UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
DEPARTAMENTO DE ELETRôNICA E MICROELETRôNICA

Modi -
Um Ambiente de Desenvolvimento para Aplicações Voltadas a Sistemas Dedicados

Marcos Antonio Vieira da Silva

ORIENTADOR: Prof.Dr. Elnatan Chagas Ferreira

BANCA EXAMINADORA:

Prof. Dr. Antonio Heronaldo de Souza (FEJ/UDESC)
Dr. Roberto Tavares (CenPRA)
Prof. Dr. José Antonio Siqueira Dias (DEMIC/FEEC/UNICAMP)
Prof. Dr. Oséas Valente de Avilez Filho (DEMIC/FEEC/UNICAMP)

Tese apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, como parte dos requisitos exigidos para a obtenção do título de Doutor em Engenharia Elétrica

Dezembro – 2002

Dedicado a:

Maria Nilza Vieira, in memorium.

Agradecimentos

Agradeço a todos os meus amigos que me deram suporte durante este período, em especial a Raquel, Francisco, Heron e Cláudia, Ademildes, Luciano, e a todos os amigos do DEMIC.

Índice

RESUMO

Este trabalho apresenta um ambiente de desenvolvimento voltado para aplicações que utilizam sistemas dedicados. O ambiente proposto utiliza os conceitos de programação visual, ou seja, trabalha com elementos gráficos (símbolos, ícones, etc) para representarem os dados e algoritmos utilizados pela aplicação. Estes conceitos visam um aumento na produtividade do programador de sistemas dedicados, pois proporcionam uma maior abstração dos algoritmos a serem implementados.

O ambiente possui três modos de entrada: fluxograma, diagrama de ícones e diagrama esquemático. Uma vez terminada a fase de descrição funcional da aplicação, passa-se para a etapa de simulação do código proposto. A observação dos parâmetros de simulação pode ser feita através de instrumentos virtuais que compõem o sistema, tais como: VU meter, osciloscópio, barra de leds, display de sete segmentos, etc.

Para interagir com o sistema em tempo de simulação foram previstos alguns elementos atuadores tais como: teclado numérico, chave liga/desliga, interruptor de pressão. Estes atuadores modificam as respostas do sistema em função do seu estado.

Depois de depurado, o aplicativo pode, então, ter seu código *Assembly* gerado pelo compilador de ícones e posteriormente codificado em linguagem de máquina. A transferência do código final do ambiente de desenvolvimento para uma memória do tipo não volátil externa se dá através da porta serial padrão RS232.

ABSTRACT

This work presents a development system turned to applications that use embedded systems. The proposed environment uses visual programming concepts, that is, it works with graphical elements (symbols, icons, etc), to represent algorithms and data used by the aplication. Theese concepts aim a increase of system programming productivity, because of their bigger abstraction of the algorithms to be implemented.

The environment has three input modes: fluxogram, icon diagram and schematics diagram. After describing the application, the next step is simulating of the code generated. To watch the simulation parameteres, we use virtual instruments such as: VU meter, osciloscope, leds bar, seven segment display, etc.

We can interact with the system in simulation time, by using some elements called actuators: Numeric pad, on/off key, push-button. Theese actuators change the system outputs, depending on their state.

After debugging, the icon compiler generates the *Assembly* code that is converted to machine code in a later process. The final code is tranferred from the environment to a external non-volatil memory, by using standard serial port RS232.

Introdução

Os atuais conceitos de ferramentas de desenvolvimento exploram as facilidades de se trabalhar em ambientes operacionais gráficos (Windows 95, NT, OS/2, Openwindows, etc). Podemos constatar através de programas tais como LabView[1] e LabWindows (National Instruments), MPLab (Microchip) entre outros, que o uso de símbolos e imagens para expressar idéias e algoritmos, está cada vez mais conquistando espaço entre os programadores de sistemas. Isto se deve ao fato de um aumento significativo no poderio computacional dos atuais computadores pessoais, que tiveram as suas capacidades de memória e velocidade de processamento expandidos enormemente.

No contexto de sistemas baseados em microcontroladores, analisando as ferramentas de desenvolvimento existentes (ANEXO 2) pode-se notar que sua grande maioria apresenta uma IDE (Integrated Development Environment) baseada na descrição textual (geralmente em linguagem C ou A*ssembly*) do código a ser gerado, o que contradiz a tendência atual de ambientes gráficos, onde o uso do mouse deve se sobrepor ao uso do teclado, como dispositivo de entrada.

A Proposta do Trabalho

Em se mantendo a tendência evolutiva da tecnologia de hardware, as linguagens visuais tendem a se tornar cada vez mais utilizadas e, por conseguinte, o seu grau de complexidade tende a aumentar, devido à evolução dos dispositivos de entrada, como por exemplo, mouses com novas funções, e dos dispositivos de saída (recursos visuais).

A disponibilidade de uma ferramenta que proporcione ao projetista uma maior abstração do sistema a ser projetado, onde se faz uso de um conjunto de símbolos bastante conhecidos extraído dos fluxogramas de engenharia, e de ícones intuitivos, vem de encontro a essa tendência visando uma maior qualidade aos sistemas gerados. É sabido das dificuldades de se conseguir um sistema que utilize linguagem visual de aplicabilidade genérica, por isso nossa proposta se concentra no nicho de aplicações em sistemas

dedicados baseados em microcontroladores, nesta versão dedicada ao microcontrolador 8051 da Intel, todavia este trabalho pode se estender para outras famílias de microcontroladores e DSPs, respeitando as características de hardware de cada um deles.

Objetivos do Trabalho

O nosso objetivo é construir uma ferramenta que faça uso dos conceitos da programação visual e se aplique na construção de sistemas dedicados microcontrolados. Esta ferramenta deve ser capaz de, partindo de elementos gráficos, gerar código em linguagem Assembly correspondente ao microcontrolador adotado, simular este código estabelecendo o funcionamento lógico do mesmo, apresentar os resultados através de instrumentos virtuais de alto nível e representar o diagrama esquemático correspondente ao hardware do sistema a ser implementado. Buscamos deixar o mais claro possível para o leitor quais as soluções escolhidas para enfrentar o problema e deixamos registrado passo-a-passo como se obter uma ferramenta semelhante.

Para isso fizemos uso de ferramentas de construção de softwares visuais comerciais (Visual C++) e utilizamos os conceitos descritos por M. Burnett e M. Baker[2] como uso de diagramas e ícones para descrever a aplicação e introduzimos um conceito novo que é o do uso de componentes eletrônicos que representam o hardware do conjunto a ser testado partindo para uma simulação lógica do sistema.

Metodologia de Trabalho

O trabalho foi desenvolvido dentro de uma metodologia analítico-construtiva seguindo os seguintes passos: Foi eleito um conjunto de elementos teóricos capazes de dar suporte aos princípios das linguagens visuais. A linguagem de fluxogramas e os ícones de operação foram adotados como formas visuais clássicas. O modo de interligação destas formas também foi objeto de análise e a opção adotada foi a de setas que indicam origem e

destino. As cores dos objetos ficam a cargo do usuário que pode selecioná-las em função das operações a ser executadas. A adoção de textos complementares às formas também foi implementada para dar o máximo de informação ao desenvolvedor. A descrição funcional de cada bloco de fluxograma se faz através de ícones intuitivos que representam as operações desejadas.

Apresentando o MODI

O Modi[3] é um ambiente de desenvolvimento integrado (IDE) que trabalha com elementos visuais para geração de sistemas dedicados que utilizam microcontroladores. Considerando os critérios das linguagens de programação visuais estabelecidos por M. Burnett e M. Baker[2], o Modi tem as seguintes características:

- Orientado a fluxo de dados (VPL*-II.A.3);
- Linguagem diagramática (VPL-II.B.1);
- Fluxo de controle (VPL-III.B);
- Utiliza tradutores (VPL-IV.D);
- Linguagem de aplicação geral (VPL-V.A);
- e teoria de ícones (VPL-VI.B).

Ele pode ser descrito como um gerador de fluxograma onde podem ser colocados os diversos blocos funcionais que, por sua vez, serão descritos através de uma linguagem visual que utiliza ícones[4]. Um outro editor gráfico complementar faz a descrição do hardware envolvido, mostrando seus principais componentes e ligações.

O projeto do Modi utiliza os recursos do ONAGRO[4] para fazer a compilação para a linguagem A*ssembly* do microcontrolador especificado (nesta primeira versão foi utilizado o microcontrolador MCS 8051 da Intel), podendo ser feita a simulação lógica do código gerado.

---

* VPL significa Visual Programing Language

Para a visualização dos resultados da simulação, o sistema conta com alguns instrumentos virtuais criados para esta finalidade, tais como: voltímetro, osciloscópio, termômetro, display, leds, etc. Pode-se, ainda, atuar no sistema em tempo de simulação, através de dispositivos virtuais tais como: interruptores, botões "push-button", etc; que irão alterar as saídas do sistema em função de seus valores.

Por fim, através da porta serial do microcomputador, pode ser feita a transferência do código gerado e testado, para um sistema gravador de memórias EPROM, encerrando o ciclo de desenvolvimento do software.

O Modi pode ser executado em qualquer microcomputador padrão IBM PC, com sistema operacional Windows 9x ou NT.

A referência [5], traz alguns comparativos entre programas semelhantes ao Modi que podem destacar melhor as qualidades deste software quanto a questões tais como: usabilidade, funcionabilidade, flexibilidade, etc.

Capítulo I: Conceitos de Programação Visual

1.1 Introdução

Programação visual se refere a qualquer sistema que permita ao usuário especificar um programa em duas ou mais dimensões[6]. Linguagens textuais convencionais não são consideradas bidimensionais uma vez que os compiladores ou interpretadores às processam como um fluxo de informações unidimensional. A programação visual inclui linguagens de programação gráficas que usam fluxogramas convencionais para criar programas. Isto não inclui sistemas que usem linguagens de programação convencional (linear) para definir figuras, como por exemplo, a linguagem postscript.

Uma linguagem visual manipula informação visual ou dá suporte a interações visuais, ou ainda, permite a programação com expressões visuais[7]. Esta última é tida como sendo a definição de uma linguagem de programação visual. Linguagens de programação visuais podem ser classificadas de acordo com o tipo e extensão da expressão visual usada, podendo ser linguagens baseadas em ícones, baseadas em formas ou diagramas, etc.

Ambientes de programação visual oferecem elementos gráficos ou ícones que podem ser manipulados pelo usuário de forma interativa, de acordo com alguma gramática espacial específica para construção de programas.

Nas linguagens visuais, as primitivas da linguagem (ícones, linhas, setas, e formas) têm sintaxes e semânticas bem definidas. As sentenças expressas nestas linguagens (por exemplo, ícones ligados por setas) podem ser interpretadas e em seguida traduzidas para linguagens tais como Smalltalk, C++, JAVA. No caso do Modi, os símbolos são traduzidos para *Assembly* do 8051.

As linguagens visualmente transformadas são inerentemente linguagens não-visuais, mas que tem representações visuais superpostas, já as linguagens naturalmente visuais têm uma inerente expressão visual para qual não existe uma expressão textual equivalente.

Uma idéia para a sintaxe da linguagem visual é o uso de um ambiente genérico que possa ser parametrizado com uma linguagem de especificação. O que é preciso para fazer isso: um ambiente de programação visual independente, uma gramática de layout de figuras e um gerenciador (parser). As vantagens de usar um "parser" em um ambiente de uso especial é que esta separação promove uma definição precisa da sintaxe da linguagem. O programa é apenas uma figura, o mesmo editor pode ser usado para muitas linguagens e o esforço para definir uma nova linguagem visual é reduzido apenas a definição de uma nova gramática.

As ferramentas de programação visual diferem significativamente daquelas usadas para visualização de programas[8]. Na visualização de programas, estes são escritos de maneira tradicional; as ferramentas simplesmente mostram uma visão gráfica deles. As ferramentas de visualização de programas usam gráficos apenas para ilustrar os aspectos de um programa ou sua execução. Como exemplo deste tipo de ferramenta podemos citar o Visual C++, Visual Basic, etc.

1.2 Linguagem visual vs. Linguagem textual

Um ponto importante de destaque de todas as linguagens textuais são as variáveis. Variáveis são símbolos que são usados para representar objetos desconhecidos. Eles servem a duas aplicações contraditórias:

a) Eles ligam pontos onde os dados são produzidos e usados.
b) Seus nomes fornecem informações sobre a intenção dos dados.

Na linguagem gráfica, as variáveis podem ser substituídas pelo uso de imagens que indiquem todos os pontos do programa onde o mesmo item de dado ocorre.

Os elementos primitivos de uma figura podem ser agrupados juntos em formas agregadas que podem ser melhor combinadas para formar conjuntos maiores. Em programas textuais estes grupos são combinados adjacentemente em sub-frases. A composição de duas frases textuais é chamada de concatenação.

Para uma figura a composição é mais complicada que a concatenação. A composição de figuras pode ser feita com formas adjacentes, onde as posições relativas possam ser especificadas.

A composição pode também ser feita pela conexão dos elementos tais como dois segmentos de linhas com um ponto final comum. Ambas as linguagens (textual e gráfica) são formadas sobre um alfabeto básico e podem ser decompostas em uma estrutura bem definida. Porém, devem ser observados os seguintes pontos:

a) No caso das linguagens textuais, as *strings* de texto são unidimensionais e têm uma ordenação linear;
b) No caso das linguagens visuais, a estrutura subjacente de um programa visual é geralmente um gráfico direto ao invés de uma árvore.

1.3 Linguagem de fluxo de dados

Os programas são representados por um gráfico direto onde os nós representam as funções e os arcos representam os fluxos de dados entre as funções. Os arcos que entram nos nós representam a entrada de dados para as funções; os arcos que saem representam a saída de dados das funções.

1.3.1 Vantagens do modelo de fluxo de dados:

O modelo de fluxo de dados, bastante utilizado por alguns aplicativos, apresenta algumas vantagens dentre as quais destacamos:

- Adequado para o domínio de aplicações específicas (ex: coleta e análise de dados (LABVIEW));
- Adequado para iniciantes em programação e não programadores;
- Adequado para a manipulação de dados (processamento de imagens, gráficos);
- Adequado para transformação de dados (filtros) e;
- Indicado para fornecer funções extras ou predefinidas.

1.3.2 Desvantagens do modelo fluxo de dados:

Apesar de apresentar algumas vantagens consideráveis, o modelo de fluxo de dados também apresenta algumas desvantagens, dentre as quais pode-se citar:

- Necessita de grande espaço para a visualização da aplicação;
- Não está muito claro quais construções de alto nível ou funções poderiam ser criadas e;
- Não é muito bem aceito por programadores profissionais.

1.3.3 Modos de execução

Quanto ao modo de execução os modelos de fluxo de dados podem ser:

- guiado por dados: Modo de execução no qual os nós são ativados tão logo todas as entradas se tornem disponíveis;
- guiado pela demanda: Modo de execução no qual os nós respondem apenas quando explicitamente solicitados.

1.4 Rede de transição de estados

Diferentemente da técnica de fluxo de dados, programas feitos com a técnica de transição de estados são simbolizados por gráficos de estado, os quais estão conectados por linhas que indicam as transições.

Um bom indicativo dessa técnica é a linguagem de fluxograma utilizada em Modi, onde cada bloco pode ser entendido como uma instância de processo ou estado. Os diagramas usados nos fluxogramas estão dentro da norma CCITT[9], sendo geralmente usados em sistemas de telecomunicações e, por isso, considerada como linguagem dos engenheiros.

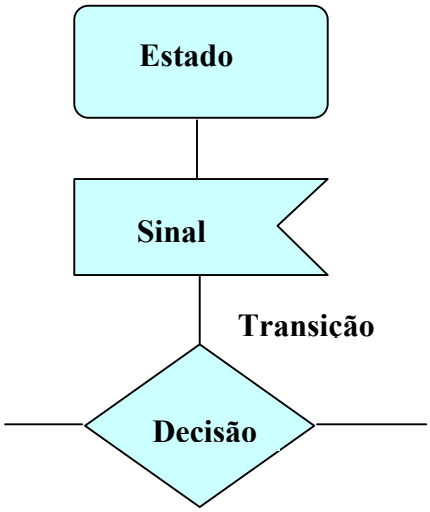


Fig. 1.1: Exemplo de Fluxograma

1.5 Ferramentas para construção de Interfaces

Uma vez definidos os modelos de aplicações existentes passamos agora à fase de elaboração da ferramenta em si considerando os conceitos estabelecidos por Brad Myers[8]. Segundo ele são as seguintes as razões que tornam difícil o projeto de interfaces, sendo aplicados também às interfaces gráficas:

- Projeto Interativo- A interface deve ser avaliada pelos usuários finais, o que resulta em modificações no projeto original. Este ciclo pode ser repetitivo. A consulta a alunos de graduação e profissionais da área de programação de microcontroladores, foi definitiva na elaboração do Modi, tendo sido obtidos resultados considerados satisfatórios pelo público usuário final.

- Multiprocessamento- É esperado que a interface rode independentemente do mecanismo que trata os eventos originados pelo teclado ou mouse. Isto habilita ao usuário interromper ou modificar a execução de uma aplicação. O Modi trabalha com multiprocessamento utilizando Threads em ambiente windows, com isso podemos alterar os conteúdos das variáveis envolvidas em tempo de execução e verificar os resultados dessas mudanças.

- Programação em Tempo-Real- Os objetos gráficos devem ser capazes de moverem-se livremente pela tela sem interrupções ou pausas. Esta característica também está presente no Modi, pois a movimentação das janelas abertas não interfere na execução das outras tarefas envolvidas no momento.

- Robustez- A interface deve responder a entradas errôneas com mensagens sensíveis ao contexto. Foram criadas janelas de diálogos que indicam erros ocorridos na entrada de alguns valores de dados incompatíveis com o tipo de dado esperado.

- Modularidade- O princípio da separação entre a interface e a camada da aplicação é muitas vezes ignorado, e mudanças na interface resultam em mudanças na aplicação e vice-versa.

- Atingir os efeitos desejados- Isto é um tanto difícil, pelo menos com as ferramentas existentes. A primeira vista os efeitos desejados foram atingidos dentro do projeto do Modi, pois as características quanto a clareza, intuitividade, facilidade de documentação, e velocidade da aplicação foram atingidos de forma satisfatória.

- Suporte a linguagem- Linguagens de computador não fornecem um suporte explícito para a interface de entrada/saída do usuário. No Modi, todas as interfaces foram criadas para permitir um uso intuitivo de todo o sistema.

O uso de uma ferramenta de interface permite que a qualidade da interface seja mais consistente, reduz o tempo de cada ciclo de projeto, reduz o esforço necessário, permite o envolvimento de não-especialistas e pode incorporar sugestões do cliente mais rapidamente. E, por fim, o código da interface pode ser mais fácil e mais econômico de criar e manter. O uso do ambiente de programação Visual C++ permitiu um controle maior sobre o comportamento da interface Modi dentro do ambiente Windows, por ser uma linguagem orientada a objetos, facilitou a modularidade e a manutenção do software em desenvolvimento.

1.6 Exemplos de Linguagens de programação visual comerciais

A seguir mostramos como o uso da linguagem visual de programação vem se tornando popular em vários nichos de aplicação, desde sistemas voltados a aplicações genéricas para PC em ambiente windows, até sistemas mais específicos voltados a instrumentação eletrônica digital.

a) LABVIEW

Linguagem de programação visual de captura e análise de dados disponível para Windows9x/2000/NT, Mac, PowerMax OS, Solaris,HP-Unix, Sun e Linux, de propriedade da National Instruments (www.natinst.com/labview). O sistema permite ao usuário construir graficamente módulos chamados de instrumentos virtuais (VI). A linguagem nativa do LabView é o G. Uma informação deve ser ressaltada, o LabView é o ambiente de desenvolvimento, enquanto que o G é o código produzido sob o sistema LabView.

O Modelo de fluxo de dados é a base na qual o LabView trabalha no seu conteúdo. A filosofia básica é que a passagem dos dados através dos nós dentro do programa determina a ordem de execução das funções do programa. As VI's do LabView têm entradas, processamento de dados e produção de saídas. Arranjando as VI's que têm entradas ou saídas em comum, é possível estabelecer uma ordem de prioridade das funções que o programador deseja para que os dados sejam manipulados. A Figura 1.2 ilustra uma vista do ambiente LabView.

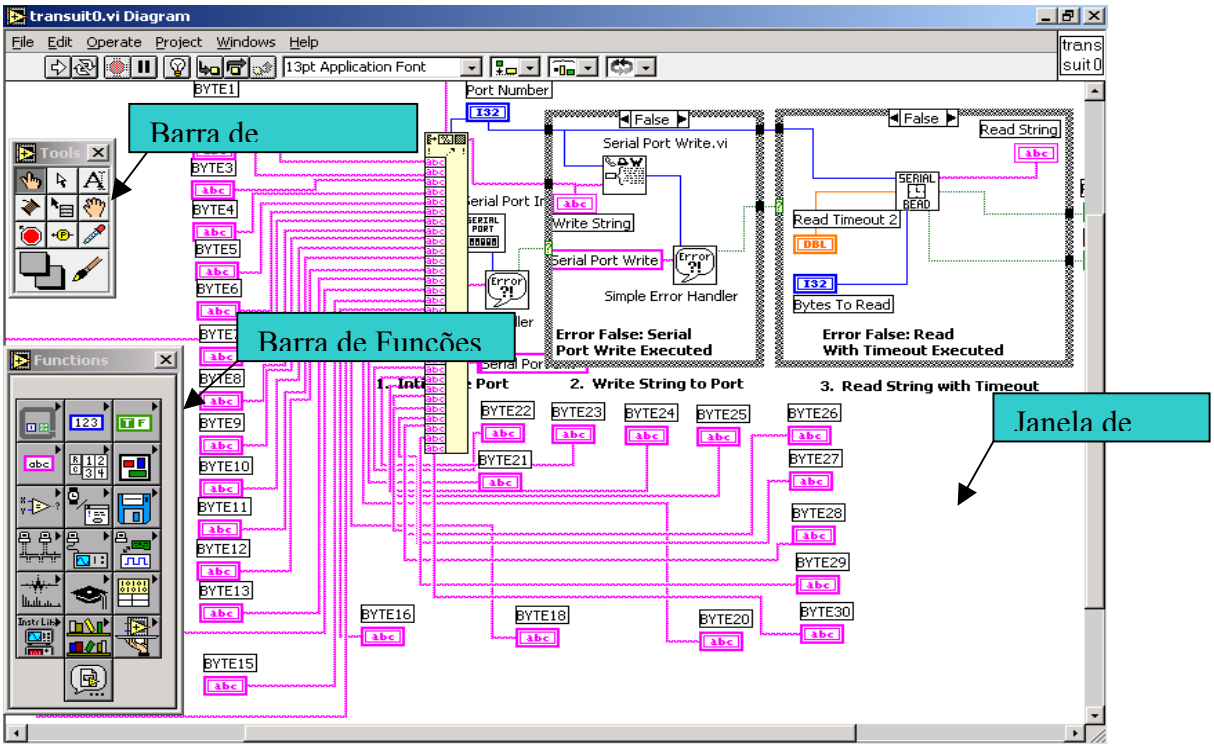


Fig. 1.2: Vista geral do LabView.

b) HPVEE

O HPVEE (www.hp.com/go/hpvee) é uma ferramenta visual criada para desenvolver programas de teste de equipamentos. Ela pode ser usada para controlar instrumentos e armazenar os dados medidos para serem mostrados em displays, salvos em arquivos ou mostrados através de gráficos. Ela permite ainda ao usuário, conectar e controlar equipamentos digitais da HP tais como osciloscópios, fontes de alimentação e geradores de funções. A plataforma utilizada é o Windows 9x ou NT. A figura 1.3 mostra um diagrama feito no HPVEE.

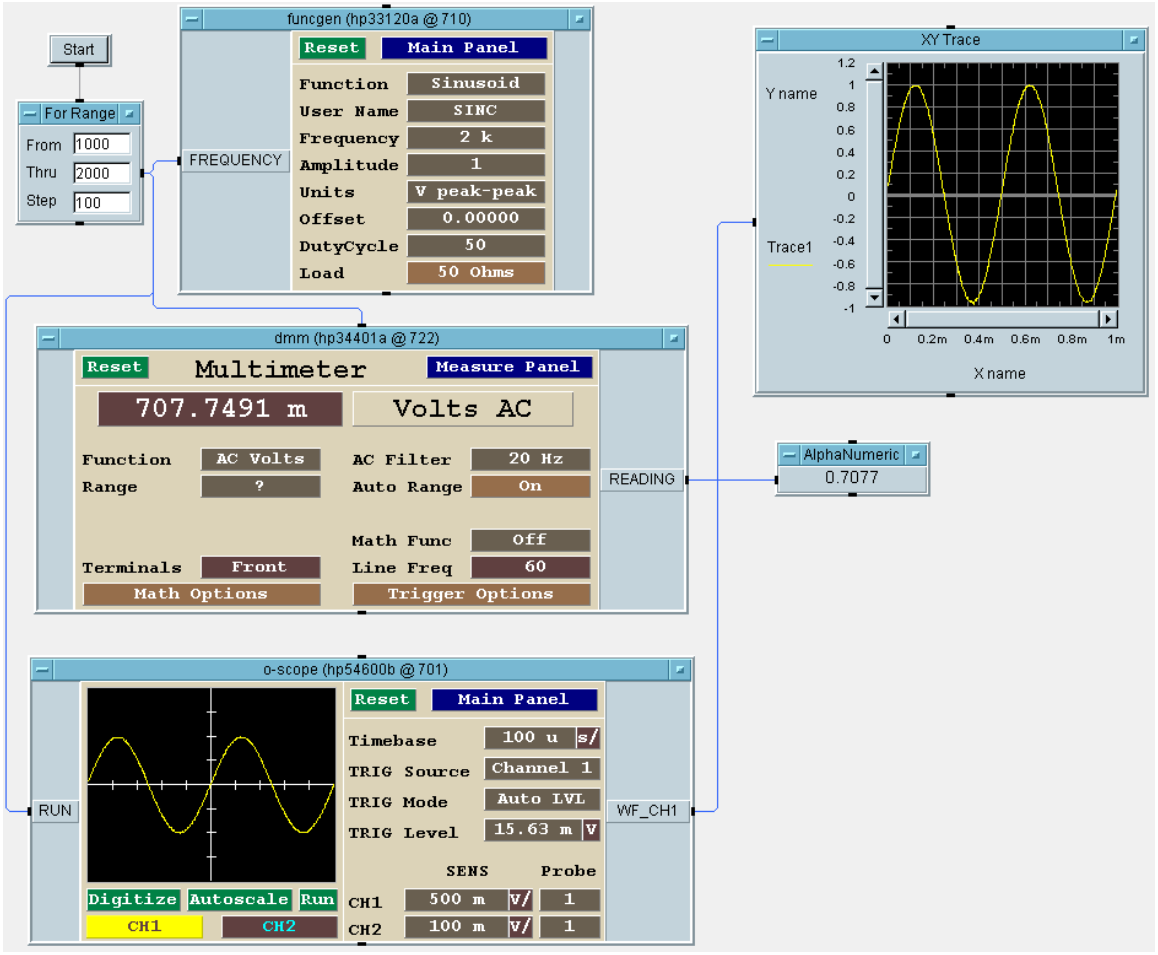


Fig. 1.3: HPVEE.

c) CREATE

Esta versátil ferramenta criada recentemente pela Sharper Software (www.create-software.com) funciona como um gerador de código a partir de fluxogramas. Ela é capaz de gerar código nas linguagens C e JAVA, podendo também gerar um arquivo executável no formato .EXE para plataformas PC Windows. A figura 1.4 mostra a interface CREATE.

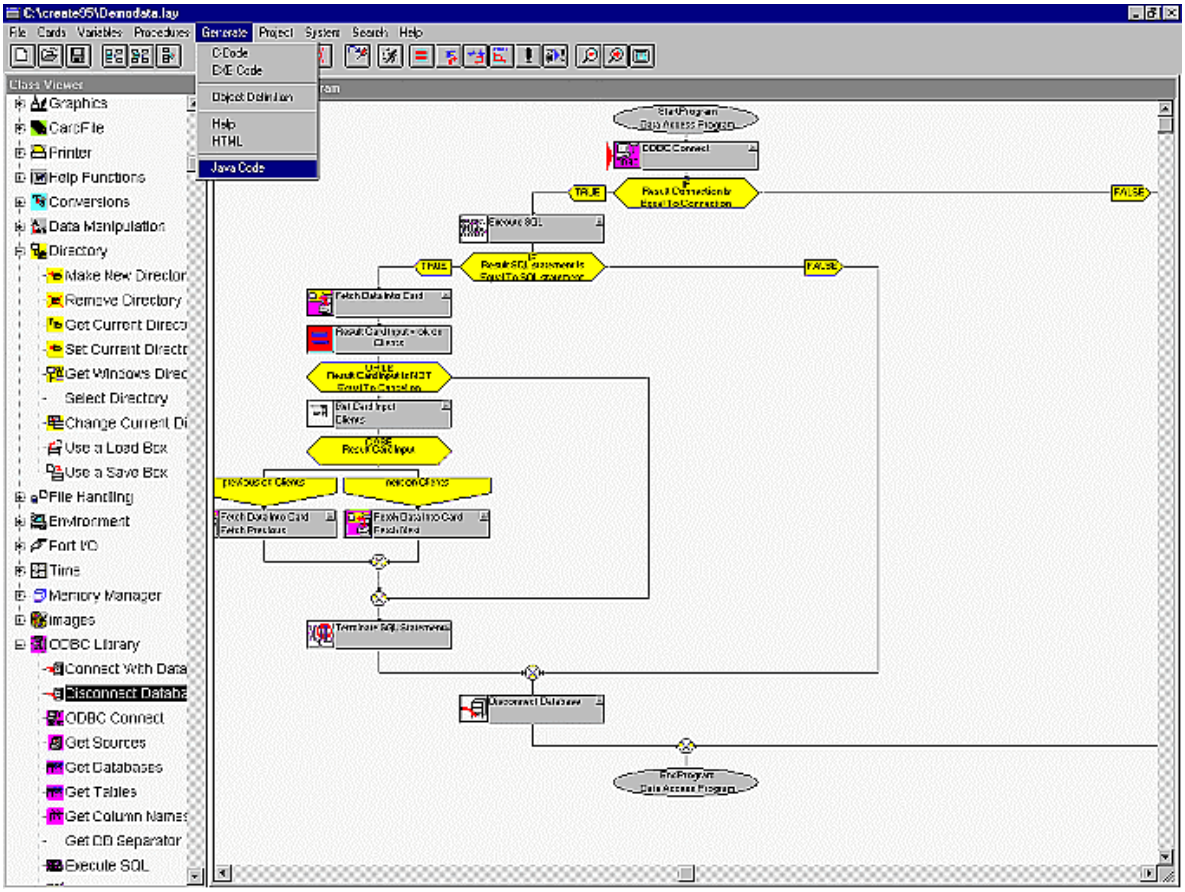


Fig. 1.4: CREATE.

O CREATE disponibiliza as funções básicas usadas na criação de programas em ambiente Windows, tais como abertura de janela, diálogos, abertura de arquivos, gerenciamento de memória e manipulação de imagens, entre outras. Estas funções são agrupadas por categorias e colocadas seqüencialmente no fluxograma relativo a aplicação em desenvolvimento.

d) Peter

Ferramenta produzida pela Gemtree Software (www.gemtree.com) voltada a criação de aplicações visuais e multimídias em ambiente Windows. As aplicações são geradas a partir de estruturas semelhantes a árvore de diretórios e é voltada a programadores iniciantes que não conhecem nenhuma das linguagens mais comuns de programação. Ela não gera código em nenhuma linguagem, gera apenas programas executáveis em ambiente Windows. A figura 1.5 mostra a API do Peter.

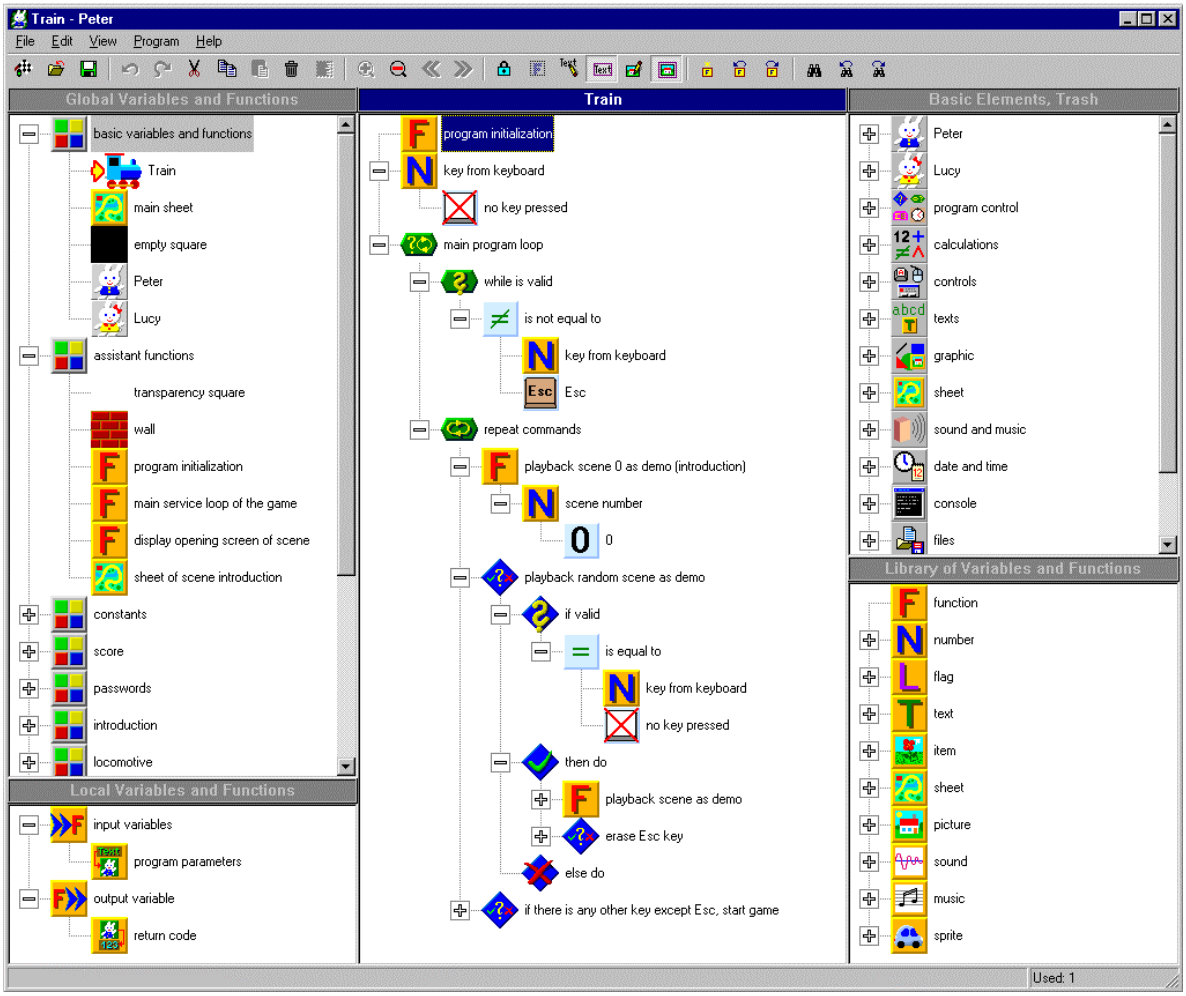


Fig. 1.5: Peter.

1.7 Exemplos de Linguagens de programação visual comerciais voltadas a microcontroladores.

Particularizando agora para ambientes de desenvolvimento de sistemas baseados em microcontroladores, encontramos os seguintes programas que utilizam soluções diferentes daquelas empregadas pelo Modi.

a) VisualSTATE

Ferramenta produzida pela IAR Systems (www.iar.com) que faz a geração de código em linguagem C para microcontroladores a partir de diagramas de estado. A ferramenta também faz a simulação do código e a sua documentação. A figura 1.6 mostra a API da ferramenta.

