UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CURITIBA

GERÊNCIA DE PESQUISA E PÓS-GRADUAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E INFORMÁTICA INDUSTRIAL - CPGEI

JULIANO DE MELLO PEDROSO

PROPOSTA DE UTILIZAÇÃO DO SISTEMA OPERACIONAL WINDOWS CE PARA AAPLICAÇÕES DIDÁTICAS NA AREA DE AUTOMAÇÃO E CONTROLE

DISSERTAÇÃO DE MESTRADO

CURITIBA
JUNHO -2007.

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial

---

DISSERTAÇÃO
apresentada à Universidade Tecnológica Federal do Paraná
para obtenção do grau de

MESTRE EM CIÊNCIAS

por

JULIANO DE MELLO PEDROSO

PROPOSTA DE UTILIZAÇÃO DO SISTEMA OPERACIONAL WINDOWS CE PARA APLICAÇÕES DIDÁTICAS NA ÁREA DE CONTROLE E AUTOMAÇÃO

Banca Examinadora:

Presidente e Orientador:

PROF. DR. CARLOS RAIMUNDO ERIG LIMA	**UTFPR**

Examinadores:

PROF. DR. JOÃO MAURÍCIO ROSÁRIO	**UTFPR**
PROF. DR. HEITOR SILVÉRIO LOPES	**UNICAMP**

Curitiba, junho de 2007.

JULIANO DE MELLO PEDROSO

PROPOSTA DE UTILIZAÇÃO DO SISTEMA OPERACIONAL WINDOWS CE PARA APLICAÇÕES DIDÁTICAS NA ÁREA DE CONTROLE E AUTOMAÇÃO

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná como requisito parcial para a obtenção do grau de "Mestre em Ciências" – Área de Concentração: Informática Industrial.

Orientador: Prof. Dr. Carlos Raimundo Erig Lima

Curitiba

2007

Ficha catalográfica elaborada pela Biblioteca da UTFPR – Campus Curitiba

P372 Pedroso, Juliano de Mello
Proposta de utilização do sistema operacional Windows CE para aplicações didáticas na área de automação e controle / Juliano de Mello Pedroso. – Curitiba : [s.n.], 2007.
xv, 136 f. : il. ; 30 cm

Orientador : Prof. Dr. Carlos Raimundo Erig Lima
Dissertação (Mestrado) – UTFPR. Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial. Curitiba, 2007.
Bibliografia : f. 96-101

1. Sistemas operacionais (Computadores). 2. Windows (Programas de computador). 3. Sistemas embutidos de computador. 4. Software - Desenvolvimento. 5. Controladores programáveis. 6. Processamento eletrônico de dados em tempo real. 7. Engenharia elétrica. I. Lima, Carlos Raimundo Erig, orient. II. Universidade Tecnológica Federal do Paraná. Curso de Pós-Graduação em Engenharia Elétrica e Informática Industrial. III. Título.

CDD : 005.43

AGRADECIMENTOS

Sem o apoio de incontáveis pessoas e algumas instituições, a realização deste trabalho não seria possível, portanto é com muita satisfação que dedico estes agradecimentos:

Ao Prof. Dr. Carlos Raimundo Erig Lima, pela orientação além do apoio e dedicação que demonstrou ao longo do desenvolvimento dessa dissertação.

Aos meus pais, Raul Pontes Pedroso e Raquiel de Mello Pedroso, meu Avô, Aécio Carneiro de Mello, que sempre acreditaram em mim.

A Eliane Silva Custódio, pelo apoio incondicional.

Eduardo Luiz Soppa, por todo tempo que dispensou em me ajudar programar e a resolver problemas com C++.

Ao Miguel Igino Valentini, Carlos Sakiti Kuriyama, Marco Túlio Siqueira, Felipe Cys Laskoski, Prof. Ms Glacy Duarte Arruda.

A Prof. Dra Faimara do Rocio Strauhs, pela ajuda nas correções e pelas palavras de apoio.

Às Instituições de Ensino: UTFPR, SENAI, enfim, a todos que contribuíram de alguma forma e, um agradecimento especial a Deus.

O meu Muito Obrigado!

Juliano

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

.DLL	Extensão (Dynamically Linked Library)
.EXE	Extensão de Arquivo Executável
.LIB	Extensão (Library)
A/D	Analógico/Digital
API	*Application Programming Interface*
ARM	*Advanced RISC Machine*
ARP	*Address Resolution Protocol*
ASIC	*application-specific integrated circuit*
ATL	*Active Template Library*
ATM	*Asynchronous Transfer Mode*
BSP	*Board Support Packages*
CAN	*Controller Area Network*
CC	*Corrente Contínua*
CD	*Compact Disk*
CLB	*Configuration Logical Blocks*
CLP	Controlador Lógico Programável
CO	*Control Output*
COM	*Component Object Model*
CPU	*Central Processing Unit*
D/A	Digital / Analógico
DHCP	*Dynamic Host Configuration Protocol*
DIC	Diagrama de Contatos
DIL	Linguagem de blocos lógicos
DRAM	*Dinamic RAM*
DVD	*Digital Video Disc*
EDIF	*Eletronic Design Interchange Format*
EEPROM	*Electrically Erasable Programmable Read-Only Memory*
FAT	*File Alocation Table*
FPGA	*Field-programmable Gate Array*
FRAM	*Ferromagnetic RAM*
FTP	*File Transfer Protocol*

FTP	*File Transfer Protocol*
GWES	*Graphics Windowing and events subsystem*
HD	*Hard Disk*
http	*HyperText Transfer Protocol*
I/O	*Input/Output*
IGMP	*Internet Group Management Protocol*
IHM	Interface Homem Máquina
IOB	*Input Output Blocks*
IP	*Intelectual Property*
IRDA	*Infrared Data Association*
ISR	*Interrupt Service Routines*
LCD	*Liquid Crystal Display*
LDAP	*Lightweight Directory Access Protocol*
LDAP	*Lightweight Directory Access Protocol*
LED	*Light Emitting Diode*
LIS	Lista de instruções
MB	*Mega Byte*
MFC	*Microsoft Foundation Classes*
MFC	*Microsoft Foundation Classes*
MMU	*Memory Management unit*
MSMQ	*Message Queuing*
MTS	*Microsoft Transation Server*
MV	Variavel Manipulada
NA	Normalmente aberto
NF	Normalmente fechado
NTFS	*New Tecnology File System*
NTLM	*New Technology LAN Manager*
OBEX	*Object exchange Protocol*
OLE	*Object Linking and Embedded*
OMAC	*Open, Modular, Architecture Control*
OOP	Programação orientada a objeto
PC	*Personal Computer*
PCI	*Peripheral Component Interconnect*

PDA	*Personal Digital Assistants*
PID	Proporcional Integral e derivativo
PID	*Proporcional, Integral,Derivativo*
POOM	*Pocket Outlook Object Model*
PPP/SLIP	*Point-to-point protocol/ Serial Line IP*
PROM	*Programmable Read-only Memory*
PV	Variavel de Processo
PWM	*Pulse-width Modulation*
RAM	*Random Acess Memory*
ROM	*Read Only Memory*
RPC	Chamadas de Procedimento Remoto
SGML	*Standard Generalized Markup Language*
SNMP	*Simple Network Management Protocol*
SO	Sistema Operacional
SOAP	*Simple Object Access Protocol*
SOTR	Sistemas Operacionais de Tempo Real
SP	*Set-point*
SRAM	*Static RAM*
SSPI	*Security Support Provider Interface*
TCP	*Transmission Control Protocol*
TCP/IP	*Transfer Control Protocol / Internet Protocol*
UDDI	*Universal Description, Discovery e Integration*
UDP	*User Datagram Protocol*
USB	*Universal Serial Bus*
VCC	Tensão continua
VGA	*Vídeo Graphics Array*
VHDL	*VHSIC Hardware Description Language*
WSDL	*Web Services Description Language*
XML	*Extensive Markup Language*

RESUMO

Com a crescente demanda por maior conectividade, acesso a Internet e funções multimídia, hoje demanda-se dos dispositivos dedicados maior sofisticação, incluindo novas funções e, muitas vezes, displays de vídeo para uma melhor experiência por parte do usuário final. A implementação de forma microcontrolada de todas essas novas funcionalidades seria bastante difícil e, em alguns casos, até mesmo inviável em função da memória máxima endereçada por microcontroladores. Os fabricantes de dispositivos resolveram a limitação dos ambientes microcontrolados buscando soluções microprocessadas, com sistemas operacionais modularizados e robustos, atendendo assim a nova demanda de mercado. Esta dissertação aborda conceitos relativos a sistemas embarcados (Embedded Systems), discorrendo sobre sua definição, principais aplicações do sistema operacional embarcado Windows CE na área de automação e controle, essas aplicações voltadas para a escola e a indústria. Este trabalho descreve a implementação de uma proposta de aplicação do Windows CE em um ambiente didático. O ambiente descrito visa atender à demanda por um sistema aberto, de fácil adaptação, custo adequado, com grande capacidade de processamento encontrada principalmente no meio acadêmico, mas extensível a aplicações industriais.

ABSTRACT

With the increasing demand for bigger conectividade, access the Internet and functions multimedia, today demand of the dedicated devices bigger sophistication, including new functions and, many times, displays of video for one better experience on the part of the final user. The implementation of microcontrolled form of all these new functionalities would be sufficiently difficult e, in some cases, even though impracticable in function of the addressed maximum memory for microcontrollers. The manufacturers of devices had decided the limitation of microcontrolled environments searching solutions microprocessed, with modularizados and robust operational systems, thus taking care of the new demand of market. This dissertação approaches concepts relative the systems embarked (Embedded Systems), discoursing on its definition, main applications of the operational system embarked Windows CE in the automation area and control, these applications come back toward the school and the industry. This work describes the implementation of a proposal of application of Windows CE in a didactic environment. The described environment aims at to take care of to the demand for an opened system, of easy adaptation, adjusted cost, with great capacity of joined processing mainly in the half academic, but extensible the industrial applications.

CAPÍTULO 1
INTRODUÇÃO

A utilização de sistemas operacionais embarcados é de fundamental importância para o funcionamento de vários equipamentos da vida moderna. Eles são encontrados nos mais variados dispositivos e sistemas, desde simples brinquedos até equipamentos de última geração da indústria eletro-eletrônica. Em geral, qualquer novo sistema ou produto que possui a característica de funcionar automaticamente apresenta um sistema operacional embarcado, controlando e gerenciando o funcionamento e o desempenho dos componentes e dispositivos envolvidos. Denominam-se Sistemas Embarcados (*Embedded Systems*) equipamentos ou dispositivos que utilizam microprocessadores e *softwares* para o controle dos mesmos. Estes equipamentos tornam-se cada vez mais sofisticados, demandando complexidade crescente no seu *hardware* e *software* embarcado (ORTIZ, 2001). Dai à importância de um estudo aprofundado das tecnologias embarcadas.

Reforçando esta importância pode-se levar em conta que são fabricados anualmente mais de dois bilhões de microprocessadores no mundo. Entretanto, somente entre 5% e 10% são utilizados em microcomputadores pessoais. O restante é utilizado em equipamentos e dispositivos que podem ser definidos como sistemas embarcados. Eles estão em todo lugar, nos telefones celulares, caixas automáticos, veículos, aparelhos de DVD, agendas eletrônicas, PDAs, consoles de jogos, fornos microondas, sistemas automotivos inteligentes, automação industrial, equipamentos médicos, sistemas de segurança e até mesmo brinquedos sofisticados (CARRO; WAGNER, 2002; TAURION, 2005).

A grande maioria destes equipamentos demanda *softwares* de gerenciamento e controle ou mesmo sistemas operacionais dedicados. Os segmentos que mais demandam uso destes *softwares* são telecomunicações (35%), eletrônica de consumo (20%), automação industrial (19%) e automotiva (10%), além de sistemas médicos e aeroespaciais (CARRO; WAGNER, 2002; TAURION, 2005).

Segundo Carro e Wagner (2002), corroborado por Taurion (2005)

> A grande maioria destes equipamentos demanda *softwares* de gerenciamento e controle ou mesmo sistemas operacionais dedicados. Os segmentos que mais demandam uso destes *softwares* são telecomunicações (35%), eletrônica de consumo (20%), automação industrial (19%) e automotiva (10%), além de sistemas médicos e aeroespaciais.

Dentre os vários sistemas operacionais dedicados a sistemas embarcados o Windows CE (MICROSOFT, 2007) tem-se apresentado como uma alternativa atraente e relativamente

pouco explorada no ambiente de ensino e pesquisa, particularmente no Brasil.

Este trabalho descreve um ambiente baseado no sistema operacional Windows CE aplicado ao ensino e estudo de problemas de automação e controle, em especial às tarefas que demandem um desempenho em tempo real ou ainda uma fácil adaptação às novas demandas de *hardware* ou *software*.

A proposta descrita visa atender à demanda por um sistema aberto, de fácil adaptação, custo adequado, com grande capacidade de processamento encontrada principalmente no meio acadêmico, mas extensível às aplicações industriais. Isto vem de encontro à rápida evolução das inovações tecnológicas de *hardware* e de *software* na área de sistemas embarcados, em especial na automação e controle, onde se torna necessário desenvolver aplicações baseadas em metodologia que levem em conta a facilidade de futuras modificações, modernizações e expansões do sistema originalmente projetado. Tem-se como público-alvo educadores da área de informática, professores e laboratórios de informática.

Justifica-se a presente pesquisa por entender que a reforma de equipamentos e a atualização de aparelhos usados a preços reduzidos são estratégias que beneficiam as escolas, oferecendo uma infra-estrutura mais adequada às escolas que podem atender de forma personalizada à demanda das indústrias em constantes transformações. Outras justificativas são:

- Modernização com economia, onde os sistemas embarcados estão cada vez mais complexos e poderosos, necessitando de um sistema operacional capaz de gerenciar tais recursos.
- Conectividade, hoje cada vez mais se exige dos sistemas embarcados a capacidade de se interconectar com outros equipamentos e dispositivos, utilizando meios e protocolos padronizados, tais como TCP/IP e *Bluetooth* (rede sem fio de curta distância).
- Ambiente Gráfico e Interface Homem-Máquina, também cada vez mais são exigidas interfaces mais complexas e fáceis de utilizar. O Windows CE fornece interface gráfica amigável em relação a outros sistemas operacionais tais como Linux.
- O Custo do Licenciamento de *Software*, preço reduzido, depois de pagar o preço de US$1000 pelo *Platform builder* o custo de um dólar por imagem criada. Este fator é particularmente importante para o desenvolvimento de produtos com baixo preço unitário, como aparelhos de MP3 e telefones IP.

- Configurabilidade: considera-se que o Windows, especialmente o *Kernel*, o núcleo do sistema, possibilita a própria configuração otimizada especificamente para cada aplicação.
- Ferramentas de Desenvolvimento, pois garante maior confiabilidade operacional o Windows CE que apresenta ferramentas de desenvolvimento em linguagem C++ e Java.

1.1 OBJETIVO GERAL

Estudar, implementar e analisar de uma proposta de aplicação do Windows CE em ambientes didáticos, particularmente no ensino e estudo de problemas de automação e controle, em especial às tarefas que demandem um desempenho em tempo real ou ainda uma fácil adaptação às novas demandas de *hardware* ou *software*.

1.1.1 Objetivos Específicos

- A proposta descrita visa demonstrar dois exemplos de uso do Windows CE. Um na área de controle e outro na área de automação, inclusive mostrando a implementação prática dos dois exemplos.
- Produzir e disponibilizar literatura em português sobre sistema operacional Windows CE.
- Estudar e ter domínio tecnológico sobre um sistema operacional embarcado.
- Aplicar os exemplos implementados em sala de aula.

1.2 ESTRUTURA DA DISSERTAÇÃO

Esta dissertação está organizada em cinco capítulos. No Capítulo 2 faz-se uma revisão da literatura sobre os principais conceitos sobre sistemas operacionais embarcados e Windows CE, bem como, sobre os tópicos teóricos. No Capítulo 3, descrevem-se em detalhes o desenvolvimento da metodologia proposta. No Capítulo 4, relatam-se os resultados obtidos. E, finalmente, o Capítulo 5 apresenta a discussão dos resultados, as conclusões do trabalho e as propostas de trabalhos futuros.

CAPÍTULO 2

FUNDAMENTAÇÃO TEÓRICA

2.1 SISTEMAS OPERACIONAIS

Um sistema operacional (SO) é responsável por alocar recursos de *hardware* e escalonar tarefas, permitindo o controle de recursos de uma arquitetura baseada em um ou mais microprocessadores. Este também deve prover uma interface para o usuário, uma maneira facilitada de acesso aos recursos do computador (OLIVEIRA, 2001).

Um sistema operacional pode ser definido como um gerenciador dos recursos que compõem o computador (processador, memória, I/O, arquivos). Os problemas centrais que um sistema operacional deve resolver são o compartilhamento de dados, a proteção dos recursos a serem usados pelas aplicações do usuário e o interfaceamento entre este e a máquina (OLIVEIRA, 2001). Portanto, se não existissem os sistemas operacionais, todo programa teria que ser configurado de forma a se comunicar com os periféricos.

Os sistemas operacionais são formados por um conjunto de programas e rotinas computacionais que têm como objetivo criar uma camada de abstração entre o usuário e o hardware propriamente dito. Entende-se por usuário todo e qualquer objeto que precise de acesso aos recursos de um computador (seja ele um usuário “real” ou um aplicativo).

Programas como *Shell* (este é um programa que recebe, interpreta e executa os comandos de usuário, aparecendo na tela como uma linha de comandos), editores de texto e compiladores não fazem parte do sistema operacional, apesar de serem normalmente oferecidos pelo fabricante do SO (TANENBAUM, 2003).

2.1.1 Estrutura de um Sistema Operacional

Pode-se criar um sistema tão grande e complexo como um sistema operacional somente dividindo-o em pequenas partes. Cada uma dessas partes deve ser uma porção bem delineada do sistema, com entradas, saídas e funções, cuidadosamente definidas. Logicamente, nem todos os sistemas têm a mesma estrutura, ou seja, não apresentam a mesma forma de ligação entre as partes. Contudo, os sistemas operacionais modernos geralmente possuem as seguintes partes:

- Gerenciamento de processos - criar e eliminar, suspender e retomar, sincronismo e comunicação entre processos;
- Gerenciamento da memória principal – manter o controle das partes da memória que estão sendo usadas e por quem, decidir que processos serão carregados para memória quando houver espaço disponível, alocar espaço de memória quando necessário;
- Gerenciamento de memória secundária – o SO é responsável pelas atividades de alocação de espaço livre.
- Gerenciamento de Entrada/Saída – manter os *device drivers* para comunicação com os deferentes dispositivos, um *buffer-caching* para o sistema;
- Gerenciamento de arquivos – criar e eliminar arquivos e diretórios, manter mapeamento dos arquivos em disco;
- Proteção do sistema – se um sistema é multiusuário e permitem múltiplos processos concorrentes, estes processos devem ser protegidos de outras atividades;
- *Networking* – em um sistema distribuído (fracamente acoplado) cada processador tem sua própria memória e seus processadores que se comunicam através do SO. A comunicação entre eles deve considerar roteamento e estratégias de conexão;
- Interpretador de comandos – um dos mais importantes programas do SO é o interpretador de comandos, que serve de interface entre o usuário e o SO. Alguns SO's incluem este programa no próprio núcleo (*kernel*). Já outros sistemas, como o DOS e o UNIX, tratam o interpretador de comandos como um programa especial que é executado quando uma sessão é iniciada. Com isso, um sistema operacional fornece um ambiente para execução, melhor dizendo, fornece serviços para os programas e também para os usuários desses programas (TANENBAUM, 2003).

2.1.2 Classificação dos Sistemas Operacionais

Os sistemas operacionais podem ser classificados de três maneiras: (i) pelo tipo do núcleo de sistema (*kernel*), (ii) pelo método adotado ao gerenciar os programas em execução ou (iii) pelo número de usuários que podem operá-lo simultaneamente.

Kernel é entendido como núcleo do sistema operacional. Ele representa a camada mais baixa de interface com o *hardware*, sendo responsável por gerenciar os recursos do sistema computacional como um todo. É no *kernel* que estão definidas funções para operação com periféricos (discos, impressoras, interfaces), gerenciamento de memória. O *kernel* é um

conjunto de programas que fornece para os aplicativos uma interface para utilizar os recursos do sistema (TANENBAUM, 2003).

i. Quanto ao tipo de *kernel* são utilizados sistemas operacionais basicamente de três tipos (TANENBAUM, 2003):

1. *Kernel* monolítico ou monobloco – tem como principal característica o fato de integrarem todas as funcionalidades possíveis do sistema em um grande bloco de software. A adição de novas funcionalidades implica na recompilação de todo o núcleo. Alguns exemplos desse tipo de *kernel* são: MS-DOS, Windows 95.
2. Micro*kernel* – é um termo usado para caracterizar um núcleo de sistema cujas funcionalidades não essenciais ao seu funcionamento são transferidas para servidores, que se comunica com o núcleo mínimo através do modo de acesso do núcleo (local onde o programa tem acesso a todas as instruções da CPU (Unidade Central de Processamento) e a todas as interrupções de *hardware*), deixando o máximo de recursos rodando no modo de acesso do usuário. Quando o processador trabalha no modo usuário, uma aplicação só pode executar instruções não privilegiadas, tendo acesso a um numero reduzido de instruções. São exemplos: *Hurd* e *Minix*.
3. Kernel híbrido – define um *kernel* baseado em micro*kernel* no qual: módulos externos a ele podem executar operações em modo *kernel* (protegido), a fim de evitar trocas de contexto e melhorar o desempenho geral do sistema.

ii. Os diversos tipos de sistemas operacionais existentes empregam diferentes maneiras de gerenciar os programas em execução pelo usuário. Existem basicamente os seguintes tipos de gerenciamento de tarefas (ou processos) (TANENBAUM, 2003):

1. Sistemas Operacionais Monotarefa – permitem a realização de apenas uma tarefa ou processo de cada vez. Um exemplo típico deste sistema é o MS-DOS.
2. Sistemas Operacionais Multitarefa – é a grande maioria dos sistemas operacionais. Dá-se o nome de multitarefa a característica dos sistemas operacionais modernos que permite repartir a utilização do processador entre várias tarefas simultaneamente.

3. Sistemas Operacionais Multitarefa Cooperativa – trabalha exatamente como dito anteriormente: o tempo de processamento é repartido entre as diversas tarefas, dando a impressão aos usuários que elas estão sendo executadas simultaneamente. Sua principal característica (ou deficiência) reside no fato de que não há controle sobre o tempo de CPU que cada processo consome. O sistema cede o controle da sobre o tempo de CPU que cada processo consome. O sistema cede o controle da CPU ao processo, e este só o devolve quando tiver terminado a sua tarefa.
4. Sistemas Operacionais Multitarefa preemptiva – realiza o gerenciamento do tempo de utilização da CPU de forma inteligente, reservando e protegendo o espaço de memória dos aplicativos evitando que programas com erros possam invadir as áreas delimitadas pelo sistema operacional. Os núcleos destes sistemas mantêm em memória um registro de todos os processos em execução através de uma árvore de processos. Entre outros atributos acerca de cada processo, a árvore de processos inclui as informações de prioridade, com a qual o núcleo calcula o tempo de CPU que deve dar a cada processo; quando esse tempo acaba, o núcleo tira do processo o controle da CPU e o passa ao processo que vem a seguir na fila. Quando a fila acaba, o núcleo volta a dar o controle da CPU ao primeiro processo, fechando assim o ciclo.

iii. Quanto à forma de execução dos programas são utilizados sistemas operacionais basicamente de três tipos (TANENBAUM, 2003):

1. Sistemas Operacionais do tipo Lote (*Batch*) – nestes sistemas os programas são agrupados fisicamente (colocados no mesmo arquivo) e processados seqüencialmente. Foram usados sistemas deste tipo na época dos *Mainframes* com cartão perfurado, onde muitas vezes o processador ficava subutilizado aguardando a leitora de cartões ou a impressora terminarem sua tarefa, pois Estes eram muito mais lentos que a CPU.
2. Sistemas operacionais do tipo *Time Sharing* – são sistemas onde o usuário tem a impressão de estar sendo atendido no instante em que se submete um comando ao computador. Estes sistemas são implementados, utilizando um relógio (*clock*) que provoca interrupções no processador a intervalos regulares

e a cada interrupção o processador alterna de processo (ou de usuário), causando a impressão de estar atendendo vários ao mesmo tempo

3. Sistemas operacionais de Tempo Real – são casos particulares de *time sharing*, onde se garante que o mínimo tempo entre um *scan* e outro processador por um processo é suficiente para evitar a perda de dados. São aplicados em sistemas de controle industriais. Ex: Um sistema computadorizado de regulagem de temperatura de uma fornalha que deve amostrar a temperatura do forno várias vezes num determinado período de tempo, cuidando para que a temperatura não sofra variação.

2.2 SISTEMAS EMBARCADOS

Os sistemas operacionais embarcados estão presentes em diversos setores da indústria atual. A demanda por equipamentos inteligentes e soluções dedicadas, capazes de apresentar resultados eficientes para os problemas cotidianos, transforma a utilização de microprocessadores e sistemas embarcados em uma parcela importante do mercado de computação. Desta forma, a demanda por sistemas operacionais embarcados capazes de orquestrar os novos dispositivos e equipamentos é crescente e irreversível (ORTIZ, 2001).

Um sistema embarcado é uma associação do *hardware* e do *software* de um computador, projetado para executar uma tarefa específica (BAR, 1999). Um grande número de equipamentos pode ser definido como sistemas embarcados, como, por exemplo, fornos de microondas, automóveis, impressoras a laser, *routers* e *switches* de rede, equipamentos médicos, robôs móveis (SIMÕES J. B. et al., 2000; SCATENA, 2003; DEHUAI; CUNXI; XUEMEI , 2005).

Os sistemas embarcados são geralmente projetados para uma aplicação específica, em oposição aos sistemas construídos para aplicações genéricas. Os computadores pessoais são exemplos de sistemas de uso geral, projetados para atender uma grande gama aplicações. São sistemas sujeitos a contínua atualização do *software* a ser executado. No caso de sistemas embarcados, tal característica não é observada. Tipicamente, uma vez programada uma aplicação específica, esta não sofrerá modificações ao longo da sua vida útil. Por exemplo, não se espera o melhoramento do *software* de uma máquina de lavar roupas após a sua venda. Contudo, com a evolução da complexidade dos sistemas embarcados, observa-se, cada vez mais, a necessidade de atualizações freqüentes dos aplicativos a serem executados nos

mesmos. Como o caso dos telefones celulares, que incorporam gradualmente diversas modificações no *software* executado.

Diversas funcionalidades são esperadas de um sistema operacional embarcado. Muitas destas funcionalidades são, inclusive, invisíveis ao usuário do sistema, sendo executadas sem a interferência do mesmo. Por exemplo, o gerenciamento de recursos de memória não é uma funcionalidade que um usuário tenha acesso, ou mesmo, ciência que esteja ocorrendo. Por outro lado, muitas funcionalidades em um sistema operacional embarcado demandam a interação com o usuário, como por exemplo, a entrada de dados para um determinado aplicativo (RIEBEEK, 2002).

Quanto à complexidade, os sistemas embarcados podem apresentar os mais diversos graus. Pode ser possível conceber sistemas, com ou sem sistema operacional, com alto grau de complexidade, como por exemplo, no trabalho de Simões et al. (2000), de Rosário et al. (2000), de Ilic et al. (2004) e de Li e Fang, 2005). Por outro lado, os trabalhos de Silva et al., (2001) e John (2002) são exemplos de sistemas embarcados de menor complexidade.

A seguir serão descritas características dos sistemas operacionais embarcados, essas características são importantes na escolha de um sistema operacional embarcado.

2.2.1 Características de um Sistema Operacional para Sistemas Embarcados

São características de um sistema operacional embarcado:

- Portabilidade

  Normalmente um sistema operacional desenvolvido para sistemas embarcados deve ser bastante portável, pois necessita de flexibilidade suficiente para ser utilizado em diferentes plataformas, trabalhando com diversos processadores como Alpha, ARM, i386, Motorola PowerPC, SPARC, Motorola 68k, Hitashi SH3, VAX, MIPS. De modo geral, estes sistemas são modulares para que possam ser configurados de acordo com as necessidades da aplicação específica (ROSA JUNIOR, 2004).

- Limite de consumo de potência

Quando se trabalha com um sistema embarcado, normalmente tem-se um quadro onde não há uma fonte de energia permanente para alimentar o sistema, utilizando-se baterias, por

exemplo. Sendo assim, o sistema operacional tem que utilizar o processador de modo otimizado para que não haja desperdício de energia (ROSA JUNIOR, 2004).

- Restrições Temporais

Normalmente, nos sistemas embarcados o resultado de um dado processo não depende apenas das respostas, mas também do instante em que esta foi dada. A principal conseqüência das restrições temporais incide sobre a política de escalonamento de tarefas. Uma tarefa nunca tem sua execução iniciada apenas porque está pronta para ser executada, como ocorre nos sistemas operacionais convencionais, mas sim porque tem prioridade máxima naquele dado instante (ROSA JUNIOR, 2004).

- Memória

Uma das grandes restrições que os sistemas operacionais embarcados enfrentam é a restrição de memória. O próprio sistema operacional pode ter um tamanho excessivo em aplicações onde a área ocupada na memória é um fator crítico (OLIVEIRA, 2001).
Problema crítico em Sistemas Embarcados, o sistema de memória tenta achar a melhor solução para ser ao mesmo tempo rápido e de grande capacidade. As memórias estáticas podem funcionar no mesmo ciclo de relógio da CPU, o que as torna rápidas (CARRO; WAGNER, 2002). O fato de memórias rápidas consumirem grande potência torna-se mais um ponto negativo para este problema. Em decorrência disto, surgiram as memórias *cache*, que eram rápidas, mas que possuíam uma capacidade limitada de armazenamento. A localidade temporal (uma posição visitada tem mais chance de ser visitada novamente que outra qualquer) e a localidade espacial (uma posição próxima de uma visitada recentemente tem mais chance de ser visitada do que outra qualquer) ocasionam o reuso de dados e instruções armazenadas, favorecendo o uso da *cache*. Novas tecnologias possibilitaram o surgimento de memórias de grande capacidade, as memórias dinâmicas, muito mais lentas que a CPU. A maioria dos processadores atuais utiliza memória *cache* (OLIVEIRA, 2001).

2.3 SISTEMAS OPERACIONAIS EM TEMPO REAL

A grande maioria dos sistemas embarcados não requer processadores com grande capacidade de processamento para atender suas necessidades, até porque estes processadores consomem muita energia e no ambiente embarcado pode-se não existir uma fonte de alimentação contínua. Utilizam-se normalmente processadores mais simples e robustos que atendam às necessidades de forma bem dimensionada, para que não haja desperdício de recursos em um ambiente onde eles normalmente são escassos. Pode-se desejar um processador com pouca capacidade de processamento, mas com grande economia de energia. Por outro lado, em algumas aplicações podem ser necessários requisitos de tempo real. Outros sistemas demandarão uma grande capacidade de armazenamento de dados. Assim, o sistema operacional embarcado está fortemente associado às funcionalidades desejadas para um determinado produto, bem como à arquitetura utilizada para implementá-lo (GOKHALE; SCHMIDT , 1999; OLIVEIRA, 2001; ORTIZ, 2001).

Normalmente o sistema operacional para sistemas embarcados utiliza muito das características dos sistemas operacionais de tempo real (SOTR). No entanto, isto não é uma regra, pois podem existir sistemas embarcados que não utilizam as características dos sistemas operacionais de tempo real. As características de tempo real são desejáveis no momento que se utilizam equipamentos e dispositivos que tenham a necessidade de atender a requisições com tempo máximo de resposta, muitas vezes disparadas por eventos externos (BARABANOV; YODAIKEN, 1996; ROSA JUNIOR, 2004).

Aplicações embarcadas com requisitos de tempo real estão se tornando cada vez mais comuns, principalmente em sistemas de controle de tráfego aéreo, de defesa militar, e de automação e controle de processos industriais. Na outra extremidade estão os sistemas mais simples, normalmente embarcados em utilidades domésticas que muitas vezes não necessitam de respostas em tempo real. (PAULIN et al., 1997; NETTER; BACELAR, 2001).

Os sistemas operacionais para sistemas embarcados precisam atender a outros requisitos além daqueles já exigidos para os SOTR. Eles devem ser escaláveis, isto é, não devem oferecer um conjunto completo de serviços de forma monolítica, mas sim como uma biblioteca de módulos, a serem facilmente selecionados no momento da geração da aplicação de acordo com as suas necessidades específicas. Estes sistemas também devem atender a restrições relacionadas ao projeto da aplicação, tais como quantidade de memória, restrições de desempenho e de consumo de energia (ORTIZ, 2001; KADIONIK, 2002; RASCHE; POLZE, 2002).

