UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO

COMPONENTES PARAMETRIZÁVEIS PARA SIMULAÇÃO DE HARDWARE DE SISTEMAS EMBARCADOS

Simulação de Sistemas

Volnir dos Santos Sobrinho

Itajaí (SC), julho de 2004

UNIVERSIDADE DO VALE DO ITAJAÍ
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA TERRA E DO MAR
CURSO DE CIÊNCIA DA COMPUTAÇÃO

RELATÓRIO DO TRABALHO DE CONCLUSÃO DE CURSO

COMPONENTES PARAMETRIZÁVEIS PARA SIMULAÇÃO DE HARDWARE DE SISTEMAS EMBARCADOS

Simulação de Sistemas

Volnir dos Santos Sobrinho

Relatório apresentado à Banca Examinadora do Trabalho de Conclusão do Curso de Ciência da Computação para análise e aprovação

Itajaí (SC), julho de 2004

EQUIPE TÉCNICA

Acadêmico

Volnir dos Santos Sobrinho

Professor Orientador

Rafael Luiz Cancian, M.Sc.

Professor Co-orientador

Cesar Albenes Zeferino, Dr.

Coordenadores dos Trabalhos de Conclusão de Curso

Anita Maria da Rocha Fernandes, Dra.

Cesar Albenes Zeferino, Dr.

Coordenador do Curso

Luis Carlos Martins, Esp.

DEDICATÓRIA

Dedico este trabalho aos meus pais e a minha noiva, que são pessoas de grande influência na minha vida.

AGRADECIMENTOS

Primeiramente agradeço a Deus, pelo seu infinito amor, e por ter me dado a alegria de viver que possuo e capacidade de engrandecer em todas etapas da minha vida.

Aos meus pais (Solange e Valdir dos Santos), pelo carinho, força e compreensão e por toda dedicação a mim proporcionada.

A meus irmãos (Gerson e Jaqueline dos Santos) que, mesmo sem paciência, são obrigados a me aturar.

Ao meu orientador Rafael Luiz Cancian, pelo incentivo e pelo grande conhecimento que sua orientação me proporcionou.

Ao meu co-orientador Cesar Albenes Zeferino, pelo seu conhecimento que auxiliou em algumas etapas deste trabalho.

A todos os meus amigos, "amigas", galera dos pagodes da vida e colegas da faculdade, pelo convívio e pela força que cada um me proporcionou.

Por fim, a empresa onde trabalho *Buettner* S/A pela força e compreensão na continuação deste trabalho.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

A/D	Analog/Digital
ABS	Antilock Brake Systems
ASCII	American Standard Code for Information Interchange
BCD	Binário Convertido para Decimal
BEDO	Burst Extended Data Output RAM
BIOS	Basic Input Output System
BPS	Bytes por segundo
CAS	Column Address Strobe
CI	Circuito Integrado
CMOS	Complementary Metal-Oxide Semiconductor
CPU	Central Process Unit
D/A	Digital/Analog
DCE	Data Comunication Equipment
DLL	Dinamic Link Library
DMA	Direct Memory Access
DST	Data Set Ready
DTE	Data Terminal Equipment
DTR	Data Terminal Ready
ECL	Lógica de Emissor Acoplada
EDO	Extended Data Output
EEPROM	Electrical Erasable Programmable Read Only Memory
EIA	Electronics Industry Association
EPROM	Erase Programmable Read Only Memory
FIFO	First In First Out
FPM	Fast Page Mode
FTP	File Transfer Protocol
GPS	Global System GPS
HTTP	Hyper Text Tranfer Protocol
IBM	Internation Business Machine
LCD	Liquid Crystal Display

LED	Light Emitting Diode
MEMS	Microeletromecânicos
PC	Personal Computer
PN	Positiva/Negativa
PSEN	Program Storage Enable
PWM	Pulse Width Modulation
RAM	Random Access Memory
RAS	Row Address Strobe
RISC	Reduced Instruction Set Computing
ROM	Read Only Memory
SDRAM	Synchronous Dynamic RAM
SSD	Sistema de Simulação Discreta
TCC	Trabalho de Conclusão de Curso
TTL	Transistor Transistor Logic
UART	Universal Asynchronous Receiver Transmitter
UC	Unidade de Controle
ULA	Unidade Lógica e Aritmética
UML	Unfied Modeling Language
UNIVALI	Universidade do Vale do Itajaí
VHDL	VHSIC Hardware Description Language
VLSI	Very Large Scale Integration

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O presente trabalho apresenta o desenvolvimento de componentes para simulação de *hardware* de sistemas embarcados em uma ferramenta de simulação genérica. Essa ferramenta, mais precisamente o SSD (Sistema de Simulação Discreta), é capaz de simular modelos descritos pelos usuários e mostrar resultados estatísticos para posterior análise. Assim, este trabalho utiliza essa ferramenta para representação de alto nível dos componentes de sistemas embarcados, verificando se é possível conseguir indicadores estatísticos de desempenho de um futuro sistema embarcado já nas etapas iniciais de projeto, sem a necessidade de sua especificação completa. A validação deste trabalho foi concebida através de dois modelos projetados, analisados e implementados em outras ferramentas de simulação específicas e também no SSD, onde foram submetidos à simulação computacional, e os resultados comparados estatisticamente através de teste de hipóteses. Os resultados indicam que a simulação de alto nível, conseguida nas etapas iniciais do projeto de um sistema embarcado, pode ser útil se a confiança estatística nos resultados for relaxada até certo ponto.

ABSTRACT

This work presents the development of components for hardware simulation of embedded systems in a generic simulation tool. This tool, in fact SSD (Sistema de Simulação Discreta), is used to simulate the models described by users and shows statistic results for a subsequent analysis. This work intends to use the tool to cerate a high level representation of embedded systems' components, verifying if it is possible to get performance metrics of future embedded systems in the beggining of its project without the necessity of complete specification. The validation of this work was conceived through two projected models, analyzed and implemented in others tools of specifics simulation and also in SSD, where were submitted to a computational simulation, and the results compared statistically through hypotheses tests. The results shows that simulation in high level, done in the beggining of embedded system projects, can be useful if the statistic trust of the results be relaxed until certain point.

I - INTRODUÇÃO

1. APRESENTAÇÃO

O presente Trabalho de Conclusão de Curso visa modelar o funcionamento dos componentes presentes em sistemas embarcados, tais como microcontroladores, memórias, barramentos, dispositivos de entrada/saída e outros sistemas digitais, através da utilização de simulação discreta de sistemas. Para isso será utilizada uma ferramenta de simulação discreta desenvolvida e expandida por Stefanes (2002), onde uma de suas características é permitir que se desenvolvam componentes que representam algum comportamento de sistemas reais. Esses componentes são bibliotecas de *software* elaborados por uma ferramenta de programação e seguem uma estrutura imposta pelo sistema. Após a sua elaboração, eles podem ser incluídos na ferramenta como *plug-ins*, fazendo parte dela, e podendo ser utilizados em conjunto com outros componentes da ferramenta, formando um modelo completo a ser simulado, fornecendo resultados referentes ao desempenho e custo, assim como uma representação gráfica resumindo os valores obtidos.

2. JUSTIFICATIVA

Para Torres (1998), a simulação de sistemas é uma das melhores alternativas para obter informações referentes ao comportamento de um sistema. Após a realização de alterações de parâmetros ou fatores, poderão vir a alterar o funcionamento do mesmo. Dessa forma, serão descartadas as operações envolvendo o sistema real.

Segundo Scriber (1991), a crescente complexidade de produtos eletrônicos tecnológicos tem conduzido projetistas e pesquisadores a elevar cada vez mais o nível de abstração de tarefas como especificação e validação de sistemas digitais.

A proposta deste trabalho é o desenvolvimento de componentes para uma ferramenta de simulação discreta capaz de representar o funcionamento de alto nível de sistemas embarcados e redes-em-chip. Também será possível parametrizar esses componentes e obter informações estatísticas de forma que possam ser analisados seus resultados. A geração de componentes para um

software de simulação vai auxiliar analistas, engenheiros e técnicos ligados nas áreas de projetos de sistemas embarcados e redes-em-chip a obterem resultados que determinarão a viabilidade ao projeto em estudo.

Assim, professores e alunos da UNIVALI (Universidade do Vale do Itajaí) se beneficiarão desses componentes para simular arquiteturas básicas de microcontroladores e na iniciação do estudo de projetos de sistemas embarcados e redes-em-chip que por ventura possam ser introduzidos nesta instituição.

Os componentes que serão desenvolvidos levam em consideração o conhecimento de simulação de sistemas, sistemas embarcados e redes-em-chip, assim como o domínio de alguns circuitos digitais típicos.

3. IMPORTÂNCIA DO TRABALHO

Este projeto deve permitir que estudantes e técnicos possam conceber um sistema digital e simulá-lo com o objetivo de compreender seu funcionamento e/ou verificar seu funcionamento/desempenho, antes de prototipá-lo.

Diversos simuladores de circuitos digitais existem no mercado. Porém, eles simulam unicamente sistemas digitais, não permitindo a inclusão no modelo de outros objetos do mundo real. Com a criação de componentes de *hardware* para uma ferramenta genérica de simulação discreta, o usuário poderá incluir o sistema digital num ambiente muito mais amplo.

Além disso, espera-se obter maior flexibilidade e rapidez na modelagem e simulação de sistemas embarcados através de sua representação em alto nível, pois não é necessário ter todo o sistema especificado para que possa ser modelado.

4. OBJETIVOS DO TRABALHO

4.1 Objetivo Geral

O principal objetivo deste trabalho é o desenvolvimento de componentes que modelem o comportamento de circuitos digitais comuns (memória, processador, coprocessadores, controladores de E/S) e arquiteturas de interconexão (barramentos, redes-em-chip) e que possam ser incorporados ao SSD (Sistema de Simulação Discreta), visando possibilitar a construção de modelos de sistemas embarcados para posterior simulação e avaliação de desempenho.

4.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Estudo e modelagem de componentes que visam a simulação de sistemas embarcados e redes-em-chip;
- Implementar componentes de simulação que representem o comportamento de pelo menos os seguintes circuitos básicos: microprocessador, memórias (RAM, ROM, EEPROM), controladores de E/S (UART, RS-232) e dispositivos periféricos (LED, LCD, A/D, D/A, motores de passo);
- Implementar componentes de simulação que representem o comportamento dos seguintes circuitos de redes-em-chip: canais, roteador genérico, núcleos;
- Criar dois modelos de sistemas embarcados que utilizem os componentes desenvolvidos, como objetivo de verificar e validar tais componentes;
- Comparar os resultados dos modelos gerados no SSD com modelos de simuladores existentes e, se possível, com sistemas reais.

5. METODOLOGIA

Para que todo trabalho científico chegue ao seu objetivo, todas as tarefas devem ser divididas em etapas bem definidas, que serão seguidas para a sua realização.

O desenvolvimento do presente trabalho está dividido em 5 fases principais: (i) Estudos; (ii) Modelagem; (iii) Implementação; (iv) Testes e validação; (v) Documentação e apresentação.

Na primeira fase, de Estudos, estão compreendidas as seguintes etapas:

i.a) Estudo de simulação de sistemas e do simulador SSD. Para que os componentes fossem desenvolvidos nesta ferramenta, realizou-se um estudo teórico sobre os conceitos da simulação, assim como o simulador cuja expansão para modelar redes de computadores, defendido num recente Trabalho de Conclusão de Curso realizado nesta instituição (STEFANES, 2002). Portanto, foram obtidas as informações necessárias deste *software,* assim como sua codificação, a fim de obter o conhecimento necessário para implementação nessa ferramenta, base para realização deste presente trabalho;

i.b) Estudo de sistemas digitais e sistemas embarcados. Em decorrência do propósito do trabalho em fornecer estruturas para poder simular elementos de *hardware* voltados a sistemas embarcados foi necessário obter o conhecimento prévio de sistemas digitais, assim como seus circuitos típicos, estruturas e funcionamento, para prosseguir com base necessária ao estudo de sistemas embarcados através do levantamento de conceitos, técnicas, modelos de sistemas já desenvolvidos, e o estudo de alguns componentes de *hardware* como microcontroladores, memórias e dispositivos de entrada/saída;

i.c) Estudo de redes-em-chip para que visa implementar os componentes relacionados aos principais circuitos de redes acima propostos. Em virtude disso, alguns elementos voltados a suportar essa arquitetura como roteadores, protocolos de comunicação e estrutura em geral serão estudados para posteriores implementações descritas na segunda e terceira fase; e

i.d) Estudo de modelagem de sistemas embarcados. É necessário obter um conhecimento sobre as metodologias e modelagens para poder conceber mais facilmente a implementação dos dois modelos de sistemas embarcados no qual servirá como teste e validação do projeto.

Na segunda fase, de Modelagem, estão compreendidas as seguintes etapas:

ii.a) Modelagem dos circuitos digitais. Todas características dos circuitos digitais que serão utilizados para construir os componentes foram modelados nesta etapa. Circuitos como controladores periféricos, memórias e microcontroladores fazem parte deste grupo;

ii.b) Modelagem dos circuitos de rede-em-chip. Uma breve descrição teórica desses circuitos e assim como a modelagem dos principais componentes de uma rede-em-chip é abordada nesta etapa; e

ii.c) Modelagem de sistemas embarcados utilizando os circuitos já modelados. Nesse ponto praticamente todos os componentes estão modelados e prontos para implementação como microcontroladores, memória externa, barramento e alguns periféricos.

Na terceira fase, de Implementação, estão compreendidas as seguintes etapas:

iii.a) Implementação dos componentes de circuitos digitais. Todos os componentes que formam o sistema embarcado modelados anteriormente serão implementados através de uma ferramenta visual capaz de gerar os arquivos dos componentes e inclusos no SSD;

iii.b) Implementação dos componentes de redes-em-chip. O mesmo procedimento que foi utilizado na implementação dos componentes de sistemas embarcados;

iii.c) Criação dos modelos de simulação de sistemas embarcados propostos. Nesta etapa dois sistemas embarcados serão modelados pela ferramenta SSD utilizando os componentes criados nas etapas anteriores.

Na quarta fase, de Testes e Validação, estão compreendidas as seguintes etapas:

iv.a) Testes de funcionamento dos componentes implementados. Os dois sistemas embarcados projetados serão testados para validar o funcionamento dos mesmos. Possivelmente um desses sistemas pode ser modelado a partir de algum trabalho de conclusão de curso envolvendo sistemas embarcados. Se for um modelo físico, será portado para uma ferramenta de simulação de sistemas embarcados obedecendo a suas características. Estatísticas como desempenho serão levadas em consideração nesta etapa.

iv.b) Experimentação dos modelos de simulação propostos. Os dois sistemas embarcados modelados com a ferramenta SSD serão submetidos a dois tipos de teste: um com valores aleatórios e outro a partir de um modelo de sistema embarcado físico implementado por algum trabalho de conclusão de curso e modelado através de uma ferramenta de simulação específica de sistemas embarcados. O objetivo desses dois testes é procurar observar seu comportamento em situações adversas, mas não fugindo de situações reais.

iv.c) Comparação dos resultados dos modelos de simulação com sistemas reais ou outros modelos provenientes de outras ferramentas de simulação para validar os componentes. As ferramentas podem ser genéricas, como Arena como também específicas para sistemas embarcados e circuitos digitas como Max+Plus II (ALTERA), *Virtual Breadboard* e Proteus/Isis; e

iv.d) Avaliação dos dados estatísticos para validação dos modelos de simulação. A coleta de informações estatísticas de ambos os sistemas (simulado e real) é necessária para validação do sistema simulado.

Na quinta fase, de Documentação e Apresentação, estão compreendidas as seguintes etapas:
v.a) Escrita do Trabalho de Conclusão de Curso;

v.b) Escrita de uma Artigo científico.

II – REVISÃO BIBLIOGRÁFICA

1. SISTEMAS EMBARCADOS

Este capítulo apresenta um panorama geral sobre Sistemas Embarcados (*Embedded System*). Serão apresentadas suas principais características. Além disso, serão descritos alguns componentes de *hardware* de sistemas embarcados visando sua futura implementação para simulação. Uma abordagem mais ampla será direcionada a microprocessadores e microcontroladores, base de um sistema embarcado. Algumas formas estatísticas de avaliação de desempenho desses sistemas também serão analisadas.

1.1. Evolução dos Sistemas Digitais

A maior parte do desenvolvimento de métodos e ferramentas para projeto de sistemas puramente digitais das últimas quatro décadas (60 a 90) derivou das necessidades relacionadas exclusivamente com projetos de computadores. Eles surgiram na década de 50 e eram submetidos à operação de técnicos especializados que não interagiam com as máquinas. Esses computadores chegavam a ocupar grandes salas devido a seu tamanho (TANENBAUM, 1999). Contudo, gradativamente, o enfoque tem se deslocado a pesquisas de produtos de uso específicos.

Segundo Schriber (1991), a crescente complexidade de produtos eletrônicos tecnológicos tem conduzido projetistas e pesquisadores a elevar cada vez mais o nível de abstração de tarefas como especificação e validação de sistemas digitais.

A evolução da microeletrônica e a redução de custo das CPU´s viabilizaram o emprego de sistemas computadorizados em diversos equipamentos, desde celulares e *palms* como televisores e máquinas de uso doméstico. Os cérebros da maioria dos equipamentos modernos são os pequenos computadores que eles trazem embutidos. Porém, esses equipamentos possuem aspectos físicos muito diferentes dos computadores de hoje em dia. De Micheli (1995) expõe que a maioria dos sistemas digitais modernos são programáveis, consistindo de componentes de *software* e *hardware*. O mesmo autor define projeto integrado de *hardware* e *software* como: “a busca do alcance dos

objetivos em nível de sistema do produto pela exploração da sinergia entre *hardware* e *software*, através do projeto concorrente dessas entidades".

1.2. Definição de Sistemas Embarcados

Sistemas embarcados são, para todos os efeitos, sistemas computacionais que executam uma função específica. Eles possuem a mesma estrutura geral de um computador, mas a especificidade de suas tarefas faz com que não sejam usados e nem percebidos como um computador (DE MICHELI, 1995). Segundo Wolf (2002), sistema embarcado é um dispositivo em que é incluída uma programação computacional, mas não pretende ser um computador de propósito geral.

Stankovic (1992) especifica que os sistemas embarcados são freqüentemente sistemas reativos em tempo-real utilizados para processamento de sinais, imagens, telecomunicações e automação. Tipicamente, eles são implementados a partir de diferentes tecnologias, como microprocessadores, microcontroladores, e microeletromecânicos (MEMS). Um sistema embarcado geralmente interage continuamente com o ambiente exterior, estando sujeito a restrições temporais (latência e cadência) e de embarcabilidade (redução de volume, peso, superfície, consumo).

Sistemas digitais baseados em computadores têm sido aplicados desde os primórdios da computação (WOLF, 2002). Um exemplo foi o *Whirlwind*, um computador da década de 1940. Ele foi o primeiro computador modelado para execução de tarefas em tempo real, também originalmente foi projetado para controle de simulação de aeronaves. Sua dimensão física era extremamente gigantesca, possuindo cerca de 4000 válvulas. Esse foi o primeiro registro da utilização de um computador totalmente desenhado para aplicações em tempo real para substituir o ser humano em determinadas tarefas.

1.3. Características de Sistemas Embarcados

Em geral, os sistemas embarcados são muito mais exigentes que sistemas projetados para serem executados em PC´s ou *Workstations*. A funcionalidade de um sistema desse tipo faz com que se observem algumas características descritas por Wolf (2002):

- Algoritmos complexos: As operações realizadas por microprocessadores podem ser muito sofisticadas, pois como tratam de dispositivos geralmente empregados em aplicações de tempo real, o desempenho e o tempo de resposta são itens de grande importância e muito relevantes na concepção de projeto de sistemas embarcados. Como exemplo pode ser citado o freio ABS (*Antilock Brake Systems*) usado em automóveis de última geração, onde demonstra que o tempo de resposta desse sistema tem que ser rápido;
- Interface com o usuário: Microprocessadores são freqüentemente usados para controlar interfaces complexas com múltiplos menus e muitas opções. O movimento de mapas de um Sistema de Posicionamento Global (*Global Positioning System, GPS)* é um exemplo de uma interface sofisticada;
- Tempo Real: Muitos sistemas embarcados são aplicados em tarefas de tempo real. Caso as informações não sejam processadas em um certo prazo limite (*deadline)*, o sistema sofre uma falha. Em alguns casos, uma falha em um desses sistemas pode ocasionar problemas graves assim como pôr em risco vidas humanas, como é o caso de circuitos *hard real-time*.;
- *Multirate:* Refere-se à quantidade de tarefas que um sistema embarcado pode executar ao mesmo tempo. Além dessas várias tarefas a serem executadas pelo sistema, muitas delas são sincronizadas. Um exemplo disso são aplicações multimídia, onde deve haver um sincronismo entre áudio e vídeo, pois qualquer atraso de tempo entre eles pode ser facilmente percebido;
- Custo de Fabricação: O custo total de um projeto de sistemas embarcados é muito importante quando um sistema é fabricado em grande escala. Esse custo é determinado por diversos fatores, entre eles, o tipo de microprocessador usado, o tamanho da memória utilizada e os dispositivos de entrada/saída utilizados;
- Consumo de Energia: Esse fator de suma importância afeta diretamente no custo total de um projeto de sistema embarcado. O consumo de energia tem impacto no tempo de vida útil de baterias, tipicamente usadas nesses sistemas. Além disso, são levados em conta fatores como o tamanho do dispositivo e a sua dissipação de calor; e
- Tamanho do Sistema: Sistemas embarcados geralmente são dispositivos portáteis ou colocados em locais onde há pouco espaço físico. A redução no tamanho dos circuitos assim como em todos os dispositivos envolvidos nos projetos são fatores desafiantes na concepção de um projeto de sistema embarcado.

1.4. Alguns Problemas em Projeto de Sistemas Embarcados

Para Wolf (2002), a concepção de projetos de sistemas embarcados possui alguns fatores de extrema importância que possam dificultar ou até mesmo invialibizar na sua realização:

- A Necessidade de *Hardware:* É sempre muito levado em consideração a quantidade e a qualidade dos componentes que irão integrar um sistema embarcado. Deve-se observar esse ponto com mais atenção no caso do sistema ter a necessidade de alto desempenho, principalmente em aplicações de tempo real.
- Consumo de Energia: Essa consideração é ainda mais importante quando o sistema será dependente de energia portátil, como baterias e pilhas. Esse consumo de energia está ligado diretamente à dissipação de calor dos componentes, que, por serem na maioria um conjunto de CIs (Circuitos Integrados), é mais sensível a variações térmicas.
- Atualização do sistema: O *hardware* pode ser reutilizado para diferentes aplicações, como também pode sofrer atualização de sua versão de *software*, pois as correções de eventuais erros sempre são necessárias (DE MICHELI, 1995).

Segundo Wolf (2002), ainda há outras considerações a serem observadas quanto ao desenvolvimento do *software* de sistemas embarcados.

- Teste Complexo dos Algoritmos: A realização de testes do *software* de sistemas embarcados é mais difícil em relação aos PCs atuais. Além de geralmente o código da linguagem de programação atingir a um nível mais baixo, a tarefa de obtenção de dados para teste é bastante dificultada pelo fato de tais sistemas na maioria dependerem de informações externas provenientes de sensores e de outros dispositivos periféricos.
- Limitações de Visualização de Controle: Sistemas embarcados geralmente são dispositivos que não interagem diretamente com o homem. Sendo assim, alguns periféricos, como teclados e monitores, não estão presentes, dificultando a entrada de dados e a visualização do que está acontecendo no momento da execução.

1.5. Componentes de Sistemas Embarcados

A Figura 1 ilustra alguns dos componentes de um sistema embarcado típico.

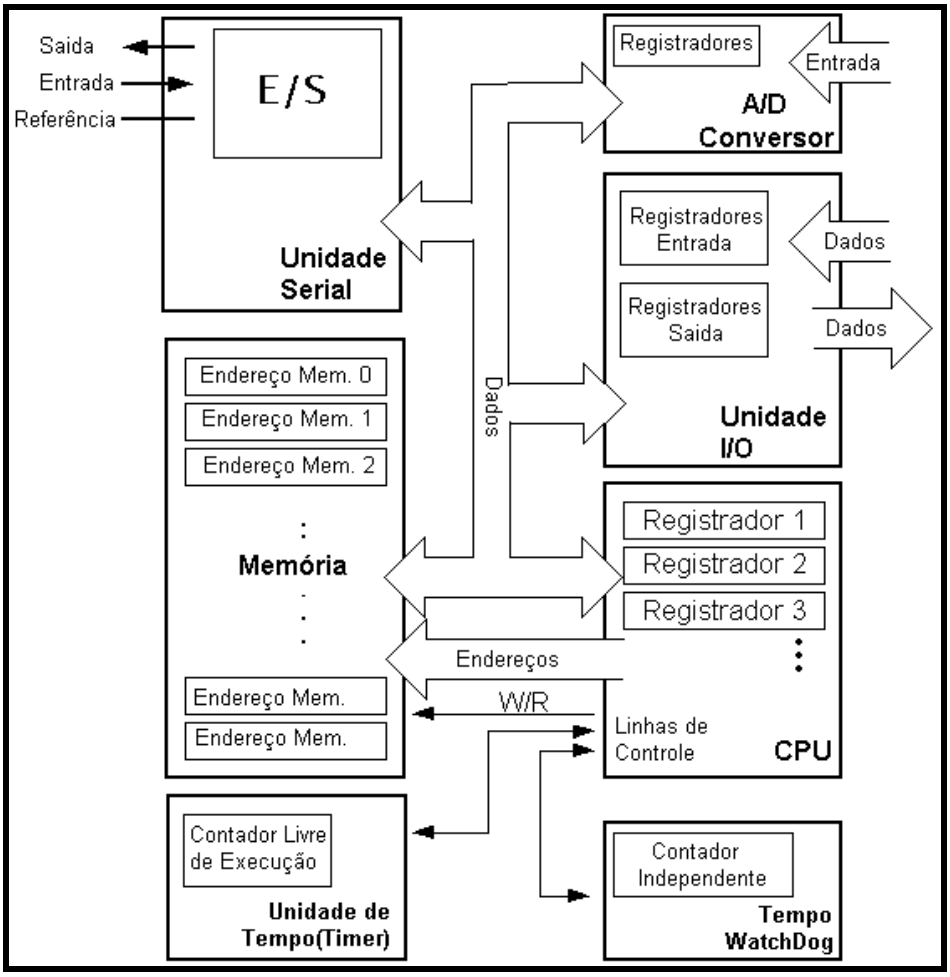

Figura 1. Componentes de um Sistema Embarcado.

Fonte: Adaptado de Matic & Andric (2001)

Abaixo é apresentada a descrição de alguns componentes utilizados em projetos de sistemas embarcados e que serão utilizados em simulação neste trabalho de conclusão de curso:

- Microcontroladores: Geralmente os microcontroladores são confundidos com microprocessadores, mas eles possuem características que os diferenciam em muitos aspectos. A primeira e a mais importante é a sua funcionalidade. Em um sistema com microprocessador, outros dispositivos tais como memória ou componentes para comunicação serial ou paralela, devem ser adicionados ao sistema. Outra característica é a maior capacidade de processamento do microprocessador, visto que as tarefas atribuídas a ele são mais complexas e de maior volume de dados (MATIC & ANDRIC, 2001). Os microcontroladores são projetados para realizarem todas essas funções em um único chip. A maioria dos componentes necessários para uma aplicação estão embutidas dentro do

microcontrolador, diminuindo principalmente seu custo e o ganho de espaço físico no projeto de *hardware*. (SILVEIRA, 2003);

- Memória: Para MORIMOTO (2003), a memória é o componente que possui a função de armazenar os dados e instruções que serão processadas. Geralmente, ela é parte do microcontrolador, mas devido a pouca capacidade de armazenamento oferecida às vezes, faz-se necessário o uso de memória externa;
- CPU (*Central Processor Unit)*: Certamente é o componente mais importante de um sistema embarcado, pois é necessária para gerenciar todo o microcontrolador. Ela é subdividida em outros pequenos dispositivos, cujas funções são descritas abaixo:
    - ULA (Unidade Lógica e Aritmética): Este componente é responsável por todas operações que modificarão os dados. Operações lógicas, como E, OU e operações aritméticas são realizadas neste dispositivo;
    - UC (Unidade de Controle): Todo o funcionamento da CPU depende de uma série de controles, como, por exemplo, a busca de instruções e dados da memória, o gerenciamento do barramento interno, entre outras;
    - Conjunto de Registradores: São um pequeno banco de memória cujo acesso é praticamente imediato. É utilizado para obter maior desempenho no processamento e também no controle da CPU em geral. Ela possui registradores específicos, para determinados componentes, como da ULA ou da UC e registradores de sinais de controle chamados de FLAGs, registradores de pilhas, de endereçamento entre outros (MATIC & ANDRIC, 2001);
    - Barramento Interno: É a via onde percorrem todas as informações entre CPU e seus componentes.
- Barramento: O caminho entre os dispositivos da CPU ou dos componentes que formam um sistema embarcado é chamado de barramento. Fisicamente, ele representa um conjunto de condutores de sinais chamados vias. Existem alguns tipos de barramento, mas os principais são barramento de dados, endereços e de controle. O barramento de endereços consiste de uma quantidade de linhas suficientes para acessar o tamanho de endereço de memória que se queira referenciar. Esse barramento é utilizado para informar à memória qual posição será acessada em um ciclo de leitura ou escrita. O barramento de dados consiste de uma quantidade de linhas correspondente ao tamanho ou magnitude do dado a ser manipulado pela CPU. O barramento de dados é responsável por transmitir uma informação entre os componentes que são direcionados e habilitados pelo barramento de controle. Para Rosh

(1996), a concepção de barramento interno trouxe uma enorme funcionalidade ao microcontrolador e aos processadores de uma maneira geral. Entretanto, até então, o microcontrolador não tem qualquer contato com mundo externo;

- Dispositivos de Entrada/Saída: Esses componentes descrevem as portas de comunicação para o mundo externo. Existem três tipos de portas: entrada, saída ou portas bidirecionais. Quando se utiliza uma porta, primeiro é necessário configurar o tipo de porta que se quer trabalhar. Depois escreve-se ou lê-se os dados, dependendo do tipo de porta selecionada. As operações com as portas funcionam como simples operações de leitura ou escrita em uma posição de memória específica. Uma simples operação de escrita na porta levará o dado correspondente para os pinos externos do circuito integrado (MORIMOTO, 2003);
- Comunicação Serial: Sua principal função é a comunicação, via barramento, de dados bit a bit. Uma grande vantagem desse tipo de comunicação é o pequeno número de linhas necessárias para transferência dos dados, geralmente duas. Com comunicação serial pode-se reduzir o número de ligações entre os sub-sistemas, sem perda de funcionalidade. Para que a comunicação possa ser feita, é necessário estabelecer determinadas regras de comunicação, denominados protocolos de comunicação. Esses protocolos possibilitam os sistemas conversarem utilizando a mesma linguagem;
- Temporizadores (*Timers*): Para Matic & Andric (2001), em algumas aplicações, é necessário controlar o sistema através de eventos no decorrer de sua execução. Para isso, utiliza-se de um dispositivo temporizador ou *Timer*. Este é contruído usando um circuito digital simples cuja sua principal funcionalidade é contar desde o início da aplicação, ou seja, um registrador que tem seu valor sempre incrementado;
- *WatchDog Timer*: Um outro aspecto que merece atenção é a condição de perfeito funcionamento do microcontrolador durante seu tempo de execução. Para a maioria das aplicações de sistemas embarcados, quando há um congelamento no sistema e ele pára de responder, é impraticável reiniciar ou desligar o sistema de forma manual. O *Watchdog* é um circuito digital que possui um contador que precisa ser zerado periodicamente pelo programa aplicativo que está rodando no microcontrolador (SILVEIRA, 2003). Caso o programa pare de responder, o contador deixará de ser zerado, e sua contagem irá estourar gerando assim um reinício automático no microcontrolador. O W*atchdog Timer* confere ao sistema um grau maior de confiabilidade sem a necessidade de supervisão humana;
- Conversor Analógico/Digital: Como os sinais de muitos dispositivos periféricos são diferentes dos sinais entendidos pelo microcontrolador (zeros e uns), eles precisam ser

convertidos para o padrão digital aceito. Essa tarefa é realizada pelo circuito de conversão analógico-digital, ou simplesmente A/D;

- Motores de Passo: Para Dorm (2003), o motor de passo é um componente que converte energia elétrica em energia mecânica, como qualquer outro motor elétrico. A sua principal diferença baseia-se em um sistema de controle que possui um circuito oscilador que gera um sinal cuja freqüência estaria diretamente relacionada com a velocidade de rotação do motor de passo. Essa freqüência é facilmente alterada dentro de um determinado valor e assim o motor apresenta uma rotação mínima ou máxima. Também possui certo grau de frenagem, que dá-se simplesmente pela inibição do sinal gerado pelo oscilador.