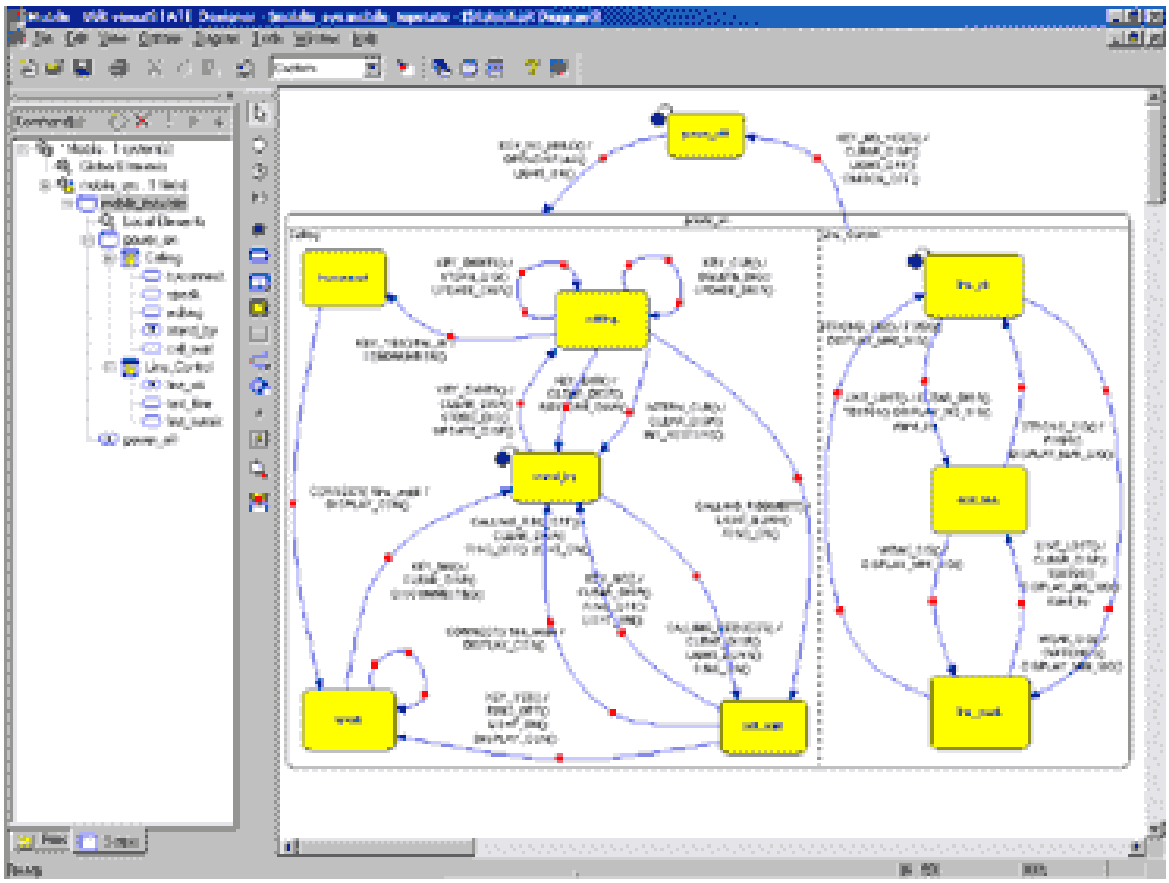


Fig. 1.6: VisualSTATE API..

O uso de diagramas de estado é bastante recomendável quando trabalhamos com a representação de sistemas paralelos e “multi-thread”.

b) Realizer

Ferramenta de desenvolvimento da Actum Solutions (www.actum.com) para programação de microcontroladores em geral. Possui três versões: Bronze, Silver e Gold. Foi projetado para a plataforma Windows 9x e NT.

O Realize trabalha com um conjunto de símbolos usados no desenho de diagramas esquemáticos que representam a aplicação em desenvolvimento como pode ser visto na figura 1.7. Esta técnica se assemelha ao usado pelo LABVIEW nos seus símbolos.

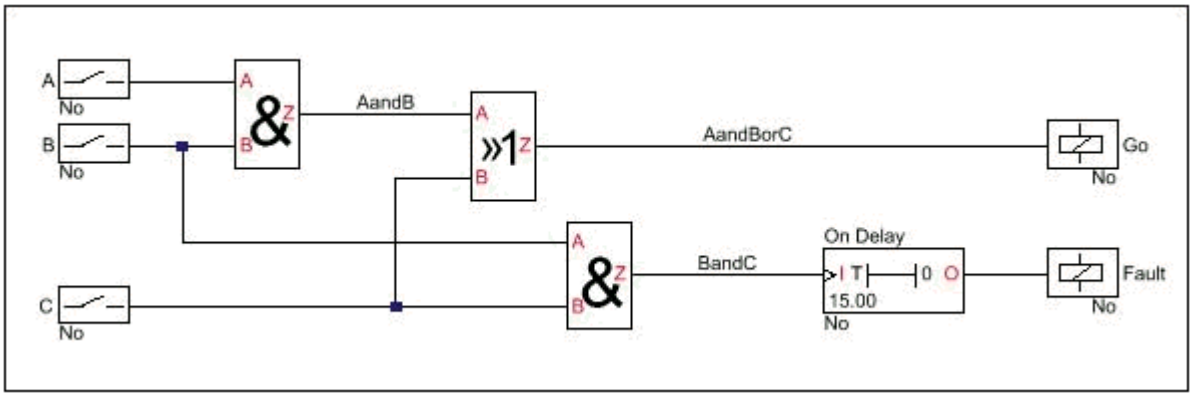


Fig. 1.7: Diagrama esquemático.

O Realizer também possui um simulador para testar o funcionamento dos aplicativos em desenvolvimento. Este simulador trabalha com dois tipos de “probes” que são posicionados no diagrama esquemático. Estas “probes” podem ser numéricas ou traçador de curvas. A figura 1.8 mostra a inserção destes “probes” dentro de um esquemático.

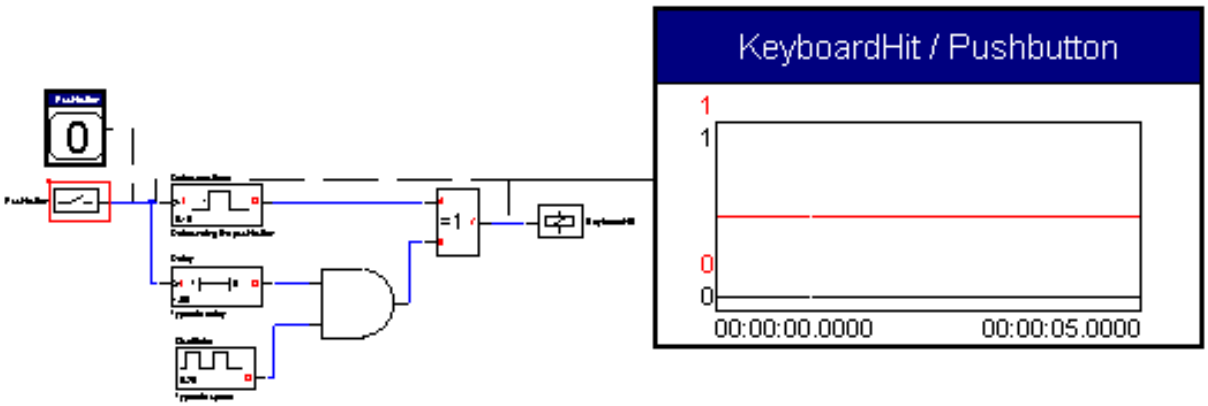


Fig. 1.8: Probes no diagrama esquemático.

O simulador do Realizer possui apenas o modo contínuo de simulação, não permitindo a simulação bloco-a-bloco dos seus componentes.

O Realizer gera código .ASM livre de erros de sintaxe para o microcontrolador escolhido. Este código pode ser convertido para o formato .HEX por um "assemblador" externo.

Os idealizadores do Realizer garantem uma redução de 80% no tempo de projeto com o uso desta ferramenta.

c) PARSIC

Ferramenta visual para programação de microcontroladores da família PIC criada por Swen Gosh (www.parsic.de). Funciona a partir de objetos que simbolizam elementos de hardware interconectados formando um diagrama de blocos. Estes objetos podem ser: porta serial RS232 e RS485, módulos LCD, Flip-flops, Timers, Multiplexers entre outros.

A ferramenta faz a geração de código em Assembly PIC para as interfaces e também possui um simulador. Este simulador trabalha em modo contínuo simulando os

blocos de funções e não utiliza “probes” para demonstrar os status do diagrama. Alguns elementos possuem um pequeno retângulo que indica o seu estado atual e outros, como o display LCD, apresenta uma vista real do seu módulo. A figura 1.9 apresenta uma tela extraída do PARSIC exemplificando o que foi dito.

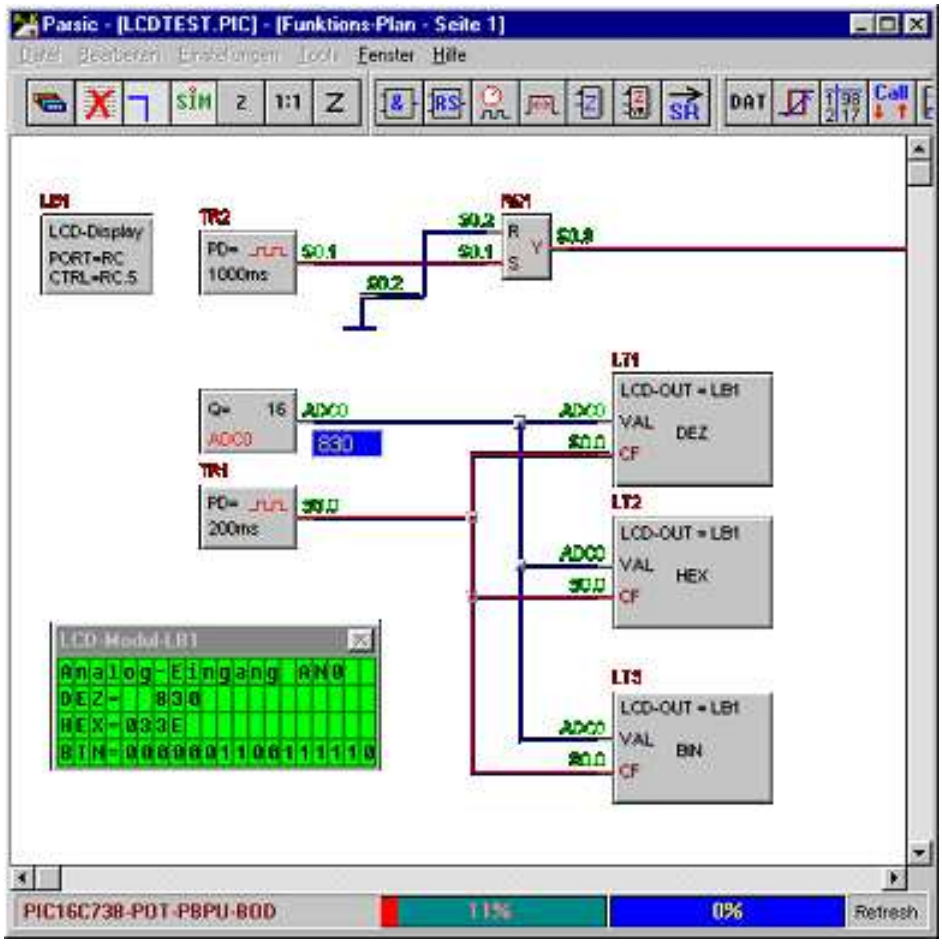


Fig. 1.8: PARSIC.

O PARSIC assim como o Realizer apresentam soluções do tipo LABVIEW com seus blocos funcionais, só que aplicados a programação de microcontroladores.

1.8 Considerações sobre o Modi.

Conforme foi apresentado no item anterior, ficamos atualizados quanto às diferentes técnicas usadas para se implementar ferramentas de desenvolvimento que utilizem os conceitos da programação visual. Devemos aqui salientar que desde a idealização do Modi (1994) até o seu primeiro protótipo (1998), poucas, senão nenhuma, ferramenta de desenvolvimento que utilizasse linguagem visual para microcontroladores estava disponível. Pois o uso desta técnica estava apenas começando. Nos dias de hoje, com o desenvolvimento de hardwares mais poderosos e velozes, temos um aumento do número de ambientes visuais voltados as mais diferentes aplicações, chegando às linguagens visuais a serem consideradas como linguagens de quarta geração. Dito isso e comparando com os programas mostrados, vimos que nenhuma das ferramentas para microcontroladores adota alguma das soluções empregadas pelo Modi. Algumas ferramentas adotaram o diagrama de estados, outras blocos funcionais para descreverem a aplicação a ser desenvolvida. Para nós, o uso de fluxograma é o mais indicado pois é nele que pensamos quando queremos descrever uma funcionalidade ou algoritmo, devido a sua grande utilização nos cursos técnicos e nas universidades.

Contudo fica claro que o Modi apresenta alguns diferenciais que o torna uma ferramenta bastante versátil e competitiva no aspecto da programação de sistemas dedicados:

- A descrição das operações executadas pelos algoritmos através do uso de ícones promove uma maior abstração ao projetista de sistemas, praticamente eliminando o uso do teclado na descrição da funcionalidade da aplicação. Isto elimina erros de digitação no código final gerado, pois o compilador de ícones gera o código automaticamente e livre de erros. Quanto à idéia de que o código gerado a partir de linguagem gráfica é maior do que o gerado a partir de linguagens textuais, como por exemplo, a linguagem C, essa se mostra não verdadeira como podemos ver na figura 1.9, tirada de um comparativo feito entre o compilador de ícones do Modi [4] e o AVOCET C para algumas aplicações tais como Varredura de teclado, controle de motor de passo, linearização de transdutor, etc. Isto deixa

livre o programador para optar se ele deseja trabalhar com uma linguagem de alto-nível textual ou linguagem de alto-nível gráfica.

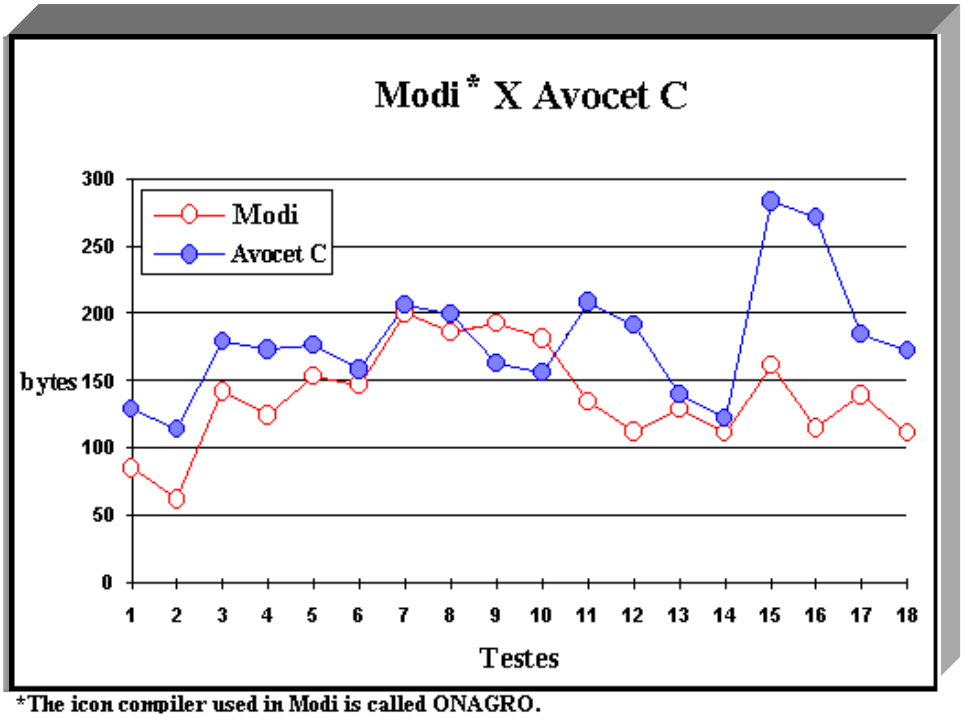


Fig. 1.9: Comparação do tamanho de código gerado.

- A simulação da aplicação em desenvolvimento utilizando recursos de alto-nível para a observação das variáveis envolvidas também representa uma característica de destaque do Modi. Ela torna bastante amigável o "debugging" da aplicação em desenvolvimento, pois faz uso de instrumentos virtuais e de atuadores virtuais que podem ser relacionados às variáveis do sistema, permitindo uma maior interação em tempo de simulação entre o projetista e o seu projeto, tendo a finalidade de estabelecer um ambiente de bancada eletrônica ao desenvolvedor, que pode conectá-los como se tivesse trabalhando no plano real.

- O editor de esquemático permite uma visualização do hardware do sistema em desenvolvimento, ajudando o projetista na concepção tanto do hardware como do software.

Ele pode também acompanhar a simulação no nível lógico dos componentes de hardware do projeto, reforçando, assim, o ambiente de bancada eletrônica onde o projetista pode verificar o funcionamento do seu projeto pino a pino. Isto traz vantagens ao projetista que pode verificar se o hardware projetado atende ou não as especificações juntamente com o software dentro de uma mesma ferramenta.

- Outro ponto de destaque no Modi é a facilidade na documentação dos projetos nele desenvolvidos. Isto é muito útil quando o projeto é feito por equipes, pois além do uso de fluxogramas e ícones para descreverem a funcionalidade e do desenho esquemático, podemos ainda adicionar outros recursos ao ambiente. Estes recursos podem ser desde um arquivo de texto do Word, por exemplo, até recursos de multimídia como sons e imagens explicativas. A figura 1.10 ilustra o que acabamos de mencionar.

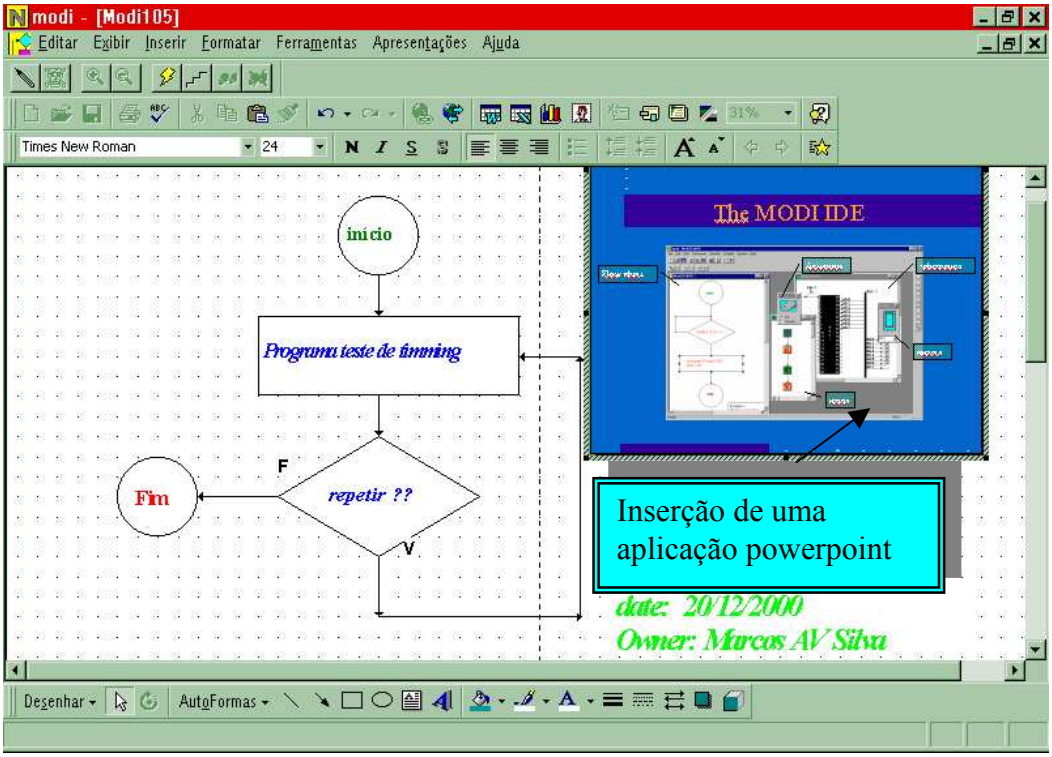


Fig. 1.10: Uso de Objetos OLE.

Por tudo isso, vemos que englobar todas estas características em uma única ferramenta trouxe uma contribuição ao universo de ferramentas de desenvolvimento de sistemas dedicados, que esperamos possa ser bem utilizada pelos projetistas de sistemas e de ferramentas.

Capítulo II: Descrição da metodologia de desenvolvimento do trabalho

2.1 Introdução

Neste capítulo será apresentada a metodologia adotada na criação do Modi, segundo os sete passos básicos da engenharia de software[10] que são os seguintes:

1) Definição da aplicação, suas metas e tarefas: Os conceitos principais que definiram o perfil do Modi já foram mencionados no resumo introdutório segundo os critérios defendidos por Burnett e Baker[2].

2) Projeto da aplicação e de suas interfaces para implementação das definições: Através da ferramenta de desenvolvimento Visual C++ [5] foi feito a elaboração do projeto do Modi e de suas interfaces, utilizando os conceitos da programação orientada a objetos dentro das definições estabelecidas no item 1.

3) Criação do protótipo: Neste ponto chegamos a versão 1.0 do Modi que contém as características básicas da implementação desejada. Todos os módulos apresentam o funcionamento desejado, ficando a implementação de outras funções e de outros blocos para versões futuras.

4) Teste do projeto com usuários para identificar falhas de projeto: Uma versão operacional do Modi foi apresentada a alguns programadores de sistemas experientes e a alguns alunos de graduação da UNICAMP que nos forneceram algumas sugestões úteis quanto ao uso da ferramenta e de futuras melhorias.

5) Corrigir as falhas, refazer o projeto e o protótipo: As sugestões acima mencionadas serviram de base para algumas alterações no projeto original, tendo sido obtido uma versão mais funcional e livre de falhas.

6) Desenvolver a aplicação, fazer testes finais e elaborar a documentação: Foram feitos testes com aplicações de média complexidade, respeitando as limitações desta versão, cujos resultados foram satisfatórios. A elaboração desta tese, que traz uma descrição detalhada do Modi juntamente com um tutorial de utilização, serve de documentação do ambiente Modi.

7) Colocar a disposição do usuário a aplicação final: Neste aspecto, poderíamos oferecer uma versão inicial do que desejamos em termos de aplicação final e completa do ambiente Modi. O capítulo V apresenta algumas sugestões que se deseja implementar para se obter uma versão final do Modi.

2.2 O Visual C++ e a MFC

O Modi foi escrito usando o pacote Visual C++ 4.0 da Microsoft e foi estruturado com base na programação orientada a objetos, o que proporciona uma maior modularidade e uma maior facilidade na manutenção do código. Uma vista do ambiente Visual C++ está mostrada na figura 2.1.

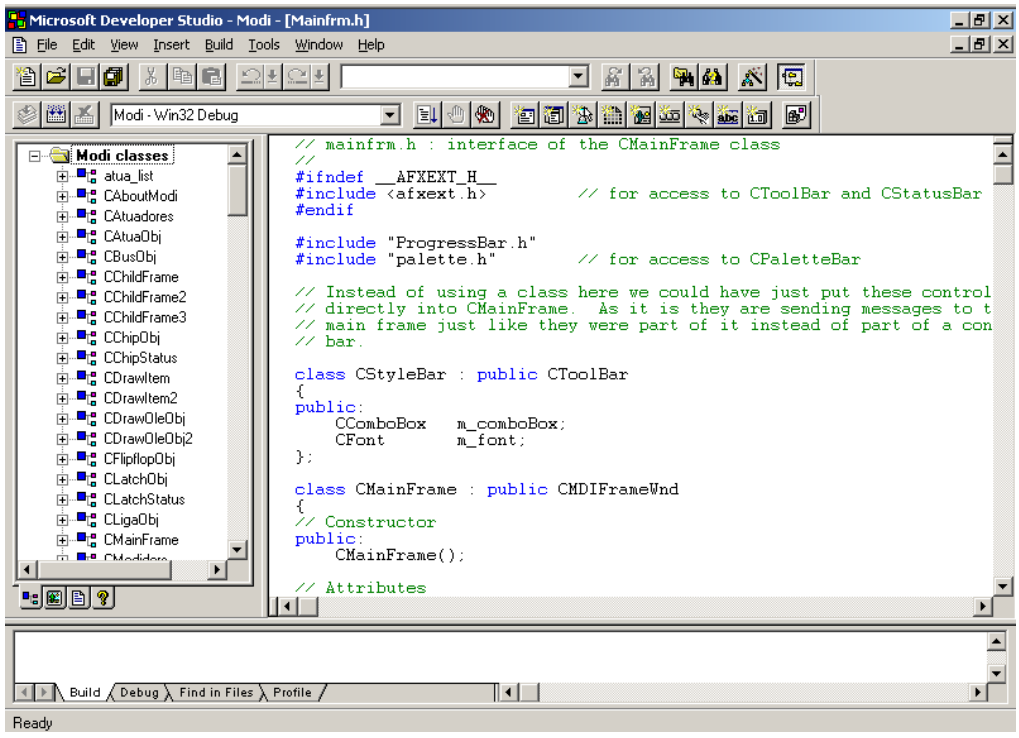


Fig.2.1: Visual C++.

O Visual C++ utiliza uma biblioteca de classes chamada MFC (Microsoft Foundation Classes) que, por ser muito eficiente, reduz o tamanho do código gerado pelo programador. A MFC atua como uma interface entre o C++ e a Windows API, fazendo o "trabalho pesado" de otimizar o código de acesso as funções do Windows. Outra característica dessa biblioteca é a sua portabilidade, ou seja, um código criado para ser utilizado em plataformas que utilizam o Windows 3.1 pode migrar para o Windows 95 ou NT. A MFC, na versão 4.0, contém aproximadamente 200 classes diferentes. O Modi utiliza cerca de 20 classes originadas da MFC.

O principal arquivo de cabeçalho (header) que contém todas as declarações de classes, funções e variáveis da MFC é o `afxwin.h`. Ele também contém outros "headers" relativos a Windows API.

O código gerado trabalha com o conceito de programação dirigida por eventos, que funciona da seguinte maneira: A aplicação é constituída por vários elementos gráficos (frames, botões, scroll bars, etc), que aguardam por um comando do usuário através do mouse ou teclado, em um loop de espera. Se for ativado um botão do mouse, por exemplo, isso é considerado um evento e a aplicação pode capturar e tratar esse evento da forma mais conveniente.

2.3 Programação Orientada a Objetos (POO)

Antes de entrarmos em maiores detalhes sobre a estrutura do programa, é necessária a definição de alguns conceitos básicos da POO.

- Abstração de dados ou encapsulamento: Processo de definição de tipo de dados chamados de tipos de dados abstratos. Esta definição utiliza a ocultação de dados para garantir uma maior flexibilidade em caso de alteração da estrutura do tipo definido sem se preocupar com as

referências a este tipo de dados no corpo do programa. Numa forma mais resumida, podemos dizer que o tipo de dados abstrato engloba um conjunto de variáveis e funções que lidam com essas variáveis. Essas variáveis podem ser chamadas de *membros de dados*, enquanto as funções são conhecidas como *métodos*. *Classe* é a forma mais conhecida de tipo de dados abstrato e é a base para criação dos *objetos*. Na verdade, quando criamos uma instância de uma *classe*, estamos criando um *objeto* com as características desta *classe*.

- Herança: É a característica que faz com que um objeto pertencente a uma classe tenha um comportamento semelhante (mas não igual) a um de uma outra classe. Isso só acontece dentro de uma hierarquia onde um objeto pode herdar os membros de dados e métodos de um outro de uma classe base. A classe que herda as características de uma outra recebe o nome de classe derivada.

- Polimorfismo: Define que uma mesma operação pode ter comportamentos diferentes para uma mesma chamada de execução (ou mensagem). Este comportamento se assemelha ao sobrecarregamento de um operador que, dependendo do contexto, realiza uma operação ou outra. Para objetos derivados da mesma classe base, podemos dizer que um mesmo método pode apresentar um comportamento diferente dependendo do tipo do objeto.

- Objetos: São conjuntos encapsulados de operações e um estado que registra e lembra o efeito dessas operações. Um objeto executa uma operação em resposta ao recebimento de mensagem que está associada à operação. O resultado da operação depende do conteúdo da mensagem recebida e do estado do objeto quando ele recebe a mensagem. O objeto pode, como parte dessa operação, enviar mensagem para outros objetos e para si mesmo.

- Mensagens: São requisições enviadas de um objeto para outro, para que este produza algum resultado desejado. A natureza das operações é determinada pelo objeto receptor. Mensagens podem ser acompanhadas de parâmetros, que são aceitos pelo objeto receptor e que podem ter algum efeito nas operações realizadas.

- Métodos: São descrições de operações que um objeto realiza quando recebe uma mensagem. Uma mesma mensagem poderia resultar em métodos diferentes, quando enviada para objetos diferentes. O método associado com a mensagem é pré-definido. Um método é similar a um procedimento.

- Classe: Uma classe é um "template" para objetos. Ela consiste de métodos e descrições de estado que todos os objetos da classe irão possuir. As classes aceitam mensagens e possuem métodos e um estado interno. Uma mensagem que muitas classes aceitam é uma mensagem de instanciação, que diz à classe para criar um objeto que é um elemento ou instância da classe receptora.

A figura 2.2 a seguir traz um exemplo que esclarece o que foi explicado:

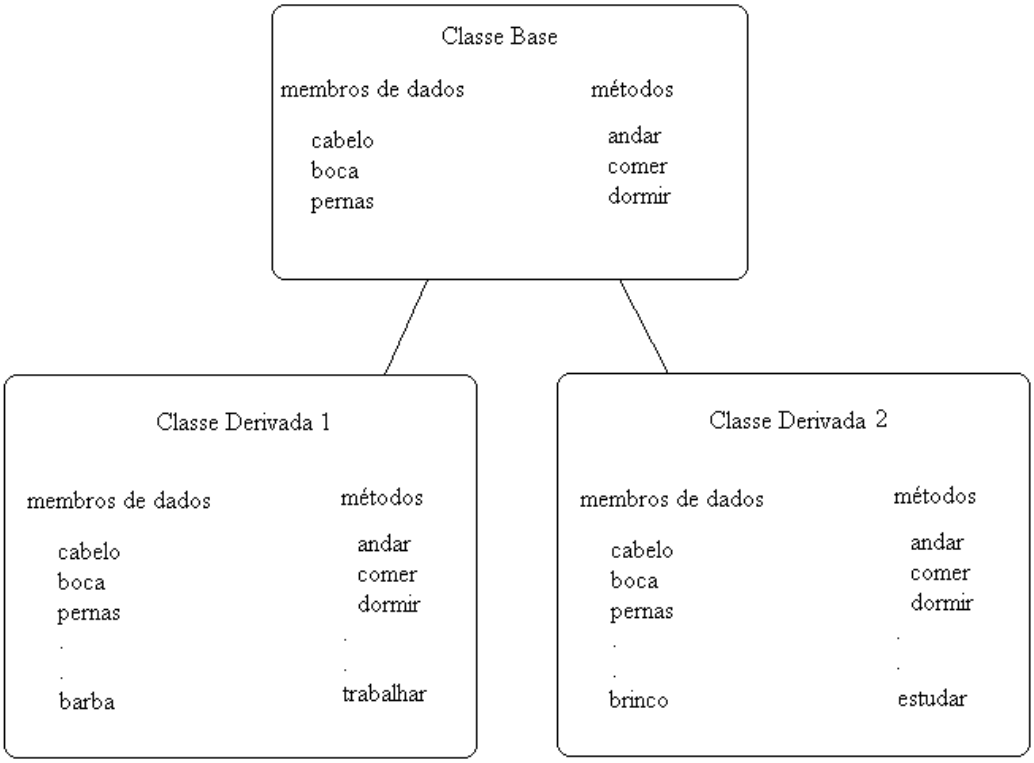


Fig.2.2: Exemplo básico de classes.

Como podemos observar na figura acima, foram derivadas duas classes a partir de uma classe base. Estas classes derivadas herdaram todos os membros de dados e todos os métodos da classe base, contudo, cada classe derivada adicionou novos membros de dados e métodos característicos de cada classe derivada, assim podemos criar objetos a partir destas classes derivadas que terão as características da classe base e características particulares à classe da qual ele for instanciado. Por exemplo, podemos criar um objeto chamado Homem, a partir da classe derivada 1, que possui as seguintes características: cabelo, boca, pernas (vindas da classe base) e barba (própria da classe derivada 1); e pode executar as seguintes atividades (funções membro): andar, comer, dormir (vindas da classe base) e trabalhar (própria da classe derivada 1).

Um outro objeto chamado Mulher poderia ser criado a partir da classe derivada 2 com as seguintes características: cabelo, boca, pernas e brinco; que executaria as seguintes atividades: andar, comer, dormir e estudar.

Podemos então dizer que a classe base seria a classe de seres humanos e as suas derivadas 1 e 2 seriam as classes de homens e mulheres.

2.4 Estrutura do Programa

Seguindo os conceitos da programação orientada a objeto, foram criadas as classes que originaram os objetos que compõem o ambiente gráfico Modi :

- Classe do documento - CModiDoc: classe responsável pelos parâmetros de cada documento iniciado ou aberto no sistema. O Modi é um ambiente multi-documentos[11] e, com isso, o usuário pode editar vários projetos ao mesmo tempo. Deve-se, contudo, evitar a abertura de vários documentos pois isso pode causar uma certa confusão devido ao grande número de janelas que o usuário terá que controlar.

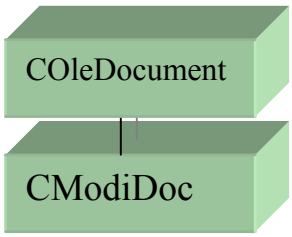


Fig. 2.3: Classe CModiDoc.

Dentre as informações que a classe CModiDoc carrega, podemos destacar as seguintes:

a) Lista dos objetos componentes do fluxograma (CModiObjList);
b) Lista geral dos ícones do documento (ti_lista);
c) Lista dos objetos componentes da janela de esquemático (CSchObjList);

d) Lista dos símbolos (constantes, variáveis, etc) utilizados (ts_lista);
e) Lista dos medidores e atuadores ativos do documento (CMedList e CAtuaList);
f) Ponteiros para todas as vistas do documento (CModiView, COnagroView e CSchView).

Das funções desempenhadas por esta classe podemos destacar:

a) Criar um novo documento;
b) Varrer as listas de objetos;
c) Adicionar ou remover objetos das listas.

- Classes das vistas - CModiView, COnagroView, CSchView: estas classes representam as vistas das três janelas de edição do sistema que são:

  - janela de edição do fluxograma(CModiView),
  - janela de edição de ícones (COnagroView) e
  - janela de edição do esquemático (CSchView).

Cada documento, portanto, pode ter até três janelas de edição abertas ao mesmo tempo.

Estas classes de janelas são derivadas da classe CScrollView da MFC que permite que a janela possua recursos de "scrolling", ou seja, que a janela permita que seu conteúdo possa ser maior que o mostrado pela área da vista. As partes que ficam ocultas podem ser vistas deslocando os botões "sliders" tanto verticalmente como horizontalmente.

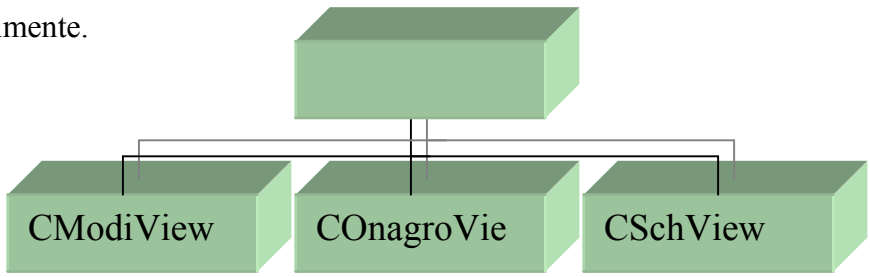
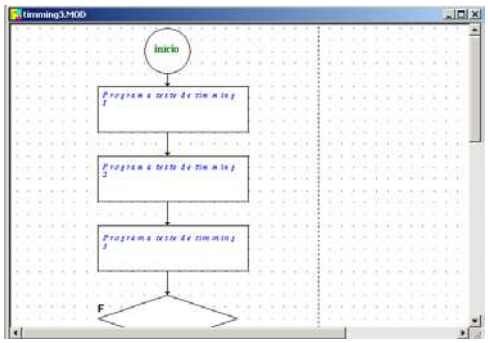


Fig. 2.4: Classes das vistas.

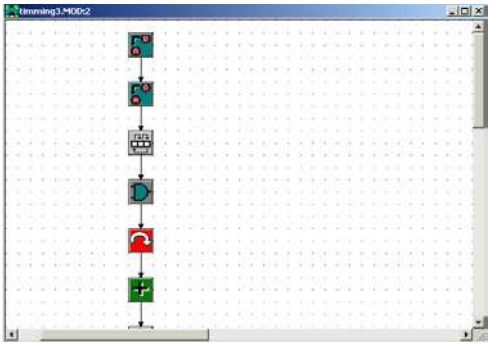
Estas vistas trazem informações tais como:

CModiView



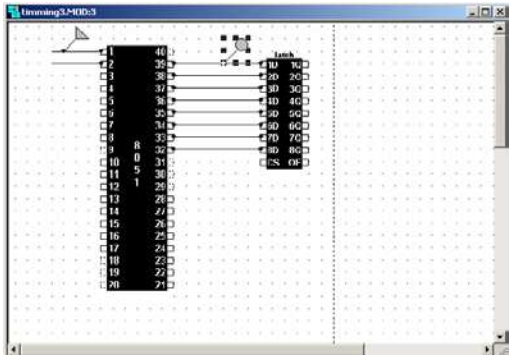
a) Item corrente selecionado;
b) Tamanho do retângulo a ser redesenhado;
c) Tamanho da grade;
d) Flag de ponto dentro de algum item.

COnagroView



a) Ponto corrente na janela;
b) Ícone capturado;
c) Tamanho de grade;
d) Flag de indicativo de simulação.

CSchView



a) Flag que indica que a janela tem o foco do windows;
b) Lista dos elementos dispostos na janela;
c) Flag para grade on/off;
d) Ponto atual na janela.