Define-se *kernel* de um sistema operacional como o núcleo do sistema operacional, contendo as funções básicas para operação do mesmo (ROSA JUNIOR, 2004). Há duas variedades de *kernels* de tempo real para sistemas embarcados: (i) *kernels homegrown* (altamente especializados para uma aplicação) e (ii) comerciais. Ambos são muito rápidos, têm alta previsibilidade e são customizados. O alto custo de desenvolver e de manter um *kernel homegrown*, aliado ao fato dos *kernels* comerciais possuírem alta qualidade, estão desestimulando o surgimento de novos *kernels homegrown*. Para tornar estes *kernels* rápidos e reduzir sobrecarga em tempo de execução, utilizam-se diversas estratégias, tais como: mudança rápida de contexto, tamanho pequeno (funcionalidades mínimas), respostas rápidas a interrupções externas, minimização do intervalo no qual uma interrupção é desabilitada, prover partições de tamanho fixo ou variável para administração de arquivos e provimento de arquivos seqüenciais que podem acumular dados a uma taxa rápida. Os requisitos citados levam o *kernel* a prover tempo de execução para a maioria das primitivas, manterem um relógio de tempo real, prover um mecanismo de escalonamento de tempo real, *timeouts*, suporte para política de escalonamento de processos, tais como *deadline* mais cedo e primitivas para alocar uma mensagem no início de uma fila de dados (ORTIZ, 2001).

Em geral, estes *kernels* são multitarefa e a comunicação e sincronização entre estas tarefas são feitas a partir de eventos, sinais e semáforos.

Sistemas de tempo real são sistemas que apresentam restrições temporais sérias. A adequação do comportamento destes sistemas não depende apenas da integridade dos resultados obtidos (*correctness*), mas também do cumprimento dos prazos em que as tarefas que implementam a funcionalidade do sistema são executadas (*timeliness*) (HERSHENSON, 2002).

Um dos conceitos mais importantes em um sistema de tempo real é a previsibilidade. Um sistema é dito previsível quando se pode ter segurança sobre o comportamento do sistema, antes mesmo dele entrar em execução. Esta previsibilidade permite garantir o cumprimento das restrições temporais e também uma alocação de recursos adequada e eficiente. Porém, para que se possa prever o comportamento de um sistema como este, é necessário levar em conta as situações mais críticas do sistema da maneira mais realista possível, o que não é uma tarefa trivial. Além das situações de falhas e carga computacional que são inerentes à aplicação, também tem outros fatores que inserem não-determinismo às aplicações. Entre estes fatores pode-se citar: as linguagens de programação e a arquitetura do *hardware*. “A previsibilidade é o requisito necessário que deve ser introduzido em paradigmas

de computação clássico para se poder atender aplicações de tempo real, em particular quando estas apresentam requisitos temporais rígidos.” (HERSHENSON, 2002).

2.4 SISTEMAS OPERACIONAIS EM LÓGICA RECONFIGURÁVEL POR *HARDWARE*

Uma das idéias deste trabalho era a construção de uma imagem do sistema operacional Windows CE que pudesse ser implementada em uma FPGA (Field Programmable Gate Arrays), mas atualmente os fabricantes de FPGAs não disponibilizam um processador embarcado com MMU (Memory Management Unit - é um bloco de hardware que transforma endereços virtuais em endereços físicos) associado, inviabilizando esta alternativa de implementação sem o desenvolvimento de um processador embarcado com esta característica específica.

2.4.1 *Field Programmable Gate Arrays* - FPGA

O *Field Programmable Gate Arrays* (FPGA) é um circuito integrado passível de ser configurado por *software* e servem para programar circuitos digitais, como processadores, interfaces, controladores e decodificadores. Basicamente, consiste de um arranjo fortemente condensado de blocos idênticos de pequenos circuitos, compostos por algumas portas lógicas e *flip-flops*, com alguns sinais de interface. As conexões entre as saídas de determinados blocos com as entradas de outros são programáveis via um protocolo simples e de fácil implementação.

FPGAs podem ser utilizados para a implementação de praticamente qualquer *design* de *hardware*. Um dos usos mais comuns é o seu uso para a prototipação de componentes que virão no futuro a transformar-se em componentes prontos, em pastilhas de silício (ASIC). No entanto, não existem problemas em se distribuir FPGAs em produtos finais. Esta decisão requer basicamente uma análise de custo, uma vez que o desenvolvimento de um ASIC é um processo bastante caro e inflexível, mas que gera componentes de custos bastante reduzidos se produzidos em larga escala. Já os FPGAS possuem flexibilidade e custo baixo de prototipação, mas preços finais pouco competitivos se comparados à ASIC com alta escala de produção (ITO, 2000; CORIC; LEESER; MILLER, 2002; DIDO et al, 2002).

2.4.2 Projeto e Desenvolvimento de *Hardware*

Uma estrutura de descrição de *hardware* é escrita em uma linguagem de alto nível (usualmente VHDL e *Verilog*), o código é compilado e copiado para ser executado. Descrever o circuito como um esquemático digital é também possível, mas isso é bem menos popular e mais complexo que utilizar ferramentas baseadas em linguagens. Nota-se que a principal diferença entre o *design* de *hardware* e *software* é a maneira que o desenvolvedor precisa pensar para resolver um problema. Desenvolvedores de *software* tendem a pensar seqüencialmente, mesmo quando estão desenvolvendo aplicações multitarefas (*multitreaded*). As linhas de código sempre são escritas para serem executadas em uma ordem, pelo menos dentro de uma tarefa (*thread*) em particular. Mesmo havendo um sistema operacional usado para criar a aparência de paralelismo, ainda há um núcleo de execução para controle disso. Durante o projeto de *hardware*, os *designers* precisam pensar, e programar, em paralelo. Todos os sinais são processados em paralelo, pois trafegam através de um caminho de execução próprio – cada uma da série de macrocélulas e interconexões – até o destino do sinal de saída. Dessa forma, a descrição do *hardware* cria estruturas que podem ser "executadas" todas ao mesmo tempo (CORIC; LEESER; MILLER, 2002; DIDO et al, 2002).

Cada vez mais existem também depuradores (*debuggers*) disponíveis que permitem a execução passo a passo da execução em dispositivos lógico programáveis. Obviamente este tipo de integração específica para componentes requer um conhecimento amplo sobre o funcionamento dos *chips*, normalmente fornecidos em parceria da empresa desenvolvedora da ferramenta com a produtora do componente (VARGAS, 2001).

As ferramentas mais populares para compilação e simulação VHDL e *Verilog* são vendidas por terceiros. Estas ferramentas geralmente possuem suporte a uma lista de FPGAs. Isso significa que as ferramentas precisam compreender o *chip* em questão e também conter informações de sincronismo relacionadas à sua ferramenta de posicionamento e roteamento (*place and route tool*) (CORIC; LEESER; MILLER, 2002; DIDO et al. 2002; ALTERA, 2006).

2.5 WINDOWS CE

Windows CE (de Compact Edition, às vezes abreviado para WinCE), é uma versão da popular linha de sistemas operativos Windows para dispositivos portáteis e Tablet PCs. Ele

equipa desde mini-computadores até telefones celulares e o videogame Dreamcast. É suportado no Intel x86 e compatíveis, MIPS, ARM, e processadores SuperH Hitachi.

2.5.1 Histórico

A Microsoft começou a demonstrar interesse em desenvolver um sistema operacional voltado para sistemas embarcados em 1992 quando inaugurou um projeto conhecido como *WinPad*. Dois anos depois a Microsoft já havia feito parcerias com sete empresas entre elas Compaq, Motorola, NEC e Sharp, que mais tarde tornar-se-iam, líderes no mercado de *Handhelds*.

No mesmo ano, o projeto foi cancelado por ser considerado a frente do seu tempo. A tecnologia da época não permitia a criação de dispositivos de *hardware* capazes de atender os requisitos do sistema operacional com baixo custo e pequeno consumo de energia, aspectos fundamentais quando se fala em sistemas embarcados.

Em paralelo, outro grupo na Microsoft denominado Pulsar estudava a criação de um dispositivo móvel multifuncional que fosse capaz de ser utilizado com poucos botões e que apresentasse configuração de *hardware* totalmente diferente.

Porém, a empresa estava certa de que o caminho que estava seguindo estava correto apesar das dificuldades encontradas por ambos os projetos. O que fizeram neste momento em diante foi tomar as lições aprendidas com os insucessos dos projetos anteriores e unir as duas equipes em uma nova equipe de desenvolvimento denominado Pegasus.

Em setembro de 1996, 21 meses depois de sua criação, a equipe Pegasus apresentou a primeira versão do Windows CE. Seis empresas (Casio, Compaq, LG Electronics, HP, NEC e Philips) começaram a produzir dispositivos móveis do tipo *Handheld* baseados nesta tecnologia.

Porém, a vida do Windows CE 1.0 foi curta. Em 29 de setembro de 1997, 12 meses depois do lançamento da primeira versão, a Microsoft anunciou o lançamento da versão 2.0 do produto.

Com o lançamento da versão 2.0 do Windows CE, o sistema operacional poderia ser configurado de diferentes formas permitindo, assim, ser utilizado em diversos outros dispositivos além de *handheld*s. O Windows CE agora poderia ser incorporado a ATMs, carros e até em eletrodomésticos. Outra característica importante da segunda versão foi a possibilidade de ser utilizado em diversos processadores de baixo consumo. Os mais utilizados na época pelos fabricantes dos dispositivos eram o MIPS e o SH3.

A versão 2.1 incorporou mais algumas funcionalidades, como suporte ao sistema de arquivos FAT-32, o controlador USB além de outras.

A partir do Windows CE 3.0, a Microsoft concentrou-se em encontrar meios de competir com o já consolidado mercado dos *Palms* e o seu sistema operacional *Palm OS*. Esta decisão iria dar origem mais tarde aos *Pocket PC's*.

Atualmente, o Windows CE encontra-se na versão 6.0. Uma das grandes vantagens do desenvolvimento deste sistema operacional são as ferramentas disponíveis. A partir do *Plataform Builder*, é possível configurar o sistema operacional para diversas aplicações utilizando *wizards*. Este tipo de auxílio faz com que o desenvolvimento na plataforma torne-se rápido, diminuindo consideravelmente os custos de desenvolvimento. Estes *wizards* estão disponíveis para as mais diversas aplicações (HPCFACTOR, 2006; MICROSOFT, 2006).

2.5.2 Exemplos de uso do Windows CE em aplicações embarcadas

Inúmeros trabalhos associados ao sistema operacional Windows CE podem ser encontrados na literatura. No trabalho de Huan et al. (2005) o Windows CE é utilizado para validação de modelos de controle de acesso em sistemas embarcados. Os autores Junju e Biquin (2002) implementam uma interface de comunicação *Bluetooth FTP* no Windows CE. Uma interessante aplicação na educação pode ser vista no trabalho de P. Ibach et al. (2005), onde robôs móveis de baixo custo são controlados com o auxílio deste sistema operacional. Uma recente comparação entre o Windows CE e outros sistemas operacionais com suporte para tempo real comerciais é feita em Santo (2005). Em Simões et al. (2000) o Windows CE é usado no desenvolvimento de um espectrômetro de massa digital. O trabalho de Fang et al.(2005) apresenta as vantagens deste sistema para coleta de dados. Uma aplicação do Windows CE em sistemas reconfiguráveis é apresentada em George e Wong (2004), sendo um exemplo de importante da versatilidade deste sistema operacional.

2.5.3 Filosofia

A Microsoft sempre defendeu a filosofia que os utilizadores quereriam funcionalidades próximas daquelas dos computadores *desktop* nos seus aparelhos portáteis. Isto tornou o Windows CE uma versão "simplificada" das versões *desktop*, que lhe traz vantagem; na similaridade entre funções e menus o Windows CE segue como os seus

"irmãos", um estilo semelhante ao Windows: Menu Iniciar, diretórios e preferências semelhantes.

Esta filosofia seguida pelo Windows CE é exatamente contrária à do seu maior concorrente, o *PalmOS*. O *Palm* é dedicado a executar tarefas simples como manter listas de contatos, listas de tarefas e reuniões. Embora o Windows CE tenha estilo semelhante às versões *desktop*, o *Palm* é mais simples de usar, pois simplifica as funções essenciais para as quais está orientado a cumprir, ao contrário do Windows CE que traz a maioria das funções do desktop. Alguns dos defeitos apontados ao Windows CE foram corrigidos quando do lançamento das *shell's Pocket* PC 2000 e 2002 (HPFACTOR, 2006; MICROSOFT, 2006).

2.5.4 Características

As características do *software*: compacto, alta velocidade em relação a outros sistemas operacionais, interface gráfica e baixo custo da licença (um dólar por imagem construída) o qualificam a entrar no mercado de automação industrial.

Foram então definidas características para o Windows CE e seguido um conjunto de princípios de projeto que permitiram criar um sistema operacional flexível de forma a ser utilizável num grande leque de dispositivos inteligentes, não só em PDA's como também em outros sistemas embarcados como, por exemplo, eletrodomésticos, sistemas de computação instalados em carros, terminais de comunicação e sistemas automáticos.

Como características do Windows CE é importante considerar:

- Ser o menor possível, isto é, ocupar pouco espaço de memória;
- Ser portável de forma a rodar em diversos processadores e tipos de *hardware;*
- Ser extremamente modular (permitir uma adaptação rápida e fácil a um sistema particular);
- Manter a compatibilidade com a API Win32;
- Conectividade;
- Disponibilizar processamento em tempo real (crítico em determinados sistemas embarcados);
- Implementar uma política agressiva de gestão de energia;
- Possibilitar que os fabricantes possam customizar o S.O. (MICROSOFT, 2006).

2.5.5 Arquitetura

O Windows CE foi desenvolvido tendo em conta uma arquitetura o mais modular possível. Assim sendo:

- O Windows CE é dividido em 202 módulos (EXE/DLL). Os módulos podem ser divididos em muitos componentes.
- Cada componente é um arquivo. LIB. Os componentes podem residir e serem executado de uma ROM.
- O Windows CE roda em cinco processadores diferentes (ARM/StrongARM, MIPS, PPC, SuperH, x86).

  Os componentes essenciais do *Kernel* do Windows CE são:

  - Gestão de processos e de *threads*.
  - Escalonamento previsível de *threads*.
  - Gestão de memória.

Estes componentes reutilizam o melhor das versões *desktop* do Windows, isto é, usam um modelo semelhante para a gestão de *threads*, processos e memória virtual. Assim como as aplicações para Windows CE são escalonadas segundo um modelo preempetivo total, multitarefa e com espaços de endereçamento protegidos. Ao contrário das versões *desktop*, o *kernel* do Windows CE usa DLL's como forma de minimizar a memória utilizada.

O Windows CE suporta até 32 processos simultâneos (sendo cada processo uma aplicação), embora cada processo possa ter um número de *threads* apenas limitado pela quantidade de memória disponível (P. SHELTON, 2000).

Apenas as *threads* podem ter um dos diferentes 256 níveis de prioridade que variam de *THREAD_PRIORITY_TIME_CRITICAL* até *THREAD_PRIORITY_IDLE*.

Cada *thread* pode rodar em um dos dois modos:

- Modo *Kernel* - Onde tem acesso a todos os recursos do S.O. e, portanto pode potencialmente alterar o próprio S.O. (Exemplos: *Threads* do S.O. e ISR's *Interrupt Service Routines*).

- Modo Utilizador - Este modo oferece um ambiente de execução protegido onde uma aplicação errada não atinge outras que eventualmente possam estar rodando. Aplicações normais e *device drivers* rodam neste modo (MICROSOFT, 2006).

O Escalonador (*Scheduler*) usa a técnica *round-robin* para selecionar a próxima *thread* a ser executada, começando naturalmente pelas *threads* com maior prioridade e descendo de nível quando as *threads* de um nível de prioridade superior terminam. O escalonador é orientado a *thread* e não ao processo, o que quer dizer que se um processo tem 10 *threads* e outro tem apenas um. Todas as *threads* recebem 1/11 de tempo do processador.

O Windows CE tem um espaço de endereçamento virtual de 2Gb usado por todas as aplicações, embora o espaço de memória de cada uma seja protegido e, portanto, não possa ser acessado por outras aplicações (MICROSOFT, 2006).

O espaço de endereçamento de 2Gb até aos 4Gb é para uso exclusivo do *Kernel*.

Quando um processo é carregado e designado para o próximo *slot* (divisão do espaço de memória) livre as bibliotecas DLL são carregadas na parte inferior do *slot*, em seguida o *stack* (estrutura baseada em pilha – *last in, first out*) e a *heap* (estrutura de dados, que na verdade é apenas um vetor ordenado seguindo um critério bem definido para trabalhar com processos) do processo e finalmente o executável da aplicação.

Quando o processo é selecionado pelo escalonador, é então copiado para o *slot* 0 para ser executado.

Os processos do *Kernel*, sistema de arquivos e de janelas rodam nos seus próprios *slots*.

O Windows CE foi desenhado para sistemas que normalmente não têm disco rígido e, portanto o CE não faz *swap* de páginas para disco não podendo dar a impressão de se ter mais memória do que realmente se tem.

Por não ter disco rígido, para efeitos de armazenamento a RAM e a ROM assumem um papel diferente dos sistemas *desktop*, que é servirem para armazenamento de longo prazo (MICROSOFT, 2006).

A RAM está dividida em duas partes: uma para armazenamento de aplicações e dados que não estejam na ROM e outra usada para a execução dos programas.

O sistema de armazenamento consiste em um ou mais *chip's* de memória RAM não volátil, divido em:

- Sistema de arquivos (contém os dados e aplicações; estes estão sempre comprimidos assim têm de ser descompactados para poderem rodar).

- Base de dados do Windows CE (proporciona um meio de armazenamento estruturado em vez de arquivos simples).
- *Registry* (Contém informação de configuração do sistema e das aplicações) (GOGGIN, 1999) (XIAO, et al 2002) (MICROSOFT, 2006).

Os blocos relevantes do Sistema operacional Windows CE são descritos na figura 1:

- *Kernel*: Núcleo de funções do Windows CE, manipulação de processos, gerenciamento de memória e manipulação de Interrupções. É construído para ser pequeno e rápido em relação aos de outros sistemas embarcados. É provido pela Microsoft e é escolhido pelo desenvolvedor com base no tipo de processador e necessidade do sistema.

- OAL: Código específico para uma plataforma de *hardware* específica que é construído através do uso de dado processador. E é responsável pela abstração e gerenciamento dos recursos de *hardware* do processador.

- *Boot Loader*: Responsável pela inicialização do sistema, configurando corretamente o processador e seus periféricos. Este trabalho consiste em iniciar a maioria dos dispositivos. Representa as primeiras linhas a serem rodadas em qualquer sistema operacional. O desenvolvimento do *Boot Loader* representa o primeiro passo do projeto.

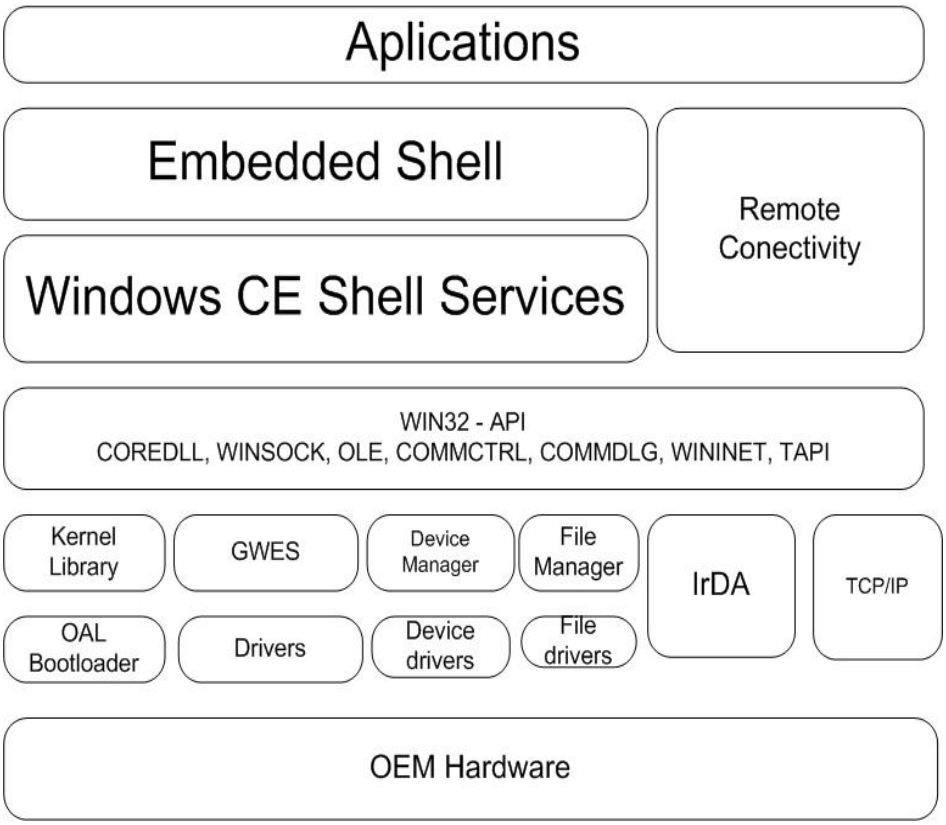


Figura 1: Arquitetura do Windows CE
Fonte: Baseado em Microsoft Corporation

- *Object Store*: Refere-se aos três tipos de armazenamento permanente, sistema de arquivos, o registro, e os bancos de dados. O total do tamanho de armazenamento é limitado em 16MB. O Windows CE suporta um sistema de arquivo proprietário e nove sistemas de arquivos modulares. O sistema de arquivos proprietário pode ser gravado em RAM (*Random Acess Memory*) ou ROM (*Read only Memory*). Se o arquivo for copiado para a RAM, o arquivo na ROM fica sobreposto pelo que está na memória RAM.

- *Graphics, Windowing and Events Subsystem* (GWES): O Windows CE pode ser implementado em qualquer *display* VGA ou monocromático.

A arquitetura do Windows CE possui uma variedade de comunicações e opções de conectividade. O Módulo de comunicações contém *drivers* padrão. Todos eles são descritos em camadas. No nível mais baixo de comunicação, o Windows CE oferece suporte à comunicação serial. No nível intermediário, ele oferece suporte a Acesso a serviços remotos e

Winsock, o qual é usado no protocolo TCP/IP e PPP/SLIP. E, no nível mais alto, são oferecidos serviços de FTP e HTTP (GOGGIN, 1999), (XIAO, et al 2002), (MICROSOFT, 2006).

2.5.6 Atendendo a requisitos especiais

No sistema operacional Windows CE é importante ressaltar duas características, uma nativa no caso do tempo real e outra onde a plataforma onde será instalado o sistema operacional deve conter, no caso da MMU.

- Tempo Real

É importante distinguir Sistema de tempo Real e Sistema operacional de Tempo Real (SOTR). Sistema de Tempo Real é um conjunto de todos os elementos especificados para essa finalidade, como *hardware*, sistema operacional e aplicativos. O SOTR é um elemento do sistema completo de tempo real (OLIVEIRA, 2001).

Um sistema é classificado como de tempo real quando a execução de toda e qualquer tarefa devem se dar dentro de uma faixa de tempo estipulada *a priori*. Isto é importante para que o sistema possa realizar tarefas periódicas ou reagir a estímulos do meio sempre dentro de um tempo previsível. No caso da automação industrial esse tempo depende das constantes de tempo do processo a ser controlado (OLIVEIRA, 2001).

Aplicações com tempo real com restrições de tempo real são menos interessadas em uma distribuição uniforme dos recursos e mais interessadas em atender requisitos tais como períodos de ativação e *deadlines*. Essas aplicações são usualmente organizadas na forma de várias *threads* ou tarefas concorrentes, logo um requisito básico para os sistemas operacionais de tempo real é oferecer suporte para tarefas e *threads* (OLIVEIRA, 2001).

Tarefas ou processos são abstrações que incluem: um espaço de endereçamento próprio (possivelmente compartilhado), um conjunto de arquivos abertos, um conjunto de direitos de acesso, um contexto de execução formado pelos registradores do processador.

Threads são tarefas leves, únicos atributos são aqueles associados com o contexto de execução.

Portanto, o chaveamento entre duas *threads* de uma mesma tarefa é muito mais rápido.

Uma aplicação tempo real é tipicamente um programa concorrente formado por tarefas e/ou *threads* que se comunicam e se sincronizam. Existem duas grandes classes de soluções para programação concorrente: Troca de mensagens e variáveis compartilhadas.

Sistemas de tempo real lidam com periféricos especiais, ou seja, diferentes tipos de sensores e atuadores, usados na automação industrial e controle de equipamentos em laboratório. O projetista da aplicação deve ser capaz de desenvolver os seus próprios tratadores de dispositivos (*device drivers*) e incorporá-los ao sistema operacional (OLIVEIRA, 2001).

O Sistema Operacional Windows CE é interessante para aplicações em tempo real dentro da automação e controle, tomamos como exemplo, a implementação de um controlador digital, como o PID (controlador proporcional, integral e diferencial), uma vez definida a equação diferenças do controlador, a taxa de atualização deve ser pequena o bastante para se obter um controle aceitável.

A aplicabilidade do Sistema Operacional Windows CE é apresentada em (ANDREW, 1999), é um sistema operacional baseado em tempo real, determinístico e *multitask.* É uma plataforma de baixo custo e construído na espinha dorsal dos padrões Microsoft.

Oferece um controle industrial em tempo real com um trajeto de migração de custo flexível e baixo controlando inclusive até o chão de fábrica. Em primeiro lugar, é baseado no mesmo Win-32 API, o que significa que os usuários podem desenvolver aplicações usando exatamente as mesmas ferramentas atuais. O Windows CE continua a tradição Microsoft de fazer o sistema virtualmente transparente. Isto significa que usuários e desenvolvedores de aplicações gastam mais tempo sobre a funcionalidade do que para aprender sobre o sistema operacional. Em segundo lugar, o Windows CE tem uma tecnologia implementada chamada COM, que possibilita a comunicação entre o chão de fabrica, e diminui a distância de comunicação entre componentes *Fieldbus Foundation* (protocolo de comunicação de redes industriais).

Neste mesmo artigo pode-se encontrar a implementação de um IHM (Interface Homem Maquina), construída baseada em Windows CE, e este exemplo servindo de controle de um robô montador de carros, usando as características de tempo real.

A Microsoft considera que seu sistema operacional é um sistema de tempo real "*hard*" baseada na definição estabelecida pelo OMAC (*Open, Modular, Architecture Control*): "um sistema de tempo real "*hard*" é um sistema que falhará se seus requisitos de tempo não forem atendidos; um sistema de tempo real "*soft*" pode tolerar variações significantes no tempo de

atendimento dos serviços do sistema como as interrupções, *timers* e o escalonador". (MICROSOFT, 2006).

Um artigo publicado em dezembro de 2002 faz um *benchmark* do determinismo entre três versões do Windows CE (TACKE; RICCI, 2002). Neste estudo, foram mensuradas a latência e o jitter nestes sistemas. A latência é uma medida do atraso que o sistema operacional leva para atender uma tarefa. O *jitter* mede a variação deste atraso. Esta análise permitiu tirar algumas conclusões sobre que tipos de aplicação de tempo real podem ser implementadas utilizando o Windows CE. O teste realizado em Tacke e Ricci (2002) é bastante simples. É gerado sinal, que por sua vez gera uma interrupção. A rotina que trata está interrupção atribui a um LED do circuito o valor '1', gera um evento do Windows e em seguida coloca o mesmo LED em '0'. Também foi implementada uma aplicação, com a prioridade mais alta do sistema, que trata o evento do Windows que foi gerado pela rotina de tratamento da interrupção. Esta aplicação, da mesma forma, atribui o valor '1' a um LED e em seguida o coloca em '0'. A partir daí, foram feitas várias medições, utilizando um osciloscópio, para calcular a latência da rotina de interrupção e a latência da aplicação de teste. Foram feitas duas análises, uma onde a aplicação era executada sozinha, e outra onde a aplicação dividia o processador com outra tarefa de mesma prioridade.

Este estudo permite concluir que, em aplicações onde às tarefas operam com intervalos de tempo da ordem de mili segundos ou até um pouco menores, o Windows CE apresenta um excelente tempo de resposta. Porém, se a aplicação apresentar tempos da ordem de micro segundos, um estudo mais detalhado deverá ser realizado sobre a arquitetura de *hardware* que será utilizada.

O Windows CE provê algumas tecnologias que são fundamentais para o desenvolvimento de aplicações de tempo real. Algumas delas são (MICROSOFT, 2006):

- 256 níveis de prioridades para as "threads". É a partir desta flexibilidade disponibilizada pelo sistema que são implementadas as políticas de escalonamento.

- Interrupções aninhadas. Permite que interrupções de prioridade mais alta sejam atendidas imediatamente ao invés de esperar que uma rotina de interrupção de prioridade mais baixa termine. O *kernel* é capaz de alinhar tantas interrupções quanto a CPU suportar.

- *Per-thread quantums*. Permite que uma aplicação defina o quantum das threads.

Inversão de prioridade. É uma situação onde o uso de um *mutex*, uma seção crítica, ou um semáforo por uma *thread* de menor prioridade impede a execução de uma *thread* de mais alta prioridade quando estas estão utilizando os mesmos recursos. Para corrigir este tipo de situação e garantir o funcionamento correto do sistema, o Windows CE permite que a *thread* de prioridade mais baixa, herde a prioridade da *thread* de maior prioridade e utiliza-se a CPU com esta prioridade mais alta até que se termine utilizar o recurso.

- MMU

A *Memory Management Unit* (MMU) é um bloco de *hardware* que transforma endereços virtuais em endereços físicos. Na MMU, o valor no registro de relocação é adicionado a todo endereço lógico gerado por um processo na altura de ser enviado para a memória. O programa manipula endereços lógicos; ele nunca vê endereços físicos reais.

A plataforma onde residirá a imagem gerada pelo *Plataform Builder* necessita da implementação antecipada ou já existente de MMU (GEORGE, 2004).

2.6 CONTROLADOR PID

Conceitos básicos

Um sistema de controle é basicamente é um sistema entradas e saídas conforme figura 2.

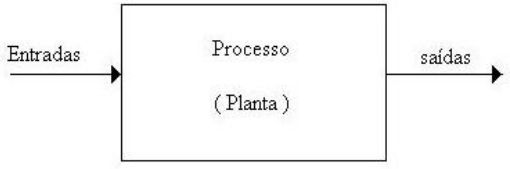


Figura 2 – Sistema de controle

Fonte: Baseado em Ogata (2003)

O sistema a ser controlado a ser controlado é, em geral, chamado de processo ou planta. O processo é um sistema dinâmico, ou seja, seu comportamento é descrito

matematicamente por um conjunto de equações diferenciais. A entrada do processo u(t) é chamada de variável de controle ou variável manipulada (MV) e a saída do processo é chamada de variável controlada ou variável de processo (PV). O conceito básico de um sistema de controle consiste em aplicar sinais adequados na entrada do processo com o intuito de fazer com que o sinal de saída satisfaça certas especificações e/ou apresente um comportamento particular. Um problema de controle consiste então em determinar os sinais adequados a serem aplicados a partir da saída desejada e do conhecimento do processo (OGATA, 2003).

2.6.1 Controle em Malha aberta

O controle em malha aberta, mostrado na figura 3, consiste em aplicar um sinal de controle pré-determinado, esperando-se que ao final de um determinado tempo a variável controlada atinja um determinado valor ou apresente um determinado comportamento. Neste tipo de controle não são utilizadas informações sobre evolução do processo para determinar o sinal de controle a ser aplicado em um determinado instante. Mais especificamente, o sinal de controle não é calculado a partir de uma medição do sinal de saída (OGATA, 2003).

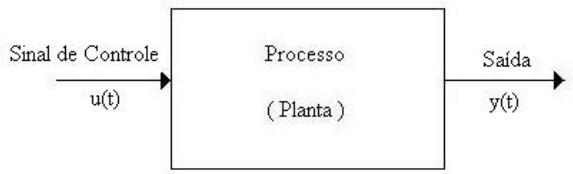


Figura 3 - Controle em malha aberta
Fonte: Baseado em Ogata (2003)

2.6.2 Controle em malha fechada

No controle em malha fechada, informações sobre como a saída de controle está evoluindo são utilizadas para determinar o sinal de controle que deve ser aplicado ao processo em um instante especifico. Isto feito a partir de uma realimentação da saída para a entrada. Em geral, a fim de tornar o sistema mais preciso e de fazer com que ele reaja a perturbações externas, o sinal é comparado com um sinal de referencia (chamado *set-point*) e o desvio (erro) entre estes dois sinais é utilizado para determinar o sinal de controle que deve efetivamente ser aplicado ao processo. Assim, o sinal de controle é determinado de forma a

corrigir este desvio entre a saída e o sinal de referência. O dispositivo que utiliza o sinal de erro para determinar ou calcular o sinal de controle a ser aplicado à planta é chamado de controlador. O diagrama básico de um sistema de controle em malha fechada é mostrado na figura 4 (OGATA, 2003).