1.5.1 Microcontroladores/ Microprocessadores

1.5.1.1. Introdução

A tecnologia de fabricação de circuitos integrados em larga escala (VLSI – *Very Large Scale Integration*) proporcionou avanços na concepção de projetos de sistemas embarcados. A partir dela foram elaborados circuitos eletrônicos miniaturizados e encapsulados num material extremamente isolante. Essas características tornaram os CIs, ou *Chips,* um dos componentes mais utilizados e de maior importância na elaboração de microcomputadores e outros sistemas digitais. Foi a partir daí que grandes computadores formados por milhares de transístores, passaram a receber apenas algumas dezenas de *Chips*.

O microcontrolador é um desses CIs que possui grande importância em sistemas embarcados: são deles, principalmente, o mérito da larga aplicação que se tem nesse tipo de sistema.

1.5.1.2. História

A história dos microcontroladores/microprocessadores, segundo Matic & Andric (2001), será apresentada a seguir:

1969: Surgiu o conceito que as funções de circuitos integrados são determinadas por um programa armazenado em sua memória, e não somente por suas características físicas.

1971: A INTEL lança no mercado o microprocessador chamado 4004. Ele foi o primeiro microprocessador de 4-bits com a velocidade de 6.000 operações por segundo.

1972: Com base na aplicação da arquitetura da INTEL, a Texas *Instruments* começa a trabalhar na fabricação do primeiro microprocessador de 8-bits, chamado 8008. Esse microprocessador foi o precursor de todos os microprocessadores de hoje.

1975: A empresa MOS *Technology* colocou no mercado os microprocessadores 6501 e 6502, a um custo de $25 cada, comparados aos $179 dos microprocessadores 8080 e 68000. Como resposta, os competidores INTEL e MOTOROLA, baixaram respectivamente, seus preços de imediato para $69.95 por microprocessador.

1976: Frederico Faggin, engenheiro idealizador dos microcontroladores da Intel, saiu dessa empresa e fundou sua própria: a Zilog, que anunciou o Z80 com total compatibilidade com o processador 8080. Assim, todos os programas já desenvolvidos no mundo para o 8080 poderiam funcionar com o processador Zilog. O Z80 foi um grande sucesso e muitos migraram do 8080 para o Z80. Pode-se dizer, sem dúvida, que o Z80 foi o microprocessador de 8-bits de maior sucesso comercial de sua época.

1.5.1.3. Microcontroladores

Segundo Silveira (2003), os microcontroladores são componentes eletrônicos que integram, numa única pastilha, um processador e vários outros dispositivos como conversores A/D (Analógico/Digital), memórias, temporizadores, *interface* de comunicação serial, entre outros. Desse modo, permitem redução de custos e de tamanho físico, além de conferir versatilidade ao *hardware*.

O termo “controlador” é usado para designar o dispositivo que controla um processo ou algum parâmetro do ambiente. No princípio, os microcontroladores usavam lógica discreta e, por isso, tinham um tamanho que dificultava seu emprego em sistemas pequenos. Hoje em dia usam-se os circuitos integrados microprocessados e programáveis onde todo o controlador cabe em uma pequena placa de circuito impresso (NICOLOSI, 2003).

O microcontrolador, com o avanço da microeletrônica, recebeu, uma quantidade de recursos e componentes cada vez maior, fazendo com que o microcontrolador possa ter a maioria dos componentes necessários para desenvolver projetos eletrônicos (ROSH, 1996).

Existe uma quantidade expressiva de microcontroladores, porém, os mais conhecidos são: 8051, 8096, 68HC705, 68HC11 e os PICs (MATIC & ANDRIC, 2001). A Figura 2 apresenta o diagrama em blocos de um típico microcontrolador.

Para Rosh (1996), a fronteira entre as definições de microcontrolador e sistema embarcado não é clara, pois há muitas semelhanças e uma profunda ligação entre eles. A diferença principal está no tamanho e complexidade. Os microcontroladores são simples**,** enquanto que os sistemas embarcados são mais complexos e usam uma grande quantidade de chips e outros dispositivos, como sensores, comutadores e relés.

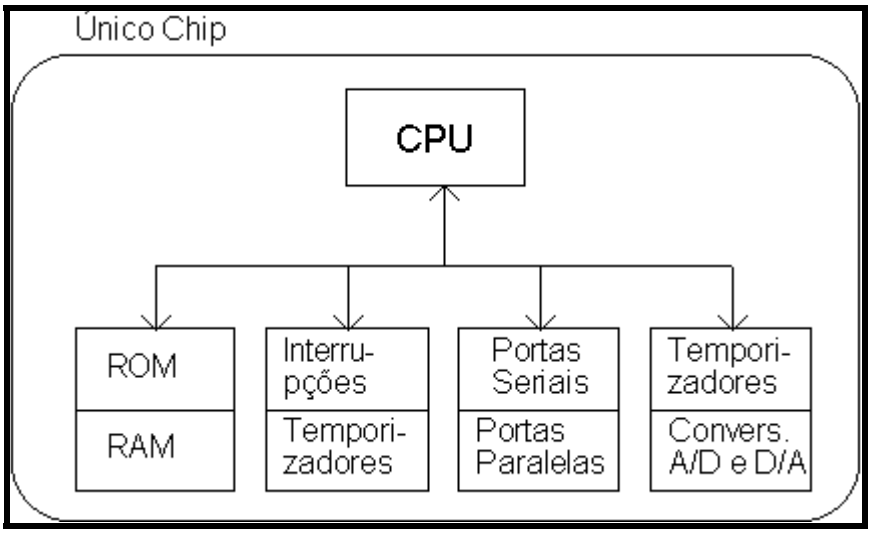

Figura 2. Diagrama de blocos de um típico microcontrolador

Fonte: Adaptada de Matic & Andric (2001)

1.5.1.4. Diferenças entre Microprocessadores e Microcontroladores

Microcontroladores, diferem em muitas formas de um microprocessador. A característica mais importante que os diferenciam é sua funcionalidade. Para que um microprocessador seja usado, outros componentes, como memória e dispositivos de entrada/saída devem ser incorporados para que o sistema esteja completo, ou seja, o microprocessador é dependente de outros elementos para seu funcionamento (MORIMOTO, 2003). Por outro lado, o microcontrolador é projetado para ter todos esses componentes em um único chip. Geralmente não há necessidade de nenhum outro componente externo, pois o microcontrolador já tem todos seus periféricos embutidos. Assim, economiza espaço e tempo de construção de dispositivos.

Em contrapartida, as CPUs dos microcontroladores são, em geral, menos poderosas do que os microprocessadores. Seu conjunto de instruções costuma se limitar às instruções mais simples, sua freqüência de relógio é mais baixa e o espaço de memória endereçável costuma ser bem menor. Vê-se daí que o campo de aplicação dos microcontroladores é diferente daquele dos microprocessadores, e que um sistema que possa ser controlado por um microcontrolador tende a ter menor complexidade e menor custo do que um sistema que exija a capacidade de processamento de um microprocessador.

A programação dos microcontroladores é, em geral, mais simples do que a dos microprocessadores, ao menos no que diz respeito às exigências de conhecimento dos componentes periféricos. Isso acontece porque os periféricos integrados nos microcontroladores são acessados de uma forma padronizada na própria linguagem de programação, dispensando o conhecimento de detalhes externos. Mas não se deve pensar, porém, que isto vá simplificar a tarefa do programador em todos os níveis, pois é necessário que ele conheça bem o *hardware* conectado ao microcontrolador para poder produzir programas que funcionem corretamente. Em contrapartida, os so*ftwares* de programação para microprocessadores, como compiladores e ligadores, são bem mais evoluídos, poupando programadores de conhecer totalmente o funcionamento do seu *hardware* (SILVEIRA, 2003).

Cabe citar ainda uma vantagem particular dos microcontroladores, que possuem memória ROM, que permite a possibilidade de armazenar programas internamente, dificultando sensivelmente a cópia ilícita do código.

1.5.1.5. Microcontrolador INTEL 8051

O 8051, da Intel, é um microcontrolador muito utilizado pelos projetistas, devido a suas excelentes características. O dispositivo em si é um microcontrolador de 8 bits relativamente simples, mas com ampla aplicação. Porém, o mais importante é que não existe somente o CI 8051 mas sim uma família de microcontroladores baseada no mesmo. Entende-se família como sendo um conjunto de dispositivos que compartilham os mesmos elementos básicos, tendo também um mesmo conjunto básico de instruções (GIMENES, 2003). A Tabela 1 descreve a família completa

dos microcontroladores Intel 8051. Na Figura 3, é mostrado o diagrama em blocos desse microcontrolador.

Para Gimenes (2003), o 8051, por ser um microcontrolador de larga utilização e baixo custo possui as seguintes características principais:

- Freqüência de *relógio* de 12 MHz, com algumas versões que alcançam os 40 MHz;
- Até 64 KB de memória de dados externa;
- Memória de 128 bytes de RAM interna;
- Até 64 KB de memória de programa, independente da anterior, e configurável;
- 4 KB internos (ROM no 8051 e EPROM no 8751) e mais 60 kB internos e 64 kB externos;
- 4 portas bidirecionais de entrada e saída, cada uma com 8 bits individualmente endereçáveis; duas dessas portas (P0 e P2) e parte de uma terceira (P3) ficam comprometidas no caso de se utilizar qualquer tipo de memória externa;
- 2 temporizadores contadores de 16 bits;
- 1 canal de comunicação serial;
- 5 fontes de interrupção (dois temporizadores, dois pinos externos e o canal de comunicação serial) com 2 níveis de prioridade selecionáveis por *software*;
- Oscilador de relógio interno.

Tabela 1. Características da família do 8051

Dispositivo	Versão sem ROM	Versão com EPROM	Capacidade da ROM	RAM	Portas de E/S de 8bits	Timers	Canais de DMA	Fontes de Interrup-ção	Modos de Baixo Consumo
8051	8031	-	4K	128	4	2	X	6/5	
8051AH	8031AH	8751AH	4K	128	4	2	X	6/5	
8052AH	8032AH	8752BH	8K	256	4	3		8/6	
80C51BH	80C31BH	87C51	4K	128	4	2		6/5	X
80C52	80C32	-	8K	256	4	3		8/6	X
83C51FA	80C51FA	87C51FA	8K	256	4	3		14/7	X

Fonte: Adaptada de Gimenes (2003)

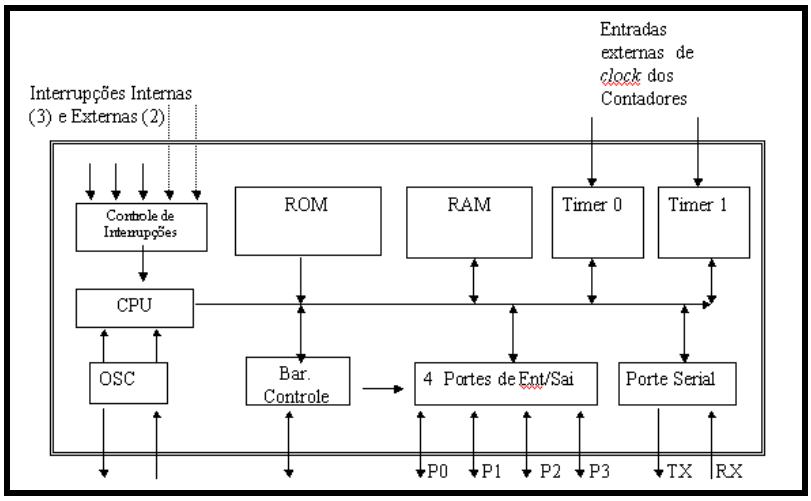

Figura 3. Diagrama de blocos microcontrolador 8051

Outras características, descritas por Silveira (2003), fazem o 8051 um microprocessador de larga aplicação. O mesmo tem dois modos básicos de funcionamento:

- Modo mínimo: onde somente recursos internos são utilizados pela CPU. Nesse modo, estão disponíveis 4 KB de ROM para memória de programa e 128 bytes de RAM para memória de dados. O modo mínimo possui a vantagem (além da economia de componentes e espaço físico) de poder utilizar as 4 portas de 8 bits E/S;

- Modo expandido: Neste modo, a memória de programa (ROM), a memória de dados (RAM) ou ambas podem ser expandidas para 64 KB, através do uso de CIs externos. No entanto, apresenta a desvantagem de "perder" duas das 4 portas para comunicação para as memórias externas.

  A pinagem para o 8051 é mostrada abaixo na Figura 4:

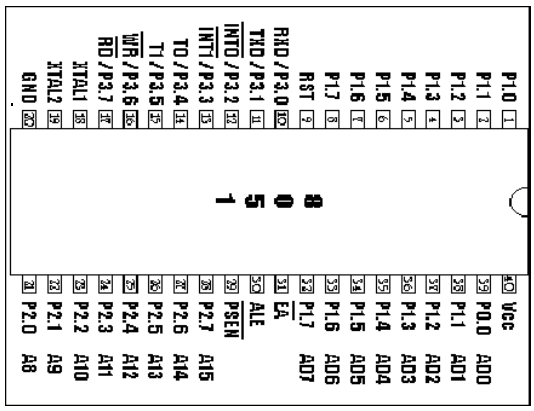

Figura 4. Pinagem do 8051

Fonte: Adaptada de Gimenes (2003)

Para Nicolosi (2003), a organização de memória do microcontrolador 8051 da Intel é dividida em dois tipos: memória de programas e memória de dados.

A memória de programa do 8051 pode ocupar até 64 KB. Aqui há duas opções: se a memória ROM interna do controlador for utilizada, então esta será mapeada nos primeiros 4 KB deste espaço de endereçamento, e os demais 60 KB serão externos. Caso não se deseje utilizar a ROM interna, então toda a memória será externa. A memória de dados é dividida em duas partes: interna e externa.

A memória interna é dividida em três blocos fisicamente distintos, conforme ilustra a Figura 5. Há dois blocos de RAM, e mais cerca de 20 registradores de funções especiais. A distinção entre os dois blocos cujos endereços coincidem é feita pelos modos de endereçamento, também indicados na Figura 5.

A memória de dados externa pode ocupar até 64 KB. Portanto, há coincidências de endereços com toda a faixa de memória de programa e com os endereços 00H a FFH da memória de dados interna. Estes conflitos são resolvidos por dois mecanismos distintos:

A distinção entre memória de programa e memória de dados é feita pelo sinal PSEN (ver Figura 5) que o *hardware* utiliza para habilitar o banco de memória correspondente. Esse sinal é ativado na leitura de instruções da memória na utilização de instruções de movimentação de dados na memória. Outra forma de resolver tais conflitos é a utilização de duas instruções distintas para acessar a memória interna (MOV) e a memória externa (MOVX).

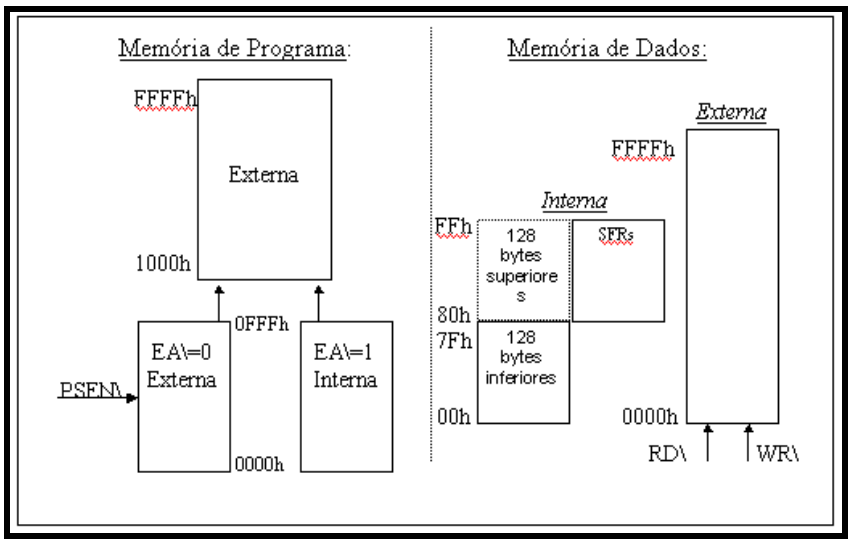

Figura 5. Organização da memória do 8051

1.5.1.6. Microcontrolador PIC

O PIC é um microcontrolador muito requisitado para concepção de projetos de sistemas embarcados. Este chip é produzido pela Microchip *Technology* Inc.. Na Figura 6 ilustra a aparência física desse componente e na Figura 7 a pinagem do PIC16F84.

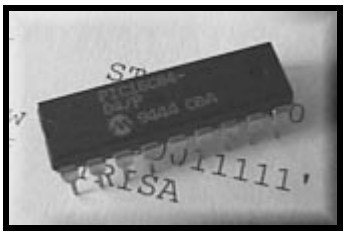

Figura 6. Aparência do PIC

Fonte: Adaptada de Matic & Andric (2001)

Segundo Souza e Lavinia (2003), o PIC possui um microprocessador RISC (*Reduced Instruction Set Computing*) e sua arquitetura é tipo *Harvard,* ou seja, o barramento de endereços e de dados são distintos e independentes, proporcionando maior desempenho sobre uma arquitetura convencional de *Von Neumann*, onde instruções e dados residem na mesma memória e são acessados através do mesmo barramento.

O PIC está disponível em uma ampla gama de modelos para melhor adaptarem as exigências de projetos específicos, diferenciando-se pelo número de linha de Entrada/Saída e pela variedade de dispositivos embutidos. Inicia-se com modelo pequeno identificado pela família PIC12Cxx dotado de oito pinos, até chegar a modelos maiores como a linha PIC17Cxx dotada de 40 pinos (SOUZA E LAVINIA, 2003)

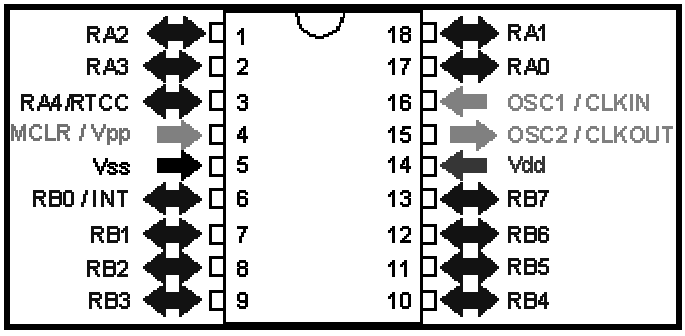

Figura 7. Pinagem do PIC16F84

Fonte: Adaptada de Matic & Andric (2001)

A Figura 8 ilustra o diagrama em blocos da arquitetura interna de um dos microcontroladores da família PIC, mais precisamente do PIC16F84.

Abaixo são descritas as características de cada bloco do desse microcontrolador (MATIC & ANDRIC, 2001):

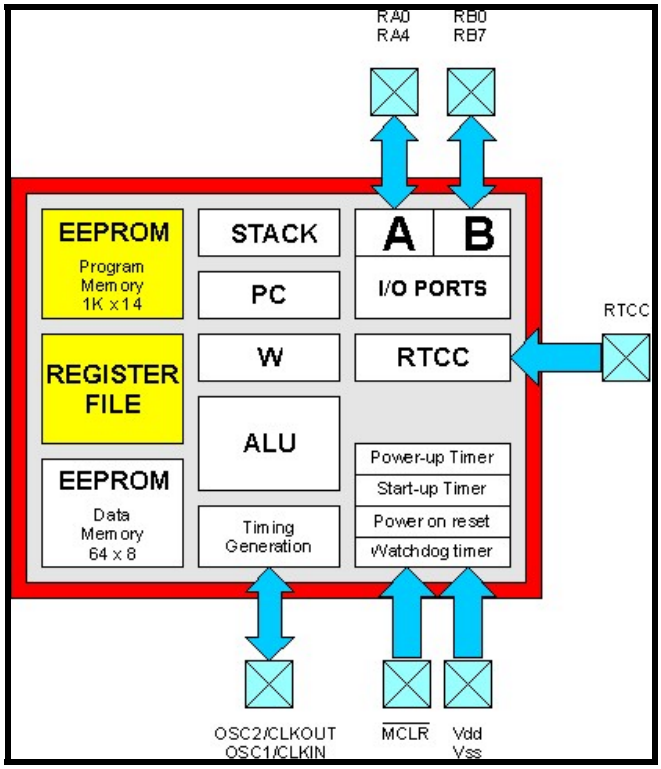

Figura 8. Diagrama em blocos do PIC16F84

Fonte: Adaptada de Matic & Andric (2001)

- Memória EEPROM para Programas: A sua capacidade de memorização é de 1024 e que poderão conter somente um *opcode* a 14 bits, ou seja, uma instrução básica do PIC. Um programa mais complexo não poderá ter mais do que 1024 instruções. Os endereços reservados para essa memória começam nos endereços que abrangem a faixa de 0000H até 03FFH. O PIC pode somente executar instruções memorizadas nestas locações. Não se pode de maneira nenhuma ler, escrever ou cancelar dados nesses endereços, para que isso ocorra é necessário um dispositivo externo chamado programador, que na verdade é um acessório do PIC (PICSTART-16+) que realiza essa tarefa. A primeira locação de memória, o endereço 0000H, deve conter a primeira instrução que o PIC deverá executar após o *reset,* e por isso é denominada *Reset Vector*;

- *Register File*: É uma parte da locação de memória RAM denominada registro. Diferente da memória EEPROM de programa, essa área de memória RAM é diretamente visível pelo resto do programa. A principal característica é que se pode ler, ou modificar tranqüilamente qualquer endereço do *register file* no programa a qualquer momento em que for necessário. A única limitação consiste de que alguns desses registros desenvolvem funções especiais pelo PIC e não pode ser utilizado para outra coisa a não ser para aquilo a qual eles estão reservados. É o caso dos registros e TRISA e TRISB, que servem para definir qual linha de entrada/saída será utilizado. O mesmo estado lógico da linha de entrada/saída depende do valor de dois registradores: PORTA e PORTB. Alguns registros reportarão o estado de funcionamento do dispositivo interno do PIC ou o resultado de operações aritméticas e lógicas. É necessário conhecer, portanto, exatamente qual função desenvolve cada um dos registros especiais e qual efeito se obtém ao manipular seus conteúdos;
- A ULA (Unidade Lógica e Aritmética) é um componente presente em todos os microprocessadores e é responsável pelas operações matemáticas e lógicas do microcontrolador. A ULA no PIC16F84, por exemplo, está preparado para operar com números de 8 bits. A ULA possui um registrador chamado "W" que se utiliza de um acumulador. A diferença entre o registrador W e outras memórias consiste no fato de que, por referenciar esse registrador, a ALU pode acessar diretamente;
- *Stack*: Esse registrador tem a função de guardar os endereços de instruções quando forem solicitadas um desvio de rotina, ou seja, o PIC possui instruções que possam desviar do ciclo normal de um programa seqüencial, assim pode-se implementar sub-rotinas, pois o microcontrolador sabe qual endereço tem que retornar a execução;
- I/O Ports: O PIC16C84 dispõe de um total de 13 linhas de entrada/saída organizadas em duas portas denominadas de PORT A e PORT B. O PORT A dispõe de 5 linhas configuráveis tanto para entrada como para saída identificadas pelas siglas RA0 até RA4. O PORT B dispõe de 8 linhas também configuráveis, identificadas pelas siglas RB0 até RB7. Para o controle da linha de entrada/saída do programa, o PIC dispõe de dois registradores internos que controlam as portas e são chamados de TRISA e TRISB. Os registros TRIS A e B determinarão o funcionamento em entrada ou em saída da mesma linha, e o registro PORT A e B determinarão o status da linha, tanto de entrada como a linha de saída;
- Timer0: O registrador TMR0 é um contador, ou seja, um registro particular no qual seu conteúdo é incrementado com cadência regular e programada diretamente pelo *hardware* do PIC. Na prática, a diferença de outro registro é que o TMR0 não mantém inalterado o valor

que é memorizado, mas o incrementa continuamente até 255, quando esse é zerado novamente. A velocidade de contagem é diretamente proporcional à freqüência de clock aplicada ao *chip* e pode ser modificada programando-se oportunamente os seus bits de configuração.

1.5.2 Memória

1.5.2.1. Introdução

Nesta secção, são descritos os tipos básicos de componentes de memória que são utilizados em sistemas embarcados. Não são abordadas características ligadas aos circuitos eletrônicos que compõem tal dispositivo, mas sua funcionalidade e características voltadas ao desenvolvimento de projetos de sistemas embarcados.

Para Tanenbaum (1999), memória é um termo genérico usado para designar as partes do computador ou dos dispositivos periféricos onde os dados e programas são armazenados. Sem uma memória de onde os processadores podem ler e escrever informações, não haveria nenhum computador digital de programa armazenado. A Figura 9 ilustra uma estrutura básica da memória.

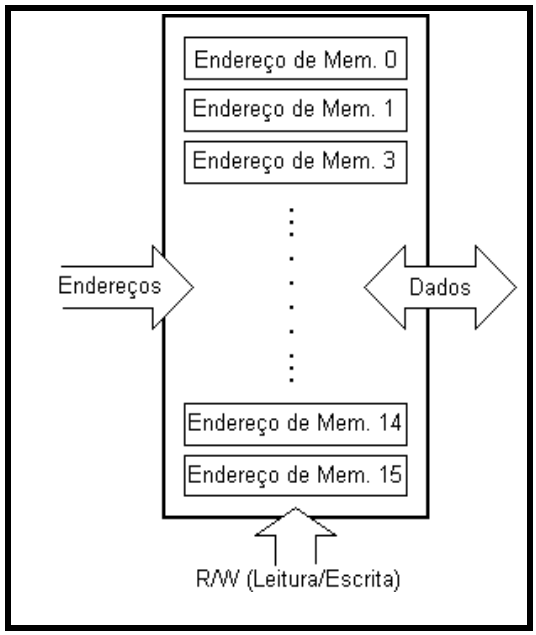

Figura 9. Estrutura básica da memória.
Fonte: Adaptada de Matic & Andric (2003)

Apesar de parecer simples como conceito, a memória exibe, talvez, a mais vasta gama de tipos, tecnologias, organizações, e custos. Nenhuma tecnologia pode ser considerada com total satisfação dos requisitos de memória. Como conseqüência, um sistema embarcado geralmente vem incorporado uma mescla dessas tecnologias.

1.5.2.2. Organização da Memória

Segundo Tanenbaum (1999), um processador só pode identificar a informação através de sua restrita capacidade de distinguir seus dois estados associados ao valor 1, e ao outro estado, o valor 0. Os dígitos 0 e 1 são os únicos elementos do sistema de numeração de base 2, sendo então chamados de dígitos binários, ou abreviadamente, bit. Entenda-se por bit a unidade básica de memória, ou seja, a menor unidade de informação que pode ser armazenada num computador.

Como o valor de um bit tem pouco significado, as memórias são estruturadas e divididas em conjuntos ordenados de bit, denominados células, cada uma podendo armazenar uma parte da informação. Se uma célula consiste em k bits ela pode conter uma em 2^k diferentes combinações de bits, sendo que todas as células possuem a mesma quantidade de bits.

Cada célula deve ficar num local certo e conhecido, ou seja, a cada célula associa-se um número chamado de seu endereço. Só assim torna-se possível a busca na memória exatamente do que se estiver querendo a cada momento (acesso aleatório). Sendo assim, a célula pode ser definida como a menor parte de memória endereçável (VELLOSO, 1994).

A maioria dos fabricantes de computadores padronizaram o tamanho da célula em 8 bits (Byte). Os bytes são agrupados em palavras, e a um grupo de bytes (2,4,6,8 Bytes) é associado um endereço particular.