Dentre as tarefas desempenhadas por estas classes poderemos destacar as seguintes:

a) Arrastar as figuras para a grade;
b) Indicar qual figura está sendo simulada;
c) Redesenhar os itens que foram movimentados na tela;
d) Estabelecer o tamanho da página de tela em uso;

e) Selecionar objetos na tela;

f) Ativar os indicadores de progresso e de simulação. (ver figura 2.5)

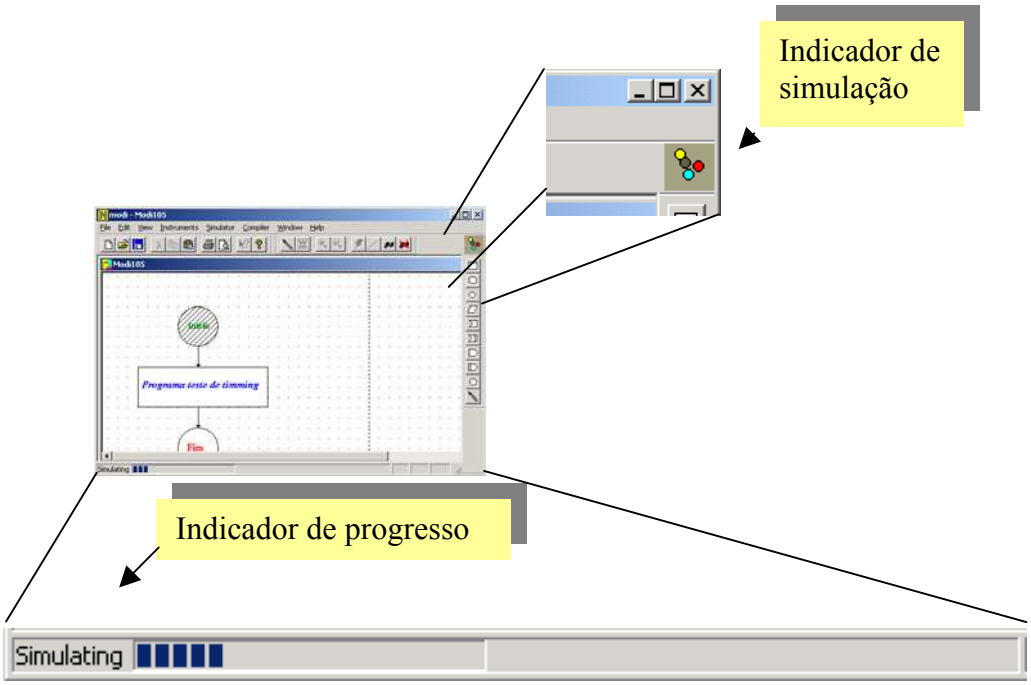


Fig.2.5: Indicadores de progresso e simulação.

- Classes das frames - CMainFrame, CChildFrame, CChildFrame2 e CChildFrame3: representam a frame (quadro ou moldura) da janela principal e das janelas secundárias das vistas de edição de fluxograma, esquemático e ícones, respectivamente.

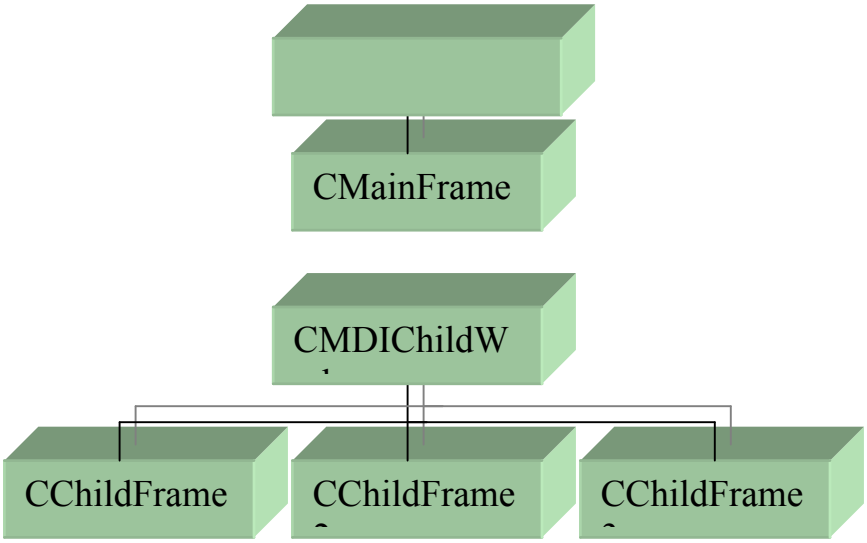


Fig. 2.6: Classe das frames.

Estas frames trazem as seguintes informações:

CMainFrame

a) Ponteiro para barras de ferramentas (CToolBar);
b) Ponteiro para barra de status (CStatusBar);
c) Ponteiro para barra de palheta (CPaletteBar);

CChildFrame, CChildFrame2 e CChildFrame3

Não possuem membros de dados significativos.

As funções mais importantes dessas classes são:

a) Criação da "frame" principal e das secundárias;
b) Criação das barras de ferramentas;
c) Ocultação das barras de ferramentas;

- Classe da aplicação - CModiApp: esta classe contém todas as inicializações e definições referentes a aplicação. Ela pode ser considerada como a principal classe do sistema.

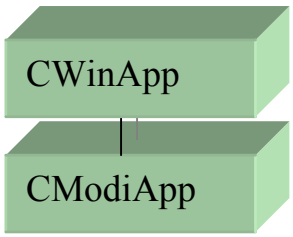


Fig. 2.7: Classe CModiApp.

Quando da inicialização da aplicação esta classe dá início a vários processos dentre os quais a criação da frame principal, da vista principal e do documento principal com seus

valores “default”. Ao se finalizar a aplicação, esta classe também é responsável pela destruição das classes principais, abrindo espaço na memória do computador.

As quatro classes acima formam um conjunto base para uma aplicação desenvolvida a partir do programa auxiliar de criação AppWizard do pacote Visual C++, ou seja, todo aplicativo desenvolvido usando este auxiliar terá um arquivo com a classe do documento, um arquivo com a classe de frame principal, um arquivo com a vista principal e um arquivo central da aplicação em sua constituição.

As outras classes específicas da aplicação que foram criadas fora do auxiliar de criação AppWizard e que são responsáveis pelas características funcionais do Modi estão descritas logo a seguir:

- Classe de medidores - CMedidors: classe base gerada a partir da classe genérica CObject para representar os instrumentos virtuais especificados pelas seguintes classes derivadas: vu, osciloscopio, termom, led, stepm, d7seg.

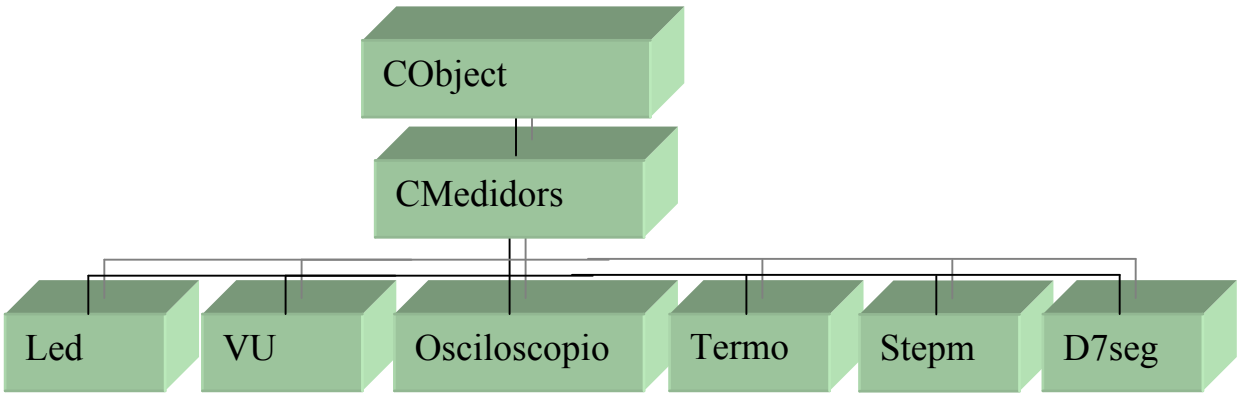


Fig. 2.8: Classe dos medidores.

Os medidores são usados para visualização das variáveis do sistema quando estamos rodando o simulador do Modi. Sua construção interna em termos de membros de dados é praticamente a mesma para todos os medidores, possuindo as seguintes informações:

a) Tipo do medidor;
b) Sua descrição;
c) Valor a ser indicado;
d) Ponteiro para a janela de diálogo correspondente;
e) Flag de estouro ou overflow.

Das funções desempenhadas pelos métodos das classes podemos destacar:

a) Criação e destruição do medidor;
b) Desenho do medidor;
c) Atualização da leitura;
d) Setar a variável observada.

- Classes dos diálogos dos medidores - dlg_digi, dlg_led, dlg_osc, dlg_step, dlg_term, dlg_vu: São classes derivadas diretamente de CDialog (da MFC) e que servem de interfaces gráficas para a representação dos instrumentos a elas associadas. Por exemplo, dlg_vu está associado com o medidor VU.

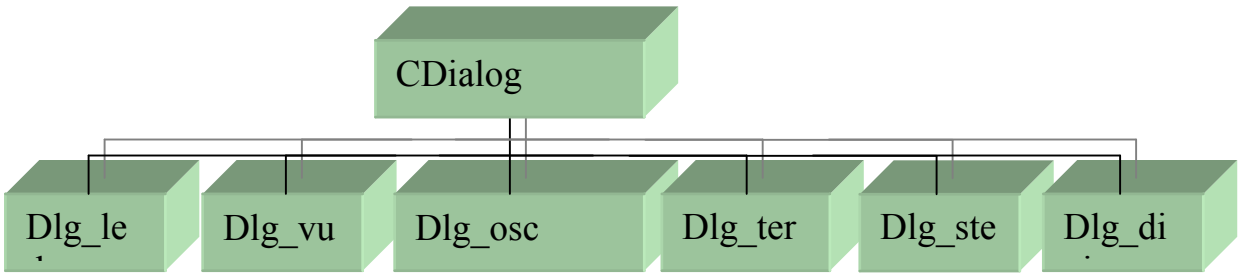


Fig. 2.9: Classes dos diálogos dos medidores.

Estes diálogos guardam as seguintes informações nos seus membros de dados:

a) Ponteiro para o medidor associado;
b) Valor da escala do medidor;
c) Variável associada ao medidor;

Suas funções básicas são particulares à cada tipo de medidor, contudo podemos destacar as mais comuns:

a) Inicialização do medidor;
b) Incremento ou decremento da medida;
c) Pintura do Bitmap do medidor.

- Classe de atuadores - CAtuadores: Esta classe dá origem a todos os atuadores utilizados para interagir com as variáveis do sistema em tempo de simulação, e engloba todos os atuadores representados pelas seguintes classes derivadas: lig_desl, teclado, interruptor e gerador.

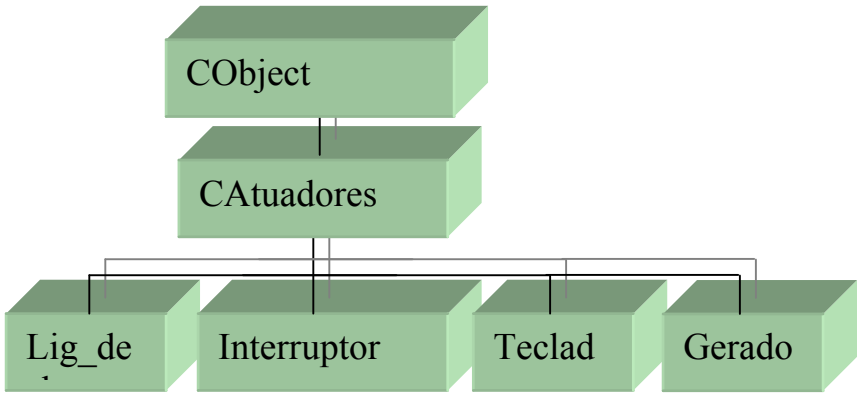


Fig. 2.10: Classe dos atuadores.

A estrutura desta classe se assemelha à classe dos medidores, tendo como informações relacionadas:

a) Tipo do atuador;
b) Sua descrição;
c) Ponteiro para a janela de diálogo correspondente;
d) Ponteiros para próximo e anterior na lista de atuadores, caso existam.

Suas funções podem ser resumidas em:

a) Criação e destruição do atuador;
b) Setar o estado do atuador;
c) Setar o objeto atuador associado.

- Classe de diálogos dos atuadores – dlg_I_O, dlg_lig_desl, dlg_tecl e dlg_gerador: Derivam-se diretamente de CDialog e têm finalidade semelhante aos diálogos dos medidores, ou seja, servem de interface gráfica para representarem as classes específicas de cada atuador.

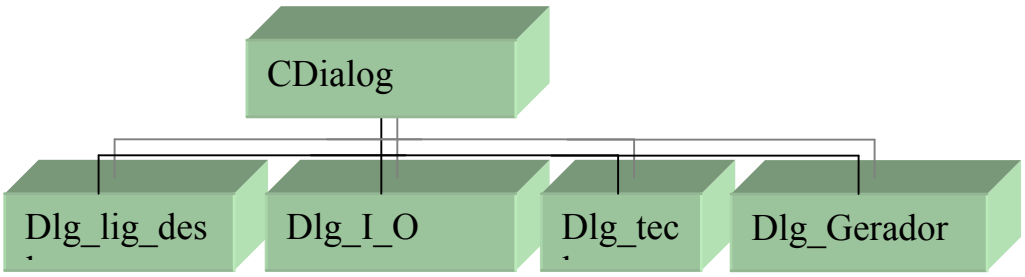


Fig. 2.11: Classes dos diálogos dos atuadores.

Cada classe de diálogo tem suas particularidades mas no geral cada um possui os seguintes membros de dados:

a) Ponteiro para o atuador associado;
b) Tipo de atuador;

As funções membros que fazem parte da classe de diálogo têm as seguintes características:

a) Inicialização do diálogo do atuador;
b) Envio de mensagens de estado;
c) Pintura do Bitmap de estado do atuador.

- Classe de listas - mi_lista, ti_lista, ts_lista, CModiObjList, COnagroViewList, CSchObjList, med_list, atua_list: O Modi guarda todas as suas principais informações sob a formas de listas. Estas listas guardam informações sobre as figuras componentes do fluxograma, dos ícones que compõe cada figura, dos símbolos (constantes, variáveis, portas, vetores, etc), do endereço das vistas onde visualizamos os ícones, dos componentes da vista de edição de esquemático, dos medidores e atuadores utilizados, respectivamente. Para criá-las derivamos cada uma delas da classe base da MFC apropriada para listas que é CObList. A figura abaixo representa melhor o que foi dito:

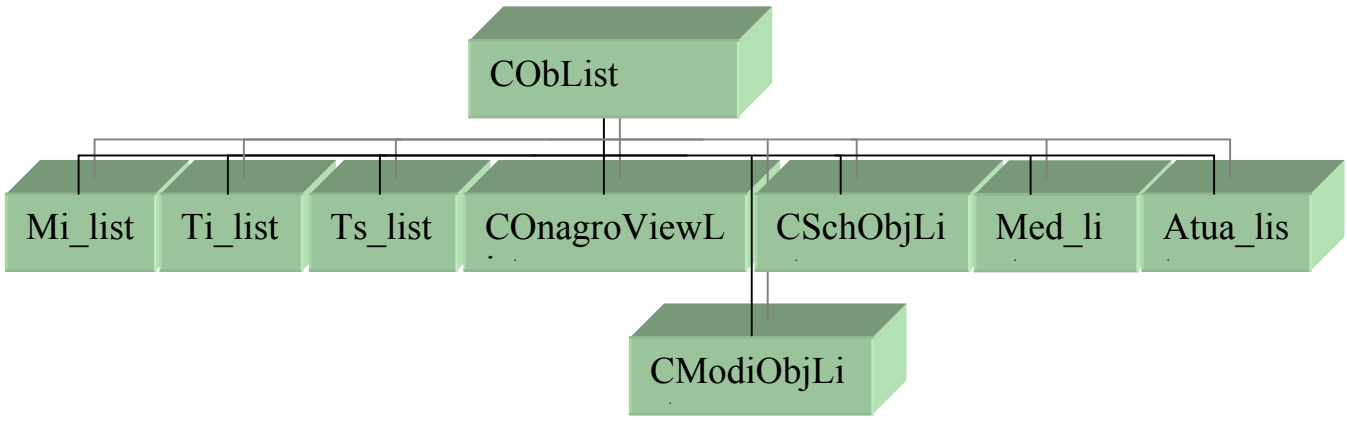


Fig. 2.12: Classe de listas.

Todas as classes de listas são muito parecidas nos seus membros de dados, possuindo aqueles membros que correspondem ao tipo de objeto que se pretende listar, ou seja, todas guardam ponteiros para o objeto a ser listado. Por exemplo, a classe Med_list tem como membro de dados os ponteiros para os medidores chamados pela aplicação.

Quanto às funções membros, estas também são bastante comuns entre todas as classes das listas. Podemos, então, destacar algumas delas:

a) Inserir objeto no fim da lista;
b) Pesquisar por algum objeto na lista;
c) Calcular o número de elementos;
d) Excluir objeto da lista.

- Classe CModiRect: classe base para representação das figuras da janela de edição de fluxograma. Cada figura é criada segundo as seguintes classes derivadas: mf_circulo, mf_external, mf_folha, mf_internal, mf_losango, mf_outextern, mf_outintern, mf_retangulo, mf_save.

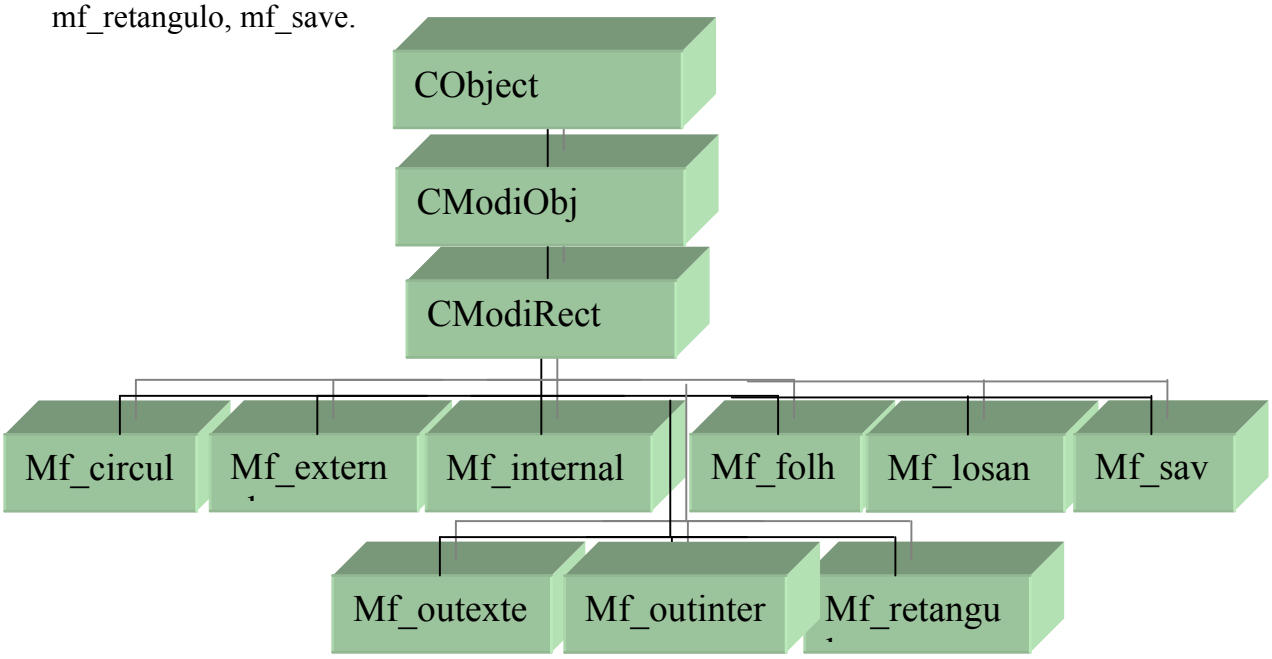


Fig. 2.13: Classe CModiRect.

Dentre os membros de dados mais comuns entre as classes acima podemos destacar os seguintes:

a) O tipo do objeto;
b) O índice dentro da lista;
c) Posição do objeto na tela;
d) Os índices dos objetos a ele conectados.

As classes derivadas possuem funções membros muito semelhantes, sendo as mais importantes as seguintes:

a) Setar cor e espessura da linha de desenho;
b) Apagar objeto da tela;
c) Mover o objeto na tela;
d) Simular o objeto selecionado.

- Classe t_icon: classe base que engloba todos os ícones da janela de edição de diagrama de ícones. Cada ícone está representado pelo tipo de função que ele desempenha, com isso foram criadas as seguintes classes derivadas: ti_aritmetica, ti_atribuição, ti_comparação, ti_comunicacao, ti_conector, ti_degrau, ti_delay, ti_juncao, ti_juntar_bits, ti_logica, ti_loop, ti_pesq_vet, ti_shift_rot, ti_start, ti_stop, ti_sub_rotina, ti_user, ti_xlat_vet.

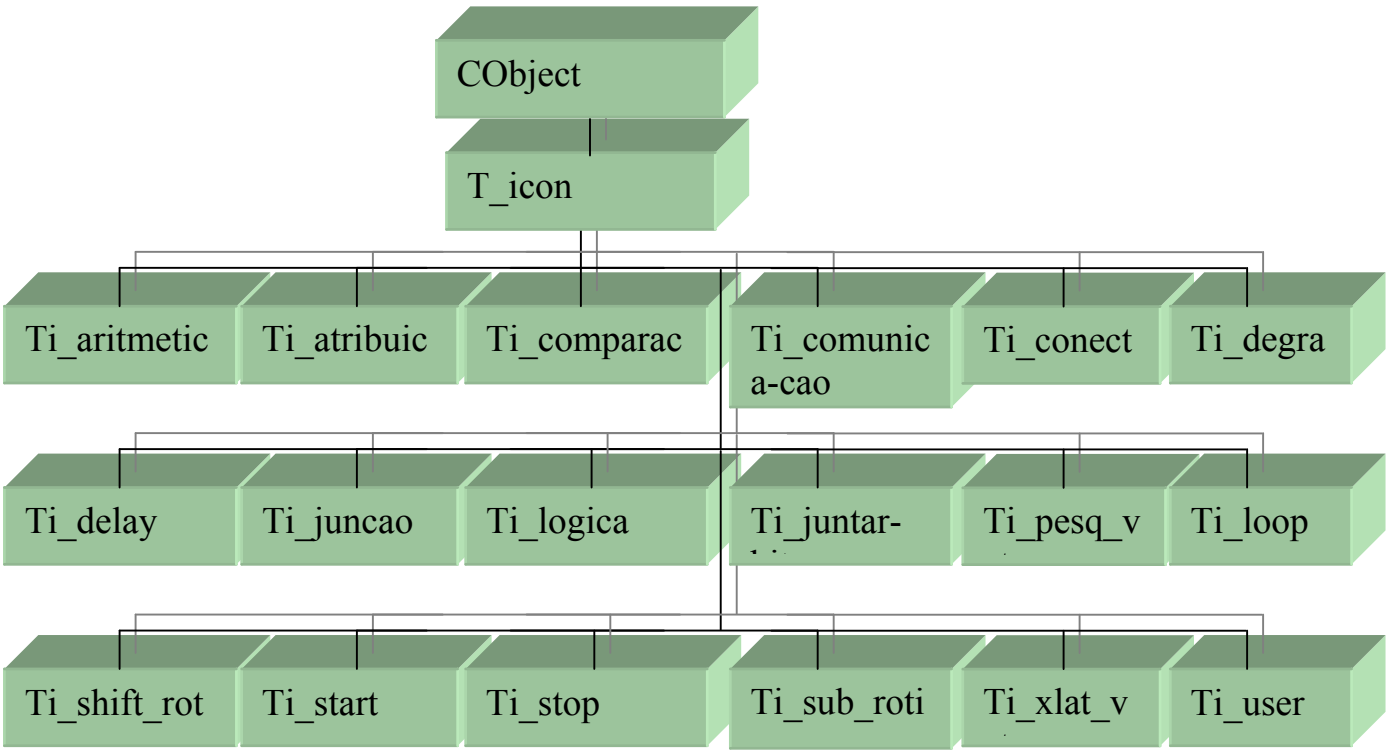


Fig. 2.14: Classe t_icon.

Estas classes derivadas trazem os seguintes membros de dados:

a) Descrição do ícone;
b) O tipo correspondente;
c) O ponteiro do objeto de origem;
d) O ponteiro para o próximo ícone da lista.

As principais funções membros de cada classe derivada estão descritas a seguir:

a) Criação do objeto;
b) Fazer as ligações entre os ícones;
c) Compilar o ícone;
d) Simular o ícone.

Como podemos ver pelas descrições acima, os ícones são capazes de gerar os próprios códigos correspondentes e de simulá-los quando solicitados. Isto está dentro da metodologia de programação que sugere que devemos criar objetos que sejam independentes, ou seja, que possam por eles mesmos atuarem de maneira plena em suas funções específicas.

- Classe CSchRect: classe base que origina todas as outras classes da vista de edição de esquemático, representada pelas classes: CChipObj, CLatchObj, CBusObj, CWireObj, CFlipflopObj, CObserObj, CAtuaObj.

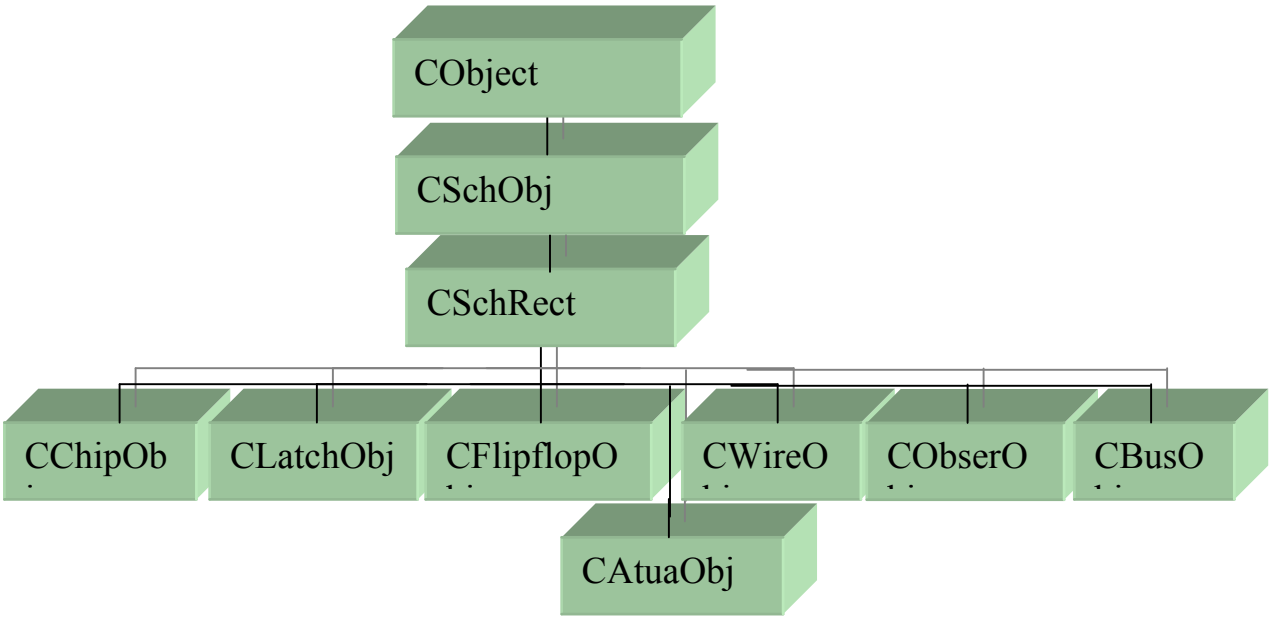


Fig. 2.15: Classe CSchRect.

Cada classe se distingue ligeiramente da outra, em seus membros de dados, mas no geral todas apresentam os seguintes membros:

a) A forma do objeto;
b) O seu "label";
c) A sua posição.

Quanto as principais funções membros de cada classe derivada, poderíamos destacar as seguintes:

a) Desenhar objeto na vista de edição de esquemáticos;
b) Editar linha e cor de preenchimento;
c) Abrir propriedades;
d) Movimentação pela tela.

- Classe de Status: Esta classe traz informações sobre o status dos componentes da janela de edição de esquemático, ou seja, traz o estado atual dos elementos CChipObj, CLatchObj, CWireObj e CbusObj sob a forma de diálogo de propriedades.

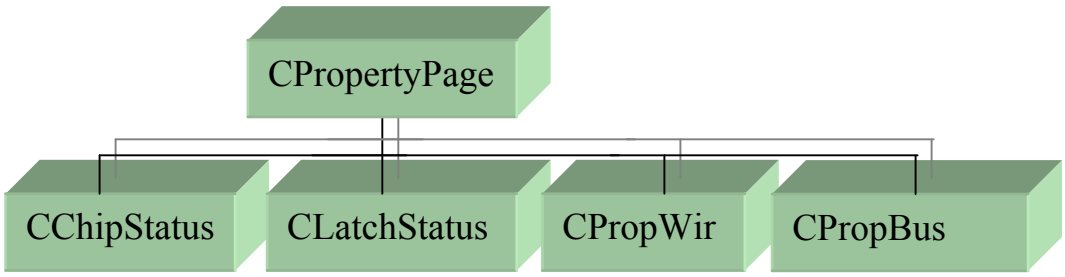


Fig. 2.16: Classe de status.

Estas propriedades são muito úteis de serem consultadas quando estamos simulando a aplicação, pois serve para acompanharmos os sinais que transitam pelo diagrama esquemático em questão.

- Classe dos processos: Processos que correm em paralelo são chamados Threads. No Modi temos duas classes que executam tarefas em paralelo, ou seja, não interrompem os processos correntes do Windows tais como movimentação do mouse, leitura de teclado, movimentação de janelas, etc. Estas classes são as seguintes: CModiThread e COnagroThread.

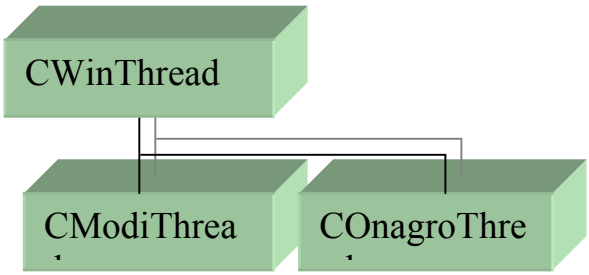


Fig. 2.17: Classe dos processos.

A Classe CModiThread é responsável pela simulação dos blocos que compõem a janela de edição de fluxograma. Não possuindo nenhum membro de dado significativo e tendo como função membro apenas aquela que realiza a simulação dos objetos da vista de fluxograma. Esta função membro faz a varredura da lista Mi_lista que contém os objetos da vista de fluxogramas, chamando a função de simulação de cada um deles e indica visualmente qual objeto está sendo simulado. Esta "Thread" é chamada tanto no modo de simulação automática como passo-a-passo.

A Classe COnagroThread responde pela simulação dos ícones da janela de edição de ícones. Ela também não possui membros de dados significativos, e sua função membro específica é a que simula os ícones da janela de edição de ícones. O seu funcionamento é semelhante ao da classe CModiThread, ou seja, sua função membro faz a varredura da lista de ícones Ti_lista correspondente a cada bloco de fluxograma ativo e chama a função de simulação de cada ícone individualmente, indicando na vista COnagroView qual ícone está sendo simulado. Esta "Thread" é chamada quando estamos simulando passo-a-passo dentro da janela de edição de ícones.

- Classe CSchTool: classe que controla a edição dos elementos da vista de edição de esquemático e que dá origem as classes: CSelectTool e CRectTool.

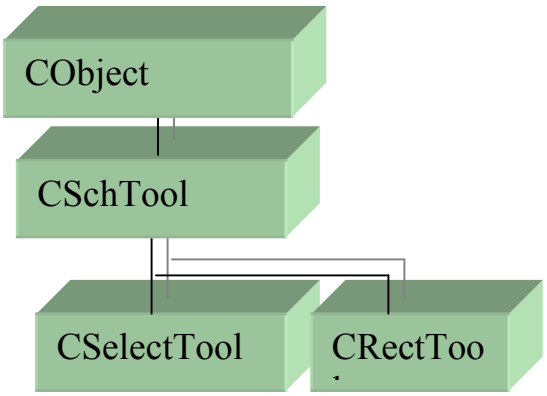


Fig. 2.18: Classe CSchTool.

Estas classes controlam a movimentação dos objetos CModiObj da janela de edição de esquemáticos, sendo responsáveis pela indicação de qual objeto foi selecionado pelo mouse e de indicar quando um objeto foi ou não conectado pelo elemento Wire, ou seja, toda vez que um fio (wire) encontra um pino de um elemento CModiObj, como por exemplo, o CChipObj, então ele indica com um pequeno círculo na terminação do fio conectado.

- Classe CSplashWnd: está classe é responsável pela tela instantânea de apresentação inicial do Modi, vista na figura 2.20 a seguir:

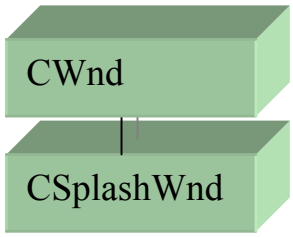


Fig. 2.19: Classe CSplash Wnd.

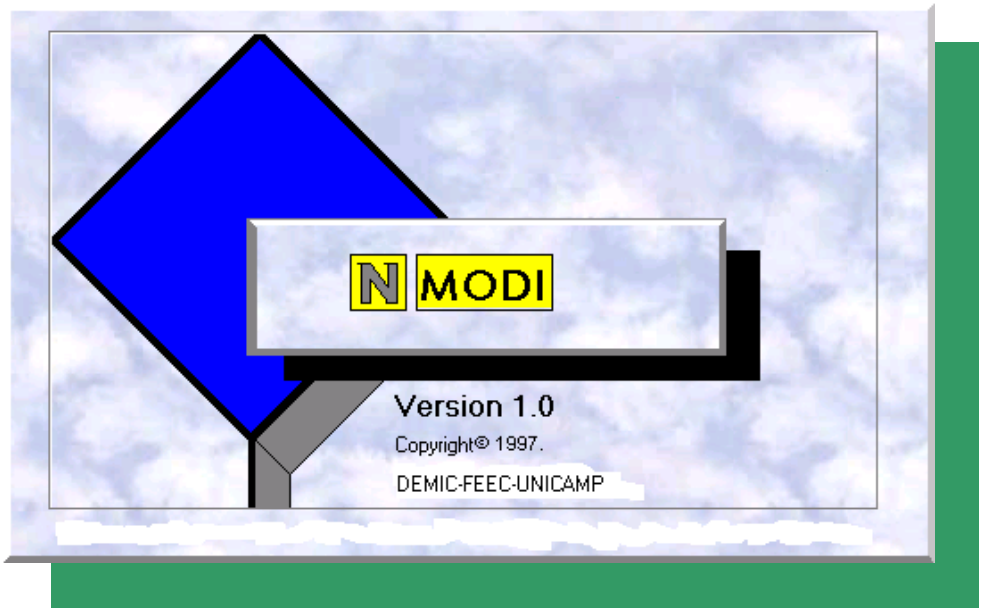


Fig.2.20: Tela de abertura do Modi.

- Classe CMicroC: Classe diretamente originada de CObject, a classe genérica da MFC. Esta classe representa o microcontrolador em uso pelo ambiente Modi. Nesta primeira versão este microcontrolador é o 8051 da Intel.

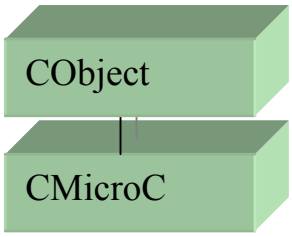


Fig.2.21: Classe CMicroc.

As informações contidas nesta classe são praticamente referentes ao hardware do microcontrolador, tais como:

a) Posições da RAM interna;
b) Os SFR's (registradores especiais);
c) O clock do microcontrolador.

Esta classe não possui funções membro, funcionando apenas como um elemento passivo quando estamos fazendo uma simulação do aplicativo, sendo seus membros de dados atualizados por acesso direto aos mesmos.

2.5 Descrevendo o Simulador do Modi

Após a descrição das principais classes componentes do Modi, é conveniente mostrar os detalhes de um dos principais componentes do ambiente Modi que é o seu simulador.

O simulador possui três modos de funcionamento, e para cada um ele se comporta de maneira diferente. Vamos começar descrevendo o modo de simulação automático. Neste modo o simulador trabalha em modo contínuo, sem a intervenção do usuário, e funciona da seguinte maneira: O acionamento do botão  da barra de ferramentas envia uma mensagem para a função membro `OnSimulaAuto` da classe CModiView, Esta, por sua vez, verifica todas as ligações entre os componentes do fluxograma e dispara a classe CModiThread, que funciona em paralelo ao processo do Modi, não travando o aplicativo dentro do Windows; a classe CModiThread, então, faz uma leitura da classe de listas Mi_lista que contém a relação dos elementos de fluxogramas CModiObj e faz a chamada da função membro `t_simule` de cada elemento. Esta função membro faz a leitura da lista de ícones Ti_lista que contém a relação de ícones de cada elemento de fluxograma chamando a função membro `t_simula_cod` que faz a simulação do ícone de operação. A figura 2.22 traz um resumo do que foi dito.