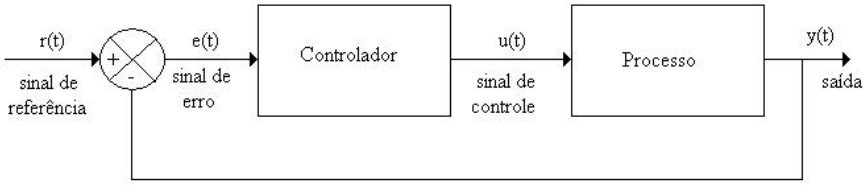


Figura 4 – Controle em malha fechada

Fonte: Baseado em Ogata (2003)

2.6.3 Controladores digitais

Um controlador digital trabalha com sinais numéricos (digitais). Um controlador digital é fisicamente implementado como uma rotina ou programa a ser executada sobre um microprocessador ou microcontrolador.

O controle digital de um processo envolve o que se chama de processo de amostragem. O sinal de saída (ou de erro) é amostrado periodicamente com um período T. O sinal amostrado (analógico) passa então por um conversor analógico/digital (A/D) onde é quantizado e transformado em sinal numérico (palavra de n bits). Este sinal digital é lido por um microprocessador (ou microcontrolador) que vai então realizar operações numéricas com este sinal e gerar outra palavra de n bits correspondente à ação de controle que deverá ser aplicada sobre a planta no próximo instante de amostragem. Este sinal numérico é então convertido novamente em um sinal analógico por um conversor digital-analógico (D/A) que disponibilizará, no próximo ciclo de amostragem, um sinal constante de tensão. Desta forma, entre dois instantes de amostragem, o sinal efetivamente aplicado pela planta é um sinal contínuo de amplitude fixa.

Em resumo, a execução de um controle digital pode ser dividida nas seguintes etapas:

- Amostragem
- Conversão analógico/digital
- Cálculo do controle através de um programa
- Conversão digital/analógica
- Aplicação do sinal de controle calculado até o próximo instante de amostragem

A escolha da freqüência de amostragem deve ser feita considerando-se o Teorema de *Nyquist* que diz que "a freqüência de amostragem, fs, deve ser no mínimo 2 vezes maior que a freqüência contida no sinal analógico a ser amostrado" a fim de evitar o fenômeno de *aliasing* (superposição de espectro). Intuitivamente, se for feita a amostra do sinal com uma freqüência muito baixa, estará, de certa forma, perdendo informações sobre a evolução do sinal que está sendo amostrado o que acarretará o cálculo de um controle incorreto. A fim de evitar este fenômeno, um filtro anti-*aliasing* pode ser acrescentado ao sistema. Este filtro, colocado antes do dispositivo amostrador, funciona como um filtro passa-baixas que se encarrega de eliminar as componentes em freqüência acima de fs/2 (OGATA, 2003).

2.6.4 Controlador PID

Um controlador conhecido por PID ou controlador proporcional-integral e derivativo é largamente utilizado em sistemas práticos. Apesar deste controlador já ser conhecido no período entre as duas guerras mundiais, continua, nos dias de hoje, a ser o controlador industrial mais utilizado. Hoje, estes controladores são implementados em *hardware* (analógico ou digital) ou *software* (controle microprocessado).

As ações de controle consideram que um sinal de controle será aplicado a uma planta a ser controlada. Este sinal de controle será gerado pelo controlador tendo como entrada o sinal de erro. Utiliza-se a seguinte notação:

u(t) – sinal de controle na saída do controlador, no domínio contínuo ou analógico.

e(t) – sinal de erro na entrada do controlador, no domínio contínuo ou analógico.

u[k] – sinal de controle na saída do controlador, no domínio discreto ou digital, no instante t=k.

e(k) – sinal de erro na entrada do controlador, no domínio discreto ou digital, no instante t=k.

Kp – constante de proporcionalidade associada ao termo proporcional.

Ki – constante de proporcionalidade associada ao termo integral.

Kd – constante de proporcionalidade associada ao termo derivativo.

O controlador PID combina as ações de controle proporcional, integral e derivativa. A relação matemática entre o sinal de erro e de controle é dada por:

$$u(t) = K_p e(t) + \frac{1}{T_i}\int e(t)dt + T_d \frac{d}{dt} e(t) \quad (1)$$

O controlador PID apresenta a seguinte função de transferência:

$$U(s) = K_p E(s) + \frac{K_i}{s} E(s) + K_d s E(s) \quad (2)$$

Os parâmetros Kp, Ti e Td são escolhidos em função do desempenho final do sistema a ser controlado desejado. À escolha destes parâmetros chama-se sintonia do PID. Vários algoritmos, numéricos ou não, são utilizados para obter esta sintonia, a exemplo do método de *Ziegler* e *Nichols*. Em função da não utilização de uma ou mais das ações de controle básicas pode-se ter as várias derivações do controlador PID (OGATA, 2003).

2.7 CONTROLADOR LÓGICO PROGRAMÁVEL (CLP)

O CLP surgiu graças à necessidade da indústria automobilística. Em 1968, a General Motors gastava tempo e dinheiro para modificar a lógica de controle dos painéis de comando para cada mudança da linha de montagem. A solução encontrada naquela época foi a construção de um *hardware* que permitia sua utilização em diversas situações, bastando para isso apenas mudanças em sua fiação. Na verdade, esse *hardware* era vários relés colocados em placas, e que eram acoplados a um bastidor (ou *rack*), e que chamada *back-plane* (placa traseira). O *back-plane* possuía inúmeros terminais que correspondiam às bobinas e aos contatos dos relés. Através da fiação determinava-se o comportamento do dispositivo.

O CLP é um equipamento eletrônico digital com *hardware* e *software* compatíveis com aplicações industriais, ou seja, é um computador especializado, baseado num

microprocessador que desempenha funções de controle de diversos tipos e níveis de complexidade (SIEMENS, 2003).

2.7.1 *Hardware* do CLP

O *hardware* do CLP pode ser separado em três unidades distintas: fonte de alimentação, CPU e Interface I/O (SIEMENS, 2003).

- Fonte de alimentação

Atualmente a maiorias das fontes do CLPs são do tipo são chaveada, e apresenta uma única tensão de saída 24VCC.

Normalmente os fabricantes de máquinas industriais adotam a fonte de 24 VCC porque, devido a sua grande amplitude em relação aos tradicionais 5Vcc, ela permite que o equipamento adquira maior imunidade a ruídos elétricos.

Além disso, essa tensão está compatível com o sistema RS-232 de comunicação (SIEMENS, 2003).

- CPU

Tem por finalidade processar um determinado programa que foi colocado em sua memória o qual fará o controle de um determinado processo. A CPU de um CLP pode ter inúmeras naturezas. As escolhas como: microprocessador ou microcontrolador, qual marca e qual modelo, são feitas pelo fabricante do CLP (SIEMENS, 2003).

- Interfaces I/O

São as entradas e saídas que podem ser divididas em dois grupos: analógicas e digitais.

Entradas analógicas – a maioria dos processos e máquinas industriais necessita do controle de grandezas analógicas, por exemplo: temperatura, umidade relativa e posicionamento de eixos. Os CLPs industriais possuem algumas entradas analógicas que através de um conversor interno analógico digital, permitem que essas grandezas sejam monitoradas e controladas.

Entradas digitais – admitem apenas dois estados, ou seja, 0 ou 1. Normalmente, o nível 1 significa 24VCC. As entradas digitais podem ser de dois tipos: ativa em baixa (ou do tipo

N) ou ativa em alta (tipo P). Para ativarmos uma entrada P devemos ligá-la em 24VCC, e para uma entrada N zero Volt.

Saídas Analógicas – assim como um CLP apresenta entradas analógicas, ele também tem saídas analógicas que funcionam com a mesma filosofia da entrada. Os níveis de tensão ou corrente são os mesmos. Os sinais das saídas analógicas são aplicados em: válvulas proporcionais, motores CC, inversores de freqüência, módulos IHM (Interface Homem Máquina), entre outros.

Saídas digitais – Analogamente as entradas digitais, as saídas digitais apresentam apenas dois estados: alto (24 VCC), ou baixo (Zero Volt). Os componentes associados às saídas digitais são: contatores, relés eletromecânicos e de estado sólido, solenóides, válvulas, alarmes, LEDs, lâmpadas, entre outros (SIEMENS, 2003).

2.7.2 Programação

O programa do CLP é estruturado em blocos. Mais precisamente em cinco blocos:

1. Bloco de organização – bloco que organiza toda a seqüência da automação. Todos os demais blocos estão contidos neste. Na prática ele é um programa do tipo executável (exe).
2. Bloco de programa – é no bloco de programa que instalamos o software residente do CLP. Normalmente a memória deste bloco é do tipo RAM (com bateria para mantê-la) ou *flash*.
3. Bloco de funções – abriga os dados das variáveis externas (temperatura, entrada e saída analógica, vazão, entre outros).
4. Bloco de dados – pode ser usado e alterado durante a execução do programa. Os dados mais comuns a esse bloco são: esclarecimentos (comentários) sobre o próprio programa: blocos de funções, referências e tempos.
5. Bloco de passos – contém os programas gráficos do CLP. É neste bloco que tratamos a sinalização do processo (sinais da IHM, *Grafcet*, Fluxograma do processo ou da máquina).

Normalmente, o CLP tem dois modos de status (funcionamento): *RUN* e *STOP*. Quando em *RUN* o programa do CLP está em execução, em *STOP* o CLP está em "*stand-by*", ou seja, em espera ou parado. Essa última condição pode ser devida a uma falha (*software* ou *hardware*), ou ser provocada com o intuito de manutenção. Cada fabricante pode optar por uma ou outra linguagem de programação, porém, a maioria dos fabricantes segue a norma IEC 1131, que permite três modalidades: DIC (linguagem de contatos ou *LADDER*), DIL (blocos lógicos) e LIS (lista de instruções) (SIEMENS, 2003).

Para exemplificar as três linguagens tomamos como base o circuito da figura 5, este é circuito função lógica E:

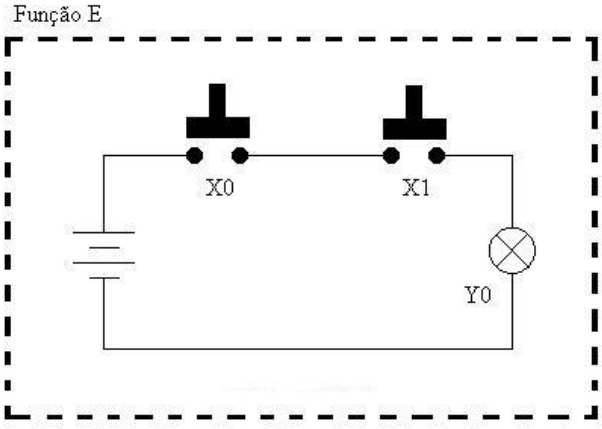


Figura 5 – Circuito elétrico
Fonte: O Autor

A partir do circuito é feito o levantamento da expressão lógica (3) que traduz o mesmo:

$$Y0=X0.X1 \qquad (3)$$

Depois de se obter a equação que descreve o funcionamento do circuito é feito a tradução para as linguagens de programação do CLP, similar aos circuitos usados em eletrônica digital (SIEMENS, 2003).

2.7.3 DIC

No sistema DIC as variáveis (entradas) são representadas por contatos, e a saída por parênteses. Basicamente as instruções em *LADDER* podem ser divididas em dois grupos, mostrados no quadro 1:

- instruções básicas;

- instruções avançadas;

Quadro1 – Instruções do CLP isto é um quadro

Básicas	Avançadas
Relé Contato	Aritmética de ponto flutuante
Bobina	Raiz quadrada
Temporizador	Movimentação de operandos
Contator	Movimentação em tabelas
Soma	Rotação de registradores
Multiplicação	Blocos de diagnósticos
Subtração	Seqüenciadores
Divisão	PID
Comparação	Rede
Função	Lógica matricial
	Pilhas

A figura 6 descreve as funções básicas usadas na programação por diagrama de contatos, que serão usadas para descrever o exemplo.

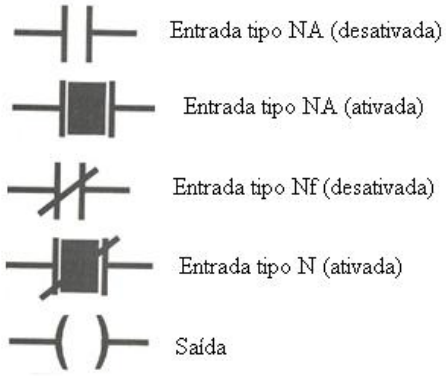


Figura 6 – Instruções básicas *LADDER*

Fonte: Manual S7-200 da Siemens

O programa na linguagem DIC que traduz o funcionamento do circuito da figura 5 é mostrado na figura 7 (SIEMENS, 2003):

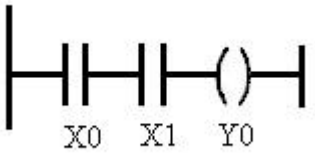


Figura 7 – linguagem *LADDER*

Fonte: Manual S7-200 da Siemens

2.7.4 DIL

Essa linguagem de programação é usada em equipamentos onde o usuário tem um *display* LCD para visualizar as funções lógicas. É uma linguagem gráfica que permite ao usuário programar elementos de tal um modo que eles aparecem interligados como em circuitos elétricos. Como exemplo de uso dessa programação é citado o LOGO do fabricante SIEMENS mostrado na figura 8.

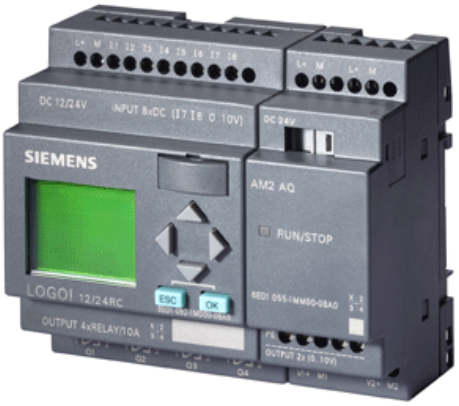


Figura 8 – LOGO da SIEMENS

Fonte: Manual S7-200 da Siemens

A figura 9 mostra o programa que descreve o funcionamento do circuito da figura 5 na linguagem DIL (SIEMENS, 2003).

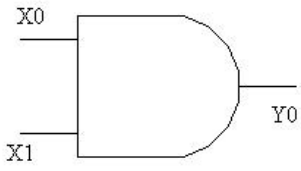


Figura 9 – Diagrama lógico

Fonte: O Autor

2.7.5 LIS

O sistema de lista de instruções escreve linha a linha as operações. É uma linguagem de baixo nível semelhante à máquina ou *assembler* usada com microprocessadores. Este tipo de linguagem é útil para aplicações pequenas, como também aplicações que requerem otimização de velocidade do programa ou uma rotina específica no programa. Como mencionado anteriormente, a IL pode ser usada para criar blocos de função. Uma aplicação típica de IL poderia envolver a inicialização para zerar (isto é, *reset*) do valor dos acumuladores de todos os cronômetros em um programa de controle (SIEMENS, 2003).

O programa a seguir mostra em linguagem de lista de instruções o funcionamento do circuito da figura 5.

LD X0 (carrega a entrada X0)
AND X1 (“operação e” com a entrada X1)
OUT Y0 (envia o resultado para a respectiva saída)

2.7.6 Considerações para escolha do CLP

- Ter a certeza de que todos os produtos já instalados são compatíveis com o CLP que será implantado.
- Definição de qualquer condição ambiental que irá afetar a sua aplicação. Existem especificas questões ambientais que irão afetar o sistema (temperatura, ruídos elétricos, vibrações, códigos específicos para sua facilidade, etc.)? Certamente o meio ambiente pode afetar na operação de um CLP. Por exemplo, um típico CLP tem sua faixa de temperatura de 0-60 graus Celsius. Se a aplicação incluir qualquer condição ambiental extrema, precisará encontrar produtos que satisfaçam tais condições, ou projetar uma instalação que reúna estas especificações.
- Determinação de quantos dispositivos analógicos e discretos sua aplicação terá. Quais tipos (AC, DC, etc.) serão necessários? O numero e o tipo de dispositivos que seu sistema incluirá, é diretamente relacionada ao número de I/O que será necessária para seu sistema. Será necessário escolher um CLP que suporte a quantidade de I/O que serão utilizadas e tenham módulos que suportem os tipos de sinal utilizados.
- Determine quando o sistema irá utilizar qualquer característica especial. Sua aplicação irá utilizar algum contador rápido ou posicionamento? Quanto a um *clock* em tempo

real ou outra função especial. Funções especiais não são necessariamente possíveis utilizando módulos de I/O padrões. Planejando primeiramente quando ou não a aplicação irá requerer tais características, irá ajudar determinar se precisarão adquirir os módulos especiais para o sistema.

- Determinar o tipo da CPU que será utilizada: Quanta memória o sistema necessita. Quantos dispositivos o sistema terá (determina a memória de dados). Qual o tamanho do programa e quantos tipos de instruções serão incluídos (determina a memória de programa)? A memória de dados se refere à quantidade de memória necessária para a manipulação de dados dinâmicos e de armazenamento do sistema. Por exemplo, contadores e temporizadores normalmente utilizam a memória de dados para armazenar os valores registrados, valores correntes e outras marcas. Se a aplicação um histórico da retenção de dados, tais como medidas dos valores dos dispositivos durante um longo espaço de tempo, os tamanhos da tabela de dados requerida vai depender de qual modelo de CPU você escolher. A memória de programa é a quantidade de memória necessária para armazenar a lista de instruções do programa que foram programadas para a aplicação. Cada tipo de instrução requer uma quantidade de memória diferente, normalmente especificada no manual de programação do CLP. Mas a memória se tornou relativamente barata e facilmente é feito um *upgrade* se necessário, se compararmos com épocas anteriores.
- Determinação onde as I/Os estarão localizadas. O seu sistema terá apenas I/Os locais, ou ambas I/Os locais e remotas? Se sua aplicação irá necessitar de elementos a uma longa distância da CPU, então irá precisar de um modelo de CLP que suporte I/O remota. Determinar se à distância e a velocidade suportada pelo CLP irão se adequar para a aplicação.
- Determinação dos requisitos de comunicação. O sistema terá que se comunicar com outra rede ou outro sistema? As portas de comunicação não são necessariamente incluídas junto com os CLPS. Sabendo primeiramente que seu sistema irá ou não comunicar com outro sistema, ajudará na escolha da CPU que suportará os requisitos de comunicação ou módulos adicionais de comunicação se necessário.
- Determine os requisitos do programa. O programa necessita apenas de funções tradicionais ou de funções especiais? Alguns CLPs não suportam todos os tipos de instruções. Será necessário escolher um CLP que suporte todas as instruções que necessite para uma aplicação especifica. Por exemplo, funções PID básicas que são até

fáceis de usar, escrevendo o seu próprio código para realizar controles de processo de ciclo fechado (SIEMENS, 2003).

CAPÍTULO 3

PROPOSTA DE UTILIZAÇÃO DO SISTEMA OPERACIONAL WINDOWS CE PARA APLICAÇÕES DIDÁTICAS NA ÁREA DE CONTROLE E AUTOMAÇÃO

3.1 INTRODUÇÃO

Tem-se como objetivo, neste capítulo, apresentar uma proposta para execução projetos de sistemas com requisitos em tempo real, particularmente nas áreas de controle e automação. Esta classe de projetos vem aumentando na indústria, também em função do aumento de tecnologia embarcada presente nos equipamentos.

Descreve-se, neste momento, a proposta de um ambiente baseado em Windows CE para desenvolvimento na área de Engenharia, contemplando-se os passos genéricos para a construção de um *Kit* didático para o uso em sistemas embarcados, utilizando-se o sistema operacional Windows CE.

Usando uma descrição detalhada das etapas necessários para a construção de uma imagem do sistema embarcado Windows CE, dentro do contexto de projeto proposto, contribuí-se para que esta proposta seja utilizada em futuros projetos em ambiente acadêmico ou industrial.

3.2 COMPARATIVO ENTRE SISTEMAS OPERACIONAIS

A tabela 1 mostra um comparativo entre alguns sistemas operacionais embarcados, as considerações tomadas para a construção dessa tabela foram: fabricante, tamanho da imagem, custo para a implantação do sistema operacional, *hardware* compatível e facilidade de implementação de aplicativos.

Tabela 1 Comparativo entre sistemas operacionais embarcados

Sistema Operacional	Fabricante	Tamanho da imagem	Custo	Hardware compatível	Facilidade de implementação de aplicativos
Windows Ce	Microsoft	500kB a 16MB	$1000 iniciais, $1 por imagem construída	Processador compatível x86, 13 MB de RAM e 16 MB de Disco	Pouca literatura escrita a respeito da linguagem proprietária
µLinux	Linux	400KB	Zero	o processador Motorola DragonBall, o Motorola ColdFire, o ARM, o NEC V850E e o Intel i960.	Pouca literatura escrita, porém facilidade de encontrar material na Internet
Netbsd	Linux	400KB	Zero	Alpha, ARM, i386, Motorola PowerPC, SPARC, Motorola 68k, Hitashi SH3, VAX, MIPS,	
VxWorks	WindRiver	Desde 400 Kbytes até 1,5 Mbytes		Intel Pentium, ARM e MIPS.	
MiniRTL	FSMLabs	500KB	Zero	x86	

Fonte: O autor

A tabela 1 um serve de subsídio para a escolha de um determinado sistema operacional embarcado. A escolha deve ser feita pelo usuário, que por sua vez, escolhe as características que mais lhe traz vantagens.

3.2.1 NetBSD

Baseado em Unix, este também é um sistema embarcado *open-source* (código fonte aberto). Uma desvantagem em relação a outros SO Embarcados é a sua relativa necessidade de memória podendo chegar até 16MB de RAM. Desta forma, ele se apresenta como uma boa opção para rodar em roteadores e outros dispositivos de rede que possuem, de modo geral, boa quantidade de memória disponível.

3.2.2 uClinux

O Micro Controller Linux é um SO Embarcado de código aberto criado em 1998 e completamente voltado a Sistemas Embarcados. Possui *kernel* de apenas 900KB e suporte ao protocolo TCP/IP, entre outros protocolos de rede, além de sistemas de arquivos diversos como NTFS e FAT16/32, entre outros. Este sistema, por ser baseado em Linux, mantém sua estabilidade e portabilidade. Outra vantagem do uCLinux é o suporte a várias arquiteturas.

3.2.3 VxWorks

Sistema embarcado de tempo real, disponível para vários tipos de processadores do mercado:

- ARM
- IBM Power PC
- Intel Pentium
- Intel i960
- Motorola PowerPC
- Entre outros

A escolha do sistema operacional embarcado e feita através das vantagens que ele proporciona ao usuário, pois existem vários fabricantes, cada fabricante com suas variantes que proporcionam alguma característica em especial. A seguir são descritas vantagens e desvantagens de três sistemas operacionais: Windows CE, XP Embedded e Linux.

3.3 PRINCIPAIS SISTEMAS OPERACIONAIS EMBARCADOS

A seguir são citadas vantagens e desvantagens sobre os três importantes sistemas operacionais, cabe ao usuário definir qual sistema operacional atende suas necessidades.

3.3.1 Windows CE

- ✓ As vantagens do sistema operacional Windows CE são:

  - Suporta Interfaces de comunicação Infravermelho via IrDA, TCP/IP e *drivers* seriais
  - Interface gráfica igual a Windows XP Professional.
  - Suporta 32 processos simultaneamente
  - Suporta número ilimitado de *threads*, usadas para monitorar eventos assíncronos, tais como: interrupção de *hardware* e atividades do usuário (depende de memória física disponível)
  - Suporta comunicação entre processos – IPC (seções críticas, mutex e eventos)
  - Suporta oito níveis de prioridades de *threads* (do ocioso ao tempo critico). As prioridades não podem mudar, exceto quando uma *thread* de baixa prioridade retém recurso que uma *thread* de alta prioridade precisa
  - Suporta Interrupções que são usadas para notificar o SO sobre eventos externos
  - Atende requisitos de um S.O. de tempo real não critico para aplicações do tipo:
  - Comutação de telefonia, controle de processos de fabricação, sistemas automotivos de navegação etc.
  - Trata os vários tipos de armazenamento da mesma forma: cartões de memória *flash*, *drivers* de disco, ROM, RAM
  - Suporta até 32MB por processo

- ✓ Desvantagens

  As desvantagens do sistema operacional Windows CE são:

  - Plataforma necessita de MMU.

- O tempo gasto na construção de *device drivers* se o mesmo não for disponibilizado pelo fabricante do *device*.
- O usuário não consegue usar os aplicativos existentes, ou seja, tem que programar novos se desejar.
- Literatura escassa.

3.3.2 Windows XP Embedded

✓ As vantagens do sistema operacional Windows XP Embedded são:

- Interface gráfica igual a Windows XP Professional
- Uso de aplicativos existentes.
- Ferramentas usadas são as mesmas do Windows XP *Professional*

✓ As desvantagens do sistema operacional Windows XP Embedded são:

- Não tem *driver* para bateria
- Uso somente na plataforma X86
- Tamanho de imagem maior em relação ao Linux e Windows CE

3.3.3 Linux Embarcado

✓ As vantagens do sistema operacional Windows XP Embedded são:

- Custo.
- Ferramentas de desenvolvimento acessíveis e fornecidas sem custo nenhum.
- SO pode ser completamente customizado conforme a aplicação.
- Literatura abundante na Internet.
- Existem vários modelos, com focos de utilidade diferente.

✓ As desvantagens do sistema operacional Windows XP Embedded são:

- O suporte técnico não é tão formal como Windows XP *embedded* nem o CE.
- Interface gráfica onde o usuário comum não está habituado.

3.4 SELEÇÃO DO SISTEMA OPERACIONAL: WINDOWS CE OU XP *EMBEDDED.*

A tabela 2 descreve características relevantes dos dois sistemas operacionais.

As características da tabela 2 ajudam a analisar algumas características desses dois sistemas embarcados. Essas características são importantes para se ter subsídios de escolha entre um e outro.

Tabela 2 – Características CE versus XP *embedded manter tabela junta*

Característica	Windows CE	Windows XP *Embedded*
Espaço em disco	500KB – 16MB	5MB – 56MB
Tamanho da imagem	200KB – 14MB	5MB – 35MB
Configurabilidade	Extremamente modular	Modular
Suporte API	*Win32 plus windows CE specific*	Completo Win32, igual ao Windows XP *professional*
CPU	X86, MIPS, SHx, ARM	Pentium *Class* X86
Device Driver	*Fine-tuned for size*	Igual ao XP
Segurança	*Improved over 3.0*	Igual ao XP
Plataforma de desenvolvimento	Windows 2000/XP *Plataform Builder*	Windows 2000/XP *Target Guitar*

Fonte: Microsoft

A justificativa da escolha do Windows CE é baseada na aplicabilidade no âmbito industrial. O sistema operacional deve ter as seguintes características para atuar no âmbito industrial:

- Imagem mais customizada que o XP;
- Ser um sistema operacional de tempo real;
- Ter *driver* para o uso de bateria;
- Usar o mínimo de memória RAM possível.

3.5 MICRO XP *PROFESSIONAL* VERSUS MICRO WINDOWS CE

As vantagens de se usar um PC com uma imagem do sistema operacional Windows CE, ao invés de se usar um micro com Windows XP *home* ou *Professional*, na sala de aula estão descritas a seguir:

- Aproveitamento de máquinas – Como a imagem gerada é bem menor que o sistema operacional XP completo a máquina a ser usada não precisa ser atual ou com características de alto desempenho, pode ser feitas experiências com micros Pentium 200MHz, por exemplo.

- Imagens pré-configuradas – com o uso de imagens pré-configuradas as aulas podem ser preparadas antes em um cd inicializável, contendo material didático e aplicativos que serão usados em sala de aula. Depois da aula ministrada e a retirada do CD o laboratório pode ser usado novamente sem problemas com instalação de *softwares* e configurações adicionais.

- Tempo Real – Característica usual e importante nos laboratórios de controle e automação, característica tal nativa no sistema operacional Windows CE.

- Custo do Licenciamento de *Software* – Uma licença do sistema operacional Windows XP *Professional* custa em torno de R$1300,00 contra o preço de US$1000 pelo *Platform builder* e o custo de um dólar por imagem criada.

- Configurabilidade - o Windows, especialmente o *Kernel*, o núcleo do sistema, possibilita a própria configuração otimizada especificamente para cada aplicação.

- Ferramentas de Desenvolvimento – O Windows CE apresenta ferramentas de desenvolvimento em linguagem C++ e Java. As ferramentas que funcionam no Windows XP necessitam serem instaladas e pagas a parte.

3.6 PROPOSTA DE UMA PLATAFORMA DESENVOLVIDA EM AMBIENTE WINDOWS CE

Uma aplicação baseada em Windows CE é composta por três componentes básicos, representados no diagrama de blocos da figura 10:

- Bloco Gerente: é onde o *Plataform builder* fica residente. Este aplicativo gera as imagens do sistema operacional Windows CE. Neste nível, além da geração de imagens, podem ser realizadas as seguintes ações:

  a) Mensagens de aplicativos – Após a programação de qualquer aplicativo e posterior download do mesmo nos computadores aplicativo, pode-se realizar o acompanhamento remoto dos aplicativos carregados através do canal de comunicação serial, recebendo mensagens pré-definidas.

  b) Aplicativos remotos – Através de comunicação serial ou via *Ethernet* pode-se executar aplicativos remotos gerados no bloco gerente nos blocos aplicativo.

  c) Acompanhamento de Processos – Através de programas específicos nativos no *Platform builder* pode ser feito à supervisão remota de aplicativos, através de mensagens seriais.

- Bloco Aplicativo: É onde a imagem gerada pelo *Platform Builder* será executada. Em função das características inerentes do Windows CE, esta imagem pode ser executada em plataformas com menores recursos de *hardware* e desempenho. Isto permite a utilização de máquinas consideradas obsoletas para desenvolvimento de experimentos em diversas disciplinas dos cursos de graduação. Além de rodar em plataformas CEPC:X86, há a possibilidade de troca da plataforma, como por exemplo instalá-lo numa plataforma baseada em processadores ARM, ou ainda, em outra plataforma que contenha MMU (*memory manager unit*). As aplicações neste nível dependem das características da plataforma escolhida. Por exemplo, a plataforma CEPC:X86 disponibiliza diversas interfaces de comunicação: USB, paralela, serial e Irda. Outras plataformas, em função do projeto utilizado, podem apresentar um conjunto mais restrito de opções de comunicação (K. S. P. CLARKE, 2002).

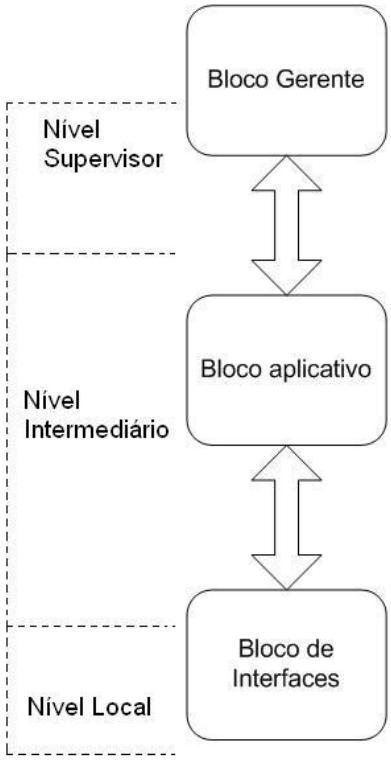


Figura 10 – Diagrama de Blocos de uma plataforma desenvolvida em Windows CE
Fonte: O Autor

- Bloco de interfaces: Neste bloco estão representadas as interfaces, atuadores e sensores utilizados para executar os experimentos. Neste bloco é possível implementar um grande número de diferentes topologias. Sensores e atuadores podem ser conectados diretamente às interfaces de comunicação disponibilizadas pelo bloco aplicativo. Neste caso as interfaces disponibilizadas por este bloco passam a ser de grande importância na construção de um experimento. Por outro lado, os sensores e atuadores podem ser conectados através de uma placa de interface adicional ao bloco aplicativo. Esta opção é bastante adequada em experimentos onde existe uma grande demanda de processamento dos sensores e atuadores, ou ainda, uma necessidade de gerenciamento de um grande número de sensores e atuadores.

As figuras 11, 12 e 13 apresentam diferentes topologias que a proposta permite implementar. Na figura 11, observa-se um exemplo de conexão de um sensor diretamente ligado ao bloco aplicativo.

O professor pode fazer um programa no bloco gerente, enviar para o bloco aplicativo onde o aluno poderá acionar o sensor através de comunicação serial.

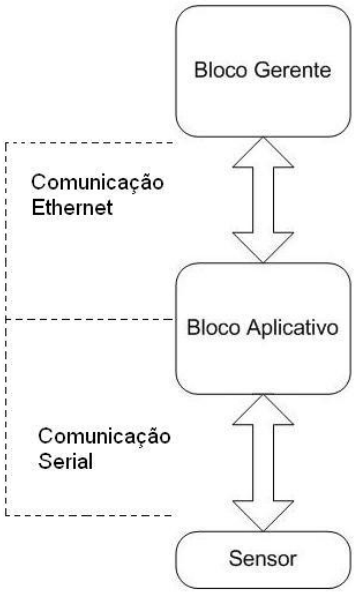


Figura 11 – Topologia 1

Fonte: O Autor

Na figura 12, observa-se a conexão de um sensor ligado diretamente ao bloco aplicativo com um atuador (motor) conectado à uma placa de potência. Nesta configuração o aluno pode fazer experiências de controle de velocidade do motor.