Os bytes em uma palavra podem ser numerados da esquerda para direita ou da direita para esquerda. O primeiro sistema, onde a numeração começa no lado de alta ordem, é chamado de *big endian*, e o outro de *little endian* ilustrada na Figura 10.

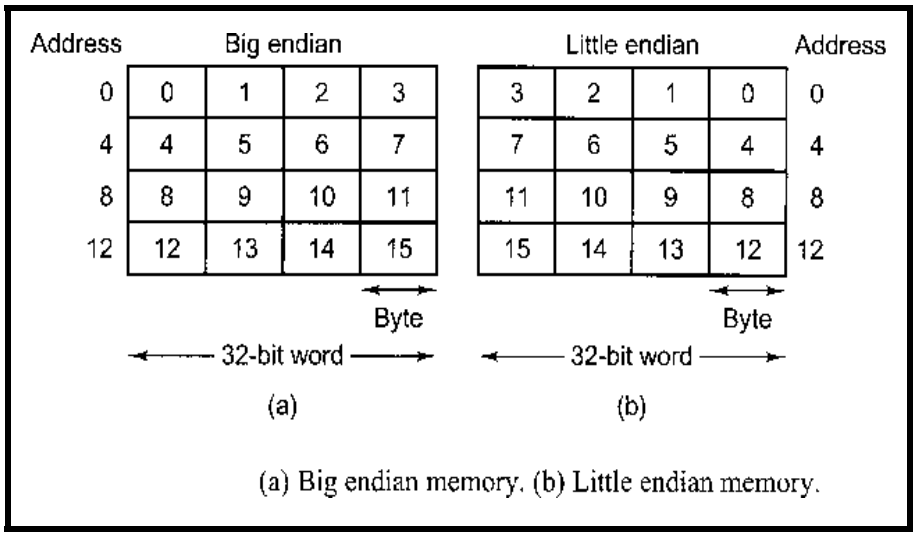

Figura 10. Diferenças na representação *Big edian* e *Little edian*

1.5.2.3. Tipos de Memória

A memória permite a realização de dois tipos de acesso: escrita e leitura. Entende-se por leitura a recuperação da informação armazenada e a escrita é a gravação (ou armazenamento) da informação na memória. Essas operações (leitura e escrita) são realizadas pela CPU e efetuada por células, não sendo possível trabalhar com parte dela.

De acordo com Toledo (1994), nem todas as memórias existentes possuem a característica de escrita. Algumas foram projetadas e gravadas uma única vez, pelo fabricante com um intuito exclusivo. Ex: Memória que comporta o programa da BIOS (*Basic Input Output System*). A seguir serão apresentados algumas tecnologias e tipos de memórias mais conhecidos em sistemas embarcados:

- Memória RAM (*Random Access Memory*). É um tipo de memória essencial para o computador, sendo usada para guardar dados e instruções de um programa. Tem como características fundamentais, a volatilidade, ou seja, o seu conteúdo é perdido quando o computador é desligado; o acesso aleatório aos dados e o suporte à leitura e gravação de dados, sendo o processo de gravação um processo destrutivo e a leitura um processo não destrutivo;

- Memória ROM (*Read Only Memory)*. É um tipo de memória que serve apenas para operações de leitura. Ela é não-volátil, ou seja, os dados não são perdidos com a ausência de energia. É também de acesso aleatório;
- EPROM (*Erase Programmable Read Only Memory*). É um tipo de ROM especial que pode ser programada pelo usuário. Todo seu conteúdo pode ser apagado pela exposição a raios ultravioletas;
- EEPROM (*Electrical Erasable Programmable Read Only Memory*). É também um tipo especial de ROM muito semelhante a EPROM, tendo como diferença apenas o fato de que seus endereços são apagados individualmente aplicando-se uma voltagem específica em um dos seus pinos de entrada;

1.5.2.4. Tecnologias de Memória

Conforme Morimoto (2003), as memórias não tem acompanhado o desenvolvimento dos processadores, mas elas também contribuíram com sua parcela de desenvolvimento. Desde as primeiras memórias do início da década de 80, até as memórias produzidas atualmente, é usada a mesma estrutura básica formada por um capacitor e um transistor para cada bit de dados. Porém, foram realizadas melhorias na forma de organização física e na forma de acesso, que permitiram melhorara consideravelmente a velocidade de acesso. Abaixo, algumas tecnologias empregadas na fabricação de memórias:

- Memórias Regulares: ou simplesmente memórias "comuns", foram o primeiro tipo de memória usada. Neste tipo antigo de memória, o acesso é feito enviando primeiro o endereço RAS (*Row Address Strobe*) indicado a linha e em seguida o endereço CAS (*Column Address Strobe*) para acessar coluna, da forma mais simples possível;
- Memórias FPM (*Fast Page Mode*): A idéia desta tecnologia é que na maioria das vezes as informações são gravadas em endereços de forma seqüencial, ou seja, são armazenados numa seqüência de colunas ou linhas continuas. Não seria então preciso enviar o endereço RAS e CAS para cada bit lido, mas simplesmente enviar o endereço RAS (linha) um vez e em seguida enviar vários endereços CAS (coluna). Quanto mais baixos forem os tempos de espera, mais rápidas serão as memórias;
- Memórias EDO (*Extended Data Output*): Além de ser mantido o "modo de acesso rápido" das memórias FPM, foram feitas algumas modificações para permitir mais um pequeno

truque, através do qual um acesso a dados pode ser iniciado antes que o anterior termine, permitindo aumentar perceptivelmente a velocidade dos acessos.

- Memórias *SDRAM* (*Synchronous Dynamic RAM*): Tando as memórias FPM quanto as memórias EDO são assíncronas. Isto significa que elas trabalham em seu próprio ritmo, independente do relógio do controlador de memória. As memórias SDRAM, por sua vez, são capazes de trabalhar sincronizadas com esse *clock* externo, sem tempo de espera sempre de uma leitura por ciclo.

1.5.3 Dispositivos Periféricos

1.5.3.1. Porta Paralela

Para Rosh (1996), portas paralelas são conexões bem definidas, convenientes e rápidas. Outrora pertenciam exclusivamente para conectar impressoras e atualmente, cada vez mais periféricos tiram proveito dessa conexão paralela rápida e segura. Paralela refere-se ao fato de que conduz os sinais por meio de oito fios separados - um para cada bit de um byte de dados - e dentro de um único cabo.

Teoricamente, oito fios significam que se pode transferir dados oito vezes mais rápido por meio de uma conexão paralela do que por meio de um único fio. As portas paralelas são intrinsecamente simples, pois transferem e recebem dados byte a byte (ibidem).

A velocidade operacional máxima de uma porta paralela é determinada por diversos fatores:

- O cabo em si define o limite superior das freqüências que podem ser usadas para os sinais.
- Quando o sincronismo do sistema é definido para gerar os tamanhos mínimos desses sinais, um ciclo de transmissão de *caracter* completo exige cerca de 10 microssegundos, suficientes para mover 100.000 bytes por segundo, ou seja, 800.000 bits por segundo.
- Microprocessador ou controlador. Quanto mais veloz, mais cedo terminará o trabalho adicional exigido.

- Como o uso de portas paralelas com controle de dados, as transferências podem ser feitas pelo controlador de DMA (*direct memory Access*) sem intervenção do microprocessador. Com um sistema veloz de DMA, o tempo de resposta deste dispositivo pode alcançar o limite da velocidade oferecida.

1.5.3.2. Porta Serial

A porta serial é muito utilizada para comunicações entre dispositivos. Até mesmo os periféricos mais primitivos aceitam uma porta serial.

A porta serial é muito usada entre diversos componentes e equipamentos. E tem uma concepção simples: uma linha para enviar dado, outra para receber dados, algumas outras para regular como os dados são enviados pelas duas linhas. Por sua simplicidade, a porta serial tem sido usada para fazer comunicação com qualquer dispositivo imaginável. Mesmo sendo lenta quando comparada à porta paralela, uma porta serial é muito utilizada para sistemas embarcados, onde a velocidade de comunicação não é necessária (ROSH ,1996).

Estabelecer comunicações seriais confiáveis significa contornar esses problemas de erro de bit e muitos outros mais. Dois dos principais métodos de comunicação serial são usados para evitar erros em bits seriais. Em um deles, os sistemas de envio e recepção são sincronizados por meio de algum tipo de sinal auxiliar, de modo que os dois lados da conexão estejam sempre alerta. Um relógio, sincronizado entre a unidade de transmissão e recepção, temporiza com precisão o período que separa cada bit de dados. Essa forma de transferência serial sincronizada pelo tempo chama-se comunicação síncrona, sendo uma técnica usada principalmente em sistemas de grande porte.

O sistema sincronizado falha sempre que os sistemas de envio e recepção perdem a sua ligação. O fluxo de dados torna-se pouco mais do que um ruído. A alternativa é incluir marcadores de lugar no fluxo de bits para ajudar acompanhar cada bit. Um marcador poderia, por exemplo, indicar a posição atribuída a um bit. Um bit ocorrendo sem o seu marcador poderia ser considerado como erro. Naturalmente, esse esquema simples teria muito desperdício, exigindo dois sinais digitais (o marcador e o bit de dados) para cada bit de informação transferido.

Um sistema intermediário funciona melhor. Em vez de indicar cada bit, o marcador poderia indicar o inicio de um pequeno fluxo de bits. A posição de cada bit no fluxo poderia ser definida sincronizando-os a intervalos regulares. A chegada de um marcador indica ao sistema receptor para começar a procurar os bits e rodar um temporizador curto. Esse sistema de curto prazo temporizado normalmente é chamado comunicação assíncrona, pois os sistemas de envio e recepção não precisam estar sincronizados exatamente um com o outro. Na maior parte dos sistemas assíncronos, os dados são divididos em pequenos pedaços, cada um correspondendo aproximadamente a um byte.

Para Morimoto (2003), paridade é o método de controle de erros mais usado na comunicação serial, oferecendo um meio de detectar erros de transmissão ao nível de bit. Essa detecção de erros funciona contando o número de bits na palavra de dados e determinando se o resultado é par ou ímpar. Na paridade ímpar, o bit de paridade é ativado (passado para o nível lógico um) quando o número de bits da palavra é impar. A paridade par ativa o bit de paridade quando o total de bits de uma palavra é par. Embora emitindo um bit de integridade de dados (que pode ser fornecido por outros meios), isso permite comunicações mais eficientes, comprimindo mais informações em um determinado número de bits transmitidos.

Por ter o modo de funcionamento distinto das portas paralelas, a velocidade mínima comum é 300 bps, embora estejam disponíveis submúltiplos mais lentos: 50, 100 e 150 bps. As velocidades mais rápidas simplesmente duplicam as velocidades anteriores, passando para 600, 1200, 4800, 9600 e 19200. Essas velocidades oficiais lentas são importantes porque o controle da porta serial por *software* impõe uma carga tão grande sobre o microprocessador do sistema que os chips mais lentos não podem trabalhar com velocidades maiores (ibidem).

1.5.3.3. Motores de Passo

Motor de passo é um transdutor que possibilita a conversão de energia elétrica em mecânica. Um motor DC, quando é alimentado, gira sempre no mesmo sentido e com rotação constante, ou seja, para que estes motores funcionem, é necessário apenas estabelecer sua alimentação. Para que um motor de passo funcione, é necessário que sua alimentação seja feita de forma seqüencial e repetida. Com o auxílio de circuitos externos de controle, estes motores de corrente contínua poderão inverter o sentido de rotação ou variar sua velocidade (DORM, 2003).

O motor de passo utiliza combinações de pulsos digitais que determinam o ângulo de deslocamento do rotor. A cada pulso, ele faz um incremento rotativo (passo). Cada passo é só uma porção de uma rotação completa. Então, vários pulsos podem ser aplicados para alcançar uma quantia desejada de rotação do eixo. A Figura 11 ilustra melhor o conceito de motor de passo.

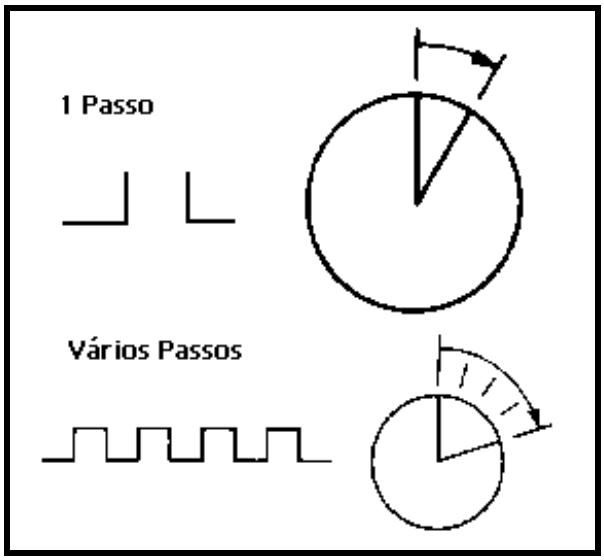

Figura 11. Demonstração dos passos do motor

Os motores DC giram em velocidade constante, pois possuem apenas dois estágios de operação, ou seja, parados ou girando (mesmo que em velocidades diferentes). Enquanto os motores de passo deslocam-se por impulsos ou passos discretos onde exibem três estágios: parados, ativados com rotor travado (bobinas energizadas) ou girando em etapas. Este movimento pode ser brusco ou suave, dependendo da freqüência, amplitude e principalmente da precisão dos passos em relação à inércia em que ele se encontre (COSTA, 2003).

Dorm (2003) também especifica que a precisão de um motor de passo é principalmente determinada pelo número de passos por rotação (quanto maior for a quantidade de passos, maior será a precisão). Para uma precisão mais alta, alguns controladores de motor de passo dividem passos completos em meio-passos ou micro passos (Figura 12).

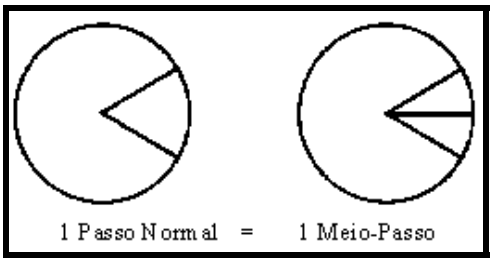

Figura 12. Precisão de passos

Para Meneses (2003), o sistema de controle se baseia em um circuito oscilador onde é gerado um sinal, cuja freqüência está diretamente relacionada com a velocidade de rotação do motor de passo. Essa freqüência é facilmente alterada (seja por atuação em componentes passivos, ou seja, por meio eletrônico) dentro de um determinado valor. Assim, o motor apresenta uma rotação mínima e uma máxima. A função "Freio" dá-se simplesmente pela inibição do sinal gerado pelo oscilador. O sinal do oscilador é injetado em *flip-flops* que fariam a conversão para sinal de caráter binário.

Para Dorm (2003) os motores de passo, por serem dispositivos eletro-mecânicos, necessitam de um consumo de energia relativamente alto comparado aos outros componentes de sistemas embarcados. Para que esse dispositivo possa ser implementado é necessário um circuito amplificador de saída pois algumas aplicações exigem uma demanda de corrente relativamente elevada. Cabe ao circuito amplificador de saída fornecer essas correntes de forma segura, econômica e rápida. Esse circuito gera uma corrente em torno de 500 mA ou mais. Motores de passo geralmente suportam correntes acima de 1,5 Ampére. O amplificador de saída é o dispositivo mais solicitado em um projeto de controle de motor de passo. Devido as variações de trabalho a que pode ser submetido o motor de passo, um amplificador mal projetado pode limitar muito o conjunto como um todo. Um exemplo destas limitações pode ser facilmente entendido: Um motor de passo girando à altas rotações, repentinamente é solicitado à inverter sua rotação. No momento da inversão as correntes envolvidas são muito altas e o circuito amplificador deve suportar tais drenagens de corrente.

1.5.3.4. Conversores A/D e D/A

De acordo com Matic & Andric (2001), freqüentemente, é necessário converter um sinal analógico em um sinal digital proporcional à sua amplitude, e vice versa. Isso é essencial em qualquer aplicação onde um sistema controla um processo ou onde técnicas digitais são utilizadas para executar um trabalho analógico.

As técnicas de conversão são essenciais para a geração de mostradores analógicos em instrumentos digitais, a exemplo disso são indicadores de medidas criados pôr um sistema digital.

1.5.3.4.1. Conversores D/A

Um conversor digital/analógico transforma a informação digital em uma diferença de pontencial analógica. A informação digital se apresenta na forma de um número binário com um número fixo de dígitos. Especialmente, quando usado em sistemas digitais, este número binário é chamado de palavra binária. Um conversor digital/analógico converte uma palavra digital em uma tensão analógica, de maneira que a da saída analógica é zero quando todos os bits são zero e algum valor máximo quando todos os bits são um, ou de maneira proporcional aos valores obtidos com a palavra digital. Na Figura 13 é mostrado o bloco simplificado de um dispositivo D/A.

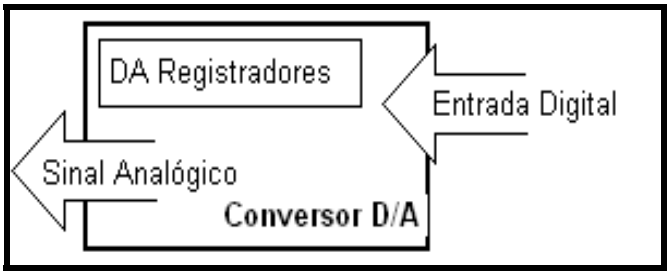

Figura 13. Conversor D/A

Fonte: Adaptada de Matic & Andric (2001)

Em aplicações modernas, a maior parte dos conversores D/A são disponíveis na forma de circuitos integrados, de maneira que, o que importa do ponto de vista são algumas característica de entrada e saída descritas abaixo:

- Entrada digital: Tipicamente, uma palavra binária paralela com um certo número de bits especificado pelo "*data sheet*" do dispositivo. Normalmente, os níveis lógicos são TTL (*Transistor Transistor Logic)*, a menos que especificados de outra forma;
- Fonte alimentação: Esta é bipolar na faixa de 12 a 18V necessário para os amplificadores internos. Alguns conversores D/A operam com uma fonte simples;
- Fonte de referência: É necessária para estabelecer o intervalo da voltagem de saída e a resolução do conversor. Esta deve ser estável e não ruidosa. Em alguns conversores é interna;
- Saída: Uma voltagem analógica representando a entrada digital. Esta voltagem muda em degrau quando a entrada digital muda seus bits. A saída pode ser bipolar caso o conversor seja projetado para interpretar sinais digitais negativos;
- *Offset:* Os conversores são projetados com amplificadores internos que podem produzir voltagem de *offsets* (zero residual) quando a entrada digital é zero. Pinos adicionais devem estar presentes no dispositivo para facilitar o ajuste de zero;
- *Data lacht:* Muitos conversores D/A têm um pino de controle do fluxo de dados de entrada. Quando um comando é dado através do "*data latch*", os dados na entrada do conversor são armazenados e a saída do conversor mostra o sinal analógico correspondente. A saída permanecerá com este valor até que outro sinal de entrada seja armazenado. Isso permite que um conversor possa ser conectado diretamente a um barramento de dados de um microprocessador que controla os dados.

1.5.3.4.2. Conversores A/D

Apesar de muitos sensores fornecerem sinais que podem ser lidos diretamente pelo pelos sistemas digitais, há uma necessidade de converter sinais analógicos em sinais digitais. Em aplicações modernas, assim como nos conversores D/A, a maior parte dos conversores A/D são disponíveis na forma de circuitos integrados. Na Figura 14 é mostrada uma forma simplificada em blocos desse dispositivo. Algumas características de suas entradas e saídas estão relacionadas abaixo:

- Entrada analógica: O sinal de entrada dos conversores normalmente é uma voltagem alternada, ou seja, varia de acordo com um determinado limite de tempo. A faixa de tensão, comumente é limitada pela fonte de alimentação**.**

- Sinais de controle digitais: Os sinais de controle digitais possuem níveis TTL, e possibilitam o interfaceamento direto do conversor com microprocessadores.
- Fonte de referência: É necessária para estabelecer o intervalo da tensão de saída e a resolução do conversor. Esta deve ser estável, e não ruidosa. Em alguns conversores, é interna.
- Saídas digitais: Saídas dos bits do conversor. Normalmente são níveis compatíveis TTL. Em conversores ultra-rápidos os níveis são ECL (Lógica de Emissor Acoplada).
- Fonte de Alimentação: Esta é bipolar na faixa de 12 a 18V necessário para os amplificadores internos. Alguns conversores A/D operam com uma fonte simples.

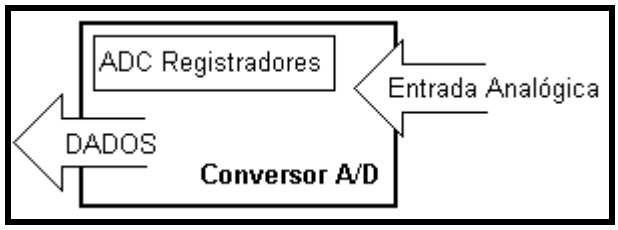

Figura 14. Conversor A/D

Fonte: Adaptada de Matic & Andric (2001)

1.5.3.5. LED

LED (*Light Emitting Diode*) é uma junção PN de um diodo que pode emitir luz devido a recombinações de elétrons e buracos entre eles. Quando uma tensão é aplicada, elétrons são lançados para a região de depleção, onde eles se recombinam. Nesse ponto são emitidos fótons, a unidade básica da luz. O comprimento de onda do fóton liberado corresponde ao nível de energia associado. A cor da luz emitida depende do material semicondutor usado na dopagem ou ao nível de energia associado (BARBACENA, 2003).

Um dos usos mais comuns do LED é com uma rápida fonte luminosa, de comprimento de onda conhecido, para indicar os estados dos circuitos eletrônicos. Outro uso típico é como indicador do estado lógico de uma porta digital. Se a tensão for alta (maior que 2.4 V), o LED emite luz com toda intensidade, se a tensão for baixa (menor que 0.8V), o LED não emite luz nenhuma (MORIMOTO, 2003). Na Figura 15 ilustra um de seus aspectos físicos:

Figura 15. Aspectos físicos mais comuns do LED.
Fonte: Morimoto (2003)

1.5.3.6. Displays LCD

De acordo com Barbacena (2003), os LCDs (*Liquid Crystal Displays*) são utilizados em muitos projetos de eletrônica para facilitar a interação do usuário com o projeto. Com o LCD pode-se mostrar informações relevantes para a utilização do projeto, assim como retornar ao usuário resultados obtidos em algum tipo de processamento. A Figura 16 ilustra o aspecto físico desse componente.

Os módulos LCD são *interfaces* de saída muito úteis em sistemas microprocessados. Esses módulos podem ser gráficos ou alfanuméricos. Os LCDs gráficos são encontrados com resoluções de 122x32, 128x64 e 240x128 pixels (entre outras resoluções) e geralmente estão disponíveis com 20 pinos para conexão. Os LCDs comuns são especificados em número de linhas e colunas e são encontrados nas configurações previstas na Tabela 2.

Tabela 2. Configurações de LCDs de Texto.

Número de Colunas	Número de Linhas	Quantidade de Pinos
8	2	14
12	2	14/15
16	1	14/16
16	2	14/16
16	4	14/16
20	1	14/16
20	2	14/16
20	4	14/16
24	2	14/16
24	4	14/16
40	2	16
40	4	16

Fonte: Barbacena (2003)

Os LCD´s também possuem junto ao seu dispositivo alguns LED´s (Diodo Emissor de Luz) que iluminam o painel facilitando as leituras durante a noite.

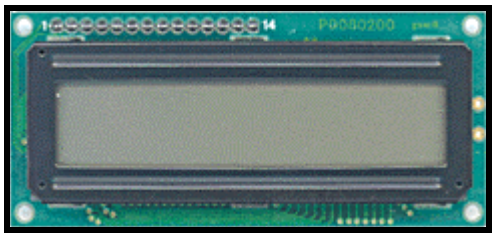

Figura 16. Aspecto físico de um LCD.

Esses módulos utilizam um controlador próprio, permitindo sua interligação com outras placas através de seus pinos, nos quais deve ser alimentado o módulo e interligado o barramento de dados e controle do módulo com a placa do usuário. Naturalmente, além de alimentar e conectar os pinos do módulo com a placa do usuário, deverá haver um protocolo de comunicação entre as partes, o que envolve o envio de bytes de instruções e bytes de dados.

1.5.3.7. Display de Sete Segmentos

O *display* de sete segmentos é o tipo de *display* mais comumente encontrado em eletrodomésticos e aparelhos eletrônicos de baixo custo (MORIMOTO, 2003). Sua estrutura baseia-se geralmente de um dispositivo que comporta sete LEDs arranjados de uma forma que possibilita a exibição de um número. A Figura 17 apresenta um exemplo de aparência do dispositivo e da distribuição dos sete segmentos (LEDs):

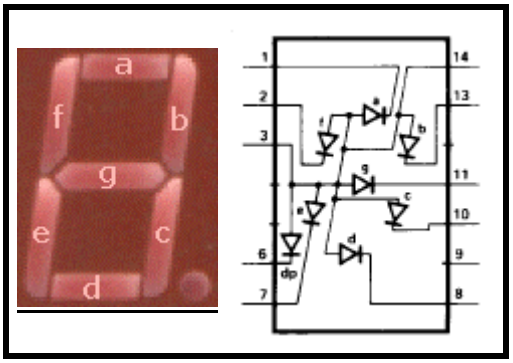

Figura 17. Aspecto físico de um *Display* de 7 Segmentos

Para que um número seja exibido corretamente, geralmente é necessário um circuito decodificador para que os segmentos de *displays* que atue de modo que cada código BCD (Binário Convertido para Decimal) de entrada, ative um grupo de segmentos, formando o caractere correspondente. Na Figura 18, é mostrado um exemplo de um circuito decodificador (BCD 8421) e sua conexão com o *display* de sete segmentos:

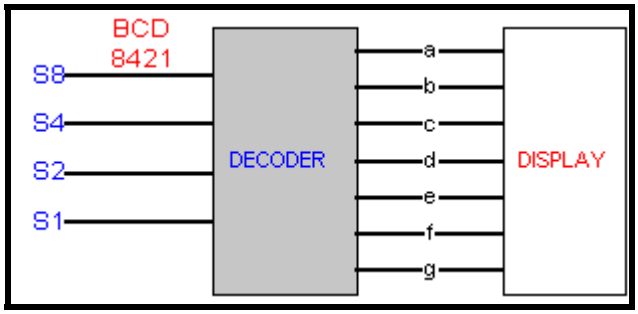

Figura 18. Ligação de um decodificador com o *display*.

1.5.4. Redes-em-Chip

1.5.4.1 Introdução

Os Sistemas de circuitos integrados estão cada vez mais avançando para um estado crítico no que diz a respeito da implementação de diversos circuitos interligados em uma única pastilha. Isso significa que, nos próximos anos, a arquitetura empregada hoje em dia para alojar e integrar circuitos num determinado chip não vai ser suficiente para suportar os avanços tecnológicos, principalmente na área da nanoeletrônica onde consegue cada vez mais a integração de milhares de transistores.

Para Karim e Dey (2002), essa tecnologia que permite a ligação entre vários dispositivos, como microprocessadores, memórias entre outros circuitos através de meios de comunicações específicos são denominados de Sistemas Integrados ou SoCs (*Systems on Chips*).

O avanço tecnológico dessa capacidade de integração resulta em um avanço significativo do suporte para comunicação entre esses circuitos. No entanto, na medida que aumenta a complexibilidade e a quantidade dos circuitos que são inseridos na pastilha, mais elaborada tem que

ser a arquitetura de comunicação, ocasionando problemas de custo e desempenho para os projetos de SoCs.

Zeferino e Susin (2003) destacam que para obter menor custo com projetos de sistemas integrados, é desejável que tais componentes sejam reutilizáveis, portanto, esses componentes chamados de núcleos (*cores*), são circuitos que já foram projetados e testados, sendo aproveitados para outros fins.

O principal problema que impede a reusabilidade é o meio de comunicação entre os núcleos. Alguns requisitos como o desempenho, a possibilidade de paralelismo em comunicação, não são plenamente suportados pela arquitetura atual.

Dispositivos de comunicação ponto-a-ponto são considerados rápidos, pois o tamanho do fio utilizado para ligar os núcleos são projetados sob medida. Por outro lado, essa forma de comunicação é a menos reusável de todas, devido o fato que, para cada projeto, um novo conjunto de comunicação deve ser analisado. Quando o número de núcleos tem de se comunicar em maior quantidade, o projeto da comunicação do sistema tende a maior dificuldade (DUTTA, JENSEN & RIECKMANN, 2001).

Arquitetura de barramentos ou multiponto são reusáveis, mas permitem apenas uma transação por ciclo, o que serializa completamente a comunicação. Como todos os núcleos do sistema estarão conectados ao mesmo barramento, e este deve passar por todo o circuito integrado, a capacitância de carga será elevada devido ao comprimento do fio, limitando excessivamente a máxima freqüência que se poderia obter com a comunicação. Esse modelo de comunicação é adequado a um processador central com vários periféricos, não a um modelo com vários processadores complexos executando diferentes tarefas (ibidem).

Para que se possam sanar os problemas de comunicação das arquiteturas acima mencionadas, está sendo proposta a utilização de redes de interconexão chaveada, semelhantes a computadores multiprocessados paralelos. Zeferino e Susin (2003) destacam as seguintes vantagens: largura de banda escalável, a utilização de conexões ponto-a-ponto curtas, o grau de paralelismo na comunicação maior, entre outras. Os mesmos autores também destacam algumas desvantagens como maiores custos e latência de comunicação entre os núcleos, fatores esses amenizados com a implementação de um suporte a essa arquitetura no futuro.

Essa arquitetura de redes chaveadas é denominada Redes-em-Chip (*Networks-on-Chips*), ou simplesmente de NoCs.

O maior compromisso no uso de uma NoC é o aumento da reutilização de componentes para melhor concepção de projeto de sistemas embarcados. A comunicação desses sistemas é implementada atualmente usando um barramento, o que não permite explorar o paralelismo na troca de mensagens entre seus componentes.