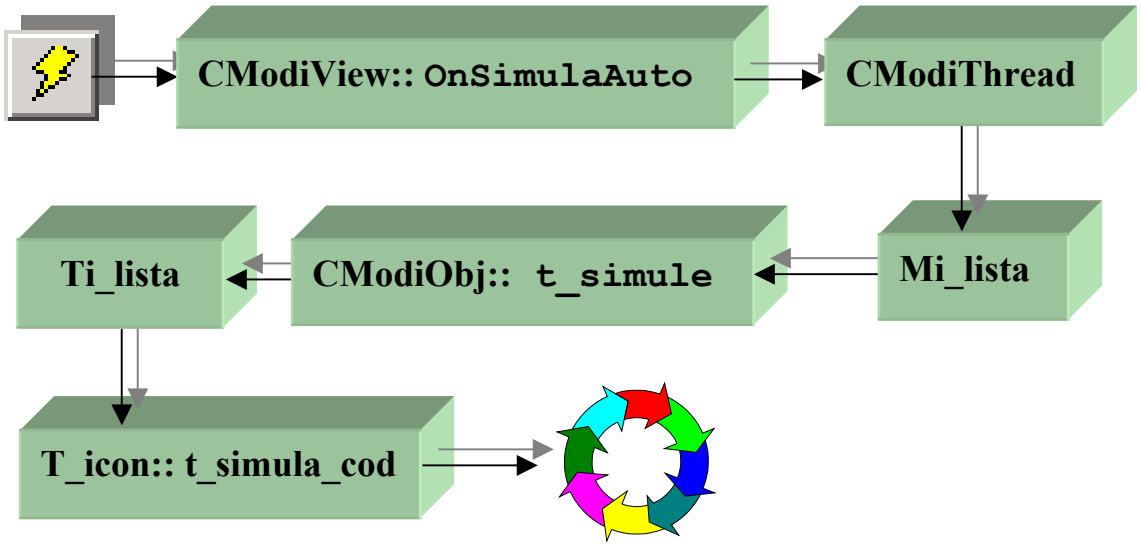


Fig.2.22: Simulação em modo automático.

O próximo modo de simulação é a passo-a-passo dentro do fluxograma. Neste modo o usuário pode fazer a simulação bloco-a-bloco controlando o fluxo de simulação através do botão  da barra de ferramentas. O funcionamento deste modo é descrito a seguir:

Ao acionarmos o botão  da barra de ferramentas, uma mensagem é enviada para a classe CModiView chamando o método `OnSimulaStep` iniciando o processo de simulação. Esta função faz a atualização das ligações dos elementos do fluxograma e prepara os flags de simulação. Os próximos passos são dados através do botão  da barra de ferramentas. Ao ser pressionado este envia uma mensagem para a classe CModiView chamando o método `OnSimula` que, por sua vez, acessa a lista Mi_lista buscando pelo elemento de fluxograma a ser simulado. Em seguida o elemento selecionado CModiObj chama a sua função membro `t_simule` que faz a simulação do elemento de fluxograma. Esta função dispara a classe COnagroThread, que, como o nome indica, é uma "thread" que corre em paralelo com o Modi. A classe COnagroThread chama a sua função membro `SimulaOnagro` que faz uma varredura da lista de ícones Ti_lista do bloco de fluxograma e aciona os ícones da classe t_icon, chamando as suas

funções membro `t_simula_cod` que fazem a simulação de cada ícone. A figura 2.23 ajuda na compreensão do que foi dito.

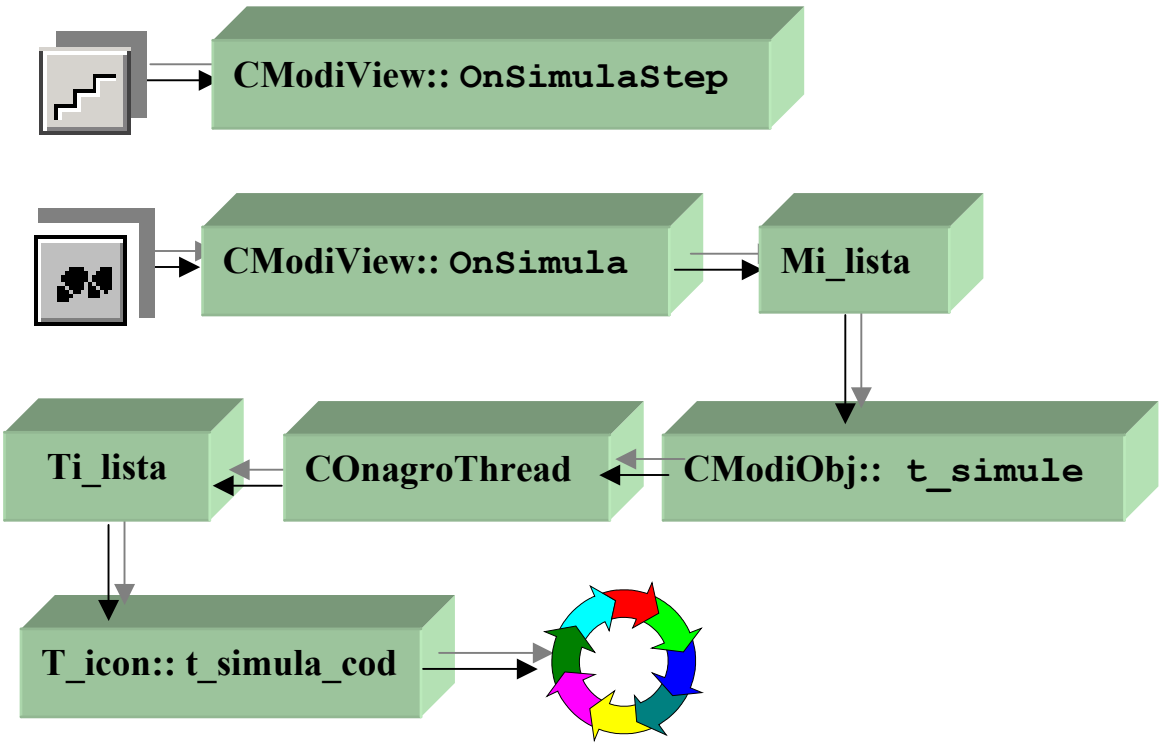


Fig.2.23: Simulação em modo passo-a-passo.

O terceiro e último modo de simulação é o passo-a-passo dentro do diagrama de ícones. Neste modo podemos fazer a simulação individual de cada ícone, podendo, com isso, controlar melhor o comportamento das variáveis do aplicativo em desenvolvimento. Este controle se dá através, novamente, do botão  da barra de ferramentas.

Este modo de simulação funciona da seguinte maneira: Para que este modo de simulação aconteça precisamos ter pelo menos uma janela de edição de ícones aberta para habilitarmos o item de menu "step-by-step – Onagro". Quando selecionamos este item de menu enviamos uma mensagem para a classe COnagroView que responde chamando a sua função membro `OnSimulaStep2`, esta função chama a classe CModiObj que é a origem da janela de ícones escolhida e ativa a sua função membro `t_simule` . Esta

função membro acessa a classe de listas Ti_lista relativa ao elemento de fluxograma correspondente e ativa a classe COnagroThread que, por sua vez, chama as classes t_icon componentes da lista e acionam a sua função membro `t_simula_cod` e entram em loop de espera pelo flag de controle `simula_step_onagro` que é modificado sempre que é acionado o botão  , passando para a simulação do ícone seguinte. A figura 2.24 traz um diagrama que explica melhor o que foi dito.

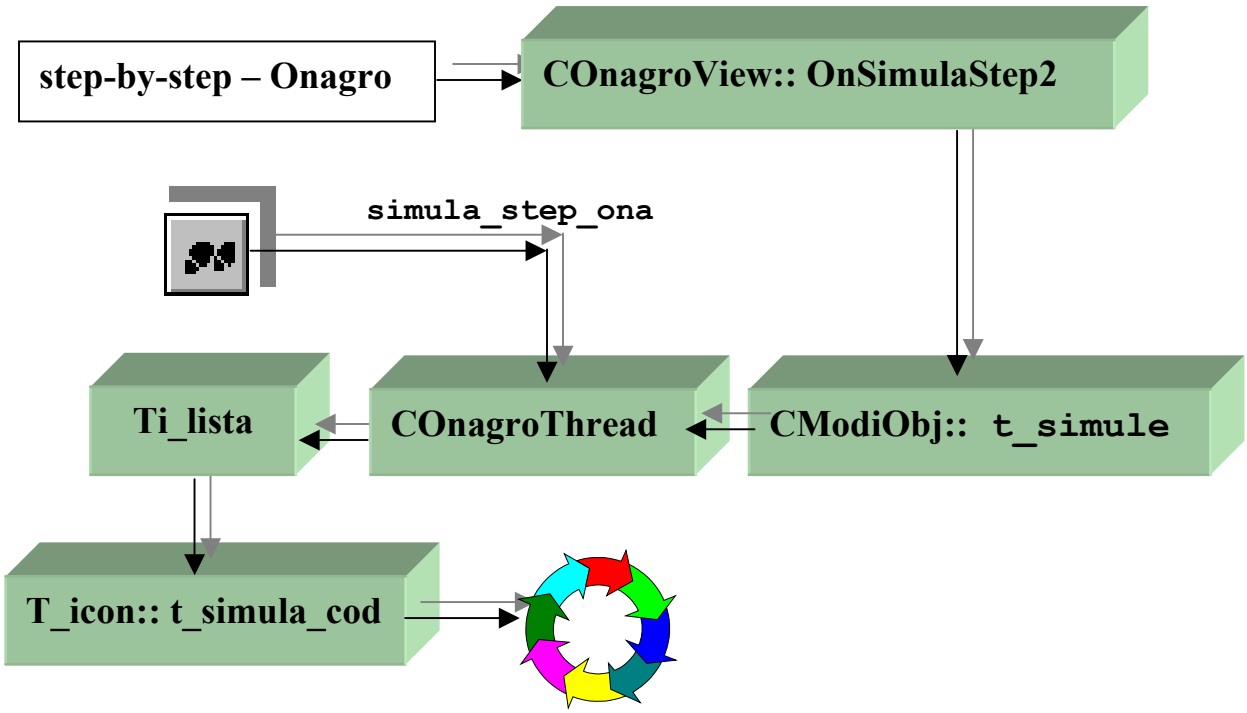


Fig.2.24: Simulação em modo passo-a-passo para ícones.

Todos os modos de operação encerram-se pressionando-se o  botão na barra de ferramentas. Este botão atua diretamente sobre o flag `stop_simula` que controla todos os "loops" de simulação contidos nas "threads" do Modi.

2.6 Descrevendo o Compilador de ícones do Modi

Um outro módulo importante no Modi é o seu compilador de ícones [4]. Ele funciona de modo semelhante ao módulo simulador. E o seu funcionamento é o que passamos a descrever a seguir:

Ao selecionarmos o item de menu Compiler e o seu sub-item Start estamos enviando uma mensagem para a classe CModiView que responde acionando a sua função membro `OnCompilarStart` que faz uma varredura na lista Mi_lista atualizando as ligações entre os elementos do fluxograma, CModiObj, e faz uma outra varredura na lista de ícones Ti_lista que por sua vez ativa uma chamada a função membro `tic_setprox` para estabelecer o encadeamento dos elementos t_icon componentes desta lista. Feito todo os encadeamentos entre todos os ícones de todos os blocos de fluxograma, é hora de chamarmos a classe t_dlg_gc que através de sua função membro `OnCompilar` dá início a criação da String principal que conterá todo o código a ser gerado. Esta função realiza ainda uma chamada a função membro `ti_compilar` da lista Mi_lista que realiza a concatenação de partes de código gerada a partir de chamadas a função membro `t_gera_cod` de seus membros de dados t_icon. O resultado mostrando a String principal aparece numa janela de diálogo chamada no final do processo. O que pode ser compreendido de tudo o que foi dito é que o compilador faz uma varredura nas listas de ícones e vai gerando uma string contendo todos os códigos gerados por cada ícone concatenados. A figura 2.25 mostra toda a seqüência de operações do compilador.

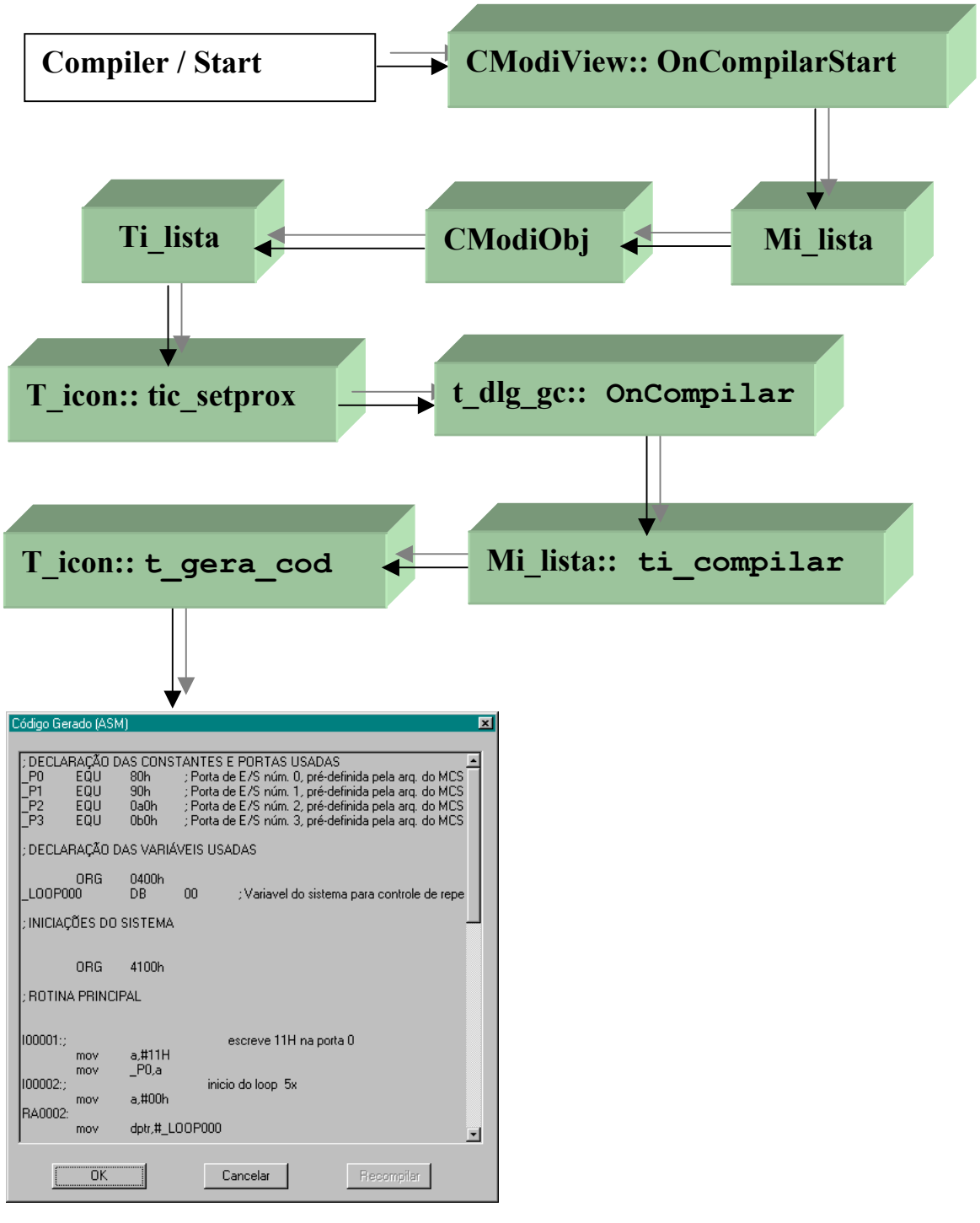


Fig.2.25: Compilação dos ícones.

Pelo o que foi exposto, o leitor já tem uma noção clara do funcionamento interno do ambiente MODI e dos potenciais usos que se possa fazer desta ferramenta. Resta agora aprender mais sobre a API e como utilizar a ferramenta na prática. Para isso passaremos para o próximo capítulo que servirá para complementar as informações de como utilizar melhor a ferramenta MODI.

Capítulo III: Descrição do Ambiente Modi

3.1 Introdução

Neste capítulo é feita uma descrição sobre o funcionamento do programa Modi. Fazemos uma descrição de suas principais janelas de edição e das respectivas barras de ferramentas e itens de menu. Também apresentamos um pequeno tutorial onde é mostrado em detalhes as etapas para a geração de um programa simples, no caso um programa para controlar um motor de passo, dentro da ferramenta Modi.

3.2 Executando Modi.exe

Ao ativarmos a aplicação Modi.exe através do explorer do Windows 9x ou NT, após a tela de inicialização com o logotipo do programa, o que iremos ver é uma tela semelhante a apresentada na figura 3.1 abaixo.

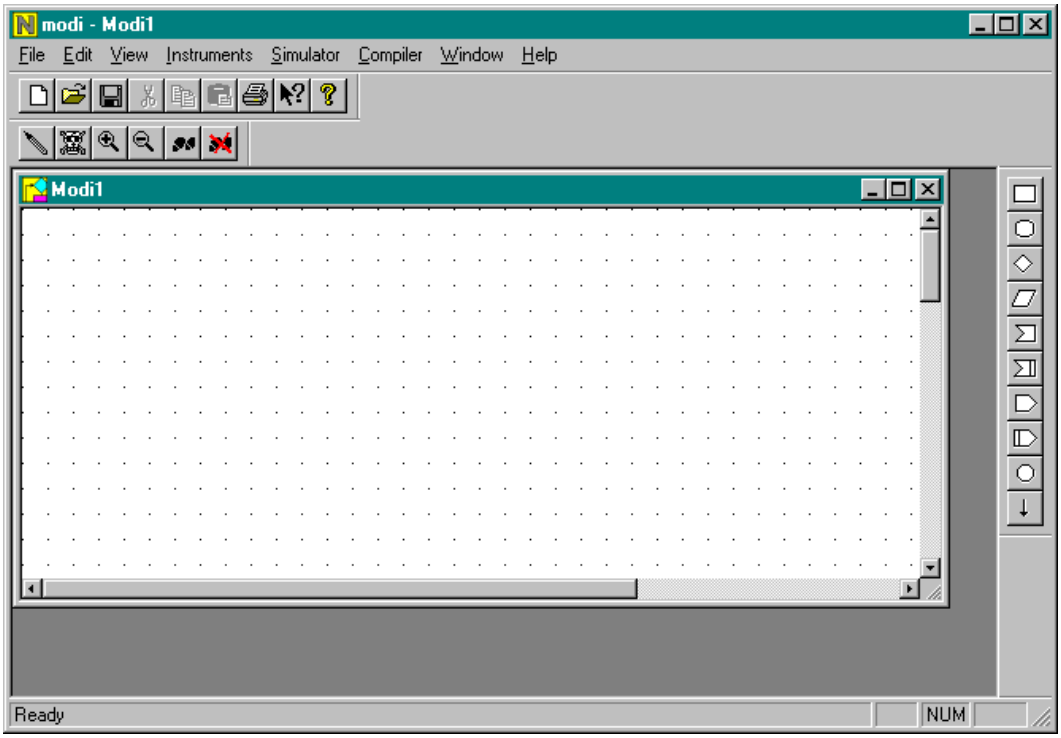


Fig.3.1: Janela principal do Modi.

Nesta figura, podemos ver os elementos básicos que compõem a interface gráfica e que dividem-se nas seguintes partes:

a) Menu Principal: O menu principal se localiza na parte superior da janela logo abaixo do título da janela e é composto pelos seguintes itens:

1) File: Este item de menu controla as operações de inicialização de um projeto ou recuperação de projetos já gravados anteriormente, bem como a impressão das vistas ativas, através dos seguintes sub-itens

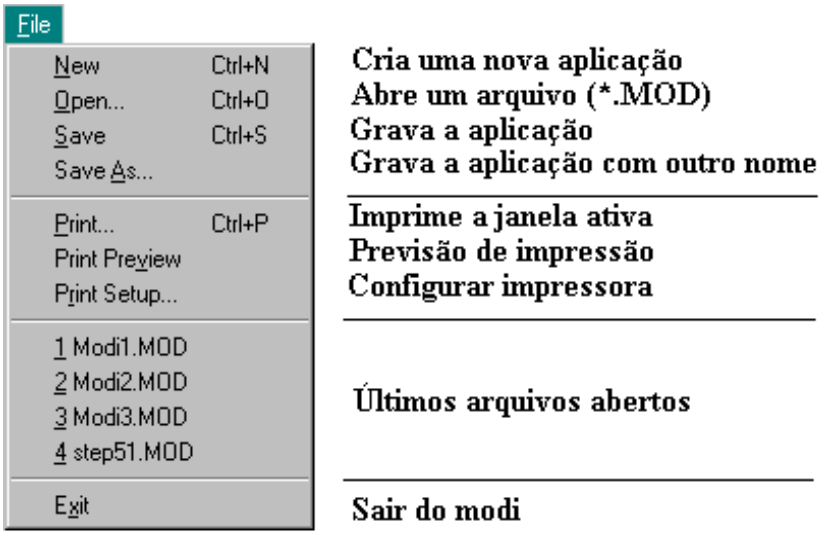


Fig.3.2: Menu File.

2) Edit: Controla as operações mais comuns de edição, sendo composto pelos seguintes sub-itens:

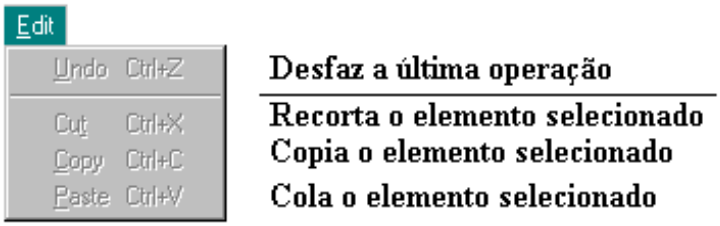


Fig.3.3: Menu Edit.

3) View: Controla a exibição das barras de ferramentas e abertura da janela de edição de esquema elétrico.

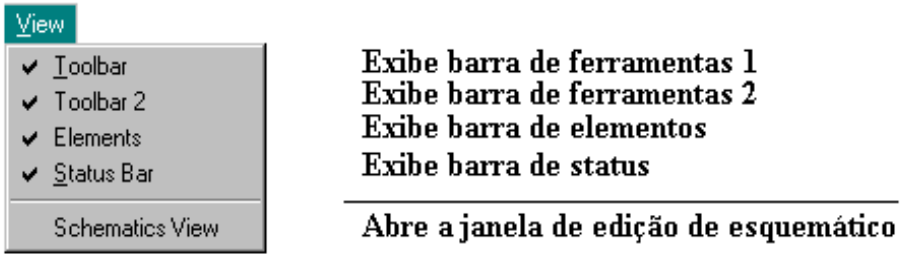


Fig.3.4: Menu View.

4) Instruments: Controla a chamada dos medidores e atuadores utilizados para visualizar e interagir quando o sistema está operando no modo de simulação do código.

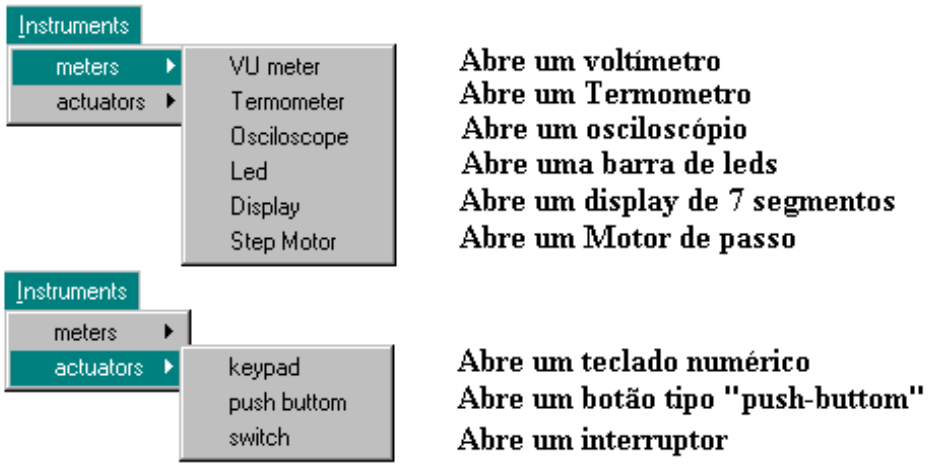


Fig.3.5: Menu Instruments.

5) Simulator: Põe o sistema para trabalhar no modo de simulação do projeto, podendo estabelecer as seguintes situações:

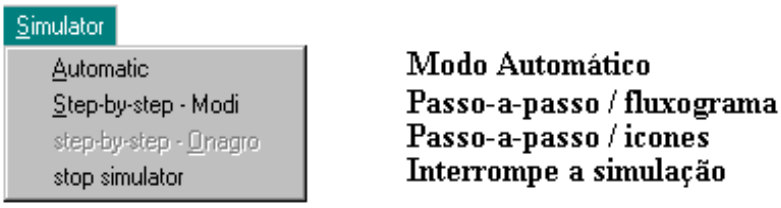


Fig.3.6: Menu Simulator.

- Modo de simulação automático: Este modo de operação pode ser selecionado através do item de menu mostrado acima, ou através do botão da barra de ferramentas. Quando no modo automático, o simulador faz a simulação do projeto seguindo a seqüência de operação determinada pelo fluxograma sem o controle do usuário, indicando no mesmo fluxograma o bloco que está sendo simulado.

- Modo passo-a-passo / fluxograma: É ativado através do menu Simulator ou pelo botão na barra de ferramentas. No modo passo-a-passo / fluxograma, o usuário inicia a simulação do projeto, mas o sistema somente passa para o próximo bloco do fluxograma se o usuário pressionar o botão de avanço de passo na barra de ferramentas. Este modo é bastante usado quando queremos acompanhar o comportamento dos blocos de menu com mais controle da situação, geralmente isto acontece quando estamos "debugando" a aplicação.

- Modo passo-a-passo/ícones: Apresenta um funcionamento similar ao caso anterior, contudo o usuário deve estar com a janela de ícones correspondente a um determinado bloco do fluxograma aberta, com isso ele pode visualizar que ícone de instrução está sendo executado naquele momento. Novamente o usuário deve pressionar o botão na

barra de ferramentas para avançar para o próximo ícone a ser simulado. Este modo permite um nível de "debugging" maior, pois as operações não aconteceram em blocos mas ícone a ícone aumentando o controle sobre as operações realizadas.

O item "stop simulator", conforme o nome sugere, interrompe a simulação corrente, podendo ser também ativado através do botão  na barra de ferramentas.

6) Compiler: Engloba os sub-itens que tratam da compilação do código gerado a partir dos ícones da janela de edição de ícones.

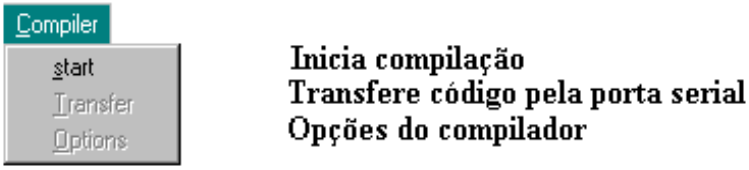


Fig.3.7: Menu Compiler.

Ao ativarmos o item Start do menu Compiler estamos pedindo ao Modi para fazer a compilação para a linguagem *Assembly* das listas de ícones correspondentes a cada bloco de fluxograma, convertendo assim, o algoritmo representado graficamente para uma linguagem textual de mais baixo nível para que esta possa ser "assemblada", ou seja, convertida novamente para a forma hexadecimal que é a que o microcontrolador entende.

7) Window: Este item é mais ou menos padrão entre as aplicações "for Windows" e faz o controle dos itens de distribuição das janelas na "frame" principal do sistema. Ela possui os seguintes sub-itens auto-explicativos:

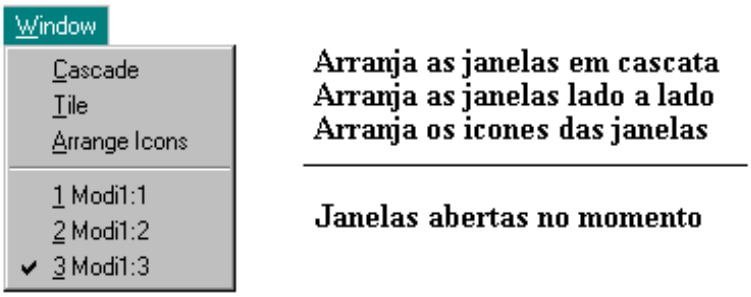


Fig.3.9: Menu Window.

8) Help: Este item traz informações sobre a versão do Modi, como se trata de uma versão inicial, esta é denominada 1.0.

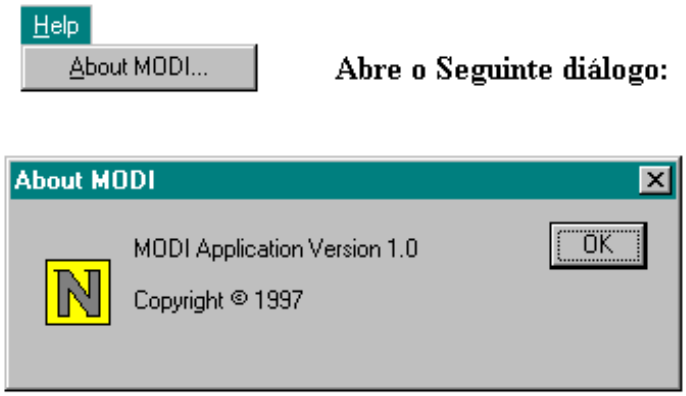


Fig.3.10 : Menu Help.

b) Barras de ferramentas: As barras de ferramentas estão localizadas logo abaixo do menu principal e no lado direito da área de trabalho. Alguns itens das barras de ferramentas são os mesmos do menu principal, por isto estas podem ser consideradas como botões de atalho aos itens desejados. Veja as descrições dos itens logo a seguir:

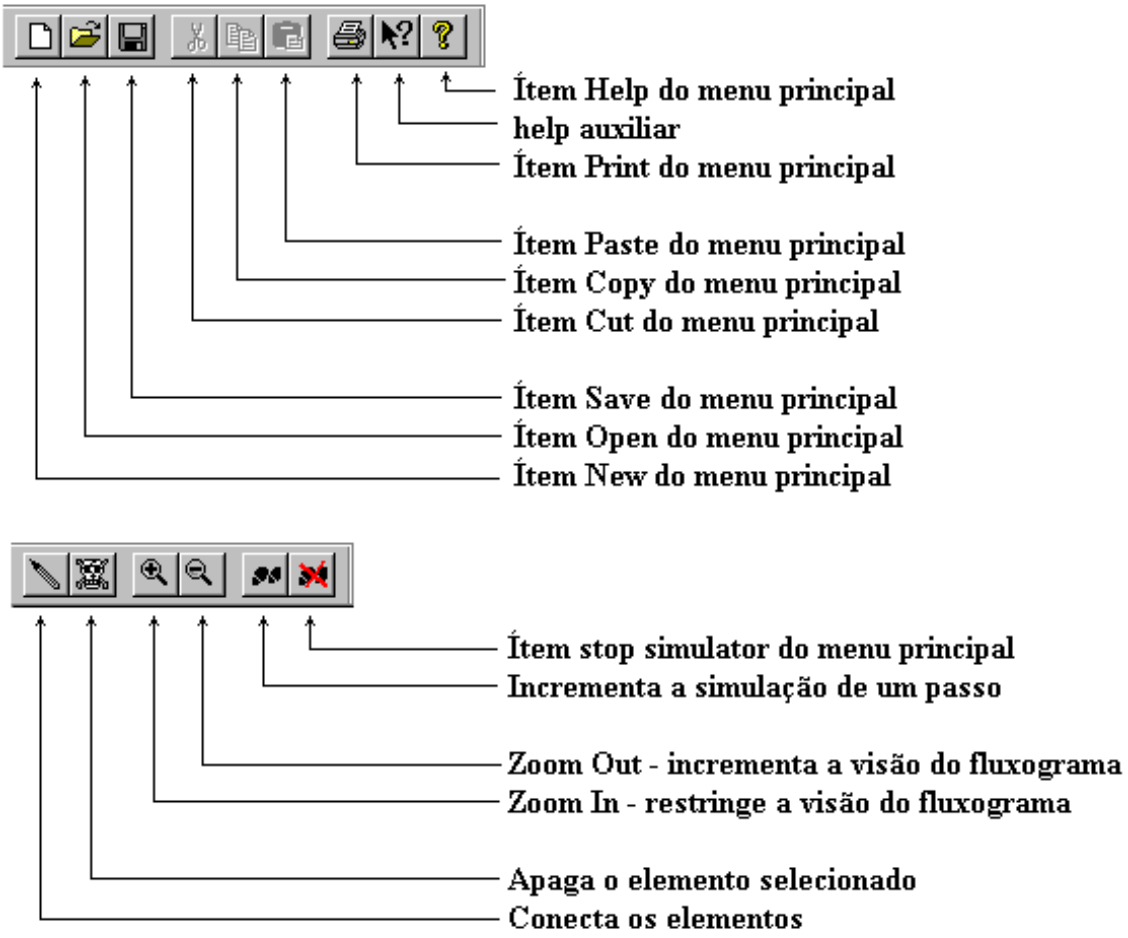


Fig.3.11: Barras de ferramentas.

A seguir podemos ver as barras de ferramentas mais específicas do Modi, ficando estas localizadas na vertical no lado direito. Contudo, estas barras de ferramenta tem a característica de "docking", ou seja, elas podem ser destacadas de suas posições originais e se tornarem uma janela independente, flutuando pela frame principal.

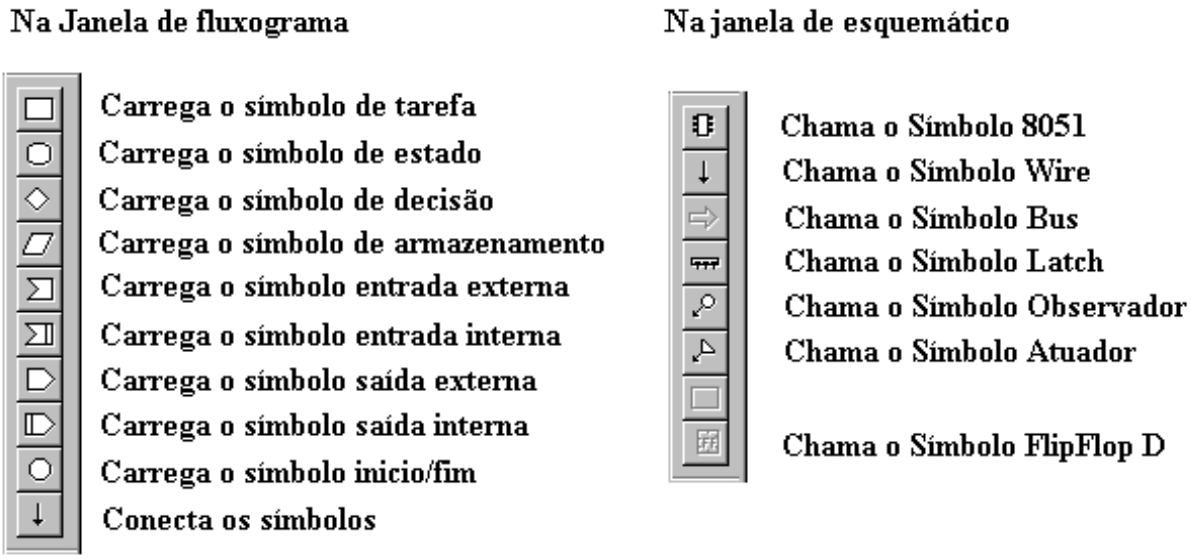


Fig.3.11: Barras de ferramentas.(cont.)

Existe ainda uma barra de ferramentas que só aparece quando ativamos a vista de edição de ícones. Ela possui botões maiores que trazem uma imagem do ícone de operação que deverá ser inserido ao se pressionar cada botão. Esta barra de ferramentas também possui o recurso de "docking", podendo flutuar pela frame principal:

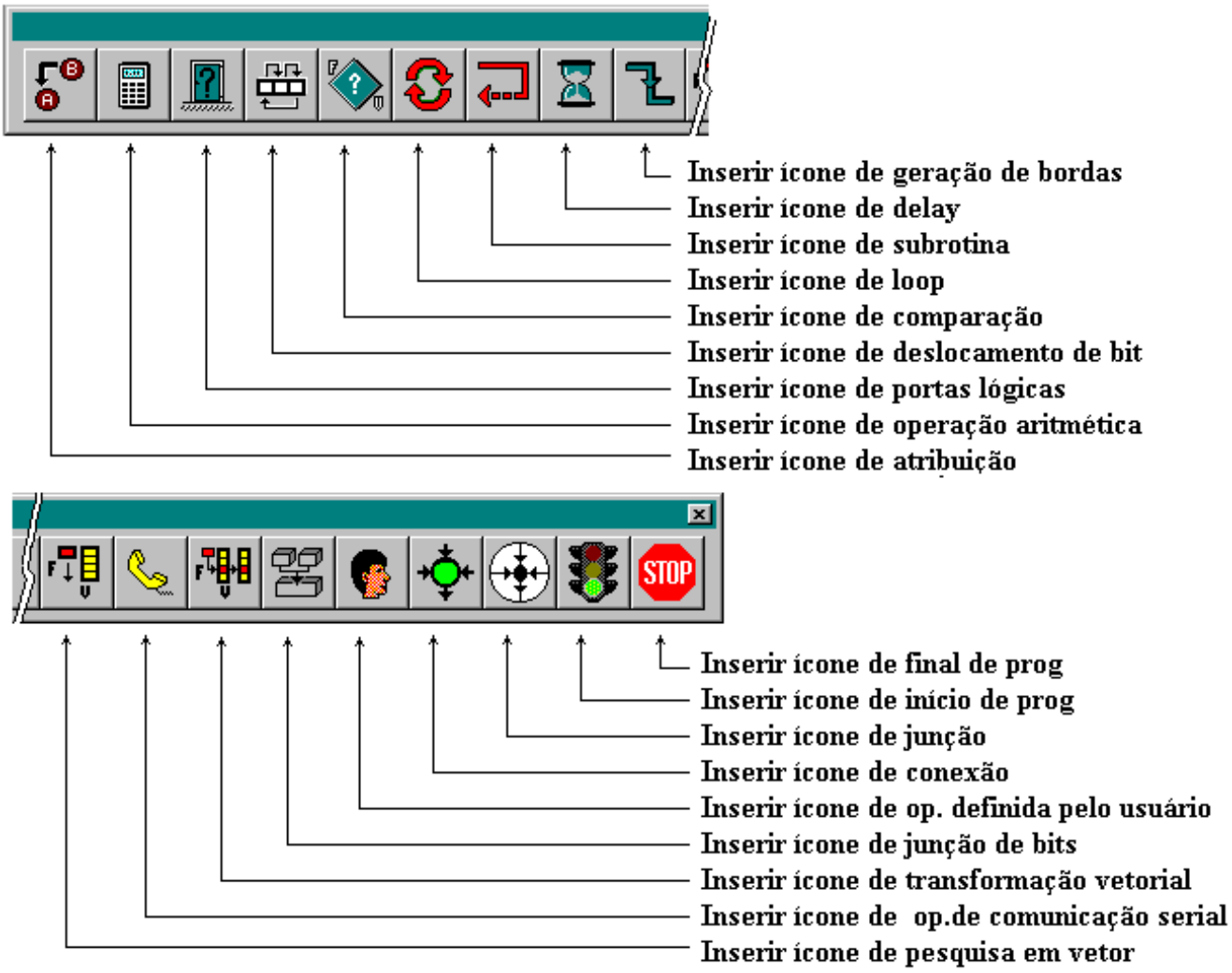


Fig.3.12: Barra de edição de ícones.

c) Barra de status: esta barra situa-se no rodapé da área de trabalho e mostra informações sobre a função que está sendo ativada no momento. A barra de status é alterada quando no modo de simulação e passa a mostrar um percentual de quanto da simulação já foi executada.