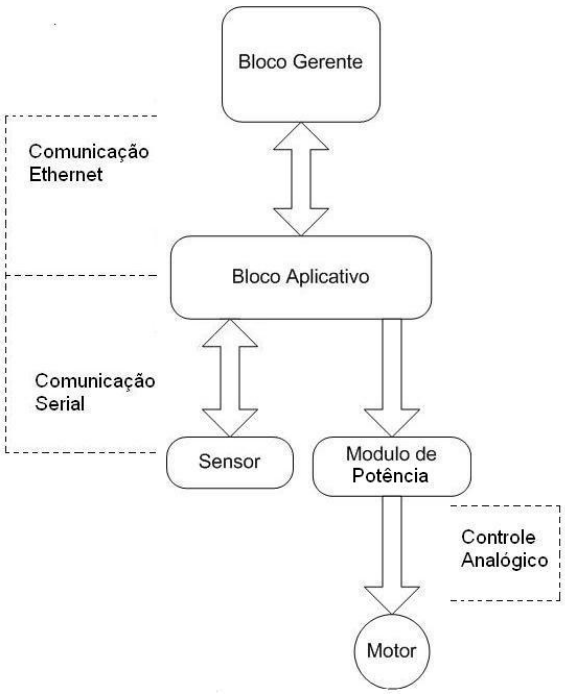


Figura 12 – Topologia 2

Fonte: O Autor

Finalmente, na figura 13, é observada a conexão de vários sensores e atuadores através de uma FPGA, nesta topologia o aluno pode fazer experiências relacionadas com: controle de velocidade do motor, comunicação entre o bloco aplicativo e a FPGA, sensores e atuadores sendo acionados pelo Windows CE através da FPGA.

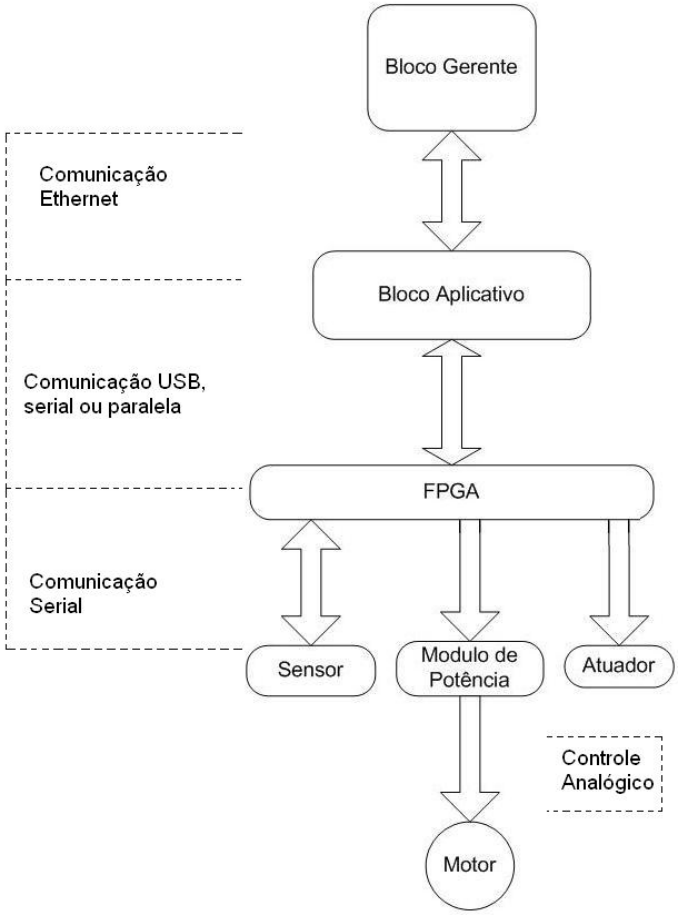


Figura 13 – Topologia 3

Fonte: O Autor

3.7 ETAPAS A SEREM SEGUIDAS PARA A CONSTRUÇÃO DE UMA IMAGEM

A descrição de como é criada uma imagem é demonstrada a seguir:

Depois que *software Platform Builder* estiver instalado no PC apropriado, a tela de apresentação é a da figura 14.

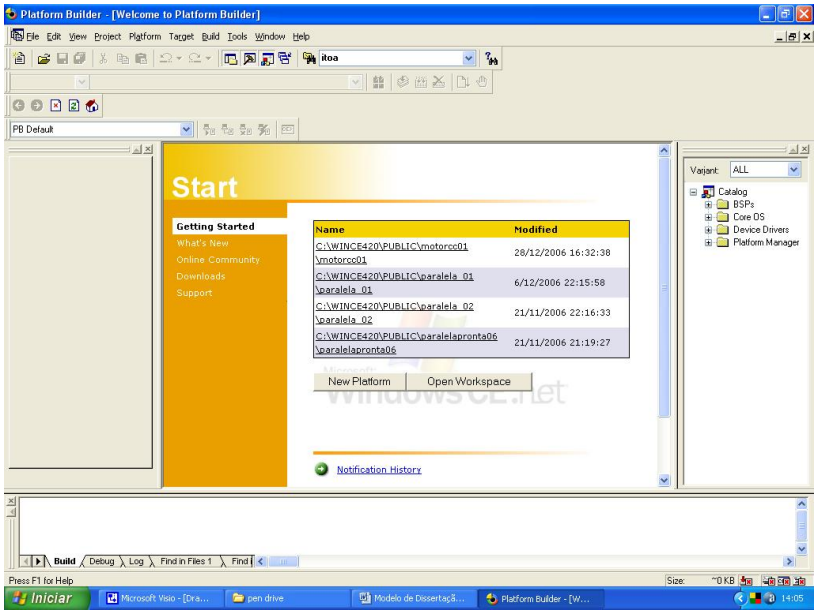


Figura 14 – Tela principal do *Plataform Builder.*

Fonte: construído a partir do *Plataform Builder*

Para começar a construir uma imagem seleciona-se o item *file* depois *New Plataform*.

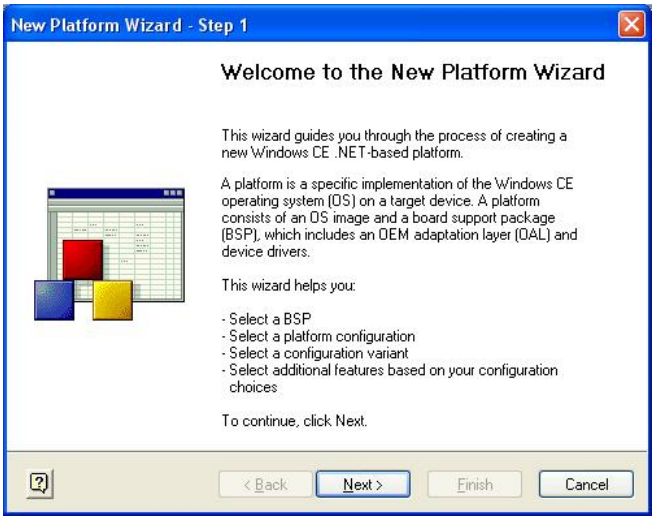


Figura 15 - *Platform Wizard*

Fonte: construído a partir do *Plataform Builder*

Na tela da figura 15, aparecerão os passos seguidos durante a construção da requerida imagem.

A tela, descrita na figura 16, indica a escolha do BSP (*Board Support Packages*), é o catálogo que contém o conjunto de *drivers* padrão que serão adicionados à plataforma.

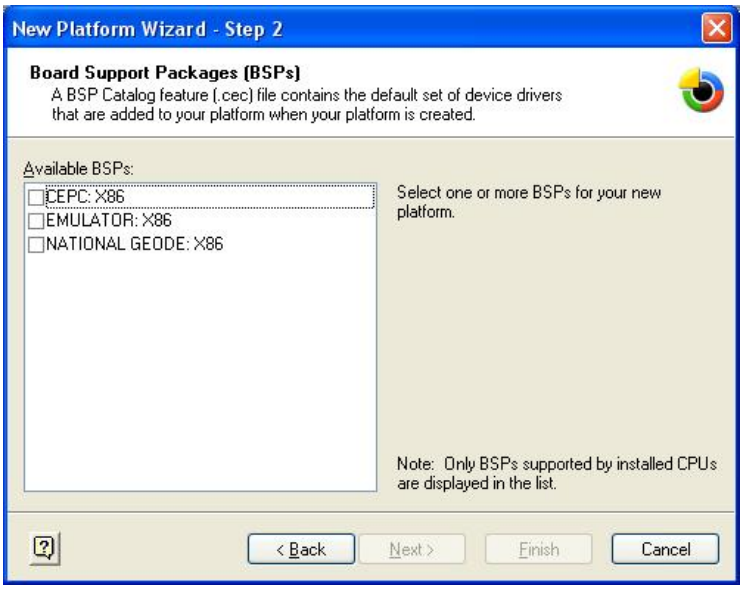


Figura 16 – Escolha do BSP.

Fonte: construído a partir do *Plataform Builder*

As opções de BSP são:

- ***CEP:X86*** – é o catálogo de *drivers* usado em uma plataforma baseada nos processadores X86, como por exemplo, Pentium 400MHz.
- ***EMULATOR: X86*** – é catálogo de *drivers* usado em uma plataforma X86, mas que funciona na forma de emulador, ou seja, a imagem roda no bloco gerente, num programa de emulação de uma plataforma X86.
- ***NATIONAL GEODE:X86*** – é o catálogo de *drivers* usado em uma plataforma baseada em processadores Geode:x86, construído pela *National Semiconductors*.

Na tela da figura 17, é definida a configuração da plataforma e esta escolha é feita a partir de uma plataforma predefinida com um conjunto customizado de *drivers* para cada especialidade como, por exemplo:

- ***Enterprise Web Pad*** – Imagem pré-definida com conjunto de arquivos e *kernel* customizado para dispositivos de acesso à Internet, incluindo o *Internet Explorer* 5.5, funcionam com dispositivos *touch screen*.
- ***Windows Thin Client*** – Imagem pré-definida para obter a configuração do sistema operacional de forma remota. Imagem mais customizada em relação às outras, mas inclui acesso a vídeo.

- ***Industrial Automation device*** – Imagem pré-definida com arquivos e *kernel* customizados para configurarem dispositivos como: painel interface homem máquina, controlador lógico programável.
- ***Internet Appliance*** – Similar ao *Entreprise Web Pad*, diferenciado pelas seguintes características: Linguagem XML (*Extensive markup Language*), rede *wireless* e configuração adicional de mouse e teclado.

Figura 17 – Configuração da Plataforma

Fonte: construído a partir do *Plataform Builder*

Estas plataformas predefinidas são usadas para resolver problemas específicos, caso não exista nenhuma plataforma adequada para o usuário escolher. Isto implica que o usuário deve construir a sua própria plataforma customizada, como descrita a seguir:

Seleciona-se o item *Custom configuration* e escolhe-se um nome para a imagem, e clica-se em *next*.

Na figura 18, é feita a escolha se aplicação em questão terá ou não suporte para vídeo.

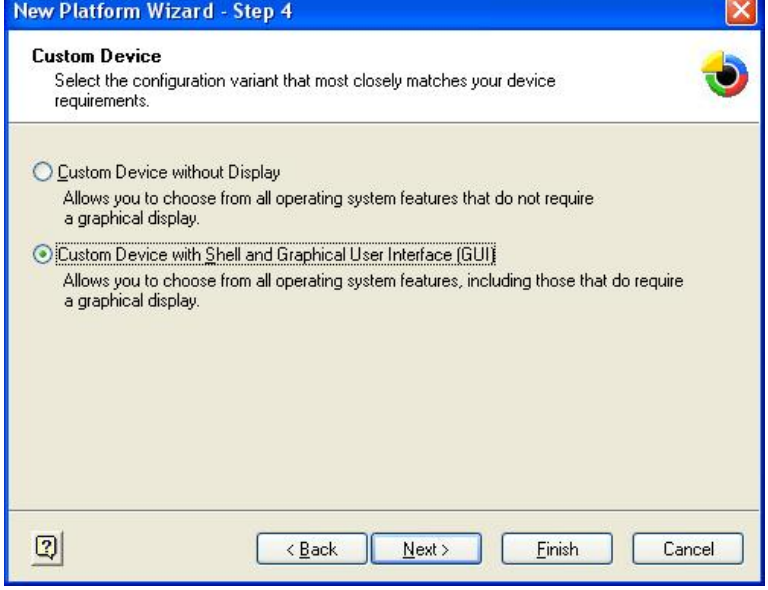


Figura 18 – Seleção de suporte para vídeo

Fonte: construído a partir do *Plataform Builder*

Na figura 19 é feita a escolha de aplicações e serviços de desenvolvimento, são eles:

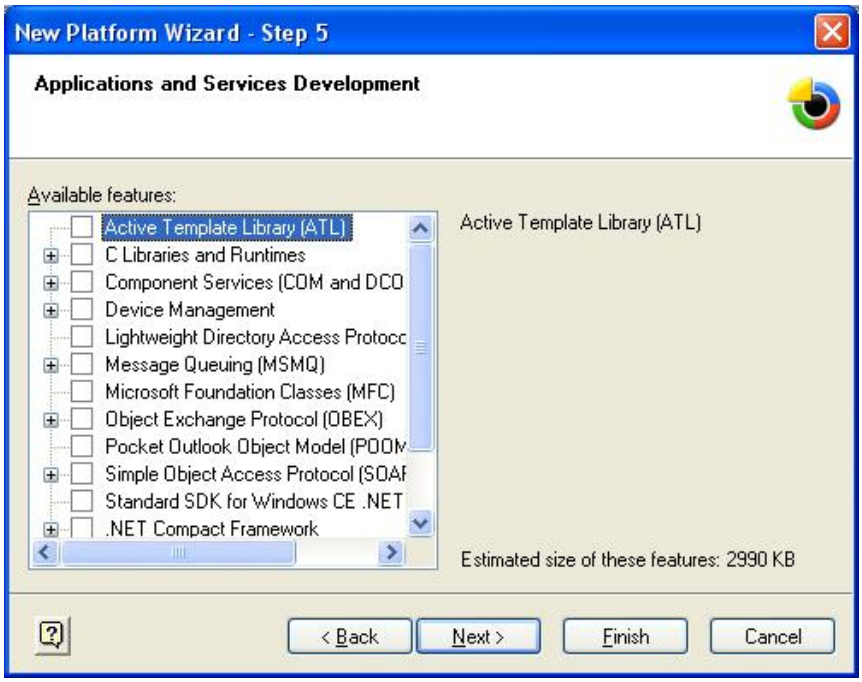


Figura 19 – Escolha de aplicações e serviços de desenvolvimento

Fonte: construído a partir do *Plataform Builder*

- ***Active Template Library* (ATL)** – é um conjunto de classes C++ baseado no modelo que permite criar objetos COM. Ele tem suporte para estes recursos incluindo

implementações de ações, dupla interface, pontos de conexão, interfaces modulares e controles *ActiveX*.

- ***C libraries and Runtimes*** – bibliotecas de linguagem C.
- ***Component Services (COM and DCOM)*** - COM (*Component Object Model*) que é a base da tecnologia do *Microsoft Transaction Server* (MTS), que permite rodar um objeto COM em outras máquinas na rede, permitindo o uso de arquitetura "3-tier" em sistemas operacionais da Microsoft (Windows 95 e Windows NT). Para entender melhor é feita uma analogia com DLLs. Uma DLL é um conjunto de funções em um formato padrão do sistema operacional Windows. As DLLs são códigos compilados e "linkeditados", prontos para uso: o sistema operacional pode carregá-las e executá-las a qualquer momento. O COM é uma versão OOP (Programação orientada a objeto) da DLL, um padrão a ser utilizado para criar objetos executáveis. Sobre este padrão foram construídos outros como OLE (*Object Linking and Embedded*) e *ActiveX*. O Padrão COM baseia-se nas DLLs, mas usa um modelo de objeto extensível e reusável. Os objetos COM estão ficando mais importantes com o passar do tempo e têm as seguintes vantagens: a) Permitem a criação de objetos executáveis, independentes da linguagem usada no desenvolvimento. b) São usados pelos controles *ActiveX* .
- ***Device manager*** – Responsável pelas notificações de quando há novas aplicações, *updates* no *software* e analisar os certificados dos *device drivers*.
- ***Lightweight Directory Access Protocol (LDAP) Client*** - O LDAP existe como um diretório de informações no qual você define usuários e grupos uma vez e os compartilha por diversas máquinas e diversos aplicativos.
- ***Message Queuing (MSMQ)*** - permite uma comunicação assíncrona entre aplicações. Uma aplicação pode deixar uma mensagem em uma fila de mensagens e outra lê esta mensagem posteriormente.
- ***Microsoft Foundation Classes (MFC)*** – Os componentes MFC encapsulam as funcionalidades que se espera encontrar nas aplicações desenvolvidas para Windows, tais como as barras de ferramentas (*toolbars*) e as barras de estado (*status bars*), as caixas de edição (*edit-boxes*), as caixas de listas (*list-boxes*), as caixas combinadas (*combo-boxes*), os diálogos comuns usados para impressão e pré-visualização da impressão, ler e salvar arquivos, interfaces de documentos simples ou múltiplos, troca e validação de dados em diálogos, caixas de diálogos, acesso a bases de dados, ajuda sensível ao contexto, etc.

- ***Object exchange Protocol (OBEX)*** - O protocolo de troca de objeto (OBEX) é um protocolo que facilita a comunicação binária entre dispositivos sem fio. Esta característica suporta dois transportes diferentes no sistema operacional, *Bluetooth* e em protocolos infravermelhos (IrDA).
- ***Pocket Outlook Object Model (POOM)*** - Com o modelo de objeto do Outlook de bolso (POOM), podem-se manipular dados de contato, do calendário e das tarefas no correio eletrônico. POOM é um subconjunto do modelo *desktop* de objeto do Outlook.
- ***Simple Object Access Protocol (Soap) Toolkit*** - é um protocolo leve para troca de informações. Parte da sua especificação é composta por um conjunto de regras de como utilizar o XML para representar os dados. Outra parte define o formato de mensagens, convenções para representar as chamadas de procedimento remoto (RPCs) utilizando o SOAP, e associações ao protocolo HTTP.
- ***Standard SDK for Windows CE.NET*** - As funções dos serviços da base dão o acesso das aplicações aos recursos do computador e às características do sistema operacional, tais como a memória e arquivos DLLs. Quando uma plataforma é desenvolvida deve-se também desenvolver um kit de desenvolvimento padrão (SDK) para Windows CE. A aplicação desenvolvida deve funcionar somente com as características fornecidas pelo SDK padrão. Se por ventura o usuário não programar o aplicativo com o SDK construído para a imagem em questão, esse aplicativo não funcionará.
- ***.NET Compact Framework*** - é um modelo de programação da Microsoft, para desenvolvimento, colocação e execução de serviços web XML e de todos os tipos de aplicações: *desktop*, móveis ou baseadas em *Web*. Em função do *.NET Compact Framework* ser um sub-conjunto do *.NET Framework*, os desenvolvedores podem usar os conhecimentos de programação e os códigos existentes através de dispositivos, *desktop* e servidores. A Microsoft liberou no *Visual Studio* .NET 2003 o *Smart Device* Application, que contém o .NET Compact Framework. Isso significa que qualquer desenvolvedor do *Visual Studio* que tem experiência com o *.NET Framework* será capaz de desenvolver aplicações para qualquer dispositivo que rode *.NET Compact Framework*. Devido a isso, muitos desenvolvedores do *Visual Studio .NET* podem ser desenvolvedores para *Smart Device*.
- ***SQL Server CE 2.0*** – Programa de controle banco de dados.
- **XML** - A *Extensible Markup Language* (XML) (em português, linguagem de marcação extensível), é uma linguagem de marcação de uso geral recomendada pela

W3C (*World Wide Web Consortium*) que serve para criar linguagens de marcação de uso específico. É um subconjunto simplificado da SGML, capaz de descrever vários tipos diferentes de dados. A sua função principal é de facilitar o compartilhamento de dados através de vários sistemas, particularmente sistemas ligados através da Internet.

Na figura 20, mostra a tela de aplicativos que serão visualizados pelo usuário final, aquele que fará uso da imagem após o sistema operacional configurado.

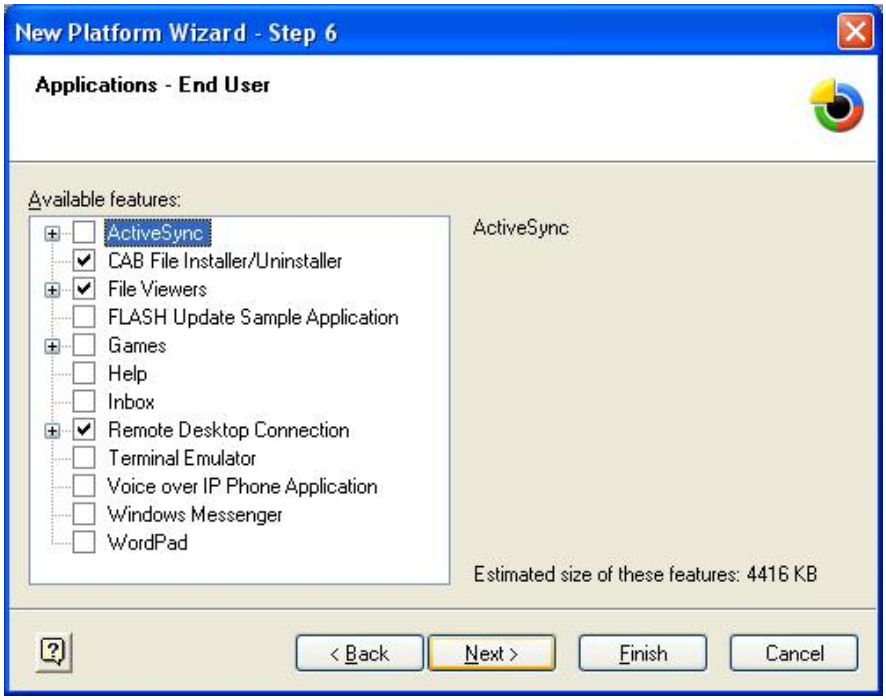


Figura 20 – Aplicativos para o usuário final

Fonte: construído a partir do *Plataform Builder*

Os aplicativos que o usuário final pode agregar a imagem são:

- ***ActiveSync*** - é o programa responsável pelo sincronismo entre um computador Desktop e os dispositivos portáteis baseados em Windows CE.
- ***CAB File Installer/Uninstaller*** – Aplicativo para instalar e desinstalar arquivos do tipo CAB. *Downloads* automáticos dos componentes e configurações necessárias em uma máquina cliente remotas são suportados por arquivos *cabinet* (CAB). Estes arquivos CAB são arquivos compactados padrão Microsoft que garantem downloads eficientes dos componentes de *software ActiveView* necessários. A tecnologia Microsoft *Authenticode* aplica assinaturas digitais aos arquivos CAB para proteger os usuários de componentes adulterados.

- ***File Viewers*** – são aplicativos para visualização de arquivos:

  - *Microsoft Excel Viewer*: Aplicativo para visualizar planilhas do Excel

  - *Microsoft Image Viewer*: Aplicativo para visualizar Imagens

  - *Microsoft PDF Viewer*: Aplicativo para visualizar arquivos em PDF, é a abreviatura de "*Portable Document Format*". Sua principal característica consiste em representar um documento com integridade e fidelidade ao seu formato original.

  - *Microsoft PowerPoint Viewer*: Aplicativo para visualizar apresentações no *PowerPoint*.

  - *Microsoft Word Viewer*: Aplicativos para visualizar documentos do *Word*.

- ***FLASH Update Sample Application*** – Uma demonstração de como instalar um S.O. num dispositivo

- ***Games*** – são aplicativos para entretenimento:

  - *Freecell*: Jogo de carta para Windows CE

  - *Solitaire*: Jogo de carta para Windows CE

- ***Help*** – Um aplicativo baseado em HTML com sistema de ajuda.

- ***Remote Desktop Protocol (RDP)*** – Um serviço que permite que um *thin Client*, com um (wbt) *Windows-based terminal*, se comunicar com um usuário de terminal através de uma LAN, WAN ou por meio de uma conexão dial-up.

- ***Terminal Emulator*** – uma aplicação demonstrativa que permite o usuário criar uma simulação do TTY ou do VT100(aplicativos para conexão remota).

- ***Voice over IP Phone Application*** – aplicativo para comunicação de voz sobre ip.

- ***Windows Messenger*** – aplicativo que permite a comunicação de usuários através de mensagens de texto ou voz e também transferência de arquivos.

- ***Word Pad*** – aplicativo para processamento de textos.

Na figura 21, é a tela onde é feita a opção dos serviços do sistema operacional:

- ***Battery driver*** – *device driver* para gerenciamento de energia, como por exemplo, nível de carga da bateria principal, bateria de backup e alimentação da fonte de corrente alternada.
- ***Serial Port Support*** – *device driver* para a porta serial.

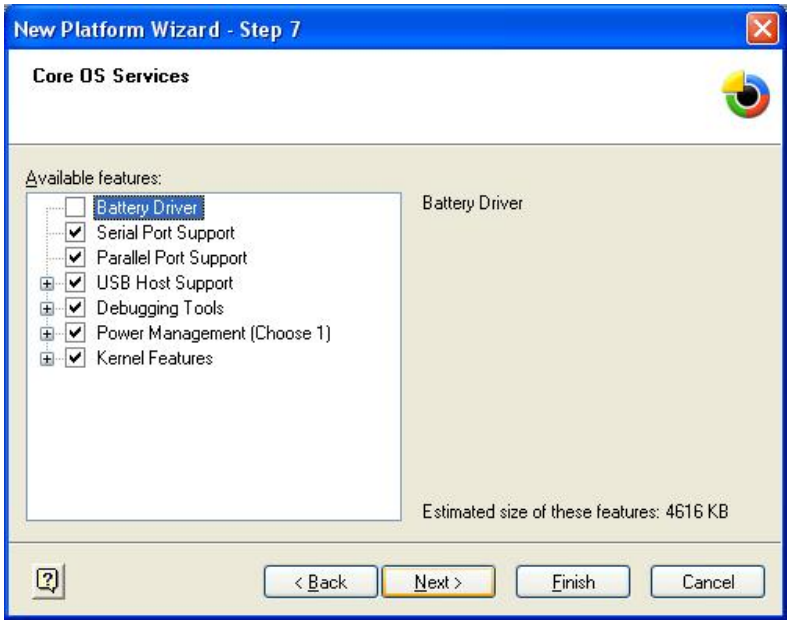


Figura 21 – Serviços de controle de *hardware*

Fonte: construído a partir do *Plataform Builder*

- ***Parallel Port Support*** – *device driver* para a porta paralela.
- ***USB Host Support*** – *device driver* para dispositivos USB, como por exemplo, mouse, teclado e impressora.
- **Debugging Tools** – Ferramentas de procura e solução de erros (*bugs*) nos *devices drivers*
- ***Power Management*** – ferramenta de gerenciamento de energia, essa ferramenta gerencia dispositivos que podem ser desligados para evitar o desperdício de energia.
- ***Kernel features*** – Contém a funcionalidade do núcleo do sistema operacional Windows CE. Este núcleo tem as seguintes funcionalidades: Arquitetura da memória, escalonador de processos, desempenho em tempo real, chamadas do sistema operacional.

A figura 22 mostra a tela da escolha dos serviços de Comunicação e rede:

- ***Networking Features*** – são escolhidos serviços de rede como:

  - *Extensive Authentication Protocol*: protocolo de autentificação que usa usuário e senha.

  - *Firewall*: serviço de rede que tem por função regular o tráfego de rede entre redes distintas e impedir a transmissão e/ou recepção de dados nocivos ou não autorizados de uma rede a outra. Dentro deste conceito incluem-se, geralmente, os filtros de pacotes

  -*Internet Connection Sharing* (ICS): serviço de rede onde múltiplos dispositivos podem compartilhar de uma única conexão da Internet.

  -*Networking Bridging*: Serviço de rede que interconecta segmentos de rede, criando rotas específicas.

Figura 22 – Configuração de serviços de comunicação e redes

Fonte: construído a partir do *Plataform Builder*

-*Network Utilities*: coleção de aplicativos que são úteis para resolução de problemas de rede. Comandos como *ping, route e ipconfig* estão neste item.

-*Real-time Communications* (RTC): coleção de API que são utilizadas na comunicação VoIP.

- *TCP\IP*: conjunto de protocolos para configuração de rede *Ethernet*. Inclui protocolos como IP, ARP, IGMP, TCP, UDP e DHCP.

- *TCP\IPv6 Support*: Um conjunto de protocolos usados numa geração acima do TCP\IP, com 128 bits de endereçamento de rede contra 32 bits do *TCP\IP*.
-*Voice over IP Phone* (VoIP): serviço de rede do VoIP.

- ***Network (Local Area Network (LAN))*** – Protocolos de rede relacionados à tecnologia *Wireless*.
- ***Network (Personal Area Network (PAN))*** - Protocolos de rede *Bluetooth* e *Irda*.
- ***Network (Wide Area Network (WAN))*** – Protocolos de rede Dial-up, PPPoE e TAPI 2.0.
- ***Servers*** – Conjunto de serviços de rede para construção de servidores de: Impressão, FTP, HTTP e Telnet.

A figura 23 mostra a tela onde é feita a configuração de sistema de arquivos e armazenamento de dados:

- ***File and Database Replication*** – serviço de sincronização de dados entre memórias diferentes. As bases de dados do sistema operacional também podem ser sincronizadas.
- ***File system Internal*** – O sistema de arquivo pode ser lido através de RAM e ROM ou somente de uma ROM.

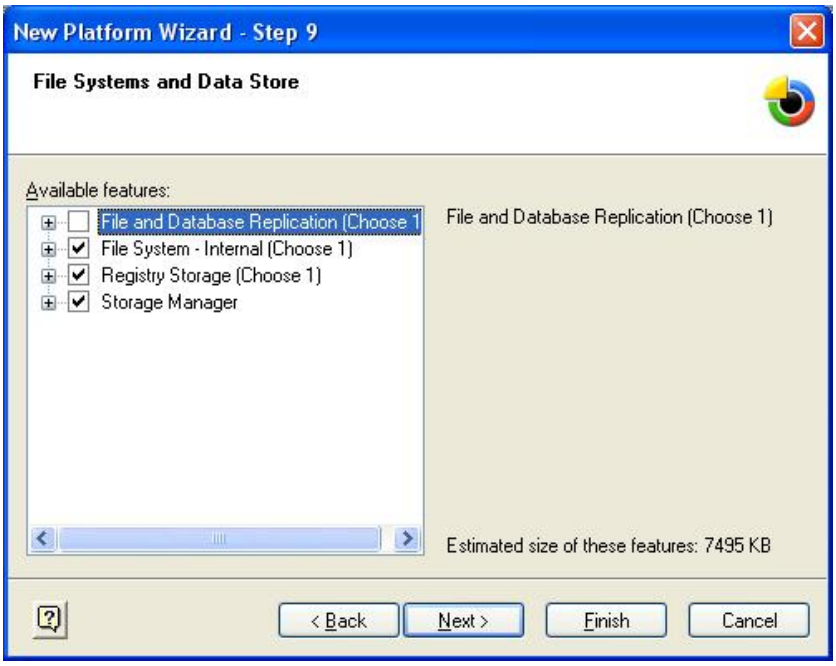


Figura 23 – Sistema de arquivos e armazenamento de dados.

Fonte: construído a partir do *Plataform Builder*

- ***Registry Storage*** – registro de arquivos que pode ser salvo na RAM ou ROM.
- ***Storage manager*** – *device drivers* para dispositivos de armazenamento como: disco rígido, *Cd-Rom*, *pen drive*, cartão *flash*, etc.

A figura 24 demonstra a tela onde é feita a escolha das fontes que serão inseridas nos aplicativos de texto presente na imagem.

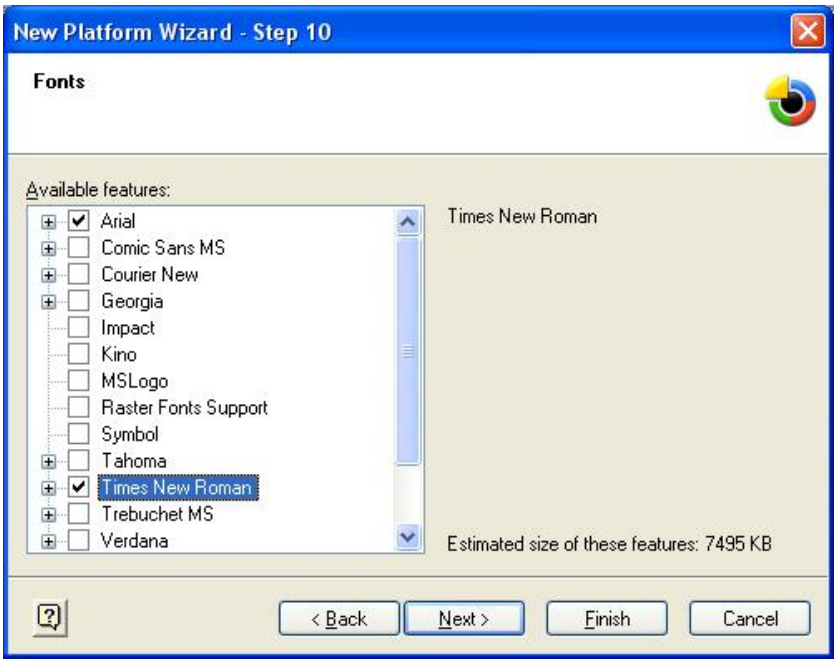


Figura 24 – Fontes
Fonte: construído a partir do *Plataform Builder*

A figura 25 mostra a seleção de linguagem e quais caracteres a imagem terá suporte.

