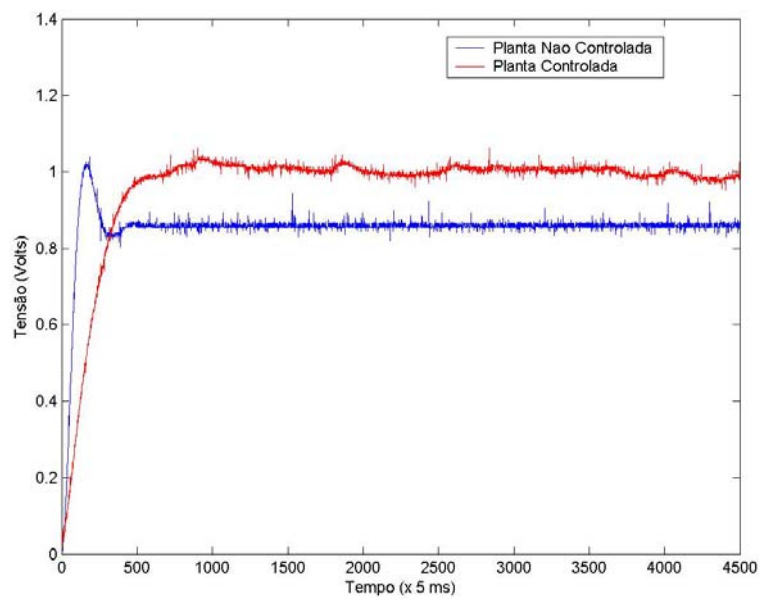
**Figura 4.13** – Teste 2 da planta de  $\xi = 0,5$  adquirida com ensaio de realimentação com relé.



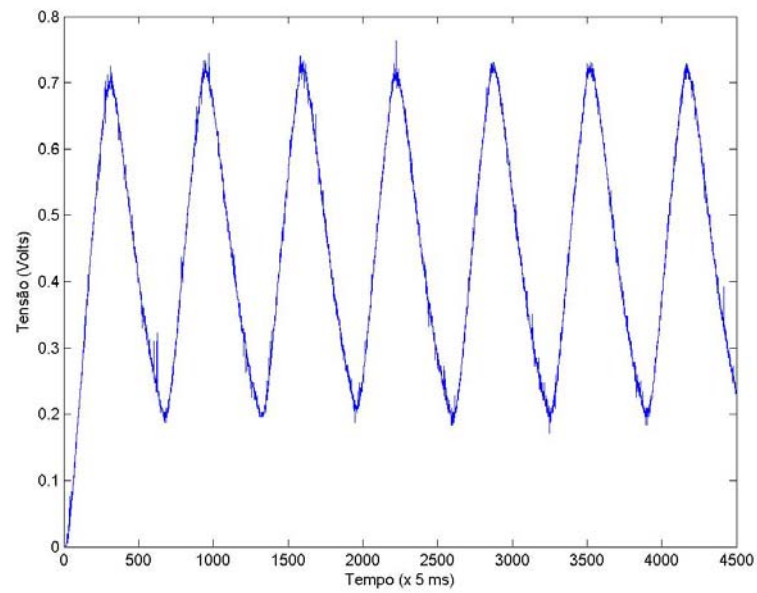
**Figura 4.14** – Teste 2 da planta de  $\xi = 0,5$  controlada e não controlada.



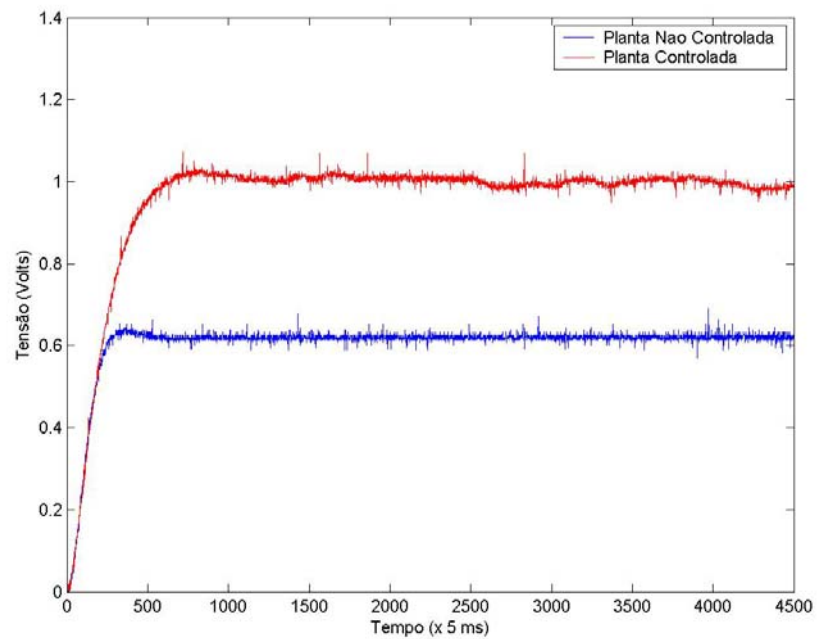
**Figura 4.15** – Teste 3 da planta de  $\xi = 0,5$  adquirida com ensaio de realimentação com relê.