Fig.3.13: Barra de status.

Todas as janelas principais do Modi são identificadas no canto superior esquerdo de seus "frames" por um pequeno ícone que facilita a distinção da função das janelas conforme a figura 3.14, abaixo:

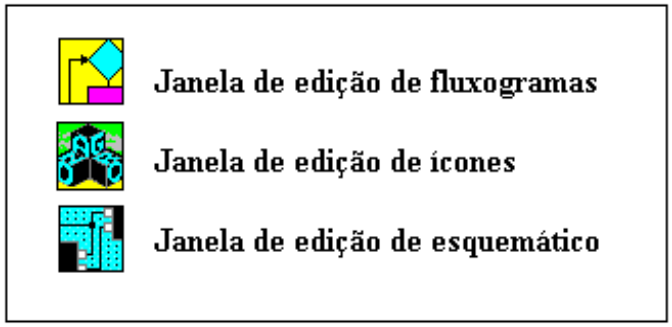


Fig.3.14: Ícones de identificação de janela.

Uma visão completa do ambiente Modi, com suas principais janelas ativadas, pode ser obtida da figura 3.15 a seguir :

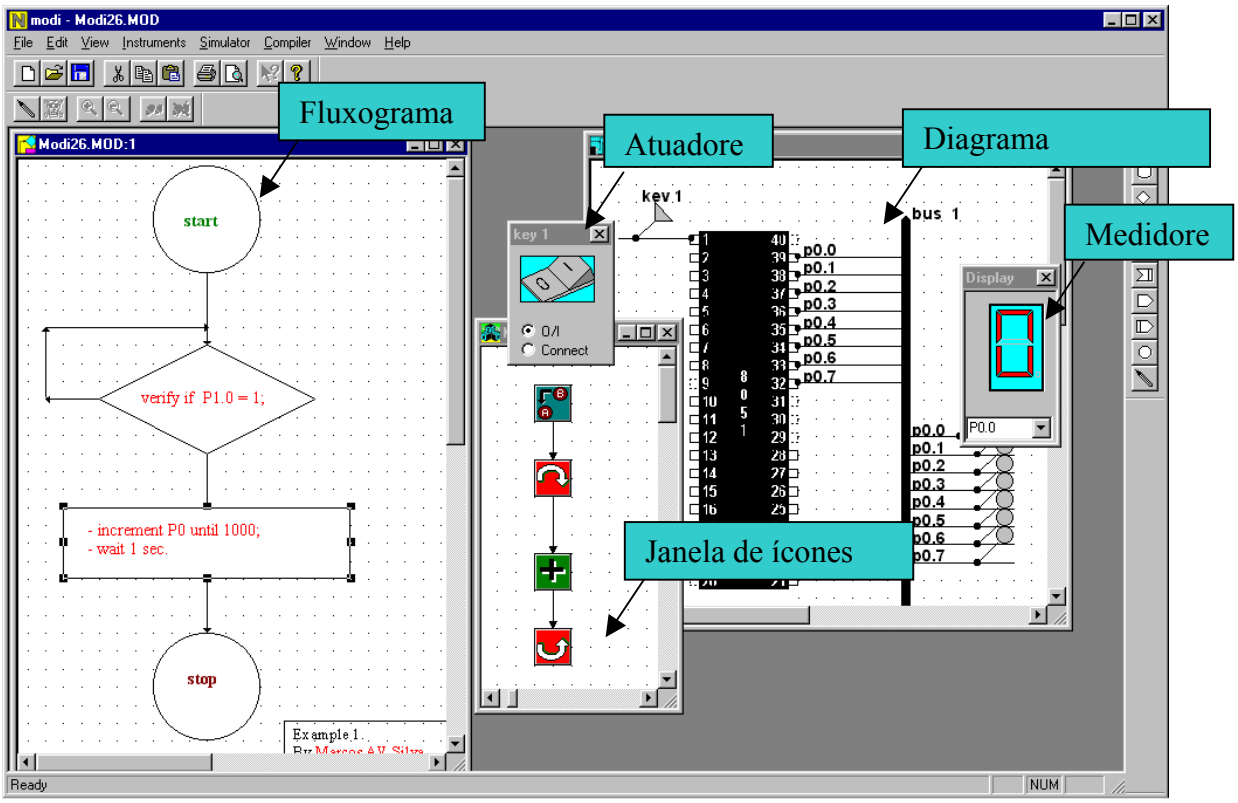


Fig.3.15: Principais janelas do Modi.

3.3 Iniciando um projeto

Neste ponto passamos a descrever quais os procedimentos para realizarmos o projeto de uma aplicação simples, o controle do sentido de rotação de um motor de passo, dentro do ambiente Modi. Esta rotina de controle será dividida em dois blocos: o primeiro fará as inicializações necessárias e o segundo será a rotina de controle propriamente dita. O motor de passo estará conectado a porta 3 do microcontrolador e o interruptor de controle estará conectado ao bit 0 da porta 0. O algoritmo de funcionamento desta rotina é bem simples e pode ser descrito da seguinte forma:

- Bloco 1: - Zerar porta de controle P0;
  - Inicializar porta P3 com valor 11H.

- Bloco 2: - Testar bit de controle P0.0;
  - Se P0.0=0, rotacionar porta P3 para a direita;
  - Se P0.0=1, rotacionar porta P3 para a esquerda;
  - Aguardar 0,5 segundo;
  - Voltar ao início do bloco.

Para iniciar um projeto no Modi, após iniciar o programa, o usuário tem uma janela de edição de fluxograma onde é feita a divisão do projeto em blocos operacionais. A simbologia adotada segue o padrão recomendado pela norma CCITT[9] e é mostrada na figura 3.16.

Fig.3.16: Simbologia utilizada.

O usuário seleciona o símbolo desejado, pressionando o botão esquerdo do mouse sobre o elemento na barra de ferramentas situada no lado direito da área de trabalho. Depois é só mover o mouse até a posição conveniente na janela de edição de fluxograma e pressionar novamente o botão esquerdo do mouse para fixar o símbolo na tela.

Esta janela de edição oferece o recurso de grade (grid), que auxilia na disposição (alinhamento) dos elementos.

Para mover os elementos dentro da janela, basta pressionar e manter pressionado o botão esquerdo do mouse enquanto movemos o mesmo.

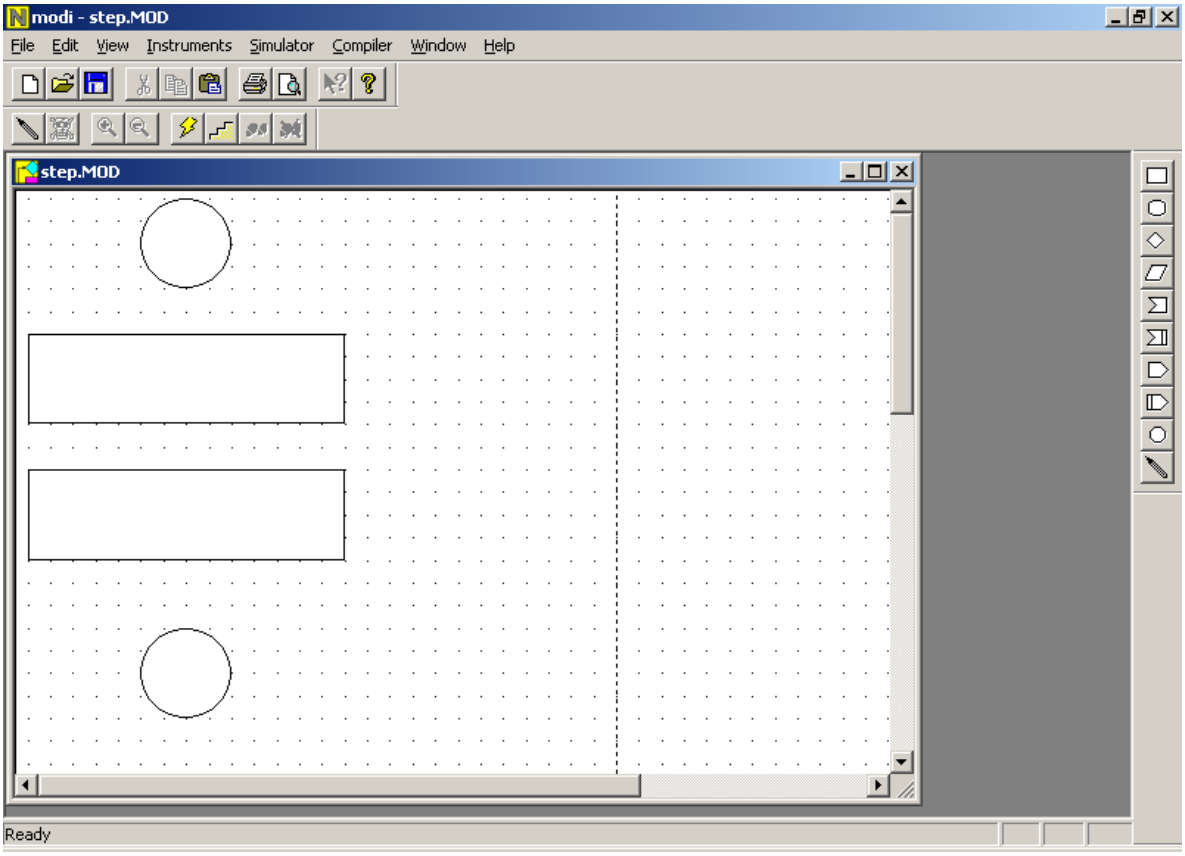


Fig.3.17: Posicionando os elementos do fluxograma.

Após o posicionamento dos elementos, pode-se conectar os mesmos através da seleção do item  da barra de ferramentas. Se for necessária a remoção de algum componente do fluxograma, basta selecionar o item na janela e apertar a tecla Del. no teclado.

A inserção de texto aos blocos do fluxograma é feita através do item Insert New Object do menu Edit. Podemos inserir outros objetos além de texto com esse item. Figuras, sons e outros aplicativos em OLE podem ser adicionados, complementando a documentação do projeto.

O resultado pode ser algo como o que é visto na figura 3.17.

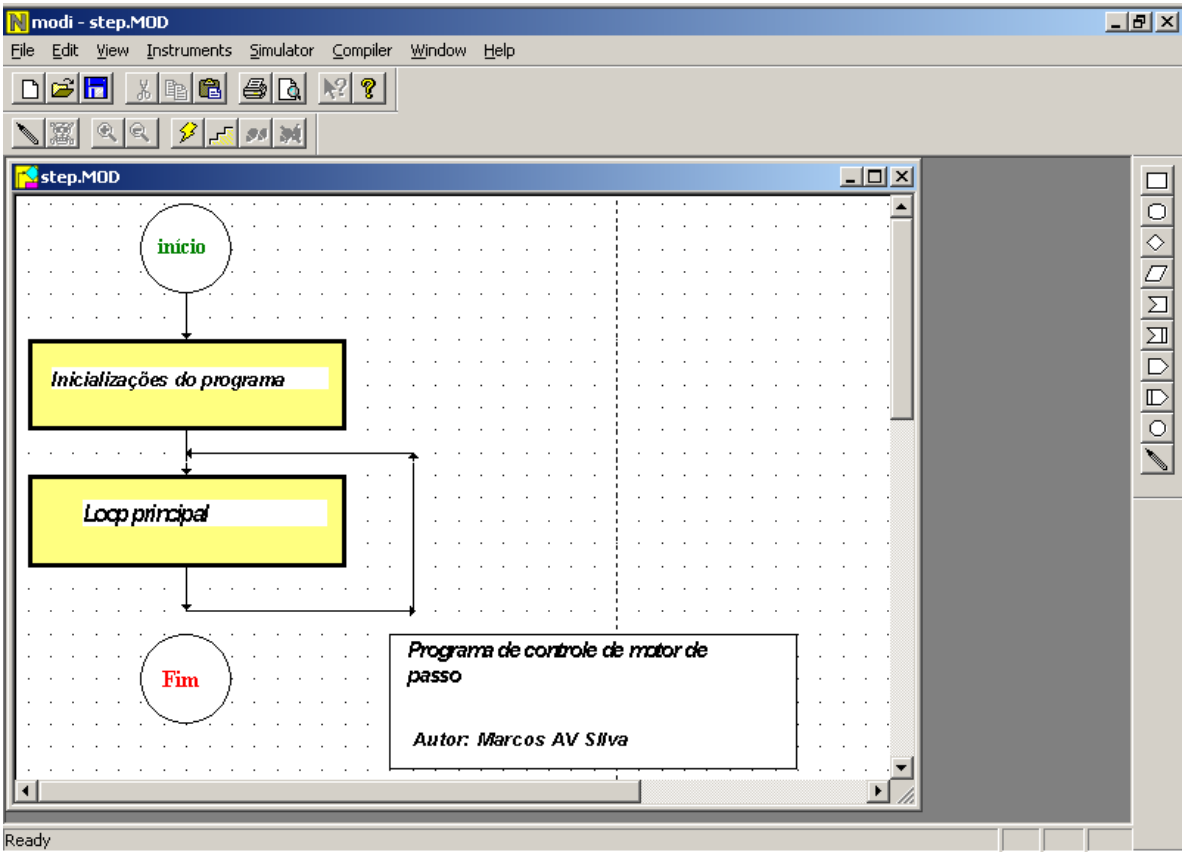


Fig.3.18: Edição de fluxogramas.

Se o fluxograma criado for muito longo e o usuário queira ter uma vista completa do que foi gerado, pode-se aumentar ou diminuir as figuras com o uso dos itens Zoom-In e Zoom-Out, representados na barra de ferramentas pelos botões : 

3.4 Adicionando ícones na janela de edição de ícones

Após a divisão do projeto em blocos funcionais, o passo seguinte é definir as operações que cada bloco irá realizar. Por exemplo, se pressionarmos duas vezes o botão do lado direito do mouse sobre o primeiro retângulo de tarefa (logo abaixo do círculo inicial), uma janela de edição de ícones irá aparecer juntamente com sua barra de ícones associada, podemos, então, entrar com os ícones que representam operações específicas do microcontrolador. Uma descrição completa dos ícones de operação pode ser vista no anexo I deste volume.

Os ícones são selecionados pressionando-se o botão esquerdo do mouse sobre o ícone desejado na barra de seleção. Depois posiciona-se o cursor na posição onde se deseja fazer a inserção na janela de edição de ícones e, ao se pressionar o botão esquerdo do mouse, irá aparecer o ícone selecionado. Se quisermos adicionar mais de um ícone do mesmo tipo, basta pressionar o botão esquerdo novamente na posição desejada, que outra inserção será efetuada. Se, ao contrário, quisermos desselecionar o ícone selecionado para escolhermos outro ícone, basta pressionar o botão direito do mouse na janela de edição de ícones, que será desfeita a seleção do ícone. Para mover o ícone, pressione e mantenha pressionado o botão esquerdo, enquanto movimenta o mouse. Se desejarmos apagar um ícone na janela de edição de ícones devemos clicar sobre o botão  na barra de ferramentas.

Uma vez colocado um ícone de operação na posição desejada na janela de edição de ícones, se pressionarmos o botão direito do mouse sobre o mesmo, a cada click será selecionado um modo de operação para o ícone em questão. Para termos acesso aos

parâmetros relativos a operação selecionada, basta pressionar duas vezes o botão esquerdo do mouse sobre o ícone desejado, que uma caixa de diálogo será aberta e o usuário terá como alterar esses parâmetros. A figura 3.19 mostra um exemplo de caixa de diálogo para o ícone de atribuição.

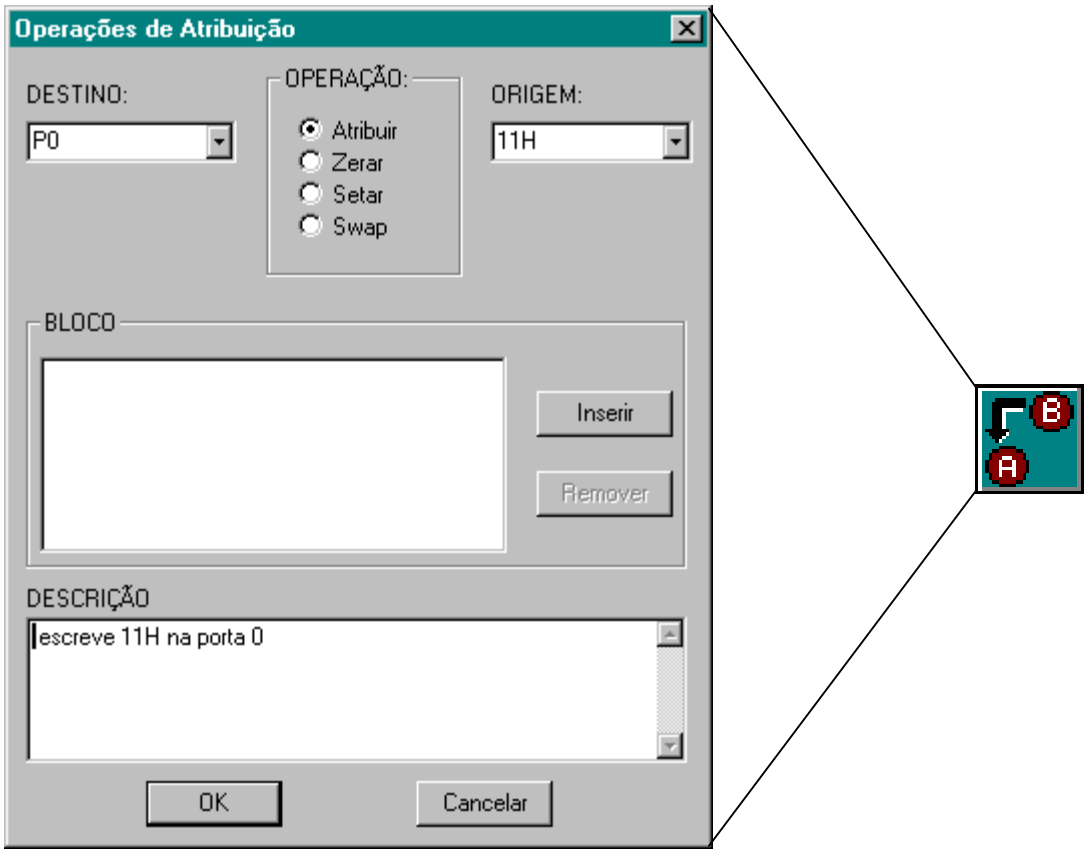


Fig.3.19: Operação de atribuição.

Para o exemplo da figura acima, estamos atribuindo a variável P0 (porta 0 do microcontrolador) o valor hexadecimal 11, com o comentário sobre o funcionamento da operação apresentado no item descrição. No item Bloco, poderíamos inserir várias

operações semelhantes a esta, ou seja, poderíamos inicializar várias variáveis através de um único ícone de atribuição, o que simplifica bastante o desenho final do código.

Os demais detalhes sobre os parâmetros referentes aos ícones de operação podem ser obtidos consultando a referência [4] deste volume.

Para o nosso projeto em questão fizemos a inserção dos ícones referentes ao bloco 1 do fluxograma, onde são feitas as inicializações, e inserimos os ícones do bloco 2, onde se realiza o controle em si do motor de passo.

Neste ponto, os resultados obtidos podem ser semelhantes aos da figura 3.20:

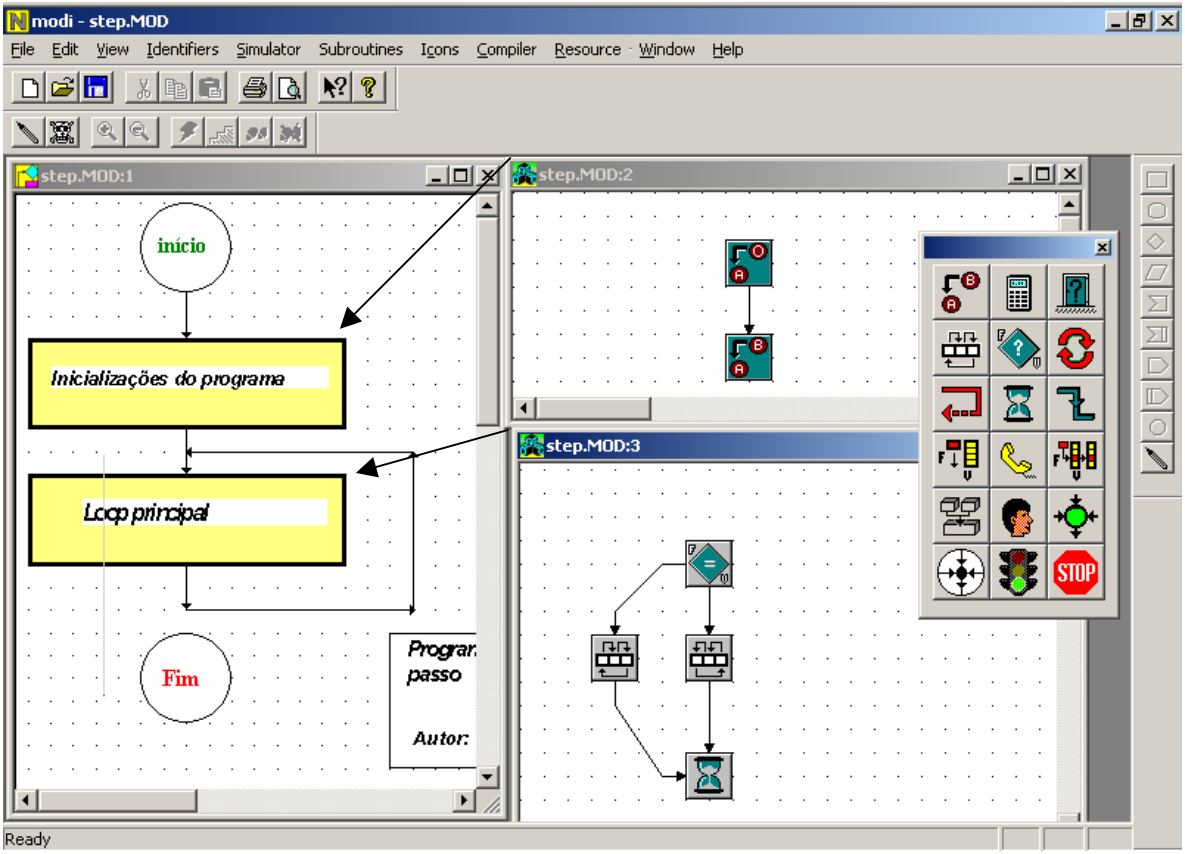


Fig.3.20: Edição de ícones.

3.5 Inserindo componentes na janela de esquemático

Após a descrição do funcionamento do projeto, podemos agora, especificar alguns elementos externos que estão ligados ao microcontrolador. Estes elementos podem ser do tipo latch, portas lógicas, flip-flops, memórias, etc; que irão apresentar saídas em função dos sinais extraídos das portas e de alguns pinos de controle do microcontrolador.

Para abrirmos a janela de edição de esquemático, procura-se pelo item View do menu principal e, dentro deste, o sub-item Schematics View.

A seleção dos componentes para a janela de edição do esquemático é feita da mesma forma que para a janela de edição de fluxograma, utilizando os elementos que se localizam na barra de ferramentas do lado direito da área de trabalho.

Uma vez selecionado o componente, pode-se fixá-lo na janela de edição de esquemático pressionando-se o botão esquerdo do mouse. Para mover o componente, basta manter pressionado botão esquerdo enquanto movimenta-se o mouse. A figura 3.21 ilustra o que dissemos.

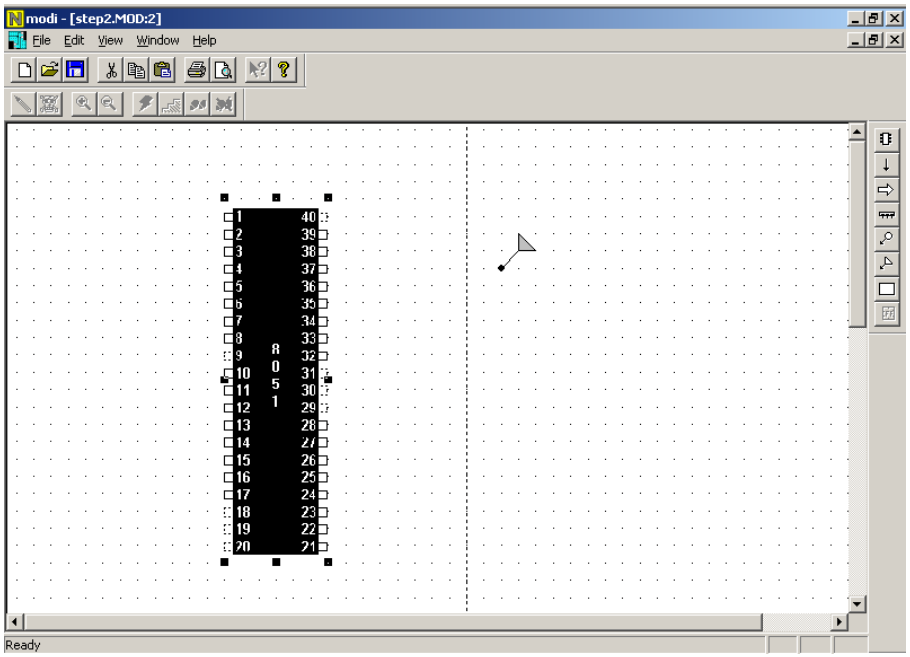


Fig.3.21: Edição de esquemático.

Se pressionarmos o botão esquerdo do mouse duas vezes sobre um componente, será aberto um diálogo com as propriedades do componente, como no exemplo abaixo:

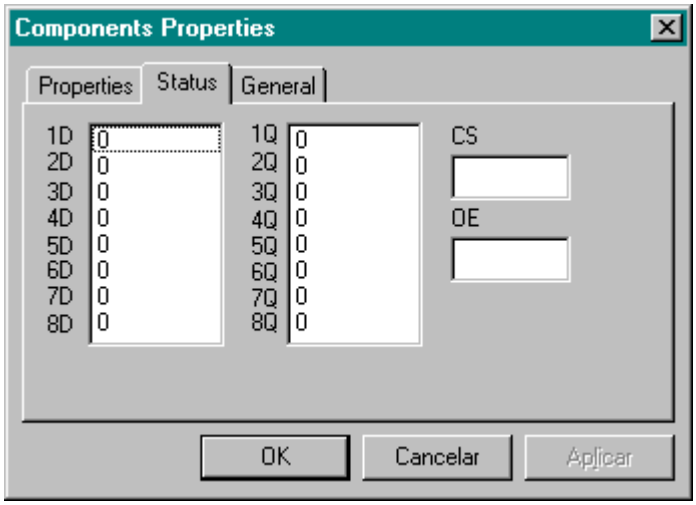


Fig.3.22: Propriedades do componente.

A figura 3.22 mostra as propriedades do elemento *latch*, destacando o estado dos pinos de entrada e de saída e dos pinos de controle.

Para efetuarmos a ligação entre os componentes, devemos selecionar o componente *wire* (fio). A inserção deste componente é feita de modo diferenciado dos demais, pois uma vez selecionado o componente na barra de ferramentas, posiciona-se o mouse na posição de inserção e pressiona-se o botão esquerdo do mouse. Mantendo-se pressionado este botão e movimentando-se o mouse, o elemento irá se expandir até o ponto onde deseja-se finalizar a conexão, quando o botão do mouse deve ser liberado.

Se a conexão for feita a contento, aparecerá um circulo preto nas terminações do *wire* . Para mover o *wire* na tela, basta pressionar e manter pressionado o botão esquerdo do mouse, enquanto movimenta-se o mesmo. Se após mover o *wire,* alguma extremidade deste não toca em nenhum outro elemento, o círculo preto na extremidade não conectada irá desaparecer.

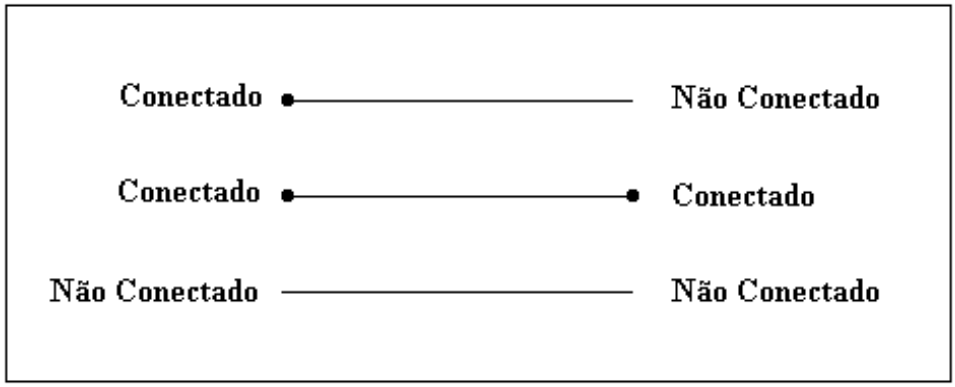


Fig.3.23: Tipos de conexões.

Para conferir os dados referente a conexão feita pelo *wire* abra o diálogo de propriedades, pressionando duas vezes o botão esquerdo do mouse sobre o fio desejado. O Resultado será algo semelhante a figura 3.24.

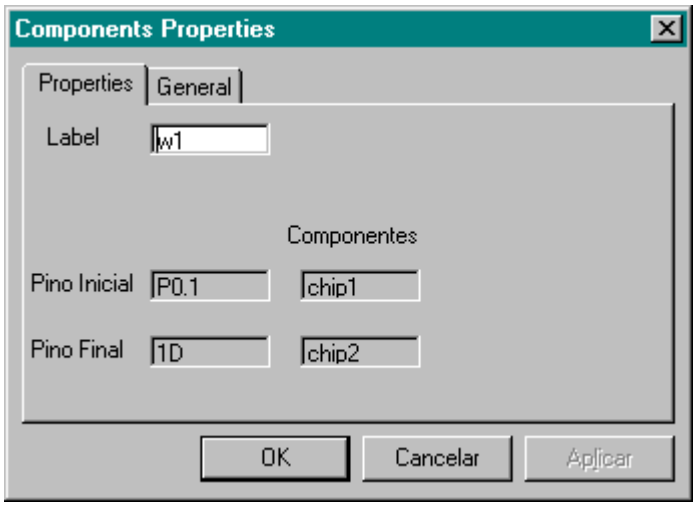


Fig.3.24: Propriedades do wire.

Se for desejada uma cópia de algum elemento, basta selecionar o elemento pressionando o botão da direita do mouse sobre o elemento na vista de edição de esquema e selecionar o item “copy” do menu instantâneo. Se quisermos remover o componente, é só seguirmos os passos anteriores e selecionar o item delete do menu, ou selecionar o componente com o botão esquerdo do mouse e pressionar a tecla Delete do teclado.

Para selecionarmos mais de um elemento na vista, pressiona-se e mantém-se pressionada a tecla Shift do teclado enquanto seleciona-se os elementos com o botão esquerdo do mouse. Usa-se esse procedimento quando se deseja mover um bloco de componentes na janela de edição de esquema.

Tendo sido feitas todas as conexões necessárias, teremos algo semelhante ao apresentado na figura 3.25.

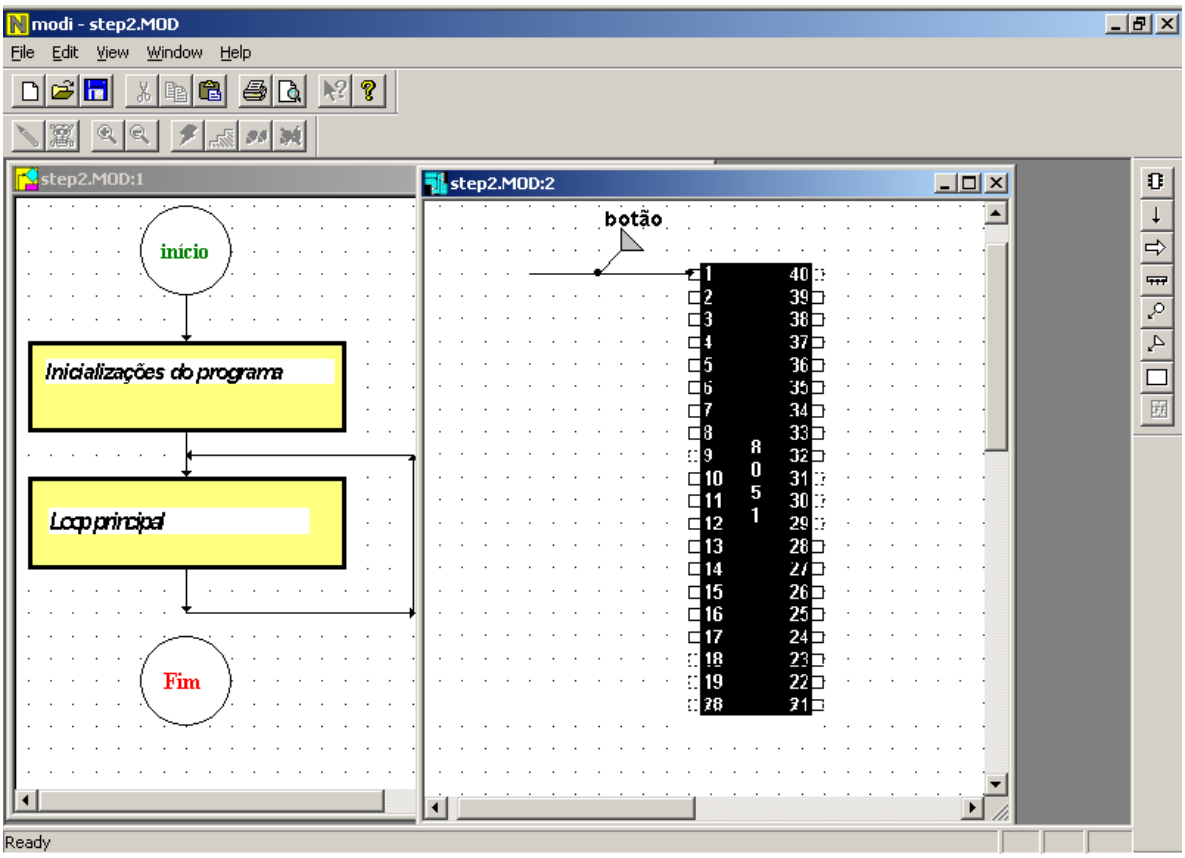


Fig.3.25: Edição de esquemático.

3.6 Simulando a aplicação

Após ter concluído o projeto da aplicação, a fase seguinte na seqüência de desenvolvimento do sistema é a simulação do protótipo. No Modi a simulação é feita fazendo-se uma varredura na lista de ícones de cada bloco do fluxograma, onde cada ícone é responsável pela sua própria simulação. A simulação pode ser acompanhada através da indicação, na janela de fluxograma, do bloco em simulação e, dentro de cada bloco, na janela de edição de ícones, do ícone simulado no momento. Esta indicação é feita através da mudança na cor do elemento simulado na janela de fluxograma, e na inversão das cores do ícone simulado na janela de edição de ícones.