Figura 25 – Suporte a linguagem
Fonte: construído a partir do *Plataform Builder*

A figura 26 mostra os serviços para os clientes de Internet:

- ***Browser application*** – é feita a escolha do browser, ou seja, qual aplicativo o usuário fará o acesso a internet, as possibilidades são *Internet Explorer 6.0* para Windows CE e *Pocket Internet Explorer* HTML *View*. A principal diferença entre os dois é que o *Pocket Internet Explorer* é usado para ambiente que tenham restrição de memória de armazenamento e *display* pequenos como os de um celular, por exemplo.
- ***Scripting*** – As duas opções existentes se relacionam os aplicativos acessados (*scripts*) via internet de programação Java e Visual Basic.

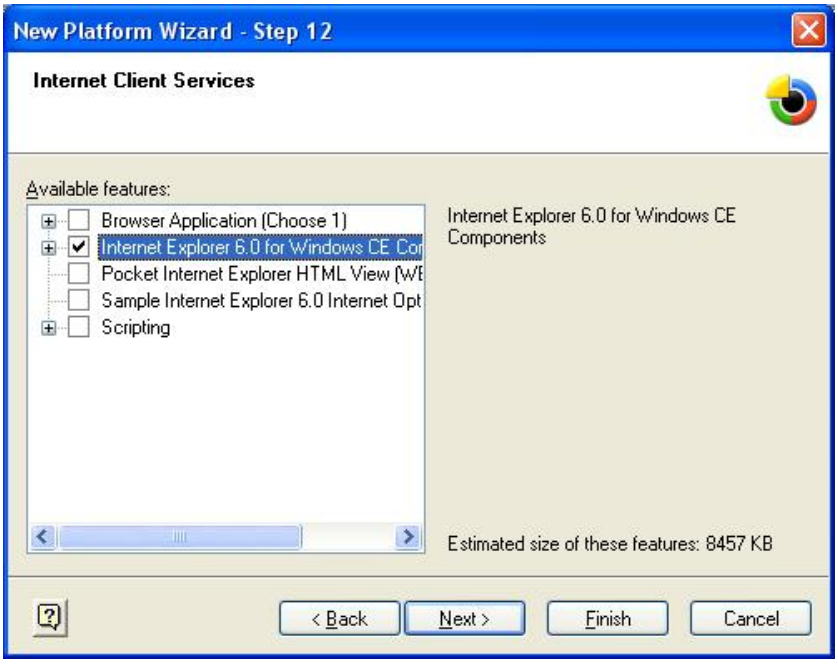


Figura 26 – Aplicativos de internet
Fonte: construído a partir do *Plataform Builder*

A figura 27 mostra a seleção de Tecnologia Multimídia onde é possível escolher:

- ***Basic Multimedia*** – onde é agregada a imagem, aplicativos que toquem arquivos de áudio com a extensão mp3 e wmv.
- ***Multimidia Components*** – onde é agregada a imagem, aplicativos multimídia como *DVD player*, *Windows Media Player*, *codecs* de áudio e vídeo.

Figura 27 – Aplicativos multimidia

Fonte: construído a partir do *Plataform Builder*

A figura 28 mostra a escolha dos serviços de segurança:

- ***Authentication Services*** – serviços de segurança para autentificação de usuários usuário, e proteções de mensagens através de serviços de segurança como SSPI. Dentro da SSPI, existem duas opções NTLM e o *Kerberos*. Pode-se também fornecer ser próprio pacote de segurança e adicioná-lo ao registro para aplicações.
- ***Cryptography Services (CryptoAPI 1.0) with high Encryption P*** – serviços que permitem aos programadores a aplicação de segurança usando criptográfica. Assim como uso de certificados digitais.

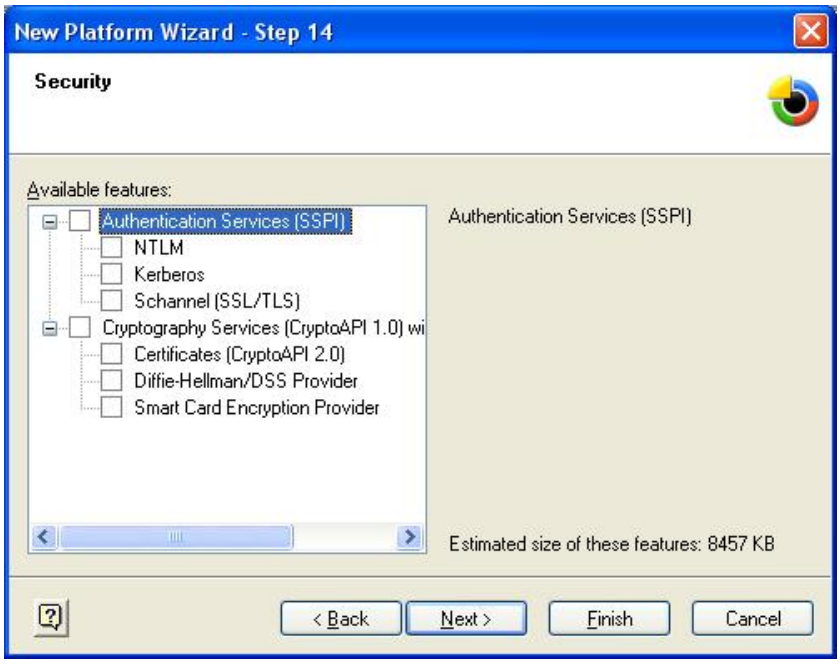


Figura 28 – Serviços de autentificação e criptografia

Fonte: construído a partir do *Plataform Builder*

Na figura 29, é demonstrada a tela onde é feita a escolha do *Shell*:

- ***Shell*** – pode ser um interpretador de comandos gráfico como Windows XP (*Graphical Shell*) ou pode ser um janela de console somente em modo texto como MS-DOS (*Command Processor*).

Figura 29 – *Shell*

Fonte: construído a partir do *Plataform Builder*

A figura 30 da interface do usuário, ou seja, opções como: acessibilidade, painel de toque podem ser escolhidos nessa etapa.

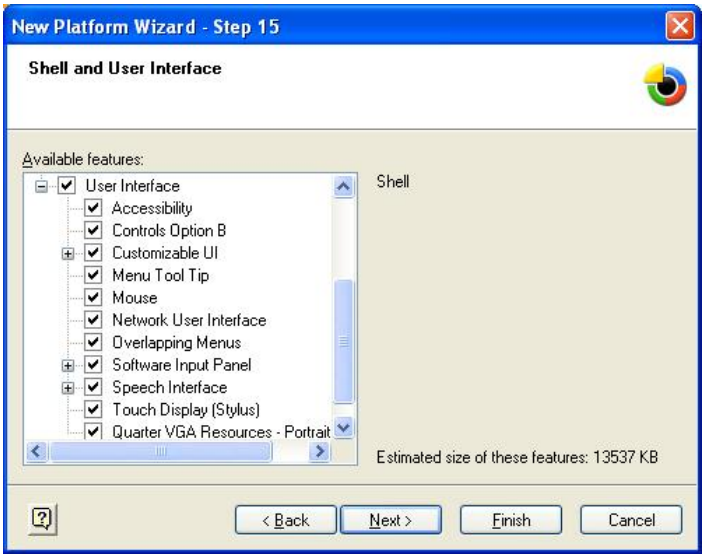


Figura 30– *User interface*

Fonte: construído a partir do *Plataform Builder*

A figura 31 mostra a tela onde aparecem mensagens referentes a protocolos específicos escolhidos na imagem, tais mensagens trazem apontamentos importantes para uso dos mesmos e também como achar documentação adicional sobre tais assuntos.

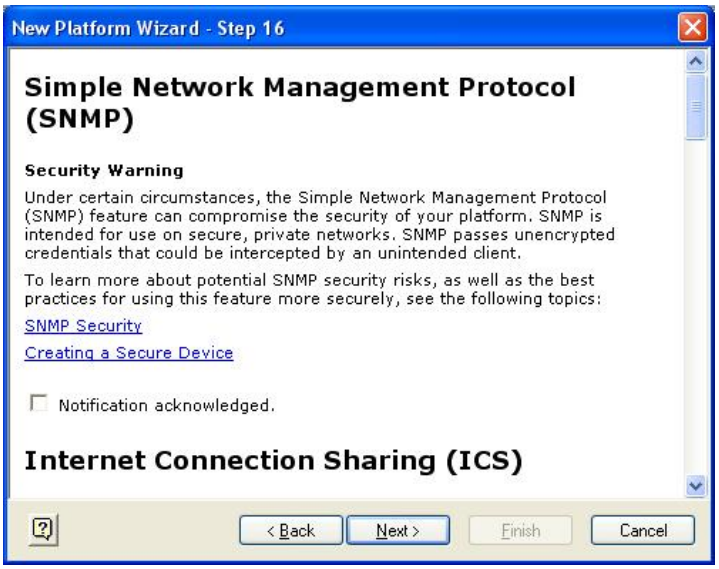


Figura 31 – Notificação adicional

Fonte: construído a partir do *Plataform Builder*

Na figura 32 são mostradas as opções de construção (build) do arquivo da imagem:

- ***Build the debug version of your plataform after this wizard closes*** – construção da imagem mais as ferramentas de *debug*.
- ***Build the release version of your plataform after this wizard closes*** – construção da versão final da imagem (Nk.bin), ou seja o arquivo contendo o sistema operacional estará pronto para o *download*.
- ***Modify the build options for your plataform after this wizard closes*** – modificação dos parâmetros de configuração de como construir a imagem, ou seja, não construir a imagem, para poder alterar, por exemplo, os *devices drivers.*

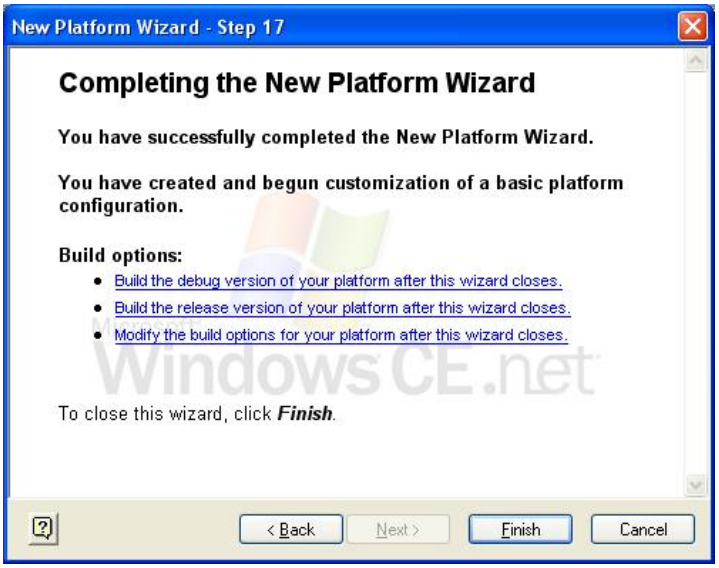


Figura 32 – Completando a nova plataforma

Fonte: construído a partir do *Plataform Builder*

Depois desta plataforma pronta deve ser feita a compilação da mesma, esse procedimento deve ser feito selecionando *build plataform* no menu *build*.

Drivers e aplicativos podem ser adicionados à imagem depois da imagem construída, mas uma nova compilação deve ser feita.

3.8 ADIÇÃO DE *DEVICE DRIVERS*

Depois do processo de construção da imagem pode–se adicionar *device drivers* que sejam necessários a imagem, com o seguinte procedimento:

- No menu *catalog*, é necessário expandir menu *device driver*, como mostra a figura 33:

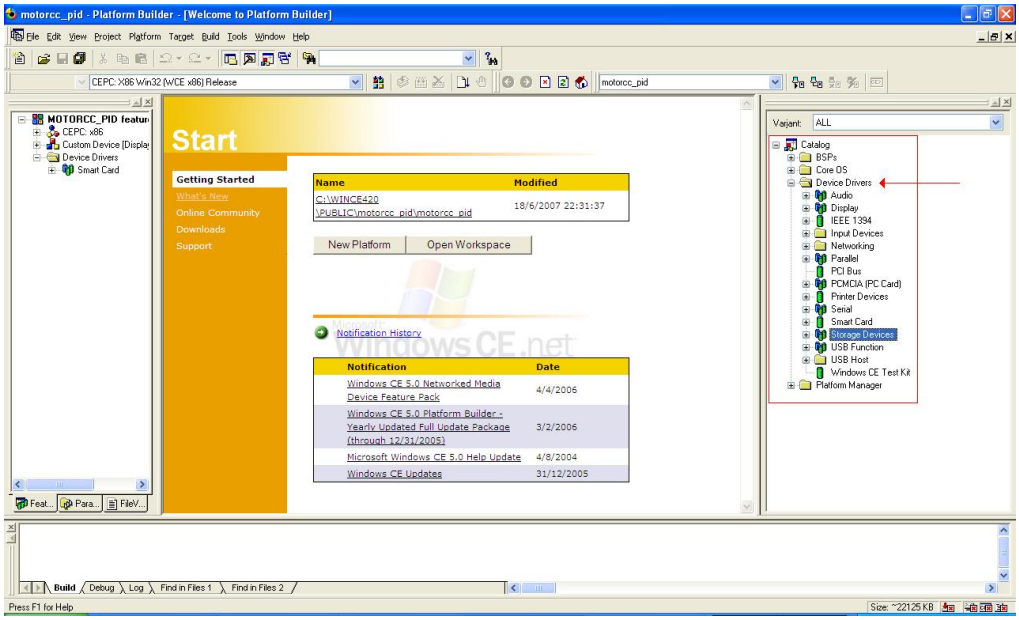


Figura 33 – Menu *device drivers*

Fonte: construído a partir do *Plataform Builder*

- Depois de feita a escolha do *driver* apropriado, clica-se com o botão direito do mouse e a opção escolhida é *add to plataform*, como mostra a figura 34:

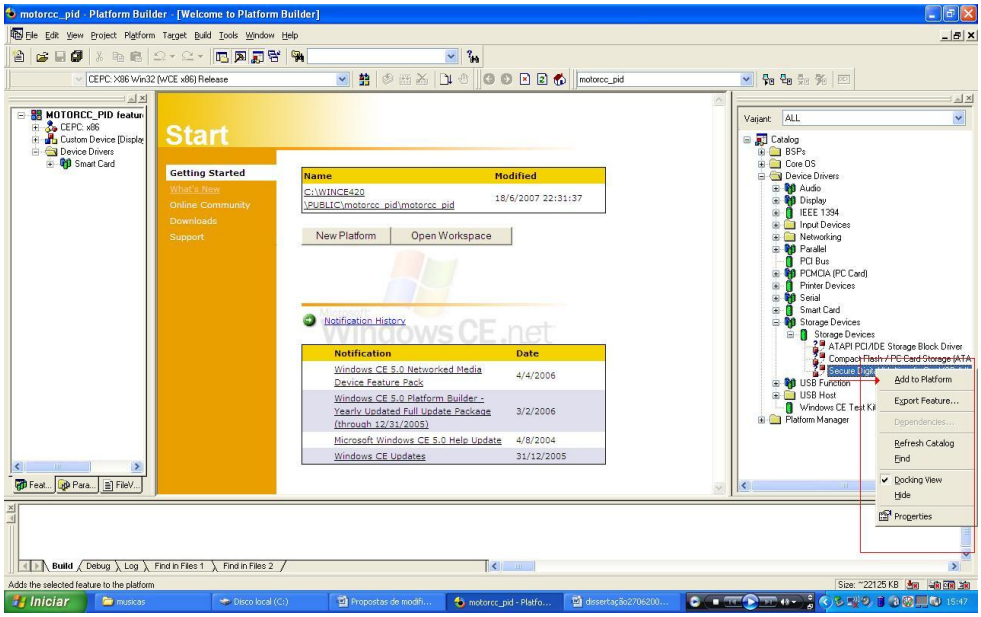


Figura 34 – *Add to Plataform*

Fonte: construído a partir do *Plataform Builder*

Ressalta-se que depois desse processo deve ser feita uma nova compilação da plataforma onde foi adicionado o *driver*.

3.9 CRIAÇÃO DE UM DISQUETE DE BOOT

Para que o micro aplicativo receba a imagem construída é necessário que se tenha um gerenciador de boot, no caso deste trabalho é usado um disquete (1,44MB) de boot, a construção desse disquete está descrita a seguir:

- Acessar um programa chamado *Websetup.exe* que se encontra numa pasta da instalação do *Platform builder*, um exemplo do caminho que pode se encontrar esse arquivo é: C:\Arquivos de programas\Windows CE Platform Builder\4.20\cepb\utilities. Esse programa mostrado na figura 35.

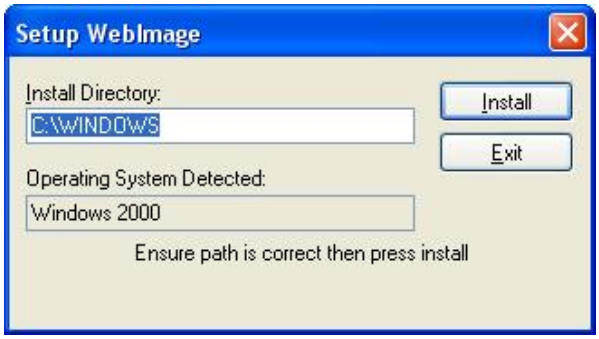


Figura 35 – *Websetup*

Fonte: construído a partir do *Plataform Builder*

- Depois de instalado o Programa *WebImage*, na mesma pasta que se encontra o executável *Websetup*, encontra-se um arquivo chamado *Cepcboot*, tal arquivo é responsável pela criação de um disquete de boot, ou seja, um disco responsável pela a incialização da máquina onde é feito o *download* da imagem do sistema operacional. Na figura 36, é mostrada a tela do programa *Cepcboot*.

Figura 36 – *Cepcboot*

Fonte: construído a partir do *Plataform Builder*

Este arquivo criará um disquete de *boot* contendo os arquivos mencionados no quadro 2.

Quadro 2 – Descrição dos arquivos contidos no disquete de *boot*

Arquivo	Descrição
Autoexec.bat	Arquivo de lote requerido no sistema operacional Microsoft MS-DOS.
Command.com	Arquivo de Comandos requerido no sistema operacional Microsoft MS-DOS.
Config.sys	Arquivo de Sistema requerido no sistema operacional Microsoft MS-DOS.
Drvspace.bin	Arquivo binário que ajusta os parâmetros e os grava no arquivo Drvspace.ini para a configuração do *drive*.
Eboot.bin	Arquivo binário que contém o *Ethernet boot loader* (faz a preparação do hardware para receber a imagem através a rede local).
Himem.sys	Arquivo de Sistema requerido no sistema operacional Microsoft MS-DOS.
Io.sys	Arquivo de Sistema requerido no sistema operacional Microsoft MS-DOS.
Loadcepc.exe	Arquivo executável que roda o arquivo binário *Eboot.bin*.
Msdos.sys	Arquivo de Sistema requerido no sistema operacional Microsoft MS-DOS.
Readme.txt	Arquivo de texto que contem instruções sobre o procedimento de boot.
Sboot.bin	Arquivo binário que contém o serial *boot loader* (faz a preparação do hardware para receber a imagem através de comunicação serial).
Sys.com	Aplicativo do sistema operacional Microsoft MS-DOS.
Vesatest.exe	Aplicativo do sistema operacional MS-DOS que testa a compatibilidade da placa de vídeo com o *driver* padrão do Windows CE. No arquivo Readme.txt estão contidas informações adicionais sobre esse assunto.

Fonte: O autor

- Nesta opção escolhida de boot para a seqüência de download do SO é necessário que o sistema em questão contenha um Servidor DHCP (*Dynamic Host Configuration Protocol*), ou seja, um computador que forneça dinamicamente um endereço IP válido na rede, para que o computador que contém o *Platform builder* instalado consiga comunicar-se com o computador que será feito o *download* da imagem. Este processo é demonstrado na figura 37.

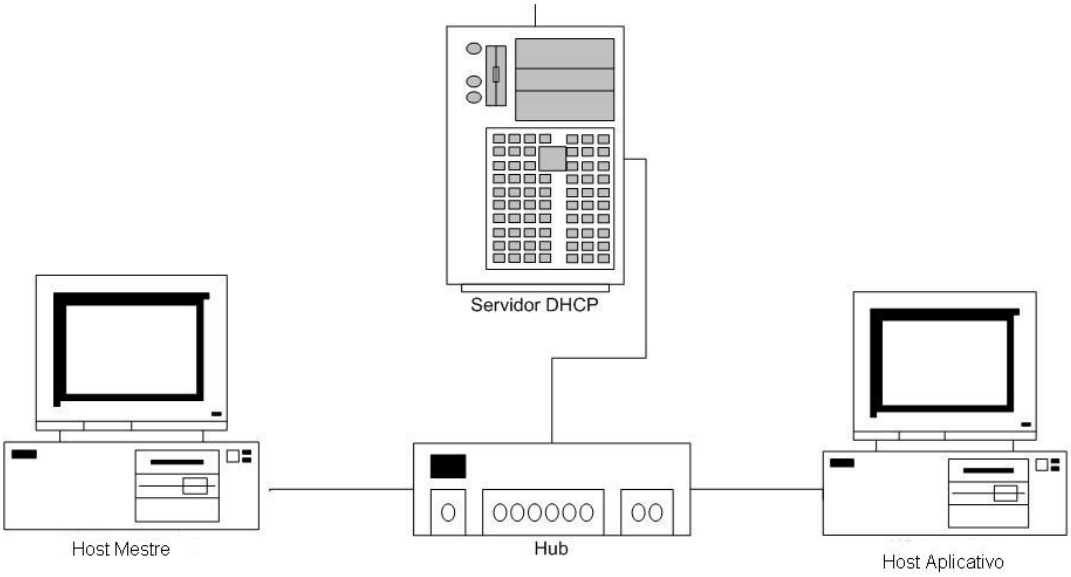


Figura 37 – Servidor DHCP
Fonte: construído a partir do *Plataform Builder*

- Após a execução do procedimento anterior, executa-se um *boot* com o disquete citado anteriormente.

3.10 *BOOT* E *DOWNLOAD* DE UMA IMAGEM

Assim que a imagem estiver construída e compilada, deve-se dar um *boot*, com o disquete feito com o processo descrito anteriormente, na máquina a ser feita o *download*. Assim que a máquina completar o *boot* a seguinte configuração deve ser feita:

- No menu *Target* é selecionado *Configure Remote Connection* onde deve ser selecionado *Ethernet* nas duas caixas de texto, tanto *Download* como *Kernel transport*, a figura 38 demonstra o resultado obtido:

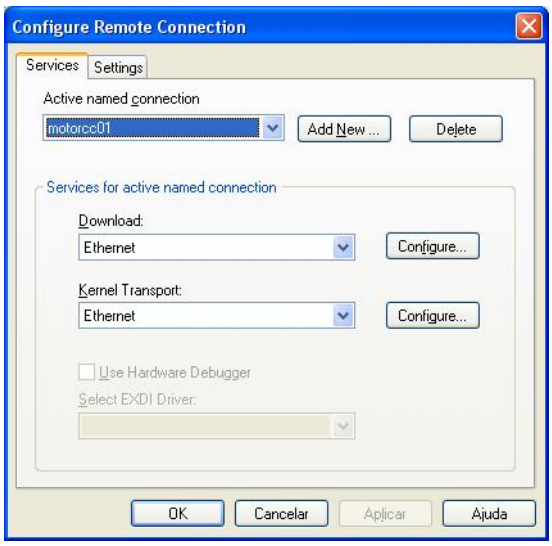


Figura 38 – Configuração de conexão remota

Fonte: construído a partir do *Plataform Builder*

Na figura 38 existe ainda a opção serial, onde pode ser feito o *download* do sistema operacional através de um cabo ligado na serial da plataforma e a opção *emulator* 4.20, que é a simulação de um *download* num emulador de plataformas.

Nesta mesma tela clica-se no botão *Configure* e é feita a escolha do *hardware* onde se deseja fazer a *download* da imagem, como é mostrado na figura 39.

Figura 39 - Seleção do dispositivo

Fonte: construído a partir do *Plataform Builder*

- O ultimo passo é fazer o *download* propriamente dito da imagem, no menu *target* selecionando o item *download\initialize* mostrado na figura 40

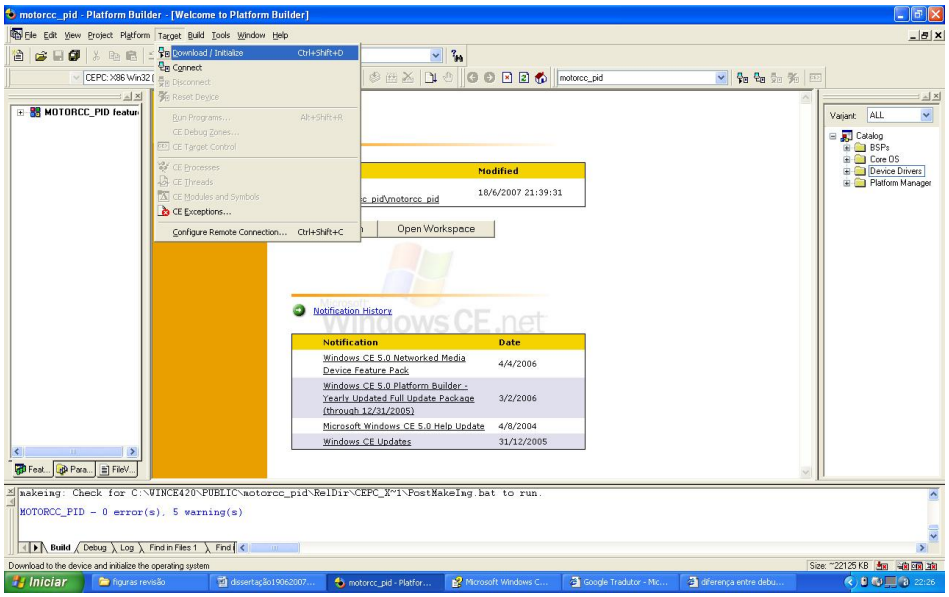


Figura 40 – *Download\initialize*

Fonte: construído a partir do *Plataform Builder*

Após a construção de uma imagem no *Platform builder* os passos para se fazer o *download* de uma imagem no micro de desenvolvimento são descritos a seguir:

3.11 PROGRAMANDO UM APLICATIVO

A programação de um aplicativo no Windows CE deve ser feita após ter sido criada no mínimo uma imagem de um sistema operacional completo.

Após a construção de uma imagem feita no *Plataform Builder* a programação de um aplicativo é feita num programa chamado *Microsoft Embedded C++4.0* que pode ser instalado usando o mesmo DVD do *Plataform Builder*.

A tela inicial do programa é mostrada na figura 41.

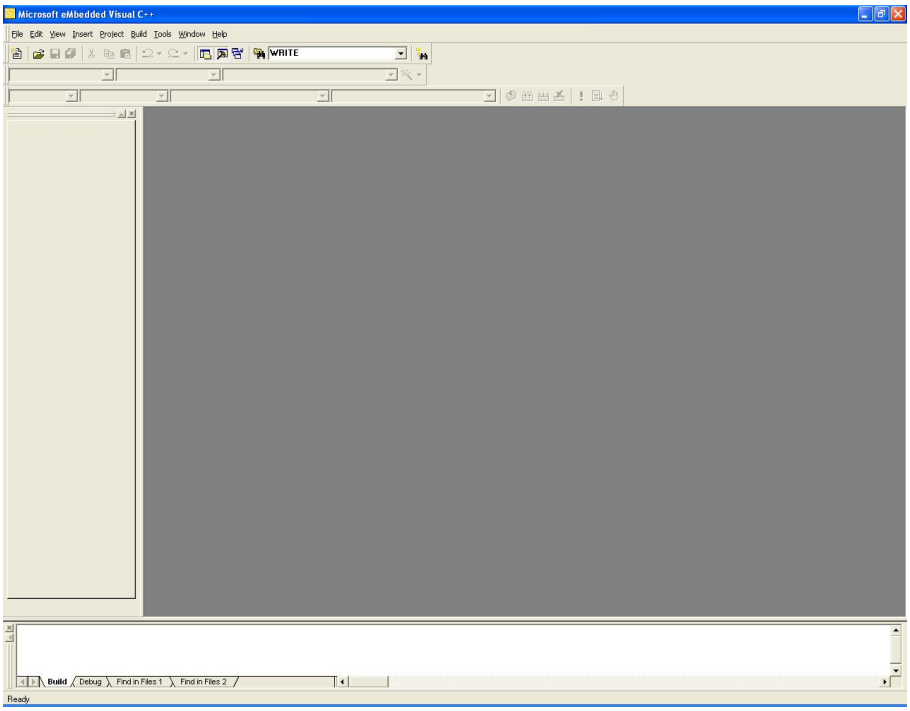


Figura 41 – *Embedded* C++ 4.0

Fonte: construído a partir do *Plataform Builder*

Depois de aberto o programa clica-se em *NEW* e as opções da figura 42 aparecem.

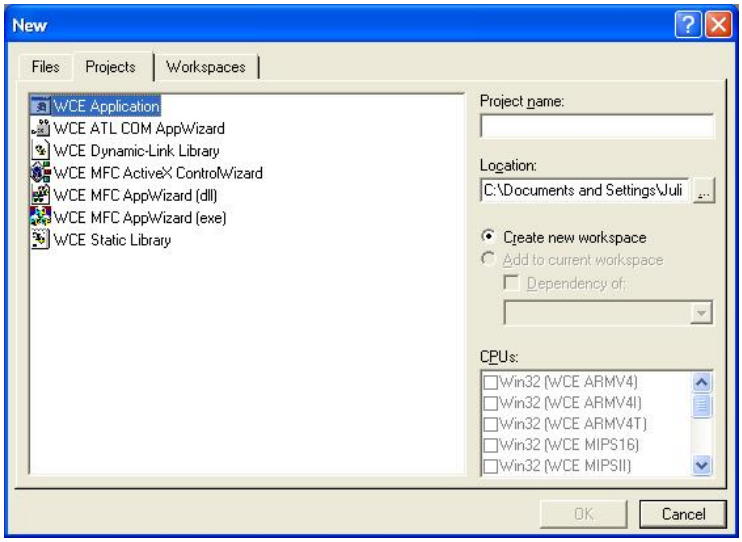


Figura 42 – Opções de programação

Fonte: construído a partir do *Plataform Builder*

Alguns arquivos descritos a seguir:

- **WCE *application*** – programação de aplicativos em C++, projeto vazio, baseado numa aplicação “Hello World”.
- **WCE *Dynamic-Link Library*** – programação de DLL.
- ***WCE ATL COM AppWizard*** – programação de objetos COM.
- ***WCE Static Library*** – É um arquivo que contém objetos, funções, e dados que estão ligados a um programa quando um arquivo executável é construído.

3.12 TRANSFERINDO DEFINITIVAMENTE A IMAGEM

Para o armazenamento local de uma imagem pode ser feito vários processos, dentre eles:

- HD (*hard disk*) – Pode se fazer o armazenamento de uma imagem num disco rígido formatando em modo DOS e salvando o arquivo gerado pelo *Plataform builder* chamado NK.bin no mesmo HD.
- *CD-ROM* – Pode-se fazer um armazenamento de uma imagem num CD criando um disco com sistema operacional do disco de *boot* criado anteriormente e salvando o arquivo NK.bin junto.

CAPÍTULO 4

IMPLEMENTAÇÃO E RESULTADOS

Na seqüência serão apresentados dois exemplos de aplicação do Windows CE, um em controle e outro em automação. Nestes exemplos serão detalhados os aspectos de construção da imagem no Bloco Gerente, *download* da imagem construída no Bloco Aplicativo e elementos de *hardware* e *software* implementados no Bloco de Interfaces.

O exemplo na área de controle é baseado na construção de um controlador de velocidade PID de um motor CC e o exemplo na área de automação é baseado na construção de um CLP virtual.