1.5.4.2 Conceitos de Redes-em-Chip

Para Karim e Dey (2002), uma NoC é composta por um conjunto de canais e de roteadores, os quais podem ser vistos como núcleos do SoC dedicados à comunicação. Cada roteador está conectado a um ou mais núcleos, e possui um conjunto de canais para se comunicar com outros roteadores. Geralmente a comunicação é feita através de troca de mensagens baseado no conceito de requisição/resposta (ZEFERINO & SUSIN, 2003). A mensagem é formada por três partes distintas: o cabeçalho, o qual indica o início da mensagem além de outras informações necessárias para seu tráfego pela rede; a área de conteúdo da mensagem propriamente dita; e o terminador, o qual sinaliza o final da mensagem. A Figura 19 ilustra o formato da mensagem de uma NoC.

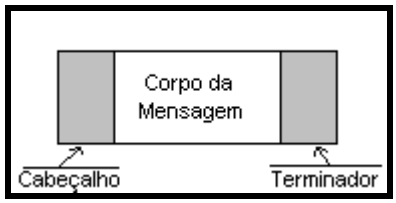

Figura 19. Formato da mensagem de uma NoC.

1.5.4.3 Arquitetura de Redes-em-Chip

Uma rede-em-chip é principalmente caracterizada pela sua topologia, ou seja, a forma em que os componentes estão organizados e interligados e pelos mecanismos de comunicação utilizados. A Figura 20 ilustra uma topologia bastante comum em redes-em-chip.

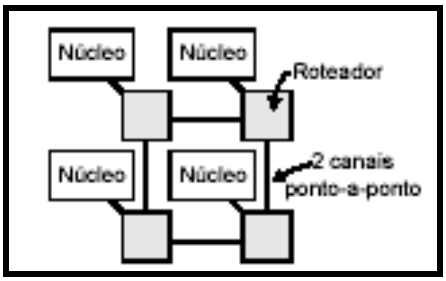

Figura 20. Topologia de uma NoC.
Adaptada de Zeferino e Susin (2003)

Uma rede-em-chip é também caracterizada pelos mecanismos de comunicação que definem a forma como as mensagens são transferidas pela rede e os principais mecanismos descritos por Duato (1997) são:

- Controle de fluxo de mensagens: encarrega-se com a alocação dos recursos para que uma mensagem possa avançar pela rede, regulando o tráfego nos canais;
- Roteamento: define o caminho a ser utilizado por uma mensagem para atingir o seu destino;
- Arbitragem: resolve conflitos internos na rede, quando duas ou mais mensagens competem por um mesmo recurso;
- Chaveamento: define a forma como uma mensagem é transferida da entrada de um roteador para um de seus canais de saída; e
- Memorização: determina qual o esquema de filas vai ser utilizado para armazenar uma mensagem que esteja bloqueada na rede.

A mensagem percorre um número necessário de roteadores entre o núcleo origem e o núcleo destino, sendo que cada roteador gasta uma certa fatia do tempo para processar a mensagem e decidir o caminho a ser usado. Quanto mais roteadores uma mensagem tem que passar para chegar a outro núcleo, maior o tempo de comunicação envolvido. Essa desvantagem é compensada por outros fatores:

- Uma NoC pode transmitir uma mensagem numa freqüência maior, pois a capacitância parasita das conexões é menor que a do barramento. Como as ligações entre roteadores são tipo ponto-a-ponto, a inclusão de mais núcleos não aumenta a carga capacitiva do fio, e, portanto, a freqüência de transmissão da informação em um fio é maior (ZEFERINO & SUSIN, 2003);

- Permite um paralelismo não presente no barramento. Caso um roteador esteja ocupado, outros roteadores podem estar transmitindo várias mensagens em paralelo;
- É facilmente escalável, basta colocar-se mais roteadores.

1.5.4.4 Canais e Roteadores de Redes-em-Chip

Segundo Zeferino e Susin (2003), os roteadores e núcleos conectados numa NoC, são interligados por meio de canais ponto-a-ponto unidirecionais e assíncronos. Esses canais são constituídos por um conjunto de fios que transportam dados de um lado para outro. Quando esses dados são maiores que o tamanho do canal, elas devem ser fragmentadas (quebradas) e transferidas em pedaços. Desde que um canal seja unidirecional, ele conecta um emissor a um receptor. Como, em geral, em um sistema integrado, cada núcleo precisa enviar e receber mensagens, cada par de componentes conectados na rede (núcleo-roteador ou roteador-roteador) requer dois canais de comunicação unidirecionais, um para cada direção, os quais constituem o que se denomina de enlace ou *link* (ZEFERINO & SUSIN, 2003).

O roteador tem como função encaminhar mensagens transferidas pela rede, sendo constituído por um conjunto de filas (*buffers*) e multiplexadores (chaves). Cada componente pode ser construído de maneira centralizada ou distribuída, sendo que, em geral, a primeira abordagem oferece uma maior taxa de utilização de recursos e a segunda propicia a construção de circuitos mais simples (ibidem).

1.5.4.5 Problemas em Redes-em-Chip

As mensagens são transportadas por uma rede-em-chip através de seus canais físicos e roteadores. A comunicação é realizada com sucesso quando a informação de origem chega ao seu destino. No entanto, existem três situações que podem impedir o êxito na comunicação:

- *Starvation*: Quando um canal é muito requisitado devido ao alto tráfego de mensagens, o mecanismo de arbitragem determina qual mensagem vai seguir em diante através da aplicação de um critério por prioridades. Dependendo de como esse critério é aplicado numa situação de alto tráfego na rede, uma mensagem pode nunca ser escolhida a utilizar o canal, vindo a sofrer *starvation*.

- *Livelock*: Quando o mecanismo de roteamento seleciona um canal de saída que afaste a mensagem do seu destino e ela nunca chegue ao mesmo é chamado de *livelock*.

- *Deadlock*: Além de impedir que uma mensagem chegue ao seu destinatário, o roteamento pode levar a paralisação da rede. O *deadlock* ocorre quando há uma dependência cíclica na rede em que cada mensagem garante a alocação de um canal e requer o uso de outro canal já alocado a outra mensagem. A Figura 21 ilustra o fenômeno da dependência cíclica.

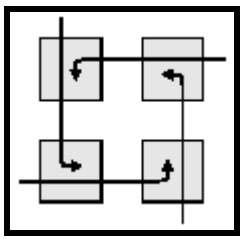

Figura 21. Dependência cíclida.
Fonte: Adaptada de Zeferino & Susin (2003)

Geralmente para garantir que esses problemas acima mencionados não ocorram, é necessário escolher mecanismos de roteamento e arbitragem que não conduzam ao *stravation*, ao *livelock* e ao *deadlock*. O *starvation* é evitado por mecanismos de arbitragem com os de prioridade rotativa, enquanto que a ausência de *livelock* e *deadlock* depende exclusivamente do algoritmo de roteamento adotado.

2. SIMULAÇÃO DE SISTEMAS

Depois de expor os conceitos mais importantes de sistemas embarcados e as principais características de cada componente que, geralmente forma esse tipo de sistema, neste capítulo é discutida a simulação de sistemas. São apresentados alguns conceitos referentes à simulação, vantagens e desvantagens, aplicações e limitações impostas. São discutidos assuntos relativos à sua classificação, as etapas que compõe uma simulação, sua possível aplicabilidade em sistemas embarcados utilizando o simulador SSD.

A simulação pode ser entendida como o ato de imitar uma situação real em um modelo que o represente, onde esse modelo deve possuir um comportamento similar ao sistema real,

considerando fatores como a variabilidade do sistema e demonstrando o que acontecerá na realidade de forma dinâmica. Isto permite que se tenha um melhor entendimento do sistema real, compreendendo as inter-relações existentes no mesmo (PEDGEN, SHANNON & SADOWSKI, 1995).

Costa (2000) complementa que há uma grande vantagem que é a interatividade com o modelo de simulação, ou seja, a facilidade de modificá-lo para fins de estudo e em alguns simuladores ter a possibilidade de se verificar através da animação como o processo está sendo conduzido.

Conforme Hill (1998), simulação computacional é o processo que permite que uma situação real possa ser representada com a ajuda de um *software* de computador mantendo as principais características, mas dando total controle sobre fatores que a realidade não permite, como o tempo.

Hill (1996) apresenta três categorias de *softwares* de simulação:

- *Softwares* dedicados com um escopo bem definido e aplicado a um domínio específico;
- As linguagens de simulação, ferramentas poderosas para modelagem de inúmeros sistemas, sendo que o único problema é o aprendizado do formalismo da linguagem; e
- Ambientes de simulação genérica.

Segundo Law (2000), historicamente, a simulação, como técnica, originou-se dos estudos de Von Neumann e Ulan. Tais estudos tornaram-se conhecidos como análise ou técnica de Monte Carlo. Essa técnica matemática é conhecida desde o século passado, na época em que os cientistas trabalhavam secretamente no projeto denominado "*Manhattan em Los Alamos*", para o desenvolvimento da bomba atômica dos aliados.

2.1. Vantagens e Desvantagens

A simulação de sistemas é vista por muitos autores e estudiosos da área como uma excelente ferramenta para solucionar problemas de diversos tipos. Mais precisamente ela é indicada nos seguintes casos:

- Sistemas do mundo real quando são com elementos estocásticos podem não ser descritos de forma precisa através de modelos matemáticos (FERREIRA, 2002);
- Quando o sistema é imaginário (SCHRIBER, 1991);
- Quando o tempo de observação da operação do sistema necessita de muito tempo;
- Quando não se deseja correr o risco de testar mudanças em determinados sistemas que já estão em funcionamento. Pode-se, então, utilizar a simulação sem correr riscos e necessidade de implementação no sistema real;
- Quando o sistema apresenta alta periculosidade e se alguma mudança no sistema real poderia causar algum tipo de acidente.

Segundo Law (2000) e Banks & Carson (1994), a simulação de sistemas apresenta as seguintes vantagens:

- Permite replicações precisas dos experimentos, podendo-se assim, testar alternativas diferentes para o sistema;
- Fornece um controle melhor sobre as condições experimentais do que seria possível no sistema real, pois pode-se fazer várias replicações do modelo, designando-se os valores que se deseja para todos os parâmetros;
- Permite simular longos períodos em um tempo reduzido;
- E, em geral, é mais econômico que testar sobre o sistema real, e evita gastos inúteis na compra de equipamentos desnecessários;
- Uma vez criado, um modelo pode ser utilizado várias vezes a fim de avaliar projetos e propostas;
- A metodologia de análise utilizada pela simulação permite a avaliação de um sistema proposto, mesmo que os dados de entrada estejam, ainda, na forma de esquemas ou rascunhos;
- Hipóteses como ou por que certos fenômenos acontecem podem ser testadas para confirmação;

- Pode-se compreender melhor quais variáveis são as mais importantes em relação a performance e como as mesmas interagem entre si e com os outros elementos do sistema;
- A identificação de gargalos pode ser obtida de forma facilitada, principalmente com ajuda visual;

A simulação de sistemas é uma ferramenta muito útil, porém pode apresentar limitações a serem observadas e levadas em consideração como:

- A simulação é muito dependente do modelo desenvolvido, ou seja, não adianta nada fazer um estudo detalhado dos dados de saída encontrando uma solução para o sistema, se o modelo criado não representa fielmente o sistema ou se os dados de entrada não são corretos Banks & Carson (1994);
- Exige-se treinamento especial para construção de modelos. Envolve arte e, portanto o aprendizado se dá ao longo do tempo, conforme vai se adquirindo experiência;
- Os resultados da simulação são, algumas vezes, de difícil interpretação e requer do usuário, conhecimento profundo do sistema que ele modelou e de técnicas estatísticas;
- A modelagem e a experimentação associadas à modelos de simulação, consomem muitos recursos, principalmente tempo e processamento. Simplificar a modelagem ou os experimentos na tentativa de economia costuma gerar resultados insatisfatórios (BANKS & CARSON, 1994);
- Devido às interpretações realizadas, erros em coletas ou até erros de arredondamento podem tornar os resultado inexatos (FERREIRA, 2002).

2.2. Classificação de Modelos e Simulação

2.2.1. Modelos

Um modelo é uma representação de um objeto, sistema ou idéia de forma diferente da entidade propriamente dita. Pode também ser entendido como um conjunto de informações que permite entender um sistema como um todo (EMSHOFF & SISSON, 1970).

Os modelos têm a uma importante função na construção de idéias que atendem ao sistema modelado (EVANS, WALLACE & SUTHERLAND, 1967).

De acordo com Ferreira (2002), os modelos podem ser classificados em:

- Modelos Físicos: Envolvem uma representação espacial, normalmente em forma diferente do sistema real. São classificados em modelos análogos e reduzidos:
    - Modelos análogos: Sistema de natureza diferente, mas com o comportamento similar;
    - Modelos reduzidos: Reprodução do objeto real em escalas diferentes.
- Modelos Lógicos / Matemáticos: Representam sistemas através de relações lógico-matemáticas. Algumas de suas características são:
    - Podem ser estáticos ou dinâmicos;
    - Podem ter solução analítica ou numérica;
    - Atualmente, a maioria dos modelos envolve equações diferenciais;
    - Representam sistemas dinâmicos complexos, normalmente sistemas físicos ou biológicos.

- Os modelos de simulação possuem as seguintes características:
    - Podem incluir relações lógico-matemáticas;
    - Permitem representar entidades individuais e a ocorrência de eventos específicos;
    - São mais intuitivos e fáceis de usar que os modelos lógico-matemáticos;
    - Podem representar o sistema real com muitos detalhes sem aumentar a complexidade do modelo na mesma proporção;
    - Exigem o uso de computadores para obter a solução em tempo aceitável;
    - Normalmente têm alta demanda de processamento: horas, dias, meses, anos...

2.2.2. Simulação

2.2.2.1 Simulação Orientada ao Tempo

Numa simulação orientada ao tempo, o tempo avança em intervalos fixos. As mudanças de estado do sistema ocorrem em pontos discretos no tempo, quando o relógio de simulação avança. (SOARES, 1992).

2.2.2.2. Simulação Orientada a Evento

Numa simulação orientada a evento, a modelagem do sistema é realizada através da identificação dos seus eventos característicos e incondicionais, dependente unicamente da simulação. Rotinas descreverão as mudanças de estado quando ocorrer um evento associado a elas. Esse processo de simulação evolui ao longo do tempo pela execução de eventos que estão numa pilha, escolhendo o evento cujo o tempo esteja mais próximo do tempo corrente da simulação (FEITAS FILHO, 2002).

Nos modelos de simulação discreta orientada a eventos, alguns conceitos são muitos utilizados e são descritos a seguir:

- Estado do sistema: é um indicador de como estão as características do modelo simulado em um determinado instante;
- Variável de estado: é um elemento do modelo que varia de acordo com uma característica do sistema;
- Evento: é um acontecimento instantâneo que causa mudanças no estado e variáveis do sistema;
- Entidade: é qualquer elemento que possa ser representado no sistema e suas características podem determinar o propósito do modelo simulado;
- Atributo: é uma propriedade da entidade que a caracteriza no mundo real;
- Recurso: é um elemento que presta serviços e modificam o comportamento das entidades;
- Período de atividade: é o espaço de tempo para que se possa programar as entidades;
- Período de espera: é um período de tempo que uma vez iniciado, somente tem o seu valor quando a simulação termina;
- Tempo simulado: é o tempo do sistema como um todo que representa o comportamento do modelo na realidade;
- Tempo de simulação: é o tempo real que o modelo leva para ser executado.

2.2.2.3. Simulação Determinística

Para Shannon (1975), a simulação determinística foi utilizada para permitir testar o funcionamento do sistema em situações bem específicas e críticas. Esse tipo de modelo considera que as mudanças no cenário da simulação entre suas replicações ocorrem de uma maneira única e pode ser prevista através do conhecimento das condições iniciais. Esse processo depende de duas

condições: que o sistema tenha um tamanho infinito e que o ambiente permaneça constante com o tempo ou que mude de acordo com regras também determinísticas. É claro que essas condições são raramente encontradas e, portanto, uma abordagem determinística pode não ser suficiente para descrever as mudanças do sistema (ZEIGLER, 1987).

2.2.2.4. Simulação Estocástica

Para Zeigler (1987), trabalhar com valores flutuantes e aleatórios requer uma abordagem matemática diferente. Os modelos estocásticos consideram que as mudanças nos cenários da simulação acontecem de forma probabilística e que o conhecimento das condições em cada replicação não permite prever, de forma precisa, o próximo resultado. Os modelos estocásticos são preferidos em relação aos determinísticos, pois são baseados em suposições mais realistas. Entretanto, modelos deterministicos são muito mais fáceis de serem tratados matematicamente e podem prover, muitas vezes, aproximações suficientemente precisas.

2.3. Etapas de um Projeto Envolvendo a Simulação

Para desenvolver um modelo de simulação é necessário todo um método planejado para que um usuário consiga obter, com sucesso, tirar todas as conclusões sobre o sistema real (PRITSKER, 1979). A figura 22 mostra os passos para se construir um modelo de simulação. A ilustração e os comentários foram baseados em propostas similares apresentadas por Banks e Carson (1994), Pegden, Shannon e Sadowski (1995) e Law (2000).

As principais etapas que compõe um projeto de simulação são as seguintes:

- Definição do problema e objetivos: O estudo do projeto de simulação inicia com a formulação do problema. Definem-se os propósitos e objetivos, impõe-se limites da abordagem de forma clara. Feito isto, é importante verificar se o projeto de simulação pretendido responde adequadamente ao que se deseja como, por exemplo, se animações e resultados estatísticos são necessários;
- Definição do Modelo Conceitual: Trata de traçar um esboço do sistema, através de fluxogramas, desenhos, e definir variáveis, componentes, relações entre partes, necessidades das partes. Uma tendência comum nessa etapa é querer modelar tudo, e não enxergar os

limites, tornando o modelo muito maior do que se tinha imaginado inicialmente. Por esse motivo é que os objetivos devem estar bem especificados. Também existe o caso se o modelo for simples demais, resultando em obter informações que podem ser imprecisas e não representar a realidade;

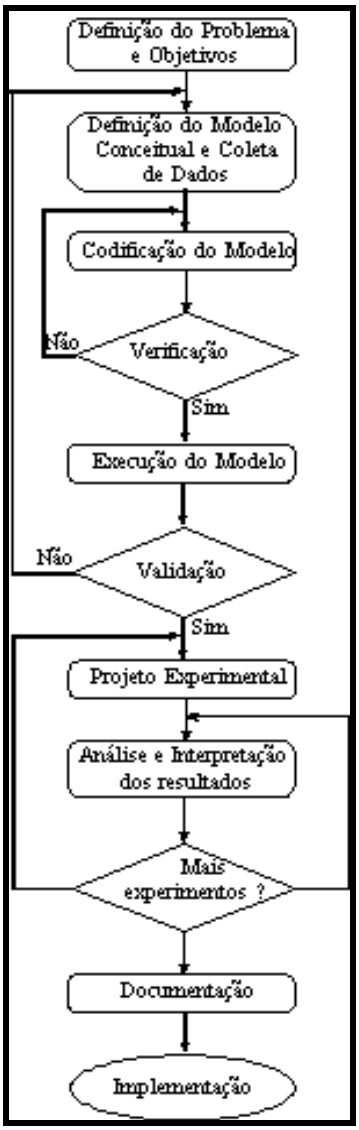

Figura 22. Etapas da Simulação.

Fonte: Adaptada de Banks e Carson (1984), Pegden, Shannon e Sadowski (1995) e Law e Kelton (1991).

- Coleta de dados: As informações de entrada do sistema devem ser adquiridas e coletadas em sistemas existentes. Caso não seja possível, devem fazer estimativas. Dificilmente o usuário possui todos os dados de entrada, necessários e na forma correta. O mais comum é encontrar tais dados numa forma resumida; a exemplo disso são informações de performance baseadas

em média de valores. O que não é interessante devido às distorções dos resultados após a simulação e, conseqüentemente, a possível validação do modelo. A construção do modelo deve ter o nível de detalhamento proporcional ao sistema. PRITSKER (1995);

- Codificação, Verificação e Validação do Modelo: A codificação nada mais é que passar o modelo de uma linguagem conceitual para uma computacional. A verificação trata-se de uma varredura no modelo codificado a fim de encontrar diferenças relacionadas ao modelo conceitual. A validação é a confirmação de que o modelo construído corresponde ao sistema real, isto é, se os resultados estimados pela simulação são confiáveis, e representam a realidade. Em geral, testes estatísticos avaliam as diferenças entre os resultados do modelo e do sistema real. Se forem estatisticamente insignificantes, o modelo é validado;
- Projeto Experimental: Nesse projeto é que estão especificados os diferentes parâmetros de entrada do modelo. Para cada configuração o modelo de simulação é executado e os resultados avaliados (PAPADOPOULOS, 1994);
- Análise e Interpretação dos Resultados: Nesta etapa são analisados todos os resultados obtidos nas simulações. Dependendo das respostas dessas análises, pode acontecer a necessidade em simular mais vezes o modelo para que se possa alcançar a precisão estatística dos resultados. Para se obter resultados com nível de precisão adequados e confiáveis, deve-se replicar inúmeras vezes o modelo simulado para obter informações sobre o seu comportamento.

2.4. Mecanismo de Avanço do Tempo

Segundo Freitas Filho (2001), devido a fatores de dinamismo que a simulação discreta proporciona, é necessário possuir total controle do tempo na medida que vai avançando na simulação. Um *software* de simulação é necessário que se implemente um mecanismo para avançar no intervalo de tempo proporcional na simulação. Geralmente é uma variável cuja unidade de tempo não é estabelecida no simulador, mas como parâmetro no início da simulação.

Para representar um mecanismo de avanço ao tempo, existem duas formas que são implementados por ferramentas ou linguagens de simulação: avanço do tempo com incremento fixo (orientado ao tempo), e avanço do tempo para o próximo evento (orientado a eventos).

Avanço do tempo para o próximo evento é um mecanismo largamente utilizado em ferramentas de simulação. Seu funcionamento começa por inicializar a variável do tempo em zero e os próximos eventos são determinados através de uma lista de eventos, onde cada vez que um evento ocorre, há um avanço na unidade de tempo da variável para o instante de ocorrência do evento. Esse processo continua até que seja satisfeita uma condição de parada ou até não existir mais eventos na lista de eventos. (FREITAS FILHO, 2001).

2.5. Estatística na Simulação

Para Milone (1993), o papel da estatística na simulação é extremamente fundamental para que se possam obter dados significativos sobre os resultados dos experimentos. Outra forma em que ela é usada constitui no fornecimento de informações de entrada, pois, é a partir do resultado da simulação desses dados, que o modelo é validado. A estatística é utilizada diretamente nas seguintes etapas da simulação:

- Busca de Informações de Entrada;
    - Cálculo do tamanho necessário da amostra;
    - Busca e verificação das amostras realizadas;
- Validação do Modelo;
    - Testes para verificação e validação do modelo através dos resultados obtidos;.
- Experimento da Simulação;
    - Duração da simulação;
    - Quantidade de replicações (simulações) necessárias;
    - Mudança de parâmetros em cada replicação.
- Análise dos resultados.
    - Teste das suposições feitas.

2.5.1. Distribuição de Probabilidade

A descrição de um conjunto de entidades que são submetidos à simulação discreta é chamada de população. Considerando-se como objetivo último da estatística a descrição de

populações, pode-se tomar uma variável para descrevê-la, sendo a freqüência o número de vezes que certo valor dessa variável ocorre no conjunto de dados. (ROCHA & NETO, 1997)

Quando é dada uma população finita, podem se realizar experimentos sem se preocupar em cálculos estatísticos. Se somente uma parte da população foi obtida (amostras), o que pode ser feito é explorar a respeito da distribuição das freqüências da população, sendo que a distribuição de freqüência relativa corresponde a um modelo teórico chamado distribuição de probabilidade.

Na simulação, onde os resultados geralmente são obtidos através de amostras, o uso de um estudo a respeito do comportamento de populações é fundamental.

Algumas das principais distribuições de probabilidade e suas características serão descritas abaixo.

- Distribuição Discreta de Bernoulli: Caracteriza-se pelo fato de que apenas dois resultados serão possíveis. Verdadeiro com probabilidade “p” e falso, com probabilidade “1–p”.
- Distribuição Discreta Binomial: Uma variável aleatória binomial caracteriza-se por ser a soma de n variáveis independentes e com idêntica distribuição de Bernoulli. Assim sendo, pode assumir apenas dois tipos de resultados para cada variável.
- Distribuição Discreta de Poisson: Aplica-se a eventos aleatórios que ocorrem num período de tempo, podendo assumir qualquer valor inteiro positivo.
- Distribuição Normal: Distribuição de variáveis aleatórias muito importante. A normalidade dos dados é fundamental em muitas deduções usualmente realizadas. Sabendo-se que, de uma variável, a distribuição mais importante é aquela que melhor a descrever, deve-se verificar os ajustes à distribuição utilizada. Objetivando facilitar a utilização das propriedades da distribuição normal e torná-la comparável em qualquer situação, é comum o emprego da distribuição normal reduzida ou padronizada. Assim sendo, pode-se utilizar uma tabela de probabilidade para esta distribuição normal padronizada.
- Distribuição Uniforme: Especifica que os valores compreendidos entre o mínimo e o máximo são equiprováveis. O uso desta distribuição geralmente significa um completo desconhecimento da variável aleatória, conhecendo apenas seus limites (ROCHA & NETO, 1997).

- Distribuição Triangular: É usada quando é possível se determinar o valor mais provável da variável aleatória, além da sua faixa mínima e máxima, e quando uma função linear parece apropriada para a descrição (BIRTWISTLE, 1979).
- Distribuição Exponencial: Se a probabilidade de acontecer um evento em um intervalo de tempo pequeno é proporcional ao tamanho desse intervalo, se a probabilidade de ocorrência de mais um evento neste intervalo é nula, ou se a ocorrência de um evento é independente da ocorrência de outros, então se trata de uma distribuição exponencial. O uso da distribuição exponencial supõe uma grande variância e possui um tratamento matemático relativamente simples (ibidem).

2.5.2. Estimação de Parâmetros

Conforme Magalhães (2002), a estimação de parâmetros tem um papel essencial na estatística, pois a idéia é estimar parâmetros de um modelo matemático que descrevam situações da vida real. Um grande número de modelos matemáticos contém equações diferenciais. Assim, juntamente com a estimação de parâmetros têm que se resolver, a maior parte das vezes numericamente a equação ou sistema de equações diferenciais em causa. O modelo matemático que é fornecido pelo criador do modelo deve pertencer a uma das seguintes categorias: equações diferenciais ordinárias com valores iniciais, equações diferenciais algébricas, equações diferenciais unidimensionais com derivadas parciais dependentes do tempo, equações diferenciais unidimensionais com derivadas parciais algébricas (WERKEMA, 1996).

Segundo Reis (1998), o roteiro para estimação de parâmetros está determinado nos seguintes passos:

1) Determinar o parâmetro a ser estimado: média ou proporção populacional.
2) Estabelecer nível de confiança (1-$\propto$) ou nível de significância ($\propto$).
3) Calcula-se as estatísticas necessárias: x, p, s.
4) Escolhe-se a variável de teste (Z ou t).
5) Encontra-se **Zcrítico** ou **Tn-1, crítico** nas tabelas apropriada.
6) Calcular os limites do intervalo de confiança.
7) Interpretar o intervalo de confiança obtido.

Grande parte das vezes, o modelo matemático contém dados adicionais de modo a ser possível aplicá-lo a uma gama alargada de casos. Podem dar-se os seguintes exemplos:

- É definido mais que um critério de ajuste, isto é, mais do que um conjunto de dados pode ser aplicado a determinado modelo;

- O critério de ajuste pode incluir funções arbitrárias que dependem dos parâmetros a ser estimado, da solução do sistema dinâmico e da variável tempo (MAGALHÃES ,2002);

- Pode ser introduzido um parâmetro independente do modelo, chamado concentração;

- O modelo possui restrições de igualdade ou desigualdade arbitrárias, assim como limite inferior e superior a serem estimados. As restrições podem depender ainda da solução da equação dinâmica a um dado tempo e concentração;

- Os modelos dinâmicos contêm pontos de mudança em que a integração é iniciada, por exemplo, para funções de entrada descontínuas ou alterações no modelo;

- As equações diferenciais parciais podem conter equações algébricas, por exemplo, para resolver sistemas de ordem superior. Os valores iniciais têm que ser calculados simultaneamente de forma a serem coerentes;

Uma forma rápida e empírica de estimar parâmetros é olhar para o gráfico resultante dos pontos experimentais e tentar encaixar visualmente uma linha, se possível reta, aos mesmos. No entanto, a maior parte das vezes a linha obtida é não linear, o que complica o processo de estimação.

Uma das técnicas mais usadas consiste em minimizar a distância entre alguns dados experimentais conhecidos e os dados teóricos calculados através do modelo. Assim, mesmo os dados que não possam ser medidos diretamente podem ser estimados por esta técnica, denominada mínimos quadrados (MAGALHÃES, 2002).

2.5.3. Testes e Validação em Estatística

Na simulação, todos resultados obtidos são somente validados se estiverem próximos de resultados de experimentos reais. Geralmente, os resultados de uma pesquisa são obtidos através de amostras que, em muitas vezes, quando mal obtidas, interferem nos resultados da simulação e, conseqüentemente, a comparação com a realidade. Para que se possa obter validação com os resultados de uma simulação, existem alguns métodos para validação baseado em amostras. Entre eles:

2.5.3.1 Intervalo de confiança

Quando se estima um parâmetro populacional com base numa amostra dessa população, o valor encontrado é uma variável aleatória e, como tal, pode ser diferente para cada amostra realizada. Portanto, uma estimativa pontual do parâmetro não é adequada. Normalmente constrói-se com um intervalo de valores sobre o valor obtido numa amostra, e se fornece uma probabilidade associada à possibilidade do real valor do parâmetro populacional estar contido nesse intervalo.

2.5.3.2 Testes de Hipóteses

Uma hipótese estatística é uma afirmativa a respeito de um parâmetro de uma distribuição de probabilidade. Para se fazer o teste de hipótese, parte-se de um valor de hipótese inicial ou nula chamada de H0. Ao testar a hipótese, toma-se uma amostra aleatória do sistema em estudo e se calcula o parâmetro desejado tendo como resultado chamada hipótese alternativa, ou H1. A rejeição de H0 implicará a aceitação de H1. A hipótese alternativa geralmente representa a suposição que o pesquisador quer provar, sendo H0 formulada com o expresso propósito de ser rejeitada a pior. (ROCHA & NETO, 1997).

2.5.3.3 Testes do Chi-Quadrado

Permite verificar igualdade (semelhança) entre categorias discretas e mutuamente exclusivas, aplica-se quando há um interesse no número de indivíduos, objetos, respostas ou medidas que se enquadra em diversas categorias entre freqüências e não escores médios. Uso do teste para duas amostras buscando provar que as mesmas diferem em relação a uma característica é fazendo um estudo relacional entre variáveis (com a mesma relação causal), isto é, o tipo de relação entre elas: independência ou dependência (MAGALHÃES, 2002).

2.5.3.4 Testes Paramétricos

Para Werkema (1996), o termo paramétrico e não-paramétrico se refere à média e ao desvio-padrão, que são os parâmetros que definem as populações que apresentam distribuição normal. Quando se estima um parâmetro populacional com base numa amostra dessa população, o valor encontrado é uma variável aleatória e, como tal, pode ser diferente para cada amostra realizada. Portanto, uma estimativa pontual do parâmetro não é adequada. Normalmente constrói-se com um intervalo de valores sobre o valor obtido numa amostra, e se fornece uma probabilidade associada à possibilidade do real valor do parâmetro populacional estar contido nesse intervalo.

O roteiro para estimação de parâmetros está determinado nos seguintes passos segundo Reis (1998):

1) Enunciar as hipóteses H0 e H1 (indicando qual o parâmetro de interesse, se o teste é unilateral ou bilateral);
2) Estabelecer o nível de significância ($\propto$) ou nível de confiança (1-$\propto$) do teste;
3) Identificar a variável de teste: Z ou t;
4) Definir a região de aceitação de H0 (de acordo com o tipo de teste);
5) Através dos valores da amostra avaliar o valor da variável (comparar valor da variável de teste com o valor crítico);
6) Decidir pela aceitação ou rejeição de H0;
7) Interpretar a decisão no contexto do problema.

2.5.4. Problemas da Estatística Relacionados com a Simulação

De acordo com Soares (1992), a estatística possui um papel fundamental na simulação de sistemas, pois é através dela que os modelos são alimentados com as informações de entrada e fornecem dados que são analisados e o modelo é validado.

Mas, quando o objetivo é obter resultados que indicam o desempenho, custo e outras medidas a serem analisadas, podem ocorrer situações onde erros são muito comuns e provenientes de diversas fontes. Fatores como a estimativa de valores imprecisos é muito encontrada em projetos de simulação. Para que um modelo funcione corretamente, devem ser fornecidas informações confiáveis que possam ser adquiridas através de experimentos cujo conjunto de dados são coletados de situações reais ou dados até mesmo aleatórios (LAW 2000).

Para as simulações, como sistemas embarcados, se as estimativas referentes a tempos de execução, duração de acessos a dispositivos, taxa de acesso à memória, entre outras, forem imprecisas, estas poderão interferir no desempenho ou simplesmente apresentar resultados não confiáveis.

2.6. Simuladores de Sistemas Embarcados

Na constante evolução em relação a projetos de sistemas embarcados, alguns simuladores específicos estão cada vez mais presentes no mercado. Com o intuito de poder melhor compreender o funcionamento desse tipo de simulador, foram pesquisadas duas ferramentas que propõem a resolução do problema da simulação de sistemas embarcados. Pontos como tipo de simulação e o nível de modelagem e abstração que estas ferramentas se propõem justificam a exposição de suas principais características.

2.6.1. MAX+PLUS II

O software MAX+PLUS II (*ALTERA Multiple Array MatriX Programable Logic User System*) é um sistema de desenvolvimento de dispositivos de lógica programáveis ou PLD (*Program Logic Devices*). Essa ferramenta, aliada a equipamentos que "queimam" esses PLDs torna muito pratico e rápido o desenvolvimento de componentes digitais, pois toda sua arquitetura é projetada via *Software*. Um dispositivo PLD bastante popular é o FPGA. O FPGA (*Field Programmable Gate Array*) é um componente que possui uma matriz de blocos lógicos onde existem diversos *flip-flops* e tabelas de função lógica LUTs (*Look Up Tables*). Quando se deseja "queimar" esse circuito, utiliza-se de um aparelho eletrônico que, conectado a um micro-computador liga esses componentes dessa matriz de forma que se obtenha o circuito final. A Figura 23 ilustra a interface do Max+Plus II.

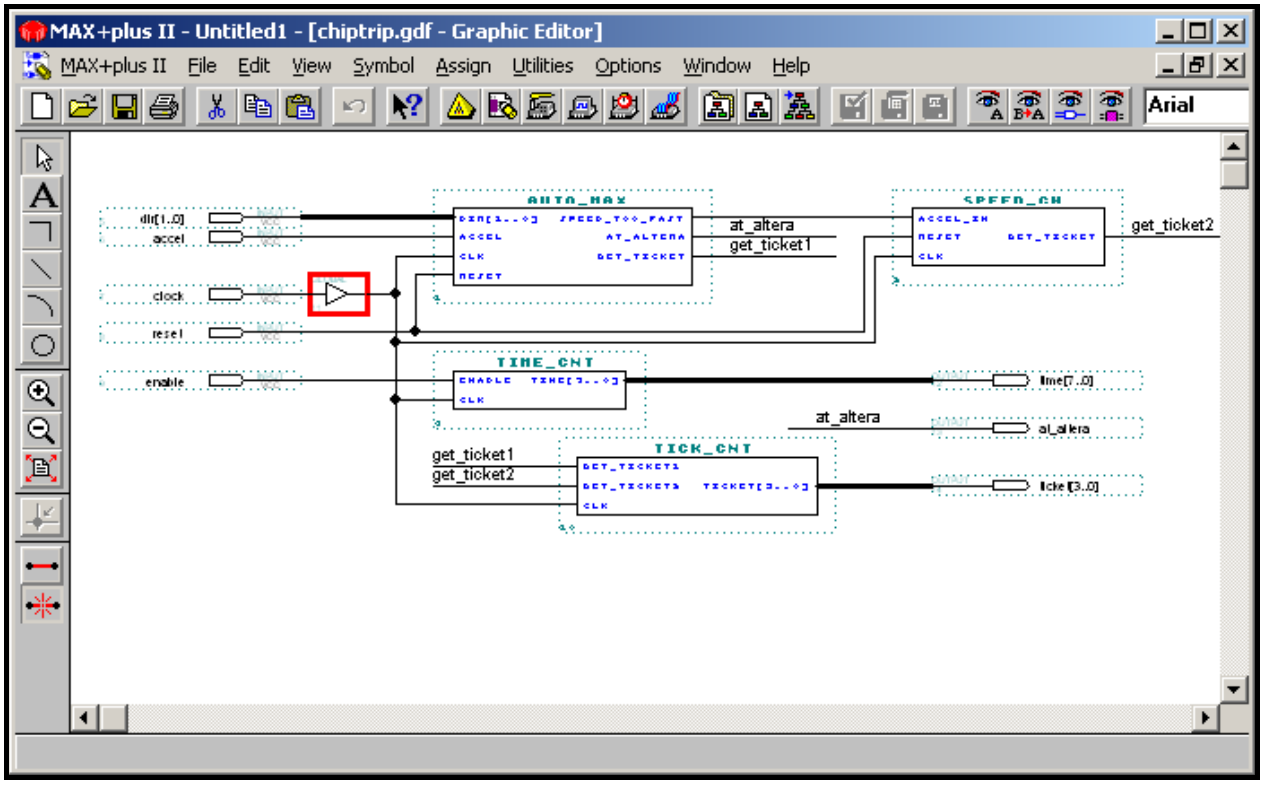

Figura 23. Interface do Max+Plus

Fonte: Adaptada de Altera (2003)

O MAX+PLUS é uma das ferramentas pesquisadas que propõem uma simulação determinística de mais baixo nível, chegando na composição do modelo partindo do esquema de circuitos digitais ou VHDL (*VHSIC Hardware Description Language*).

O desenvolvimento nesse *software* propõe ao usuário, uma rica coleção de ferramentas que auxiliam cada parte do projeto, desde seu desenho até na parte de geração de sinais para testes. Cada ferramenta é um modo que o Max+Plus II pode trabalhar, abaixo, segue algum desses modos:

- Esquemático: Nesse modo é feito a parte do desenho do circuito lógico utilizando símbolos que representam os componentes básicos. Esses componentes como portas lógicas, *flip-flops*, circuitos integrados e outros componentes são interligados através de canais de conexão.

- AHDL (Altera *Hardware Description Language*): É a linguagem utilizada para descrição do comportamento de todo hardware proposto pela empresa ALTERA.

- Editor de Forma de onda: Esse modo é capaz de desenhar algumas formas de onda que vão gerar sinais de entrada para testes no circuito. A grande vantagem de se utilizar uma ferramenta como o Max+Plus II é que podemos testar o comportamento do circuito através de sua simulação sem que antes seja preciso gravar no dispositivo PDL.

Além desses modos, é possível agregar ao *sofware* outros componentes e funcionalidades como o suporte para outros dispositivos PDLs. Por ser uma ferramenta comercial, o Max+Plus II é um pouco restrito em termos de aquisição profissional. No meio acadêmico a Altera possui uma licença dedicada a estudantes que, por ser limitada, ainda oferece todos os principais recursos para concepção de pequenos projetos.

2.6.2. Isis/Proteus

Isis também conhecido como *Proteus* é um *software* produzido pela *Labcenter Eletronics* é um simulador específico de sistemas embarcados. Ele modela inúmeros componentes que simulam esse tipo de sistema: microcontroladores, memórias, motores de passo, portas seriais, *displays*, entre outros. A Figura 24 ilustra a *interface* e o funcionamento do *Isis* com um projeto confeccionado.

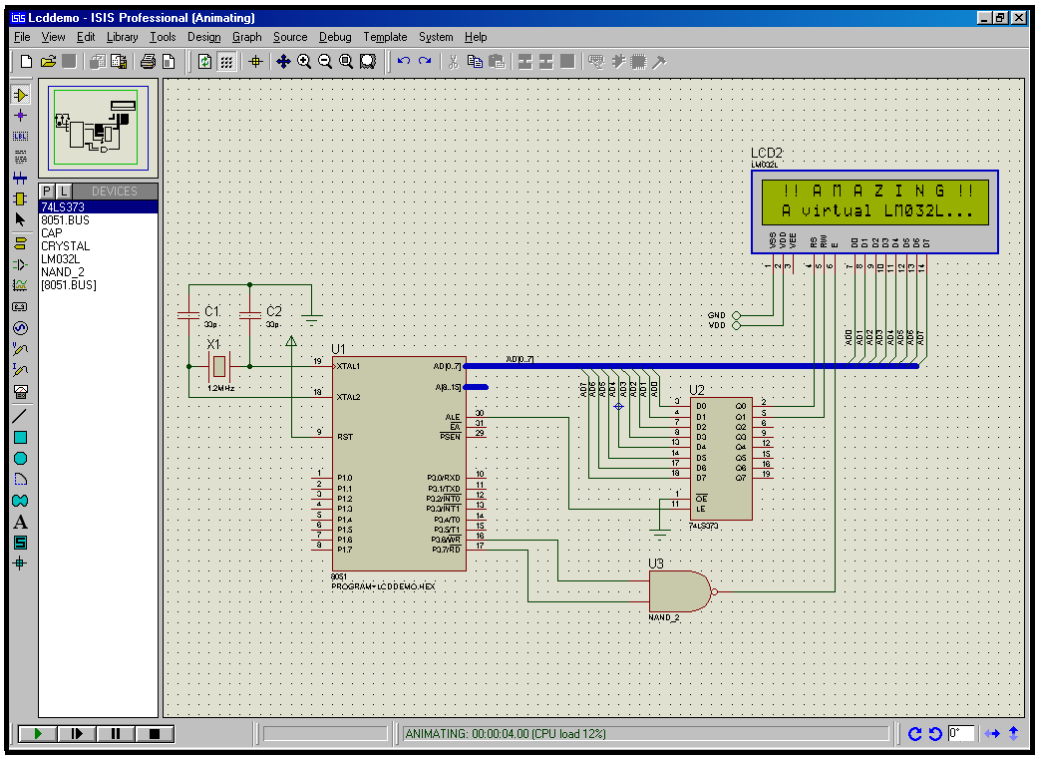

Figura 24. Interface do *Isis*

Uma característica muito importante da ferramenta é que possui uma grande variedade de componentes, principalmente dispositivos digitais integrados (como microcontroladores, memórias EEProm, *timers*, entre outros). Além de possuir uma grande variedade de microcontroladores, esse simulador aceita diversos formatos de programas para esses circuitos. Outra característica muito importante é a possibilidade de depuração do modelo. Informações sobre o status da memória interna, registradores e pilha, podem ser observados no instante da depuração. A Figura 25 ilustra um modelo sendo depurado.

Uma vantagem do uso de um simulador como o *Isis* é a facilidade de simular um sistema embarcado, completo sem a necessidade de adquirir os componentes para implementar o projeto real. Assim, pode-se validar um projeto proposto ou simplesmente um componente como um microcontrolador recém lançado, para que possa partir posteriormente para uma implementação física do sistema.

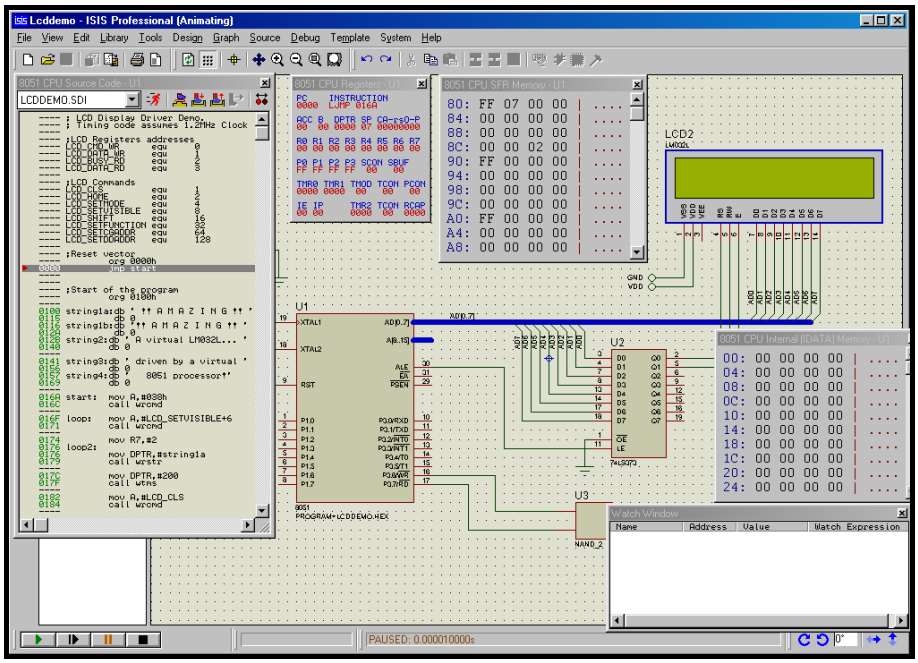

Figura 25. Editor de Código Fonte do *Isis*

Mesmo com as facilidades visuais que o *Isis* possui, ele trabalha com simulação de sistemas embarcados em médio nível, pois o mesmo não se trata de simular *flip-flops* como o Max+Plus, mas é necessário que todos os componentes estejam ligados corretamente obedecendo a critérios como pinagem e código do microcontrolador estejam coerentes com o modelo. Além disso, são necessários conhecimentos mais profundos de componentes de sistemas embarcados como microcontroladores e sua programação para utilizarem corretamente.

Outra desvantagem encontrada é que o *Isis* não possui nenhuma ferramenta de análise estatística incorporada, obrigando o usuário que deseja obter métricas de desempenho, custo e validação a buscar outros *softwares* para esse fim.

2.7. O Simulador SSD

O SSD é uma ferramenta de simulação genérica que foi desenvolvida em projetos e pesquisa para simulação discreta de sistemas e possui a característica de modelar componentes que possam ser desenvolvidos por usuários e adicionados a ele. Nessa ferramenta, além dos componentes básicos para simulação genérica, foram incorporados por Stefanes (2002), alguns componentes que simulam redes locais. Em sua abordagem abrangeu elementos de *hardware* (microcomputador, *interface* de rede e roteadores), como em *software* implementando componentes que representam

protocolos como FTP (*File Transfer Protocol*), HTTP (*Hyper Text Tranfer Protocol*), entre outros. O SSD por ser uma ferramenta de simulação genérica, permite ao usuário construir diferentes tipos de modelos, podendo simular diversos tipos de aplicação (STEFANES, 2002).

O SSD é uma ferramenta de simulação discreta orientada a eventos, possuindo uma *interface* visual que permite que a modelagem seja feita de forma mais fácil, diminuindo a margem de erros e tempo de modelagem e também possui uma interface em modo texto que permite executar os modelos sem animações gráficas, aumentando o desempenho. A Figura 26 ilustra o SSD e sua interface gráfica.

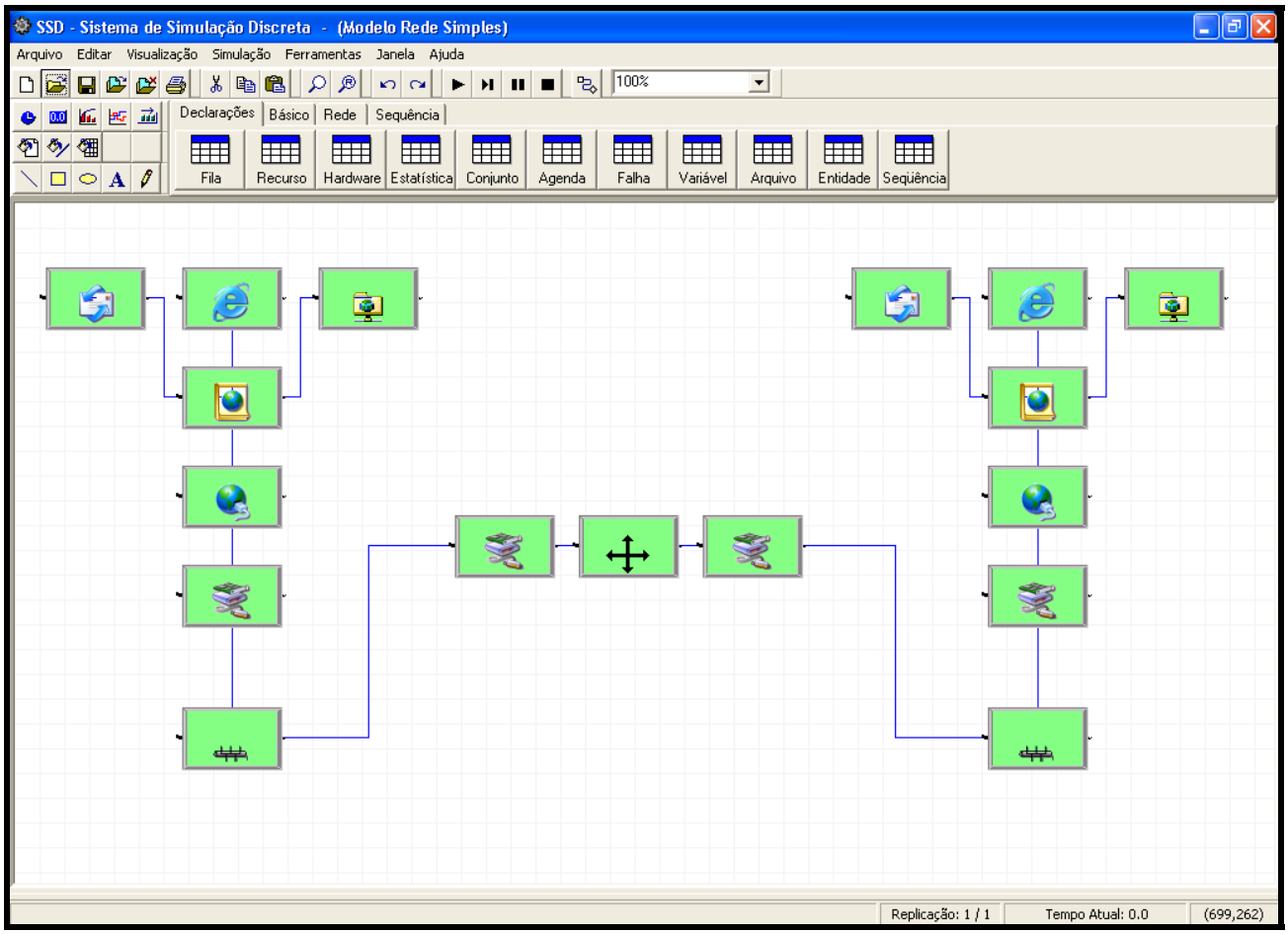

Figura 26. Interface gráfica do SSD.

Fonte: Adaptada de Stefanes (2002)

2.7.1. Inclusão de componentes no SSD

Os componentes desenvolvidos para o simulador SSD são incluídos de forma dinâmica e transparente, na forma de *plug-in*. Para incluir componentes ao sistema basta escolher o arquivo de biblioteca dinâmica (DLL, *Dinamic Link Library*) e a paleta de ferramentas que deseja incluir. Algumas informações extras sobre o componente são descritas como nome, autor, tipo, versão e descrição. Após sua seleção ele já faz parte da coleção de componentes que possam ser utilizados com o simulador. A Figura 27 mostra o processo de Inclusão de componentes.

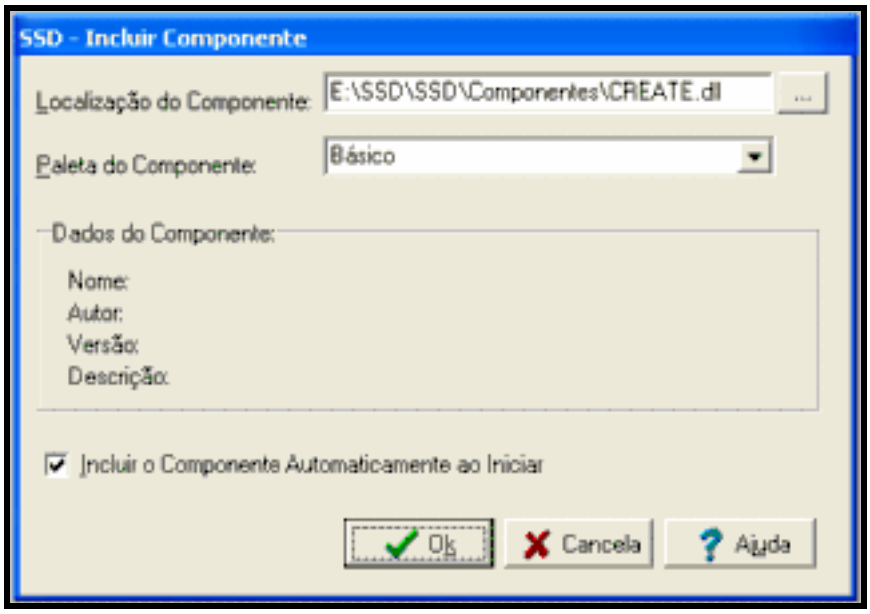

Figura 27. Inclusão de componentes no SSD.

Fonte: Adaptada de Stefanes (2002)

Esses arquivos DLLs, para funcionar corretamente no simulador, seguem um padrão bem definido para que todas funções possam ser acessadas. Uma vez que estejam dentro dessa especificação, o simulador interpreta e reconhece seu formato e inclui no seu sistema tornando disponível para o usuário (STEFANES, 2002).

Se caso o usuário desejar, esses componentes também podem ser removidos da ferramenta através da opção de remoção dos componentes. Nessa situação, o simulador remove a referência da DLL selecionada com o simulador e acusa erro quando este componente existir em um modelo aberto após a sua exclusão.

2.7.2 Classes básicas

O simulador SSD é formado por diversas classes que fazem o controle, tanto da aplicação em si (o *software SSD*), como os modelos que são criados. Um esquema de generalização de classes muito comum na abordagem orientada a objetos é utilizada na implementação do SSD. As principais classes que compõe o SSD são descritas a seguir:

- TSSDApplication: Esta classe gerencia os *plug-ins* que são carregados no simulador e os modelos que são criados neles. Uma aplicação do SSD, pode possuir um ou mais modelos abertos totalmente distintos;

- TModel. Esta classe é responsável pelo controle da simulação. Suas principais funções são escalonar os eventos, controlar o tempo da simulação, controlar a simulação (parar, executar, pausar), manter informações do projeto (nome do analista, título do projeto, versão), mostrar as estatísticas, ler e salvar os modelos, entre outras funções;

- TSIMAN: As principais funções que os modelos utilizam para representar um sistema que se pretende simular estão disponíveis nesta classe. Ela controla a criação e manutenção de todos elementos disponíveis para confecção dos modelos. Elementos como entidades, atributos, variáveis, filas, recursos e coleta de estatísticas fazem parte constante de um *plug-in* implementado;

- TCSTAT: Sua principal função é facilitar o componente a coletar estatísticas. Um componente pode possuir diversas informações estatísticas utilizando vários TCSTATS. Algumas das informações que são extraídas desta classe são média, valor mínimo, valor máximo, Intervalo de confiança, entre outras.

III – DESENVOLVIMENTO

1. APRESENTAÇÃO

Neste capítulo são apresentadas as etapas no desenvolvimento e modelagem deste trabalho de conclusão de curso, assim como os testes necessários para a sua validação.

Os componentes desenvolvidos nesse trabalho permitem ao SSD simular aspectos básicos de sistemas embarcados e redes-em-chip. Através da modelagem em alto nível do SSD e da utilização de dois modelos de sistemas embarcados desenvolvidos e já testados em outras ferramentas específicas, são comparadas através de informações estatísticas extraídas das duas ferramentas e verificando sua equivalência.

2. MODELAGEM

A análise utilizada para desenvolvimento dos componentes foi a orientada a objeto, pois esta possui a capacidade de modelar aspectos conceituais, criando as visões necessárias para o desenvolvimento do sistema.

A linguagem de modelagem utilizada é a UML (*Unfied Modeling Language*), que suprime satisfatoriamente os requisitos mencionados. Dentre os diagramas que compõe a UML, foram utilizados o Diagrama de Componentes e o Diagrama de Classes. Foram utilizados algoritmos para a descrição do funcionamento dos componentes.

2.1. Componentes de Sistemas Embarcados

Cada componente desenvolvido neste trabalho tem a finalidade de simular/representar o funcionamento de uma parte de um sistema embarcado. Além disso, cada componente armazena informações estatísticas referentes ao seu funcionamento durante o decorrer da simulação, como, por exemplo, a sua taxa de utilização. Essas estatísticas são próprias de cada componente, e, após o término da simulação, serão apresentadas ao usuário, como métricas de desempenho do sistema.

2.1.1. Diagrama de Componentes

O diagrama de componentes representa o aspecto físico da composição de um sistema. Ele é apresentado na Figura 28 onde demonstra a relação das bibliotecas (*plug-ins*, dll) com o simulador. Dessa forma, as bibliotecas dinâmicas dependem do simulador para que possam ser acessadas.

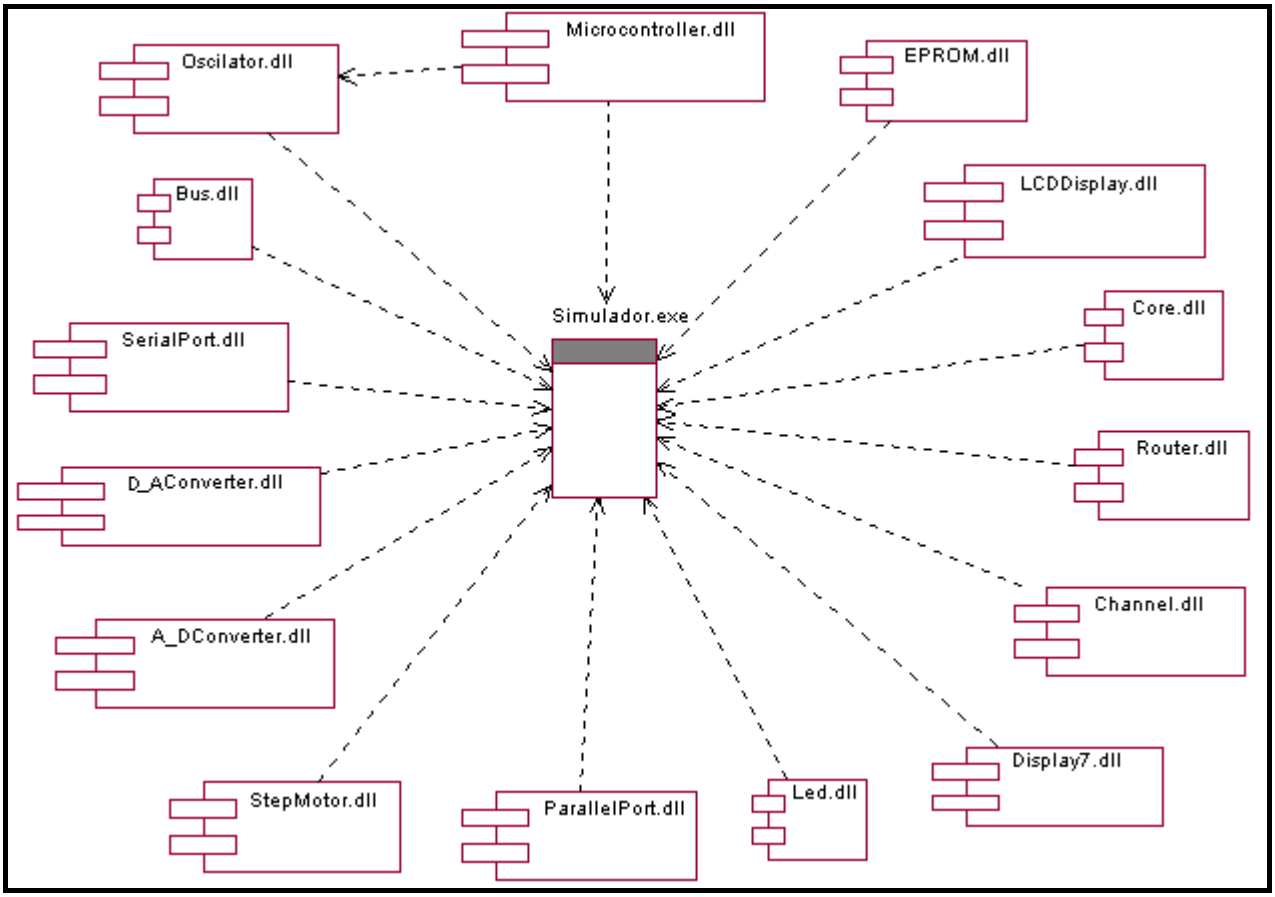

Figura 28. Diagrama de Componentes no SSD.

2.1.2. Diagrama de Classes

O diagrama de classes demonstra a estrutura estática das classes de um sistema. Classes podem se relacionar com outras através de determinados tipos de relacionamentos (associação, dependência, especialização, entre outras). Estes relacionamentos e outros são mostrados no diagrama de classes juntamente com as suas estruturas internas, que são os atributos e métodos.

A Figura 29 apresenta as classes dos componentes implementados neste trabalho, suas heranças e relacionamentos. Nota-se que todos os componentes principais possuem características herdadas da classe TModule do SSD, que faz todo o controle de todos componentes inseridos na simulação.

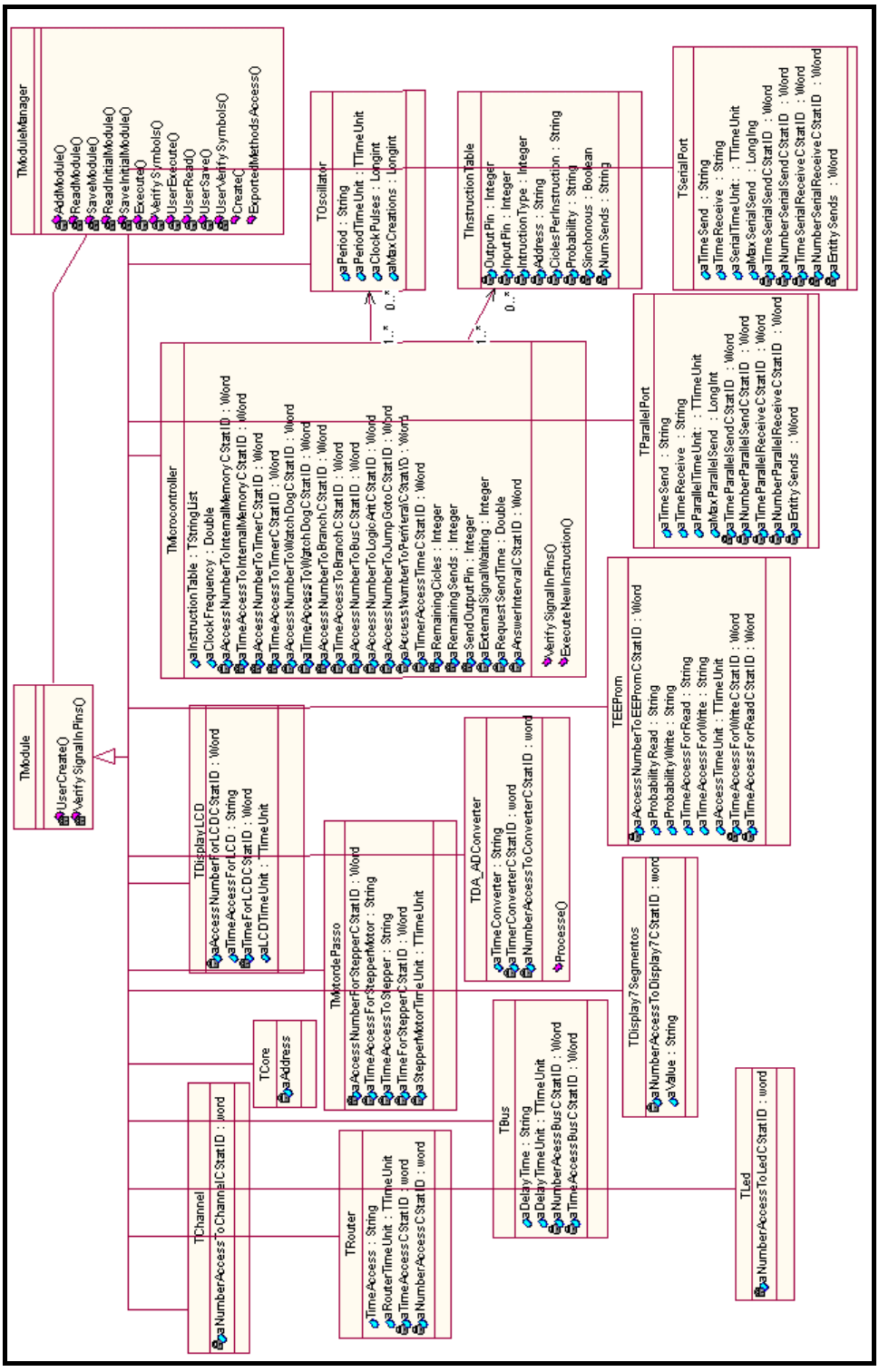

Figura 29. Diagrama de Classes dos Componentes no SSD.

2.1.3. Estrutura e Funcionamento

A estrutura e o funcionamento modelado dos componentes de *hardware* considerados neste trabalho são apresentados a seguir:

2.1.3.1 Microcontrolador

O componente microcontrolador é um dos principais elementos no projeto de sistemas embarcados, e é através dele que são acessados todos os outros componentes, como memória e periféricos. Durante o decorrer da simulação, o microcontrolador gera as entidades com base na sua freqüência de operação. Cada entidade é interpretada como um novo pulso de relógio, o que deve disparar a execução de uma nova instrução, de determinado tipo, conforme parâmetros do usuário. Cada instrução está associada a um tempo de execução (pulso de relógio) e pode estar associada também a pinos de entrada/saída, representando o acesso a dispositivos externos. Neste caso, a entidade será transferida a outro componente, que pode ser um barramento, uma porta de comunicação ou diretamente um dispositivo periférico. Ainda nesse caso, acessos a dispositivos externos podem ser síncronos ou assíncronos. No caso assíncrono, a entidade é repassada ao componente conectado ao pino, e o microcontrolador fica livre para executar outras instruções. Caso o acesso seja síncrono, o microcontrolador bloqueia a geração de novas instruções até que receba uma entidade por certo pino de entrada associado. A Figura 30 apresenta a estrutura geral do algoritmo para esse componente.

```
Inicio:
Pino_Entrada = SSD.Model.SIMAN.AttributeIndex('InputNumber')
SE (Pino_Entrada = 0) ENTÃO
        Nova_Instrucao = SSD.Model.SIMAN.EntityCreate;
        Novo_Pulso = SSD.Model.TNOW + (1/thisModule.aClockFrequency)*TimeUnitConvert(tuS,
        SSD.Model.BaseTimeUnit)
        SSD.Model.Mostrar_Usuarios(5,Microcontrolador.ID,entity,'Recebendo um novo pulso de clock.
        Próximo pulso no tempo '+ newPulse)
        SSD.Model.SIMAN.CalendarInsertEvent(newPulse, newEntityID, Microcontrolador.ID)
        SE (Microcontrolador.Cliclos_Faltando  > 0) ENTÃO
                Microcontrolador.Ciclos_Faltando = Microcontrolador.Ciclos_Faltando – 1
                SSD.Model.TraceExec(6,Microcontrolador.ID,entity,'Continue executing the current
                instruction')
                SE (Microcontrolador.aRemainingSends > 0) ENTÃO
                        enviou = true
                        Microcontrolador.aRemainingSends := Microcontrolador.aRemainingSends - 1
                        SSD.Model.SIMAN.EntitySendToModuleNumber(entity,
                        Microcontrolador.NextID[Microcontrolador.aSendOutputPin-1],0)
                 FIMSE
        SENAO
                SE (Microcontrolador.aExternalSignalWaiting = 0) ENTÃO
                        PROCEDIMENTO Verifica_Sinal_No_Pino()
                        SE (input = 0) ENTÃO
                                PROCEDIMENTO Executa_Nova_Instrução()
                        FIM SE
                 FIM SE
        FIM SE
FIM SE
SE (Pino_Entrada > 0) ENTÃO
        Atendeu = FALSO
        SE (Microcontrolador.Esperando_Sinal_Externo > 0) ENTÃO
                SE (Microcontrolador.Esperando_Sinal_Externo == Pino_Entrada) ENTÃO
                        Microcontrolador.Esperando_Sinal_Externo = 0
                        SSD.Model.SIMAN.CStatAddValue(Microcontrolador.Intervalo_Resposta_Estatistica,
                        (SSD.Model.TNOW – Microcontrolador.Tempo_Requisitado_Envio))
                        i = SSD.Model.SIMAN.QueueIndexID(i)
                        SSD.Model.SIMAN.Queue[i].RemoveElementByID(Entidade)
                FIM SE
        FIM SE
        SE (Microcontrolador.Esperando_Sinal_Externo = 0) ENTÃO
                PROCEDIMENTO Verifica_Sinal_No_Pino()
        FIM SE
FIM SE
SE NÃO Atendeu ENTÃO
         SSD.Model.TraceExec(6,thisModule.ID,entity,'Entity raimais waiting in queue')
FIM SE
(Continua na próxima página...)
```

```
(…continuação)
PROCEDIMENTO Verifica_Sinal_No_Pino()
Inicio:
I = 0
        Pino_Entrada = 0
        ENQUANTO (i < SSD.Model.SIMAN.QueuesCount) and (Pino_Entrada = 0) FAÇA
                Fila := SSD.Model.SIMAN.Queue[i];
                SE  (Fila.ModuleID = Microcontrolador.ID) and (Fila.Waiting.Count > 0) ENTÃO
                        ENQUANTO (Fila.Waiting.Count > 1) FAÇA
                                Fila.RemoveElementByRank(0);
                        FIM ENQUANTO
                        thisWait := Fila.RemoveElementByRank(0)
                        Pino_Entrada := Fila.ModuleInput + 1
                FIM SE
                i = i + 1
        FIM ENQUANTO
        SE (Pino_Entrada > 0) ENTÃO
                i = SSD.Model.SIMAN.AttributeIndex('Address')
                addr = trunc(SSD.Model.SIMAN.EntityAttribute[Fila.EntityID, i])
                i = 0
                ENQUANTO (i < Microcontrolador.Tabela_Instrucoes.Count) FAÇA
                  Linha_Tabela_Instrucao := Microcontrolador.Tabela_Instrucao[i];
                   SE (Linha_Tabela_Instrucao.Pino_Entrada == Pino_Entrada)  E
                  (SSD.Model.SIMAN.StringEvaluateFormula(Linha_Tabela_Instrucao.Address) == addr)
                  ENTÃO
                        Microcontrolador.Ciclos_Faltando =
                        Truncar(SSD.Model.SIMAN.StringEvaluateFormula(Linha_Tabela_Instrucao.Ciclos))
                         i = Microcontrolador.Tabela_Instrucao.Count
                  FIM SE
                   i = i + 1
                FIM ENQUANTO
FIM PROCEDIMENTO

PROCEDIMENTO Executa_Nova_Instrução()
                  Número = Aleatório(1)
                  Prob_Acumulada = 0
                  ENQUANTO (Prob_Acumulada < Número) FAÇA
                        Prob_Acumulada = Prob_Acumulada +
                        Microcontrolador.TabelaInstruções[i].Probabilidade
                        I = I + 1
                  FIM ENQUANTO
                  I = I - 1
                  CASO Microcontrolador.Tabela_Instrucoes[I].InstructionType DE
                  itLogicArit :
                  SSD.Model.SIMAN.CStatAddValue(thisModule.aAccessNumberToLogicAritCStatID,1);
                  itJumpGoto :
                  SSD.Model.SIMAN.CStatAddValue(thisModule.aAccessNumberToJumpGotoCStatID,1);
                  itBranch :
                  SSD.Model.SIMAN.CStatAddValue(thisModule.aAccessNumberToBranchCStatID,1);
                  itTimer :
                  SSD.Model.SIMAN.CStatAddValue(thisModule.aAccessNumberToTimerCStatID,1);
                  itWatchDog :
                  SSD.Model.SIMAN.CStatAddValue(thisModule.aAccessNumberToWatchDogCStatID,1);
itInternalMemory :
(Contínua na próxima página...)
```

```
(...Continuação)
                SSD.Model.SIMAN.CStatAddValue(thisModule.aAccessNumberToInternalMemoryCStatID,1);
                itPeriferal :
                 SSD.Model.SIMAN.CStatAddValue(thisModule.aAccessNumberToPeriferal,1);
                itBus :
                SSD.Model.SIMAN.CStatAddValue(thisModule.aAccessNumberToBusCStatID,1);
                FIM CASO
                Microcontrolador.Ciclos_Faltando = Microcontrolador.Tabela_Instrucoes[i].Ciclos
                SE (Microcontrolador.Tabela_Instruções[i].Pino_Saída > 0) ENTÃO
                        SE (Microcontrolador.Tabela_Instruções[i].Síncrono ==True) ENTÃO
                                Microcontrolador.Esperando_Sinal_Externo =
                                Microcontrolador.Tabela_Instruções[i].Pino_Entrada
                                Microcontrolador.Tempo_Requisitado_Envio = SSD.Model.TNOW
                        FIM SE
                SSD.Model.SIMAN.EntitySendToModuleNumber(Nova_Instrucao,Microcontrolador.NextID[in
                Tabela_Instrução.Pino_Saida-1],0);
FIM PROCEDIMENTO
```

Figura 30. Algoritmo do Microcontrolador

2.1.3.2 Memória EEPROM Externa

A memória EEProm externa é um componente muito utilizado, pois, devido a grandes rotinas e algoritmos mais complexos, a memória interna do microcontrolador pode não conseguir comportar um grande número de instruções e dados. O microcontrolador pode acessar a memória externa através de um barramento externo ou ser diretamente ligado a ela.

A Figura 31 apresenta o algoritmo de funcionamento proposto para esse componente.

```
Inicio:
ProbRandom = Aleatório(1)
fltProbAcc = 0
//Calcula probabilidade de leitura
ProbRead = SSD.Model.SIMAN.StringEvaluateFormula(EEProm.aProbabilityRead) / 100
fltProbAcc = ProbRead
//Calcula o tempo de acesso da memoria, avalia uma expressão e converte para a unidade de tempo padrão e
adiciona estatísticas.
SE (fltProbAcc > ProbRandom) ENTÃO
          TimeAccess = SSD.Model.SIMAN.StringEvaluateFormula(EEProm.aTimeAccessForRead)
           TimeAccess = TimeAccess * TimeUnitConvert(tuS, SSD.Model.BaseTimeUnit)
           SSD.Model.SIMAN.CStatAddValue(EEProm.aTimeAccessForReadCStatID,TimeAccess)
SENÃO
          TimeAccess = SSD.Model.SIMAN.StringEvaluateFormula(EEProm.aTimeAccessForWrite)
           TimeAccess = TimeAccess * TimeUnitConvert(EEProm.aAccessTimeUnit, SSD.Model.BaseTimeUnit)
           SSD.Model.SIMAN.CStatAddValue(EEProm.aTimeAccessForWriteCStatID,TimeAccess)
 FIM SE
SSD.Model.SIMAN.EntitySendToModuleNumber(entity,EEProm.NextID[0],TimeAccess)
```

Figura 31. Algoritmo para memória externa

2.1.3.3 Barramento Externo

O barramento externo é o caminho onde os componentes se comunicam. Sua função para simulação é, além de transmitir as entidades de um dispositivo para o outro, indicar a taxa de ocupação ou o número de acessos a dispositivos. Na Figura 32, o algoritmo de um barramento externo.

```
Inicio:
Barramento.aNumberAcessBus = Barramento.aNumberAcessBus + 1
delay := SSD.Model.SIMAN.StringEvaluateFormula(Barramento.aDelayTime) *
TimeUnitConvert(Barramento.aDelayTimeUnit, SSD.Model.BaseTimeUnit)
PARA I = 0 to Barramento.NextCount - 1 do begin
        SSD.Model.SIMAN.EntitySendToModuleNumber(entity,Barramento[i],delay);
END PARA;
```

Figura 32. Algoritmo do barramento externo

2.1.3.4 Periféricos

Periféricos englobam toda parte de entrada e saída de um sistema embarcado. Foram também classificados dois tipos de periféricos: síncronos e assíncronos. O primeiro tipo define que o microcontrolador deve esperar que a operação de E/S do periférico termine para continuar o processamento, enquanto o segundo tipo, o processamento do microcontrolador pode continuar normalmente.

2.1.3.4.1 Motor de Passo

Motor de passo é um periférico de saída utilizado com o projeto de sistemas embarcados que exige interação com dispositivos mecânicos, pois esse dispositivos transformam pulsos elétricos em energia mecânica. Sua função base na simulação é o tempo que leva para dar um passo, e o número de acessos que esse dispositivo teve. Se uma entidade segue ao motor de passo e o mesmo estiver em estado ocupado, (dando um passo), ela é descartada. A Figura 33 apresenta o seu algoritmo.

```
Inicio:
TimeAccess := SSD.Model.SIMAN.StringEvaluateFormula(MotorDePasso.aTimeAccessForStepperMotor);
TimeAccess := TimeAccess * TimeUnitConvert(MotorDePasso.aStepperMotorTimeUnit,
SSD.Model.BaseTimeUnit);
TimeAccess := TimeAccess + SSD.Model.SIMAN.StringEvaluateFormula(MotorDePasso.aTimeAccessToStepper)
                    * TimeUnitConvert(MotorDePasso.aStepperMotorTimeUnit, SSD.Model.BaseTimeUnit);
SSD.Model.SIMAN.CStatAddValue(MotorDePasso.aAccessNumberForStepperCStatID,1);
SSD.Model.SIMAN.CStatAddValue(MotorDePasso. aTimeForStepperCStatID,TimeAccess);
SSD.Model.SIMAN.EntitySendToModuleNumber(entity,thisModule.NextID[0],TimeAccess);
```

Figura 33. Algoritmo do Motor de Passo.

2.1.3.4.2 Porta Paralela/Serial

As portas paralelas e seriais possuem funções de enviar e receber dados através de um padrão de comunicação, obedecendo ao número de bits que são transmitidos por vez e sua velocidade. A diferença desses dispositivos, além da velocidade e taxa de bytes, é a forma de comunicação e preparo de envio de sinal. A Figura 34 ilustra o algoritmo proposto para porta paralela e a Figura 35 o algoritmo para serial:

```
Inicio:
 SE (PortaParalela.aEntitySends < PortaParalela.aMaxParallelSend) OU
     (PortaParalela.MaxParallelSend = -1)  ENTÃO
          PortaParalela.aEntitySends := PortaParalela.aEntitySends + 1
          newInstructionID := SSD.Model.SIMAN.EntityCreate
           newCreationTime := SSD.Model.TNOW +
          SSD.Model.SIMAN.StringEvaluateFormula(PortaParalela.aTimeSend) *
          TimeUnitConvert(PortaParalela.aParallelTimeUnit, SSD.Model.BaseTimeUnit)
          {Coleta Estatisticas do envio}
          SSD.Model.SIMAN.CStatAddValue(PortaParalela.aTimeParallelSendCStatID,SSD.Model.SIMAN.Strin
          gEvaluateFormula(PortaParalela.aTimeSend) * TimeUnitConvert(PortaParalela.aParallelTimeUnit,
          SSD.Model.BaseTimeUnit))
          SSD.Model.SIMAN.CStatAddValue(PortaParalela.aNumberParallelSendCStatID,1)
          SSD.Model.SIMAN.CalendarInsertEvent(newCreationTime, newInstructionID,
          PortaParalela.NextID[0])
          SSD.Model.TraceExec(3, PortaParalela.ID, entity, 'Arrival of Parallel Sends ' +
          IntToStr(newInstructionID) + ' scheduled for time ' + FloatToStr(newCreationTime))
FIM SE
SSD.Model.SIMAN.CStatAddValue(PortaParalela.aTimeParallelReceiveCStatID,SSD.Model.SIMAN.StringEval
uateFormula(PortaParalela.aTimeReceive) * TimeUnitConvert(thisModule.aParallelTimeUnit,
SSD.Model.BaseTimeUnit))
SSD.Model.SIMAN.CStatAddValue(PortaParalela.aNumberParallelReceiveCStatID,1)
SSD.Model.SIMAN.EntityDispose(entity)
```

Figura 34. Algoritmo da porta paralela

```
Inicio:
 SE (PortaSerial.aEntitySends < PortaSerial.aMaxSerialSend) OU
    (PortaSerial.MaxSerialSend = -1)  ENTÃO
        PortaSerial.aEntitySends := PortaSerial.aEntitySends + 1
        SE (RESTO(PortaSerial.aEntitySends) DE 8 = 0) ENTÃO
                {Formou um Byte, contabilizando Estatísticas}
                newInstructionID := SSD.Model.SIMAN.EntityCreate
                 newCreationTime := SSD.Model.TNOW +
                SSD.Model.SIMAN.StringEvaluateFormula(PortaSerial.aTimeSend) *
                TimeUnitConvert(PortaSerial.aSerialTimeUnit, SSD.Model.BaseTimeUnit)
                {Coleta Estatisticas do envio}
                SSD.Model.SIMAN.CStatAddValue(PortaSerial.aTimeSerialSendCStatID,SSD.Model.SIMAN.
                StringEvaluateFormula(PortaSerial.aTimeSend) *
                TimeUnitConvert(PortaSerial.aSerialTimeUnit, SSD.Model.BaseTimeUnit))
                SSD.Model.SIMAN.CStatAddValue(PortaSerial.aNumberSerialSendCStatID,1)
                SSD.Model.SIMAN.CalendarInsertEvent(newCreationTime, newInstructionID,
                PortaSerial.NextID[0])
                SSD.Model.TraceExec(3, PortaSerial.ID, entity, 'Arrival of Serial Sends ' +
                IntToStr(newInstructionID) + ' scheduled for time ' + FloatToStr(newCreationTime))
        FIM SE
 FIM SE
SSD.Model.SIMAN.CStatAddValue(PortaSerial.aTimeSerialReceiveCStatID,SSD.Model.SIMAN.StringEvaluate
Formula(PortaSerial.aTimeReceive) * TimeUnitConvert(thisModule.aSerialTimeUnit,
SSD.Model.BaseTimeUnit))
SSD.Model.SIMAN.CStatAddValue(PortaSerial.aNumberSerialReceiveCStatID,1)
SSD.Model.SIMAN.EntityDispose(entity)
```

Figura 35. Algoritmo da porta serial

2.1.1.4.3 Conversores A/D e D/A

Os conversores Analógicos/Digitais e Digitais/Analógicos são elementos de entrada e de saída, respectivamente. Possuem a característica de transformar um sinal em outro, para isso ocupam uma certa fatia de tempo. Esses componentes recebem sinais todo o tempo de funcionamento, porém somente converte tais sinais quando há sinal num determinado pino de entrada enviado por qualquer outro dispositivo (geralmente o microcontrolador). A Figura 36 mostra o algoritmo de funcionamento desses dois dispositivos.

```
Pino_Entrada = trunc(SSD.Model.SIMAN.AttributeValue['Entity.InputNumber'])
SE (Pino_Entrada = 2) ENTÃO
        ConversorADDA.aConverter = Verdadeiro
        SSD.Model.SIMAN.EntityDispose(entity)
FIM SE
SE (Pino_Entrada = 1) ENTÃO
        SE (ConversorADDA.aConverter) ENTÃO
        TimeAccess  = SSD.Model.SIMAN.StringEvaluateFormula(ConversorADDA.
        aTimeAccessForDAConverter)
        TimeAccess := TimeAccess * TimeUnitConvert(ConversorADDA.aDAConverterTimeUnit,
        SSD.Model.BaseTimeUnit)
        SSD.Model.SIMAN.CStatAddValue(ConversorADDA.aAccessNumberForDAConverterCStatID,1);
        SSD.Model.SIMAN.CStatAddValue(ConversorADDA.aTimeForDAConverterCStatID,TimeAccess);
        SSD.Model.SIMAN.EntitySendToModuleNumber(entity, ConversorADDA.NextID[0],TimeAccess);
SENÃO
        SSD.Model.SIMAN.EntityDispose(entity)
FIM SE
```

Figura 36. Algoritmo de conversores A/D e D/A

2.1.3.4.4 Displays

Dispositivos como LCDs e *display* de sete segmentos são simplesmente dispositivos de saída que informam ao utilizador, algumas informações sobre o andamento do sistema através de mensagens e luzes sinalizadoras. Nesse dispositivo, possui um controlador interno que realiza operações de escrita no *display*, portando, possui um tempo gasto no processamento dessa informação. A Figura 37 mostra seu comportamento através de um algoritmo.

```
Inicio:
TimeAccess := SSD.Model.SIMAN.StringEvaluateFormula(LCD.aTimeAccessForLCD);
TimeAccess := TimeAccess * TimeUnitConvert(LCD.aLCDTimeUnit, SSD.Model.BaseTimeUnit);
SSD.Model.SIMAN.CStatAddValue(LCD.aAccessNumberForLCDCStatID,1);
SSD.Model.SIMAN.CStatAddValue(thisModule.aTimeForLCDCStatID,TimeAccess);
SSD.Model.SIMAN.EntitySendToModuleNumber(entity,LCD.NextID[0],TimeAccess);
```

Figura 37. Algoritmo de *displays*

2.1.3.4.5 Leds

LEDs indicam ao usuário, o estado do sistema através de sua luz, portanto, sua função na simulação é indicar que uma instrução que programa ascender um LED passou por esse dispositivo. A Figura 38 mostra seu comportamento através de um algoritmo.

```
Inicio:
SSD.Model.SIMAN.CStatAddValue(LED.AccessNumberForLEDCStatID,1);
SSD.Model.SIMAN.EntityDispose(entity);
```

Figura 38. Algoritmo do leds

2.2. Componentes de Redes-em-Chip

2.2.1. Estrutura e Funcionamento de Redes-em-Chip

Uma rede-em-chip se caracteriza por um conjunto de canais e de roteadores, os quais podem ser vistos como núcleos (conjunto de componentes integrados num chip com função específica) dedicados à comunicação. Cada roteador está conectado a um ou mais núcleos, e possui um conjunto de canais para se comunicar com outros roteadores. (KARIM E DEY, 2002)

2.2.2. Componentes de Redes-em-Chip

Para que se possa representar o funcionamento de uma rede-em-chip através do SSD foram modeladas três classes que representam o funcionamento de uma rede-em-chip: TCanal, TRoteador e TNucleo. Essas classes estão ligadas entre si formando um nodo de uma rede.

As entidades geradas pelo simulador SSD representarão as mensagens, que têm dois atributos que indicam seu o endereço de origem e o endereço de destino. O funcionamento modelado dos principais componentes que compõe uma rede-em-chip são descritos a seguir:

2.2.2.1 Canal

Os canais de uma NoC têm a função semelhante ao barramento, ou seja, levar mensagens de um dispositivo a outro. Sua principal característica é a largura de banda, pois dela depende se uma mensagem tem que ser dividida em várias outras mensagens para poder chegar ao próximo dispositivo ou não. Caso essa fragmentação aconteça, há um custo adicional com a fragmentação no roteador e os vários envios de fragmentos da mensagem. A Figura 39 ilustra o algoritmo que representa um canal em rede-em-chip.

```
Inicio:
SSD.Model.SIMAN.CStatAddValue(Canal.aNumberChannelCStatID, 1);
SSD.Model.SIMAN.EntitySendToModuleNumber(entity, Canal.NextID[0],0);
```

Figura 39. Algoritmo de canal em rede-em-chip.

2.2.2.2 Núcleo

Em uma NoC, os núcleos (*cores*) são responsáveis pela funcionalidade do chip projetado. São dispositivos como microprocessadores, memórias entre outros controladores que são interligados por uma NoC. Seu funcionamento na simulação independe do tipo de circuito implementado, pois simplesmente ele cria e consome mensagens provenientes de outros núcleos. A Figura 40 ilustra o algoritmo proposto.

```
SE (Nucleo.aEntitySends < Nucleo.aMaxCoreSend) or (Nucleo.MaxCoreSend = -1) ENTÃO
        SSD.Model.SIMAN.AttributeValue['Entity.CoordX'] := Nucleo.aCoordXOrig
        SSD.Model.SIMAN.AttributeValue['Entity.CoordY'] := Nucleo.aCoordYOrig
        Nucleo.aEntitySends := Nucleo.aEntitySends + 1
        newInstructionID := SSD.Model.SIMAN.EntityCreate
        newTimeSend := SSD.Model.TNOW +
        SSD.Model.SIMAN.StringEvaluateFormula(Nucleo.aTimeBetweenSends) *
        TimeUnitConvert(Nucleo.aCoreTimeUnit, SSD.Model.BaseTimeUnit)
        SSD.Model.SIMAN.CStatAddValue(Nucleo.aNumberCoreCStatID,1)
        SSD.Model.SIMAN.CalendarInsertEvent(SSD.Model.TNOW + newTimeSend, newInstructionID,
        Nucleo.NextID[0])
        SSD.Model.TraceExec(3, Nucleo.ID, entity, 'Arrival of Core Sends ' + IntToStr(newInstructionID) + '
        scheduled for time ' + FloatToStr(newTimeSend))
FIM SE
 newCoreTime := SSD.Model.TNOW + SSD.Model.SIMAN.StringEvaluateFormula(Nucleo.aTimeCore) *
TimeUnitConvert(Nucleo.aCoreTimeUnit, SSD.Model.BaseTimeUnit)
SSD.Model.SIMAN.CStatAddValue(Nucleo.aTimeCoreCStatID, newCoreTime)
SSD.Model.SIMAN.CalendarInsertEvent(newCoreTime, newInstructionID, 0)
```

Figura 40. Algoritmo de núcleo em rede-em-chip.