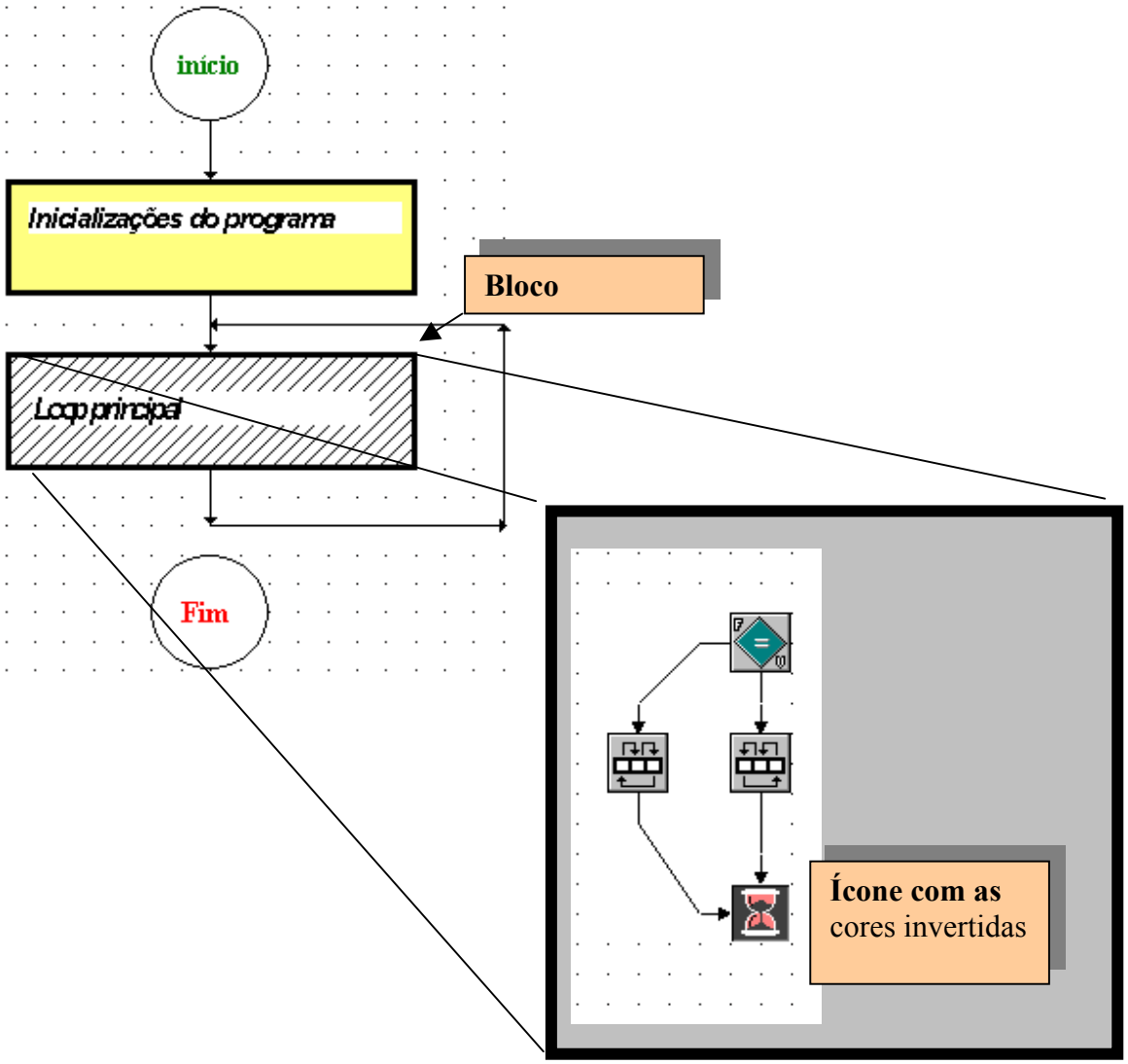


Fig.3.26: Indicação de simulação.

O Simulador tem três modos de operação:

1) Automática: A simulação do sistema corre independente do usuário, sendo os blocos e ícones executados na seqüência em que estão dispostos nas suas respectivas janelas.

2) Passo-a-Passo / fluxograma: Neste modo de operação, o Modi simula o fluxograma bloco-a-bloco, porém a simulação só avança para o bloco seguinte se o usuário apertar o botão  na barra de ferramentas.

3) Passo-a-Passo / ícones: Funciona basicamente como no caso anterior, mas esta opção somente está ativada quando estamos trabalhando na janela de edição de ícones. Se tivermos várias janelas de ícones abertas ao mesmo tempo o Modi irá simular a que estiver ativada no momento. Novamente o simulador só avança para o próximo ícone se for pressionado o botão  na barra de ferramentas

A simulação pode ser interrompida a qualquer momento através do botão  na barra de ferramentas.

3.7 Medidores e Atuadores

Para acompanhar o desenrolar da simulação das variáveis do sistema, o Modi possui um conjunto de medidores (ou observadores) que proporcionam uma visualização do estado destas variáveis. Estes medidores dividem-se em: Voltímetro (VU meter), Termômetro, Osciloscópio, Led, Display e Motor de Passo, e se apresentam como janelas de diálogos semelhantes as da figura 3.27.

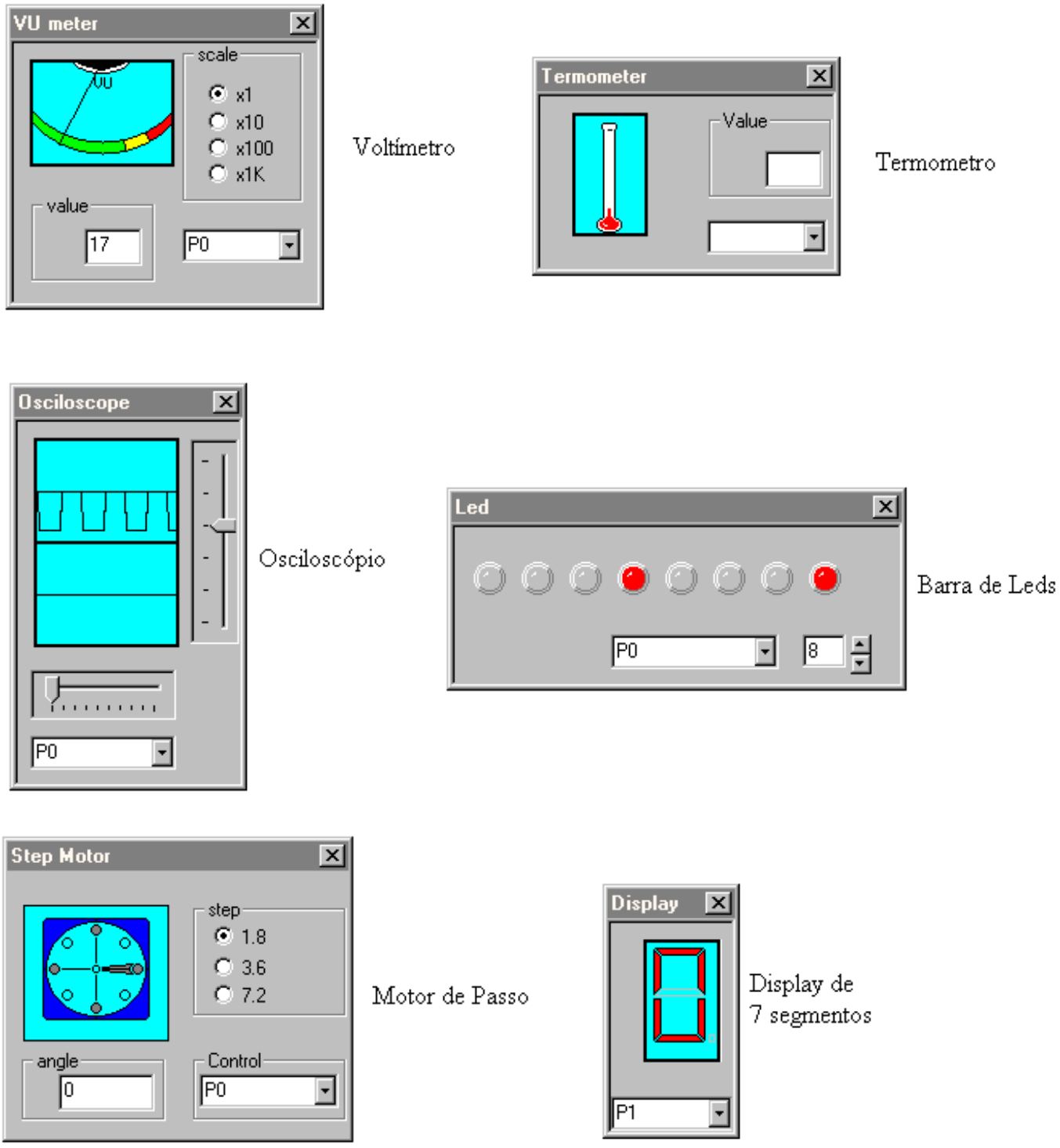


Fig.3.27: Tipos de medidores.

A operação destes medidores é bastante intuitiva, sendo a escolha da variável a ser observada feita pelo item de seleção comum a todos os diálogos.

Além dos medidores, existem os atuadores que podem ser chamados a partir do menu principal e que realizam funções básica tais como: teclado numérico (key pad), botões tipo "push-button", e interruptores (switches). Estes atuadores podem ser vistos na figura 3.28.

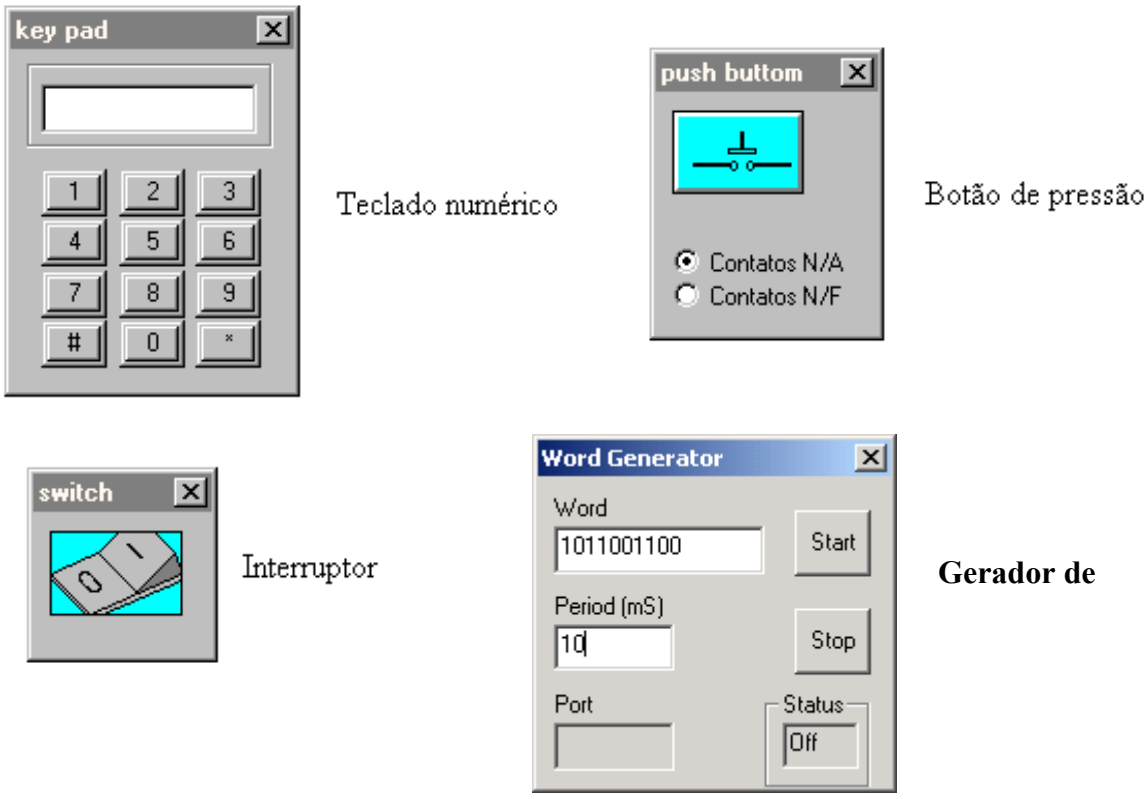


Fig.3.28: Tipos de atuadores.

Existe uma outra maneira de se acessar os diálogos de medidores a atuadores além do menu principal, que é através da janela de edição de esquemático através dos símbolos:

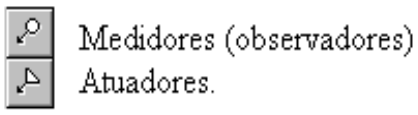


Fig.3.29: Medidores e atuadores.

Após selecionar um desses itens e posicioná-lo na janela de edição de esquemático, se for pressionado o botão direito do mouse sobre o elemento, um menu será aberto com as opções de medidores ou atuadores.

No nosso projeto em questão, podemos agora adicionar os medidores e atuadores apropriados para fazermos a simulação do mesmo.

Como estamos controlando um motor de passo, devemos selecionar o motor de passo dentre os medidores existentes e em seguida devemos selecionar a porta 3 do microcontrolador como porta de controle do motor conforme a figura 3.30 abaixo:

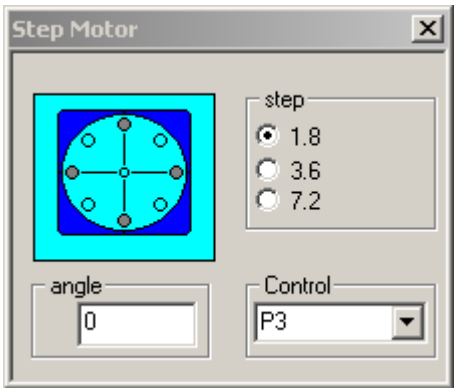


Fig.3.30: Motor de passo.

O Atuador escolhido para controlar o sentido de rotação do motor de passo foi o interruptor, pois ele apresenta dois estados estáveis (ligado ou desligado). Ele deve estar conectado ao pino P1.0 do microcontrolador conforme desenhado no diagrama esquemático da figura 3.25.

Após a seleção dos medidores e atuadores, já estamos prontos para realizar a simulação do projeto. A figura 3.31 mostra o resultado desta simulação.

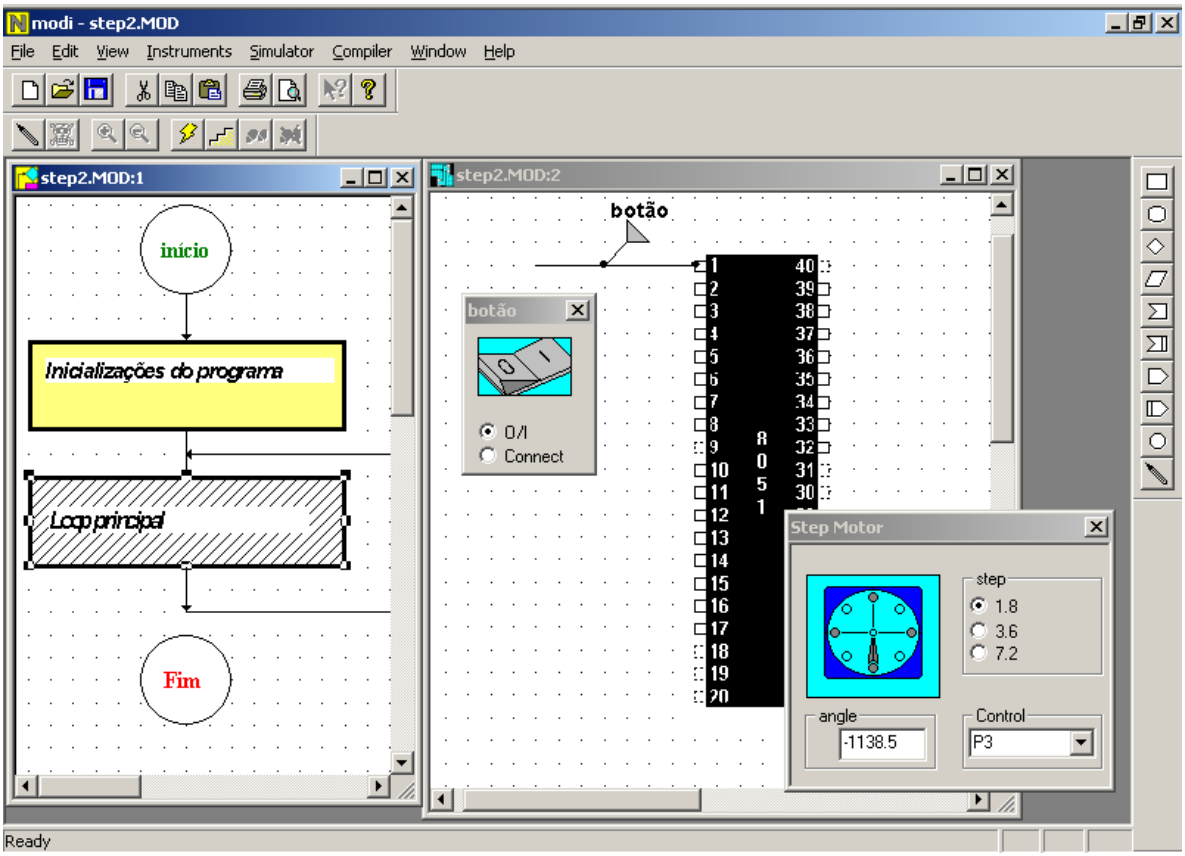


Fig.3.31: Resultado da simulação.

Nesta simulação vemos um interruptor selecionando o sentido de rotação de um motor de passo conectado a porta P3 do microcontrolador. Vemos também que o segundo bloco do fluxograma está hachurado, indicando que ele está sendo simulado. Como é um loop infinito, então não existe próximo bloco.

3.8 Gerando o código

Tendo sido feitas todas as simulações necessárias e, comprovando-se o funcionamento correto do sistema, o próximo passo no desenvolvimento da aplicação é a geração do código em *Assembly* do microcontrolador. Esta etapa é feita pelo compilador de ícones[4] acionado pelo item de menu "compiler" e seu sub-item "start".

A compilação funciona de maneira similar a simulação, ou seja, quando iniciada ela promove uma varredura na lista ícones de cada bloco do fluxograma onde cada ícone é responsável pela geração do seu código correspondente como pode ser visto na figura 3.32.

Fig.3.32: Compilação do ícone de atribuição.

Neste exemplo o ícone que representa uma atribuição de zero a uma variável, no caso a porta P1, gera o código em *Assembly* correspondente.

O resultado da compilação pode ser visto através de um diálogo semelhante ao da figura 3.33.

O código gerado apresenta-se livre de erros de sintaxe, bem documentado (através das linhas de comentários) e dentro de uma "template" padrão gerada automaticamente. Com isso estamos prontos para posteriormente gerarmos o código em hexadecimal e transferi-lo para o microcontrolador através de um hardware apropriado.

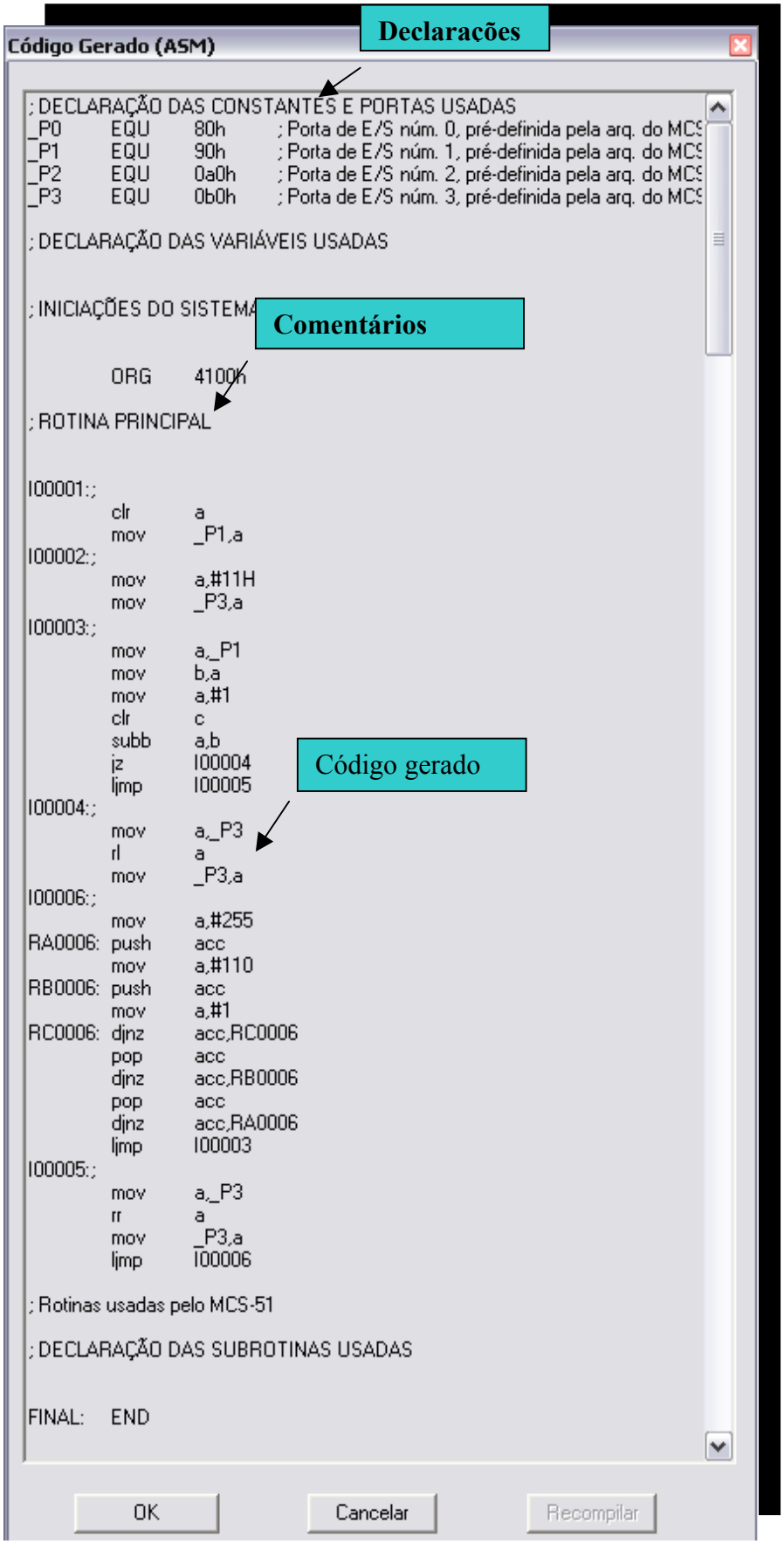


Fig.3.28: Resultado da compilação.

Capítulo IV: Resultados e Conclusões.

Dentro do que foi proposto no início deste trabalho, atingimos resultados que satisfizeram a maioria dos item mencionados na proposta de trabalho. Ou seja, construímos uma ferramenta de desenvolvimento visual para microcontroladores com características próprias e inéditas até o momento, apesar de termos no mercado algumas ferramentas também visuais, ainda sim o Modi traz soluções únicas na programação de sistemas dedicados.

Conceituamos os problemas próprios da elaboração de ferramentas e apontamos soluções para os mesmos, deixando uma "receita de bolo" aos que se propõem a implementar ferramentas desta natureza. Por fim, deixamos um tutorial de utilização da ferramenta Modi.

Quanto a sua utilização, o ambiente Modi se mostrou bastante simplificado até mesmo para pessoas não muito acostumadas a lidar com programas baseados em linguagem visual, uma vez que são utilizados recursos já bastante conhecidos por programadores de microprocessadores, tais como: elementos de fluxograma, esquemáticos de componentes eletrônicos e ícones intuitivos.

Podemos então afirmar que essa ferramenta é indicada tanto para programadores iniciantes, ainda não familiarizados com a linguagem *Assembly* do microcontrolador, como para programadores experientes que desejam migrar para o uso de ferramentas de mais alto nível.

Falando em termos de documentação de software, o Modi pode ser considerada uma ferramenta bastante versátil e atual, pois torna a representação de um software praticamente auto-explicativa, uma vez que faz uso de elementos OLE (Object Linked Embedded) do Windows que podem ser adicionados quando da edição do fluxograma do sistema em desenvolvimento, com isso o usuário pode fazer uso de editores de texto, sons pré-

gravados, mostradores de slides, etc, para complementar a documentação do software. Além disso, o código gerado também pode ser totalmente comentado quando da geração do mesmo.

A indicação visual dos símbolos em tempo de execução de uma simulação de código, como visto na figura 4.1, também facilita em muito o acompanhamento do que está sendo executado naquele momento. Tanto a nível de blocos de fluxograma como a nível de diagramas de ícones.

O tempo gasto na simulação dos códigos no modo automático é influenciado diretamente pelo tipo de computador em uso, contudo mesmo em máquinas mais lentas a simulação acontece numa velocidade considerada compatível. Para uma aplicação com 10 ícones de operação o tempo gasto na simulação foi de 40ms. O que dá uma média de 4 ms por ícone simulado. Estes tempos foram tomados com a janela de edição de ícones aberta e os ícones sendo apontados pelo simulador o que demanda tempo de processamento do computador. Se fecharmos a janela de edição de ícones, acompanhando a simulação apenas pela janela de edição de fluxogramas, o tempo gasto para esta mesma aplicação se reduz para 10 ms. Dando uma média de 1 ms por ícone simulado. Nos testes realizados foi utilizado um computador Pentium 233 MHz, que nos tempos atuais pode ser considerado uma máquina lenta, mesmo assim o tempo de execução foi considerado adequado.

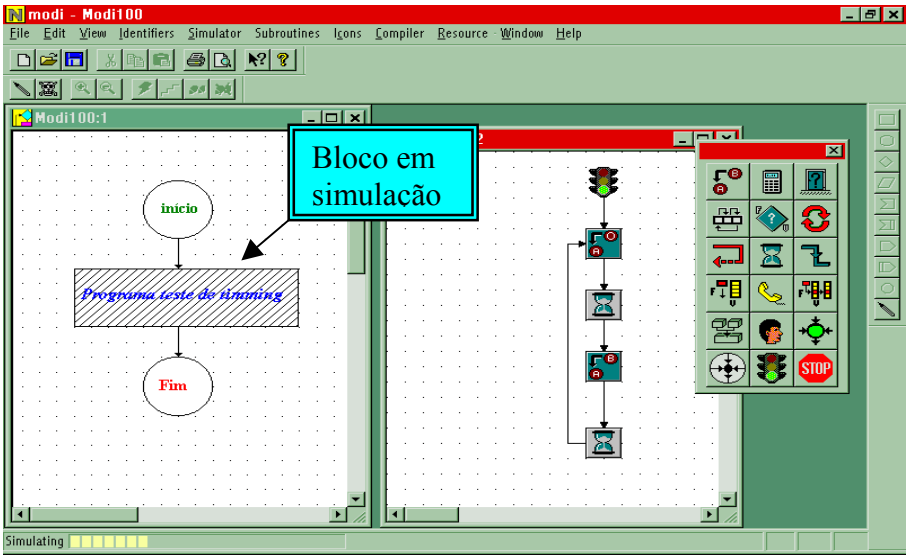


Fig.4.1: Simulação de código.

O uso de instrumentos virtuais proporcionou um alto nível na interação com as variáveis da aplicação em desenvolvimento, permitindo visualizar de maneira mais clara o que está se passando com elas em tempo de simulação.

Quanto ao uso de atuadores juntamente com a edição de diagramas esquemáticos, isto mostrou-se bastante prático pois permite ao desenvolvedor projetar e documentar tanto o hardware quanto o software. Acompanhando o funcionamento de ambos em tempo de simulação e atuando diretamente no sistema como se estivesse numa bancada virtual.

Por fim, esta versão 1.0 do Modi dá uma idéia de como podemos ter uma ferramenta 100% visual, com pouquíssimas entradas textuais, que utiliza o poderio computacional dos atuais computadores pessoais e dismistifica os conceitos que uma ferramenta gráfica não possa gerar código em tamanho reduzido, conforme foi demonstrado, colocando de vez a programação de sistemas dedicados baseados em microcontroladores no grupo das linguagens de programação de quarta geração.

4.1 Conclusões

Esta tese apresentou uma proposta de ferramenta de desenvolvimento que utiliza os conceitos de programação visual, voltada à aplicações de sistemas dedicados baseados em microcontroladores.

Um dos objetivos desta tese foi tornar a programação de microcontroladores mais independente da digitação de linhas de código como em linguagens textuais tradicionais como Assembly ou C, tornando o mouse uma interface de entrada de informação mais usual. O código gerado automaticamente pela ferramenta é livre de erros de sintaxe, totalmente comentado e o seu tamanho, quando comparado com outros compiladores, não é tão extenso, como foi demonstrado no decorrer deste trabalho.

Um outro objetivo alcançado foi o de ter uma ferramenta que facilitasse a documentação do software em desenvolvimento, pois a adição de recursos de outros programas, como por exemplo editores de texto, editores gráficos e gravadores de som, pode ser feita dentro do ambiente Modi, facilitando o intercâmbio de informações entre grupos de trabalho.

O uso didático dessa ferramenta no ensino de disciplinas voltadas ao desenvolvimento de aplicações microcontroladas também pode ser cogitado, não excluindo o uso pleno dessa ferramenta no desenvolvimento de aplicações a serem usadas pelos consumidores de projetos eletrônicos dedicados.

Por fim, o aparecimento de programas tais como Renoir (Mentor Graphics) e Realizer (Actum Soluction), posteriores ao início da idealização do Modi[12], reforça ainda mais a idéia de se utilizar linguagens visuais na conceituação de ferramentas de desenvolvimento, e, em especial, voltadas a programação de microcontroladores.

4.2 Sugestões ao Modi

Como melhorias à atual versão do Modi, afim de que tenhamos verdadeiramente um produto comercial, podemos sugerir os seguintes itens:

- ✓ Extensão dos recursos existentes a outros microcontroladores de outras famílias como, por exemplo, ao modelo PIC da Microchip. A adição de novos microcontroladores ao ambiente Modi é possível pelo uso da programação orientada a objetos que aumenta a modularização do programa, com isso novos objetos que representem o hardware e o conjunto de instruções do novo microcontrolador podem ser concebidos;

- ✓ Ampliar o nicho de aplicações do Modi para processadores do tipo DSP. Com isso, poderíamos expandir a filosofia de programação utilizada no Modi para outros sistemas dedicados similares aos microcontroladores;

- ✓ Utilização dos instrumentos de medição criados pelo LabWindows, da National Instruments, que substituiriam os atuais medidores. Isso pode ser feito gerando-se os instrumentos virtuais na ferramenta LabWindows e as exportando sob a forma de DLLs que iriam interagir com o programa Modi, resultando numa melhor apresentação e funcionalidade dos atuais instrumentos virtuais;

- ✓ Adoção de uma referência de tempo interna que possibilitasse a simulação de sistemas em tempo-real e interrupções. Estabelecendo-se um padrão de tempo interno, poderíamos obter uma simulação mais realística dos elementos de hardware componentes dos projetos, aumentando, assim, a qualidade da simulação como um todo;

- ✓ Geração de código referente a interrupções. Com isso, poderíamos projetar sistemas mais complexos e que utilizem melhor a capacidade do hardware do microcontrolador;

- ✓ Criar uma versão multiplataforma que pudesse rodar com outros sistemas operacionais, como por exemplo, o Unix, Linux, etc, isso pode ser feito utilizando-se linguagens de programação tais como JAVA, que geram aplicativos que podem ser rodados em computadores pessoais e estações de trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS:

[1] JamaL, Rahman; Wenzel, Lothar: " The Applicability of the Visual Programming Language LabVIEW to Large Real-World Applications" - Proceedings 11th International IEEE Symposium on Visual Languages - Darmstadt, Germany – September/ 1995.

[2] Brunett, Margaret M., Baker, Maria J.; "A Classification System for Visual Programming Languages" – Department of Computer Science- Oregon State University; Technical Report – 1994.

[3] da Silva, Marcos Antonio Vieira; de Sousa, Antonio Heronaldo, Ferreira, Elnatan Chagas; *MODI – A proposal of a visual tool to simulate and synthesize software applied to embedded systems;* VII Workshop IBERCHIP; 20/03/2001 a 23/03/2001 Montevideo – Uruguay.

[4] de Sousa, Antonio Heronaldo; "Onagro Um Ambiente Gráfico para Desenvolvimento de Software para Microcontroladores" – tese de mestrado – UNICAMP/1995. (http://www.demic.fee.unicamp.br//onagro/)

[5] Valaer, Laura A.; Babb II, robert G.; "Choosing a User Interface Development Tool" - IEEE SOFTWARE Vol. 14, No. 4: JULY/AUGUST 1997, pp. 29-39.

[6] Nickerson, Jeffrey V. "Visual Programming" - Ph.D. Dissertation. New York University- 1994.

[7] Freeman, Elisabeth; Gelernter, David; Jagannathan, Suresh : "In Search of a Simple Visual Vocabulary" – Proceedings 11th International IEEE Symposium on Visual Languages - Darmstadt, Germany – September/ 1995.

[8] Myers, Brad A.;"*Why are Human-Computer Interfaces Difficult to Design and Implement?*" Carnegie Mellon University School of Computer Science Technical Report, no. CMU-CS-93-183 July 1993.

[9] "Functional Specification and Description Language (SDL)"; CCITT (The International Telegraph and Telephone Consultative Committee) – Volume VI – fascicle VI.7. Recomendations Z.100-Z.104 (http://www.itu.ch/itudoc/itu-t/rec.html).

[10] Sears, Andrew; Lund, Arnold M.; “Creating Effective User Interfaces” - IEEE SOFTWARE Vol. 14, No. 4: JULY/AUGUST 1997, pp. 21-24.

[11] Koike, Yuichi; Maeda, Yasuyuki; Koseki, Yoshiyuki; “Improving Readability of Iconic Programs with Multiple View Object Representation” - Proceedings 11th International IEEE Symposium on Visual Languages - Darmstadt, Germany – September/ 1995.

[12] da Silva, Marcos Antonio Vieira; de Sousa, Antonio Heronaldo, Ferreira, Elnatan Chagas; *MODI – A Visual Environment to Develop Systems Based on Microcontrollers;* XIII SBMicro International Conference on Microelectronics and Packaging ICMP’98; 12/08/1998 a 14/08/1998 Curitiba/PR – Brazil.

[13] de Sousa, Antonio Heronaldo; da Silva, Marcos Antonio V.; Ferreira, Elnatan Chagas; *ONAGRO - A Graphical Environment to the Development of Microcontrollers Software*; II Congresso Mundial de Automação; 7/05/1996 a 30/05/1996 Montpellier - FRANÇA.

BIBLIOGRAFIA:

- Calvert, Charles ; "Programando Aplicações em Windows com C e C++"; Berkeley, 1994.

- Barkakati, Nabajyoti; "Visual C++ Guia de Desenvolvimento Avançado", Berkeley, 1994.

- Leinecker, Richard C. e Nye, Jamie; "Visual C++: ferramentas poderosas", Berkeley, 1995.

- Brain, Marshall; "Introduction to MFC Programming with Visual C++ Version 4.x"; Series Overview (http://www.iftech.com/oltc/vc4mfc/vc4mfc0.stm)

- Microcomputer Components – SAB 8051 Family of Single Chip Microcomputers – User's Manual 7.81- SIEMENS

- MCS 51 Microcontroller Family User's Manual - Intel Corporation – Feb. 1994.

- Stewart, James W.; "*The 8051 mocrocontroller- Hardware, Software and Interfacing"* – Regents/Prentice Hall – 1993.

- Yeralan, Sencer e Ahluwalia, Ashutosh; "*Programming and Interfacing the 8051 microcontroller"*- Addison-Wesley, 1995.

- Drakos, Nikos; "Constructing Object-Oriented Software in Interactive Graphical Programming Environments: An Anthology" - Computer Based Learning Unit, Universityof Leeds, UK . March 1993 . (http://cbl.leeds.ac.uk/nikos/tex2html/examples/concepts/concepts.html)

- Chorafas, Dimitris N.; "Visual Programming Technology"; McGraw-Hill, 1997.

- Tkach, Daniel; Fang, Walter; So, Andrew; “Visual Modeling Technique”; Addison-Wesley, 1996.

- Reckers, Jan; “Visual Languages”; Vakgroep Informatica, Universiteit van Leiden, 1995.

ANEXO I: Descrição dos ícones de operação do Onagro.

OS ÍCONES DE OPERAÇÃO

Os ícones de operação são símbolos gráficos usados para representar as operações que podem ser executadas pela linguagem. Durante sua elaboração houve a preocupação em torná-los o mais intuitivo possível. A maioria deles foi derivada da própria simbologia usada nos fluxogramas. Entretanto, alguns ícones mais específicos foram criados para possibilitar uma gama maior de operações, como por exemplo: comunicação serial, pesquisa em vetores e geração de atrasos, dentre outros.

Eles estão organizados em classes de operações, onde cada classe é representada por uma figura diferente e indica um conjunto de operações semelhantes. São disponíveis, por exemplo, quatro operações de atribuição (atribuição propriamente dita, atribuir zero, atribuir um e permutar dados) agrupadas numa mesma classe. Na maioria dos casos o ícone que representa a classe é o mesmo que indica a operação mais usada daquela classe (veja figura AI.1). No entanto, existem classes que possuem um ícone específico para designá-la, diferente dos demais ícones de operação da classe. É o que ilustra a figura AI.2.

É importante ressaltar que mesmo havendo diferentes ícones numa classe, eles têm características visuais semelhantes. A cor de preenchimento é a mesma e a simbologia entre eles só muda no que eles têm de peculiar. Isto pode ser observado na figura AI.1. Os últimos quatro quadros representam ícones diferentes relacionados com a operação de atribuição. Existem pequenas, mas importantes, mudanças no desenho a fim de diferenciá-los. Em contra partida, existe ao mesmo tempo entre eles uma semelhança que nos leva a perceber que eles pertencem a mesma classe.