4.1 CONSTRUÇÃO DA IMAGEM NO BLOCO GERENTE

A imagem construída para a execução dos experimentos práticos tem um conjunto de características. É importante ressaltar que estas características devem ser observadas na futura construção da imagem. As mesmas definem os elementos a serem integradas, as configurações de hardware, os *device drivers* utilizados, entre outros. O conjunto completo de características é apresentado:

- ✓ *Board Support Packages (BSPs)*
  - *CEPC:X86*
- ✓ *Plataform Configuration*
  - *Custom Configuration*
- ✓ *Custom Device*
  - *Custom Device with Shell and Grafical User Interface (GUI)*
- ✓ *Aplications and Services Development*
  - *Active Template Library*

- *C libraries and Runtimes*
- *Standard SDK for Windows CE.NET*
- *.Net Compact Framework*

✓ *Aplications - End User*

- *Cab files Installer/Unistaller*
- *File Viewers*
- *Remote Desktop Connection*

✓ *Core OS Services*

- *Battery Driver*
- *Serial Port Support*
- *Parallel Port Support*
- *Usb Host Support*
- *Debugging Tools*
- *Power Management*
- *Kernel Features*

✓ *Communication Services and Networking*

- *Networking – Local Area Network*

✓ *File Systems and Data Store*

- *File system – Internal*
- *Registry Storage*
- *Storage Manager*

✓ *Fonts*

- *Arial*
- *New Times Roman*

✓ *Internet Client Services*

- *Internet Explorer 6.0 for Windows CE Components*

- ✓ *Multimedia Technologies*
    - *Audio*

- ✓ *Shell and User Interface*
    - *Shell*
    - *User Interface*

Essas características adicionadas à imagem, foram usadas na implementação das duas experiências, tanto na experiência de controle quanto automação.

4.2 EXEMPLOS DE APLICAÇÃO

Tomando como base o diagrama de blocos da figura 43, é demonstrado a seguir um exemplo de configuração de uma plataforma baseada no sistema operacional Windows CE. Juntamente com a plataforma um exemplo de controle de elementos externos através deste sistema operacional.

A figura 16 apresenta o diagrama de blocos do exemplo implementado.

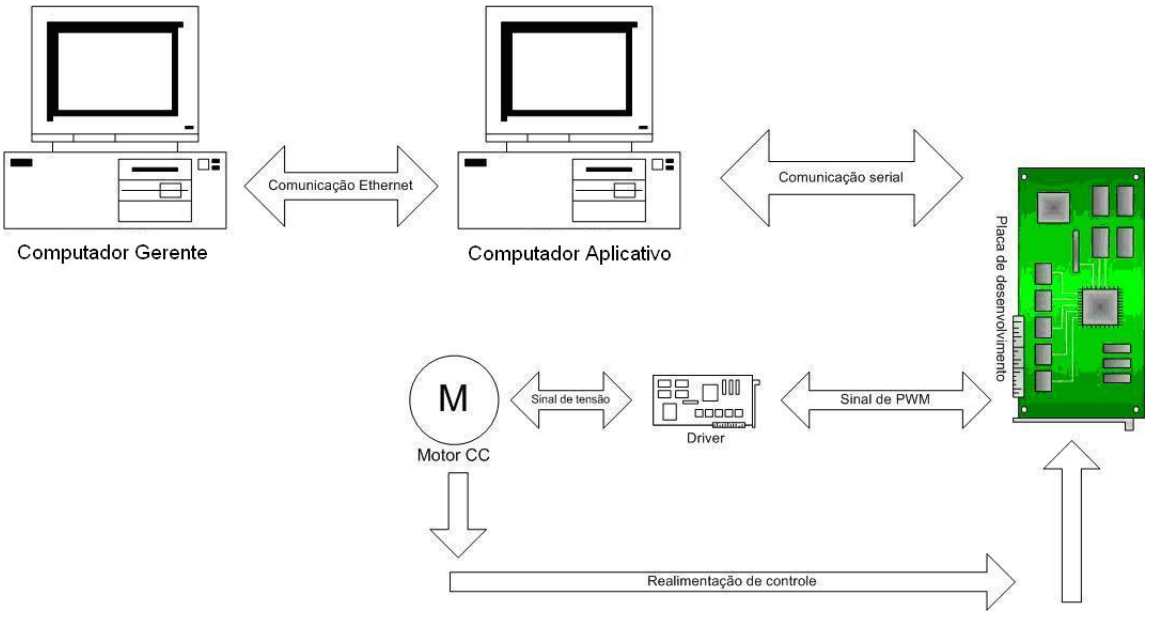


Figura 43 – Diagrama de blocos descrevendo o exemplo de aplicação.

Fonte: O Autor

A experiência descrita na figura 16 descreve o controle de velocidade PID de um motor CC.

4.2.1 Implementação de um programa de controle PID no Bloco Aplicativo e Construção e programação da placa de desenvolvimento no Bloco de Interfaces

Implementação do PID

Para exemplificar a proposta em questão, na área de controle, um controle PID foi escolhido. Esse programa foi desenvolvido no Ambiente instalado chamado Microsoft *Embedded* C++ (BARR, 1999).

Esse programa faz o controle PID em tempo real de um motor CC. Para fazer a interface entre o motor cc e o *Host* Aplicativo há uma Placa de desenvolvimento com o propósito de agir como atuador e Sensor. A placa age como atuador lendo dados seriais do controle PID que são enviados pelo PC com imagem do Windows CE e enviando o sinal de PWM (*Pulse-width modulation*) para um motor CC, e como sensor quando recebe um sinal.

O diagrama de blocos da figura 44 demonstra de forma detalhada esse processo:

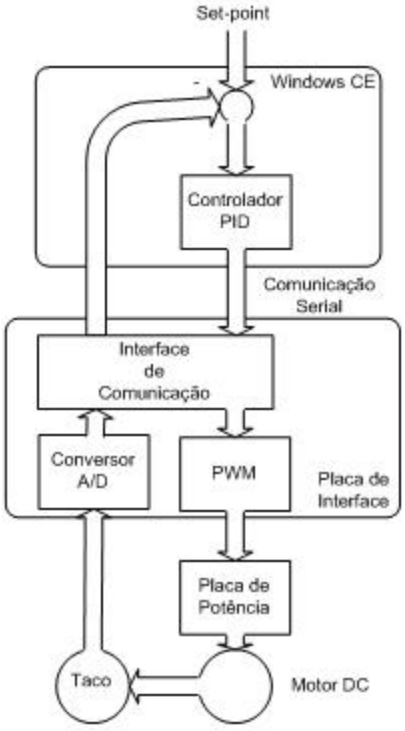


Figura 44 – Diagrama de blocos do controle e placa de desenvolvimento

Fonte: O Autor

O Bloco aplicativo, representado por um microcomputador Pentium 200MHz, 64MB de memória RAM, calcula as variáveis de processo, através de um controlador PID (proporcional, derivativo e integral), e envia para a placa de desenvolvimento através de comunicação serial. A placa de desenvolvimento trata esses valores e transmite em forma de PWM para o motor cc.

O fluxograma da figura 45 demonstra como é feito o cálculo do PID implementado no Bloco Aplicativo.

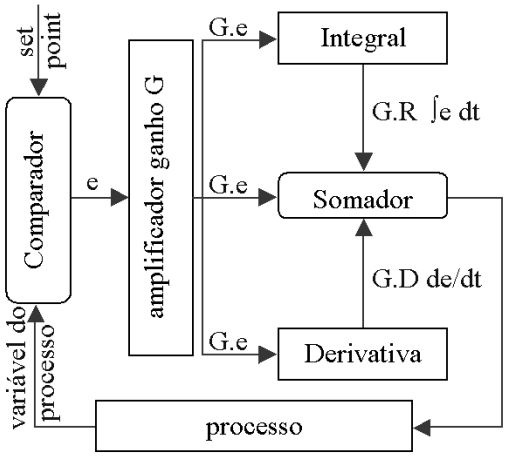


Figura 45 – Fluxograma do funcionamento do PID

Fonte: O Autor

Em função da programação utilizada no Windows CE a rotina a seguir mostra as equações diferença usadas na implementação do programa PID.

As equações a seguir descrevem os parâmetros usados com os ajustes do controlador (ganho proporcional(Kp), tempo derivativo (Td), tempo integral (Ti), tempo de amostragem (T0)), ultimo fixado pelo programador:

$$q0=Kp*(1+(Td/T0)) \quad (3)$$

$$q1=(-1)*Kp*(1+2*(Td/T0)-(T0/Ti)); \quad (4)$$

$$q2=Kp*(Td/T0) \quad (5)$$

As equações a seguir descrevem o armazenamento dos valores de erro usados no cálculo da rotina do PID:

$$erro[0]=erro[1] \quad (6)$$

$$erro[1]=erro[2] \quad (7)$$

$$erro[2]=sp-pv \quad (8)$$

As equações a seguir descrevem o cálculo da variável de manipulada (sinal de saída do controlador), e armazenamento do último valor calculado.

$$lastCo=newCo \quad (9)$$

$$newCo = lastCo + (q0 * error\_0) + (q1 * error\_1) + (q2 * error\_2) \quad (10)$$

O fluxograma da figura 46 mostra o fluxograma do programa implementado no microcontrolador PIC.

Ao iniciar a placa de desenvolvimento o *software* que está gravado no microcontrolador PIC entrará em execução. A primeira rotina a ser inicializada é a comunicação serial, pois está será a utilizada para a comunicação com Windows CE via um protocolo específico desenvolvido para essa aplicação. Feito isto, será inicializado o conversor analógico digital de 10 bits (interno ao PIC), que será utilizado para a leitura do valor da velocidade do motor para que seja feita a realimentação do controlador que está sendo executado no Windows CE.

O LCD (display de cristal líquido) será inicializado e terá a função de uma interface homem máquina e estará sempre mostrando os valores da saída do controlador (CO) e a variável de processo (PV). O PWM será inicializado e terá a função de atuar no modulo de potência que está acoplado diretamente no motor. Neste momento a placa de desenvolvimento

está pronta para receber os dados do aplicativo, ou seja, fica esperando o recebimento de dados via serial.

O Aplicativo no Windows CE calcula a velocidade inicial para o motor (nesse caso 0% por motivos de segurança), nesse momento a placa de desenvolvimento ajusta o PWM para 0%, aguarda um tempo predeterminado pelo programador para a estabilização do sinal, e depois faz a conversão do sinal de velocidade do motor (analógico para digital) para digital. A placa de desenvolvimento mostra no LCD as variáveis atuais (em porcentagem) e através do protocolo de comunicação serial retorna a variável de processo para o controlador que se encontra no micro com Windows CE e fica aguardando um novo comando do controlador (Windows CE).

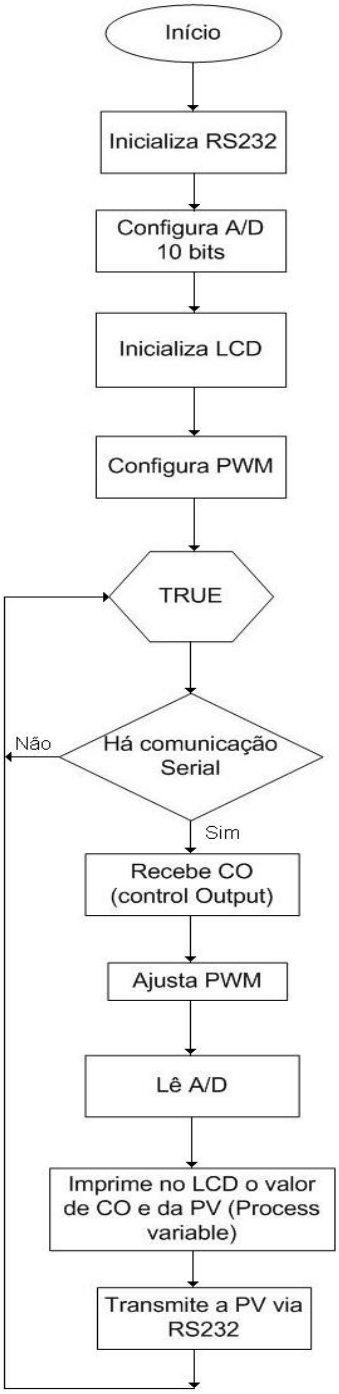


Figura 46 – Fluxograma do programa implementado na placa de desenvolvimento

Fonte: O Autor

O fluxograma detalhado do controlador PID está descrito nas figuras 47, 48, 49, 50. Este é um aplicativo desenvolvido no micro aplicativo onde se encontra a imagem do Windows CE.

O fluxograma da figura 47 traduz o diagrama de estados do programa PID. Depois do programa em questão configurar a comunicação serial e testar se a porta serial está funcionando pode-se executar os seguintes estados.

- Ajuste do controlador – opção na qual pode ser feito os ajustes dos parâmetros de sintonia do PID, ou seja, as ações do controlador e também o ajuste do set point (velocidade “setada” (desejada)).
- Controle automático – opção para se obter o controle em malha fechada automaticamente.
- Controle manual – opção do controlador onde o usuário atua diretamente na saída do controlador.
- Sair – fim do programa

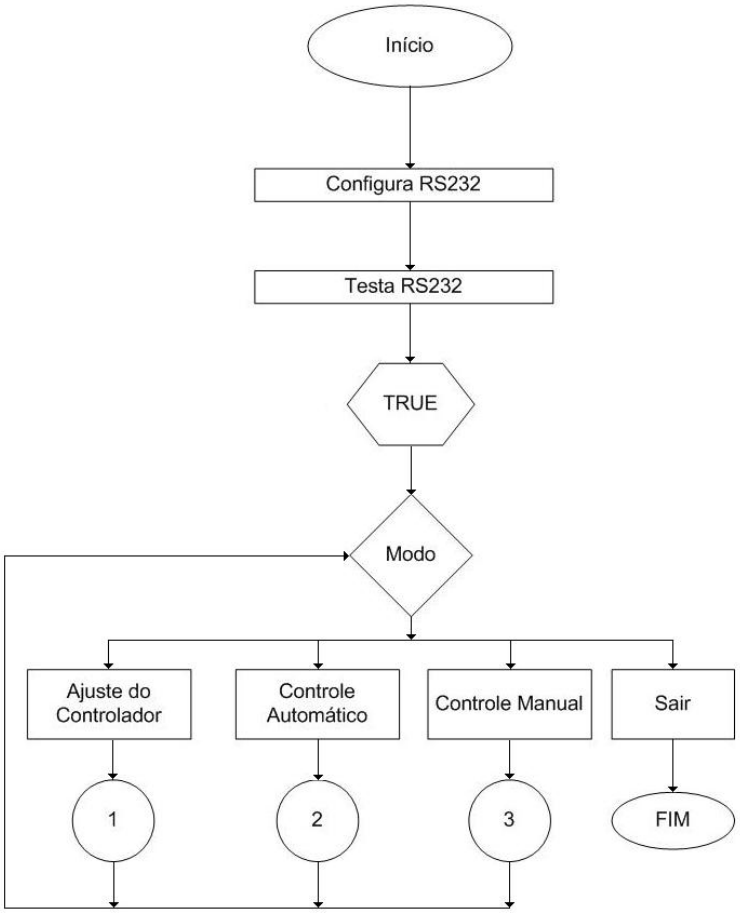


Figura 47 - Fluxograma Inicial
Fonte: O Autor

Na figura 48 é apresentado o fluxograma de ajustes dos parâmetros de sintonia do controlador PID.

Os parâmetros são:

- SP – variável desejada (set point) na velocidade do motor. 0% a 100%
- KP – ganho proporcional, 1 – 100, padrão (default): 1
- TI – tempo integral, 0 – 999, padrão: 999
- TD – tempo derivativo, 0 – 999, padrão: 0

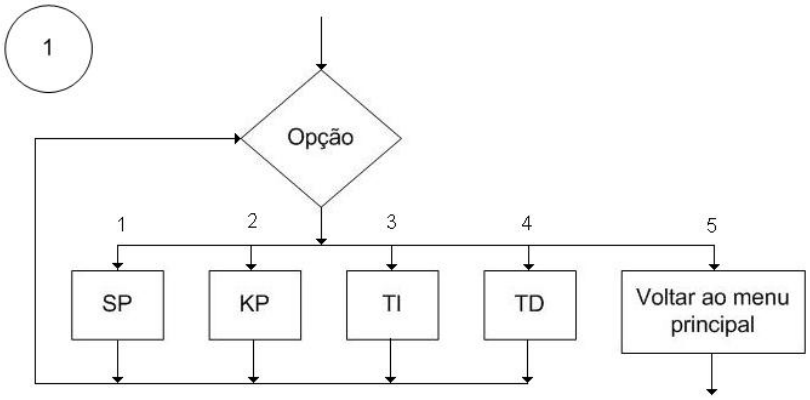


Figura 48 – Fluxograma do ajuste do controlador
Fonte: O Autor

A figura 49 mostra o fluxograma do controle automático. O controle automático é responsável pelo ajuste do CO. Quando essa opção é escolhida é enviado o valor do CO atual para a placa de desenvolvimento e o controlador no micro aplicativo fica aguardando o retorno da variável PV atual.

Enquanto nenhuma tecla for pressionada, o programa executado no micro aplicativo vai fazer o calculo do erro (valor desejado – valor da velocidade do motor (SP-PV)), depois com os valores das ações de controle é calculado um novo CO (COatual), feito isso é salvo o CO antigo, imprime o valor de CO na tela do computador aplicativo para que o usuário saiba o valor do controle, envia o CO atual para a placa de desenvolvimento e por fica aguardando a resposta.

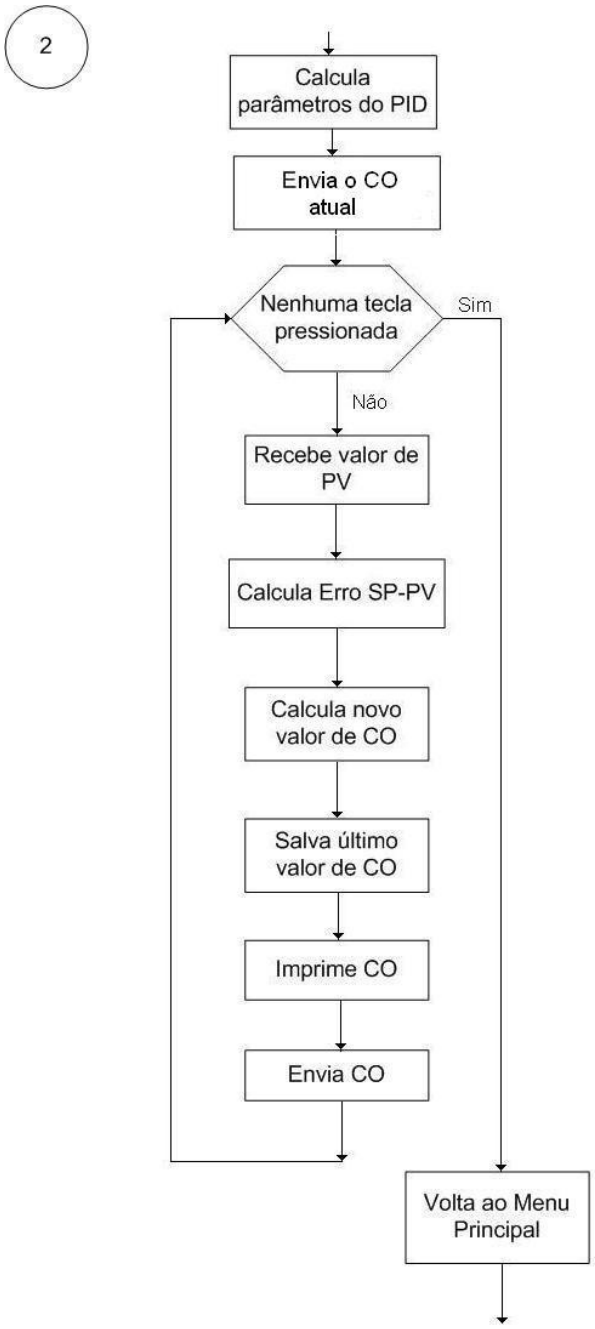


Figura 49 – Fluxograma do controle automático
Fonte: O Autor

A figura 50 mostra o fluxograma do controle manual onde o usuário assume o papel de controlador, interagindo diretamente no elemento final de controle, aumentando ou diminuindo o valor de CO.

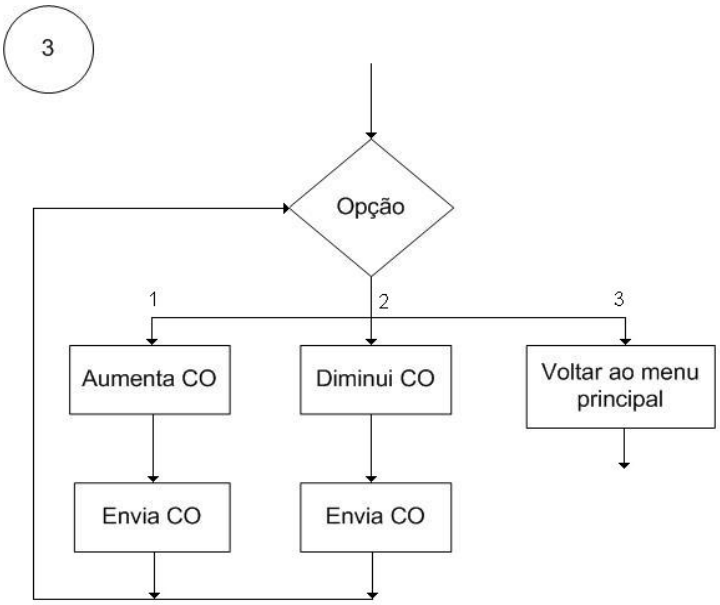


Figura 50 – Fluxograma do controle manual
Fonte: O Autor

A figura 51 apresenta a resposta em malha aberta do motor utilizado sem o controlador, onde é possível estimar uma taxa de amostragem adequada para operação do PID. Nas curvas das figuras 52 e 53 são apresentados os resultados obtidos quando a malha de controle é fechada usando-se o controlador. Estes resultados foram obtidos com uma taxa de amostragem de 110 ms. Por motivo de clareza foram representados em curvas separadas dois grupos de resultados. Na figura 52 são utilizados (Kp = 1, Ti = 1 e Td = 0), (Kp = 3, Ti = 1 e Td = 0) e (Kp = 5, Ti = 1 e Td = 0). Já na figura 53 são utilizados (Kp = 1, Ti = 1 e Td = 5), (Kp = 3, Ti = 5 e Td = 1). Os parâmetros de sintonia foram escolhidos de forma aleatória, visando ilustrar o funcionamento do controlador.

A curva é demonstrada na figura 51.

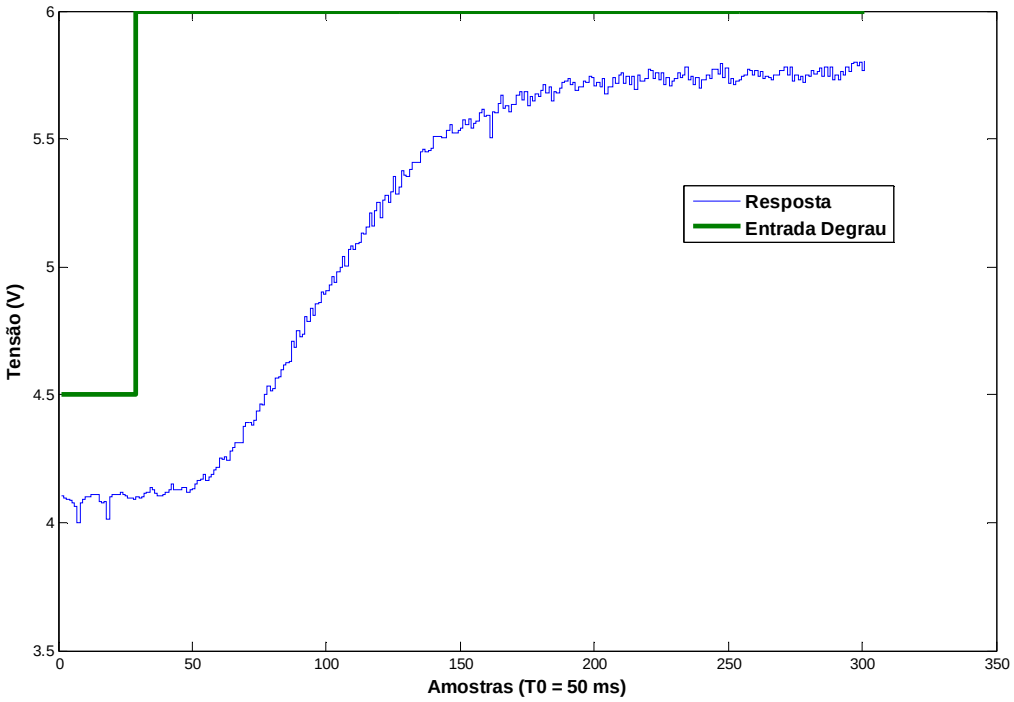


Figura 51 – Resposta ao degrau em malha aberta do motor utilizado.
Fonte: O Autor

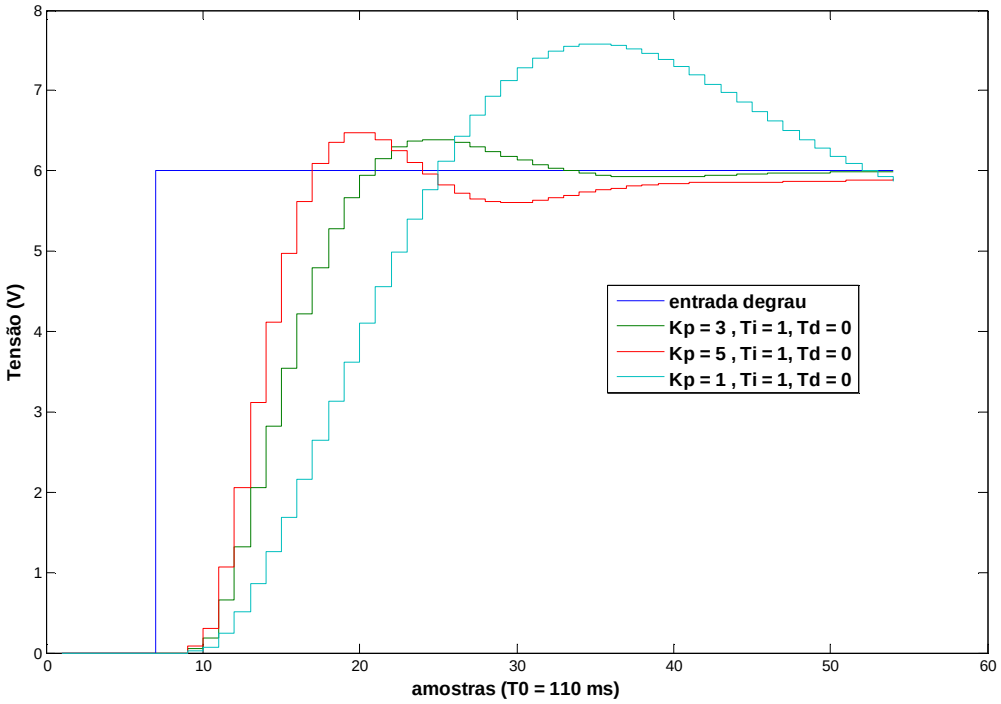


Figura 52 – Resposta ao degrau em malha fechada com o controlador PID.
Fonte: O Autor

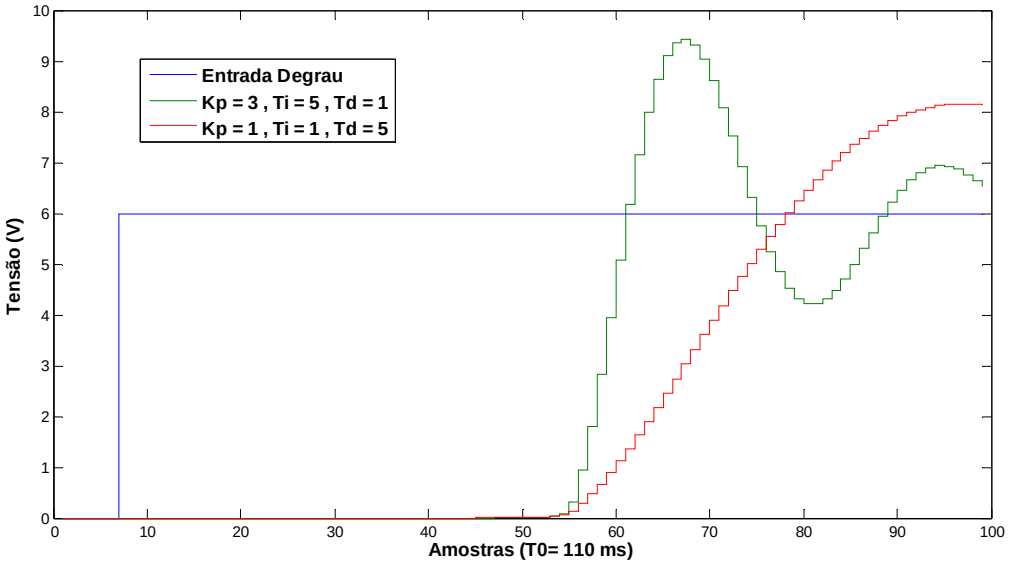


Figura 53 – Resposta ao degrau em malha fechada com o controlador PID.
Fonte: O Autor

4.2.2 Implementação de um CLP virtual

O objetivo dessa experiência é a implementação de um CLP virtual. A tomada de decisão será feita no bloco aplicativo, ou seja, no Windows CE.

Na engenharia acadêmica, normalmente, o termo controle implica em controle analógico ou digital de sistemas dinâmicos que possuem comandos contínuos. Exemplos destes tipos de sistemas incluem controle de vôo, controle de posicionamento em máquinas CNC, controle de temperatura, etc. Entretanto, muitas vezes máquinas industriais requerem controle nos quais as entradas e saídas são sinais on/off. Em outras palavras, os estados são modelados como variáveis que apresentam somente dois valores distintos. Embora os sistemas tenham dinâmica, esta é ignorada pelo controlador. O resultado é um desempenho mais limitado, no entanto, com um controle mais simples.

- Diagrama de blocos

Para exemplificar a proposta em questão, na área de automação, um controlador lógico programável virtua foi escolhido. Esse programa tambemfoi desenvolvido no Ambiente instalado Microsoft *Embedded* C++.

Esse programa faz o controle de entradas e saídas digitais, tal como um CLP. Para atuar como entradas e saídas físicas, é usada a mesma placa de desenvolvimento do controle PID de velocidade do motor CC.

O diagrama de blocos dessa experiência está demonstrado na figura 54.

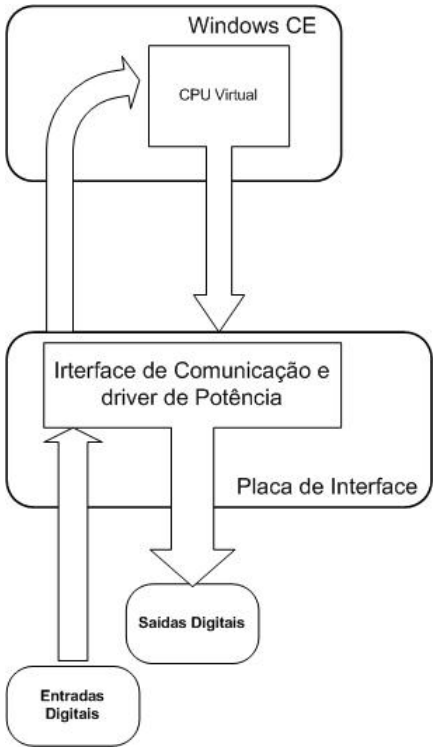


Figura 54 – Diagrama de blocos do CLP virtual

Fonte: O Autor

Funcionamento do programa

O programa do CLP virtual usa a mesma infra-estrutura do programa do controlador PID, ou seja, mesma placa de interface e o mesmo computador aplicativo.

A figura 55 mostra o fluxograma do programa executado no Windows CE que faz o papel de um CLP virtual.

A experiência dispõe de três entradas digitais e três saídas digitais.

O programa forma geral, executa uma, duas ou três expressões lógicas fornecidas pelo usuário. Cada expressão é vinculada a uma saída. As expressões só podem ser digitadas com algumas limitações impostas pelo programador, são elas:

1) A equação digitada pode ter até três variáveis.
2) As equações têm as operações lógicas OU e E.

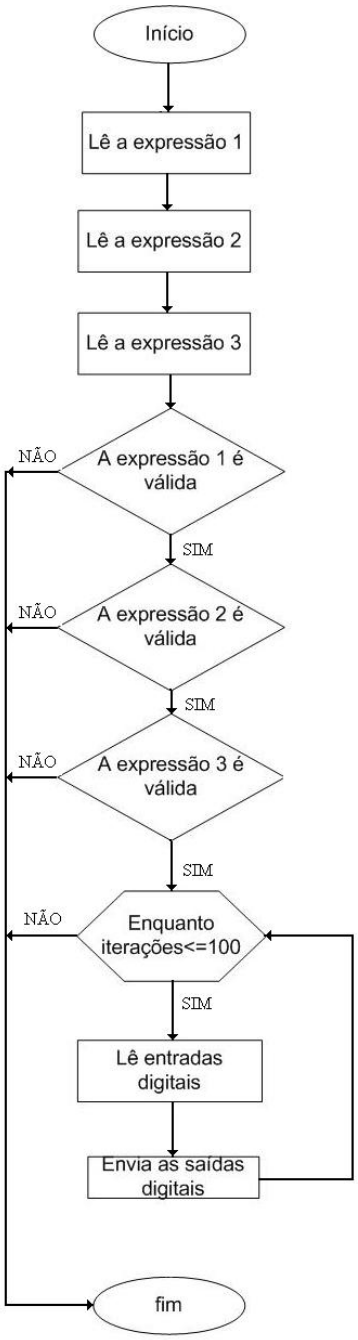


Figura 55 – Fluxograma do programa de CLP vrtual no Windows CE

Fonte: O Autor

Então o usuário, depois de digitado as equações, o programa irá receber o estado das entradas digitais que ficam na placa de interface. A partir desse momento o programa instalado no Windows CE fará as operações lógicas armazenadas e enviará os estados das saídas digitais.