2.2.2.3 Roteadores

Em uma NoC, os roteadores e núcleos são interligados através de canais de comunicação ponto-a-ponto unidirecionais e assíncronos (ZEFERINO & SUSIN, 2003). O roteador tem a função de encaminhar as mensagens pela rede. Um roteador é composto por um conjunto de filas e multiplexadores (responsáveis pelo chaveamento) e outros controladores realizam a comunicação

necessária para que as mensagens possam ser transferidas. Quando uma mensagem chega no roteador, ele verifica o destino da mensagem e busca qual canal de comunicação que mais aproxima a mensagem do núcleo de destino. A Figura 41 ilustra o algoritmo de um roteador genérico.

```
Inicio:
RouterX = Trunc(Roteador.aCoreCoordX)
RouterY = Trunc(Roteador.aCoreCoordY)
CoreX = Trunc(SSD.Model.SIMAN.AttributeValue['Entity.CoordXTo'])
CoreY = Trunc(SSD.Model.SIMAN.AttributeValue['Entity.CoordYTo'])
RoteadorIndex = 0
//Verifica se este é o roteador com o núcleo procurado
SE ((RouterX - CoreX) = 0) E ((RouterY - CoreY) = 0)  ENTÃO
        RoteadorIndex := 0
SENÃO
        //Decide Qual vai mandar a entidade
        SE ((RouterX - CoreX) = 0) ENTÃO
                //Manda Para N, antes Verifica
                SE (Roteador.aNCoordX >= 0) E (Roteador.aNCoordY >= 0)  ENTÃO
                        RoteadorIndex := 1;
         SE ((RouterY - CoreY) = 0) ENTÃO
                //Manda para E, antes Verifica
                SE (Roteador.aECoordX >= 0) E (Roteador.aECoordY >= 0) ENTÃO
                        RoteadorIndex := 4;
         SE ((RouterX - CoreX) < 0) ENTÃO
                //Manda Para S, antes Verifica
                SE (Roteador.aSCoordX >= 0) E (Roteador.aSCoordY >= 0) ENTÃO
                        RoteadorIndex := 2;
         SE ((RouterY - CoreY) = 0) ENTÃO
                //Manda para W, antes Verifica
                SE (Roteador.aWCoordX >= 0) E (Roteador.aWCoordY >= 0) ENTÃO
                RoteadorIndex := 3;
        FIM SE
        SSD.Model.SIMAN.CStatAddValue(Roteador.aNumberMessagesCStatID, 1);
         TimeWaiting := SSD.Model.SIMAN.StringEvaluateFormula(Roteador.aTimeRouter) *
        TimeUnitConvert(Roteador.aRouterTimeUnit, SSD.Model.BaseTimeUnit);
        SSD.Model.SIMAN.EntitySendToModuleNumber(entity,
        Roteador.NextID[RoteadorIndex],TimeWaiting);
```

Figura 41. Algoritmo de canal em rede-em-chip.

2.3. Simulação de Sistemas Embarcados no SSD

A criação de modelos no SSD é feita de forma gráfica, interconectando componentes que representam comportamentos específicos. Cada componente possui uma ou mais entradas e pode ter um ou mais saídas.

Assim, as conexões num modelo do SSD são todas unidirecionais, ou seja, não há conexões

de entrada e saída. Essa característica difere dos circuitos eletrônicos convencionais e dos simuladores específicos para sistemas embarcados, nos quais as conexões representam fios, que são intrinsicamente bidirecionais, para onde envia e de onde recebe dados. Neste caso, devem ser substituídos, no SSD, por duas conexões. Uma de envio e outra de entrada. Para melhor ilustrar a modelagem pretendida, a Figura 42 mostra um sistema embarcados feito pela ferramenta Proteus e na Figura 43, um sistema semelhante pretendido na ferramenta de simulação genérica SSD.

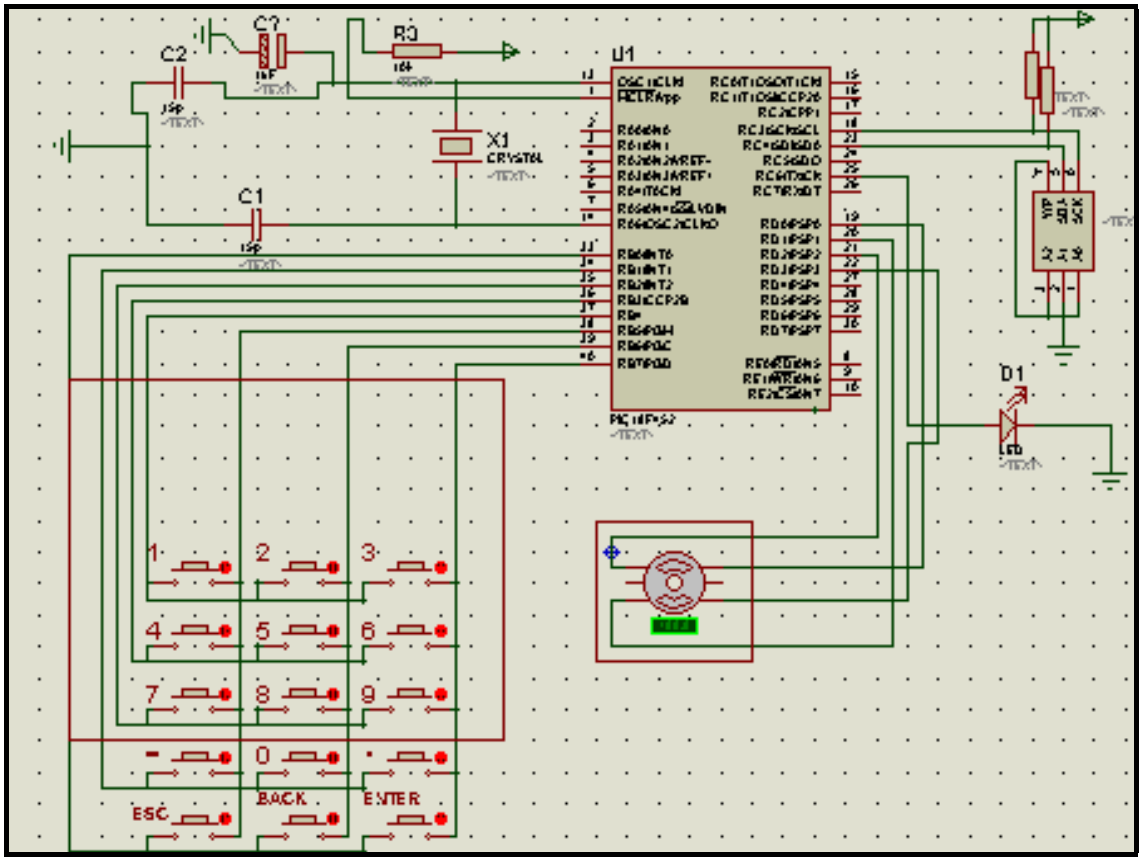

Figura 42. Sistema Embarcado no Proteus

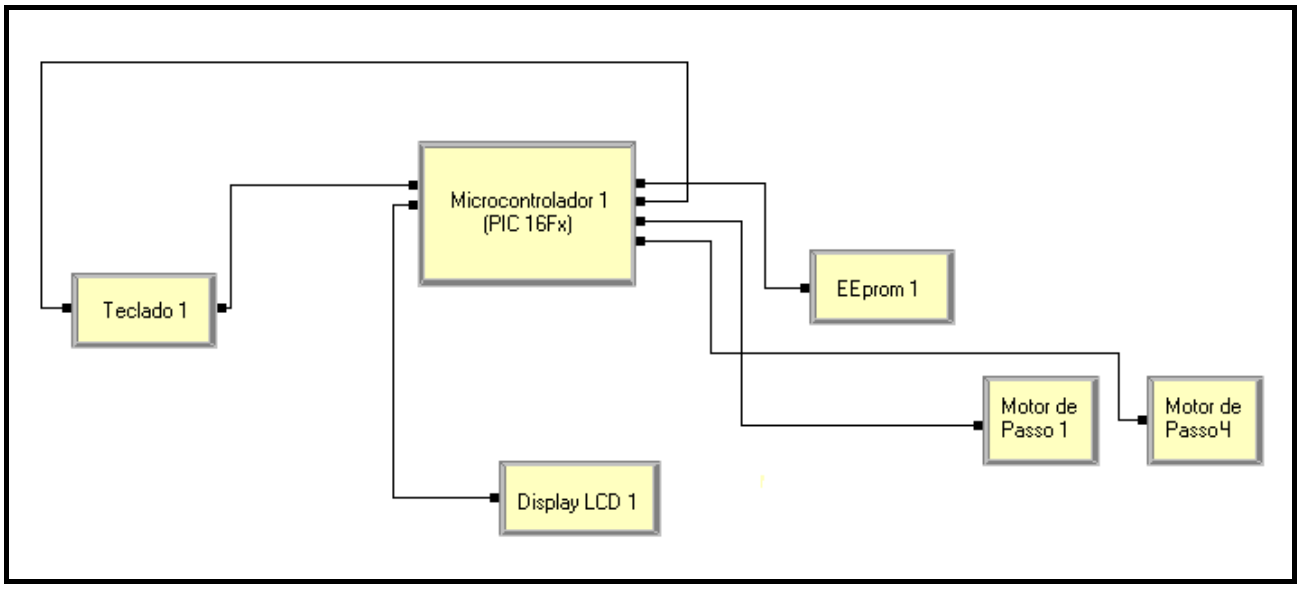

Figura 43. Sistema embarcado com SSD

3. IMPLEMENTAÇÃO E VALIDAÇÃO

Neste capítulo serão discutidas todas etapas para implementação dos componentes de sistemas embarcados e redes-em-chip para o SSD, assim como a validação de um modelo criado com esses componentes. Uma abordagem geral será descrita sobre as principais classes que compõem o simulador SSD, o arquivo de modelo (*template*) utilizado para fazer um componente e o arquivo que modela um sistema para simulação.

3.1 Estrutura para Implementação dos Componentes

De acordo com Stefanes (2002), o Simulador SSD e os componentes implementados foram desenvolvidos na linguagem *Object Pascal* utilizando a ferramenta *Borland Delphi* que ofereceu todo o suporte a orientação a objeto para o seu desenvolvimento.

3.1.1 Arquivo Modelo (*Template*)

De acordo com Stefanes (2002), para facilitar e até mesmo manter de forma organizada e sujeitando a menos erros na implementação dos componentes, foi criado um arquivo modelo (*Template*) que, utilizando a ferramenta *Borland Delphi* para compilação do código nele escrito,

gera um arquivo de biblioteca (DLL) que pode ser utilizado no SSD como um *plug-in*. Esse arquivo de *template* segue um padrão recomendado pelo autor contendo rotinas específicas (usersCreate, userRead, userSave, userVerify, userExecute) que devem ser preenchidas pelo desenvolvedor.

3.1.1.1 Definição das Características do Componente

Os componentes desenvolvidos são descendentes da classe TModule, que representa um módulo na interface do SSD. Esta classe é responsável pelos controles elementares de todos os módulos, como identificação única e conexão com outros módulos que compõem o modelo.

Quando se deseja que o componente gere entidades, é utilizada, ao invés da classe TModule, a classe TSourceModule, que é uma sub-classe de TModule e possui atributos exigidos pelo sistema para inclusão de entidades, sendo eles: FirstCreation e EntitiesPerCreation. A Figura 44 ilustra o preenchimento deste código no editor.

```
type
    TNewModule = Class(TModule)
    private
          {place here the attributes and methods of your new module}
          {attribute1: type1}
          {... your code here ...}
          aBusy : Boolean;
          aDelayTime: string;
          aDelayTimeUnit: TTimeUnit;
          aNumberAcessBus : LongInt;
          {procedure Methode1}
          {function Methode2}
          {... your code here ...}
          procedure UserCreate(novoId: word; novoNome, novoKind: string);
     public
          constructor Create(novoId: word; novoNome, novoKind: string);
          override;
     published
          {place in this region the property of this module that can be}
          {edit by the user}
          {property MyProperty1:type1  read attribute1  write attribute1}
          {... your code here ...}
          property Delay:String            read aDelayTime     write aDelayTime;
          property DelayTimeUnit:TTimeUnit read aDelayTimeUnit write aDelayTimeUnit;
     end;
```

Figura 43. Atributos e métodos inseridos na classe TNewModule

3.1.1.2 Informações Sobre o Componente

Além de representar um comportamento específico, cada componente deve também coletar informações sobre seu funcionamento para fins de animação, avaliação e análise estatística. As principais informações que devem ser preenchidas na implementação de um novo componente são descritas abaixo:

- MODULE_KIND: O nome do componente que será criado. Este nome será reconhecido pelo SSD quando fizer a leitura do arquivo através do componente que foi previamente carregado pelo sistema.;
- MODULE_AUTHOR: O nome do autor do componente;
- MODULE_VERSION: A versão e data atual do componente;
- MODULE_DESCRIPTION: Descrição referente o que ele vai representar;
- MODULE_IS_VISUAL: Se o componente será desenhado no modelo ou fica invisível;
- MODULE_IS_SOURCE: Especifica se este componente vai criar entidades, caso seja positivo, a classe TNewModule será descendente da classe TSourceModule;
- MODULE_IS_DISPOSE: Indica se o componente elimina entidades;
- MODULES_DEPENDENCES: Caso o componente precise de outro componente para funcionar, este é especificado aqui. Assim, ele será carregado automaticamente.

A Figura 44 ilustra a disposição dessas informações para preenchimento;

```
const
   {fill the followind informations about your new module}
   MODULE_KIND         = 'BUS';
   MODULE_AUTHOR       = 'Volnir dos S. Sobrinho';
   MODULE_VERSION      = '1.0.0' + ' in '+ '26/04/2004';
   MODULE_DESCRIPTION = 'Barramentos de um SE';
   MODULE_IS_VISUAL    = true;
   MODULE_IS_SOURCE    = false;
   MODULE_IS_DISPOSE   = false;
   MODULE_DEPENDENCES = '';
   {Se esse novo componente usar algum outro componente do SSD,}
   {fica estabelecida uma dependência,uma vez que esse outro   }
   {componente também deve ser incluído no SSD para que seu }
   {componente funcione. Apenas se houver dependências com outos}
   {componentes (DLLs), coloque o nome dessas DLLs acima, separadas}
   {por um ponto-e-vígula (;)}
```

Figura 44. Informações que caracterizam o componente

3.1.1.3 Implementação dos Métodos do Componente

A implementação do comportamento de um componente se dá através de métodos determinados e disparados pelo simulador SSD, desde o momento da abertura do arquivo que contém o modelo até sua execução propriamente dita. O SSD permite a implementação dos seguintes métodos:

- UserCreate: Esse é o primeiro métodos disparado pelo simulador, e permite ao componente inicializar seus atributos e suas estruturas internas necessárias para seu funcionamento. A Figura 45 ilustra a implementação desse método;

```
procedure TNewModule.UserCreate(novoId: word; novoNome, novoKind: string);
begin
   {initialize here the necessary attributes of your new module}
   {attribute1 := ''}
   {attribute2 := 0}
   {... your code here ...}
   aBusy := False;
   aDelayTime := '0';
   aDelayTimeUnit := tuS;
   aNumberAcessBus := 0;
end;
```

Figura 45. Implementação do método UserCreate

- UserRead: Esse método é executado quando um módulo desse tipo é carregado de um arquivo de modelo. Cada módulo possui uma linha no arquivo de modelo que descreve suas características. A Figura 46 ilustra a implementação desse evento;

```
procedure TModuleManager.UserRead(var thisModule:TNewModule; words:TStringList);
begin
    {coloque aqui o código para ler os atributos do seu componente}
    {os valores lidos do arquivos estão na StringList 'palavra', }
    {a partir de .Strings[10] }
    {observe os exemplos abaixo}
    {thisModule.atributo1 := words.Strings[10]}
    {thisModule.atributo2 := StrToInt(words.Strings[10])}
    {... your code here ...}
    thisModule.aDelayTime := words.Strings[10];
    thisModule.aDelayTimeUnit := IntToTimeUnit(StrToInt(words.Strings[11]));
end;
```

Figura 46. Implementação do método UserRead

- UserSave: Quando o usuário deseja salvar o modelo, esse método é disparado. Ele realiza a gravação dos dados relacionados ao módulo no arquivo de modelo. Para isso se utiliza o mesmo vetor a partir da posição dez. A Figura 47 ilustra a implementação desse evento;

```
procedure TModuleManager.UserSave(thisModule:TNewModule; var words: TStringList);
begin
    {coloque aqui o código para salvar os atributos do seu componente}
    {os valores que serão salvos em arquivo devem estar na StringList 'palavra'}
    {observe os exemplos abaixo}
    {words.Add(thisModule.atributo1)}
    {words.Add(IntToStr(thisModule.atributo2))}
    {... your code here ...}
    words.Add(thisModule.aDelayTime);
    words.Add(IntToStr(TimeUnitToInt(thisModule.aDelayTimeUnit)));
end;
```

Figura 47. Implementação do evento UserSave

- UserVerifySymbols: Alguns atributos que o usuário determina no modelo podem ser *strings* que contêm expressões lógicas, funções matemáticas e estatísticas, e que não são valores. Quando essa situação ocorre, o simulador possui uma rotina interna que interpreta e valida essas expressões. Atributos e variáveis também são verificados. A Figura 48 ilustra a implementação desse evento;

```
procedure TModuleManager.UserVerifySymbols(thisModule:TNewModule;
var verifyList:TStringList);
begin
    {coloque aqui a relação dos atributos "string" cujo valor é uma expressão}
    {e deve ser avaliada}
    {VerifySymbol(moduleName, description, expression, resultType, mandatory}
    {... your code here ...}
    VerifySymbol(thisModule.Name, 'Delay', thisModule.aDelayTime, cEXPRESSION,
    false);
end;
```

Figura 48. Implementação do evento UserVerifySymbols

- UserExecute: Este evento é disparado quando uma entidade chega ao respectivo componente. Toda lógica que determina o comportamento do componente é implementada neste evento. A Figura 49 ilustra a implementação desse evento;

```
procedure TModuleManager.UserExecute(var thisModule:TNewModule; entity:word);
var i: integer;
    delay: double;
begin
   {coloque aqui o código principal do seu componente, indicando o que ele deve
   fazer sempre que uma entidade chegar nele}
   {utilize "SSD." para obter acesso aos métodos do simulador}
   {... your code here ...}
   //Espera Tempo de Acesso
   thisModule.aNumberAcessBus := thisModule.aNumberAcessBus + 1;
   delay := SSD.Model.SIMAN.StringEvaluateFormula(thisModule.aDelayTime) *
   TimeUnitConvert(thisModule.aDelayTimeUnit, SSD.Model.BaseTimeUnit);
   for i := 0 to thisModule.NextCount - 1 do begin
      SSD.Model.SIMAN.EntitySendToModuleNumber(entity,thisModule.NextID[i],
      delay);
   end;
end;
```

Figura 49. Implementação do evento UserExecute

3.1.2 Estrutura para Criação dos Modelos no SSD

A ferramenta de simulação SSD permite o desenvolvimento dos modelos de forma amigável através de sua interface gráfica padrão *Windows*, a qual proporciona maior rapidez na modelagem de um sistema para simulação. No entanto, essa ferramenta permite a edição dos modelos de forma direta, através da edição do arquivo texto utilizando qualquer editor de textos presente no mercado. Esse recurso é muito interessante, pois, além de obter um conhecimento mais aprofundado de como está organizado o modelo, é indispensável caso o usuário apenas possua a ferramenta em modo texto.

3.1.2.1 Declaração de Módulos Internos

Os módulos internos que o modelo utilizará para a simulação devem ser declarados explicitamente no arquivo do modelo. A Figura 50 mostra a parte do arquivo que declara dos módulos internos do SSD, que são descritos a seguir:

```
# Internal Modules:

# ATTRIBUTE; Attribute Name 1; Attribute Name 2; ...; Attrinute
ATTRIBUTE;DEVICE
# ENTITYTYPE; ID; Name; InitialPicture; InitialVACost; InitialNV
# PROJECT; AnalystName; BaseTimeUnits; InitializeStatistics; Ini
PROJECT;Volnir S. Sobrinho;0;1;1;1;0;Sistema Embarcado;31;0;;0;0
# QUEUE; ID; Name; Type; Attribute
# RESOURCE; ID; Name; NumberBusy; Capacity; QueueName
# VARIABLE; Name; Dimension1; Dimension2; Value[1,1]; Value[2,1]
```

Figura 50. Declaração de módulos internos no modelo

- PROJECT: Nesta declaração são reunidas informações relativas ao projeto em geral, como informações sobre o analista, unidade de tempo base da simulação, número de replicações e tempo da simulação, entre outras;

- ATTRIBUTE: Aqui são declarados os atributos que serão utilizados no modelo de simulação;

- ENTITYTYPE: Esse módulo interno é responsável em determinar o comportamento de uma entidade e classificá-la em um tipo. Informações sobre custos e a figura gráfica que ela utiliza na simulação também são acrescentados nesse módulo interno;

- QUEUE: Todas as filas que serão utilizadas na simulação são declaradas neste módulo interno. Os parâmetros são um identificador, tipo (caso seja uma fila tipo FIFO, *First In First Out*) e o atributo que ela gerenciará;

- RESOURCE: Neste são inseridas informações sobre os recursos do sistema simulado. Suas propriedades são identificador, capacidade e fila associada;

- VARIABLE: As variáveis são declaradas nessa seção. O SSD suporta variáveis simples (inteiros, e ponto flutuante), como mais complexas (vetor de uma ou mais dimensões).

3.1.2.2 Declaração dos Componentes

Todos os componentes criados utilizando o arquivo modelo (*template*) de construção de componentes são declarados nesta seção. Na medida que se utiliza um componente, ele deve estar

indicado no arquivo seguindo um padrão estabelecido para leitura das propriedades do componente do arquivo. De Forma geral, a declaração dos componentes possuem a seguinte sintaxe:

< Modulo>;<Id_Entrada>;<Nome_Bloco;<Id_Saida>;<X>;<Y>;<Z>;<Res1>;<Res2>;<Res3>

Onde:

- Modulo: O componente a ser declarado. Ex: CREATE;

- Id_Entrada: Lista dos identificadores de entrada, ou seja, um conjunto único de números que indicam sua existência a fim de interligação. Quando o componente possui várias entradas elas são declaradas nesta seção separadas pelo caractere “,” (vírgula) sendo que o primeiro identificador representa o componente;

- Nome_Bloco: Nome do componente que o identifica visualmente caso exista mais de um módulo com o mesmo componente;

- Id_Saida: Indica quais blocos esse módulo estará conectado. Tendo as mesmas observações quanto a módulos de múltiplas entradas;

- X,Y,Z: Estas informações referem-se as coordenadas da posição onde será desenhado o bloco do módulo. Válido apenas na *interface* gráfica;

- Res1,Res2,Res3: Esse é um espaço reservado para mais três outros parâmetros que futuramente possam ser julgados necessários;

A declaração acima mencionada é a forma básica de declarar um módulo diretamente no arquivo de modelo do SSD. No entanto, como indica na etapa de desenvolvimento do componente, observa que são carregadas outras características do arquivo para cada componente específico. Para isso, após o espaço reservado de número três (Res3), o usuário especifica os parâmetros extras que foram implementados para determinado componente onde são definidos no *template* através dos eventos UserRead e UserWrite. A Figura 51 ilustra a declaração de quatro módulos:

```
# Plug-in Modules:

OSCILLATOR;108;Oscilator 1;118;0;50;0;-;-;-;10;30
MICROCONTROLER;109;Microcontroler 1;120;0;100;0;-;-;-;10;05;05;05
BUS;120;Barramento 1;130;0;150;0;-;-;-;10;05
EEPROM;130;Memoria EEPROM 1;118;0;200;0;-;-;-;05;10;10
DISPOSE;118;Dispose 1;118;400;50;0;-;-;-;1;1006
```

Figura 51. Declaração dos componentes no modelo

3.2 Modelos de Sistemas Embarcados Avaliados

Com o objetivo de comprovar o funcionamento e validação dos componentes para sistemas embarcados, foram utilizados dois modelos criados em uma ferramenta de simulação de sistemas embarcados específica, como o Proteus, já que essa ferramenta demonstrou resultados de simulação muito precisos, comparados com resultados reais.

Porém, devido às dificuldades de extrair dados estatísticos dos componentes de sistemas embarcados no Proteus, foi proposto, para comprovação dos resultados, um modelo mais robusto, com cerca de seis componentes e 8100 instruções e outro mais simples, com três componentes e 130 instruções. A Figura 52 mostra o modelo de um sistema para controle de um telescópio, modelado no Proteus, baseado no modelo apresentado por Silva (2003). A Figura 53 ilustra um modelo mais simples, um cronômetro digital também modelado nesta ferramenta.

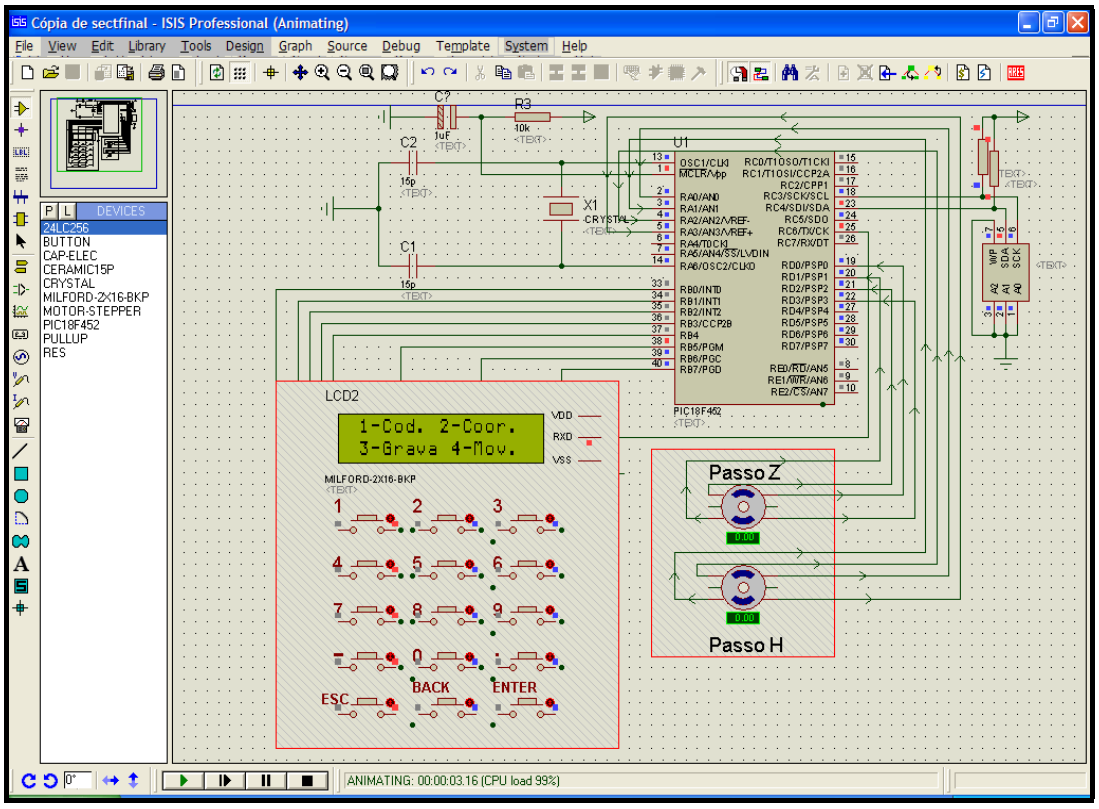

Figura 52. Modelo do Telescópio no Proteus

Fonte: Adaptado de Silva (2003)

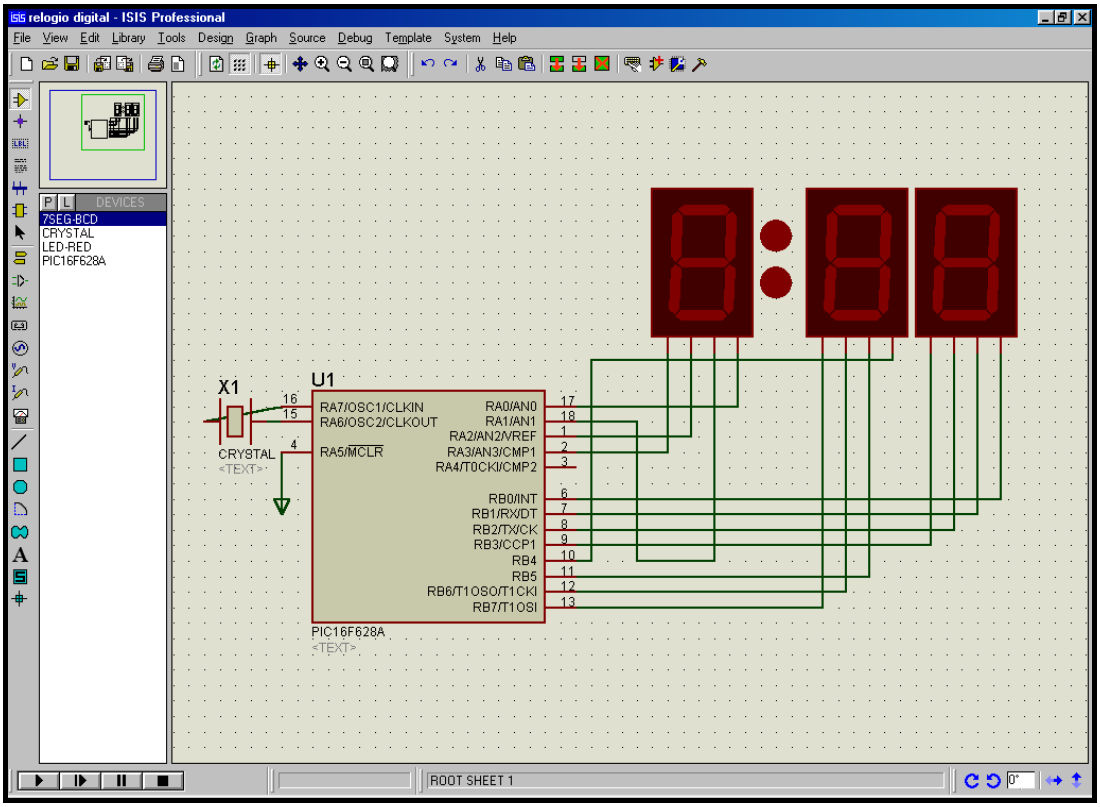

Figura 53. Modelo do cronômetro digital no Proteus

3.3 Validação de Modelos de Sistemas Embarcados

Para que os dois modelos feitos no Proteus pudessem ser modelados no SSD, foi necessário obter algumas informações a respeito dos modelos. Essas informações são descritas a seguir:

- Classificação das instruções: As instruções possuem categorias como instruções de acesso a memória, lógica/aritméticas, acessam *timers*, entre outras. Para que essa informação pudesse ser extraída, foi desenvolvido um *software* que faz a análise estática, ou seja, conta o número de instruções do código e as classifica, e dinâmica, que classifica através das instruções realmente executadas pelo microcontrolador. Isso é possível através da obtenção dos endereços das instruções quando estão no registrador PC (Program Couter). A Figura 54 ilustra esse *software* classificando o programa do cronômetro digital.

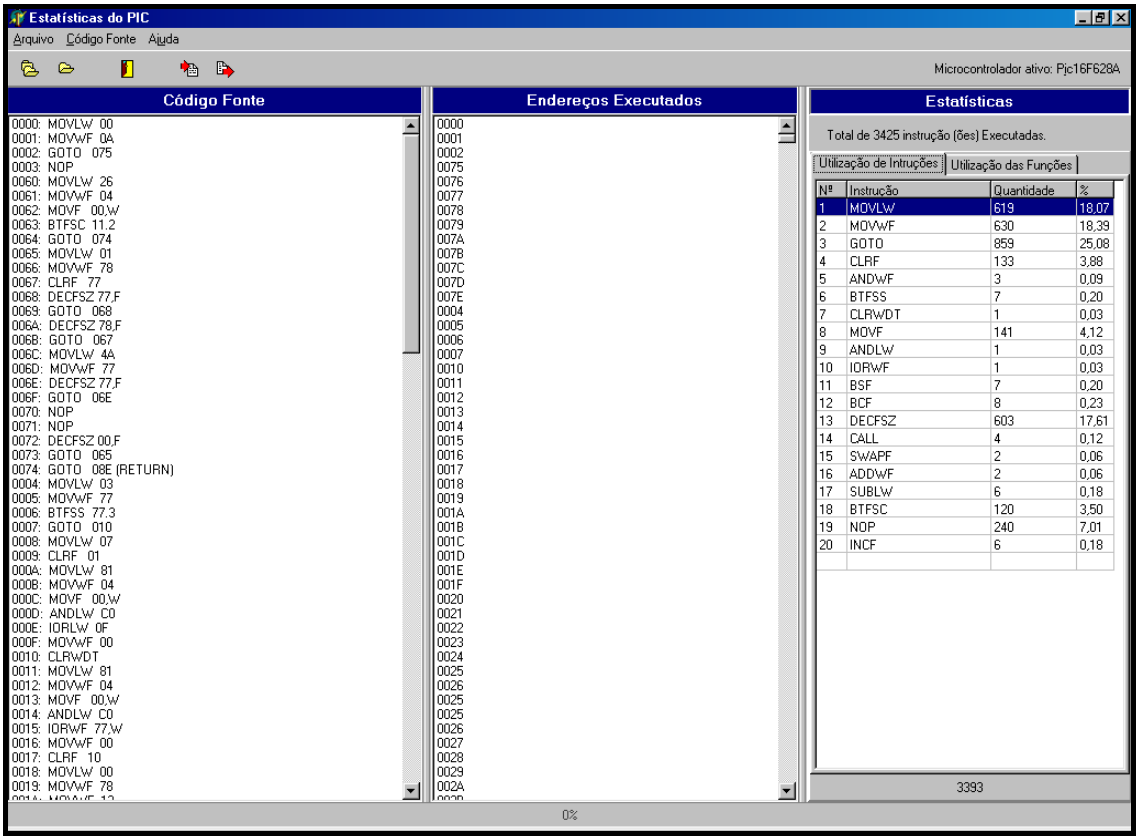

Figura 54. *Software* que classifica instruções do microcontrolador

- Extração de Estatísticas no Proteus: O Simulador SSD permite ao usuário conseguir várias informações estatísticas a respeito do modelo. Informações como taxa de utilização de um componente, a quantidade e tempo de acesso, entre outras. Como o proteus não foi projetado

para obter qualquer tipo de estatística, foi incorporado nos modelos, um componente no Proteus que permite incrementar um contador quando um sinal elétrico chega em seu pino de entrada. Esse componente se chama *Counter-Time*. A Figura 55 ilustra o formato desse componente, assim como sua conexão no Proteus. Com esse componente foi possível contar quantas vezes um determinado dispositivo foi acessado, assim permitindo a extração de estatísticas como a taxa de utilização.

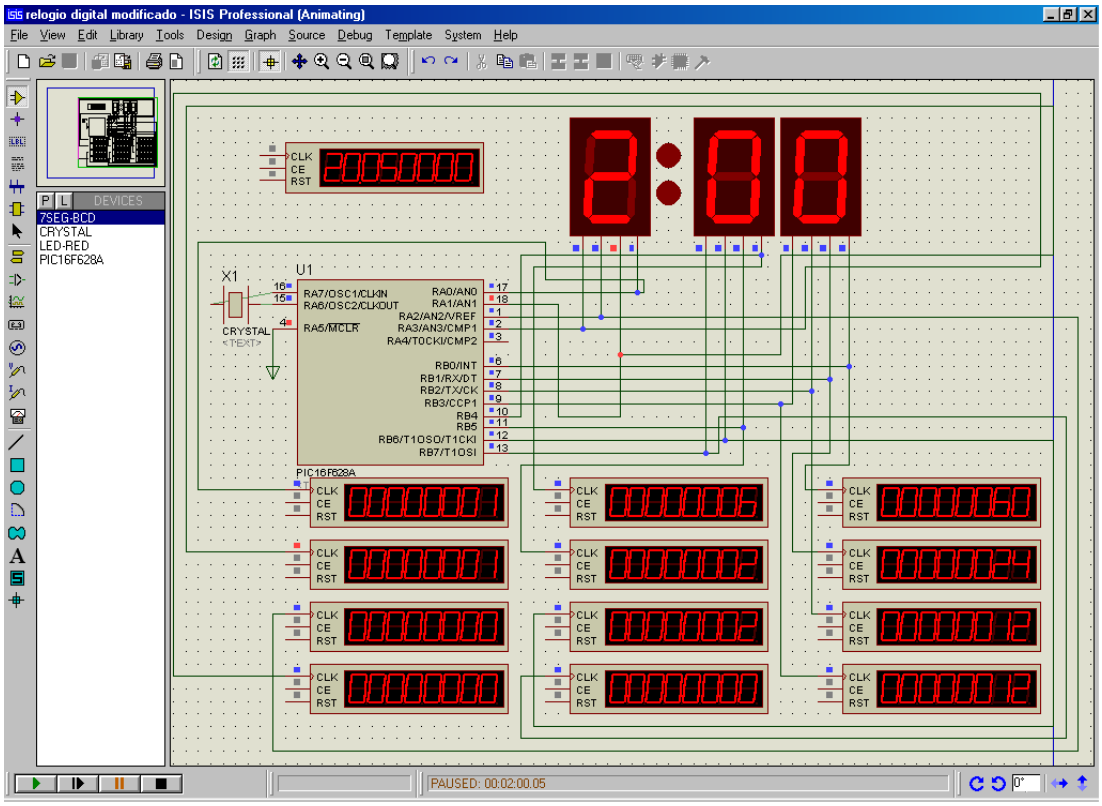

Figura 55. Componente *counter-time* interligado ao Proteus

3.3.1 Análise dos Resultados

Os dois modelos propostos foram submetidos a uma simulação na ferramenta Proteus onde foram obtidos parâmetros de utilização de alguns componentes. Como o Proteus é uma ferramenta de simulação determinística, a simulação é executada uma única vez. O SSD, por possuir componentes estocásticos em seus modelos, necessita que várias simulações do mesmo modelo sejam executadas, de forma a obter uma amostragem suficiente para a extração de resultados. A Figura 56 mostra as estatísticas obtidas no arquivo de saída do SSD. A Tabela 3 ilustra dos resultados das simulações feitas no Proteus e as simulações no SSD para o Modelo 01 (cronômetro digital). As demais tabelas ilustram os cálculos de validação. Os resultados referentes para o modelo

02 (telescópio) estão a partir da tabela 8. As estatísticas avaliadas foram o número de acessos para alguns componentes de cada modelo.

```
Replicação 1
Microcontrolador_1.AnswerTime: Average=0 Half=0 Min=0 Max=0 Sum=0 Num=0
Microcontrolador_1.AccessNumberInternalMemory: Average=1 Half=0 Min=1 Max=1 Sum=50562 Num=50562
Microcontrolador_1.TimeAccessInternalMemory: Average=0 Half=0 Min=0 Max=0 Sum=0 Num=0
Microcontrolador_1.AccessNumberTimer: Average=0 Half=0 Min=0 Max=0 Sum=0 Num=0
Microcontrolador_1.TimeAccessTimer: Average=0 Half=0 Min=1 Max=1 Sum=120 Num=120
Microcontrolador_1.AccessNumberWatchDog: Average=0 Half=0 Min=0 Max=0 Sum=0 Num=0
Microcontrolador_1.TimeAccessWatchDog: Average=0 Half=0 Min=0 Max=0 Sum=0 Num=0
Microcontrolador_1.AccessNumberBranch: Average=0 Half=0 Min=0 Max=0 Sum=0 Num=0
Microcontrolador_1.TimeAccessBranch: Average=0 Half=0 Min=0 Max=0 Sum=0 Num=0
Microcontrolador_1.AccessNumberBus: Average=0 Half=0 Min=0 Max=0 Sum=0 Num=0
Microcontrolador_1.AccessNumberLogicArit: Average=1 Half=0 Min=1 Max=1 Sum=62955 Num=62955
Microcontrolador_1.AccessNumberJumpGoto: Average=1 Half=0 Min=1 Max=1 Sum=9509 Num=9509
Microcontrolador_1.AccessNumberPeriphiral: Average=1 Half=0 Min=1 Max=1 Sum=3180 Num=3180
Display_Un.AccessNumberForDisplay7: Average=1 Half=0 Min=1 Max=1 Sum=104 Num=104
Display_Dz.AccessNumberForDisplay7: Average=0 Half=0 Min=0 Max=0 Sum=12 Num=12
Display_Hr.AccessNumberForDisplay7: Average=0 Half=0 Min=0 Max=0 Sum=2 Num=2
```

Figura 56. Estatísticas obtidas no arquivo de saída do SSD.

Análise do Modelo 1 (cronômetro digital)

Tabela 3. Resultados da Simulação do Modelo 01 no Proteus

Parâmetros do Modelo		
Quantidade de Acessos ao Display UN	**Quantidade de Acessos ao Display DZ**	**Quantidade de Acessos ao Display HR**
108	10	2

Tabela 4. Resultados das Simulações do Modelo 01 no SSD

Simulação	Parâmetros do Modelo		
	Quantidade de Acessos ao Display UN	**Quantidade de Acessos ao Display DZ**	**Quantidade de Acessos ao Display HR**
1	110	12	2
2	128	6	1
3	110	10	2
4	101	12	3
5	104	10	0
6	93	11	2
7	125	9	3
8	132	6	1
9	100	9	2
10	104	10	0
11	109	9	3
12	100	12	1
13	102	9	1
Média	**109,077**	**9,615**	**1,615**
Desvio	**12,003**	**1,981**	**1,044**

Análise da quantidade de acessos ao display UN

Tabela 5. Análise da quantidade de acessos ao display Un

Intervalo de Confiança (95%):	100,554 a 117,599
Semi-Intervalo de Confiança (e_0):	8,523
Valor da Variável de Teste (t):	0,323
Probabilidade da Variável de Teste (-t a +t):	87,595%

O intervalo de confiança da quantidade média de acessos ao display engloba o valor obtido no modelo do Proteus (108), indicando que o mesmo faz parte da estimativa desse parâmetro, feita a

partir do modelo do SSD. A probabilidade da variável de teste indica que os resultados dos dois modelos são equivalentes com 87,595% de confiança ou menos (erro máximo de 12,405%). Essa probabilidade corresponde à área sob a curva da distribuição *t de Student* (uma vez que a variância populacional é desconhecida e o tamanho da amostra é inferior a 30 elementos) entre os valores de –t até +t (eixo x, variável de teste), onde as diferenças entre as médias (Proteus e SSD) é mais próxima de zero.

Análise da quantidade de acessos ao display DZ

Tabela 6. Análise da quantidade de acessos ao display Dz

Intervalo de Confiança (95%):	8,209 a 11,022
Semi-Intervalo de Confiança (e_0):	1,406
Valor da Variável de Teste (t):	0,700
Probabilidade da Variável de Teste (-t a +t):	74,859%

O intervalo de confiança da quantidade média de acessos ao display engloba o valor obtido no modelo do Proteus (10), indicando que o mesmo faz parte da estimativa desse parâmetro feita a partir do modelo do SSD. A probabilidade da variável de teste indica que os resultados dos dois modelos são equivalentes com 74,859% de confiança ou menos (erro máximo de 25,141%).

Análise da quantidade de acessos ao display HR

Tabela 7. Análise da quantidade de acessos ao display Hr

Intervalo de Confiança (95%):	0,874 a 2,357
Semi-Intervalo de Confiança (e_0):	0,741
Valor da Variável de Teste (t):	1,328
Probabilidade da Variável de Teste (-t a +t):	60,437%

O intervalo de confiança da quantidade média de acessos ao display engloba o valor obtido no modelo do Proteus (2), indicando que o mesmo faz parte da estimativa desse parâmetro feita a partir do modelo do SSD. Entretanto, o semi-intervalo de confiança obtido é muito amplo (45,88% da média), pois a quantidade de amostras foi pequena. Seriam necessárias 274 simulações para que o semi-intervalo de confiança fosse reduzido a 10% da média (por exemplo), produzindo resultados

mais precisos. A probabilidade da variável de teste indica que os resultados dos dois modelos são equivalentes com 60,437% de confiança ou menos (erro máximo de 39,563%).

Os resultados da análise dos modelos, para os três parâmetros coletados, indicam que a equivalência (confiança) entre os modelos varia de 60,437% a 87,595% (nível de confiança médio de 74,297%). Ou seja, poderia-se obter, em princípio, com modelos de alto nível, parâmetros equivalentes aos obtidos posteriormente em modelos específicos, com erro médio (nível de significância) de 25,703% (100% - 74,297%).

Análise do Modelo 2 (telescópio)

Tabela 8. Resultados da Simulação do Modelo 02 no Proteus.

Parâmetros do Modelo			
Quantidade de Acessos ao Motor H	**Quantidade de Acessos ao Motor Z**	**Quantidade de Acessos ao LCD**	**Quantidade de Teclas Acionadas**
11	13	19	30

Tabela 9. Resultados das Simulações do Modelo 02 no SSD

Simulação	**Parâmetros do Modelo**			
	Quantidade de Acessos ao Motor H	**Quantidade de Acessos ao Motor Z**	**Quantidade de Acessos ao LCD**	**Quantidade de Teclas Acionadas**
1	8	5	42	30
2	10	20	31	39
3	9	13	19	23
4	7	58	21	55
5	13	15	8	35
6	10	2	75	32
7	9	17	9	28
8	12	15	8	31
9	8	14	40	72

10	10	13	20	44
11	8	8	10	15
12	13	15	12	23
13	10	10	10	25
Média	**9,769**	**15,769**	**23,462**	**34,769**
Desvio	**1,922**	**13,609**	**19,471**	**15,145**

Análise do Número de Acessos do Motor de Passo H

Tabela 10. Análise do número de acessos ao motor de passo H

Intervalo de Confiança (95%):	8,405 a 11,134
Semi-Intervalo de Confiança (e_0):	1,364
Valor da Variável de Teste (t):	2,309
Probabilidade da Variável de Teste (-t a +t):	51,976%

O intervalo de confiança da quantidade média de acessos ao display engloba o valor obtido no modelo do Proteus (11), indicando que o mesmo faz parte da estimativa desse parâmetro feita a partir do modelo do SSD. A probabilidade da variável de teste indica que os resultados dos dois modelos são equivalentes com 51,976% de confiança ou menos (erro máximo de 48,024%).

Análise do Número de Acessos ao Motor de Passo Z

Tabela 11. Análise do número de acessos ao motor de passo Z

Intervalo de Confiança (95%):	6,107 a 25,432
Semi-Intervalo de Confiança (e_0):	9,662
Valor da Variável de Teste (t):	0,734
Probabilidade da Variável de Teste (-t a +t):	73,861%

O intervalo de confiança da quantidade média de acessos ao display engloba o valor obtido no modelo do Proteus (11), indicando que o mesmo faz parte da estimativa desse parâmetro feita a partir do modelo do SSD. Entretanto, o semi-intervalo de confiança obtido é muito amplo (61,272%

da média), e seriam necessárias 489 simulações para que o semi-intervalo de confiança fosse reduzido a 10% da média, produzindo resultados mais precisos. A probabilidade da variável de teste indica que os resultados dos dois modelos são equivalentes com 73,861% de confiança ou menos (erro máximo de 26,139%).

Análise do Número de Acessos LCD

Tabela 12. Análise do número de acessos ao LCD

Intervalo de Confiança (95%):	9,637 a 37,286
Semi-Intervalo de Confiança (e_0):	13,825
Valor da Variável de Teste (t):	0,826
Probabilidade da Variável de Teste (-t a +t):	71,241%

O intervalo de confiança da quantidade média de acessos ao display engloba o valor obtido no modelo do Proteus (19), indicando que o mesmo faz parte da estimativa desse parâmetro feita a partir do modelo do SSD. Entretanto, o semi-intervalo de confiança obtido é muito amplo (58,925% da média), e seriam necessárias 452 simulações para que o semi-intervalo de confiança fosse reduzido a 10% da média, produzindo resultados mais precisos. A probabilidade da variável de teste indica que os resultados dos dois modelos são equivalentes com 71,241% de confiança ou menos (erro máximo de 28,759%).

Análise do Número de Teclas Acionadas

Tabela 13. Análise do número de teclas acionadas

Intervalo de Confiança (95%):	24,016 a 45,522
Semi-Intervalo de Confiança (e_0):	10,753
Valor da Variável de Teste (t):	1,135
Probabilidade da Variável de Teste (-t a +t):	63,918%

O intervalo de confiança da quantidade média de acessos ao display engloba o valor obtido no modelo do Proteus (30), indicando que o mesmo faz parte da estimativa desse parâmetro feita a partir do modelo do SSD. Entretanto, o semi-intervalo de confiança obtido é muito amplo (30,927% da média), e seriam necessárias 125 simulações para que o semi-intervalo de confiança fosse

reduzido a 10% da média, produzindo resultados mais precisos. A probabilidade da variável de teste indica que os resultados dos dois modelos são equivalentes com 63,918% de confiança ou menos (erro máximo de 36,082%).

Os resultados da análise dos modelos, para os quatro parâmetros coletados, indicam que a equivalência (confiança) entre os modelos varia de 51,976% a 73,861% (nível de confiança médio de 65,249%). Ou seja, poderia-se obter, em princípio, com modelos de alto nível, parâmetros equivalentes aos obtidos posteriormente em modelos específicos, com erro médio (nível de significância) de 34,751% (100% - 65,249%).

4. CONSIDERAÇÕES E RECOMENDAÇÕES

Este trabalho surgiu da necessidade de verificar se é possível obter métricas de desempenho sobre sistemas embarcados através de simulação de alto nível, sem que seja necessário especificar completamente tal sistema para prototipá-lo ou mesmo simulá-lo em ferramenta específica (simulação de baixo nível). Com isso poderia-se obter indícios do atendimento dos requisitos ou não de tal sistema num tempo menor, evitando o projeto de um sistema embarcado com componentes que, posteriormente, verificaria-se não serem suficientes para obter o desempenho inicialmente planejado.

Nesse sentido, neste trabalho foram implementados componentes parametrizáveis que representam o comportamento de alto nível dos principais circuitos utilizados em sistemas embarcados. Também foram desenvolvidos dois modelos distintos que usam os componentes criados, e esses modelos foram comparados a modelos equivalentes representados num simulador específico para circuitos eletro-eletrônicos (Proteus/ISIS). Deste modo, os objetivos propostos foram alcançados.

Verificou-se que a necessidade de ter o projeto totalmente especificado para poder ser avaliado em simuladores específicos pode não ser a opção mais adequada, dependendo do nível de confiança desejado para os resultados. Conforme as simulações realizadas, os modelos de alto-nível e específico mostraram ser equivalentes com confianças que variaram de 51,976% a 87,595%.

O modelo que não possuía interação com o usuário e tinha comportamento determinístico (modelo 1), apresentou nível de confiança médio de 74,297%, enquanto o modelo com comportamento que depende da ação do usuário (modelo 2), apresentou nível de confiança médio de 65,249%. Disso conclui-se que a ação do usuário pode alterar de maneira significativa a freqüência de execução das instruções num sistema embarcado, diminuindo a confiança estatística num modelo de alto nível.

Ressalta-se que os modelos de alto-nível foram desenvolvidos para representar sistemas embarcados dos quais já se conhecia o código executável, que foi base para a tabela que representa a freqüência dinâmica execução de instruções no componente desenvolvido para o

microcontrolador. Desta forma, os modelos de alto nível desenvolvidos neste trabalho podem ter uma freqüência dinâmica de execução de instruções mais próxima do real do que seria conseguido com modelos de alto nível desenvolvidos sem o conhecimento prévio do código executável do sistema o qual representam.

Ou seja, numa aplicação real, os modelos de alto nível, que seriam desenvolvidos antes da implementação completa do sistema embarcado, devem estimar as freqüências das instruções que devem ser executadas após o sistema ser implementado, pois elas não estarão disponíveis a priori, como foi o caso neste trabalho, necessário para a validação estatística dos modelos. Isso significa que, desconhecendo a freqüência real de execução das instruções, o modelo desenvolvido em alto nível pode ter nível de confiança menor do que os modelos apresentados neste trabalho.

Em relação as principais dificuldades encontradas no desenvolvimento deste trabalho, destacam-se: (i) A necessidade de aprendizado de novos conteúdos, não contemplados no Curso, como sistemas embarcados e as ferramentas de simulação utilizadas; (ii) a extração de estatísticas dos simuladores específicos para sistemas embarcados (Virtual Breadboard e Proteus/ISIS); (iii) a análise dinâmica das instruções de um software embarcado que requer interação com o usuário (comportamento não determinístico); (iv) problemas na implementação do simulador SSD, que passou por várias adaptações para suportar os novos componentes desenvolvidos; e (v) problemas com a linguagem de programação Delphi, que apresentou instabilidade na depuração do código.

Sugere-se ainda alguns avanços e melhoramentos neste trabalho: (i) Validar vários componentes que não foram utilizados nos modelos desenvolvidos, incluindo os componentes de redes em chip; (ii) permitir que o simulador SSD execute em paralelo, diminuindo o tempo total de simulação; (iii) representar aspectos de consumo de energia nos componentes simulados.

BIBLIOGRAFIA

ALTERA. Max+plus II. Disponível em <http://www.altera.com>. Acesso em: 04 de nov. 2003.

BANKS, J.; CARSON, J. S. II. **Discrete event system simulation.** New Jersey: Ed. Prentice Hall, 2.ed., 1994. 548p.

BARBACENA, I.L. **Display LCD**. Disponível em: <http://www.geocities.com/CapeCanaveral/6744/p_04.html >. Acesso em: 02 jun. 2003.

BIRTWISTLE, G. M. **Discrete event modelling on simula**. McMillan, 1979. 234p.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I**. UML. Guia do usuário:** O mais avançado tutorial sobre Unified Modeling Language (UML). Rio de Janeiro: Campus, 2000. p. 134-158.

COSTA. A. **Motores de passo**. Disponível em: < http://www.mrshp.hpg.ig.com.br/rob/m_passo.htm/>. Acesso em: 15 mai. 2003.

COSTA, I. **Proposta de sistematização da simulação para fabricação em lotes**. Dissertação de mestrado no programa de Pós-Graduação em Engenharia de Produção, Itajubá, UNIFEI, 2000.

DE MICHELE, N.A. **Systems modeling and computer simulation.** New York: Marcel Dekker. 2. ed. 1995. 345 p.

DEVETSIKIOTIS, M. **Simulation of rare events in communications networks**. IEEE Comunications Magazine. Ago. 1998.

DE MICHELE, Jarry. **Simulation a problem-solving approach.** Reading: Addison-Wesley. 1995. p 234-250.

DORM, J.**Conceitos de Motores de Passo.** Disponível em: http://www.geocities.com/CollegePark/Dorm/8863/motordepasso.htm/>. Acesso em: 13 mai. 2003.

DUATO, J. et al.: **Interconnection Networks: An engineering approach. los alamitos**: IEEE Computer Society Press. 1997. 515p.

DUTTA, S.; JENSEN, R.; RIECKMANN, A. **Viper: A multiprocessor SoC for advanced set-top box and digital TV systems. IEEE Design & Test of Computers**, v.18, n. 5. p.21-31, set./out, 2001.

EMSHOFF J.R.; SISSON, R.L. **Design and use of computer simulation models.** New York: MacMillan, 1970. 150p.

EVANS, G. W; WALLACE, G.F.; SUTHERLAND, G.L. **Simulation using digital computers.** Englewood Cliffs, N.J.: Prentice-Hall, 1967. p.45-48.

FERREIRA, R.L. **Simulação de sistemas**. Material Didático. Engenharia de Transportes. Instituto Militar de Engenharia, Rio de Janeiro. 2002.

FREITAS FILHO, P.J. **Introdução à modelagem e simulação de sistemas**. Florianópolis: Visual Books, 1.ed. 2001. 322 p.

FURLAN, J.D.. **Modelagem de objetos através da UML**: Análise e projeto orientados a objeto. São Paulo: Makron Books, 1998. 329 p.

GIMENES, S. **A família de microcontroladores 8051 da intel**. Disponível em: <http://www.malbanet.com.br/professorelmo/microcon.htm> . Acesso em: 13 abr. 2003.

HILL, D.R.C. **Object-oriented analysis and simulation**. Harlow: Addison-Wesley, 1996. 33 p.

KARIM, F.N.A; DEY, S. **An Interconnect architecture for networking systems on chips**. IEEE Micro, v.22, n.5, set./out. 2002. p 36-45.

LAW, A.M. **Simulation modeling and analysis**. Boston: McGraw-Hill, 3. ed. 2000. 56 p.

MAGALHÃES, A.N; LIMA, A.C.P. **Noções de probabilidade e estatística**. São Paulo: EDUSP, 4 ed. 2002.

REIS, M. **Introdução à estatística para ciências biológicas.** Apostila de aula. UFSC, 1998.

MATIC, N.; ANDRIC, D. **The PIC controler: book 1 for intermediate**. São Paulo: Makron Books, 3.ed. 2001.

MILONE, G; ANGELINI, F. **Estatística geral:** Amostragem, distribuições amostrais, decisão estatística. São Paulo: Atlas, 2.ed. 1993.

MENESES. R. **Técnicas de fabricação de motores**. Disponível em: http://users.hotlink.com.br/rmenezes/informa/mpasso/mpasso.htm. Acesso em: 17 mai. 2003.

MORIMOTO, C.E. **Curso de hardware completo**. Disponível em: <http://www.guadohardware.net/>. Acesso em: 05 mai. 2003.

NICOLOSI, D.E.C. **Microcontrolador 8051 detalhado**. São Paulo: Érica, 4.ed. 2003. p 30-56.

PAPADOPOULOS, H.T., **Queueing theory in manufacturing systems analysis and design**, Editora Chapman & Hall, 1994. 267 p.

PARAGON. **Paragon Tecnologia LTDA**. Disponível em: <http://www.paragon.com.br/>. Acesso em: 17 abr. 2003.

PEDGEN, C. D.; SHANNON, R. E.; SADOWSKI, R. P**. Introduction to simulation using Siman.** New York: McGraw-Hill, 2**.**ed. 1995.

PRITSKER, A. A. B. **Compilation of definitions of simulation. Simulation 33**. n.33, 1979. p 61-63.

ROCHA, J.G.C; NETO, F. **Journal of statistical computation and simulation**, 58, n.1, 1997. p 21-35.

ROSH, W.L. **Desvendando o hardware do PC.** Rio de Janeiro: Campus, 2.ed. 1996. p 15-18.

SALIBY, E. **Repensando a simulação: A amostragem descritiva**. São Paulo: Atlas, 1989. 98 p.

SHANNON, R.E. **Systems simulation: The art and the science**, Prentice-Hall Englewood Cliffs, New Jersey. 1975. p 150-154.

SCHRIBER, T. J. **An introduction to simulation using GPSS/H**. New York: John Wiley & Sons, 1991. 30 p.

SILVA, M. R. **Sistema embarcado para controle de telescópio.** Trabalho de conclusão de curso. Itajaí, Univali, 2003.

SILVEIRA, E. D; **Microcontroladores e automação: Panorama atual e perspectivas futuras**. Disponível em: http://www.malbanet.com.br/professorelmo/microcon.htm. Acesso em: 20 abr. 2003.

SOARES, L.F.G. **Modelagem de simulação discreta de sistemas**. Rio de Janeiro: Campus, 1.ed. 1992.

SOUZA, D.J; LAVINIA, N. C. **Conectando o PIC – Recursos avançados.** São Paulo: Érica, 2003. p 22-57.

STANKOVIC, J.A. **Real time computing**. Byte, Aug. 1992. p 155-160.

STEFANES, R. **Ferramenta de simulação discreta com ênfase em redes locais.** Trabalho de conclusão de curso. Itajaí, Univali, 2002.

TANENBAUM, A.S. **Organização estruturada de computadores.** Rio de Janeiro: Campus, 3.ed. 1999. 210 p.

TOLEDO, N. **Introdução a arquitetura de computadores.** Rio de Janeiro: Campus, 1.ed. 1994. 35 p.

TORRES, M.C.S. **O uso da simulação em uma das perspectivas do balanced scorecard.** 1998. 45f. Dissertação (Mestrado) - Instituto Militar de Engenharia, Rio de Janeiro. 1998.

TRIOLA, M.F. **Introdução à estatística**. Rio de Janeiro: Livros técnicos e científicos, 7.ed. 1998. 410 p.

VENDRAMIM, J.T. **Projeto de redes de computadores**. Notas de aula. CEFET, Curitiba, 2001. 89 p.

VELLOSO, F. C. **Informática: conceitos básicos.** Rio de Janeiro: Editora Campus, 1994. p 23-24.

WERKEMA, M.C.C., AGUIAR, S. **Planejamento e análise de experimentos**. Belo Horizonte: FCO, 1996.

WOLF, J. **Embeded systems**. Harlow: Addison-Wesley, 2002.

ZEIGLER, B.R. **Simulation methodology and model manipulation**. In: Shingh M.G. 1.ed. 1987. 23 p.

ZEFERINO, C.A., SUSIN, A.A. : **SoCIN: A parametric and scalable network-on-chip. In: symposium on integrated circuits and systems**, n.16, 2003, São Paulo. Los Alamitos: IEEE Computer Society Press. 2003. p 3-6.