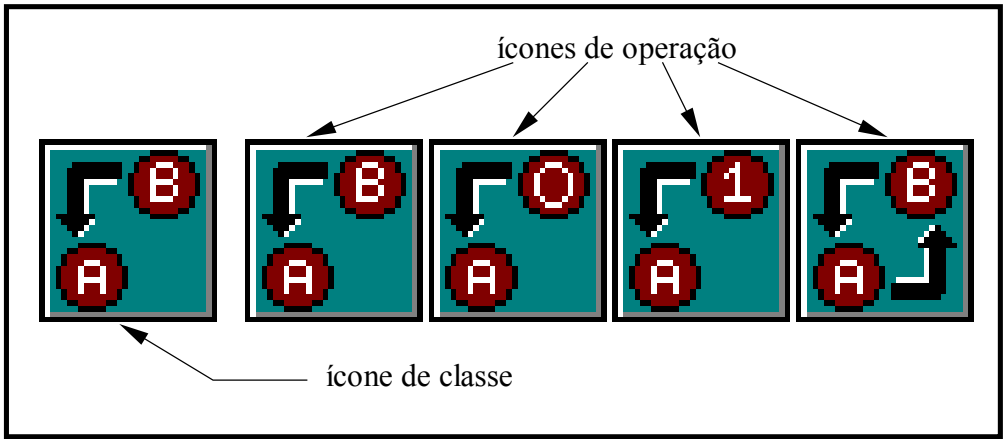


Figura AI.1: Ícones de atribuição

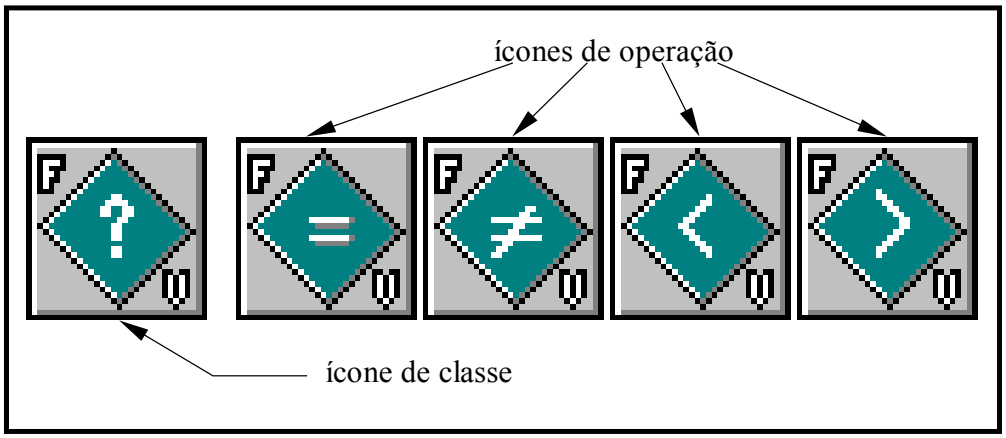


Figura AI.2: Ícones de comparação

Alguns ícones, de classes distintas, possuem a característica de poder mudar o fluxo de execução do programa. Muitas vezes estas mudanças (paradas, instrução de repetição, chamadas e retorno de subrotinas, principalmente) dificultam o entendimento e a depuração do programa. Para diminuir os inconvenientes pertinentes às mudanças no fluxo de execução, adotou-se a mesma cor de fundo para a esses ícones, o que melhora a clareza do programa.

Para se inserir um ícone de operação na área de desenho, basta que o usuário entre no modo de inserção (opção Inserir do menu de linha) e escolha a classe do ícone desejado. O usuário pode poupar tempo indicando o ícone a ser inserido, por meio de um clique do botão esquerdo do mouse sobre o ícone correspondente na barra de ícones de operação.

Após inserir um ícone de classe na área de desenho, o usuário ativa uma caixa de diálogo para especificar os detalhes da operação. Isto é feito pressionando-se duas vezes seguidas o botão esquerdo do mouse com o cursor posicionado dentro do ícone. Neste diálogo, o usuário é conduzido a indicar a operação, dentre as disponíveis na classe, que ele deseja realizar e preencher os campos de dados usados na operação selecionada. Além disto, todos os diálogos possuem um campo destinado à documentação, onde o usuário pode descrever textualmente o que a operação vai realizar. Esta informação é útil quando o usuário desejar depurar o programa, através do modo de inspeção. Este modo é ativado, conforme dito no início deste capítulo, pela opção Editar-Inspecionar do menu de linha ou através da barra de ferramenta.

O preenchimento dos campos é dirigido pelo diálogo, de tal forma, que o usuário possa conhecer os valores permitidos para aquele campo, dentro do contexto da operação. Para isso o diálogo dispõe, para cada campo de dado, de uma lista com os valores permitidos.

Alternativamente, o usuário pode preencher os campos do diálogo de forma direta, digitando os identificadores de símbolos envolvidos na operação. Mesmo assim, o diálogo só aceita os valores permitidos pela operação, sendo geradas mensagens de erro quando o usuário tentar fechar o diálogo, caso tenha sido cometido algum erro.

Assim, logo no detalhamento das operações, procura-se evitar erros comuns que ocorrem nas linguagens tradicionais, onde o programador tem que ficar atento a "contextualidade" da operação, no que se refere aos seus parâmetros. Normalmente isto é fonte de erros posteriores de compilação.

Para fechar o diálogo o usuário pode confirmar as informações inseridas, usando o botão OK, ou cancelá-las através do botão CANCELAR. No encerramento da caixa de diálogo o sistema faz uma crítica das informações introduzidas notificando qualquer irregularidade sobre elas, como por exemplo, um aviso sobre um identificador que ainda não tenha sido definido.

Imediatamente após o fechamento do diálogo, o ícone de classe é modificado para indicar a operação selecionada. O usuário pode também mudar o ícone de operação pressionando o botão direito do mouse duas vezes seguidas, com o cursor posicionado dentro do ícone.

Nos ícones, existem quatro áreas para conexão, como mostra a figura AI.3. Na maioria dos casos, eles possuem uma área de saída de ligação e três de entrada. A ligação se faz quando o usuário entra no modo de ligação (opção Editar-Ligar do menu de linha ou através do segundo botão da barra de ferramenta) e pressiona o botão esquerdo do mouse numa das áreas livres do ícone inicial, arrastando-o até uma área livre do ícone final.

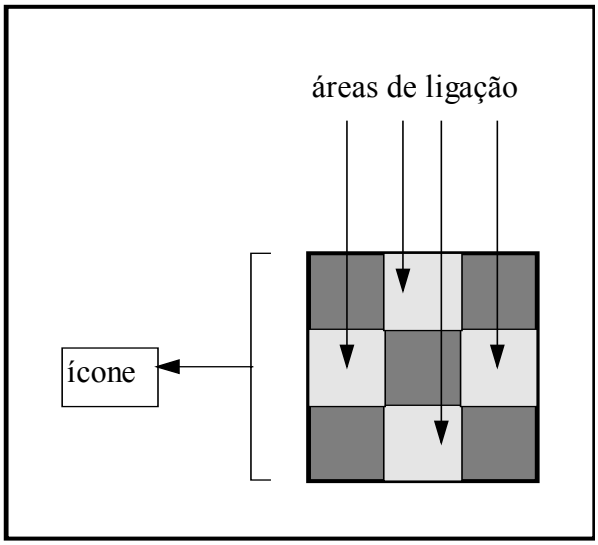


Figura AI.3: Áreas do ícone usadas para ligação

Após ter sido introduzido o programa-fonte na área de desenho, pode-se ativar o compilador para que o mesmo gere o correspondente código objeto. Isto é feito de duas formas: a primeira é através do menu de linha usando a opção Compilar-Começar e a segunda é por meio da barra de ferramentas, ativando seu terceiro botão da direita para a esquerda.

Feito isto, o compilador abre uma caixa de diálogo, mostrada na figura AI.4, e reporta os erros ocorridos ou, caso não seja detectado erros, mostra o código em Assembly

do referido programa-fonte. Neste momento, o usuário pode pressionar o botão OK, fazendo com que o compilador ative um montador/linkeditor padrão para a geração do código em linguagem de máquina e armazene, em disco, os arquivos gerados: o arquivo com o programa em linguagem Assembly (.ASM), o arquivo de listagem (.LST), o arquivo objeto (.OBJ), o arquivo em binário (.BIN) e o arquivo no formato hexadecimal (.HEX).

Caso tenha sido gerado algum erro de compilação, o compilador conduz o usuário ao diálogo de descrição do ícone de operação onde ocorreu o erro e permite que se faça as correções necessárias. Após isto, pode-se ativar o botão **Recompilar** do diálogo de compilação, solicitando ao compilador que recomece o processo de tradução.

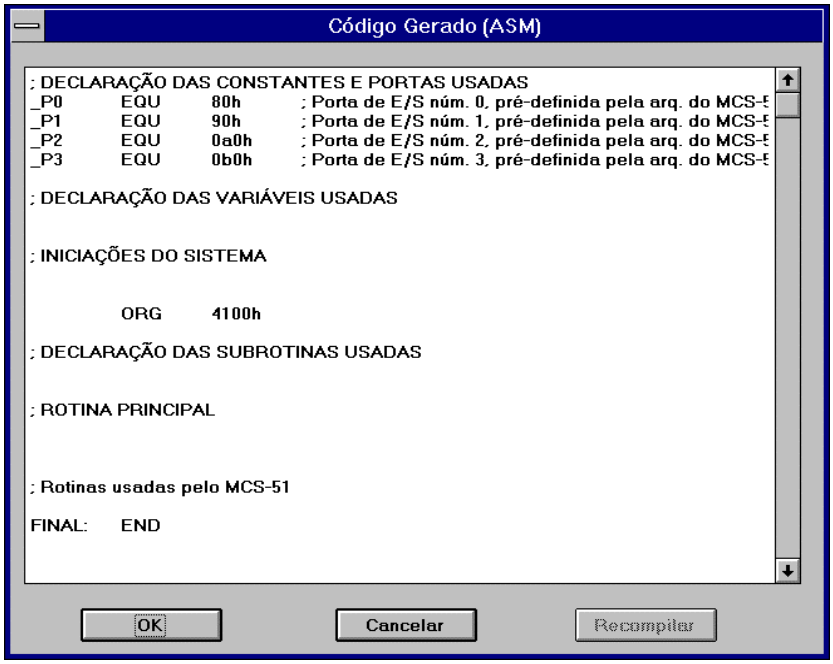


Figura AI.4: Caixa de diálogo do resultado da compilação

Apesar das verificações feitas logo na descrição dos ícones de operação, podem ser gerados erros posteriores de compilação, como por exemplo: tipos misturados na instrução de comparação, valor do campo destino inválido, incompatibilidade entre parâmetros, dentre outros. Tais erros podem ser conseqüência de alterações posteriores feitas em identificadores usados na operação onde foi detectado o erro.

Operação de atribuição

Como mencionado anteriormente, existem quatro operações de atribuição disponíveis na linguagem: atribuição propriamente dita, atribuir zero, atribuir um e permutar dados. No diálogo da operação de atribuição (figura AI.5), o campo DESTINO possui uma lista de identificadores de símbolos que podem receber valores (variáveis, portas de entrada e portas de entrada/saída). Já o campo ORIGEM do mesmo diálogo terá sua lista de valores modificados de acordo com a operação selecionada. Na operação de atribuição propriamente dita, esse campo tem sua lista composta por constantes, variáveis e portas de entrada e/ou saída. Entretanto, na operação de permuta de dados (SWAP) essa lista é também formada apenas por identificadores que possam receber valores. Por último, nas operações de ZERAR e SETAR o campo ORIGEM é "desabilitado", pois o valor de origem está implícito na própria operação.

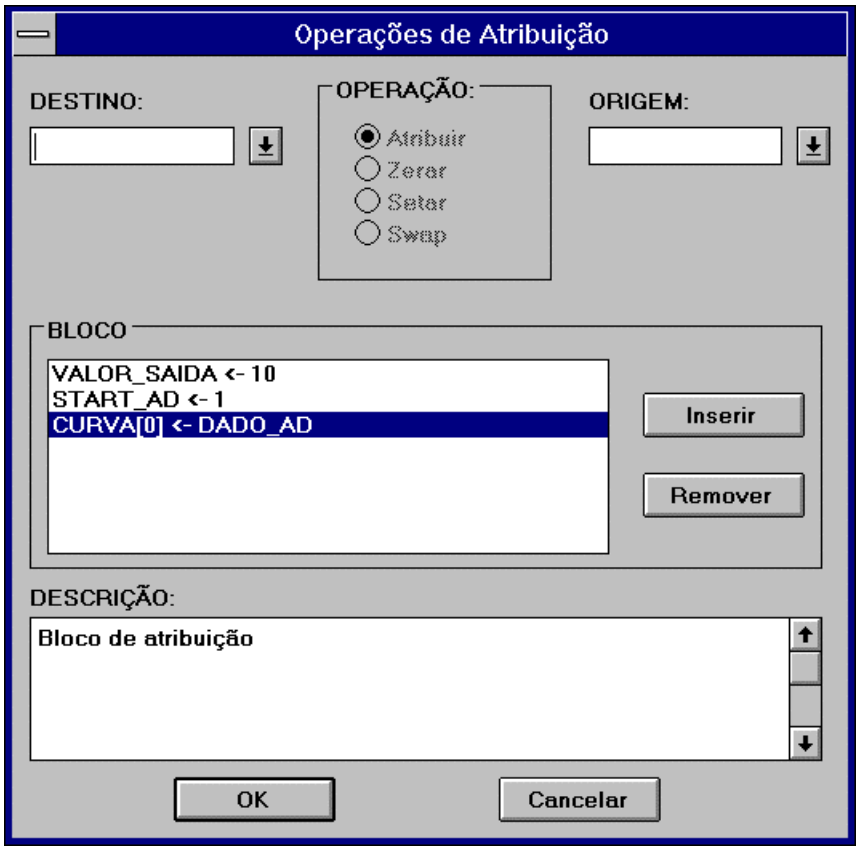


Figura AI.5: Caixa de diálogo das operações de atribuição

Caso seja envolvido algum identificador de porta de e/s na operação de atribuição, o ícone de operação é modificado para realçar que a operação manipula uma porta de e/s, conforme mostra a figura AI.6.

Como a operação de atribuição é uma das mais comuns dentro de um programa, seu diálogo dispõe de um campo opcional, onde o usuário pode criar um bloco de atribuições, que serão executadas uma após a outra. Isto evita repetição de ícones de atribuição, pois um único ícone pode representar um conjunto destas operações.

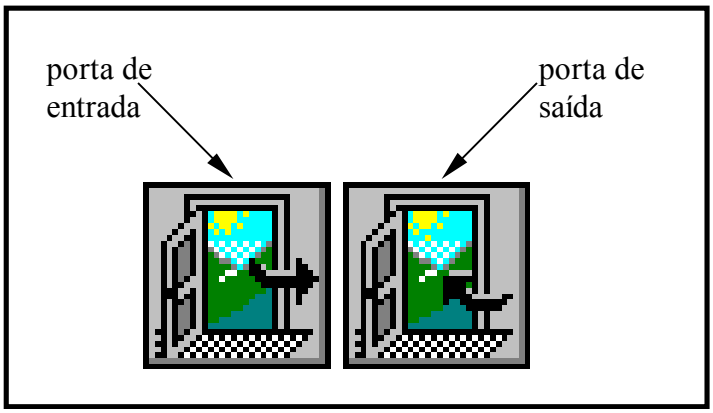


Figura AI.6: Ícone de operação com portas de e/s

Caso se deseje criar um bloco de atribuições, deve-se selecionar a operação e os identificadores desejados e ativar o botão INSERIR para cada uma das operações do bloco. Pode-se também eliminar uma operação do bloco através da seleção da operação que se pretende excluir e da ativação do botão REMOVER ou simplesmente, pressionado-se duas vezes seguidas o botão direito do mouse sobre a operação que se quer eliminar.

O diálogo mostrado na figura AI.5 possui um bloco com três operações de atribuição, todas elas representadas por um único ícone.

Operações aritméticas

As quatro operações aritméticas: soma, subtração, multiplicação e divisão possuem ícones próprios (ver figura AI.7) e executam cálculos entre dois operandos, levando o resultado para um identificador que possa receber valor (variáveis, portas de saída e portas bidirecionais). Assim, o diálogo de descrição destas operações (figura AI.8) inicia as caixas de listagens dos campos RESULTADO, VALOR1 e VALOR2 com identificadores compatíveis. Porém, nos casos de tipos misturados de valores, o compilador assume o tipo base como sendo o do identificador que receberá o resultado. Desta forma, pode haver truncamento de valores, contribuindo para a geração de resultados não esperados.

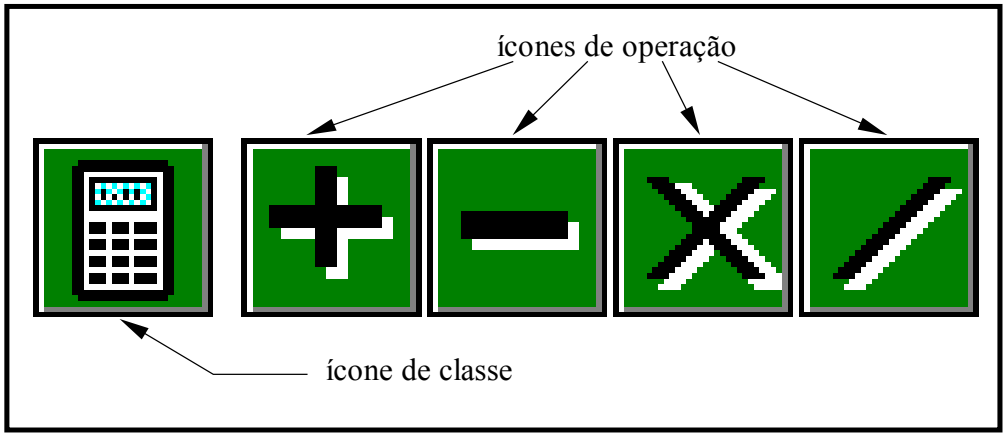


Figura AI.7: Ícones de operações aritméticas

Figura AI.8: Caixa de diálogo das operações de aritméticas

Operações lógicas

Os ícones de operações lógicas são usados para realizar as operações lógicas not, or, and e xor bit a bit entre dois operandos. Da mesma forma como ocorre nas operações aritméticas, o resultado de uma operação lógica deve ser conduzido para um identificador onde este resultado possa ser armazenado. Estas operações lógicas não geram fluxos condicionais de execução. Desta forma, os ícones (figura AI.9) só apresentam um ponto de saída. Este tipo de operação é muito útil para mascarar bits dentro de uma palavra onde nem todos os bits são efetivamente utilizados num processamento.

Na figura AI.10 temos a vista do diálogo relativo às operações lógicas. Ele é semelhante ao das operações aritméticas. A diferença está nos tipos de operadores envolvidos; porém, os dois possuem as mesmas restrições quanto ao procedimento de preenchimento dos campos.

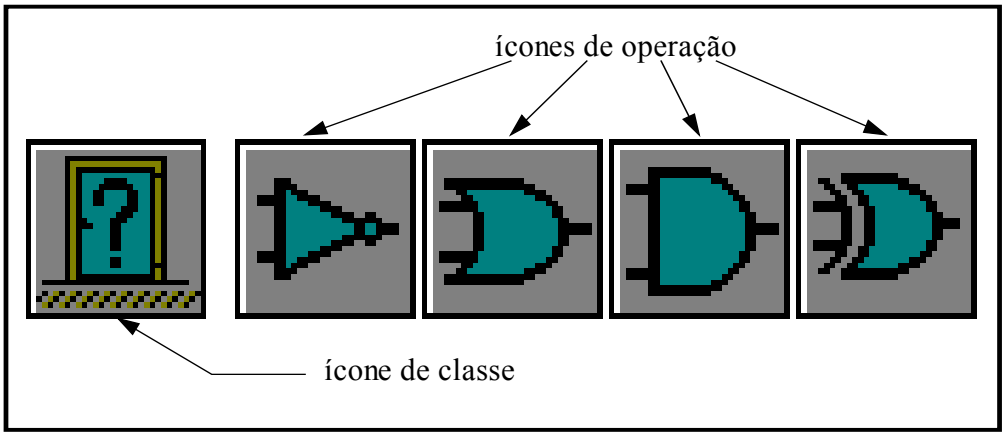


Figura AI.9: Ícones de operações lógicas

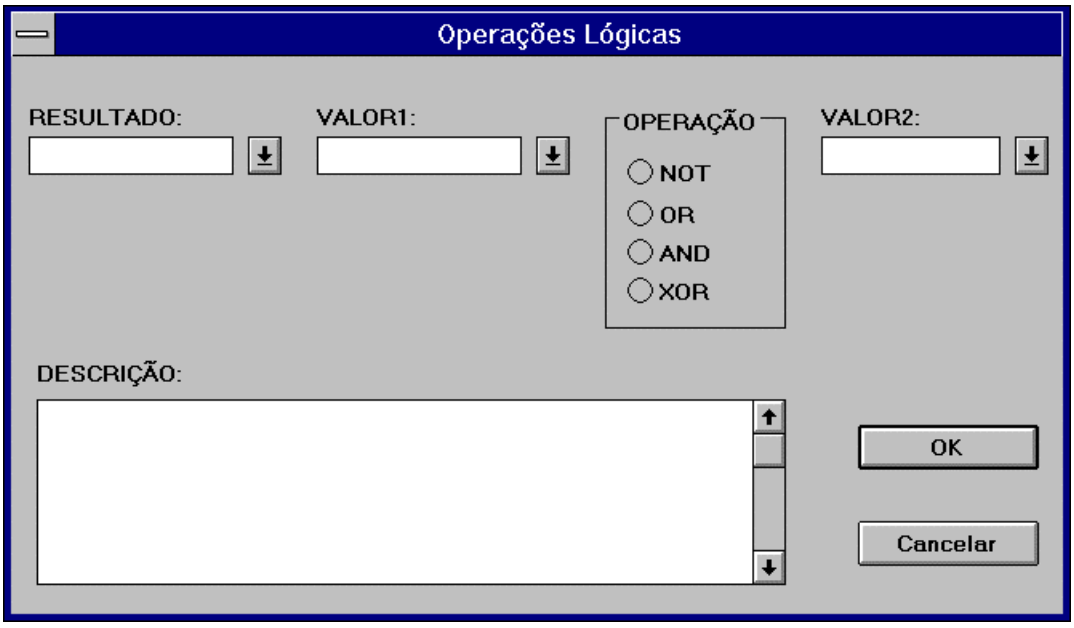


Figura AI.10: Caixa de diálogo das operações lógicas

Operações de rotação e deslocamento

Foram implementadas as operações tradicionais de rotação e deslocamento disponíveis na maioria de microcontroladores. Estas operações permitem realizar rotação e deslocamento, em ambos os sentidos, de identificadores do tipo variável ou portas de e/s.

Os ícones que representam estas operações podem ser vistos na figura AI.11. Podemos observar que nas operações de deslocamento (segundo e terceiro ícones) há a entrada de um zero no LSB (deslocamento a esquerda) e no MSB (deslocamento a direita). Os últimos dois ícones simbolizam as operações de rotação a esquerda e a direita, respectivamente.

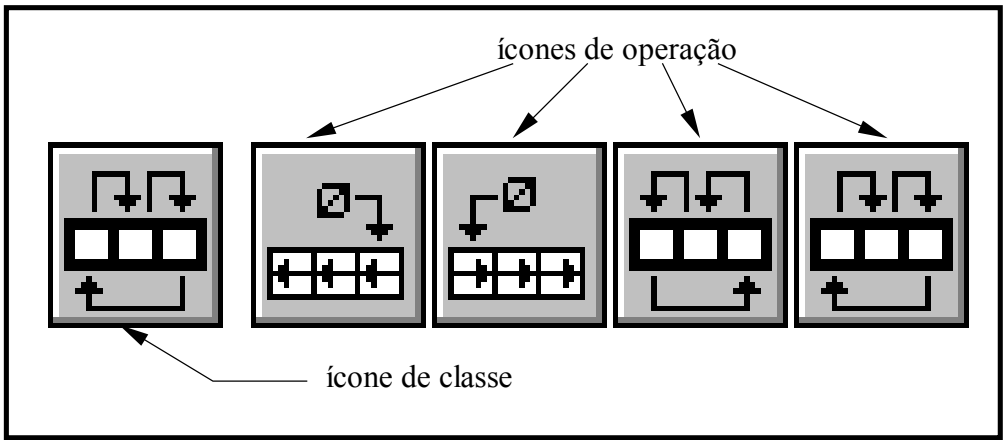


Figura AI.11.: Ícones de rotação e deslocamento

Para descrever a operação desejada, basta abrir o diálogo de preenchimento e indicar o identificador (campo LABEL) e o tipo de operação de que se quer executar, conforme figura AI.12.

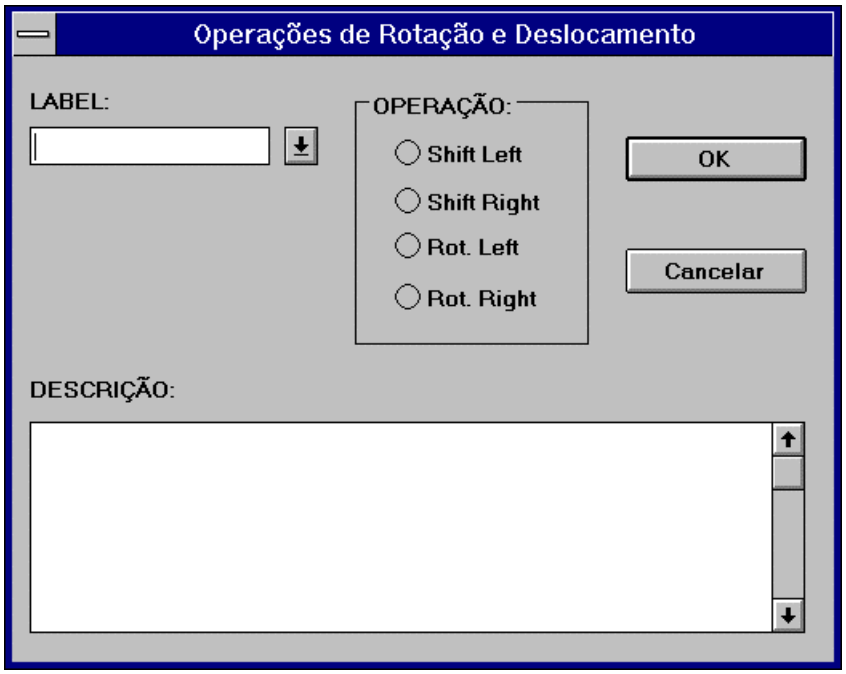


Figura AI.12.: Caixa de diálogo das operações de rotação e deslocamento

Operações de comparação

Conforme mostrado na figura AI.2, as tomadas de decisão usam os ícones de comparação: igual a, diferente de, menor que e maior que, que implementam os operadores relacionais =, **!=**, **< e >**, respectivamente. Desta forma, constroi-se as condições lógicas simples. Estes ícones podem ainda ser encadeados para formar condições compostas com AND, OR e NOT através de adequada interligação entre eles, como será mostrado posteriormente.

As condições lógicas são discriminadas por um diálogo próprio, visto na figura AI.13, e relaciona dois valores de mesmo formato através da indicação de uma das quatro operações possíveis. O preenchimento dos campos VALOR1 e VALOR2 pode ser feito diretamente usando o teclado ou através das caixas de listagens que contêm a lista de identificadores permitidos.

Os ícones de comparação, quando adequadamente combinados com outros ícones, podem ser usados para implementar as estruturas clássicas de decisão (if-else) e de repetição condicional (while-do e do-while) como mostra as figuras AI.14 eAI.15.

Figura AI.13: Caixa de diálogo das operações de comparação

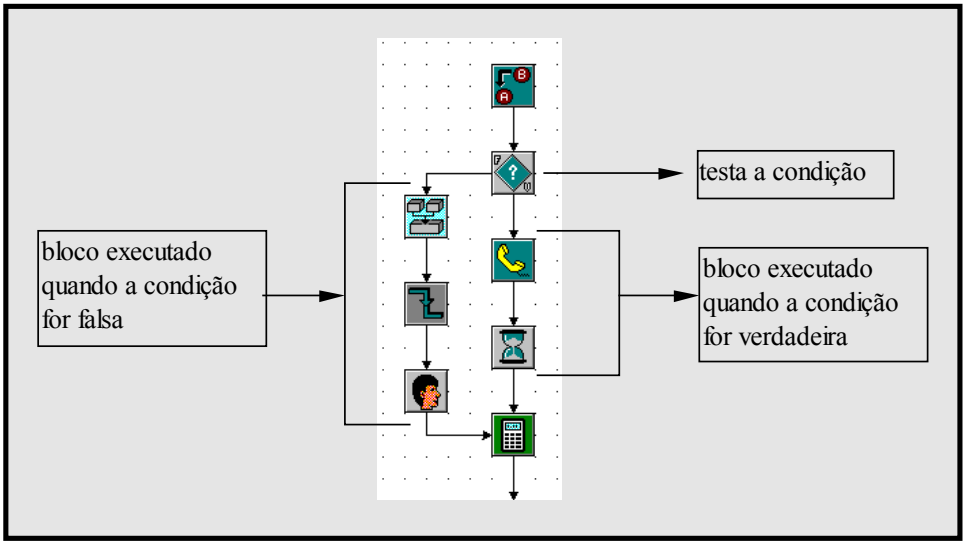


Figura AI.14: Estrutura **if-else**

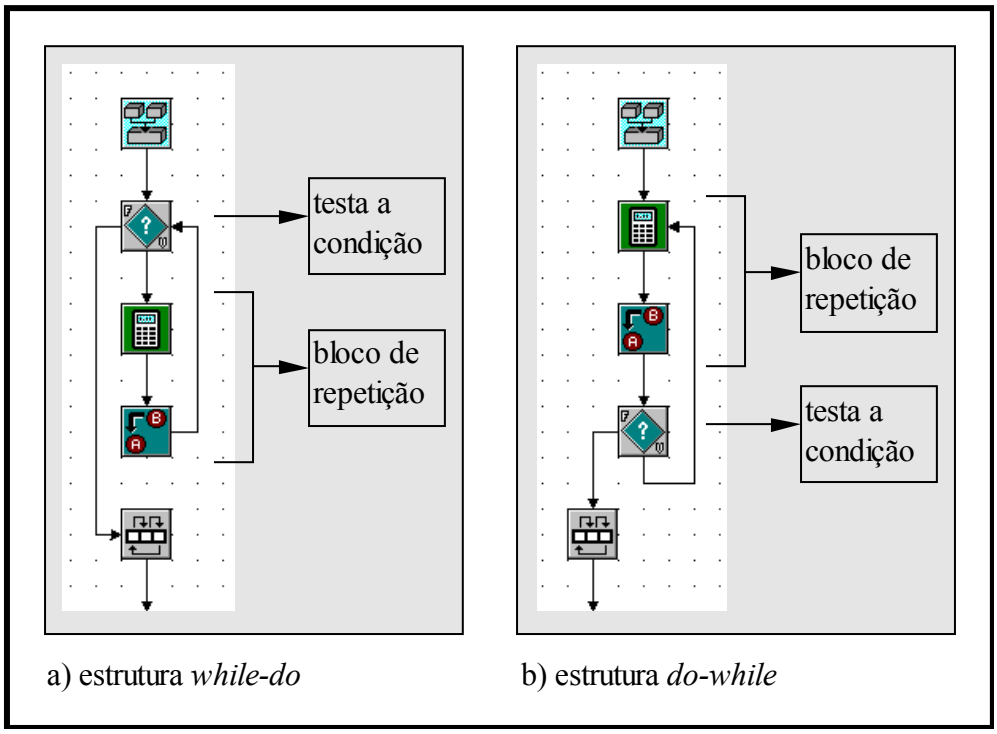


Figura AI.15: Estruturas **while-do** *e* **do-while**

Para implementar condições compostas, que envolvem expressões relacionais com operadores lógicos, deve-se combinar ícones de comparação tomando por base o fluxo de execução. Assim, se tivermos que implementar uma condição composta, por exemplo **X and Y**, teremos dois ícones de operação. Um para cada condição simples (X e Y). A figura AI.16 mostra como essas duas condições podem ser combinadas para formar a condição completa. Note que o ponto C só será executado quando as duas condições forem verdadeiras.

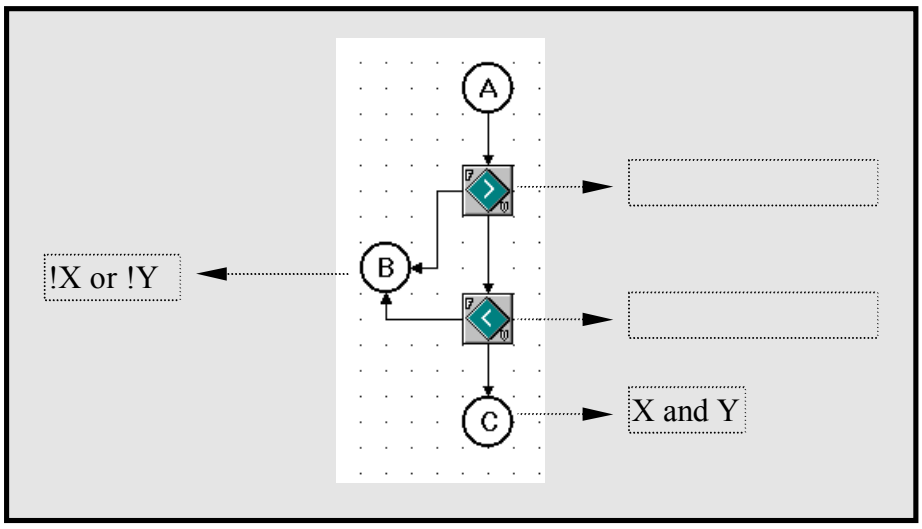


Figura* AI.16*: Exemplo de uma condição composta

Os ícones de comparação, diferentemente da maioria, possuem duas saídas em locais fixos indicados pelas letras V e F (ver figura AI.2). Uma para sinalizar o fluxo de execução no caso de a condição ser verdadeira e outra para o caso da condição ser falsa.

Operação de iteração

Para melhor visualização dos blocos que são executados repetidas vezes, através de "loops" de iterações, foram criados os ícones delimitadores de blocos de repetição, vistos na figura AI.17. O primeiro ícone é o ícone de classe, representando tanto o início como o final do bloco de repetição. Os outros dois são os ícones de delimitação de bloco propriamente dito. Eles sempre trabalham em conjunto, sendo permitido o aninhamento de blocos de repetição. Assim, para que não ocorra erro de compilação o número de ícones delimitadores de blocos deve ser sempre par, ou seja, o bloco deve começar e terminar por um delimitador de bloco. Estes ícones são semelhantes aos símbolos tradicionais BEGIN e END, usados nas linguagens textuais para marcação de blocos.

A repetição é controlada por uma variável de controle designada no diálogo da operação de iteração (figura AI.18). Esta variável pode ser automaticamente gerada pelo próprio ícone de operação. Para isto, basta deixar em branco o campo VAR. CTRL. Esta variável pode ser usada normalmente pelo programa para indicar a iteração corrente. O

próprio programador, no entanto, deve tomar cuidado para não alterá-la, o que pode gerar possíveis erros de execução.

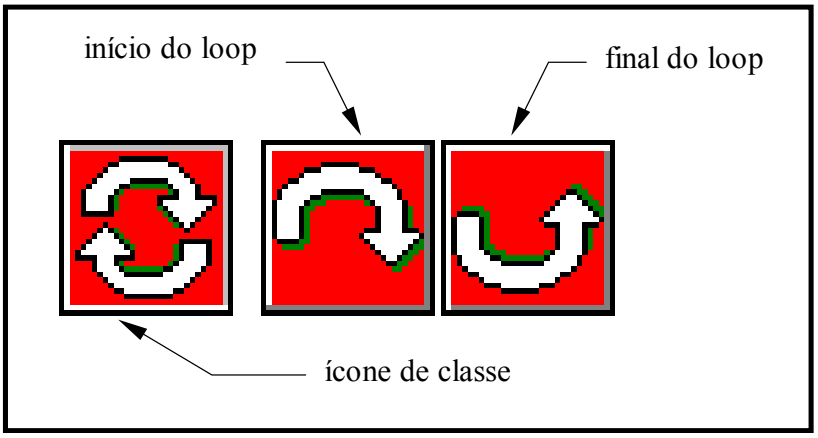


Figura AI.17: Ícones de operação de iteração

As opções presentes no diálogo referem se ao tipo de atualização que será feito na variável de controle, que pode ser incrementada ou decrementada, e a indicação do tipo de ícone (início ou final de bloco). Também é necessário que o usuário digite no campo NUM. REP. um valor inteiro ou um identificador que indicará o número de repetições do bloco. Vale salientar que para terminar um bloco só é preciso que se selecione o ícone de final de bloco, sendo os valores restantes automaticamente inseridos pelo compilador. Para evitar prováveis erros de compilação ou mesmo de lógica, os campos inseridos pelo compilador no ícone de final de bloco ficam desativados para o usuário.

Figura AI.18: Caixa de diálogo da operação de iteração

Operação de geração de atraso

Freqüentemente, em aplicações dedicadas, ocorre a necessidade de se promover um atraso entre duas ações. Isto é comum quando a aplicação controla um sistema cujo tempo de resposta é bem maior que o do microcontrolador. Caso não seja preciso realizar nenhuma outra ação durante o tempo de espera, podemos usar um ícone de operação específico para a geração de atraso de tempos derivados do clock da CPU.

Temos na figura AI.19 o ícone que simboliza esta operação e o diálogo usado para especificá-la. Neste diálogo devemos indicar o tempo desejado (em segundos) para o atraso, usando o campo **Tempo-Desejado**. Entretanto, o tempo efetivo de atraso nem sempre é igual ao tempo desejado, pois a implementação desse ícone utiliza laços aninhados para "gastar o tempo da CPU". Assim, o tempo efetivo está relacionado com a velocidade do microcontrolador e com o número de iterações dos laços envolvidos em sua implementação. Todavia, para que o usuário conheça este tempo o sistema reporta, através do campo **Tempo-Obtido** (em segundos), o seu valor.