O fluxograma da figura 56 mostra o fluxograma do programa implementado no microcontrolador PIC.

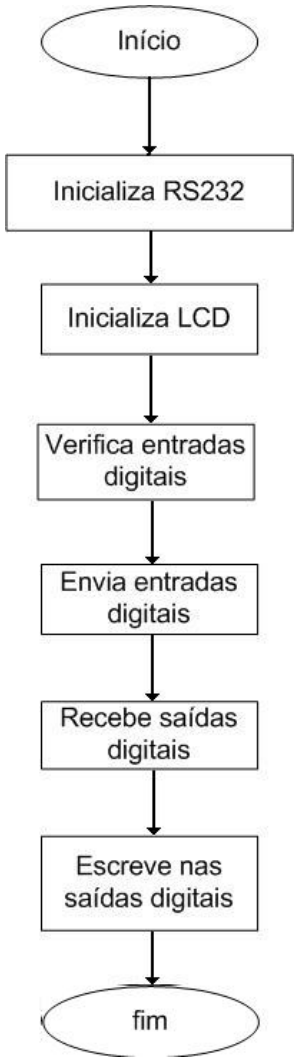


Figura 56 – Fluxograma do programa de CLP vrtual no microcontrolador PIC

Fonte: O Autor

A função do programa implementado na placa de interface é ler as entradas digitais e enviar para o Windows CE através do mesmo protocolo serial que é usado pelo programa de controle PID.

4.2.3 Exemplificando uma aula com Windows CE

O sistema operacional Windows CE pode ser usado em laboratórios de informática para preparar imagens com o sistema operacional e aplicativos que serão usados em sala de aula. Em um laboratório de controle e automação pode ser implementados nos computadores dos alunos, imagens que contenham aplicativos que simulem um CLP (Controlador Lógico Programável) virtual, por exemplo. Esse procedimento pode ser visto na figura 57. O download das imagens pode ser feito pelos servidores DHCP existentes no laboratório de informática.

Utilizando um micro com o *Plataform Builder* instalado o professor pode fazer o *download* das imagens, programar aplicativos com os alunos, usar aplicativos pré-programados e visualizar o andamento das atividades dos alunos, a partir de mensagens seriais.

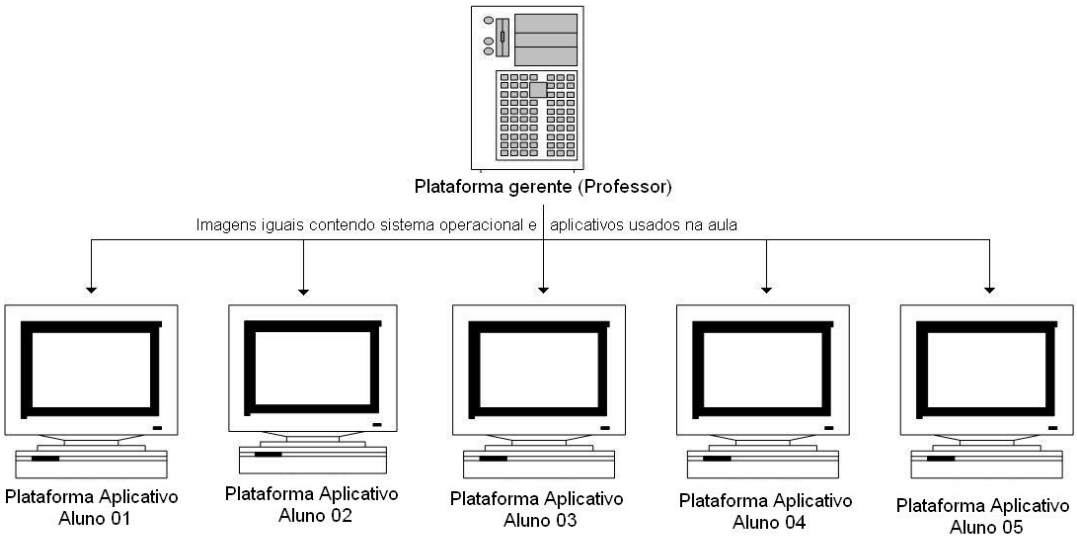


Figura 57 – Exemplo de uma aula com Windows CE
Fonte: O Autor

A experiência executada a partir da figura 57 foi realizada com a conexão micro a micro, ou seja, o *download* das imagens feitas para executar um exemplo de aula no Windows CE, foi executada um micro por vez.

CAPÍTULO 5

DISCUSSÃO E CONCLUSÕES

O Windows como sistema operacional embarcado é uma alternativa viável para o desenvolvimento de dispositivos dedicados, permitindo a seleção dos componentes do sistema operacional em uma vasta base de dados, garantindo total flexibilidade e um rápido desenvolvimento.

O Windows CE permite que você trabalhe com os mesmo ícones, barras de tarefas e botões do Windows 95. E, além disso, você ainda pode usar versões compactas de aplicações que já são familiares, como o Pocket Internet Explorer e o Pocket Excel.

O fabricante de equipamentos, ao utilizar as funcionalidades do Windows CE em seus dispositivos, estará garantindo ao "desenvolvedor" compatibilidade de aplicações com as APIs Win32 e com a tecnologia. NET. Com isso, o trabalho de readaptação das aplicações é mínimo ou nulo na maior parte dos casos.

Com este trabalho espera-se contribuir para o ensino e a difusão de sistemas operacionais embarcados, particularmente o Windows CE, bem como, a partir de um exemplo prático, demonstrar sua aplicabilidade no ambiente acadêmico. A utilização de ferramentas de desenvolvimento como o *Platform Builder* e de *device drivers* disponíveis simplificam o projeto de um novo aplicativo. As diversas ferramentas integradas no ambiente permitem a visualização de possíveis erros de projeto, antes mesmo da execução física do mesmo.

Alguns aspectos promissores do ambiente proposto são:

- Flexibilidade – Há uma grande variedade de configurações possíveis na implementação de soluções para diversos problemas de automação e controle.
- Ambiente aberto – Permite que seus blocos componentes sejam totalmente acessados e eventualmente modificados para agregar novas soluções.
- Facilidade de expansão – Novos recursos podem ser facilmente adicionados a um projeto, permitindo a adição de novas funcionalidades.
- Suporte para tempo real: é possível garantir que requisitos de tempo real serão atendidos, particularmente para aplicações na área de automação industrial.
- Domínio tecnológico – O projeto proposto apresenta-se como oportunidade de domínio de tecnologias associadas a sistemas microprocessados, sistemas operacionais, interfaces, lógica reconfigurável, entre outras.

Os dois experimentos apresentados demostraram a abrangência de uso do Windows CE em sala de aula e na indústria. Máquinas consideradas obsoletas se tornam soluções atrativas com diversas possibilidades de implementação de exemplos em controle e automação.

A reprodução deste experimento em vários computadores aplicativos, situação típica em sala de aula, é questão de simples *download* via rede. Por outro lado a utilização proposital de uma plataforma de baixo desempenho ("obsoleta" em termos de *hardware*: Pentium 200MHz, 64MB de memória RAM) para os experimentos em questão, aponta para uma sobre-vida de equipamentos similares nas instituições de ensino.

O trabalho apresentado mostrou uma limitação importante no objetivo proposto. Quanto à linguagem de programação usada na construção de aplicativos (C++), apesar das instruções usadas na programação embarcada serem similares a programação usada em Windows XP, existem instruções especificas e diferenças de sintaxe que podem provocar dificuldade de implementação. A solução adotada foi de recorrer à bibliografia especializada importada, pois a literatura portuguesa possui poucas opções ou são voltadas para programação especificas para PDA ou *Handheld*.

Entretanto com a contrução dos exemplos descritos nesse trabalho é evidente que a gama de possibilidades não se restrigem a controle e automação, pois tópicos como: Interfaceamento, eletrônica de potência, microcontroladores, linguagem de programação C++, conversores A/D e D/A, comunicação serial, fazem parte desse estudo.

A partir dos experimentos executados na área de automação e controle é possível afirmar que o sistema operacional Windows CE pode ser aplicado no ambiente didático e também industrial, pois cada vez mais se vê o aumento de PCs industriais no chão de fábrica.

Entre os possíveis trabalhos futuros destacam-se:

- Estudo sobre a implementação do sistema operacional Windows CE em um processador embarcado em FPGA, com possível desenvolvimento de um processador embarcado com suporte MMU.
- Estudo sobre o desenvolvimento de integração do Windows CE com protocolos de Redes Industriais como CAN, *Profibus* ou *Fieldbus Foundation*.
- Produzir e disponibilizar literatura em português sobre a linguagem de programação embedded C++, pois ainda existem poucos livros sobre o assunto no idioma português e considerando-se também outros tipos de bibliografia, como artigos por exemplo.

- Estudos sobre a viabilidade de se implementar módulos de controle de automação residencial, comercial e industrial.
- Aplicar o ambiente proposto em disciplinas dos cursos de Engenharia Elétrica e Engenharia da Computação da UTFPR.

Computadores com imagens do sistema operacional Windows CE podem ser ferramentas úteis dentro da sala de aula, podendo ajudar o professor a exxemplificar o cotidano tanto na indústria, como itens do dia-a-dia. Esses computadores podem ser desde estações que sejam acesso a internet até PCs indutriais que contenham softwares supervisórios.

REFERÊNCIAS

ALTERA. **Altera Corporation**, 2006. Disponível em: <http://www.altera.com>. Acesso em Acesso: 8 de Ago. 2007 (ultimo acesso).

ALASSAAD, Z. M.; SAGHIR, M. A. R. CERPID A Reconfigurable Plataform Interface Driver for Windows CE.NET. In: INTERNATIONAL CONFERENCE ON ENABLING TECHNOLOGIES FOR THE NEW KNOWLEDGE SOCIETY. 3., 2005, Beirut. **Anais Eletronicos.** USA: IEEE, 2005, p.839-850. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1609670> Acesso: 8 de Ago. 2007 (último acesso).

ANDREW, B. Windows CE brings real time control to the PC platform. **Journal Assembly Automation**, v. 19 n. 4 p. 306-308. 1999. Disponível em: <http://www.emeraldinsight.com/10.1108/01445159910295195> Acesso: 8 de Ago. 2007 (ultimo acesso).

BARABANOV, M.; YODAIKEN, V. Real-Time Linux. 1997**. CiteSeer.IST Scientific Literature Digital Library.** Disponivel em: <http://citeseer.ist.psu.edu/6239.html> New Mexico Institute of Technology, 1996.

BARR, M. **Programming Embedded Systems in C and C++.** Sebastopol: O' Reilly & Associates, 1999.

COMPTON, K.; HAUCK, S. Reconfigurable computing: a survey of systems and software. **ACM Computing Surveys**, New York, v. 34, n. 2, p. 171-210. jun. 2002. Disponivel em: <http://mesl.ucsd.edu/gupta/cse237c-s07/Readings/reconfigurablecomputingCompton.pdf> Acesso: 10 de ago. 2007 (ultimo acesso).

CORIC, S.; LEESER, M.; MILLER, E. Parallel- bean backprojection: An FPGA implementation optimized for medical imaging. **The Journal of VLSI Signal Processing**. Boston, 2002, p. 217-226.

CARRO, L.; WAGNER, FLAVIO. **Capítulo 2 das Jornadas de Atualização em Informática**. In: XXII JAI 2003. (Org.). Sistemas Computacionais Embarcados. Campinas: Sociedade Brasileira de Computação, 2003, v. 1, p. 45-94.

DEHUAI, Z.; CUNXI, X.; XUEMEI, L. **Design and Implementation of a Security and Patrol Robot System**. Proceedings of the IEEE International Conference on Mechatronics & Automation Niagara Falls, Canada, v. 4, p.1745 – 1749. 2005. Disponivel em: <http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/10831/34149/01626823.pdf> Acesso em: 10 de ago. 2007 (ultimo acesso).

DIDO, J. et al. A flexible floating-point format for optimizing data-paths and operators in FPGA-based dsps. **Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays.** Monterey, California, USA p. 50 – 55. 2002 Disponivel em: <http://portal.acm.org/citation.cfm?doid=503048.503056> Acesso: 10 de ago. 2007

FANG, H.; BAO,Y.; HE, Y. Study on Field Information Fast Collection System. **Instrumentation and Measurement Technology Conference**. IMTC 2005. Proceedings of the IEEE, vol. 2, p.1325 – 1329. 2005.

GALLAS, B; VERMA, V. Embedded Pentium Processor System Design for Windows CE. **WESCON/98.** Anaheim, CA, USA, p. 114-123.1998
Disponivel em :<http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?tp=&arnumber=716432> Acesso em: 10 de ago. 2007 (ultimo acesso).

GEORGE, M.R.; WONG,W.F. Windows CE for a Reconfigurable System-on-a-Chip Processor. INTERNATIONAL CONFERENCE ON FIELD-PROGRAMMABLE TECHNOLOGY, 2004. Brisbane, Australia. **Anais Eletronicos…** USA: IEEE p. 201-208 2004. Disponível em: <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1393269> Acesso: 8 de Ago. 2007 (último acesso).

GOGGIN, T.; DAVID, L. H.; JASON, M. M. **Windows CE Developer's Handbook.** SYBEX, INC, 1999.

GOKHALE, A.; C. SCHMIDT, D. Techniques for Optimizing CORBA Middleware for Distributed Embedded Systems. **Proceedings of INFOCOM '99**,New York, USA, Mar. 1999. Disponível em: <http://citeseer.ist.psu.edu/gokhale99techniques.html> Acesso em: 11 de ago. 2007 (último acesso).

GUPTA, N.; SRIVASTAV, V.; BHATIA, M. P. S. Middleware – An Effort towards Making Applications Platform Independent. **International Conference on Systems and Networks Communication (ICSNC'06).** Tahiti, French Polynesia, p. 62, 2006.

HARTENSTEIN, R. "A Decade of Reconfigurable Computing: a Visionary Retrospective". DATE'01 – **Design, Automation and Test in Europe**, Munich, Alemanha, p.642-649 Mar. 2001. Proceedings, IEEE Computer Society Press, 2001.

HERSHENSON, M. Considerations For Designing Chipsets for Windows CE Systems, **WESCON/98,** Anaheim, CA, USA, p. 124-127,1998
Disponivel em :< http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?tp=&arnumber=716433> Acesso em: 10 de ago. 2007 (ultimo acesso).

HPFACTOR. **A Brief History of Windows CE**, 2006. Disponível em: <http://www.hpcfactor.com/support/windowsce/> Acesso em: 10 de ago. 2007 (último acesso)

HUAN, L.; HANG, L.; XIA, Y. Access Control Technology Research in Embedded Operating System. **Second International Conference on Embedded Software and Systems**, P.R. China, p.1-7, 2005.
ILIC, S.; KATUPITIYA, J.; TORDON, M. In-vehicle data logging system for fatigue analysis of drive shaft, **Proceedings of the IEEE 2004 International Workshop on Robot Sensing**, Graz, Austria, pp 30-34, 2004.

ITO, S. A.; CARRO, L. A comparison of microcontrollers targeted to FPGA-based embedded applications. **In: Proceedings of IEEE 13th Symposium on Integrated Circuits and Systems Design**, p. 397-402, 2000.

ISKANDAR, J. I. **Normas da ABNT comentadas para trabalhos científicos**. Curitiba: Champagnat, 2000.

JOHANSSON, A.; SURI, N. Error Propagation Profiling of Operating Systems. **Proceedings of International Conference on Dependable Systems and Networks (DSN),** Yokohama, Japão, jun - jul, 2005.

JOHN, K. Control System Development for Test Handler. **In: IEEE International Conference on Semiconductor Electronics**, USA 2002 p. 415 – 418 2005 Disponivel em: <http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/8641/27381/01217855.pdf> Acesso em: 8 ago de 2007 (último acesso).

JUNJU, WANG; BIQUIN, LIN. Implementation of Bluetooth FTP under Windows CE International Conference on Signal Processing, China. **Anais Eletronicos…** USA: IEEE vol.2 p. 1697- 1700, 2005 Disponivel em: <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1180128> Acesso em: 8 ago de 2007 (último acesso).

KADIONIK, P. Linux Embarqué: Le Project uClinux. **Linux Magazine**, France, n. 36, p. 16-23, 2002.

KALRA, P. Reconfigurable FPGAs help build upgradeable systems**. Embedded Developers Journal**, 2001, p. 16-21.

KAWAKAMI, I.; NIMURA, Y.; HAMADA, K. Real-time Extension for Windows NT/CE used for Control Systems. **Proceedings of the 39th SICE Annual Conference.** p. 26–28, 2000. Disponivel em: < ieeexplore.ieee.org/iel5/7141/19236/00889702.pdf > Acesso em: 8 ago de 2007 (último acesso).

JUNJU, WANG; BIQUIN, LIN. Implementation of Bluetooth FTP under Windows CE International Conference on Signal Processing, China. **Anais Eletronicos…** USA: IEEE vol.2 p. 1697- 1700, 2005 Disponivel em: <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1180128>Acesso em: 8 ago de 2007 (último acesso).

K. S. P. CLARKE, D. KERSHAW. The System Design of a Windows CE ARM based Micro-controller. **Proceedings of the International Conference on Computer Design**, p.236, 2002. Disponivel em: <http://doi.ieeecomputersociety.org/10.1109/ICCD.1998.727056> Acesso em: 8 ago de 2007 (último acesso).

LAKATOS, E. M., MARCONI, M. de. **Fundamentos da metodologia do trabalho científico.** 3. ed. São Paulo: Atlas, 2000.

LEE, T.; HSIUNG, P. Embedded Software Synthesis and Prototypin**. IEEE Transactions on Consumer Electronics**, USA Vol. 50, No. 1, 2004. <http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/30/28566/01277888.pdf?arnumber=1277888> Acesso em: 8 ago de 2007 (último acesso).

LEHRBAUM, R. *Bully in the (embedded) playground*. **Linux Journal**, Disponível em: <www.linuxjournal.com/article.php?sid=5698>, 2002. Acesso em 8 de ago. 2007 (último acesso)

LI, N.; FANG, Y. Software/Hardware Partition in Multiple Processors Embedded System. **Proceedings of the Fourth International Conference on Machine Learning and Cybernetics**, Guangzhou, p. 18-21, 2005.

LIMA, C. R. E.; ROSARIO, J. M. Implementação de cadeira de rodas com dispositivos lógicos programáveis. **Primeiro Congresso Temático de Dinâmica e Controle da Sociedade Brasileira de Matemática Aplicada e Computacional** - DINCON 2002, São José do Rio Preto, SP, Brasil, v. 1, 2002, p. 27-33.

NETTER,C. M.; BACELLAR, F. L. Assessing the Real-Time Properties of Windows CE 3.0. **Fourth International Symposium on Object-Oriented Real-Time Distributed Computing** p. 179, 2001. < http://doi.ieeecomputersociety.org/10.1109/ISORC.2001.922835 > Acesso em: 10 de ago. 2007(último acesso)

MICROSOFT. **Microsoft corporation.** Lista de processadores suportados pelo Windows CE. 2007. Disponível em: <http://www.microsoft.com/windows/embedded/eval/wince/techspecs.mspx> Acesso em: 10 de ago. 2007(último acesso)

MICROSOFT. **Microsoft corporation**, *2006*. Real Time and Windows CE Disponível em: <http://msdn.microsoft.com/embedded/usewinemb/ce/techno/realtme/default.aspx?_r=1> Acesso em: 10 de ago. 2007 (último acesso)

MIYAZAKI, T. Reconfigurable systems: a survey**. In: Proceedings of the IEEE Design Automation Conference ASP-DAC 98**, p. 447-452, 1998.

OGATA, K. **Engenharia de Controle Moderno**. 4. ed. Brasil: Editora Pearson, 2003.

OLIVEIRA, R.; CARISSIMI, A; TOSCANI, S. **Sistemas Operacionais**. 2. ed. Brasil: Editora Sagra Luzzatto, 2001.

ORTIZ, S. JR. Embedded OSs Gain the Inside Track. **IEEE Computer**, n. 11, vol. 34, p. 14-16, Nov. 2001.

P. IBACH; N. M.; J. RICHLING;V. STANTCHEV; A. WIESNER; M. MALEK. CERO: CE Robots community. **IEE Proc.-Softw**, v. 152, n. 5, p. 210-214, Oct. 2005.

P. SHELTON, CHARLES; KOOPMAN, PHILIP; DEVALE, KOBEY. Robustness Testing of the Microsoft Win32 API.In: INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS AND NETWORKS**.** 2000, New York **Anais**... IEEE Computer Society DL, 2000. Disponivel em: <http://www.ece.cmu.edu/~koopman/ballista/dsn2000/index.html> Acesso em: 8 de ago. de 2007 (último acesso).

PALADINI, E. P. **Gestão da Qualidade: Teoria e Prática**. São Paulo: Editora Atlas, 2000.

PAULIN, P. G et al.. Embedded Software in Real-Time Signal Processing Systems: Application and Architecture Trends. **PROCEEDINGS OF THE IEEE**, vol. 85, n. 3, p. 436– 454, Mar. 1997.

RASCHE, Andreas; POLZE, Andreas. Configurable Services for Mobile User. In: INTERNATIONAL WORKSHOP ON OBJECT-ORIENTED REAL-TIME DEPENDABLE SYSTEMS, 7., 2002. San Diego. PROCEEDINGS… **PROCEEDINGS OF THE IEEE**, 2002. <http://eexplore.ieee.org/iel5/7841/21579/01000049.pdf?isnumber=&arnumber=1000049> Acesso em: 10 de ago. 2007 (último acesso).

RECONF **Reconfigurable Computing Definition**, 2006 Disponível em: <http://www.acm.uiuc.edu/sigarch/projects/reconf/report_1.html> Acesso em: 10 de ago. 2007 (último acesso).

RIEBEEK, H. Brasil Holds All-Electronic National Election. **IEEE Spectrum**, 2002. Disponivel em:<http://ieeexplore.ieee.org/iel5/6/22049/01045582.pdf> Acesso em: 8 de ago. 2007 (último acesso).

ROSA JUNIOR, LEOMAR SOARES DA. **Um Estudo sobre Sistemas Operacionais Embarcados de Tempo Real**. [S.L.:s.n.], 2004.

ROSÁRIO, J. M.; LIMA, C. R. E.; SILVA, N. C. da. A proposal of flexible architecture for mobile robotics**. In: Procedure of The 7th Mechatronics Forum International Conference and Mechatronics Education**, v. 2, 2000, p. 139-145.

SIEMENS. S7-200, **Programmable Controller System Manual**, [S.L.:s.n.], 2003.

SANTO, B. Embedded Battle Royale. **IEEE: Spectrum**, n. 12, vol. 38, 2001, p. 36-41.

SILVA, N. C. da; LIMA, C. R. E.; ROSÁRIO, J. M. A computational solution to simplify prostheses control. **In: 22nd Iberian Latin-American Congress on Computational Methods in Engineering**, São Paulo: Campinas, 2001.

SIMÕES, E. V. Development of an embedded evolutionary controller to enable collision-free of a population of autonomous móbile robots, **thesis submitted to the University of Kent at Canterbury in the subject of Electronic Engineering for the degree of Doctor of Philosophy**, 2000.

SIMÕES, J. B., et al.. A Windows CE Stand-Alone Digital Spectrometer. In: **Nuclear Science Symposium,** v.1, p.255-259, 1999. Disponivel em: <http://>

STAR. **Star Bridge Systems**, 2006. Disponível em: <http://www.starbridgesystems.com>. Acesso em: 10 de ago. 2007 (último acesso)

TAURION, C. **Software Embarcado**. Brasport, 2005.

TACKE, C.; RICCI, L. Benchmarking Real-Time Determinism in Windows CE. **White Paper** Disponível em: <http://msdn2.microsoft.com/en-us/library/ms836535.aspx>, 2002

TANENBAUM, A. S., **Sistemas Operacionais Modernos**. 2. ed.. cidade: ediçãopearson-prentice-hall do brasil, 2003

VARGAS, F. L.; FAGUNDES, R. D. R.; BARROS, J. D. A FPGA-Based Viterbi Algorithm Implementation for Speech Recognition Systems In: **International acoustics, speech, and signal processing**, v.2 p.1217-1220, 2001.

XIAO, T.; JIN, X. Implement of Bluetooth Protocol Stack on Windows CE. Institute of Modern Comunication**, Proceedings of IEEE TENCON´02**, p.1249- 1252, 2002.

XILINX. *Xilinx Technology*, Inc, 2007. Disponível em: <http://www.xilinx.com>. Acesso em: 10 de ago. 2007 (último acesso)