**Figura 4.16** – Teste 3 da planta de  $\xi = 0,5$  controlada e não controlada.

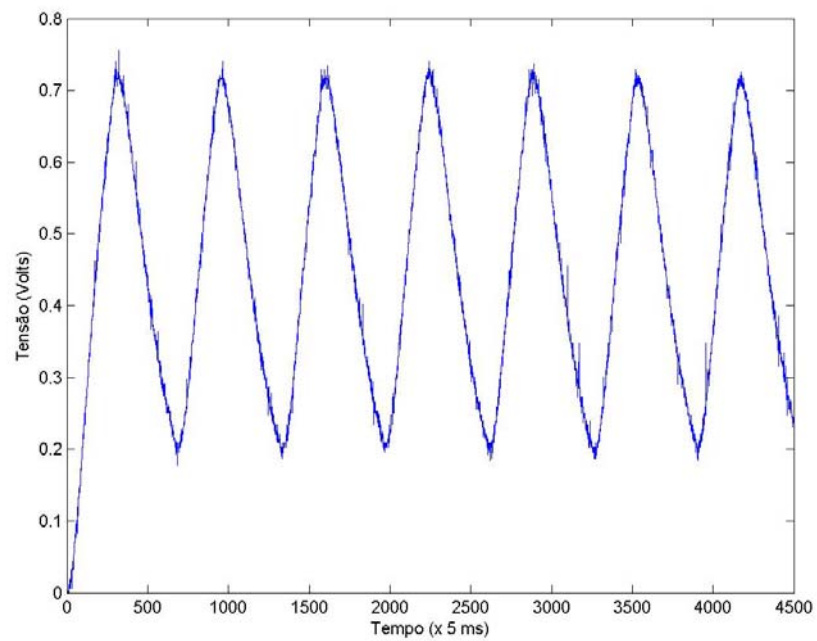


**Figura 4.17** – Teste 1 da planta de  $\xi = 0,8$  adquirida com ensaio de realimentação com relê.

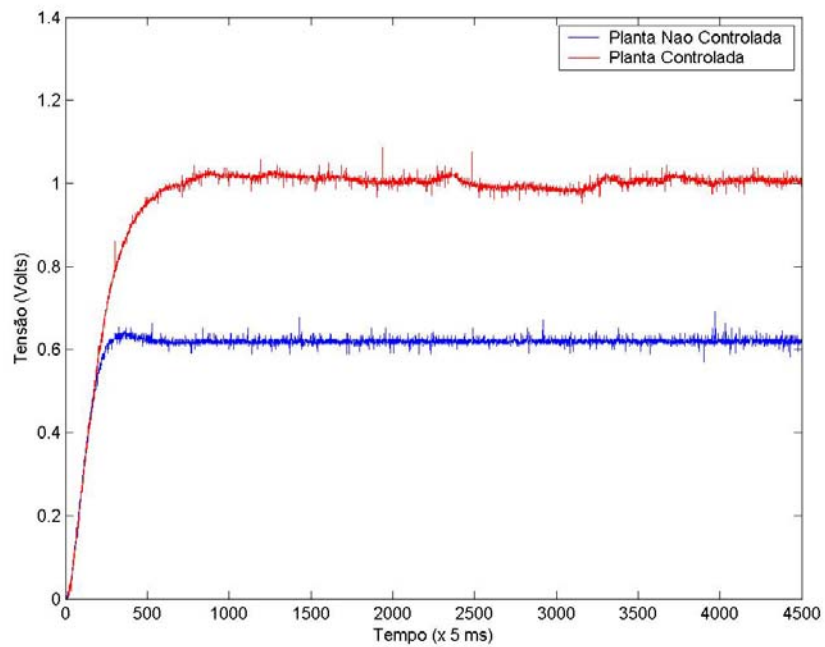


**Figura 4.18** – Teste 1 da planta de  $\xi = 0,8$  controlada e não controlada.

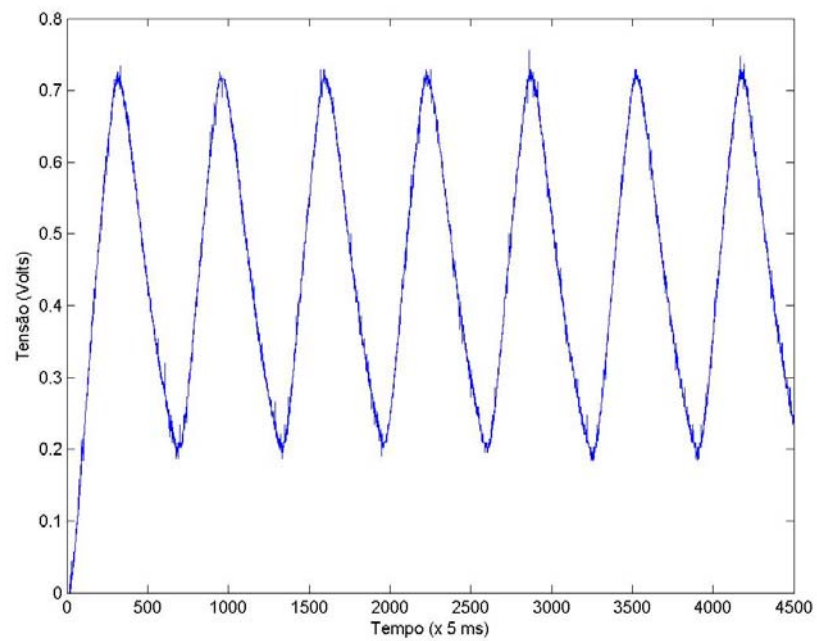




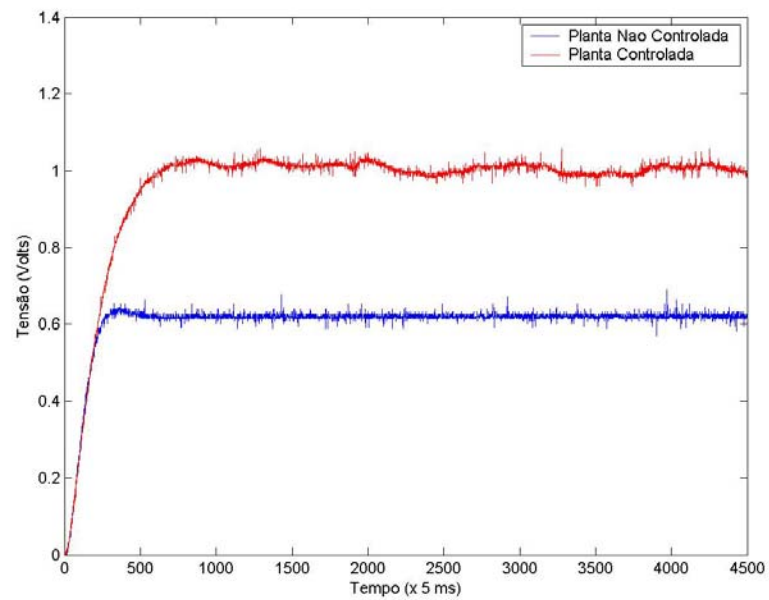
**Figura 4.19** – Teste 2 da planta de  $\xi = 0,8$  adquirida com ensaio de realimentação com relé.



**Figura 4.20** – Teste 2 da planta de  $\xi = 0,8$  controlada e não controlada.



**Figura 4.21** – Teste 3 da planta de  $\xi = 0,8$  adquirida com ensaio de realimentação com relé.



**Figura 4.22** – Teste 3 da planta de  $\xi = 0,8$  controlada e não controlada.

## 5. CONCLUSÃO

O objetivo deste trabalho foi o de desenvolver um PID com sintonia automática implementado com um processador de 32 bits para controlar plantas de segunda ordem, utilizando o método de Ziegler e Nichols.

A partir de testes realizados, foi comprovado que o método do período crítico em que eleva-se o ganho  $K_p$  gradualmente até a planta começar a oscilar é pouco eficiente. Com este método não foi possível encontrar o ganho crítico nas plantas testadas, conseqüentemente estas não obtiveram ajuste no controlador.

Com o método de Ziegler-Nichols de realimentação com relé foi possível encontrar o ganho crítico e, assim, obter os parâmetros de ajuste. A única dificuldade encontrada na implementação foi a de fazer a planta oscilar de forma simétrica.

Frente aos objetivos propostos conseguiu-se desenvolver um *software* eficiente para a aquisição e controle das plantas de segunda ordem. A validação do software foi realizado com duas plantas distintas conforme apresentado no capítulo 4.

Para dar continuidade a proposta deste trabalho, propõe a implementação de uma interface mais acessível com o usuário. Também a utilização de constantes baseadas em outros autores [11], [12], para melhoria do controle realizado nas plantas.

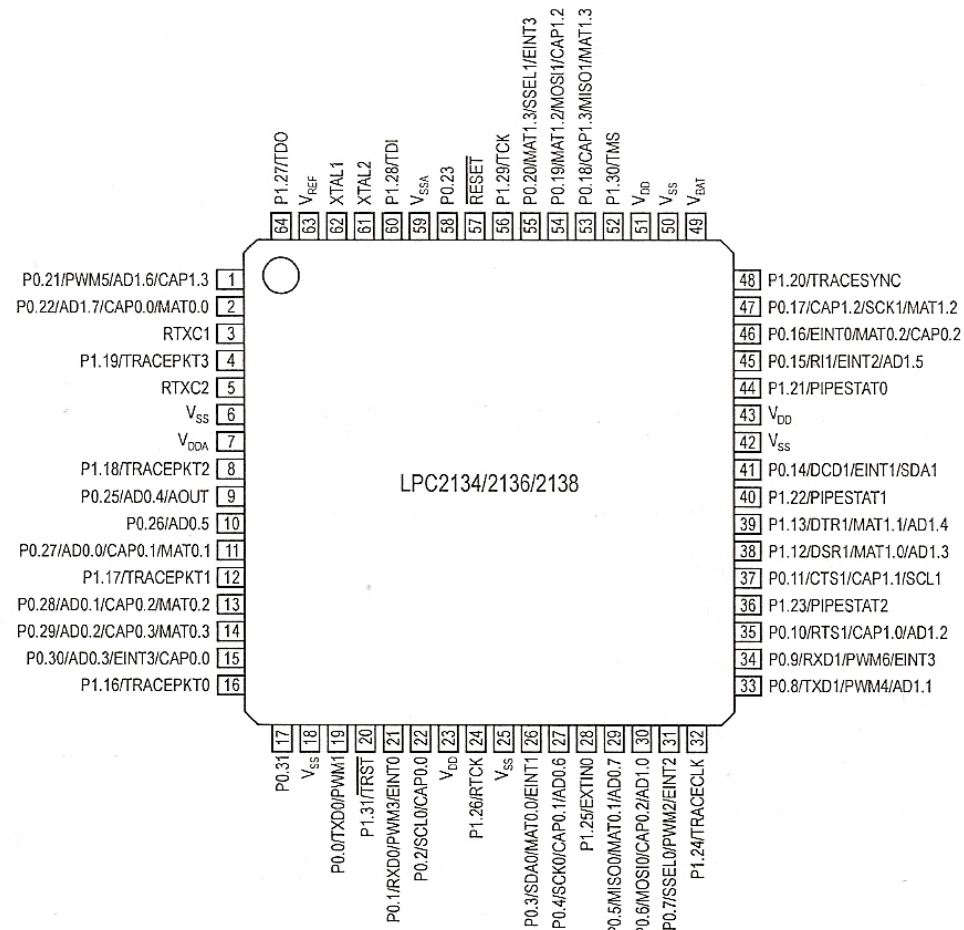
## REFERÊNCIAS BIBLIOGRÁFICAS

- [1] ASTRÖM, K.J. ; HÄGGLUND T. - **The future of PID Control.** - Control Engineering Practice, 1995.
- [2] ASTRÖM, K.J. ; HÄGGLUND T. - **The future of PID Control.** - Control Engineering Practice, 2001.
- [3] Novus Produtos Eletrônica, [www.novus.com.br](http://www.novus.com.br).
- [4] Siemens, [www.siemens.com.br](http://www.siemens.com.br).
- [5] OGATA, KATSUHIKO. - **Engenharia de Controle Moderno.** - Editora Prentice-Hall do Brasil – RJ – 1998.
- [6] SEBORG, D. E. ; EDGAR, T. F.; MELLICHAMP, D. A. - **Process dynamics and control.**- New York: John Wiley & Sons, 1989.
- [7] BAZANELLA, ALEXANDRE SANFELICE ; SILVA, JOÃO MANOEL DA. - **Ajuste de Controladores PID.** - [www.ece.ufrgs.br/~jmgomes/pid/Apostila/apostila/](http://www.ece.ufrgs.br/~jmgomes/pid/Apostila/apostila/).
- [8] BAZANELLA, ALEXANDRE SANFELICE ; SILVA, JOÃO MANOEL DA. - **SISTEMAS DE CONTROLE: PRINCIPIOS E METODOS DE PROJETO.** - Editora UFRGS, 1ª Edição.
- [9] SOUZA, DANIEL RODRIGUES DE. - **Microcontroladores ARM7.** - Editora Érica - São Paulo - 2006.
- [10] Keil Embedded Development Tools, [www.keil.com](http://www.keil.com).
- [11] HANG, C. C. ; ASTRÖM, K. J.; HO, W. K. - **Refinements of the Ziegler–Nichols tuning formula.** - Proc Inst. Elect. Eng.—Part D: Control Theory and Applications, vol. 138, no. 2, 1991.
- [12] BASILIO, J. C. ; MATOS, S. R. - **Design of PI and PID Controllers With Transient Performance Specification.** - IEEE TRANSACTIONS ON EDUCATION, VOL. 45, NO. 4, NOVEMBER 2002.

## **ANEXOS**

## ANEXO A - PINAGEM DO LPC 2136

Pinagem do LPC 2136:



A tabela seguinte tem a descrição dos pinos do LPC 2136

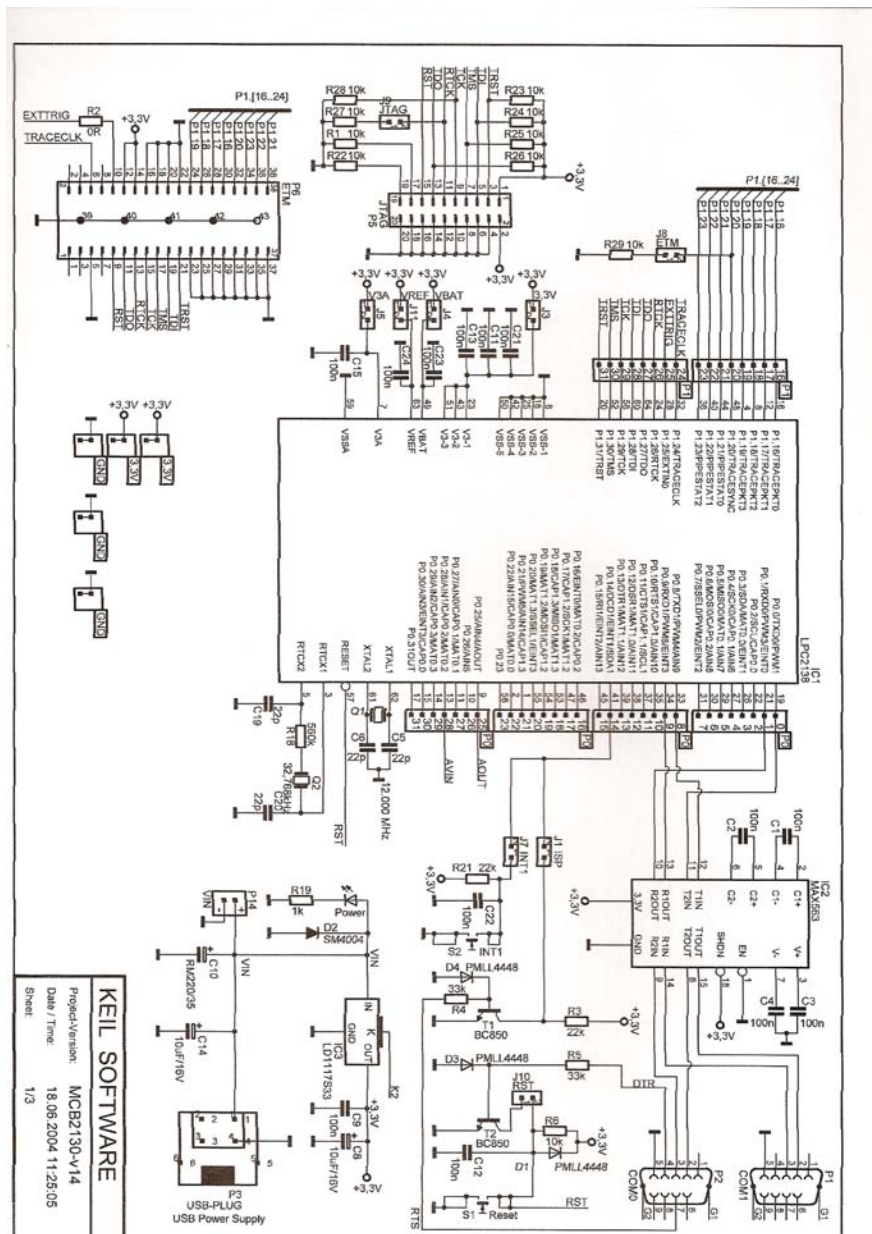
Pino	Nome	Tipo	Descrição
1	P0.21/PWM5/AD1.6/CAP1.3	I/O	P0.21 - Pino Digital
		O	PWM5 – Saída PWM 5
		I	AD1.6 – Conversor A/D 1, canal 6
		I	CAP1.3 – Entrada de captura do timer 1, canal 3
2	P0.22/AD1.7/CAP0.0/MAT0.0	I/O	P0.22 – Pino Digital
		I	AD1.7 – Conversor A/D 1, canal 7
		I	CAP0.0 – Entrada de captura do timer 0, canal 0
		O	MAT0.0 – Saída do comparador do timer 0, canal 0
3	RTCX1		Cristal oscilador de 32768Hz para o RTC interno
4	P1.19	I/O	P1.19 – Pino Digital
5	RTCX2		Cristal oscilador de 32768Hz para o RTC interno
6	VSS		GND
7	VDD		VDD (3,3V)
8	P1.18	I/O	P1.18 – Pino Digital
9	P0.25/AD0.4/AOUT	I/O	P0.25 – Pino Digital
		I	AD0.4 – Conversor A/D 0, canal 4
		O	AOUT – Saída do D/A
10	P0.26/AD0.5	I/O	P0.26 – Pino Digital
		I	AD0.5 – Conversor A/D 0, canal 5
11	P0.27/AD0.0/CAP0.1/MAT0.1	I/O	P0.27 – Pino Digital
		I	AD0.0 – Conversor A/D 0, canal 0
		I	CAP0.1 – Entrada de captura do timer 0, canal 1
		O	MAT0.1 – Saída do comparador do timer 0, canal 1
12	P1.17	I/O	P1.17 – Pino Digital
13	P0.28/AD0.1/CAP0.2/MAT0.2	I/O	P0.28 – Pino Digital
		I	AD0.1 – Conversor A/D 0, canal 1
		I	CAP0.2 – Entrada de captura do timer 0, canal 2
		O	MAT0.2 – Saída do comparador do timer 0, canal 2
14	P0.29/AD0.2/CAP0.3/MAT0.3	I/O	P0.29 – Pino Digital
		I	AD0.2 – Conversor A/D 0, canal 2
		I	CAP0.3 – Entrada de captura do timer 0, canal 3
		O	MAT0.3 – Saída do comparador do timer 0, canal 3
15	P0.30/AD0.3/EINT3/CAP0.0	I/O	P0.30 – Pino Digital
		I	AD0.3 – Conversor A/D 0, canal 3
		I	EINT3 – Interrupção externa 3
		I	CAP0.0 – Entrada de captura do timer 0, canal 0
16	P1.16	I/O	P1.16 - Pino Digital
17	P0.31	O	P0.31 - Pino Digital
18	GND		GND
19	P0.0/TXD0/PWM1	I/O	P0.0 - Pino Digital
		O	TXD0 – TX da UART0
		O	PWM1 – Saída PWM 1
20	P1.31	I/O	P1.31 - Pino Digital
21	P0.1/RXD0/PWM3/EINT0	I/O	P0.1 - Pino Digital
		I	RXD0 – RX da UART0
		O	PWM3 – Saída PWM 3
		I	EINT0 – Interrupção externa 0
22	P0.2/SCL0/CAP0.0	I/O	P0.2 – Pino Digital
		I/O	SCL0 – Clock do módulo I2C0
		I	CAP0.0 – Entrada de captura do timer 0, canal 0
23	VDD		VDD (3,3V)
24	P1.26	I/O	P1.26 – Pino Digital
25	VSS		GND
26	P0.3/SDA0/MAT0.0/EINT1	I/O	P0.3 – Pino Digital
		I/O	SDA0 – Dados do módulo I2C0
		O	MAT0.0 – Saída do comparador do timer 0, canal 0
		I	EINT1 – Interrupção externa 1
27	P0.4/SCK0/CAP0.1/AD0.6	I/O	P0.4 – Pino Digital
		I/O	SCK0 – Clock do módulo SPI0

		I	CAP0.1 – Entrada de captura do timer 0, canal 1
		I	AD0.6 – Conversor A/D 0, canal 6
Pino	Nome	Tipo	Descrição
28	P1.25	I/O	P1.25 – Pino Digital
29	P0.5/MISO0/MAT0.1/AD0.7	I/O	P0.5 – Pino Digital
		I/O	MOSO0 – Entrada de dados do módulo SPI0
		O	MAT0.1 – Saída do comparador do timer 0, canal 1
		I	AD0.7 – Conversor A/D 0, canal 7
30	P0.6/MOSIO/CAP0.2/AD1.0	I/O	P0.6 – Pino Digital
		I/O	MOSIO – Saída de dados do módulo SPI0
		O	CAP0.2 – Entrada de captura do timer 0, canal 2
		I	AD1.0 – Conversor A/D 1, canal 0
31	P0.7/SSEL0/PWM2/EINT2	I/O	P0.7 – Pino Digital
		I	SSEL0 – Seleção do escravo do módulo SPI0
		O	PWM2 – Saída de PWM 2
		I	EINT2 – Interrupção externa 2
32	P1.24	I/O	P1.24 – Pino Digital
33	P0.8/TXD1/PWM4/AD1.1	I/O	P0.8 – Pino Digital
		O	TXD1 – TX da UART1
		O	PWM4 – Saída de PWM 4
		I	AD1.1 – Conversor A/D 1, canal 1
34	P0.9/RXD1/PWM6/EINT3	I/O	P0.9 – Pino Digital
		I	RXD1 – RX da UART1
		O	PWM6 – Saída de PWM 6
		I	EINT3 – Interrupção externa 3
35	P0.10/RTS1/CAP1.0/AD1.2	I/O	P0.10 – Pino Digital
		O	RTS1 – RTS da UART1
		I	CAP1.0 – Entrada de captura do timer 1, canal 0
		I	AD1.2 – Conversor A/D 1, canal 2
36	P1.23	I/O	P1.23 – Pino Digital
37	P0.11/CTS1/CAP1.1/SCL1	I/O	P0.11 – Pino Digital
		I	CTS1 – CTS da UART1
		I	CAP1.1 – Entrada de captura do timer 1, canal 1
		I/O	SCL1 – Clock do módulo I2C1
38	P0.12/DSR1/MAT1.0/AD1.3	I/O	P0.12 – Pino Digital
		I	DSR1 – DSRS da UART1
		O	MAT1.0 – Saída do comparador do timer 1, canal 0
		I	AD1.3 – Conversor A/D 1, canal 3
39	P0.13/DTR1/MAT1.1/AD1.4	I/O	P0.13 – Pino Digital
		O	DTR1 – DTR da UART1
		O	MAT1.1 – Saída do comparador do timer 1, canal 1
		I	AD1.4 – Conversor A/D 1, canal 4
40	P1.22	I/O	P1.22 – Pino Digital
41	P0.14/DCD1/EINT1/SDA1	I/O	P0.14 – Pino Digital
		I	DCD1 – DCD da UART1
		I	EINT1 – Interrupção externa 1
		I/O	SDA1 – Dados do módulo I2C1
42	VSS		GND
43	VDD		VDD (3,3V)
44	P1.21	I/O	P1.21 – Pino
45	P0.15/RI1/EINT2/AD1.5	I/O	P0.15 – Pino Digital
		I	RI1 – RI da UART1
		I	EINT2 – Interrupção externa 2
		I	AD1.5 – Conversor A/D 1, canal 5
46	P0.16/EINT0/MAT0.2/CAP0.2	I/O	P0.16 – Pino Digital
		I	EINT0 – Interrupção externa 0
		O	MAT0.2 – Saída do comparador do timer 0, canal 2
		I	CAP0.2 – Entrada de captura do timer 0, canal 2
47	P0.17/CAP1.2/SCK1/MAT1.2	I/O	P0.17 – Pino Digital
		I	CAP1.2 – Entrada de captura do timer 1, canal 2
		I/O	SCK1 – Clock do módulo SPI1
		O	MAT1.2 – Saída do comparador timer 1, canal 2
48	P1.20	I/O	P1.20 – Pino Digital



49	VBAT		Bateria para uso do RTC interno
50	VSS		GND
51	VDD		VDD (3,3V)
--	--	--	--
Pino	Nome	Tipo	Descrição
52	P1.30	I/O	P1.30 – Pino Digital
53	P0.18/CAP1.3/MISO1/MAT1.3	I/O	P0.18 – Pino Digital
		I	CAP1.3 – Entrada de captura do timer 1, canal 3
		I/O	MISO1 – Entrada de dados do módulo SPI1
		O	MAT1.3 – Saída do comparador do timer 1, canal 3
54	P0.19/MAT1.2/MOSI1/CAP1.2	I/O	P0.19 – Pino Digital
		O	MAT1.2 – Saída do comparador do timer 1, canal 2
		I/O	MOSI1 – Saída de dados do módulo SPI1
		I	CAP1.2 – Entrada de captura do timer 1, canal 2
55	P0.20/MAT1.3/SSEL1/EINT3	I/O	P0.20 – Pino Digital
		O	MAT1.3 – Saída do comparador timer 1, canal 3
		I	SSEL1 – Seleção do escravo do módulo SPI1
		I	EINT3 – Interrupção externa 3
56	P1.29	I/O	P1.29 – Pino Digital
57	RST		Reset
58	P0.23	I/O	P0.23 – Pino Digital
59	VSS		GND
60	P1.28	I/O	P1.28 – Pino Digital
61	X1		Cristal
62	X2		Cristal
63	VREF+		Tensão de referência
64	P1.27	I/O	P1.27 – Pino Digital.

## ANEXO B - KIT DE DESENVOLVIMENTO DA KEIL



## ANEXO C - CÓDIGO FONTE DO SOFTWARE

Código fonte do projeto:

### **TCC.C:**

```
#include <stdio.h>
#include <serial.c>
#include <LPC213x.H>
#include "Timer.h"

//Funções externas
extern long volatile timeval;
extern void init_serial (void);

//Funções do programa
void dac(void);
void wait (int j);
void zerar_valores(void);
float adc(void);
void pid(float valor, int op);
float aquisicao_planta(int opcao);
float achar_maximo_minimo(int option);
float ganho_critico(float maximo, float minimo);
float periodo_critico(float maximo, float minimo);
float ziegler_nichols(float kcr, float pcr, int alternativa);

//Variaveis gloabais
float atuacao[2], erro[3], a0, a1, a2, b0, b1, b2, last_value[4500], atuar;
float t = 0.005;
float rele = 0;

void main (void)    //Função principal
{
    int i;
    float maximo=0, minimo, media=0, pico;
    float kcr, pcr;
```

```

init_timer();
init_serial();

PINSEL1 = 0x00080000; //Habilita o D/A;
PINSEL1 |= 0x00400000; //Habilita o A/D;
IODIR0 = 0xFF;
IOCLR0 = 0xFF;

printf("\nCONTROLADOR PID\n");
zerar_valores();
wait(10000);
printf("Obtendo parametros de ajuste");

while(media<.45 || (maximo-minimo)<.47)
{
zerar_valores();
wait(4000);
rele = rele + .05;
aquisicao_planta(0);
maximo = achar_maximo_minimo(0);
minimo = achar_maximo_minimo(1);
media = (maximo+minimo)/2;
printf(".");
}

kcr = ganho_critico(maximo, minimo);
pcr = periodo_critico(maximo, minimo);
kcr = 0.447117;
pcr = 3.28;
printf("\nParametros de ajuste calculados:");
printf("\nGanho Critico = %f\n",kcr);
printf("\nPeriodo = %f\n",pcr);
ziegler_nichols(kcr, pcr, 2);

printf("\n\n\n*****\n\n");
printf("\n\nOscilacao\n\n");
for(i=0;i<4500;i++)
{
printf("%2.3f\n", last_value[i]);
}

printf("\n\n\n*****\n\n");
printf("\n\n\nAquisicao Ganho 1\n\n");
zerar_valores();
wait(10000);
aquisicao_planta(1);

for(i=0;i<4500;i++)

```

```

{
    printf("%.2f\n", last_value[i]);
}

printf("\n\n*****\n\n");
printf("\n\nAquisicao Final\n\n");
zerar_valores();
wait(10000);
aquisicao_planta(2);

for(i=0;i<4500;i++)
{
    printf("%.2f\n", last_value[i]);
}
}

float ziegler_nichols(float kcr, float pcr, int alternativa)
{
    float kp, ki, kd, ti=0, td=0;

    switch(alternativa)
    {
        case 0:    kp = 0.5 * kcr;
                   ki = 0;
                   kd = 0;
                   break;
        case 1:    kp = 0.4 * kcr;
                   ki = (0.25 * kcr * t) / pcr;
                   kd = 0;
                   ti = 0.8 * pcr;
                   break;
        case 2:    kp = 0.6 * kcr;
                   ki = (0.6 * kcr * t) / pcr;
                   kd = (0.075 * kcr * pcr) / t;
                   ti = 0.5 * pcr;
                   td = 0.125 * pcr;
                   break;
    }

    printf("\nKp = %f\nTi = %f\nTd = %f",kp,ti,td);
    a0 = kp + ki + kd;
    a1 = ki - kp - (2 * kd);
    a2 = kd;

    b0 = kd * kp;
    b1 = kp;
    b2 = kp * ki;

```

```

        return(0);
    }

float ganho_critico(float maximo, float minimo)
{
    float gcr, amplitude;

    amplitude = maximo - minimo;
    gcr = (4 * rele)/(3.1416 * amplitude);
    return(gcr);
}

float aquisicao_planta(int opcao)
{
    int i;
    float ad;

    for(i=0;i<4500;i++)
    {
        IOSET0 = 0XFF;
        IOCLR0 = 0XFF;
        wait(5);
        ad = adc();
        pid(ad,opcao);
        dac();
        last_value[i] = ad;
    }
    return(0);
}

float achar_maximo_minimo(int option)
{
    int i;
    float maximo_minimo;
    float variavel = 0;
    float variavel1 = 0;
    float variavel2 = 3.2;

    switch(option)
    {
    case 0:
        for(i=0;i<4500;i++)
        {
            wait(2);
            variavel = last_value[i];

            if(variavel1<variavel)
            {
                wait(5);

```

```

                                variavel1 = variavel;
                                maximo_minimo = variavel;
                                }
                                }
                                break;
case 1:
    for(i=1000;i<4500;i++)
    {
        wait(2);
        variavel = last_value[i];
        if(variavel2>variavel)
        {
            wait(5);
            variavel2 = variavel;
            maximo_minimo = variavel;
        }
    }
    break;
}
return(maximo_minimo);
}

```

```

float periodo_critico(float maximo,float minimo)
{

```

```

    int i = 0, j = 0, k;
    int tempo[5];
    float tempo_medio = 0;
    float variavel = 0;
    float variavel1 = 3.2;
    float media, maximo1;

    media = (maximo + minimo) / 2;
    maximo1 = maximo*.95;
    tempo[3] = 0;

```

```

    for(k=0;k<3;k++)
    {
        while(variavel<=maximo1)
        {
            variavel = last_value[j];
            j++;
            wait(2);
        }
        j = j + 10;

        while(variavel1>=media)
        {
            variavel1 = last_value[j];
            j++;
            wait(2);
        }
    }
}

```

```

    }

    variavel1 = 0;
    j = j + 10;

    while(variavel1 < media)
    {
        variavel1 = last_value[j];
        i++;
        j++;
        wait(2);
    }
    variavel1 = 3.2;
    j = j + 10;

    while(variavel1 >= media)
    {
        variavel1 = last_value[j];
        i++;
        j++;
        wait(2);
    }
    tempo[k] = i + 20;
    variavel = 0;
    variavel1 = 3.2;
}

tempo[3] = tempo[0] + tempo[1] + tempo[2];
tempo_medio = ((float)tempo[0] * t) / 4;
return(tempo_medio);
}

float adc(void) // Conversão A/D;
{
    unsigned int valor;
    float val;

    AD0CR = 0x00200400;
    AD0CR |= 0x01000000;

    do
    {
        valor = AD0DR;
    } while((valor & 0x80000000) == 0);

    valor = (valor >> 6) & 0x03ff; // Valor = número de bits do A/D;
    val = ((float)valor / 1024) * 3.24; // Cálculo para transformar bits em
    valor de tensão
    return(val); // Retorna o valor real da tensão;
}

```



```

void dac(void)
{
    unsigned long da;
    float resp;

    resp = atuar;
    resp = (resp / 3.24) * 1024;
    da = ((unsigned long)resp)<<6;
    da = da&0x0000FFC0;
    DACR = da;
}

void pid(float valor, int op)
{
    atuacao[1]=atuacao[0];
    erro[2]=erro[1];
    erro[1]=erro[0];
    erro[0]=1-valor;

    switch(op)
    {
    case 0:      if(erro[0]>.8)
                  {
                      atuacao[0]=rele;
                  }
                  else
                  {
                      if(erro[0]<.3)
                      {
                          atuacao[0]=0;
                      }
                      else
                      {
                          atuacao[0]=atuacao[1];
                      }
                  }
                  break;
    case 1:      atuacao[0] = erro[0];
                  break;
    case 2:      atuacao[0] = (a0 * erro[0]) + (a1 * erro[1]) + (a2 * erro[2]) +
atuacao[1];
                  break;
    }

    if(atuacao[0] >= 3.24)
    {

```

```

        atuar = 3.2;
    }
    else
    {
        if(atuacao[0]<=0)
        {
            atuar = 0;
        }
        else
        {
            atuar = atuacao[0];
        }
    }
}

```

```

void zerar_valores(void)
{
    DACR = 0x00000000;
    atuacao[0]=0;
    atuacao[1]=0;
    erro[0]=0;
    erro[1]=0;
    erro[2]=0;
    DACR = 0x00000000;
}

```

```

void wait (int j)                //Quando j=1, o tempo será 1 ms;
{
    unsigned long i;

    i = timeval;
    while ((i + j) != timeval);
}

```

### **SERIAL.C:**

```
#pragma INTERWORK
```

```

#include <LPC213x.H>                /* LPC21xx definitions      */
#define CR    0x0D

void init_serial (void)             /* Initialize Serial Interface */
{
    PINSEL0 = 0x00050000;           /* Enable RxD1 and TxD1      */
    U1LCR = 0x83;                   /* 8 bits, no Parity, 1 Stop bit */
    U1DLL = 97;                     /* (97)9600 Baud Rate @ 15MHz VPB Clock */
}

```

```

U1LCR = 0x03;                /* DLAB = 0 */
}

int putchar (int ch)          /* Write character to Serial Port */
{
    if (ch == '\n')
    {
        while (!(U1LSR & 0x20));
        U1THR = CR;           /* output CR */
    }
    while (!(U1LSR & 0x20));
    return (U1THR = ch);
}

int getchar (void)           /* Read character from Serial Port */
{
    while (!(U1LSR & 0x01));

    return (U1RBR);
}

```

### **TIMER.C**

```

#include <LPC213X.H>           // LPC21XX Peripheral Registers
#include "Timer.h"

long volatile timeval;

/* Timer Counter 0 Interrupt executes each 10ms @ 60 MHz CPU Clock */
void tc0 (void) __irq {
    ++timeval;
    TOIR = 1;                  // Clear interrupt flag
    VICVectAddr = 0;           // Acknowledge Interrupt
}

/* Setup the Timer Counter 0 Interrupt */
void init_timer (void) {
    TOMR0 = 14999;             // 1mSec = 15.000-1 counts
    TOMCR = 3;                 // Interrupt and Reset on MR0
    TOTCR = 1;                 // Timer0 Enable
    VICVectAddr0 = (unsigned long)tc0; // set interrupt vector in 0
    VICVectCntl0 = 0x20 | 4;   // use it for Timer 0 Interrupt
    VICIntEnable = 0x00000010; // Enable Timer0 Interrupt
}

```