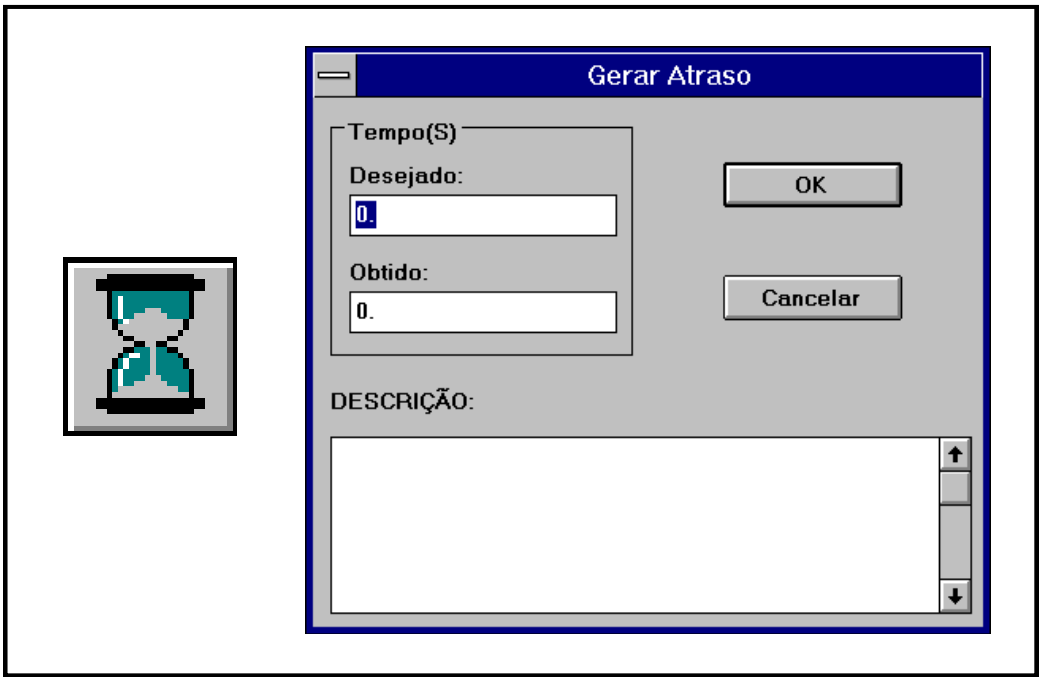


Figura AI.19: Ícone e caixa de diálogo da operação geração de atraso

Para se ter uma noção dos tempos possíveis para o ícone de geração de atraso, considere por exemplo, que o clock do microcontrolador seja de 6,144MHz, então o tempo máximo será de 65,66s com uma resolução de 33,20μs.

Operação de comunicação serial

Em virtude da maioria dos microcontroladores disporem, em sua arquitetura, de uma interface de comunicação serial, optou-se em implementar ícones de operação para manuseio desta interface. Assim, são disponíveis, na linguagem, dois ícones de operação para transferir dados do microcontrolador para dispositivos externos e destes para o primeiro. Estes ícones, mostrados na figura AI.20, são responsáveis pela transmissão de qualquer identificador do programa para o meio exterior, bem como pela recepção dos dados de dispositivos externos, que são armazenados em identificadores do programa.

Para descrever a operação de comunicação serial, basta abrir o diálogo associado a esta operação (figura AI.21) e indicar o identificador que será usado, bem como o tipo de operação: enviar ou receber.

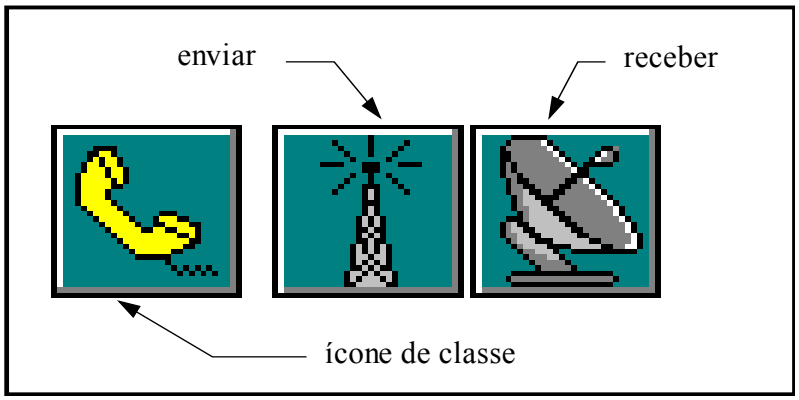


Figura AI.20: Ícones de operação de comunicação serial

Evidentemente, antes de usar qualquer um dos ícones de comunicação serial, deve-se programar os parâmetros da interface, como é indicado na seção 3.4.2.

Figura AI.21: Caixa de diálogo da operação comunicação serial

Operação de pesquisa em vetor

Esta operação, cujo ícone e a caixa de diálogo são mostrados na figura AI.22, pode ser utilizada para verificar se um dado elemento pertence a um vetor. Tal operação é condicional, pois o elemento pode ou não pertencer ao vetor pesquisado. Portanto, existem

dois fluxos de execução possíveis para o ícone dessa operação. Um associado à condição verdadeira, indicando que o elemento pertence ao vetor (saída V do ícone) e outro à condição falsa, indicando que o mesmo não foi encontrado (saída F do ícone).

Na caixa de diálogo o usuário deve indicar o nome do vetor utilizado na pesquisa (campo VETOR) e este deve ser previamente declarado, pois é feita uma verificação sobre a consistência das informações introduzidas. Além do nome do vetor, deve ser indicado o valor a ser pesquisado (campo **Valor Pesquisado**) e, opcionalmente, o usuário pode requerer que o índice do elemento encontrado no vetor seja armazenado em um identificador. Este índice se constitui no retorno da operação de pesquisa. Entretanto, nos casos em que o elemento pesquisado não pertença ao vetor, o valor de retorno não será usado pelo compilador.

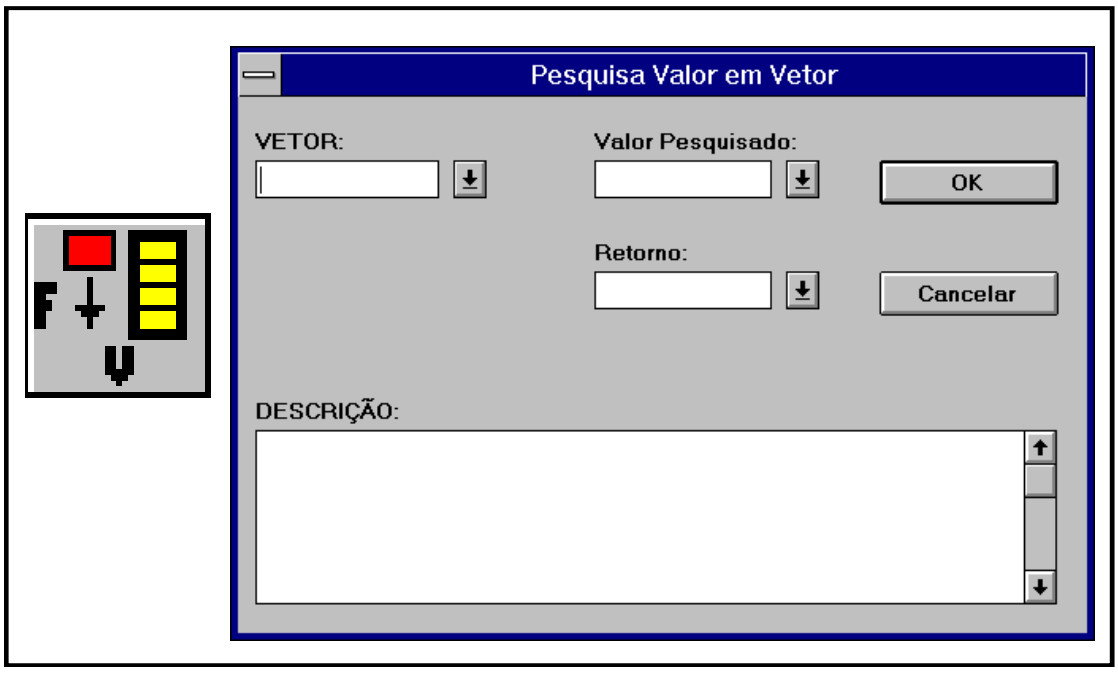


Figura* AI.22*: Ícone e caixa de diálogo da operação pesquisa em vetor

Operação de transformação vetorial

Este é outro ícone de operação destinado a manipulação de vetores. Sua função é relacionar elementos de dois vetores. Ele pesquisa um valor de entrada em um vetor e a partir do índice gerado, atribui a um identificador o elemento correspondente do outro vetor.

Um dos objetivos da implementação deste ícone de operação foi o de diminuir o esforço do programador em projetar rotinas de conversão de códigos, usuais em aplicações dedicadas.

Como pode ser visto na figura AI.23, o diálogo de especificação dessa operação solicita os nomes dos vetores utilizados (campos VETOR1 e VETOR2); o dado de entrada (campo FONTE); e o nome do identificador que receberá o dado de saída (campo DESTINO).

Da mesma forma que o ícone de operação de pesquisa em vetor, o ícone de operação de transformação vetorial (figura AI.23) possui duas saídas, pois o valor de entrada pode não ser encontrado no primeiro vetor. Neste caso, o identificador indicado pelo campo DESTINO não será modificado e o fluxo de execução assumirá a saída F. Caso contrário, o fluxo assumirá a saída complementar V.

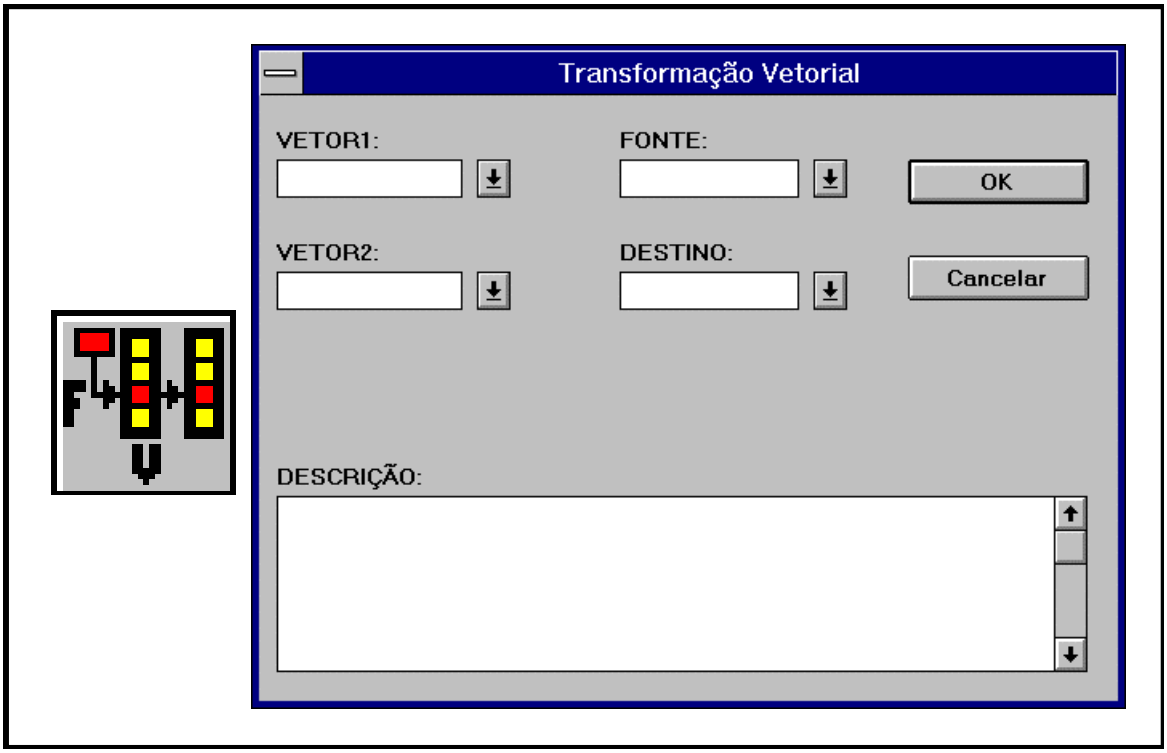


Figura AI.23: Ícone e caixa de diálogo da operação de transformação vetorial

Operação de geração de bordas

Freqüentemente em aplicações com microcontrolador é comum aplicar a bits externos, bordas de subida ou descida, a fim de disparar um determinado evento (gerar um sinal de START em um conversor analógico-digital, por exemplo). Este tipo de operação pode ser realizado com o uso dos ícones de geração de bordas, expostos na figura AI.24.

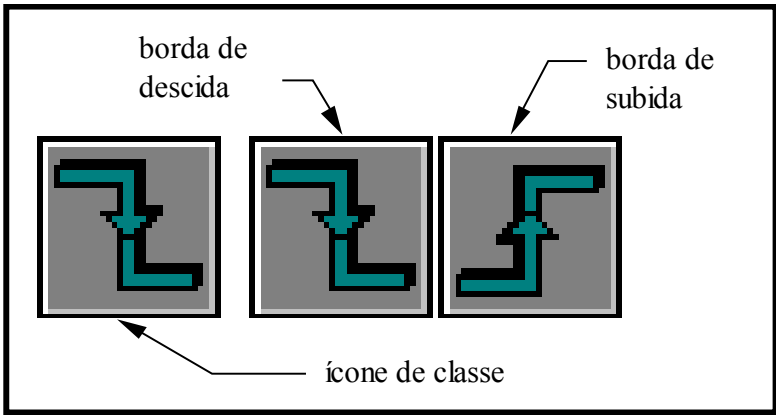


Figura AI.24: Ícones de geração de bordas

Deve-se tomar um certo cuidado com essa operação, pois tanto na borda de subida como na de descida, o tempo de transição depende da velocidade do microcontrolador. O código gerado simplesmente liga e desliga o bit do identificador (borda de descida) ou faz a operação inversa (borda de subida).

É importante observar que esta operação só faz sentido se o identificador (campo LABEL do diálogo mostrado na figura AI.25) for uma porta de e/s. Entretanto, o compilador permite que a operação seja executada também com variáveis, para fins de simulação.

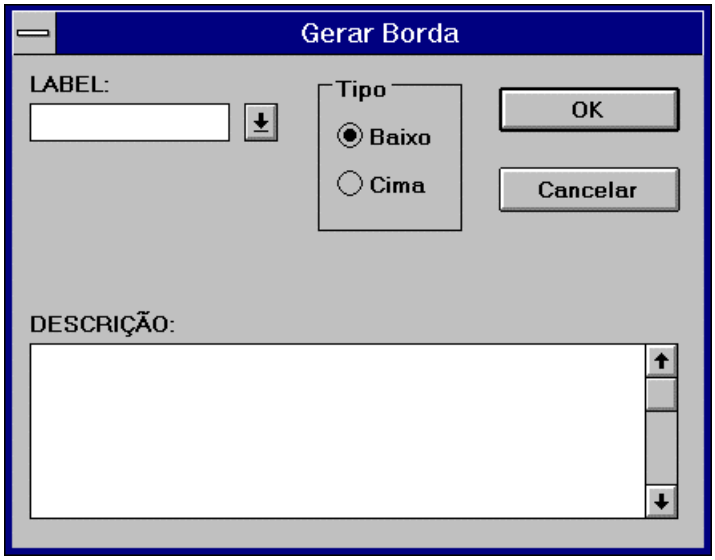


Figura AI.25: Caixa de diálogo da operação geração de bordas

Operação juntar bits

Em virtude de a linguagem dispor de mecanismos para criação de portas de e/s, com número de bits programáveis pelo usuário, e da necessidade de agrupar os bits de diferentes portas de e/s em um único identificador, foi implementado o ícone de operação juntar bits.

Na figura AI.26 temos o ícone e o diálogo dessa operação. No diálogo devem ser indicados os identificadores envolvidos na operação: o que vai receber os bits agrupados (campo RESULTADO), o que indica os bits mais significativos (campo HIGHBITS) e o que indica os bits menos significativos (campo LOWBITS). Nesta operação o número total de bits agrupados não pode exceder a oito.

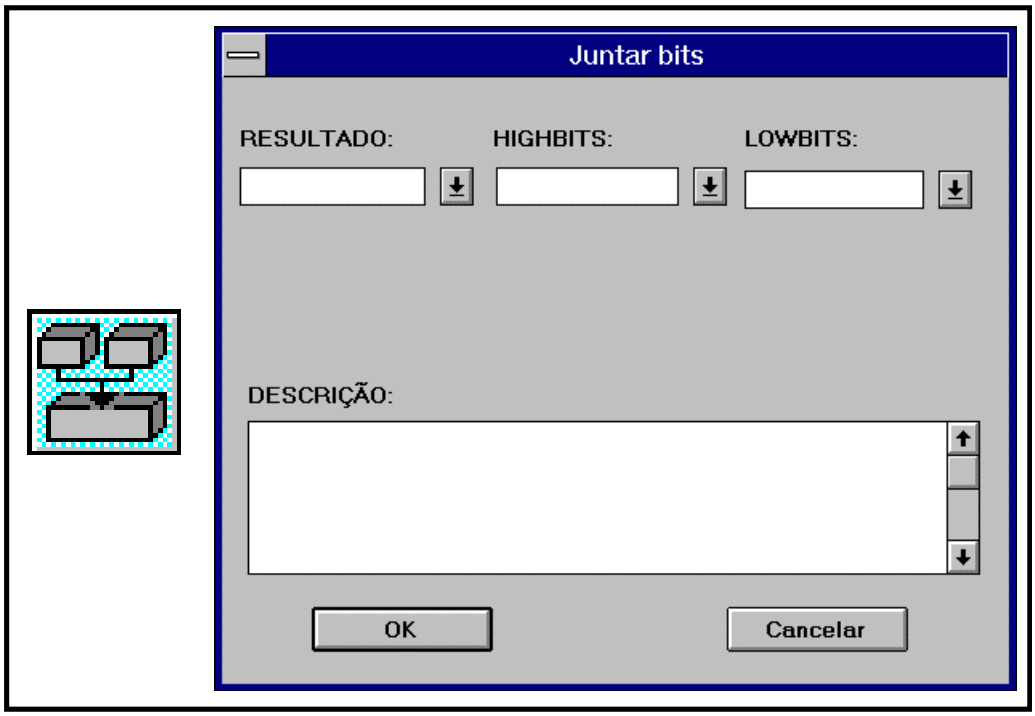


Figura AI.26: Ícone e caixa de diálogo da operação juntar bits

Operação com subrotinas

Os ícones utilizados para indicar o início de uma subrotina, ativá-la e promover o retorno dela para o módulo que a ativou, são chamados ícones de operação com subrotina, mostrados na figura AI.27. Estes ícones interagem com os identificadores de subrotinas permitindo a estruturação do programa em blocos funcionais, a fim de melhorar sua organização e também, promover um maior reaproveitamento das rotinas já implementas.

Conforme visto anteriormente, podemos entrar no modo de edição de uma subrotina, selecionando a opção Identificadores-Subrotinas do menu de linha e acionando o botão **Editar** do diálogo de subrotina. Podemos também realizar esta mesma operação por meio da caixa de listagem localizada na barra de ferramentas. Quando entramos pela primeira vez no modo de edição de subrotina, o sistema automaticamente insere dois ícones de operação. Um para indicar o início da subrotina e outro para indicar o final da mesma. Este último ícone de operação indica também o ponto de retorno ao módulo que a ativou. Portanto, estes dois ícones só podem aparecer dentro de uma subrotina. O editor de fluxogramas não permite que eles sejam parte integrante do programa principal.

Os ícones de subrotina geram o código de máquina necessário para criar a interface da subrotina com o bloco que a ativou. Eles trabalham com a pilha do microcontrolador para garantir a integridade dos dados dos módulos, salvando e recuperando o contexto destes sempre que houver chamada ou retorno de subrotina. Eles ainda passam, através da pilha, valores de argumentos e de retorno, permitindo a comunicação entre os módulos envolvidos.

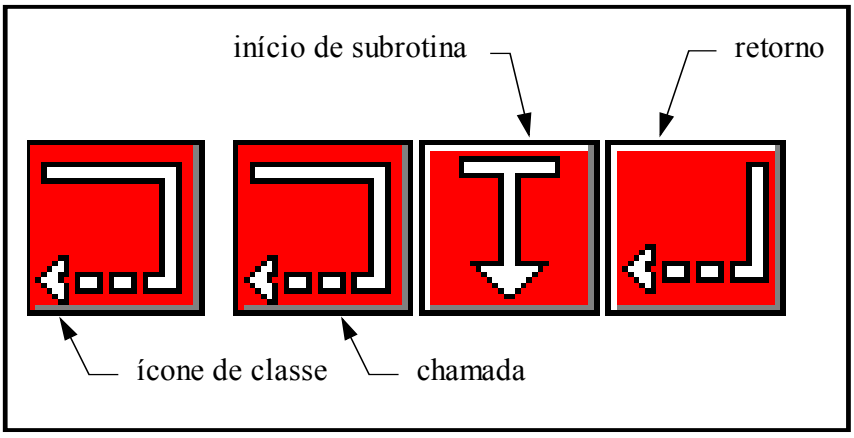


Figura AI.27: Ícones de subrotinas

Para utilizar um dos ícones associados às subrotinas, deve-se preencher o diálogo mostrado na figura AI.28. Para os ícones de início e retorno será preciso apenas indicar a operação (campo **Operação**) e o identificador da subrotina (campo LABEL). Entretanto, para o ícone de chamada, além destas informações, é necessário que o usuário indique a

lista de argumentos (campo ARGUMENTOS) que serão passados e qual identificador receberá o valor de retorno (campo RETORNO), desde que a subrotina em questão tenha sido definida com estas especificações. A caixa de verificação **Interrupção** é usada pelo sistema para indicar ao usuário a natureza da subrotina, ou seja, se ela é de interrupção ou não.

Nos casos em que se seleciona o ícone de chamada de uma subrotina que tenha sido descrita com ícone próprio, o editor de fluxogramas troca o ícone padrão do sistema pelo ícone indicado na definição da subrotina (visto na seção 3.2). Isto é feito logo após o fechamento do diálogo, permitindo expressar numa única imagem a idéia fundamental do processamento efetuado por aquela subrotina, o que facilita o entendimento do programa.

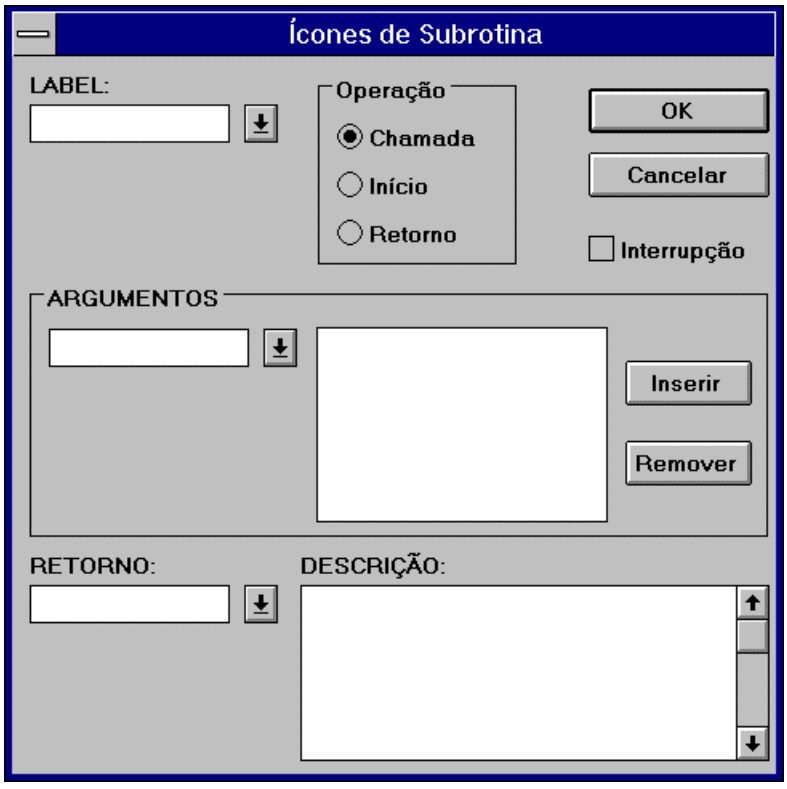


Figura AI.28: Caixa de diálogo dos ícones de subrotinas

Ícones complementares

A linguagem dispõe de quatro ícones que não são propriamente ditos ícones de operação. Eles são ícones auxiliares usados para facilitar a introdução do programa-fonte através do editor gráfico. Estes ícones, mostrados na figura AI.29, são opcionais porém, eles permitem, quando convenientemente utilizados, uma melhor organização do programa.

Dentre estes ícones, dois são usados para demarcar o início e o final do programa-fonte. Eles orientam o compilador sobre onde começa o fluxo de execução (ícone de início) e onde termina o programa (ícone de final). Caso não seja incluído o ícone de início de programa, o compilador assume que o primeiro ícone a ser executado é o que fica mais próximo do canto superior esquerdo da área de desenho.

O ícone de início de programa, que só deve aparecer uma vez dentro do programa-fonte, é responsável também, pela geração de parte do chamado "start-up code" ou código de partida, onde se inicia alguns registradores e posições de memória. Já o ícone de final de programa, pode aparecer em mais de uma posição do programa entretanto, para uma boa estruturação da aplicação, recomenda-se que ele seja usado em um único ponto do programa.

Os outros dois ícones são utilizados para criar junções e conexões entre ícones. O ícone de junção deve ser utilizado nos casos onde o número de pontos de entrada de um determinado ícone for menor que o número de ícones cujas saídas se direcionam a ele, pois como já mencionado, cada ícone possui no máximo três pontos de entrada.

O ícone de conexão tem a função de conectar um ou mais ícones a um outro que não esteja geograficamente próximo, onde a ligação usual entre eles se mostraria esteticamente inconveniente.

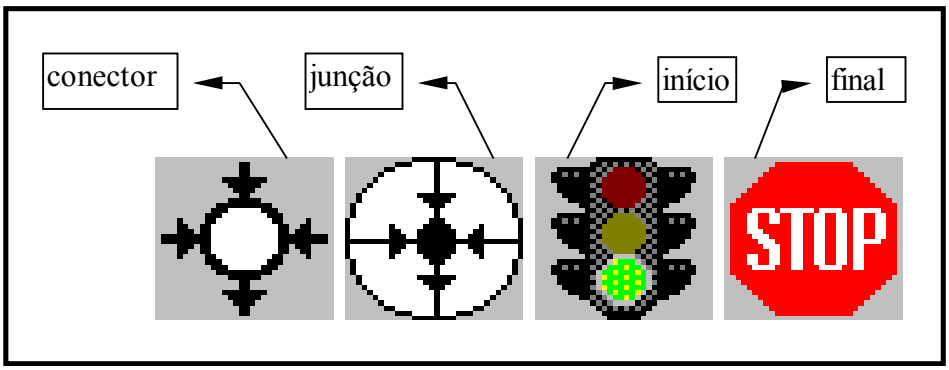


Figura AI.29.: Ícones complementares

ANEXO II: Softwares para desenvolvimento de sistemas microcontrolados.

- **8051 Embedded Workbench for Windows, C-SPY 8051 for Windows - IAR Systems**
  Home Page: http://www.iar.se/

- **8051 MX Development Tools – Raisonance**
  Home Page: www.raisonance.com/products/8051/mx.php

- **8051 Software Development Tools V6.0 – Tasking Software**
  Home Page: http://www.tasking.com/products/8051

- **8052 Simulator for Windows - Vault Information Services**
  Home Page: http://www.vaultbbs.com/sim8052/

- **9-Bit Solution Microcontroller Network - Cimetrics Technology**
  Home Page: http://www.cimetrics.com/

- **ApBuilder – Intel Corporation**
  Home Page: http://www.intel.com/

- **AvCASE51 Embedded Software Dev. Pkg. - Avocet Systems**
  Home Page: http://www.avocetsystems.com/

- **BASIC Stamp**
  Home Page: www.parallaxinc.com

- **BL51 CODE Banking Linker, RTX51 Real Time Operating Systems, DS51 Source Level Debugger/Simulator, 8051-DriveWay Code Generation Tool, MCS® 51, MCS® 251 Microcontroller Software Development Tools, ProView Turbo Tools, A51 Macro Assembler Kit, DK51 Complete 8051 Developer's Kit, PK51 Professional Developer's Kit, C51 C Language Compiler - Franklin Software**
  Home Page: http://www.fsinc.com/

- **BXC-51 IDE - TAVVE Software Co.**
  Home Page: http://www.mindspring.com/~tavve/8051.html

- **C51 Professional Developer's Kit (PK51) - Keil Software**
  Home Page: http://www.keil.com/

- **C-8051 v5.0 & SimCASE - Archimedes Software**
  Home Page: www.archimedesinc.com/8051.htm

- **C166 Software Development Tools 7.5 – Tasking Software**
  Home Page: www.tasking.com/products/c166-st10/c166.htm

- **CCS C Compiler – VOGEL Elektronik**
  Home Page: www.vtec.ch

- **ChipView-51 High-Level/Low-Level Debug - ChipTools**
  Home Page: http://www.chiptools.com/

- **CMX-RTX, CMX-TINY, CMX-TINY+ RTOS - CMX Company**
  Home Page: http://www.cmx.com/

- **CodeVisionAVR – HP Infotech**
  Home Page: www.hpinfotech.ro

- **COMPASS/51 - Production Languages Corp.**
  Home Page: http://www.plcorp.com/

- **Embedded C++ - Green Hills Software Inc.**
  Home Page: www.ghs.com/ec++.html

- **FED Compiler – Forest Electronic Developments**
  Home Page: www.fored.co.uk

- **MICRO/C-51 C Compiler Kit - Micro Computer Control**
  Home Page: http://www.mcc-us.com/

- **MicroVision – Keil Software**
  Home Page: www.keil.com

- **MPLAB - Microchip**
  Home Page: http://www.microchip2.com/

- **PARSIC - Swen Gosh**
  Home Page: www.parsic.de

- **PicBasic Compiler – MicroEngineering Labs**
  Home Page: www.melabs.com

- **PowerBASIC**
  Home Page: www.powerbasic.com

- **Read51 – Rigel Corporation**
  Home Page: www.rigelcorp.com/read51.htm

- **Realizer - Actum Solutions**

  Home Page: www.actum.com

- **STIMGATE Target Controller - RAMTEX A/S**

  Home Page: http://www.ramtex.dk/

- **Timing Designer - Chronology**

  Home Page: http://www.chronology.com/

- **Total 8051 Family Development Solution ,CrossView 8051 Debugger, 8051 Assembler - BSO/TASKING**

  Home Page: http://ww.tasking.com/

- **UMPS, the universal microprocessor simulator/assembler/debugger - Virtual Micro Design**

  Home Page: http://idls.izarbel.tm.fr/entp/techer/index.htm

- **VisualSTATE - IAR Systems**

  Home Page: www.iar.com

ANEXO III – Companhias relacionadas a microcontroladores

- **Actum Solutions**
  Home Page: www.actum.com

- **Advanced Educational Systems - AES**
  Home Page: http://www.aesmicro.com/

- **Advanced RISC Machines**
  Home Page: www.arm.com

- **Aisys Intelligent Systems**
  Home Page: http://www.aisysinc.com/

- **Analog Devices Inc.**
  Home Page: www.analog.com

- **Archimedes Software, Inc.**
  Home Page: http://www.archimedesinc.com/

- **ARC Cores**
  Home Page: www.arccores.com

- **Ashling Microsystems**
  Home Page: http://www.ashling.com/

- **Atmel**
  Home Page: http://www.atmel.com

- **Avocet Systems, Inc.**

  Home Page: http://www.avocetsystems.com

- **AWC Electronics**

  Home Page: www.al-williams.com

- **BASIC MICRO**

  Home Page: www.basicmicro.com

- **CEIBO**

  Home Page: http://www.ceibo.com/

- **ChipTools Inc.**

  Home Page: http://www.chiptools.com

- **CMX Company**

  Home Page: http://www.cmx.com

- **Crossware Products**

  Home Page: http://www.crossware.com/

- **Cygnal Integrated Products**

  Home Page: www.cygnal.com

- **Cypress MicroSystems**

  Home Page: www.cypressmicro.com

- **Dallas Semiconductor**

  Home Page: http://www.dalsemi.com/

- **DemoBoard-51**

  Home Page: http://www.wirehub.nl/lumino/demoboard-51-eng.html

- **Embedded System Products, Inc.**

  Home Page: http://www.esphou.com

- **EMICROS**

  Home Page: http://www.emicros.com/

- **Enea OSE Systems**

  Home Page: http://www.enea.se/ose/

- **Flight Electronics International Ltd.**

  Home Page: http://www.demon.co.uk/flight/

- **Forth, Inc.**

  Home Page: http://www.forth.com

- **Franklin Software**

  Home Page: http://www.fsinc.com/

- **Green Hills Software Inc.**

  Home Page: www.ghs.com

- **Hannover Software**

  Home Page: www.mindspring.com/~tavve

- **HITACHI**

  Home Page: www.hitachi.com

- **Hi-Tech Software**

  Home Page: http://www.hitech.com.au/

- **Hitex Development Tools**

  Home Page: http://www.hitex.com

- **HP Infotech**

  Home Page: www.hpinfotech.ro

- **Huntsville Microsystems Inc. - HMI**

  Home Page: http://www.hmi.com

- **IAR Systems**

  Home Page: http://www.iar.se/

- **IBM Microelectronics**

  Home Page: www.chips.ibm.com

- **Infineon Technologies**

  Home Page: www.infineon.com

- **Intel Corporation**

  Home Page**:** http://www.intel.com/

- **Introl Corporation**

  Home Page: www.introl.com

- **ISSI – Integrated Silicon Solution, Inc.**

  Home Page: http://www.issiusa.com/

- **Kanda Systems Ltd.**

  Home Page: http://www.kanda-systems.com

- **Keil Software Inc.**

  Home Page: http://www.keil.com

- **Lakeview Research**

  Home Page: http://www.lvr.com/

- **Lauterbach**

  Home Page: www.lauterbach.com

- **MCC - Micro Computer Control Corp.**

  Home Page: http://www.mcc-us.com

- **MDL Labs**

  Home Page: http://www.mdllabs.com/

- **Metalink**

  Home Page: http://www.metaice.com/

- **Microchip**

  Home Page: http://www.microchip2.com

- **Mitsubishi Semiconductors**

  Home Page: www.mitsubishichips.com

- **Motorola**

  Home Page: www.motorola.com

- **National Semiconductor**

  Home Page: www.national.com

- **NetMedia Inc.**

  Home Page: www.basicx.com

- **Nohau Corporation**

  Home Page: http://www.nohau.com/

- **OKI Semiconductor, Inc.**

  Home Page: http://www.okisemi.com/

- **Orion Instruments, Inc.**

  Home Page: http://www.oritools.com

- **Parallax Inc.**

  Home Page: www.parallaxinc.com

- **Philips Semiconductor**

  Home Page: http://www.semiconductors.philips.com

- **Phyton Inc.**

  Home Page: www.phyton.com

- **PLC - Production Languages Corp.**

  Home Page: http://www.plcorp.com/

- **PLS Development Tools**

  Home Page: www.pls-mc.com

- **PowerBASIC Inc.**

  Home Page: www.powerbasic.com

- **PseudoCorp**

  Home Page: http://www.teleport.com/~rhowden

- **Rabbit Semiconductor**

  Home Page: www.rabbitsemiconductor.com

- **Ramtex A/S**

  Home Page: http://www.ramtex.dk

- **Raisonance s.a.**

  Home Page: www.raisonance.com

- **Rigel Corporation**

  Home Page: www.rigelcorp.com

- **Savage Innovations**

  Home Page: www.oopic.com

- **Scenix Semiconductor**

  Home Page: www.scenix.com

- **Siemens Components, Inc.**

  Home Page: http://www.sci.siemens.com/

- **Signum Systems**

  Home Page: http://www.signum.com/

- **Silicon storage technologies**

  Home Page: www.superflash.com

- **Softools**

  Home Page: http://www.softools.com/

- **Software Science**

  Home Page: http://www.softwarescience.com

- **Soluctions Cubed**

  Home Page: www.soluctions-cubed.com

- **SPJ Systems**

  Home Page: http://www.prime-digest.w1.com/spj

- **ST Microelectronics**

  Home Page: www.st.com

- **Syndesis Ltd.**

  Home Page: http://www.flde.com/

- **Systronix Inc.**

  Home Page: http://www.systronix.com/

- **Tasking**

  Home Page: http://www.tasking

- **TAVVE Software Co.**

  Home Page: http://www.mindspring.com/~tavve/8051.html

- **TEXAS Instruments**

  Home Page: www.ti.com

- **TEMIC IC Page**

  Home Page: http://www.temic.de/semi/

- **Tribal Microsystems, Inc.**

  Home Page: http://www.tribalmicro.com

- **Triscend Corporation**

  Home Page: www.triscend.com

- **U S Software**

  Home Page: http://www.ussw.com

- **Virtual Micro Design**

  Home Page: http://idls.izarbel.tm.fr/entp/techer

- **Willert Software Tools GmbH**

  Home Page: www.willert.de

- **ZILOG**

  Home Page: www.zilog.com

- **Z-World Engineering**

  Home Page: http://www.zworld.com/