APÊNDICE 1

PROGRAMA DESENVOLVIDO NO WINDOWS CE PARA O CONTROLE PID

```
#include "stdafx.h"
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <keybd.h>
#include <winuser.h>
#include <windows.h>
float lastCo;
int A,B,C; //váriaveis que representam a entradas digitais
char buffer_rec_clp[2];
int Retorna_saida(char exp_log[5]);//função que valida a expressão lógica e retorna estado
lógico da saída do clp virtual
void Menu_Inicial()//menu inicial do programa de controle e automação
{
        printf("\n\nDigite '1' para Controlador PID");
        printf("\nDigite '2' para Controlador Logico Programavel (CLP)");
        printf("\nDigite '3' para sair");
        printf("\nOpcao:");
}
void Menu_manual()//menu para controle manual de velocidade do motor cc
{
        int opcao=3;
        printf("\n\nControle em Modo Manual....");
        printf("\n\nDigite '1' para aumentar o Co (Control Output)");
        printf("\nDigite '2' para diminuir o Co (Control Output)");
        printf("\nDigite '3' para voltar ao menu principal\n\n");
}
void Menu_ajuste()// menu de ajuste dos parametros do controlador PID
{
        int opcao=5;
        printf("\n\nAjuste dos paramentros do controlador PID");
        printf("\n\nDigite '1' para entrar com o valor do SP (0~100%)");
        printf("\nDigite '2' para entrar com o valor do kp (default 1)");
        printf("\nDigite '3' para entrar com o valor do Ti (default 999)");
        printf("\nDigite '4' para entrar com o valor do Td (default 0)");
        printf("\nDigite '5' para voltar ao menu principal");
        printf("\n\nOpcao:");
}
int Menu_principal()// menu para a escolha do tipo de controle
{
        int opcao;
        printf("\n\nDigite '1' para Ajuste das Acoes de Controle");
        printf("\nDigite '2' para modo de controle Automatico");
```

```
        printf("\nDigite '3' para modo de controle Manual");
        printf("\nDigite '4' para voltar a Menu Inicial\n");
        printf("\nOpcao:");
        scanf("%d",&opcao);
        return(opcao);
}
void Verifica_dados_entrada(char dado) //essa função verifica o estado lógico das entradas do CLP virtual
{
        switch(dado)
        {
                case('P'):
                {
                        A=0;
                        B=0;
                        C=0;
                }
                break;
                case('Q'):
                {
                        A=0;
                        B=0;
                        C=1;
                }
                break;
                case('R'):
                {
                        A=0;
                        B=1;
                        C=0;
                }
                break;
                case('S'):
                {
                        A=0;
                        B=1;
                        C=1;
                }
                break;
                case('T'):
                {
                        A=1;
                        B=0;
                        C=0;
                }
                break;
                case('U'):
                {
                        A=1;
                        B=0;
```

```
                C=1;
            }
            break;
            case('V'):
            {
                A=1;
                B=1;
                C=0;
            }
            break;
            case('X'):
            {
                A=1;
                B=1;
                C=1;
            }
            break;
        }

} //função principal
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPTSTR
lpCmdLine,int nCmdShow)
{ // Declaração das variáveis
        HANDLE hPort1, hPort2, hPort3, hPort4;
        BOOL result;
        DWORD dwBytesTransferred;
        int opcao,i,modo,opcao_manual,contador,menu_inicial,conti;
        int Saida_1, Saida_2, Saida_3;
        float newCo,Kp,Ti,Td,T0,PV;
        float SP;
        float q0;       // PID parametro
        float q1;       // PID parametro
        float q2;       // PID parametro
        float error_0;    // Erro atual
        float error_1;    // Erro final
        float error_2;    // Error intermediario
        char buffer[5];
        char buffer_rec[6];
        char msg1[1];
        char exp_log_s1[5], exp_log_s2[5],exp_log_s3[5];
        DWORD written;
        //Abre porta Serial
        hPort1 = CreateFile(_T("COM1:"), GENERIC_READ | GENERIC_WRITE, 0,
NULL, OPEN_EXISTING, 0, 0);
        hPort2 = CreateFile(_T("COM2:"), GENERIC_READ | GENERIC_WRITE, 0,
NULL, OPEN_EXISTING, 0, 0);
        printf("COM1 port handle = %x\n", hPort1);
        printf("COM2 port handle = %x\n", hPort2);
        DCB dcb;
        GetCommState(hPort1, &dcb);
```

```
dcb.BaudRate = 9600;
dcb.ByteSize = 8;
dcb.fParity = NOPARITY;
dcb.StopBits = ONESTOPBIT;
dcb.fRtsControl = RTS_CONTROL_DISABLE;
BOOL result_DCB = SetCommState(hPort1, &dcb);
result_DCB = SetCommState(hPort2, &dcb);
result = WriteFile ( hPort1, msg1, sizeof(msg1), &written, NULL);
printf("Write Result = %s, Bytes Written:%d\n", result?"OK":"NOK", written);
exp_log_s1[0]='t';
exp_log_s1[1]='t';
exp_log_s1[2]='t';
exp_log_s1[3]='t';
SP=10;
PV=0;
Kp=1;
Ti=999;
Td=0;
T0=1;
newCo=0;
modo = 4;
menu_inicial = 3;
error_0=0;    // Actual Error
error_1=0;    // Last Error
error_2=0;
conti=0;
while(true)
{
        //modo de controle
        switch(menu_inicial)
        {
                case(1):
                {
                        switch(modo)
                        {
                                case(1): //Ajuste do controlador
                                {
                                        switch(opcao)
                                        {
                                                case(1): //valor do SP
                                                        printf("\nSP:");
                                                        printf(" %f\n",SP);
                                                        scanf("%f",&SP);
                                                        printf(" %f\n",SP);
                                                        Menu_ajuste();
                                                        scanf("%d",&opcao);
                                                break;
                                                case(2):
                                                        printf("\nKp:");
                                                        scanf("%f",&Kp);
```

```
                                        printf("\n%f",Kp);
                                        Menu_ajuste();
                                        scanf("%d",&opcao);
                                break;
                                case(3):
                                        printf("\nTi:");
                                        scanf("%f",&Ti);
                                        Menu_ajuste();
                                        scanf("%d",&opcao);
                                break;
                                case(4):
                                        printf("\nTd:");
                                        scanf("%f",&Td);
                                        Menu_ajuste();
                                        scanf("%f",&opcao);
                                break;
                                case(5):
                                        modo=4;
                                break;
                        }
                }
                break;
                case(2): //modo automatico
                {
                        for(i=0;i<6;i++)
                        buffer_rec[i]='\0';
                        //Calcula os paramtros do PID
                        q0 =  Kp * ( 1+ (Td/T0) );
                        q1 = (-1) * Kp * ( 1 + (2*(Td/T0)) - (T0/ (Ti *
60)) );
                        q2 =  Kp * (Td/T0);
                        //Envia dado especifico para que o pic envie o
valor da PV
                        printf("SP=%f\n",SP);
                        WriteFile(hPort1,"B", sizeof(msg1), &written,
NULL);
                        Sleep(200);
                        contador = 0;
                        while(contador < 50)
                        {
                                contador++;
                                printf(" ");
                                ReadFile
(hPort1,&buffer_rec,5,&dwBytesTransferred,NULL);
                                        i=0;
                                while(buffer_rec[i]!='A')
                                        i++;
                                buffer_rec[i]='\0';
                                PV=(float)atoi(buffer_rec);
                                printf("Contador:%d\n",contador);
```

```
                                        printf("\nPV=%.2f",PV);

                                        //Calcular o erro
                                        error_0 = SP - PV;
                                        //Calculo do controle no instante u(k)
                                        newCo = lastCo + (q0 * error_0);
                                        //Salva os novos erros deslocados de um
                                        error_2 = error_1;
                                        error_1 = error_0;

                                        //Limitador do valor final de 0% à 100%
                                        if (newCo >= 95.0)
                                        {
                                                newCo = 951;
                                                error_1 = 0;
                                        }
                                        if (newCo < 0.0)
                                        {
                                                newCo = 0;
                                        }
                                        //Salva novo ultimo Co
                                        lastCo = newCo;
                                        //Envia o Co para o PIC
                                        _itoa((int)newCo, buffer, 10 );
                                        strcat(buffer,"A");
                                        i=0;
                                        while(buffer[i]!='A')
                                        {
                                                msg1[0]=buffer[i];
                                                printf(" ");
                                                WriteFile(hPort1,msg1,
sizeof(msg1), &written, NULL);

                                                i++;
                                        }
                                        printf("\nEnviando Co %f\n",newCo);
                                        printf(" ");
                                        WriteFile(hPort1,"A", sizeof(msg1),
&written, NULL);

                                        Sleep(200);

                                }
                                modo=4;
                                getchar();
                        }
                        break;
                        case(3): //modo manual
                        {
                                while(true)
                                {
                                        printf("Co=%f%% | Opcao:",newCo);
```

```
                                        scanf("%d",&opcao_manual);
                                        if(opcao_manual==1)//aumenta o Co
                                        {
                                                if(newCo>=95)
                                                        newCo=95;
                                                else
                                                        newCo=newCo+5;
                                                //Envia Co para o micro
controlador
                                                _itoa((int)newCo, buffer, 10 );
                                                strcat(buffer,"A");
                                                i=0;
                                                while(buffer[i]!='A')
                                                {
                                                        msg1[0]=buffer[i];
                                                        printf(" ");
                                                        WriteFile(hPort1,msg1,
sizeof(msg1), &written, NULL);
                                                        i++;
                                                }
                                                printf(" ");
                                                WriteFile(hPort1,"A",
sizeof(msg1), &written, NULL);
                                                Sleep(200);
                                                //Recebe PV
                                                printf(" ");
                                                ReadFile
(hPort1,&buffer_rec,5,&dwBytesTransferred,NULL);
                                                i=0;
                                                while(buffer_rec[i]!='A')
                                                        i++;
                                                buffer_rec[i]='\0';
                                                PV=(float)atoi(buffer_rec);
                                        printf("\nPV=%.2f",PV);
                                        }
                                        if(opcao_manual==2) //diminui o Co
                                        {
                                                if(newCo<=0)
                                                        newCo=0;
                                                else
                                                        newCo=newCo-5;
                                        //Envia Co para o micro controlador
                                                _itoa((int)newCo, buffer, 10 );
                                                strcat(buffer,"A");
                                                i=0;
                                                while(buffer[i]!='A')
                                                {
                                                        msg1[0]=buffer[i];
                                                        printf(" ");
```

```
                                        WriteFile(hPort1,msg1,
sizeof(msg1), &written, NULL);
                                        i++;
                                }
                                printf(" ");
                                WriteFile(hPort1,"A",
sizeof(msg1), &written, NULL);
                                Sleep(200);
                                //Recebe PV
                                printf(" ");
                                ReadFile
(hPort1,&buffer_rec,5,&dwBytesTransferred,NULL);
                                i=0;
                                while(buffer_rec[i]!='A')
                                        i++;
                                buffer_rec[i]='\0';
                                PV=(float)atoi(buffer_rec);
                                printf("\nPV=%.2f",PV);
                        }
                        if(opcao_manual==3)
                        {
                                modo=4;
                                break;
                        }
                }
        }
        break;
        case(4)://modo menu
        {
                modo=Menu_principal();
                if(modo==1)
                {
                        Menu_ajuste();
                        scanf("%d",&opcao);
                }
                if(modo==2)//modo automatico
                {
                        modo=2;
                }
                if(modo==3)//modo manual
                {
                        Menu_manual();
                }
                if(modo==4)
                        menu_inicial=3;
        }
        break;
    }
}
break;
```

```
                    case(2): //Controlador Lógico Programável Virtual
                    {
                            ini_um:
                            printf("\nDigite a expressao logica da Saida 1, (digite 'f' para
voltar):");
                            scanf("%s",&exp_log_s1); //aguarda a expressão lógica a ser
digitada
                            if(exp_log_s1[0]!='f') //caso digitado f o programa volta para o
menu inicial
                            {
                                    if(exp_log_s1[0]=='p') //caso digitado p vai para próxima
expressão
                                    {
                                            Saida_1=0;
                                            goto ini_dois;
                                    }
                                    Saida_1 = Retorna_saida(exp_log_s1); //valida a
expressão digitada
                                    if(Saida_1==10)
                                    {
                                            printf("\nFuncao Invalida...");
                                            goto ini_um;
                                    }
                                    ini_dois:
                                    printf("\nDigite a expressao logica da Saida 2, (digite 'f'
para voltar):");
                                    scanf("%s",&exp_log_s2); //aguarda a expressão lógica a
ser digitada
                                    if(exp_log_s2[0]!='f') //caso digitado f o programa volta
para o menu inicial
                                    {
                                            if(exp_log_s2[0]=='p') //caso digitado p vai para
próxima expressão
                                            {
                                                    Saida_2=0;
                                                    goto ini_trez;
                                            }
                                            Saida_2 = Retorna_saida(exp_log_s2);
                                            if(Saida_2==10)
                                            {
                                                    printf("\nFuncao Invalida...");
                                                    goto ini_dois;
                                            }
                                            ini_trez:
                                            printf("\nDigite a expressao logica da Saida 3,
(digite 'f' para voltar):");
                                            scanf("%s",&exp_log_s3); //aguarda a expressão
lógica a ser digitada
```

```
                                        if(exp_log_s3[0]=='f' || exp_log_s3[0]=='p')
//caso digitado f o programa volta para o menu inicial ou caso digitado p vai para próxima expressão

                                        {
                                                menu_inicial=3;
                                        }
                                        else
                                        {
                                                Saida_3 = Retorna_saida(exp_log_s3);
                                                if(Saida_3==10)
                                                {
                                                        printf("\nFuncao Invalida...");
                                                        goto ini_trez;
                                                }
                                                conti=0;
                                                while(conti<=100)
                                                {
                                                //Escreve na serial
                                                WriteFile(hPort1,"C", sizeof(msg1),
                                        &written, NULL);
                                                        //Espera a resposta do pic
                                                        ReadFile
(hPort1,&buffer_rec_clp,2,&dwBytesTransferred,NULL);
                                                        //Verifica qual lógica executar
                                                        if(exp_log_s1[0]=='p')
                                                                Saida_1=0;
                                                        else
                                                                Saida_1 =
Retorna_saida(exp_log_s1);
                                                        if(exp_log_s2[0]=='p')
                                                                Saida_2=0;
                                                        else
                                                                Saida_2 =
Retorna_saida(exp_log_s2);
                                                        if(exp_log_s3[0]=='p')
                                                                Saida_3=0;
                                                        else
                                                                Saida_3 =
Retorna_saida(exp_log_s3);
                                                        conti++;
                                                        if((Saida_1==0) && (Saida_2==0)
&& (Saida_3==0)) // depedendo dos estados das saídas envia um comando para o microcontrolador ligar ou desligar as saídas digitais
                                                                WriteFile(hPort1,"D",
sizeof(msg1), &written, NULL);
                                                        if((Saida_1==0) && (Saida_2==0)
&& (Saida_3==1))
                                                                WriteFile(hPort1,"E",
sizeof(msg1), &written, NULL);
```

```
                                        if((Saida_1==0) && (Saida_2==1)
&& (Saida_3==0))
                                                WriteFile(hPort1,"F",
sizeof(msg1), &written, NULL);
                                        if((Saida_1==0) && (Saida_2==1)
&& (Saida_3==1))
                                                WriteFile(hPort1,"G",
sizeof(msg1), &written, NULL);
                                        if((Saida_1==1) && (Saida_2==0)
&& (Saida_3==0))
                                                WriteFile(hPort1,"H",
sizeof(msg1), &written, NULL);
                                        if((Saida_1==1) && (Saida_2==0)
&& (Saida_3==1))
                                                WriteFile(hPort1,"I",
sizeof(msg1), &written, NULL);
                                        if((Saida_1==1) && (Saida_2==1)
&& (Saida_3==0))
                                                WriteFile(hPort1,"J",
sizeof(msg1), &written, NULL);
                                        if((Saida_1==1) && (Saida_2==1)
&& (Saida_3==1))
                                                WriteFile(hPort1,"K",
sizeof(msg1), &written, NULL);
                                        Sleep(10);
                                                }
                                        }
                                }
                                else
                                        menu_inicial=3;
                        }
                        else
                                menu_inicial=3;

                        }
                break;
                case(3):
                {
                        Menu_Inicial();
                        scanf("%d",&menu_inicial);
                        if(menu_inicial==1)
                        {
                                printf("\n\n --Controlador PID--");
                        }
                        if(menu_inicial==2)
                        {
                                printf("\n\n --Controlador Logico Programavel--");

                        }
                        if(menu_inicial==3)
```

```
                                {
                                        return(0);
                                }
                        }
                        break;
                }
        }
}
int Retorna_saida(char exp_log[5])
{

        int Saida0;
        Saida0=10;
        if(exp_log[0]=='\0')
                Saida0=0;
        //UMA VARIAVEL
        if(strcmp(exp_log,"a")==0)
        {
                Verifica_dados_entrada(buffer_rec_clp[0]);
                Saida0 = (A);
        }
        if(strcmp(exp_log,"b")==0)
        {
                Verifica_dados_entrada(buffer_rec_clp[0]);
                Saida0 = (B);
        }
        if(strcmp(exp_log,"c")==0)
        {
                Verifica_dados_entrada(buffer_rec_clp[0]);
                Saida0 = (C);
        }
        //DUAS VARIAVEIS
        if(strcmp(exp_log,"a.a")==0)
        {
                Verifica_dados_entrada(buffer_rec_clp[0]);
                Saida0 = (A&A);
        }
        if(strcmp(exp_log,"a.b")==0)
        {
                Verifica_dados_entrada(buffer_rec_clp[0]);
                Saida0 = (A&B);
        }
        if(strcmp(exp_log,"a.c")==0)
        {
                Verifica_dados_entrada(buffer_rec_clp[0]);
                Saida0 = (A&C);
        }
        if(strcmp(exp_log,"b.a")==0)
        {
                Verifica_dados_entrada(buffer_rec_clp[0]);
```

```
        Saida0 = (B&A);
}
if(strcmp(exp_log,"b.b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B&B);
}
if(strcmp(exp_log,"b.c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B&C);
}
if(strcmp(exp_log,"c.a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&A);
}
if(strcmp(exp_log,"c.b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&B);
}
if(strcmp(exp_log,"c.c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&C);
}
if(strcmp(exp_log,"a+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|A);
}
if(strcmp(exp_log,"a+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|B);
}
if(strcmp(exp_log,"a+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|C);
}
if(strcmp(exp_log,"b+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B|A);
}
if(strcmp(exp_log,"b+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
```

```
        Saida0 = (B|B);
}
if(strcmp(exp_log,"b+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B|C);
}
if(strcmp(exp_log,"c+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|A);
}
if(strcmp(exp_log,"c+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|B);
}
if(strcmp(exp_log,"c+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|C);
}
//3 VARIAVEIS
if(strcmp(exp_log,"a.a.a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A&A&A);
}
if(strcmp(exp_log,"a.a.b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A&A&B);
}
if(strcmp(exp_log,"a.a.c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A&A&C);
}
if(strcmp(exp_log,"a.b.a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A&B&A);
}
if(strcmp(exp_log,"a.b.b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A&B&B);
}
if(strcmp(exp_log,"a.b.c")==0)
{
```

```
		Verifica_dados_entrada(buffer_rec_clp[0]);
		Saida0 = (A&B&C);
}
if(strcmp(exp_log,"a.c.a")==0)
{
		Verifica_dados_entrada(buffer_rec_clp[0]);
		Saida0 = (A&C&A);
}
if(strcmp(exp_log,"a.c.b")==0)
{
		Verifica_dados_entrada(buffer_rec_clp[0]);
		Saida0 = (A&C&B);
}
if(strcmp(exp_log,"a.c.c")==0)
{
		Verifica_dados_entrada(buffer_rec_clp[0]);
		Saida0 = (A&C&C);
}
if(strcmp(exp_log,"b.a.a")==0)
{
		Verifica_dados_entrada(buffer_rec_clp[0]);
		Saida0 = (B&A&A);
}
if(strcmp(exp_log,"b.a.b")==0)
{
		Verifica_dados_entrada(buffer_rec_clp[0]);
		Saida0 = (B&A&B);
}
if(strcmp(exp_log,"b.a.c")==0)
{
		Verifica_dados_entrada(buffer_rec_clp[0]);
		Saida0 = (B&A&C);
}
if(strcmp(exp_log,"b.b.a")==0)
{
		Verifica_dados_entrada(buffer_rec_clp[0]);
		Saida0 = (B&B&A);
}
if(strcmp(exp_log,"b.b.b")==0)
{
		Verifica_dados_entrada(buffer_rec_clp[0]);
		Saida0 = (B&B&B);
}
if(strcmp(exp_log,"b.b.c")==0)
{
		Verifica_dados_entrada(buffer_rec_clp[0]);
		Saida0 = (B&B&C);
}
if(strcmp(exp_log,"b.c.a")==0)
{
```

```
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B&C&A);
}
if(strcmp(exp_log,"b.c.b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B&C&B);
}
if(strcmp(exp_log,"b.c.c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B&C&C);
}
if(strcmp(exp_log,"c.a.a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&A&A);
}
if(strcmp(exp_log,"c.a.b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&A&B);
}
if(strcmp(exp_log,"c.a.c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&A&C);
}
if(strcmp(exp_log,"c.b.a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&B&A);
}
if(strcmp(exp_log,"c.b.b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&B&B);
}
if(strcmp(exp_log,"c.b.c")==0
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&B&C);
}
if(strcmp(exp_log,"c.c.a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&C&A);
}
if(strcmp(exp_log,"c.c.b")==0)
{
```

```
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&C&B);
}
if(strcmp(exp_log,"c.c.c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&C&C);
}
if(strcmp(exp_log,"a+a+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|A|A);
}
if(strcmp(exp_log,"a+a+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|A|B);
}
if(strcmp(exp_log,"a+a+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|A|C);
}
if(strcmp(exp_log,"a+b+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|B|A);
}
if(strcmp(exp_log,"a+b+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|B|B);
}
if(strcmp(exp_log,"a+b+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|B|C);
}
if(strcmp(exp_log,"a+c+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|C|A);
}
if(strcmp(exp_log,"a+c+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|C|B);
}
if(strcmp(exp_log,"a+c+c")==0)
{
```

```
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|C|C);
}
if(strcmp(exp_log,"b+a+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B|A|A);
}
if(strcmp(exp_log,"b+a+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B|A|B);
}
if(strcmp(exp_log,"b+a+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B|A|C);
}
if(strcmp(exp_log,"b+b+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B|B|A);
}
if(strcmp(exp_log,"b+b+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B|B|B);
}
if(strcmp(exp_log,"b+b+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B|B|C);
}
if(strcmp(exp_log,"b+c+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B|C|A);
}
if(strcmp(exp_log,"b+c+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B|C|B);
}
if(strcmp(exp_log,"b+c+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B|C|C);
}
if(strcmp(exp_log,"c+a+a")==0)
{
```

```
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|A|A);
}
if(strcmp(exp_log,"c+a+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|A|B);
}
if(strcmp(exp_log,"c+a+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|A|C);
}
if(strcmp(exp_log,"c+b+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|B|A);
}
if(strcmp(exp_log,"c+b+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|B|B);
}
if(strcmp(exp_log,"c+b+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|B|C);
}
if(strcmp(exp_log,"c+c+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|C|A);
}
if(strcmp(exp_log,"c+c+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|C|B);
}
if(strcmp(exp_log,"c+c+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|C|C);
}
if(strcmp(exp_log,"a+a.a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|A&A);
}
if(strcmp(exp_log,"a+a.b")==0)
{
```

```
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|A&B);
}
if(strcmp(exp_log,"a+a.c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|A&C);
}
if(strcmp(exp_log,"a+b.a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|B&A);
}
if(strcmp(exp_log,"a+b.b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|B&B);
}
if(strcmp(exp_log,"a+b.c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|B&C);
}
if(strcmp(exp_log,"a+c.a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|C&A);
}
if(strcmp(exp_log,"a+c.b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|C&B);
}
if(strcmp(exp_log,"a+c.c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A|C&C);
}
if(strcmp(exp_log,"b+a.a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B|A&A);
}
if(strcmp(exp_log,"b+a.b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B|A&B);
}
if(strcmp(exp_log,"b+a.c")==0)
{
```

```
            Verifica_dados_entrada(buffer_rec_clp[0]);
            Saida0 = (B|A&C);
}
if(strcmp(exp_log,"b+b.a")==0)
{
            Verifica_dados_entrada(buffer_rec_clp[0]);
            Saida0 = (B|B&A);
}
if(strcmp(exp_log,"b+b.b")==0)
{
            Verifica_dados_entrada(buffer_rec_clp[0]);
            Saida0 = (B|B&B);
}
if(strcmp(exp_log,"b+b.c")==0)
{
            Verifica_dados_entrada(buffer_rec_clp[0]);
            Saida0 = (B|B&C);
}
if(strcmp(exp_log,"b+c.a")==0)
{
            Verifica_dados_entrada(buffer_rec_clp[0]);
            Saida0 = (B|C&A);
}
if(strcmp(exp_log,"b+c.b")==0)
{
            Verifica_dados_entrada(buffer_rec_clp[0]);
            Saida0 = (B|C&B);
}
if(strcmp(exp_log,"b+c.c")==0)
{
            Verifica_dados_entrada(buffer_rec_clp[0]);
            Saida0 = (B|C&C);
}
if(strcmp(exp_log,"c+a.a")==0)
{
            Verifica_dados_entrada(buffer_rec_clp[0]);
            Saida0 = (C|A&A);
}
if(strcmp(exp_log,"c+a.b")==0)
{
            Verifica_dados_entrada(buffer_rec_clp[0]);
            Saida0 = (C|A&B);
}
if(strcmp(exp_log,"c+a.c")==0)
{
            Verifica_dados_entrada(buffer_rec_clp[0]);
            Saida0 = (C|A&C);
}
if(strcmp(exp_log,"c+b.a")==0)
{
```

```
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|B&A);
}
if(strcmp(exp_log,"c+b.b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|B&B);
}
if(strcmp(exp_log,"c+b.c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|B&C);
}
if(strcmp(exp_log,"c+c.a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|C&A);
}
if(strcmp(exp_log,"c+c.b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|C&B);
}
if(strcmp(exp_log,"c+c.c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C|C&C);
}
if(strcmp(exp_log,"a.a+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A&A|A);
}
if(strcmp(exp_log,"a.a+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A&A|B);
}
if(strcmp(exp_log,"a.a+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A&A|C);
}
if(strcmp(exp_log,"a.b+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A&B|A);
}
if(strcmp(exp_log,"a.b+b")==0)
{
```

```
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A&B|B);
}
if(strcmp(exp_log,"a.b+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A&B|C);
}
if(strcmp(exp_log,"a.c+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A&C|A);
}
if(strcmp(exp_log,"a.c+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A&C|B);
}
if(strcmp(exp_log,"a.c+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (A&C|C);
}
if(strcmp(exp_log,"b.a+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B&A|A);
}
if(strcmp(exp_log,"b.a+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B&A|B);
}
if(strcmp(exp_log,"b.a+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B&A|C);
}
if(strcmp(exp_log,"b.b+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B&B|A);
}
if(strcmp(exp_log,"b.b+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B&B|B);
}
if(strcmp(exp_log,"b.b+c")==0)
{
```

```
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B&B|C);
}
if(strcmp(exp_log,"b.c+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B&C|A);
}
if(strcmp(exp_log,"b.c+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B&C|B);
}
if(strcmp(exp_log,"b.c+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (B&C|C);
}
if(strcmp(exp_log,"c.a+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&A|A);
}
if(strcmp(exp_log,"c.a+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&A|B);
}
if(strcmp(exp_log,"c.a+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&A|C);
}
if(strcmp(exp_log,"c.b+a")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&B|A);
}
if(strcmp(exp_log,"c.b+b")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&B|B);
}
if(strcmp(exp_log,"c.b+c")==0)
{
        Verifica_dados_entrada(buffer_rec_clp[0]);
        Saida0 = (C&B|C);
}
if(strcmp(exp_log,"c.c+a")==0)
{
```

```
                Verifica_dados_entrada(buffer_rec_clp[0]);
                Saida0 = (C&C|A);
        }
        if(strcmp(exp_log,"c.c+b")==0)
        {
                Verifica_dados_entrada(buffer_rec_clp[0]);
                Saida0 = (C&C|B);
        }
        if(strcmp(exp_log,"c.c+c")==0)
        {
                Verifica_dados_entrada(buffer_rec_clp[0]);
                Saida0 = (C&C|C);
        }
                return(Saida0);
}
```

APÊNDICE 2

PROGRAMA IMPLEMENTADO NO MICROCONTROLADOR PIC

```
#include <16f877A.h>
// Configura o compilador para conversor A/D de 10 bits
#device adc=10
#use delay(clock=4000000)
#fuses XT,NOWDT,PUT,NOBROWNOUT,NOLVP
#include <rs232.c>
#include <mod_lcd.c>
#include<stdlib.h>
long int newCo=0;
//Converte de char para int
int Conv_char_int(char temp)
{
  if(temp=='0')
    return(0);
  if(temp=='1')
    return(1);
  if(temp=='2')
    return(2);
  if(temp=='3')
    return(3);
  if(temp=='4')
    return(4);
  if(temp=='5')
    return(5);
  if(temp=='6')
    return(6);
  if(temp=='7')
    return(7);
```

```
   if(temp=='8')
      return(8);
   if(temp=='9')
      return(9);
}
//Funcao principal
void main()
{
// Declaração das variaveis do programa
   char temp;
   char temp1[7],PV_string[3];
   long int newDutyCicle=0;
   boolean status=0;
   unsigned int teste,i,j,t,temp_int=0;
   unsigned long int num_final=0;
   long int valor,atual_PV;
   int32 val32,val;
   output_bit(pin_d7,0);
//Inicializacao do protocolo RS232 via software
 rs232_inicializa();
//Configuracao do RA0 para entrada analogica
   setup_ADC_ports (RA0_analog);
   setup_adc(ADC_CLOCK_INTERNAL );
   set_adc_channel(0);
//Configuracao do PWM CCP2
//O tempo do ciclo é calculado por (1/clock)*4*t2_div*(periodo+1)
   setup_timer_2(T2_DIV_BY_16, 61, 1);  // timer 2 = 1,008 khz
   setup_ccp2 (ccp_pwm);  // configura CCP2 para modo PWM
   delay_ms(100);
   set_pwm2_duty (0);
 // configura o ciclo ativo em 0 (desligado)
   delay_ms(100);
   lcd_ini();
   lcd_escreve("\f>>PID + WinCe<<\n\r");
```

```
//Laço infinito.
 for(i=0;i<=3;i++)
   PV_string[i]='A';
 j=temp=i=0;
 while(true)
        {
   if(!pino_rx)
   {
     temp=rs232_recebe();
     temp1[i]=temp;
     i++;
   }
   //configuracoes de escrita do CLP (Dados enviado PELO controlador PARA o PIC)
   //000
   if(temp=='D')
   {
     output_bit(PIN_C4,0);
     output_bit(PIN_C5,0);
     output_bit(PIN_C6,0);
     lcd_escreve("\f  CLP Embedded\n\r");
     temp=0;
   }
   //001
   if(temp=='E')
   {
     output_bit(PIN_C6,1);
     output_bit(PIN_C5,0);
     output_bit(PIN_C4,0);
     lcd_escreve("\f  CLP Embedded\n\r");
     temp=0;
   }
   //010
   if(temp=='F')
   {
```

```
   output_bit(PIN_C6,0);
   output_bit(PIN_C5,1);
   output_bit(PIN_C4,0);
   lcd_escreve("\f  CLP Embedded\n\r");
   temp=0;
}
//011
if(temp=='G')
{
   output_bit(PIN_C6,1);
   output_bit(PIN_C5,1);
   output_bit(PIN_C4,0);
   lcd_escreve("\f  CLP Embedded\n\r");
   temp=0;
 }
//100
if(temp=='H')
{
   output_bit(PIN_C6,0);
   output_bit(PIN_C5,0);
   output_bit(PIN_C4,1);
   lcd_escreve("\f  CLP Embedded\n\r");
   temp=0;
}
//101
if(temp=='I')
{
   output_bit(PIN_C6,1);
   output_bit(PIN_C5,0);
   output_bit(PIN_C4,1);
   lcd_escreve("\f  CLP Embedded\n\r");
   temp=0;
}
//110
```

```
    if(temp=='J')
    {
      output_bit(PIN_C6,0);
      output_bit(PIN_C5,1);
      output_bit(PIN_C4,1);
      lcd_escreve("\f  CLP Embedded\n\r");
      temp=0;
    }
    //111
    if(temp=='K')
    {
      output_bit(PIN_C4,1);
      output_bit(PIN_C5,1);
      output_bit(PIN_C6,1);
      lcd_escreve("\f  CLP Embedded\n\r");
      temp=0;
    }
    //configuracoes de leitura do CLP (Dados enviados para o controlador pelo PIC)
    if(temp=='C')
    {
//COMBINACOES POSSIVEIS 000->P,001->Q,010->R,011->S,100->T,101->U,110->V,111->X,
      delay_ms(10);
      if(!input(pin_d1))
      {
        if(!input(pin_d2))
        {
          if(!input(pin_d3))
          {
            //000
            temp='P';
            lcd_escreve("recA=0,B=0,C=0 ");
          }
          else
```

```
      {
        //001
        temp='Q';
        lcd_escreve("recA=0,B=0,C=1   ");
      }

    }
    else
    {
      if(!input(pin_d3))
      {
        //010
        temp='R';
        lcd_escreve("recA=0,B=1,C=0   ");
      }
      else
      {
        //011
        temp='S';
        lcd_escreve("recA=0,B=1,C=1   ");
      }
    }
  }
  else
  {
    if(!input(pin_d2))
    {
      if(!input(pin_d3))
      {
        //100
        temp='T';
        lcd_escreve("recA=1,B=0,C=0   ");
      }
      else
```

```
        {
          //101
          temp='U';
          lcd_escreve("recA=1,B=0,C=1   ");
        }
      }
      else
      {
        if(!input(pin_d3))
        {
          //110
          temp='V';
          lcd_escreve("recA=1,B=1,C=0   ");
        }
        else
        {
          //111
          temp='X';
          lcd_escreve("recA=1,B=1,C=1   ");
        }
      }
    }
    //Envia dados pela Serial para o PC
    printf(rs232_transmite,"%c",temp);
    lcd_escreve("\f  CLP Embedded\n\r");
    i=temp=0;
}
//Envia para o controlador a PV inicial do sistema
if(temp=='B')
{
    delay_ms(100);
    valor = read_adc();
    if (valor)
      valor += 1;
```

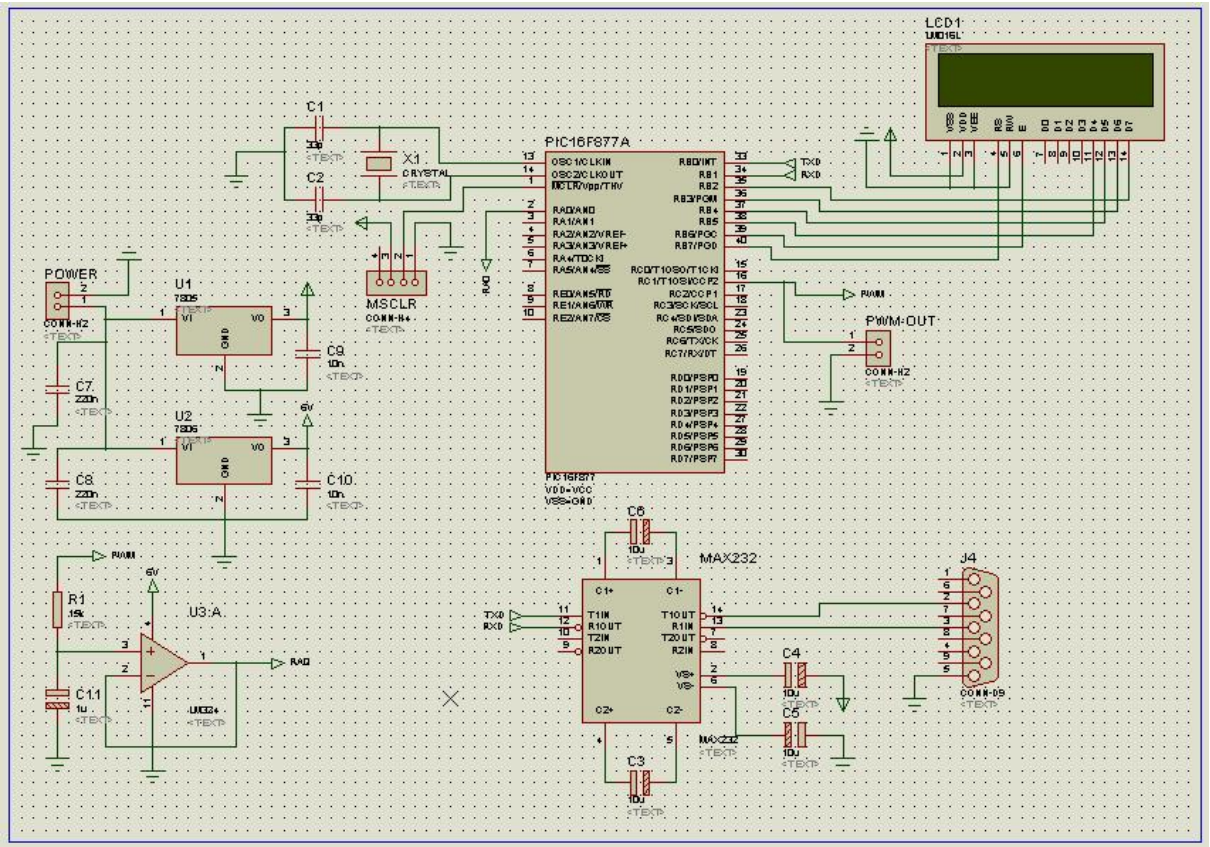
```
    val32 = valor * 4 + ((int32)valor * 113)/128;

     //Calculo da PV em porcentagem.
    atual_PV = (((val32) * 100) / (5000));
    printf(lcd_escreve,"\fPV: %lu %%\n\rCo: %lu %%",atual_PV,num_final);
    printf(rs232_transmite,"%luA",atual_PV);
    temp=i=0;
}
//Parte do protocolo que recebe informacoes do PC e trata o mesmo
if(temp=='A')
{
    output_bit(pin_d7,1);
          lcd_escreve("\f");
    i=i-2;
    for(t=0; t<=i;t++)
    {
      temp_int=Conv_char_int(temp1[t]);
      if(i==2)
      {
        if(t==0)
          num_final+=temp_int*100;
        if(t==1)
          num_final+=temp_int*10;
        if(t==2)
          num_final+=temp_int*1;
      }
      if(i==1)
      {
        if(t==0)
          num_final+=temp_int*10;
        if(t==1)
          num_final+=temp_int*1;
      }
      if(i==0)
```

```
            if(t==0)
              num_final+=temp_int*1;
        }
        i=temp=0;
        //Calcula_PWM(num_final);
        if(num_final<=0)
          num_final=1;
        if(num_final>=100)
          num_final=100;
        //Calcula PWM
        val=((255 * num_final) / 100);
        //ajusta a saida PWM
        set_pwm2_duty(val);
        delay_ms(100);
        valor = read_adc();
        if (valor)
          valor += 1;
           val32 = valor * 4 + ((int32)valor * 113)/128;
        //Calculo da PV em porcentagem.
        //O calculo abaixo que esta comentado foi utilizado para uma transmissao de 4~20mA.
        atual_PV =(((val32) * 100) / (5000));
        printf(rs232_transmite,"%luA",atual_PV);
        output_bit(pin_d7,0);
        printf(lcd_escreve,"\fPV: %lu %%\n\rCo: %lu %%",atual_PV,num_final);
        num_final=0;
           }
          }
}
```

APÊNDICE 3

CIRCUITO IMPLEMENTADO NA PLACA DE DESENVOLVIMENTO



RESUMO:

Com a crescente demanda por maior conectividade, acesso a Internet e funções multimídia, hoje demanda-se dos dispositivos dedicados maior sofisticação, incluindo novas funções e, muitas vezes, displays de vídeo para uma melhor experiência por parte do usuário final. A implementação de forma microcontrolada de todas essas novas funcionalidades seria bastante difícil e, em alguns casos, até mesmo inviável em função da memória máxima endereçada por microcontroladores. Os fabricantes de dispositivos resolveram a limitação dos ambientes microcontrolados buscando soluções microprocessadas, com sistemas operacionais modularizados e robustos, atendendo assim a nova demanda de mercado. Esta dissertação aborda conceitos relativos a sistemas embarcados (Embedded Systems), discorrendo sobre sua definição, principais aplicações do sistema operacional embarcado Windows CE na área de automação e controle, essas aplicações voltadas para a escola e a industria. Este trabalho descreve a implementação de uma proposta de aplicação do Windows CE em um ambiente didático. O ambiente descrito visa atender à demanda por um sistema aberto, de fácil adaptação, custo adequado, com grande capacidade de processamento encontrada principalmente no meio acadêmico, mas extensível a aplicações industriais.

PALAVRAS-CHAVE

Windows CE, Automação, Controle, eletrônica embarcada.

ÁREA/SUB-ÁREA DE CONHECIMENTO

3.00.00.00-9 - Engenharias

3.04.00.00-7 - Engenharia Elétrica

1.03.00.00-7 - Ciência da computação

Ano 2007

N°:449

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo