UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

AMBIENTE DE DESENVOLVIMENTO E DEPURAÇÃO PARA MICROCONTROLADOR DSPIC30F4011

RAFAEL ANGELO GARDINI

BLUMENAU
2011

2011/2-22

RAFAEL ANGELO GARDINI

AMBIENTE DE DESENVOLVIMENTO E DEPURAÇÃO PARA MICROCONTROLADOR DSPIC30F4011

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Antonio Carlos Tavares, Mestre - Orientador

BLUMENAU
2011

2011/2-22

AMBIENTE DE DESENVOLVIMENTO E DEPURAÇÃO PARA MICROCONTROLADOR DSPIC30F4011

Por

RAFAEL ANGELO GARDINI

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Antonio Carlos Tavares, Mestre – Orientador, FURB
Membro:	Prof. Miguel Alexandre Wisintainer, Mestre – FURB
Membro:	Prof. Francisco Adell Péricas, Mestre – FURB

Blumenau, 12 de dezembro de 2011

Dedico este trabalho aos membros de minha família, aos amigos e a todos os professores da Fundação Universidade Regional de Blumenau.

AGRADECIMENTOS

A Deus, pela sua proteção divina.

À minha família e minha namorada, pela paciência e apoio.

Ao meu orientador, Antonio Carlos Tavares, pelo conhecimento passado, pelas idéias compartilhadas, pelo apoio dado e pela dedicação empregada.

O degrau de uma escada não serve simplesmente para que alguém permaneça em cima dele, destina-se a sustentar o pé de um homem pelo tempo suficiente para que ele coloque o outro um pouco mais alto.

Thomas Huxley

RESUMO

O presente trabalho descreve uma ferramenta de desenvolvimento e depuração para o microcontrolador DSPIC30F4011 chamada ADDM30F. Esta ferramenta é responsável pela criação de softwares embarcados através de uma linguagem de programação concebida com o auxílio de técnicas de análise léxica, sintática e semântica, que, ao final do processo de desenvolvimento, será traduzida para código intermediário em linguagem C e será compilado e gravado no microcontrolador. A ferramenta é orientada a componentes e utiliza *drag and drop*, onde os mesmos podem ser arrastados para um editor gráfico e manipulados por um editor de propriedades. A depuração criada para a ferramenta demonstra em tempo real a execução do software embarcado, obtendo resultados visuais nos componentes criados no editor, que aperfeiçoam os testes e que isentam o programador do software embarcado de realizar a gravação física no microcontrolador para visualização de resultados, prevenindo-o de falhas.

Palavras-chave: Microcontrolador. Dspic. Depuração. Sistemas embarcados. Sistemas operacionais.

ABSTRACT

This paper describes a tool for development and microcontroller dsPIC30F4011 debugging called ADDM30F. This tool is responsible for creating embedded software using a programming language designed with the help of techniques of lexical analysis, syntactic and semantic, that at the end of the development process will be translated to intermediate code in C language and will be compiled and written to the microcontroller. The tool is component-oriented and uses drag and drop where they may be dragged into a graphics editor and manipulated by a property editor. The debugging tool designed to show real-time implementation of embedded software, achieving visual components created in the editor, which enhance testing and exempting the developer of embedded software to perform the physical write in the microcontroller for displaying results, preventing it faults.

Key-words: Microcontroller. Dspic. Debug. Embedded systems. Operating systems.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

AD – Analógico Digital

API - *Application Programming Interface*

BNF – Forma Normal de Backus

CAN – *Controller Area Network*

CI – Circuito Integrado

CPU – *Central Processing Unit*

DSPIC – *Digital Signal Programmable Interrupt Controller*

EEPROM - *Electrically-Erasable Programmable Read-Only Memory*

FIFO – *First In First Out*

GALS - Gerador de Analisadores Léxico e Sintático

GCC - GNU *Compiler Collection*

I2C - *Inter-Integrated Circuit*

IDE – *Integrated Development Environment*

LCD - *Liquid Crystal Display*

LED - *Light-emitting diode*

LPT - *Line Print Terminal*

MCU – *Micro-Controller Unit*

MHZ – Mega Hertz

PDA - *Personal Digital Assistant*

PIC - *Programmable Interrupt Controller*

PWM - *Pulse-Width Modulation*

RAD - *Rapid Application Development*

RAM - *Random Access Memory*

RF – Requisitos Funcionais

RISC - *Reduced instruction set computing*

RNF – Requisitos Não Funcionais

ROM - *Read Only Memory*

RTC – *Real-Time Clock*

SPI - *Serial Peripheral Interface*

UART - *Universal Asynchronous Receiver-Transmitter*

UML - *Unified Modeling Language*

USB – *Universal Serial Bus*

SUMÁRIO

1 INTRODUÇÃO

Considerando que a automação[1] está cada vez mais presente na vida em sociedade, o desenvolvimento de novas tecnologias é uma busca contínua. Com isso, a necessidade de encontrar produtos e soluções que atendam esta demanda aumenta exponencialmente.

Uma das áreas que tem crescido vertiginosamente é a automação industrial. Criar tecnologias que possam prover mais rapidez e facilidade na obtenção dos resultados comerciais é o principal objetivo, refletido claramente através das linhas de montagem que fazem o uso aplicado de soluções de hardware e software a fim de evitar a utilização do trabalho humano de forma direta e acelerar a conclusão de trabalhos que necessitam de precisão e qualidade.

Convém salientar que, conforme Ribeiro (2001, p. 10), a automação quando aplicada, tende a reduzir a mão de obra empregada, automatizando processos manuais que isentam o esforço humano em grande parte.

Dentre os mais diversos motivos que acarretam a troca gradativa do trabalho direto de homens por soluções programáveis, destaca-se a periculosidade em certas situações, a diminuição de erros e, como já mencionado, a aceleração do processo industrial na fabricação dos produtos.

Diante do exposto, foi desenvolvida uma ferramenta para a criação de sistemas embarcados. Esta ferramenta permite o trabalho interativo com a utilização de módulos funcionais, seja de código fonte ou de bibliotecas compiladas em separado.

Tal ferramenta consiste em uma *Integrated Development Environment* (IDE) responsável pela criação de projetos visuais para o microcontrolador[2] DSPIC30F4011 (MICROCHIP, 2005), plataforma utilizada no desenvolvimento de sistemas embarcados. O principal foco desta ferramenta, além de aplicar o método de desenvolvimento *Rapid Application Development* (RAD), é facilitar a geração de código para o dispositivo de forma transparente, poupando o desenvolvedor de se preocupar com detalhes específicos como parâmetros de compilação[3], gravação do código executável e de inclusão das bibliotecas internas de desenvolvimento.

---

[1] Ribeiro (2001, p. 10) define automação como “a substituição do trabalho humano ou animal por máquina”.
[2] Silva (2006, p. 27) define microcontrolador como “Um circuito integrado programável que contém todos os componentes de um computador”.

Aliada a estas funcionalidades, uma área de depuração foi criada para que se tenha um ambiente seguro de testes e que previnam riscos na utilização da própria plataforma de produção.

1.1 OBJETIVOS DO TRABALHO

Este trabalho tem como objetivo desenvolver uma IDE para programação e geração de sistemas embarcados[4] para o microcontrolador DSPIC30F4011.

Os objetivos específicos deste trabalho são:

a) disponibilizar um ambiente de desenvolvimento RAD;
b) prover uma paleta de componentes visuais para acesso às interfaces de entrada e saída do microcontrolador;
c) desenvolver analisadores léxico, sintático e semântico para a linguagem de programação da ferramenta;
d) gerar código intermediário na linguagem C;
e) possibilitar a gravação de programas no microcontrolador;
f) disponibilizar um módulo de depuração.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está organizado em quatro capítulos. O segundo capítulo contém a fundamentação teórica que contempla os itens necessários para o entendimento da solução do problema proposto, que estão englobadas as metodologias de desenvolvimento RAD, a geração de código que aborda os analisadores léxico, sintático e semântico e a ferramenta responsável por auxiliar na criação das regras na Forma Normal de Backus (BNF) da linguagem de programação, o desenvolvimento para microcontroladores onde estão detalhados o microcontrolador do presente estudo, os componentes auxiliares do

[3] Aho, Sethi e Ullman (1995, p. 1) definem compilação como o ato de ler um programa escrito em uma linguagem, que é denominada linguagem fonte, e realizar a tradução para uma linguagem equivalente denominada linguagem alvo.

microcontrolador, as ferramentas de compilação, gravação e o ambiente de testes utilizado no presente estudo e por fim os trabalhos correlatos.

O terceiro capítulo trata dos requisitos principais do problema a ser trabalhado. É apresentada a especificação do problema através do diagrama de casos de uso, classes, e de sequência, apresentação da especificação das regras BNF da linguagem de programação. A explanação sobre o desenvolvimento da ferramenta conectando as técnicas de programação utilizadas e as estruturas de código desenvolvidas para atingir tal objetivo, e por fim os resultados e discussão.

O último capítulo apresenta as conclusões do presente trabalho e as sugestões de trabalhos futuros utilizando tal ferramenta.

[4] Sistemas embarcados e sistemas embutidos são tratados como sinônimos no decorrer do texto.

2 FUNDAMENTAÇÃO TEÓRICA

No presente capítulo são apresentados temas como a metodologia de desenvolvimento RAD, que descreve as técnicas para uma programação rápida. Os geradores de código responsáveis por conceber a linguagem de programação criada para a ferramenta e o software Gerador de Analisadores Léxicos e Sintáticos (GALS) (GESSER, 2003). O desenvolvimento para microcontroladores que detalha as técnicas e métodos de desenvolvimento bem como as interfaces de entrada e saída do microcontrolador que estão englobados na ferramenta do presente estudo como componentes. Os softwares de compilação Mikroc (MIKROELEKTRONIKA, 1998) e gravação Winpic800 (FONT, 2010). É descrito também alguns comentários sobre o ambiente de testes Kit Dspicgenios (MICROGENIOS, 2009) utilizado para gravação de programas e visualização dos resultados e por fim são apresentados os trabalhos correlatos concernentes ao tema.

2.1 METODOLOGIA DE DESENVOLVIMENTO RAD

A metodologia de desenvolvimento RAD é uma técnica utilizada pelas IDEs com o propósito de obter maior produtividade na construção de aplicações, nas quais o programador pode concentrar suas atenções para as regras de negócio do processo (PISKE; SEIDEL, 2006).

> Sua principal característica é que o produto de software seja desenvolvido em componentes, pois a reutilização de código permite que a equipe de desenvolvimento possa desenvolver um sistema completamente funcional em pouco tempo. (SOUZA NETO, 2004, p. 13).

A produção de sistemas utilizando a tecnologia supracitada destaca-se pela redução de custos de desenvolvimento, que, atualmente, está diretamente ligada à redução de tempo na fase de criação. Uma produção mais ágil possibilita a realização de mais testes, que acarretam a redução de erros.

Ainda, segundo Souza Neto (2004, p. 13), outro importante aspecto que distingue a metodologia de desenvolvimento RAD é sua modularização. Quanto a isso, assevera-se que os requisitos do projeto deverão ser minuciosamente definidos e com grau acentuado de independência entre eles.

Por fim, as vantagens descritas para a metodologia em contexto somente são possíveis tendo em vista a alta interatividade que o modo de desenvolvimento RAD proporciona através da reutilização de componente.

2.2 GERADORES DE CÓDIGO

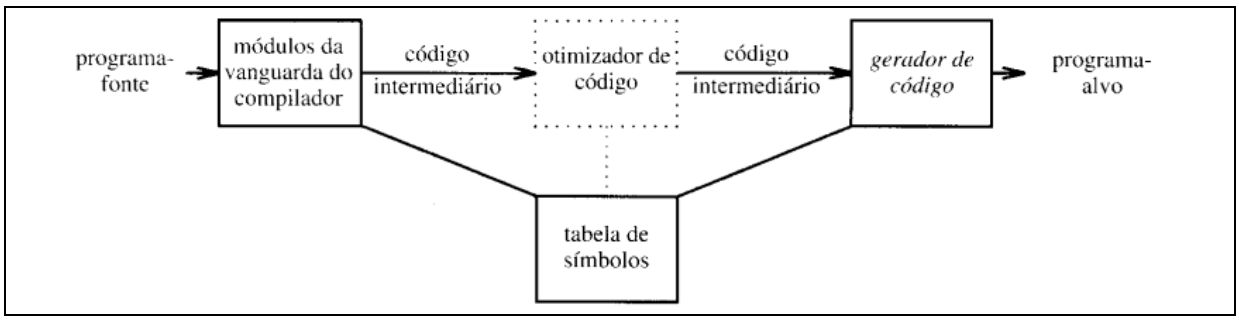
A geração de código é uma técnica aplicada na área de compiladores, com a finalidade de solucionar questões de tradução de código para uma máquina alvo. Assim, uma entrada com o código proposto é produzida, obtendo-se uma saída esperada, consoante com a plataforma a que se destina.

> As exigências tradicionalmente impostas a um gerador de código são severas. O código de saída precisa ser correto e de alta qualidade, significando que o mesmo deve tornar efetivo o uso dos recursos da máquina-alvo. Sobretudo, o próprio gerador de código deve rodar eficientemente. (AHO; SETHI; ULLMAN, 1995, p. 222).

Segundo Price e Toscani (2001, p. 11), a geração do código objeto final por meio do código fonte proposto, utilizando-se de técnicas aliadas, principalmente, ao analisador sintático, possibilitam a criação de um código mais eficiente e rápido, podendo, inclusive, facilitar na transformação do código.

Ainda, Aho, Sethi e Ullman (1995, p. 224) salientam que é primordial que se tenha conhecimento aprofundado da máquina-alvo para obter um bom código final gerado, porém, devida à grande complexidade que esta tarefa exige, torna-se difícil a produção de um código abrangente ao ponto de atender a todos os recursos que a máquina-alvo oferece.

A figura 1 mostra o processo de geração de código.



Fonte: Aho, Sethi e Ullman (1995, p. 222).

Figura 1 - Processo de geração de código

Para a criação de uma solução de geração de código condizente com as exigências atuais, é necessária a aplicação de técnicas de compiladores que auxiliam na produção de um

bom gerador de código. Estas técnicas são descritas nos próximos itens do presente capitulo.

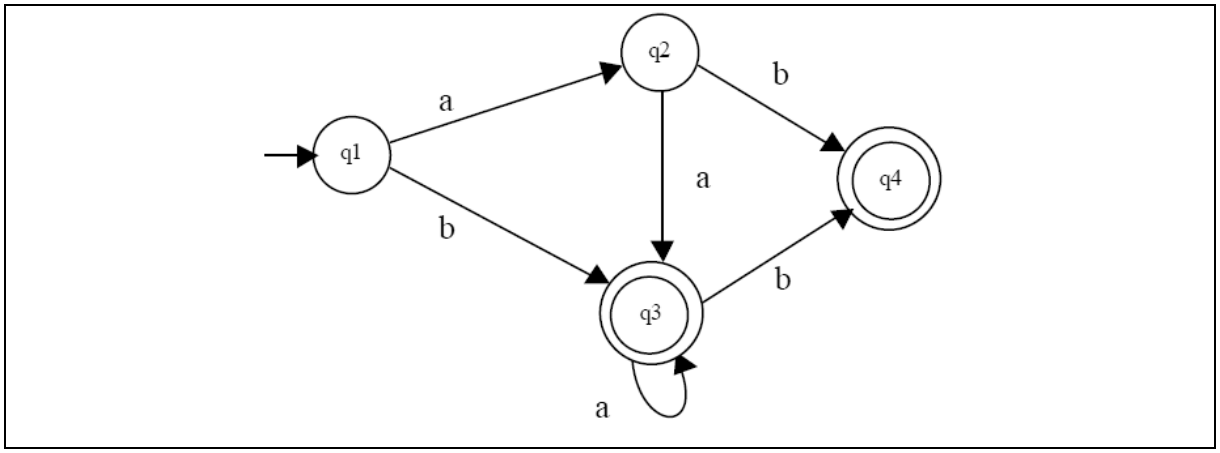
2.2.1 Analisador léxico

O analisador léxico também conhecido como *scanner* é um dos módulos que compõem um compilador ou interpretador responsável por identificar e armazenar os símbolos presentes na linguagem de programação, estes símbolos são popularmente conhecidos como *tokens*, contidos em um alfabeto que podem ser de palavras reservadas, identificadores, símbolos da linguagem definida, espaços em branco, quebras de linha ou símbolos tabulares, que normalmente serão ignorados no processo léxico e passados ao analisador sintático ao final do processo.

> Algumas vezes, os analisadores léxicos são divididos em duas fases em cascata, a primeira chamada de "varredura" (*scanning*) e a segunda de “análise léxica". O scanner é responsável por realizar tarefas simples, enquanto o analisador léxico propriamente dito realiza as tarefas mais complexas. (AHO; SETHI; ULLMAN, 1995, p. 38).

Ainda segundo Aho, Sethi e Ullman (1995, p. 38), com a aplicação de um analisador léxico em separado, a eficiência do compilador, interpretador ou gerador de código é potencializada, pois isentam as etapas seguintes de realizar o tratamento de símbolos restritos ao analisador léxico, excluindo palavras especiais como caracteres em branco, caracteres tabulares ou quebras de linha, e contando também com a alta portabilidade do código pertencente ao analisador, uma vez que a modularização específica torna a aplicação do analisador léxico viável em várias linguagens.

A figura 2 representa o autômato finito utilizado pelo analisador léxico no processo de reconhecimento da palavra.



Fonte: Gesser (2003, p. 14).

Figura 2 - Autômato finito para reconhecimento da palavra

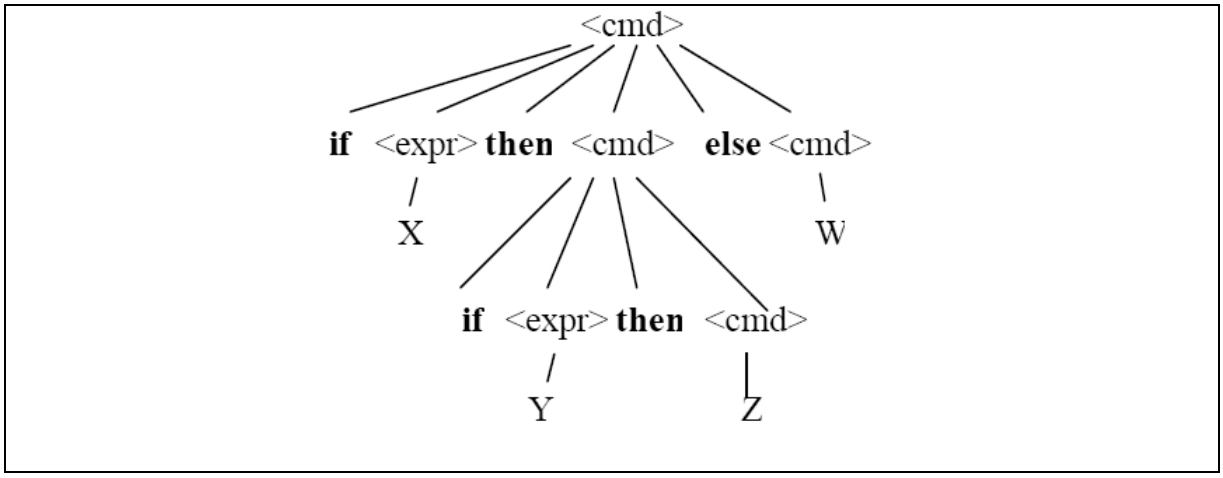
2.2.2 Analisador sintático

Analisadores sintáticos fazem parte da estrutura interna de um compilador, também conhecidos como analisadores gramaticais. Sua principal função é atuar diretamente em conjunto com o analisador léxico, realizando a junção dos *tokens* da linguagem especificada para análise e construção de uma árvore de *tokens* necessária para reconhecimento da palavra, através da análise de símbolos pertencentes às regras gramaticais.

> O Analisador Sintático identifica sequências de símbolos que constituem estruturas sintáticas (por exemplo. expressões, comandos), através de uma varredura ou "parsing" da representação interna (cadeia de tokens) do programa fonte. O Analisador Sintático produz (explícita ou implicitamente) uma estrutura em árvore, chamada árvore de derivação**,** que exibe a estrutura sintática do texto fonte, resultante da aplicação das regras gramaticais da linguagem. (PRICE; TOSCANI, 2001, p. 9).

Ainda, segundo Price e Toscani (2001, p. 9), os analisadores sintáticos têm por função adicional e primordial a detecção de erros de compilação referentes à gramática analisada, e devem ser capazes de apontar e identificar claramente a exceção quanto ao tipo, gênero e localização. Outra característica que valoriza o desenvolvimento de um analisador sintático é capacidade de apontar os erros após análise completa da gramática, sem interrupções, ao longo do processo, tornando mais rápida a correção dos erros e impedindo na maioria das vezes a repetição do processo de compilação.

A figura 3 mostra a estrutura da árvore de derivação utilizada pelo analisador sintático.



Fonte: Gesser (2003, p. 21).

Figura 3 – Estrutura da árvore de derivação

2.2.3 Analisador semântico

Analisadores semânticos pertencem ao terceiro e último módulo essencial de um compilador e têm por objetivo principal analisar e detectar erros de consistência na programação empregada pela linguagem alvo. Este módulo utiliza como informação base as tabelas geradas pelos dois outros módulos auxiliares no processo de compilação, que são os analisadores léxicos e sintáticos. Em linguagens de programação com utilização forte de tipos *strong typin,*onde a declaração explícita do tipo de variável é obrigatória, a utilização de um analisador semântico é importante, tendo em vista a grande possibilidade de ocorrência de erros na construção do programa fonte, como, por exemplo, erros de conversão de tipos e problemas de atribuição de valores em variáveis não compatíveis ou não definidas.

> Ações semânticas são associadas às regras de produção da gramática de modo que, quando uma dada produção é processada (por derivação ou redução de uma forma sentencial no processo de reconhecimento), essas ações são executadas. A execução dessas ações pode gerar ou interpretar código, armazenar informações na tabela de símbolos, emitir mensagens de erro, etc. (PRICE; TOSCANI, 2001, p. 84).

Ações semânticas associadas às regras de produção dão certa flexibilidade e maior entendimento das regras semânticas, segundo Price e Toscani (2001, p. 84), pois é possível associar variáveis aos símbolos terminais ou não terminais da gramática. Sendo assim, os símbolos gramaticais podem ser identificados mais facilmente durante o processo de reconhecimento da regra semântica, podendo gerar mensagens de erro na validação de consistência e realizar o reconhecimento de tipos de variáveis.

> Os aspectos semânticos não são facilmente especificáveis. Os principais mecanismos formais para a especificação dos aspectos semânticos são as gramáticas de atributos, as semânticas denotacionais e as semânticas de ações. Porém sua complexidade os torna inviáveis na prática, e acaba-se partindo para mecanismos semiformais, ou até mesmo informais. (GESSER, 2003, p. 11).

Diante do exposto, finaliza-se a explanação sobre os módulos essenciais de um compilador ou interpretador, analisando-se o próximo item, que aborda a ferramenta auxiliar responsável pela geração de analisadores léxicos e sintáticos.

2.2.4 GALS

GALS - Gerador de Analisadores Léxicos e Sintáticos - é uma ferramenta de livre utilização bastante útil no meio acadêmico, pois tem como principal enfoque a introdução dos acadêmicos da área de computação nas técnicas de desenvolvimento de compiladores,

trazendo um ambiente específico, com foco de aproximar as aulas teóricas com as práticas, tornando o aprendizado mais dinâmico ante a produção de resultados reais.

O GALS, em sua concepção, tem por finalidade a geração de código fonte para dois importantes módulos de um compilador ou interpretador de código, como os analisadores léxicos e sintáticos, explanados nos itens anteriores. O GALS traz, como auxílio extra, a criação de símbolos identificadores introduzidos nas regras de produção que auxiliam o programador na criação de analisadores semânticos. A produção dos analisadores se dá através da especificação das regras de produção através da BNF, auxiliando o utilizador na criação, correção e geração das regras de produção. Segundo Gesser (2003, p. 5), GALS pode ser considerada uma ferramenta prática e versátil, e seu uso se dá tanto no meio acadêmico como profissional, promovendo o uso das técnicas formais de análise.

A figura 4 mostra a interface gráfica da ferramenta GALS.

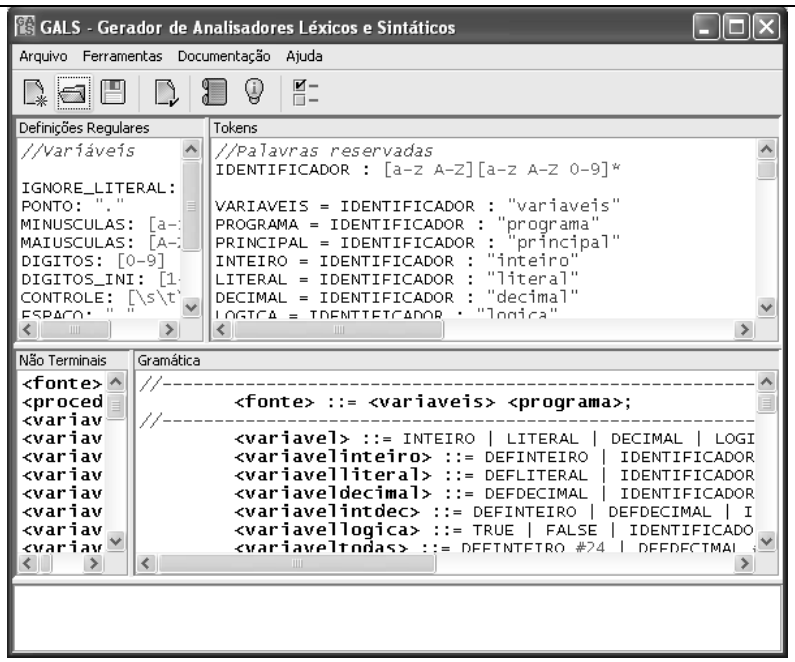


Figura 4 – Interface gráfica GALS

2.3 DESENVOLVIMENTO PARA MICROCONTROLADORES

As linguagens de programação existentes realizam um grande trabalho no desenvolvimento de soluções embarcadas, já que reduzem o tempo de trabalho despendido no projeto dessas mesmas soluções, ou seja, um projeto que não aplica a utilização de

microcontroladores tem seu tempo de criação e de projeção acentuados, por não contarem com a facilidade de desenvolvimento que um microcontrolador traria, reduzindo a complexidade e, conseqüentemente, o tamanho da solução.

Programação pode ser definida como o ato de "[...] fornecer uma sequência determinada de comandos ou instruções para ser executada por uma pessoa ou máquina" (PEREIRA, 2002, p. 18).

Por sua vez, despontam os microcontroladores.

> Os microcontroladores, ao contrário de seus irmãos microprocessadores, são dispositivos mais simples com memórias RAM e ROM internas, oscilador interno de clock, I/O interno, entre outros, sendo por isso chamados muitas vezes de computadores em um único chip. Tais características tornam mais simples o projeto de dispositivos inteligentes, pois MCU´s raramente necessitam de CI´s externos para funcionar, o que contribui para diminuição de custos e tamanho. (PEREIRA, 2002, p. 18).

A criação de soluções microprocessadas ou programáveis ganham espaço de forma gradativa no mercado, um cenário cada vez mais abrangente. Naturalmente, projetos menores que possibilitem diminuição de custos e manutenção possuem uma maior demanda, salienta Pereira (2002, p. 18-19).

Neste ínterim, Matic e Andric (2000, p. 13) destacam que o desenvolvimento de soluções para microcontroladores pode ser executado utilizando as seguintes linguagens de programação:

a) linguagem *Assembly*: linguagem de baixo nível que possui uma curva de aprendizado mais acentuada, requerendo maior domínio do programador. Com isso, conta com a vantagem de ser mais objetiva e, conseqüentemente, mais rápida, economizando em tamanho de programa e em recursos de memória;
b) linguagem C: possui uma curva de aprendizado mais curta com um nível maior de abstração, podendo produzir soluções mais rápidas. Assim sendo, acarreta a perda de desempenho, se comparada com *assembly,* por possuir geração de código de forma intermediária;
c) linguagem *Basic*: a linguagem mais fácil entre as duas anteriores, tendo como característica, assim como a linguagem C, ser mais lenta do que o a*ssembly*, devido ao alto grau de abstração.

Ainda, Matic e Andric (2000, p. 13) salientam que a escolha entre as alternativas citadas é importante no processo de desenvolvimento. Isso porque, deve-se examinar com cuidado as opções, atentando-se ao projeto que se pretende atender e questões como velocidade, complexidade e tamanho de programa.

Cada linguagem de programação possui aspectos singulares que influirão no *set* de instruções da arquitetura de microcontroladores, que são utilizadas na programação direta ou como suporte aos geradores de código.

> Chamamos de Set de instruções o conjunto de instruções que comandam o microcontrolador. Estes comandos já vêm gravados de fábrica e ficam na memória ROM. A CPU RISC contém um número reduzido de instruções isto o torna mais fácil de aprender, pois temos menos instruções. No entanto temos que reutiliza-las mais vezes para realizar operações diferentes. (SILVA, 2006, p. 52).

Por fim, cumpre salientar que estas instruções são utilizadas de forma direta ou indireta no processo de criação de um sistema embarcado em microcontrolador, estando condicionada a escolha da linguagem de desenvolvimento, aliando-se ao modelo de microcontrolador escolhido e aos componentes acessórios de entrada e saída que são de suma importância no projeto de criação e programação para microcontroladores e que são explanados nos itens seguintes.

2.3.1 Microcontrolador DSPIC30F4011

O microcontrolador DSPIC30F4011 pertence à classe de *Micro-Controller Unit* (MCU) de dezesseis bits, da família com tecnologia digital da empresa Microchip. Segundo Microchip (2005, p. 1), o microcontrolador DSPIC30F4011 possui arquitetura de *harvard* modificada para alta performance, contendo otimização de instruções na base de endereços para compiladores C.

As principais características que compõem este MCU são descritas abaixo:

a) oitenta e quatro instruções físicas;
b) 2 KB de memória *Random Access Memory* (RAM);
c) 48 KB de memória *flash*;
d) três interrupções externas;
e) 1 KB de memória não volátil *Electrically-Erasable Programmable Read-Only Memory* (EEPROM);
f) 40 Mega Hertz (MHZ) de clock externo;
g) oscilador interno de 4 MHZ.

Ainda, segundo Microchip (2005, p. 3), é importante salientar que, para uma programação mais rápida e direta, este MCU conta com módulos auxiliares que dão plena assistência ao desenvolvedor de sistemas embarcados, pois os mesmos estão encapsulados em

um único chip. Sendo assim, não há a necessidade de desenvolvimento de circuitos externos para chegar ao mesmo objetivo que a aplicação deste mesmo MCU faz, possuindo estes módulos embutidos. Os módulos que auxiliam o desenvolvedor de sistemas embarcados que compõem este importante microcontrolador são:

a) um módulo *Controller Area Network* (CAN);
b) dois módulos *Universal Asynchronous Receiver-Transmitter* (UART) com *First In First Out* (FIFO) buffer;
c) módulo *Inter-Integrated Circuit* (I2C);
d) módulo *Serial Peripheral Interface* (SPI);
e) módulo *Pulse-Width Modulation* (PWM);
f) temporizador de dezesseis bits;
g) *prescaler* programável.

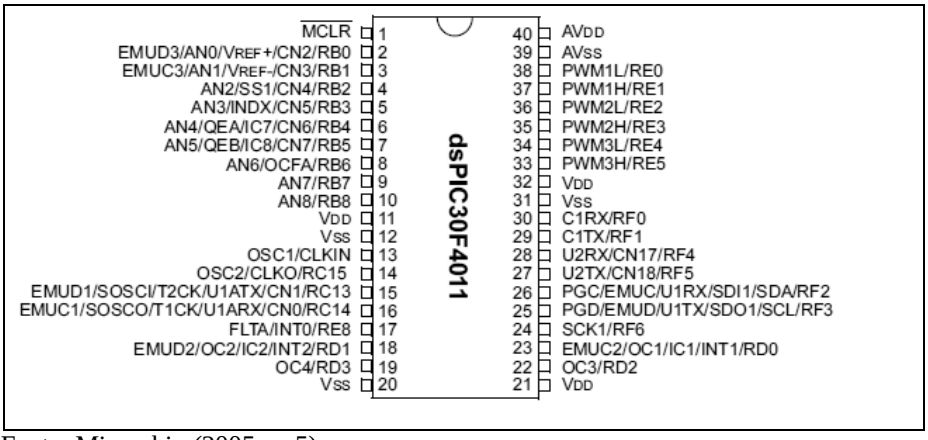
Os microcontroladores *Programmable Interrupt Controller* (PIC) e *Digital Signal Programmable Interrupt Controller* (DSPIC) possuem uma gama enorme de modelos, que cumprem funções distintas para cada aplicação que projetos de diversas características necessitam. Abaixo, é mostrado na figura 5 um comparativo de alguns modelos de microcontroladores da família digital DSPIC30F e suas especificações de hardware.

Device	Pins	Program Mem. Bytes/ Instructions	SRAM Bytes	EEPROM Bytes	Timer 16-bit	Input Cap	Output Comp/Std PWM	Moto Control PWM	A/D 10-bit 500 Ksps	Quad Enc	UART	SPI™	I²C™	CAN
dsPIC30F2010	28	12K/4K	512	1024	3	4	2	6 ch	6 ch	Yes	1	1	1	-
dsPIC30F3010	28	24K/8K	1024	1024	5	4	2	6 ch	6 ch	Yes	1	1	1	-
dsPIC30F4012	28	48K/16K	2048	1024	5	4	2	6 ch	6 ch	Yes	1	1	1	1
dsPIC30F3011	40/44	24K/8K	1024	1024	5	4	4	6 ch	9 ch	Yes	2	1	1	-
dsPIC30F4011	40/44	48K/16K	2048	1024	5	4	4	6 ch	9 ch	Yes	2	1	1	1
dsPIC30F5015	64	66K/22K	2048	1024	5	4	4	8 ch	16 ch	Yes	1	2	1	1
dsPIC30F6010	80	144K/48K	8192	4096	5	8	8	8 ch	16 ch	Yes	2	2	1	2

Fonte: Microchip (2005, p. 4).

Figura 5 – Comparativa entre microcontroladores DSPIC30F

O modelo de microcontrolador aplicado no presente estudo possui uma estrutura física de 40 pinos, onde cada um recebe determinadas funções e característica específica conforme mostra a figura 6.



Fonte: Microchip (2005, p. 5).

Figura 6 – Especificação visual do microcontrolador DSPIC30F4011

Para atingir os objetivos do presente estudo, é importante salientar que um microcontrolador necessita de auxílio de componentes periféricos externos que realizam a interação com os pinos acima explanados para realizar a comunicação com o mundo exterior e cumprir os resultados que lhe são atribuídos, tomando como premissa esta afirmação, o próximo item nos demonstra alguns desses componentes e suas aplicações.

2.3.2 Componentes associados aos microcontroladores

Microcontroladores são freqüentemente utilizados em projetos de alta complexidade de desenvolvimento e que geralmente agregam funções extremante especializadas, que necessitam de auxílio de periféricos externos de entrada e saída a fim de produzir e receber resultados. Estas aplicações possuem alta aplicabilidade nos dias atuais e necessitam de um alto grau de clareza e a apresentação de informações consistentes. Diante do exposto, importantes interfaces de entrada em saída são apresentadas neste item, atendendo as necessidades impostas por esta exigente demanda.

2.3.2.1 LCD 16x2

Liquid Crystal Display (LCD) é um componente alfanumérico, de baixo custo, largamente utilizado em projetos que fazem o uso de microcontroladores. Sua utilidade é acentuada onde o objetivo é apresentar informações ao mundo exterior. Segundo Microgenius (2009, p.8), o componente LCD possui dezesseis colunas por duas linhas e trabalha em modo

quatro bits. Conta também com memória interna o que facilita e muito no desenvolvimento de soluções embarcadas, tendo em vista a necessidade de manter as informações de forma constante.

A fim de esclarecer o modo de operação com microcontrolador, a figura 7 mostra o esquema de ligação de um componente LCD.

Pino	Descrição
nc	DATA0 do display
nc	DATA1 do display
nc	DATA2 do display
nc	DATA3 do display
RB0	DATA4 do display
RB1	DATA5 do display
RB2	DATA6 do display
RB3	DATA7 do display
RE5	Pulso de ENABLE (EN)
RE4	Pulso de comando (RS)
GND	Pulso de escrita /leitura (R/W)

Fonte: Microgenius (2009, p. 8).

Figura 7 – Estrutura de ligação do componente LCD

Para realizar a programação de um componente LCD em conjunto com um microcontrolador são necessárias algumas rotinas de programação essenciais para a operacionalização do componente, como configuração, escrita e limpeza.

O quadro 1 demonstra essas rotinas de programação utilizando a biblioteca do software Mikroc.

```
void main()
{
      // programa todas os pinos do portb como I/O de uso geral
      ADPCFG = 0xFFFF;
      //define porte como saida
      TRISE = 0;
      //define portb como saida
      TRISB = 0;

      //configura e inicializa LCD no modo 4 bits
      Lcd_Custom_Config(&PORTB, 3,2,1,0, &PORTE, 4,0,5);
      // apaga display
      Lcd_custom_Cmd(Lcd_CLEAR);
      // desliga cursor
      Lcd_custom_Cmd(Lcd_CURSOR_OFF);
      // escreve "Kit dsPICGenios" na primeira linha, primeira coluna
      Lcd_Custom_Out(1,1, "Kit dsPICGenios");
}
```

Fonte: adaptado de Microgenius (2009, p. 9).

Quadro 1 – Estrutura das rotinas de programação para LCD

A figura 8 demonstra o componente LCD propriamente dito apresentando a mensagem escrita pelo exemplo citado no quadro 1.

Figura 8 – Componente LCD 16x2

2.3.2.2 LED 7 Segmentos

Light-emitting diode (LED) de sete segmentos, ou display de sete segmentos, é um componente eletrônico criado com o objetivo de prover informações numéricas em projetos eletrônicos. Sua estrutura interna conta com sete segmentos de LED que, em conjunto, através de operações de liga e desliga, formam os algarismos numéricos. Não possui memória interna e trabalha com operações de varredura, necessitando de permanente atualização para que sua informação fique visível. Segundo Microgenius (2009, p. 14), a visibilidade dos algarismos através de varredura possui a vantagem de economizar pinos de I/O do microcontrolador e de baratear os custos de produção do circuito. Ainda, segundo Microgenius (2009, p. 14), os segmentos de LED, popularmente chamados de pinos, são identificados conceitualmente através de letras do alfabeto, facilitando o entendimento de projetos ou especificações de circuitos eletrônicos, com exceção do segmento de ponto que não recebe nenhuma nomenclatura específica.

A figura 9 mostra a representação dos pinos através de letras e, em conjunto, apresenta a estrutura de ligação do componente a um microcontrolador, descrevendo os segmentos individualmente.

Pino	Descrição
RB0	Segmento A
RB1	Segmento B
RB2	Segmento C
RB3	Segmento D
RB4	Segmento E
RB5	Segmento F
RB6	Segmento G
RB7	Segmento Ponto

Fonte: adaptado de Microgenius (2009, p. 15).

Figura 9 – Estrutura de ligação do componente LED 7 segmentos

Para realizar o desenvolvimento de soluções que utilizem os componentes de sete segmentos, o quadro 2 demonstra um pequeno exemplo capaz de esclarecer as formas de se executar a programação necessária para operar este componente utilizando a biblioteca interna do Mikroc.

```
void main()
{
      // inicialização de variaveis
      int a = 0;
      int b = 0;

      // programa todas os pinos do portb como I/O de uso geral
      ADPCFG = 0xFFFF;

      // define portd como saída
      TRISD = 0;
      PORTd = 0;

      // define portb como saída
      TRISb = 0;
      PORTf = 0;
      portd = 0xfff;
      portb = 0xfff;

      //inicio da rotina de loop
      do {
            //liga primeiro display
            PORTD.F0= 1;
            //escreve digito 6
            PORTB = 79;

            //delay de 4 milisegundos
            Delay_ms(4);
            //desliga primeiro display
            PORTD.F0= 0;
            // pausa a rotina durante 10 milisegundos
            Delay_ms(10);

      } while(1);
}
```

Fonte: adaptado de Microgenius (2009, p. 16).

Quadro 2 – Estrutura das rotinas de programação para o LED de sete segmentos

Para demonstrar as rotinas expressas no quadro 2, a figura 10 sinaliza a forma como os LEDs segmentados trabalham para realizar a amostragem do algarismo seis conforme código proposto.

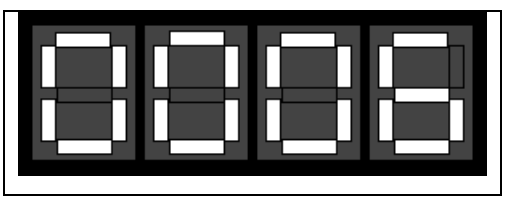


Figura 10 – Componente LED 7 segmentos

2.3.2.3 Porta serial RS232

As portas seriais padrão RS232 foram criadas em meados de 1969 objetivando interconectar os computadores e periféricos da época, e têm seu uso estendido aos dias atuais, atuando de forma importante e permanente em aplicações de uso geral. O setor de automação industrial adota este padrão em larga escala pela simplicidade de operação e pela facilidade de criação das soluções que precisam de comunicação. Em contrapartida, as tendências atuais estão abandonando este padrão de forma gradativa para uso geral e dando mais ênfase em tecnologias novas como o *Universal Serial Bus* (USB), permanecendo apenas em aplicações industriais e comerciais ou como auxílio em pesquisas acadêmicas.

Segundo Parma (2006, p. 1), a comunicação serial entre equipamentos é feita através do receptor transmissor assíncrono universal, mais conhecido como UART, que faz interface com o padrão RS232 responsável por interconectar os equipamentos entre si. Ainda, segundo Parma (2006, p. 1), para realizar a conexão entre periféricos através de comunicação serial são necessários quatro passos básicos, porém importantes, para a conclusão da conexão, são eles:

a) velocidade de comunicação;
b) *stop bits*;
c) tamanho da palavra ou *buffer* dos dados;
d) paridade.

Diante do exposto, salienta Parma (2006, p. 1) que esta configuração deve ser realizada pelo transmissor e receptor, pois por se tratar de uma transmissão e recepção assíncrona, não existe um sinal de clock responsável pelo sincronismo entre os periféricos, estando sujeitos a erros de *framing* caso esta condição não seja atendida.

A figura 11 mostra as conexões entre os pinos da porta serial e os sinais de controle RS232.

DB9	Nome	Descrição
-	GND	Terra de proteção
3	TX	Transmissão de dados
2	RX	Recepção de dados
7	RTS	Permissão para transmitir
8	CTS	Permissão para transmissão concedida
6	DSR	Modem pronto
5	GND	Terra do sinal
1	DCD	Portadora de dados detectada
4	DTR	Terminal pronto
9	RI	Indicador de chamada

Fonte: adaptado de Parma (2006, p. 2).

Figura 11 – Conexão entre os pinos da porta serial e a controladora RS232

O quadro 3 demonstra o exemplo de desenvolvimento com porta serial RS232 utilizando as bibliotecas do software Mikroc.

```
void main()
{
      // inicialização de variáveis
      char a = 0;

      //inicializa serial com baudrate 9600 bps
      Uart1_Init(9600);

      while (1)
      {
            //verifica se o buffer serial esta cheio
            if (Uart1_Data_Ready())
            {
                  // lê o buffer serial e salva valor em a
                  a = Uart1_Read_Char();

                  //envia o valor de "a' pela serial 2
                  Uart1_Write_Char(a);
            }
      }
}
```

Fonte: Adaptado dos exemplos Microgenius (2009).

Quadro 3 – Estrutura das rotinas de programação para porta serial RS232

A figura 12 demonstra a aparência do conector DB9 da porta serial RS232.

Figura 12 – Conector DB9 porta serial RS232

2.3.3 Ferramentas auxiliares no desenvolvimento

Coleciona-se a seguir os itens abordados no presente estudo como ferramentas auxiliares de desenvolvimento, Mikroc como compilador de programa fonte e Winpic800 como gravador do binário executável no microcontrolador. Ao final do item, contamos com a explicação básica sobre o ambiente de testes Kit Dspicgenios.

2.3.3.1 Mikroc

A ferramenta Mikroc disponibiliza um ambiente gráfico para desenvolvimento na plataforma PIC e dsPIC e contém diversas funcionalidades essenciais em um projeto que contenha microcontroladores. Contém uma biblioteca de desenvolvimento extensa e disponibiliza suporte a uma gama grande de microcontroladores modelo PIC e dsPIC.

Segundo Mikroelektronika (1998), o ambiente de desenvolvimento Mikroc é uma poderosa ferramenta que possui características ricas, que fornecem ao programador uma solução extremamente fácil para o desenvolvimento de sistemas embarcados, sem comprometer o desempenho da solução desenvolvida e nem ocasionar a perda de controle pelo programador. Ainda, segundo Mikroelektronika (1998), o ambiente possui características que o diferenciam da maioria dos ambientes de desenvolvimento de sistemas embarcados e são elas:

a) auto-correção;
b) realce de sintaxe;
c) possui bibliotecas para aquisição de dados, operações com memória, displays, conversores e comunicação;
d) possui um módulo chamado *code Explorer,* que tem por finalidade monitorar a gama de funções e variáveis presentes em projetos;
e) compilação e montagem de código binário compatível com a maioria dos gravadores disponíveis no mercado;
f) possui módulo de depuração integrado ao software;
g) possui mapas de memória, estatísticas de código, listagem de montagem e árvores de chamada;
h) conta com uma gama vasta de códigos exemplos e documentação.

A figura 13 demonstra algumas funcionalidades acima citadas.

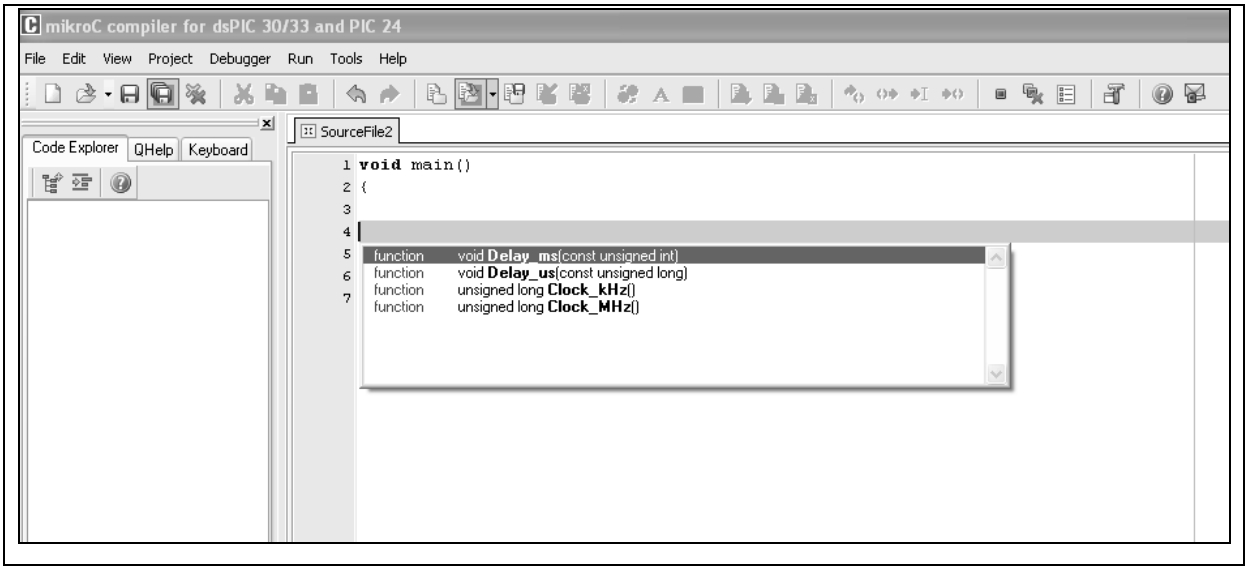


Figura 13 – Interface gráfica Mikroc

2.3.3.2 Winpic800

Winpic800 é um gravador de programas executáveis para sistemas embarcados. Sua plataforma suportada é a PIC, dsPIC e AVR (ATMEL, 2011), possuindo diversas opções de interface de saída por onde o código binário pode ser gravado como portas seriais, USB e *Line Print Terminal* (LPT). Trabalha com uma interface gráfica multilíngüe capaz de carregar o arquivo hexadecimal e que permite selecionar o microcontrolador desejado, contando também com uma tela de parametrização que seta as diretrizes a qual a gravação irá respeitar. Dentre as funcionalidades principais estão:

a) ler informação hexadecimal do microcontrolador;
b) verificação da integridade do código no microcontrolador;
c) gravação do código executável;
d) exclusão do programa do microcontrolador;
e) execução de testes de hardware;
f) parametrização externa que permite o software ser executado via linha de comando.

A figura 14 expõe a interface gráfica da ferramenta Winpic800.

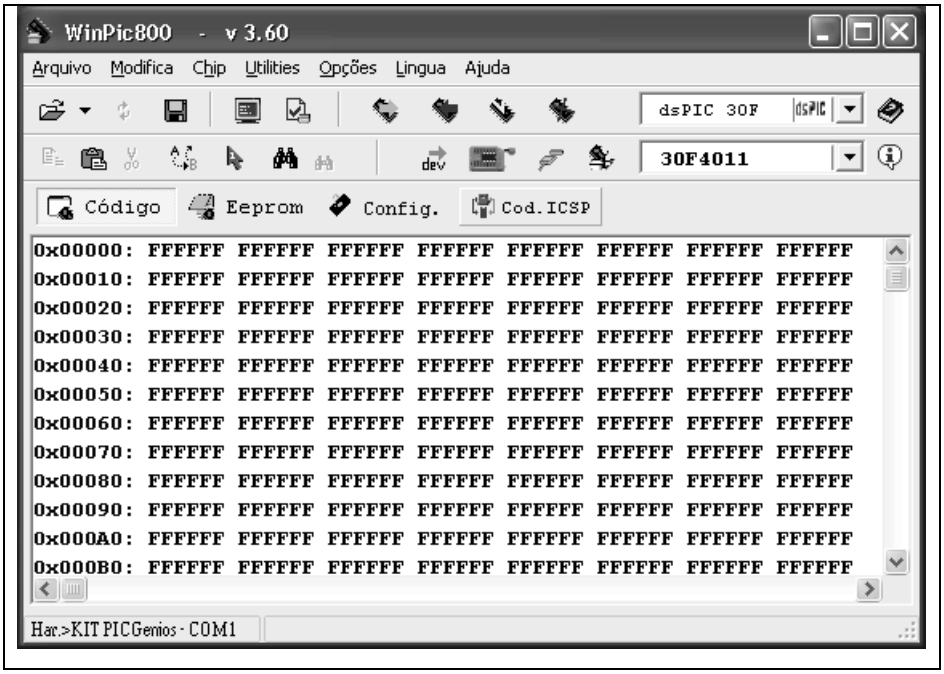


Figura 14 – Interface gráfica Winpic800

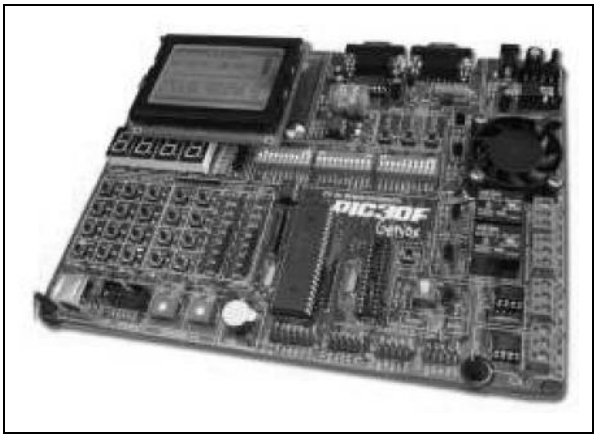
2.3.3.3 Kit Dspicgenios DSPIC30F

O Kit Dspicgenios é um módulo profissional de desenvolvimento para microcontroladores que tem por objetivo introduzir estudantes e profissionais no aprendizado das técnicas de se desenvolver sistemas embarcados utilizando microcontroladores PIC ou dsPIC. Microgenios (2009) salienta que, através da grande gama de recursos presentes no kit, o desenvolvedor pode realizar uma série de experimentos em linguagens de livre escolha que tenham suporte para microcontroladores e, através disso, manipular uma série de periféricos ligados ao microcontrolador do kit e obter os resultados práticos. Ainda, conforme Microgenios (2009), os periféricos disponíveis no kit são:

a) LCD alfanumérico 16x2 utilizados na maioria dos projetos eletrônicos;
b) LCD gráfico 128x64;
c) quatro displays de sete segmentos acionados por varredura;
d) matriz de teclado com doze teclas;
e) onze teclas de acesso direto ao pino de interrupção do microcontrolador;
f) dezesseis LEDs para controle lógico visual;
g) dois relés na/nf;
h) relógio de tempo real *Real-Time Clock* (RTC);
i) duas portas seriais RS232;

j) transceiver CAN;
k) porta PS/2;
l) sensor de temperatura LM35;
m) resistência para aquecimento com possibilidade de regulagem;
n) *buzzer* para efeito sonoro;
o) ventoinha acionada via PWM;
p) sensor infravermelho sendo possível realizar a medição da rotação da ventoinha;
q) portas de expansão que ligam o kit ao mundo externo;
r) memória serial EEPROM via canal serial I2C;
s) dois trimpots para programação de canal Analógico Digital (AD);
t) canal de comunicação RS485;
u) microcontrolador DSPIC30F4011.

O kit de desenvolvimento recebe os programas externos através da porta serial RS232 ligada a um computador que possua um programa gravador. A figura 15 demonstra o kit de desenvolvimento Dspicgenios.



Fonte: Microgenius (2009).
Figura 15 – Kit de desenvolvimento Dspicgenios

2.4 TRABALHOS CORRELATOS

Em atenção ao tema proposto, os trabalhos correlatos aqui analisados apresentam características que serão muito úteis para o desenvolvimento deste trabalho. Dentre as principais similaridades entre a ferramenta desenvolvida e estes, estão à capacidade de empregar uma interface gráfica amigável, possibilitando uma maior abstração ao

desenvolvedor, as características de desenvolvimento RAD e a geração de código para uma plataforma alvo.

Os trabalhos analisados são Flowchart Editor (FONTANIVE, 1999) e Zexus++ (ZIMERMANN, 2005).

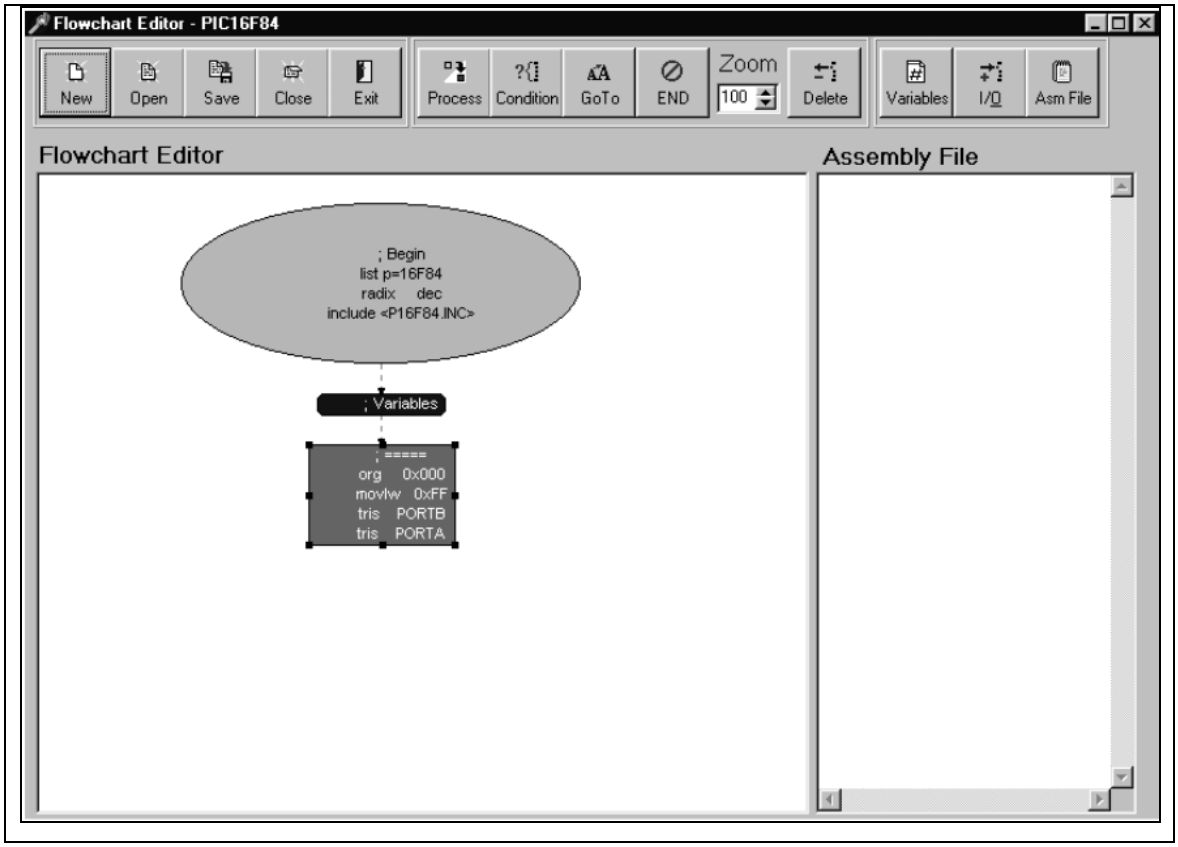
2.4.1 Flowchart editor

Segundo Fontanive (1999, p. 1), o editor fluxo programático Flowchart Editor pode ser definido como um protótipo de uma ferramenta que tem como objetivo principal prover ao usuário a capacidade de criar fluxogramas e usando-os como base, realizar a geração de código *assembly* para o microcontrolador PIC16C84 (MICROCHIP, 1997).

A ferramenta contém um editor gráfico intuitivo, possuindo formas e conexões, permitindo ao usuário desenhar uma estrutura hierárquica e lógica de forma prática. Conta também com um controle de inconsistências que é responsável por prevenir erros que possam dificultar a geração de código.

No que tange a geração de código, o fluxograma é utilizado como referência básica para a interpretação das instruções que deverão ser passadas ao microcontrolador, sendo traduzidas em um conjunto de símbolos que posteriormente são transformados em código *assembly*.

Para que se faça entender tal funcionalidade, a figura 16 mostra a interface da ferramenta.



Fonte: Fontanive (1999, p. 48).

Figura 16 – Interface gráfica Flowchart Editor

2.4.2 Zexus++

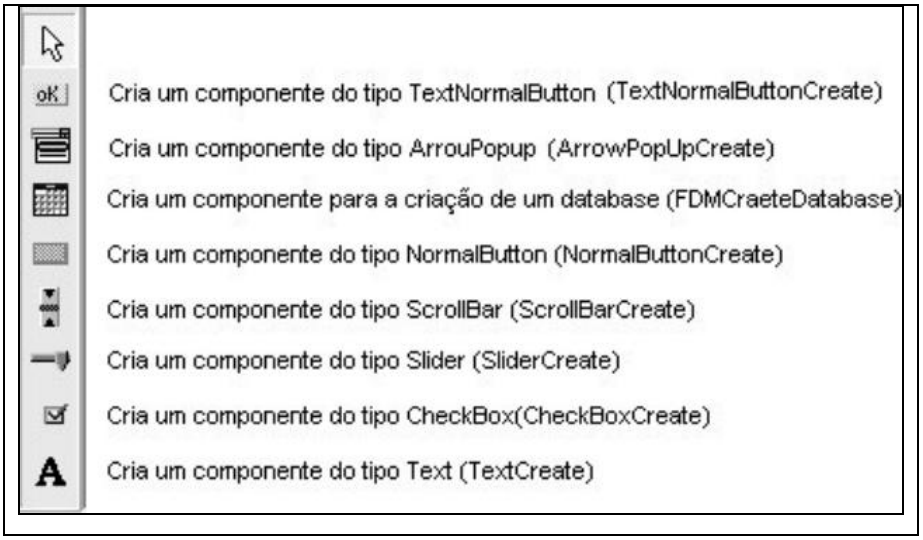
A ferramenta Zexus++ tem como principal finalidade disponibilizar um ambiente de desenvolvimento visual para a plataforma *Personal Digital Assistant* (PDA) denominada Zexus, sendo capaz de realizar a geração de código em linguagem C++ baseado no *Application Programming Interface* (API) do sistema operacional Wuji OS, em conjunto com o compilador *GNU Compiler Collection* (GCC). A base de referência usual da ferramenta para geração de código advém dos componentes que compõem a interface gráfica.

Conforme Zimermann (2005, p. 11), devido à escassez de recursos e a pouca funcionalidade prática que as ferramentas de criação existentes para tal plataforma possuem, a ferramenta Zexus++ tende a facilitar o desenvolvimento de soluções e a tornar esta tarefa mais intuitiva.

A interface gráfica que compõe a ferramenta possui uma paleta de componentes que podem ser arrastados para o que pode ser denominado como formulário do PDA, simulado

pelo editor gráfico.

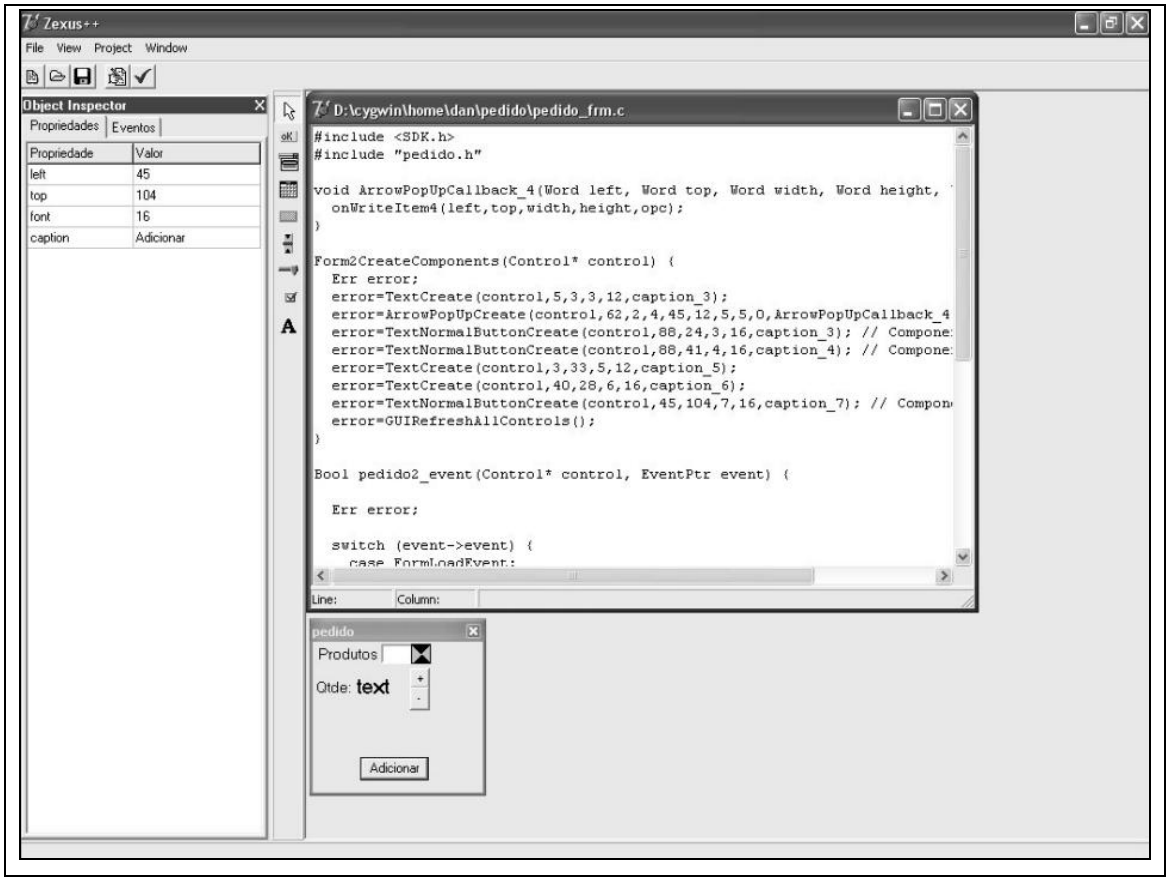
Na figura 17 pode-se visualizar a paleta de componentes supracitada e suas respectivas descrições.



Fonte: Zimermann (2005, p. 47).

Figura 17 – Paleta de componentes Zexus++

A figura 18 ilustra como a interface gráfica da ferramenta é disposta.



Fonte: Zimermann (2005, p. 43).

Figura 18 – Interface gráfica Zexus++

3 DESENVOLVIMENTO DA FERRAMENTA

Para a realização do desenvolvimento da ferramenta ADDM30F os seguintes passos foram realizados:

a) definição dos requisitos principais do problema a ser trabalhado (seção 3.1);
b) especificação do diagrama de casos de uso (seção 3.2.1);
c) especificação do diagrama de classes (seção 3.2.2);
d) especificação do diagrama de sequência (seção 3.2.3);
e) especificação da forma normal de backup BNF (seção 3.2.4);
f) implementação da criação de projetos embarcados (seção 3.3.1.1);
g) implementação do editor visual com *drag and drop* (seção 3.3.1.2);
h) implementação da paleta de componentes (seção 3.3.1.3);
i) implementação dos analisadores léxico, sintático e semântico (seção 3.3.1.4);
j) implementação da interpretação e tradução da linguagem (seção 3.3.1.5);
k) implementação da compilação da linguagem criada (seção 3.3.1.6);
l) implementação da gravação do código da linguagem compilada (seção 3.3.1.7);
m) implementação do módulo de depuração (seção 3.3.1.8).

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os Requisitos Funcionais (RF) e Não Funcionais (RNF) da ferramenta são:

a) permitir a criação de projetos de implementação para microcontroladores, podendo selecionar os parâmetros e modelos possíveis (RF);
b) possuir um editor visual que possibilite a técnica *drag and drop*[5] (RF);
c) possuir uma paleta de componentes contendo as interfaces de entrada e saída de um microcontrolador (RF);
d) possuir uma linguagem de programação intuitiva e objetiva através da construção de analisadores léxicos, sintáticos e semânticos e possibilitar a interação da mesma com os componentes visuais (RF);

[5] *Drag and drop*, em sua tradução literal, significa arraste e solte.

e) interpretar a linguagem criada previamente e traduzir o código para a linguagem C (RF);
f) compilar o código gerado através da ferramenta Mikroc (RF);
g) gravar o código no microcontrolador através da ferramenta Winpic800 (RF);
h) possuir um módulo de depuração (RF);
i) ser desenvolvido no ambiente Visual Studio 2008 Express utilizando a linguagem C# (RNF);
j) ser compatível com o sistema operacional Windows (RNF).

3.2 ESPECIFICAÇÃO

A especificação da ferramenta ADDM30F foi desenvolvida utilizando os diagramas padrões da *Unified Modeling Language* (UML). Foram utilizados os diagramas de casos de uso, diagramas de classes e de sequência que teve sua criação elaborada com auxilio da ferramenta Enterprise Architect 7.1.

Em conjunto com os diagramas acima mencionados também é feita a abordagem sobre a especificação da BNF da linguagem de programação da ferramenta criada para o desenvolvimento de sistemas embarcados.

3.2.1 Especificação do diagrama de casos de uso

A figura 19 demonstra o diagrama de casos de uso e os passos necessários a serem seguidos pelo usuário para que a operacionalização da ferramenta seja feita de forma correta.

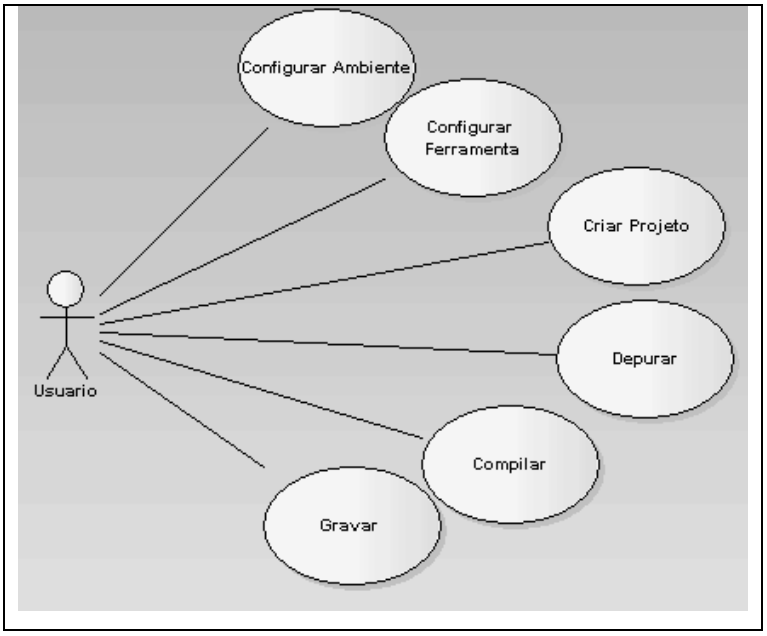


Figura 19 – Diagrama de casos de uso

3.2.1.1 Configurar ambiente

O caso de uso `Configurar ambiente` (Quadro 4) explana o ato de preparar o computador no qual a ferramenta será utilizada, instalando os softwares auxiliares necessários para o pleno funcionamento da ferramenta do presente estudo.

Configurar ambiente: demonstra a configuração do ambiente básico para ferramenta	
Pré Condição	O sistema operacional deve ser Windows XP ou superior e deve possuir porta serial padrão RS232.
Cenário Principal	1) Instalação do software Mikroc. 2) Instalação do software Winpic800.
Exceção	No passo 2, caso o computador não possua porta serial padrão RS232 a ferramenta não será capaz de gravar o software no microcontrolador.
Pós Condições	A ferramenta será capaz de compilar o código fonte e gravar o software compilado no microcontrolador.

Quadro 4 – Caso de uso `Configurar ambiente`

3.2.1.2 Configurar ferramenta

O caso de uso `Configurar ferramenta` (Quadro 5) exibe as formas para se configurar a ferramenta do presente estudo, cadastrando os caminhos dos softwares onde se encontram instalados no computador.

Configurar ferramenta: demonstra a configuração da ferramenta do presente estudo	
Pré Condição	Os softwares Mikroc e Winpic800 necessitam estar instalados no computador.
Cenário Principal	1) Selecionar caminho do software Mikroc. 2) Selecionar caminho do software Winpic800.
Exceção	No passo 1 e no passo 2 caso não hajam os arquivos necessários para a plena execução dos softwares Mikroc ou Winpic800, não será possível realizar a compilação do código fonte e realizar a gravação do binário no microcontrolador.
Pós Condições	A ferramenta será capaz de realizar a criação de projetos de sistemas embarcados, compilando o código fonte e gravando o software compilado no microcontrolador.

Quadro 5 – Caso de uso `Configurar ferramenta`

3.2.1.3 Criar projeto

O caso de uso `Criar projeto` (Quadro 6) demonstra os passos necessários para realizar a criação de um projeto embarcado na ferramenta do presente estudo.

Criar projeto: demonstra a criação de um projeto embarcado	
Pré Condição	As configurações de ambiente e da ferramenta necessitam estarem concluídas.
Cenário Principal	1) Inserir o nome do projeto. 2) Inserir a descrição do projeto. 3) Selecionar o caminho onde o projeto será salvo. 4) Selecionar a família de microcontroladores. 5) Selecionar o modelo do microcontrolador. 6) Inserir a freqüência de *clock* de trabalho. 7) Inserir a quantidade de memória utilizada pelo microcontrolador. 8) Selecionar os *flags* utilizados no projeto.
Exceção	Caso as configurações de ambiente e da ferramenta não estejam configurados, não será possível salvar o projeto e conseqüentemente não será possível realizar a compilação do código fonte nem gravação do software compilado no microcontrolador. A ferramenta irá mostrar uma mensagem de exceção.
Pós Condições	A ferramenta será capaz de salvar o projeto e posteriormente realizar a compilação e gravação do software compilado. Uma mensagem sinalizando a criação do projeto será mostrada.

Quadro 6 – Caso de uso `Criar projeto`

3.2.1.4 Compilar

O caso de uso `Compilar` (Quadro 7) demonstra os passos necessários para realizar a

compilação de um projeto para sistemas embarcados.

Compilar: demonstra os passos necessários na compilação do projeto	
Pré Condição	Um projeto para sistemas embarcados precisa estar corretamente configurado e criado e o software Mikroc necessita estar instalado e funcionando.
Cenário Principal	1) Desenvolver código da linguagem de programação da ferramenta do presente estudo. 2) Validar código fonte. 3) Executar compilação.
Exceção	Caso haja algum erro na validação do código fonte ou a ferramenta Mikroc não esteja instalada corretamente, o software executável não será gerado e a ferramenta irá apresentar a mensagem de exceção correspondente ao erro.
Pós Condições	A ferramenta irá gerar o software executável e irá mostrar uma mensagem ressaltando o sucesso da operação.

Quadro 7 – Caso de uso `Compilar`

3.2.1.5 Depurar

O caso de uso `Depurar` (Quadro 8) descrevem os passos que disponibilizam o módulo de simulação a fim de realizar os testes de projeto da ferramenta do presente estudo.

Depurar: demonstra os passos necessários na simulação do projeto	
Pré Condição	Um projeto para sistemas embarcados precisa estar corretamente configurado e criado e o código da implementação da linguagem necessita ser válido.
Cenário Principal	1) Desenvolver código da linguagem de programação. 2) Validar código fonte. 3) Visualizar os resultados em tela.
Exceção	Caso haja algum erro na validação do código fonte a ferramenta do presente estudo não entrará em modo de simulação e uma mensagem de erro será gerada informando a que exceção se refere.
Pós Condições	A ferramenta irá abrir uma área própria de visualização e simulação e a linguagem previamente concebida irá apresentar os resultados em tela.

Quadro 8 – Caso de uso `Depurar`

3.2.1.6 Gravar

O caso de uso `Gravar` (Quadro 9) especificam os passos para realizar a gravação do software executável no microcontrolador.

Gravar: demonstra os passos necessários na gravação do software executável	
Pré Condição	Um projeto para sistemas embarcados precisa estar corretamente configurado e criado e o código da implementação da linguagem necessita estar compilado bem como o software de gravação Winpic800 instalado corretamente e funcionando.
Cenário Principal	1) Desenvolver código da linguagem de programação. 2) Validar código fonte. 3) Compilar código fonte. 4) Gravar executável no microcontrolador.
Exceção	Caso o código da linguagem não esteja compilado ou a ferramenta Winpic800 não esteja instalada, o software executável não será gravado.
Pós Condições	A ferramenta irá chamar o software Winpic800 e executará a gravação do software executável e apresentará uma mensagem sinalizando o sucesso da operação.

Quadro 9 – Caso de uso `Compilar`

3.2.2 Especificação do diagrama de classes

Este item está subdividido em vários subitens para melhor entendimento da especificação através de classes.

3.2.2.1 Pacotes principais das classes

A figura 20 demonstra visualmente a organização de pacotes das classes do projeto da ferramenta do presente estudo.

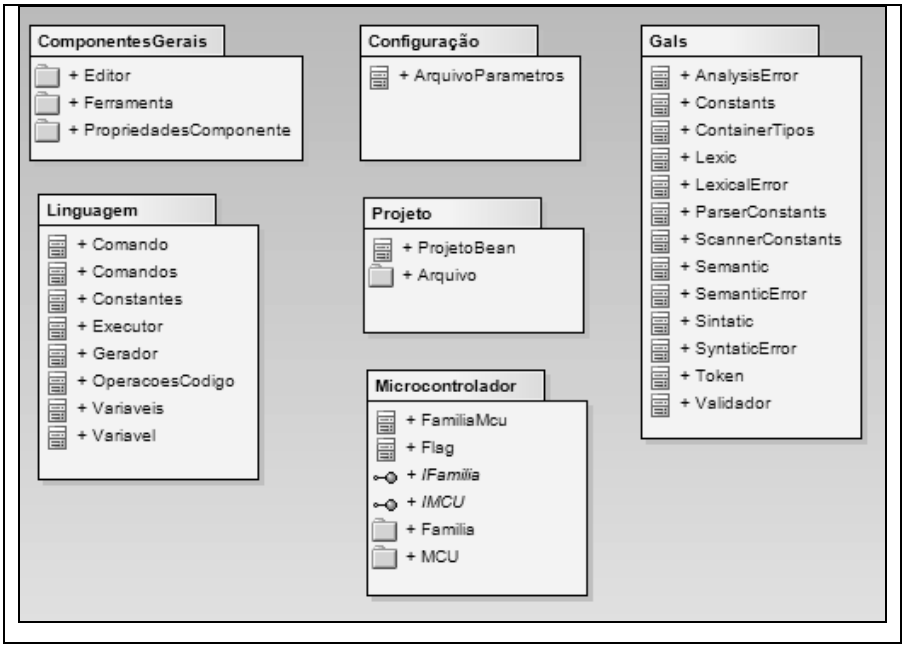


Figura 20 – Pacotes principais das classes

Os pacotes organizam de forma hierárquica as classes do projeto tendo as seguintes atribuições:

a) `ComponentesGerais`: responsável por organizar os componentes da ferramenta e seu detalhamento está especificado na seção 3.2.2.2;

b) `Configuracao`: contém a classe responsável por gerar o arquivo de configuração da ferramenta;

c) `Gals`: as classes geradas pelo software GALS que foram traduzidas para linguagem C# estão contidas nesse pacote e que realizam análise léxica, sintática e semântica;

d) `Linguagem`: contém as classes que realizam as interpretações de código bem como tradução da linguagem da ferramenta do presente estudo para linguagem C, seu detalhamento está especificado na seção 3.2.2.3;

e) `Projeto`: este pacote é responsável por conter as classes necessárias para criação do projeto de sistemas embarcados e a geração dos arquivos do software Mikroc;

f) `Microcontrolador`: responsável por conter as classes de parametrização do microcontrolador e tem seu detalhamento especificado na seção 3.2.2.4.

3.2.2.2 Pacote `ComponentesGerais`

O pacote `ComponentesGerais` contém a organização das classes que fazem a

subdivisão entre componentes para desenvolvimento da ferramenta em si, bem como para os componentes de entrada e saída do microcontrolador, que são usados nas ferramenta como componentes visuais. A figura 21 detalha para maior entendimento o pacote.

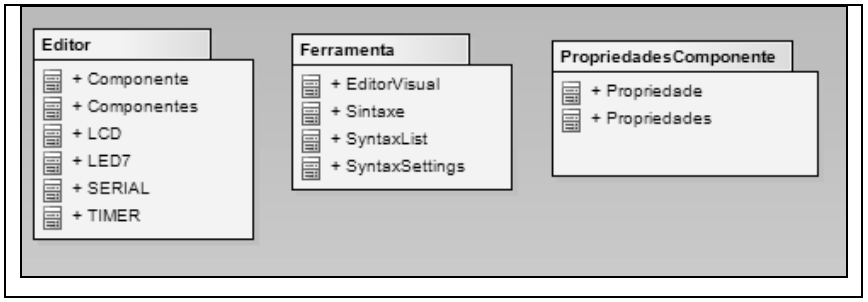


Figura 21 – Pacote `ComponentesGerais`

A figura exposta acima mostra três sub-pacotes que estão subdivididos em `Editor`, `Ferramenta` e `PropriedadesComponente`.

3.2.2.2.1 Pacote `Editor`

Estão organizadas nesse pacote as classes que compõem a gama de componentes de entrada e saída do microcontrolador como as classes bases para criação de componentes e os componentes LCD, serial, timer e LED 7 segmentos. A figura 22 demonstra a visualização do pacote `Editor`.

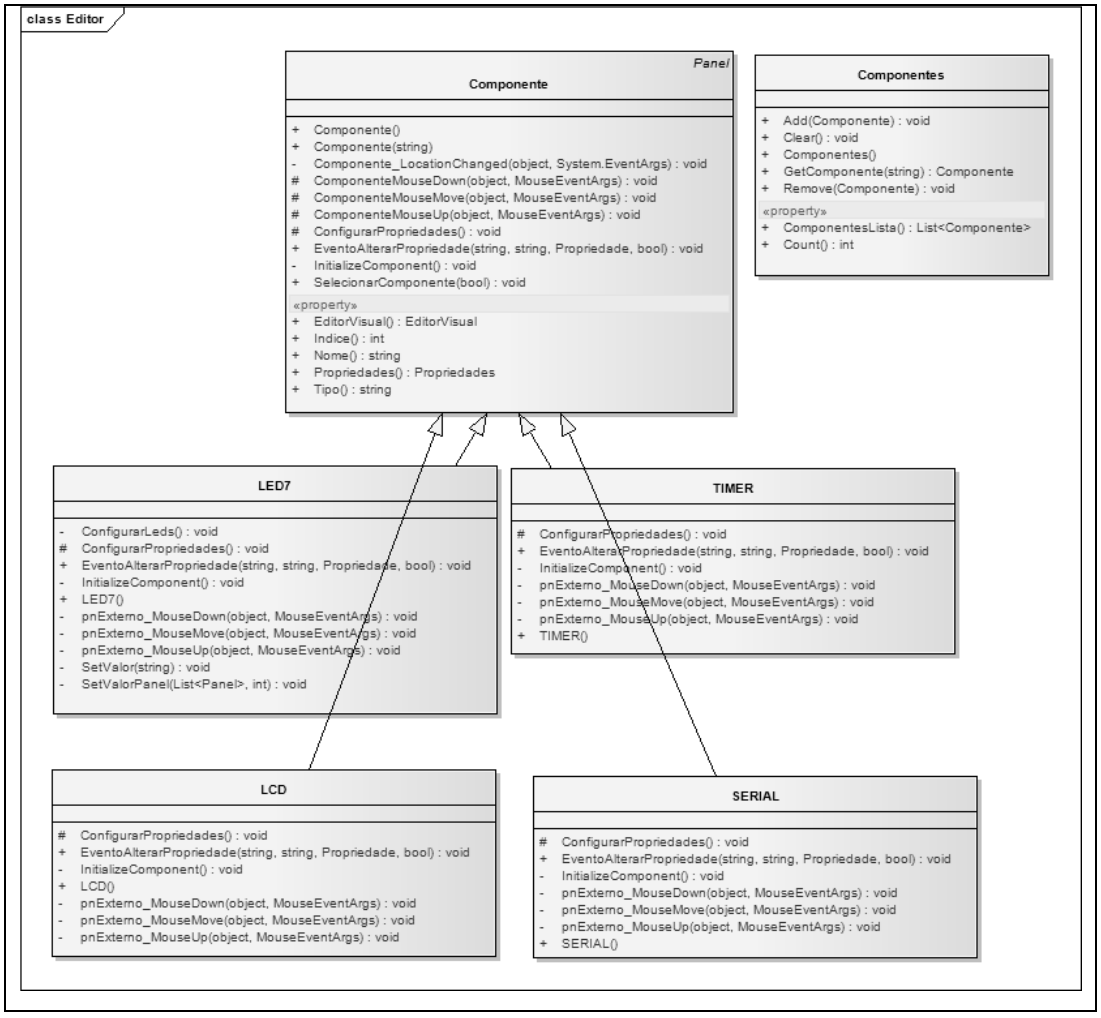


Figura 22 – Pacote `Editor`

A figura 23 demonstra a classe `Componente` que é a classe base para criação de todos os outros componentes visuais da ferramenta e isto ocorre através da extensão desta classe.

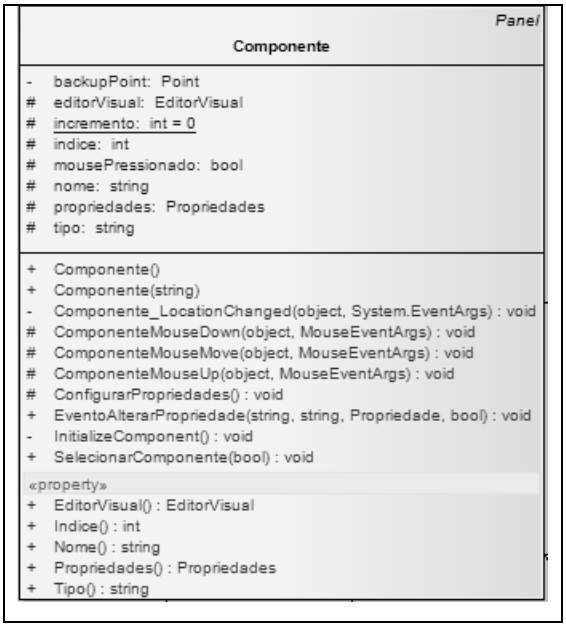


Figura 23 – Classe `Componente`

Os principais métodos da classe `Componente` estão descritos abaixo:

a) `ConfigurarPropriedades`: este método é estendido em todas as classes que estendem a classe componente, sendo responsável por criar as propriedades visíveis na ferramenta;
b) `EventoAlterarPropriedade`: este método quando chamado faz as alterações nas propriedades do componente específico e procede com as alterações da propriedade de forma visual;
c) `ComponenteMouseDown`: evento que detecta a ação do mouse no momento do pressionamento do botão esquerdo e realiza os controles necessários;
d) `ComponenteMouseMove`: através dos controles aplicados pelo evento `ComponenteMouseDown`, o evento `ComponenteMouseMove` é acionado procedendo com o movimento do componente em tela a próprio gosto do usuário sendo posicionado conforme movimento;
e) `ComponenteMouseUp`: evento disparado ao cancelar o pressionamento do botão esquerdo desabilitando os controles que possibilitam o movimento do componente visual da ferramenta.

3.2.2.2.2 Pacote `Ferramenta`

Este pacote agrupa as classes auxiliares no desenvolvimento da ferramenta do presente estudo como a classe que recebe os componentes visuais da ferramenta e a classe que realiza a função de colorir a sintaxe da ferramenta. A figura 24 demonstra a visualização do pacote.

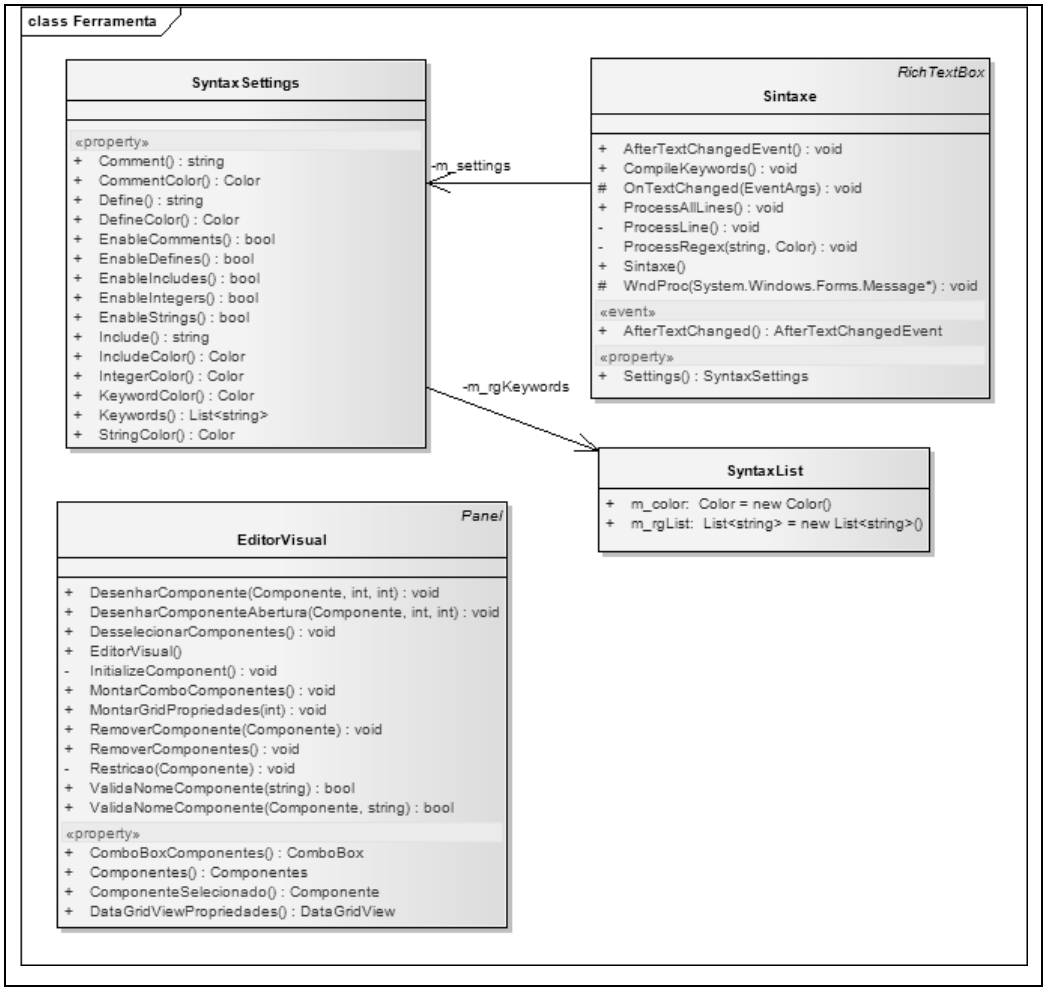


Figura 24 – Pacote `Ferramenta`

A figura 25 demonstra a principal classe deste pacote que é a classe `EditorVisual`.

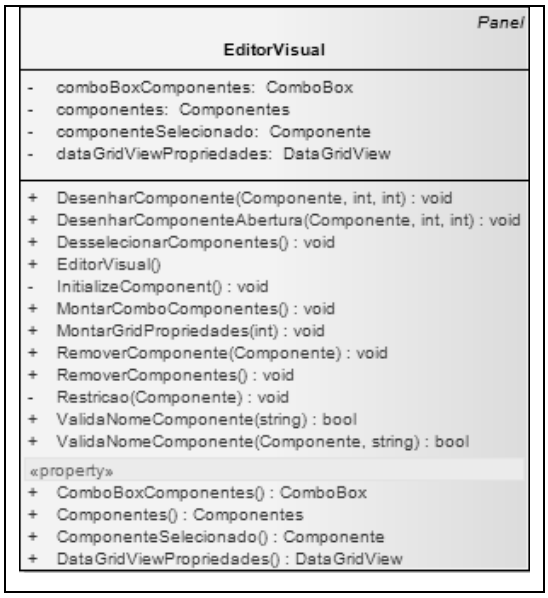


Figura 25 – Classe `EditorVisual`

Os principais métodos desta importante classe são:

a) `DesenharComponente`: para desenhar um componente visual no editor este método

deve ser chamado utilizando as coordenadas x e y;

b) `DesselecionarComponentes`: remove a seleção de todo e qualquer componente selecionado no editor gráfico;
c) `MontarComboComponentes`: de acordo com os componentes adicionados no editor gráfico um componente `ComboBox` é preenchido contendo os nomes dos componentes e seu tipo;
d) `MontarGridPropriedades`: de acordo com a seleção do componente, tanto em tela como no `ComboBox` um grid de propriedades a serem preenchidas para o componente é montada;
e) `RemoverComponente`: remove um determinado componente visual selecionado em tela do editor gráfico;
f) `RemoverComponentes`: remove todos os componentes visuais do editor gráfico;
g) `Restricao`: função responsável por determinar restrições na adição de componentes como, por exemplo, a não possibilidade de adição de dois componentes visuais iguais;
h) `ValidaNomeComponente`: verifica se um determinado componente visual não possui o mesmo nome de um já adicionado no editor visual.

3.2.2.2.3 Pacote `PropriedadesComponente`

Este pacote armazena as classes responsáveis por especificar determinada propriedade de um determinado componente, detalhando informações como nome e características de tipo de propriedade, tamanho dos dados inseridos e se é permitida a utilização desta mesma propriedade na linguagem criada pela ferramenta. A figura 26 demonstra a visualização das classes.

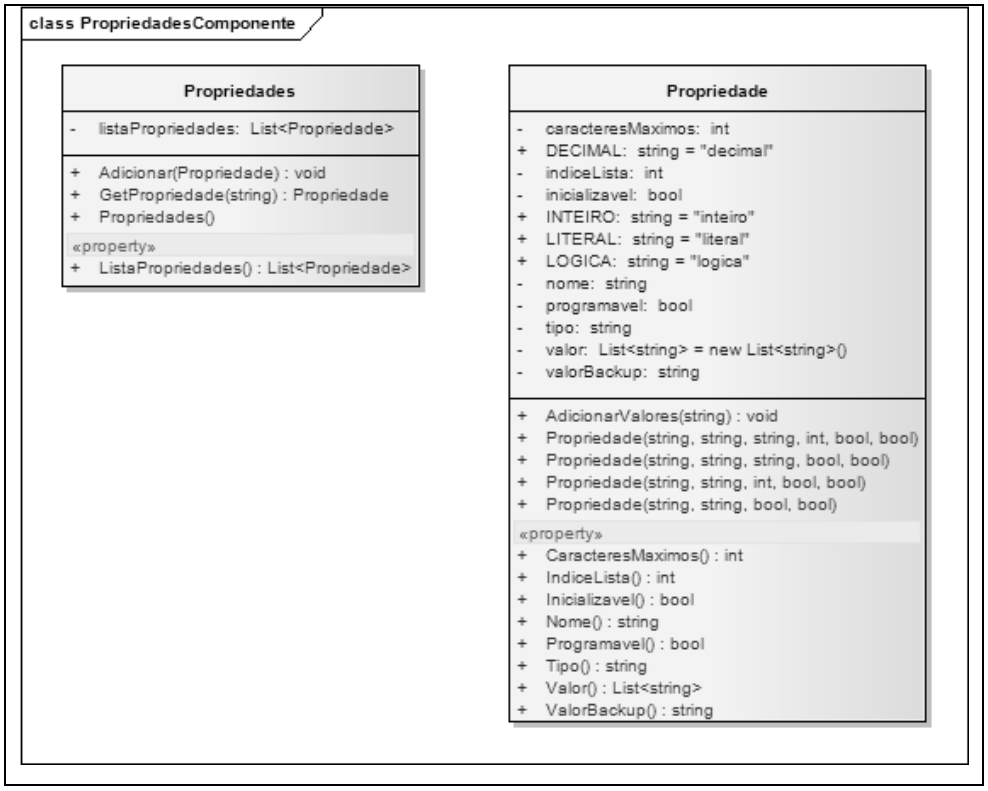


Figura 26 – Pacote `PropriedadesComponente`

A figura 27 mostra a imagem da classe `Propriedade` responsável por reunir as características das propriedades dos componentes.

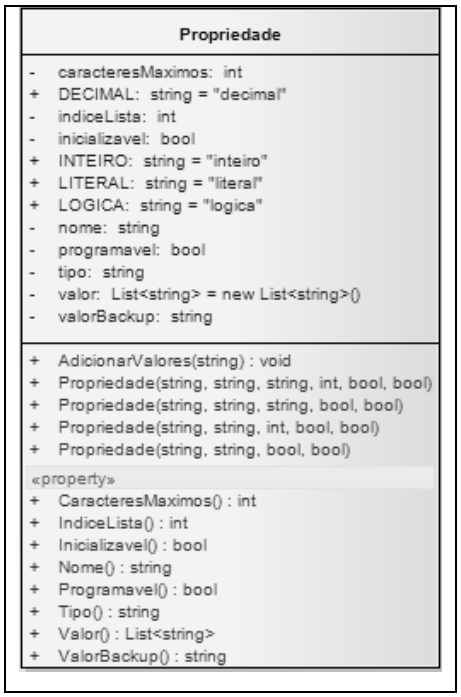


Figura 27 – Classe `Propriedade`

O principal método da classe `Propriedade` é seu construtor por onde são passadas as informações como nome, tipo, quantidade de caracteres da propriedade e se pode ou não ser utilizada em código programável da linguagem da ferramenta proposta.

3.2.2.3 Pacote `Linguagem`

O pacote `Linguagem` armazena as classes que realizam as operações sobre a linguagem de programação desenvolvida para a ferramenta, aplicando os analisadores léxicos, sintáticos, semânticos e dando procedimento a geração de código, bem como a interpretação da linguagem. A figura 28 demonstra as classes do pacote dispostas.

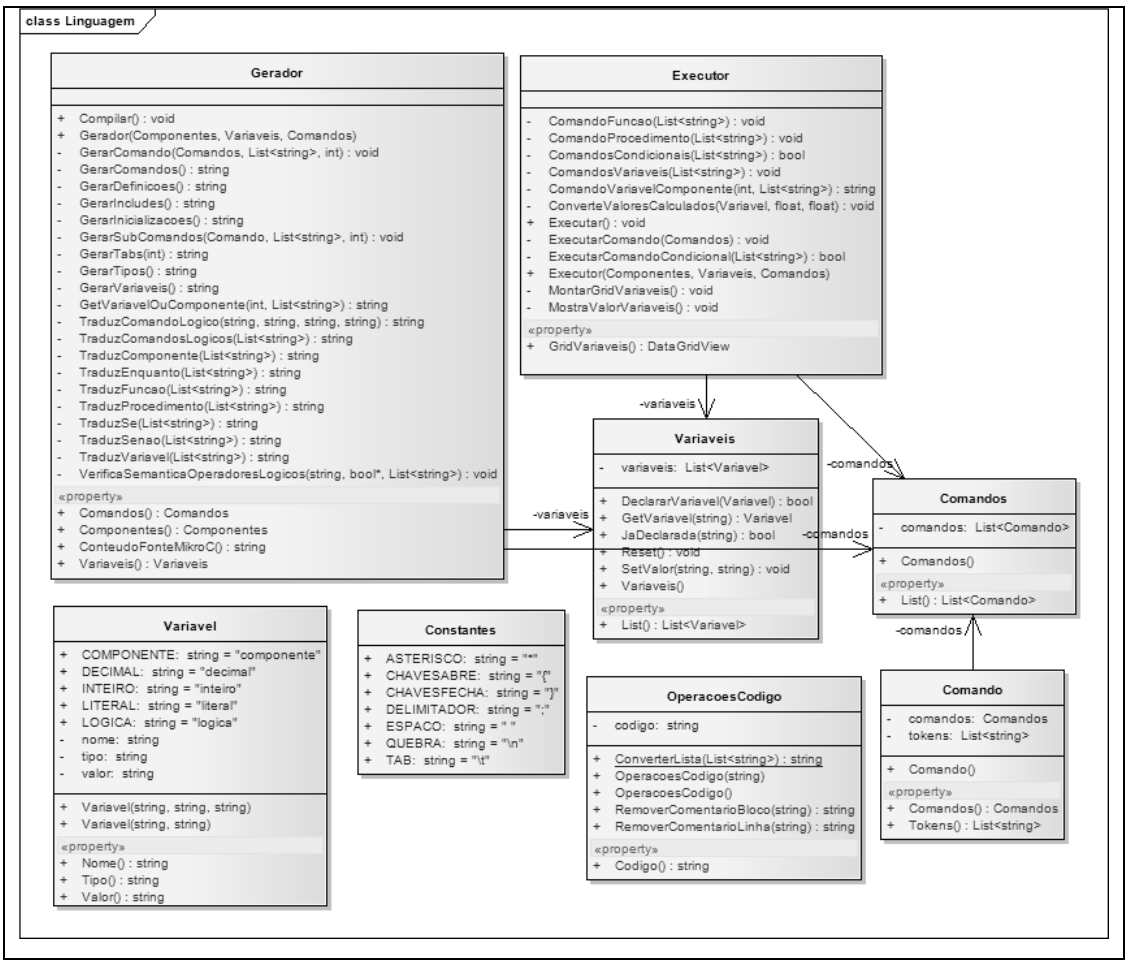


Figura 28 – Pacote `Linguagem`

O pacote `Linguagem` tem como principais classes a `Gerador`, `Executor`, `Variaveis` e `Comandos` que serão melhores descritas nos subitens abaixo.

3.2.2.3.1 Classe `Variaveis`

A classe `Variaveis` é uma importante coleção de dados responsável por armazenar todas as declarações de variáveis do código da linguagem de programação da ferramenta do presente estudo. Possui importantes funções objetivando realizar as verificações de

duplicidade de declarações e alterações de valor. A figura 29 demonstra a classe propriamente dita.

Figura 29 – Classe `Variaveis`

Os principais métodos da classe `Variaveis` são descritos abaixo:

a) `DeclararVariavel`: função que faz a declaração de uma determinada variável através de seu nome e tipo;
b) `GetVariavel`: faz a busca de uma variável através do nome passado como parâmetro;
c) `Reset`: método que refaz os valores das variáveis para vazio quando chamado;
d) `SetValor`: realiza a atribuição de um valor para a variável correspondente ao atributo passado.

3.2.2.3.2 Classe `Comandos`

A classe `Comandos` armazena uma árvore que contém todos os *tokens* organizados de forma a tornar possível o reconhecimento de cada um deles objetivando interpretar e gerar código de acordo com as condições em que o código está organizado. A figura 30 demonstra a ligação entre a classe `Comandos` e `Comando` que torna a efetividade da implementação possível.

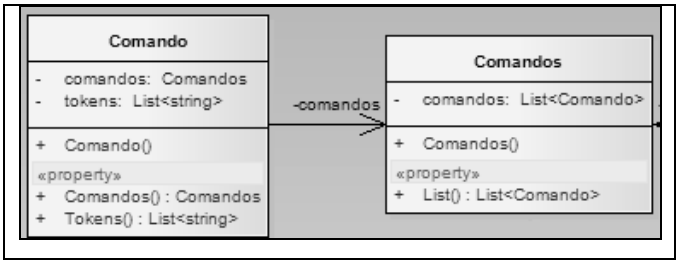


Figura 30 – Classe `Comandos`

Este relacionamento como mostra a figura 30 tem por objetivo conceber uma árvore de vários nós, onde um comando pode receber outros vários comandos, desta forma temos uma

estrutura hierárquica que atende os requisitos para se implementar estruturas condicionais e repetição.

3.2.2.3.3 Classe `Gerador`

A classe `Gerador`, uma das mais vitais na concepção deste sistema, realiza a tradução da linguagem criada para a ferramenta do presente estudo para linguagem C para que, futuramente, seja compilada e gravada. Esta classe utiliza como base a coleção de variáveis, comandos e componentes para realizar as verificações e transformações de código. A figura 31 nos apresenta a classe.

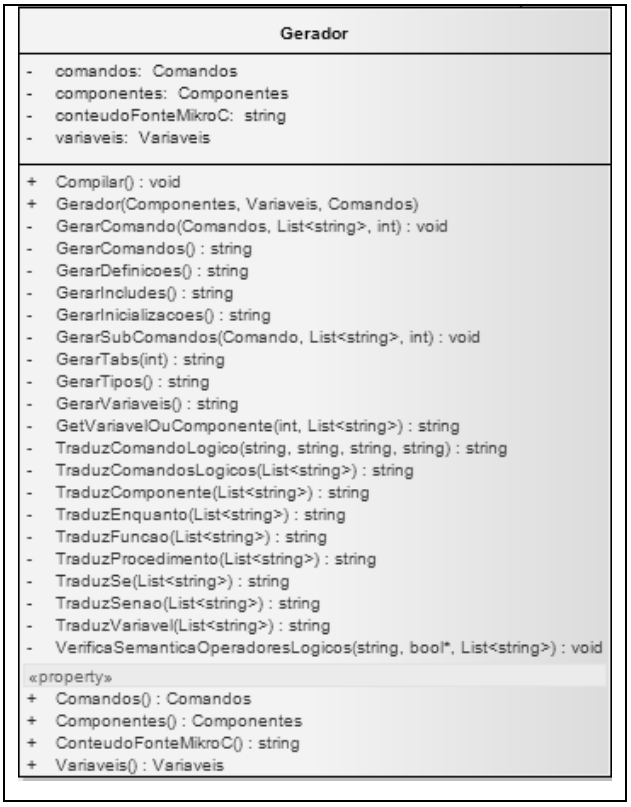


Figura 31 – Classe `Gerador`

Os métodos principais da classe `Gerador` são descritos abaixo:

a) `Gerador`: método construtor por onde são passadas as coleções de dados `Componentes`, `Variaveis` e `Comandos` para realização das operações;
b) `GerarComando`: método recursivo principal por onde todos os outros métodos são invocados e as verificações sobre as estruturas condicionais são realizadas;
c) `GerarDefinicoes`: cria os valores `true` e `false` ausentes na linguagem C;
d) `GerarIncludes`: faz a inclusão dos arquivos necessários para a compilação do

software embarcado, como as bibliotecas dos componentes criados;

e) `GerarInicializacoes`: faz as inicializações dos componentes visuais utilizados no projeto de sistemas embarcados como parâmetros de execução e instalação;
f) `GerarTabs`: faz o cálculo de quantos níveis a recursividade atingiu para gerar as endentações corretas para o código C;
g) `GerarTipos`: realiza a criação dos tipos simbólicos `literal` e `boolean` que não estão presentes na linguagem C;
h) `GerarVariaveis`: percorre a coleção de variáveis e realiza a tradução destas mesmas variáveis para linguagem C, bem como todos os componentes adicionados ao projeto;
i) `GetVariavelOuComponente`: faz a verificação se o componente passado como parâmetro se trata de um componente ou uma variável qualquer;
j) `TraduzComandoLogico`: traduz os comandos utilizados dentro de um comando condicional, seja ele `se` ou `enquanto`;
k) `TraduzComponente`: faz a tradução de código para todo e qualquer componente adicionado ao projeto;
l) `TraduzEnquanto`: traduz o comando `enquanto` chamando a função `TraduzComandosLogicos` conforme definições já demonstradas acima;
m) `TraduzFuncao`: realiza a tradução das funções do código, seja ela para atribuição de variáveis ou para propriedades de componentes;
n) `TraduzProcedimento`: traduz os procedimentos associados as variáveis de acordo com a sintaxe da linguagem C;
o) `TraduzSe`: traduz o comando `se` chamando a função `TraduzComandosLogicos` conforme definições já demonstradas acima;
p) `TraduzSenao`: faz a tradução do comando `senao` para `else` conforme sintaxe da linguagem C;
q) `TraduzVariavel`: faz a verificação se a variável é seguida de uma atribuição seja ela direta, função ou por um procedimento.

3.2.2.3.4 Classe `Executor`

A classe `Executor` tem como função atingir o objetivo de simular a execução do

código fonte da ferramenta do presente estudo. Possui importantes métodos que realizam as transformações das funções escritas em resultados visuais. A figura 32 mostra a classe conforme descrito.

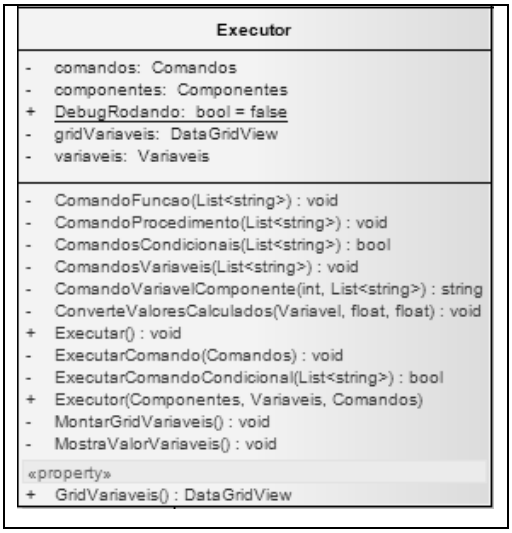


Figura 32 – Classe `Executor`

Os principais métodos da classe `Executor` estão descritos abaixo:

a) `Executor`: principal construtor da classe que recebe como parâmetro as coleções `Componentes`, `Variaveis` e `Comandos` responsáveis por dar a possibilidade de realizar as simulações conforme linguagem;

b) `ComandoFuncao`: método que faz a distinção entre variável ou componente e chama a função apropriada para simulação de procedimentos, funções ou propriedades de ambos;

c) `ComandoProcedimento`: método que simula os procedimentos associados as variáveis;

d) `ComandosCondicionais`: faz a simulação dos comandos condicionais como o `se` e `enquanto` e trata dos operadores lógicos;

e) `ComandosVariaveis`: trata das variáveis que fazem uso de procedimentos ou propriedades que englobam as variáveis de tipos ou componentes;

f) `ComandoVariavelComponente`: método que simula a execução das funções pertencentes as variáveis;

g) `ConverteValoresCalculados`: método responsável por realizar a conversão de valores no momento da execução da simulação;

h) `Executar`: método que faz a chamada de todos os outros métodos e é responsável por invocar a execução da simulação propriamente dita;

i) `ExecutarComando`: método recursivo que varre todo o código de entrada e executa

os comandos de acordo com a simulação;

j) `ExecutarComandoCondicional`: faz a simulação dos comandos condicionais fazendo a distinção entre as funções lógicas executando a simulação de acordo;

k) `MostraGridVariaveis`: método que faz a atualização dos valores de variáveis da simulação em tela no momento da execução do código;

l) `MontarGridVariaveis`: percorre a coleção de variáveis do código e constrói uma listagem de variáveis em tela.

3.2.2.4 Pacote `Microcontrolador`

O pacote `Microcontrolador` contém as classes que realizam a padronização da criação da parametrização dos microcontroladores da ferramenta do presente estudo, contendo as *interfaces* que ditam as diretrizes corretas para criação de uma classe atribuída a um microcontrolador, estando acompanhadas pelas mesmas *interfaces* que realizam a padronização das famílias de microcontrolador. A figura 33 demonstra a visualização do pacote.

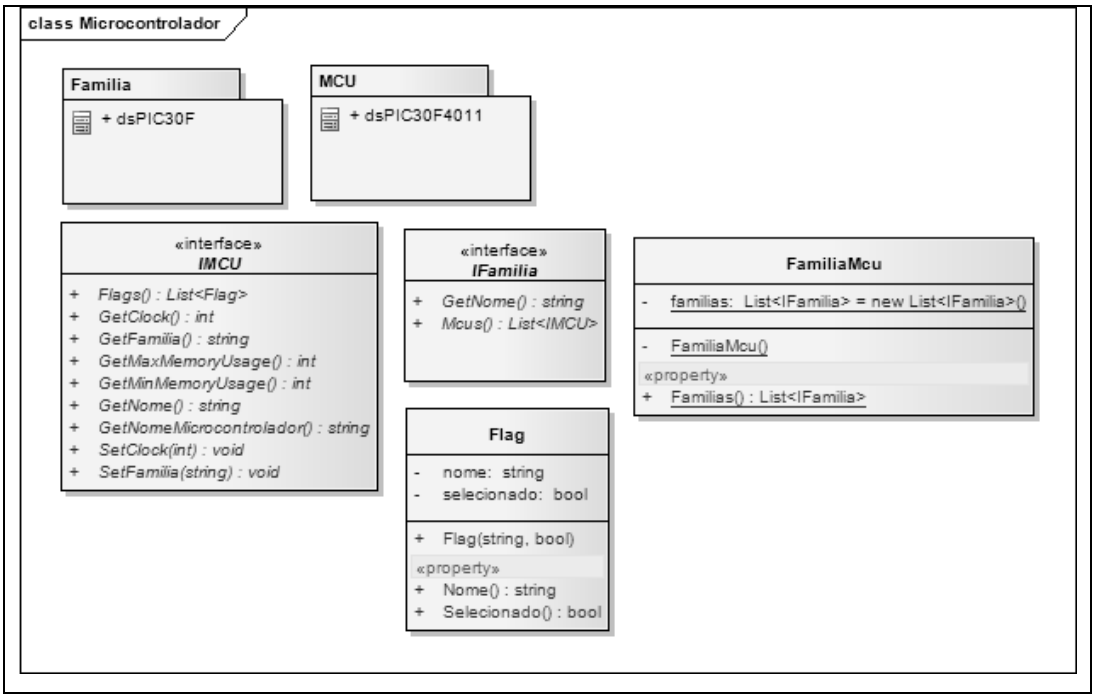


Figura 33 – Pacote `Microcontrolador`

3.2.2.4.1 *Interface* `IMCU`

A *interface* `IMCU` contém as funções obrigatórias para se desenvolver uma classe para microcontrolador possuindo as funções necessárias para se aplicar a um projeto de sistemas embarcados da ferramenta do presente estudo. A figura 34 mostra a *interface* `IMCU`.

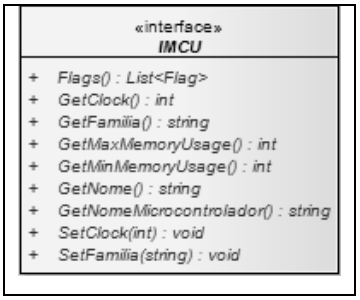


Figura 34 – *Interface* `IMCU`

Os métodos que compõem a *interface* `IMCU` e suas definições são:

a) `Flags`: método que retorna a coleção de *flags* pertencentes ao microcontrolador;
b) `GetClock`: retorna a quantidade de *clock* do microcontrolador;
c) `GetFamilia`: retorna o nome da família do microcontrolador;
d) `GetMaxMemoryUsage`: retorna o máximo de memória usada pelo microcontrolador;
e) `GetMinMemoryUsage`: retorna o mínimo de memória usada pelo microcontrolador;
f) `GetNome`: retorna o nome popular do microcontrolador;
g) `GetNomeMicrocontrolador`: retorna o nome específico do microcontrolador conforme *datasheet*;
h) `SetClock`: atribui a quantidade de *clock* do microcontrolador;
i) `SetFamilia`: atribui o nome da família do microcontrolador.

3.2.2.4.2 *Interface* `IFamilia`

A *interface* `IFamilia` padroniza a criação de classes que tenham como função básica armazenar as informações sobre famílias de microcontroladores e contém duas funções básicas que cumprem este objetivo. A figura 34 representa a classe supracitada.

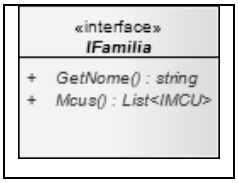


Figura 34 – *Interface* `IFamilia`

Os dois principais métodos estão especificados abaixo:

a) `GetNome`: retorna o nome da família de microcontrolador;
b) `Mcus`: retorna a coleção de microcontroladores adicionados a esta família especifica.

3.2.2.4.3 Classe `FamiliaMcu`

A classe `FamiliaMcu` é uma classe de função estática que armazena todas as famílias de microcontroladores da ferramenta do presente estudo e que possui apenas o método `Familias` que retorna uma coleção de famílias. A figura 35 mostra a classe.

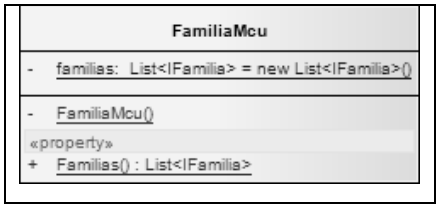


Figura 35 – Classe `FamiliaMcu`

3.2.2.4.4 Classe `dsPIC30F`

A classe `dsPIC30F` nada faz além da extensão da *interface* `IFamilia`, retornando os dados correspondentes conforme definição, e possui um método auxiliar chamado `CarregarMcus`, que faz a adição dos microcontroladores pertencentes a esta família. A figura 36 demonstra.

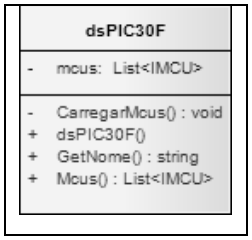


Figura 36 – Classe `dsPIC30F`

3.2.2.4.5 Classe `dsPIC30F4011`

A classe `dsPIC30F4011` estende a *interface* `IMCU` contendo as definições dos métodos que retornam as principais informações sobre o microcontrolador. Possui um método auxiliar para realizar a inserção dos *flags* do microcontrolador chamado `CarregarFlags`. A figura 37 demonstra a classe.

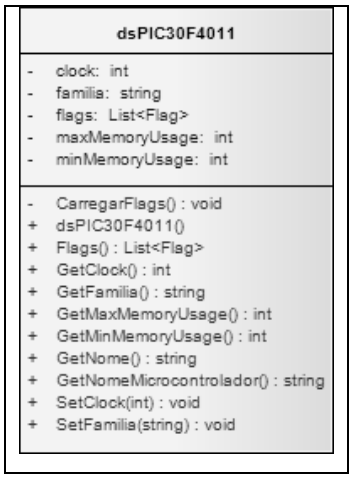


Figura 37 – Classe `dsPIC30F4011`

3.2.3 Especificação do diagrama de sequência

A figura 38 apresenta o diagrama de sequência resumido da ferramenta do presente estudo, mostrando os passos principais que o desenvolvedor de projetos embarcados executa para a correta operacionalização do sistema.

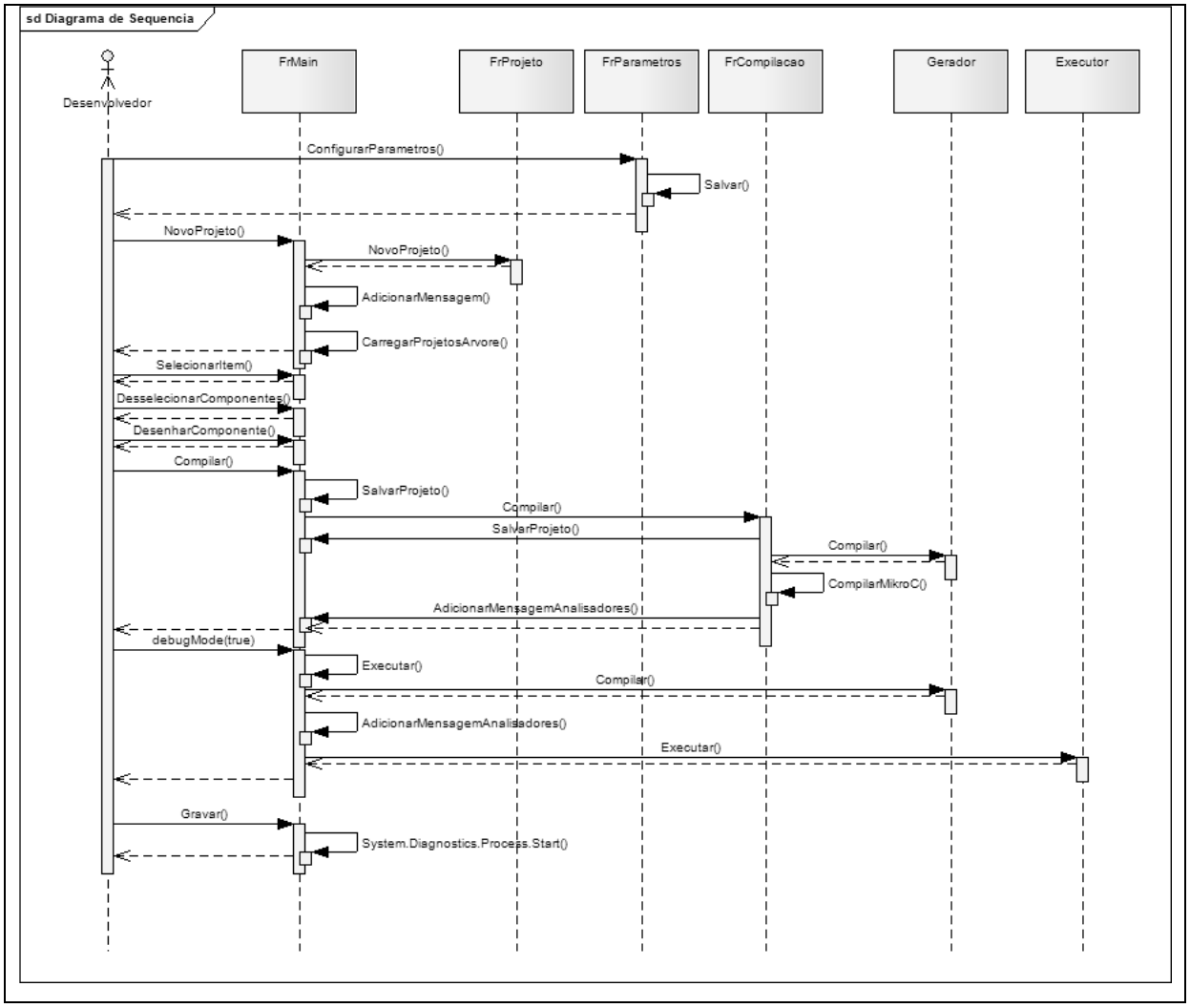


Figura 38 – Diagrama de sequência

3.2.4 Especificação da forma normal de backus BNF

A forma normal de backus mais conhecida como BNF é um formalismo utilizado para representação e criação de gramáticas livres de contexto. Na ferramenta do presente estudo foi utilizada esta técnica para conceber a linguagem de programação em conjunto com a ferramenta GALS. Nesta linguagem estão contemplados os tipos de dados `literal`, `lógico`, `inteiro` e `decimal` e para estruturas de repetição criou-se apenas o comando `enquanto`, possuindo ainda o verificador condicional `se` em conjunto com todas as outras funções especificadas para atender o uso da linguagem. As regras da BNF foram criadas de forma a atender também a utilização de componentes internamente no código considerando também a aplicação de propriedades. O quadro 10 demonstra o exemplo de construção da linguagem para declaração de variáveis.

```
variaveis
{
	varInteira : inteiro;
	varDecimal : decimal;
	varLiteral : literal;
	varLogica  : logica;
}
```

Quadro 10 – Exemplo de declaração de variáveis

O quadro 11 demonstra a utilização de um comando condicional `se`.

```
se ( (varLogica.naoigual(false)) )
{

}
```

Quadro 11 – Exemplo de comando condicional `se`

O quadro 12 demonstra a utilização de uma estrutura de repetição `enquanto`

```
enquanto ( (varLogica.naoigual(true)) )
{

}
```

Quadro 12 – Exemplo de estrutura de repetição `enquanto`

O quadro 13 mostra todas as funções criadas para atender o uso da linguagem.

```
varInteira.sub(2);
varInteira.div(2);
varInteira.mul(2);
varDecimal.add(2);
varLiteral.concatenar("");
varLiteral = varLiteral.subliteral(varInteira, varInteira);
varInteira = varLiteral.indice(varLiteral);
varInteira = varLiteral.ultimoindice(varLiteral);
varInteira = varLiteral.tamanho();
varInteira = varLiteral.parainteiro();
varDecimal = varLiteral.paradecimal();
varLiteral = varDecimal.paraliteral();
varLiteral = varLiteral.caractere(1);
varLiteral = varLiteral.tirarespacos();
```

Quadro 13 – Exemplo de funções utilizadas na linguagem

A especificação da BNF foi subdividida em duas partes para melhor visualização das regras. O quadro 14 mostra o primeiro quadro da BNF.

```
<fonte> ::= <variaveis> <programa>;
<variavel> ::= INTEIRO | LITERAL | DECIMAL | LOGICA ;
<variavelinteiro> ::= DEFINTEIRO | IDENTIFICADOR #16;
<variavelliteral> ::= DEFLITERAL | IDENTIFICADOR #15;
<variaveldecimal> ::= DEFDECIMAL | IDENTIFICADOR #18;
<variavelintdec> ::= DEFINTEIRO | DEFDECIMAL | IDENTIFICADOR #19;
<variavellogica> ::= TRUE | FALSE | IDENTIFICADOR #17;
<variaveltodas> ::= DEFINTEIRO #24 |
                    DEFDECIMAL #23 |
                    DEFLITERAL #7 | TRUE #25 | FALSE #25 |
                    IDENTIFICADOR #20;
<variaveis> ::= VARIAVEIS "{" #1 <declaracao> #4 "}";
<programa> ::= PROGRAMA PRINCIPAL "{" <comandos> "}";
<declaracao> ::= IDENTIFICADOR #2 ":" <variavel> #3 ";" declaracao> | î ;
<comandos> ::= <atribuicao> | <comandoslogicos> | î ;
<atribuicao>::=IDENTIFICADOR #5 <atribuicaometodos> #26 ";" #22 <comandos>;
<atribuicaometodos> ::= "=" <atribuicaoopcoes> | "." <funcoespropriedades>;
<funcoespropriedades> ::= <procedimentos>| <funcoes> |
                          <propriedade> "=" <atribuicaoopcoes>;
<atribuicaoopcoes> ::= <atribuicaovariavel> | <atribuicaodireta>;
<atribuicaovariavel> ::= IDENTIFICADOR #6 <atribuicaofuncao>;
<atribuicaodireta> ::= DEFINTEIRO #24 | DEFDECIMAL #23 |
                       DEFLITERAL #7 | TRUE #25 | FALSE #25;
<atribuicaofuncao> ::= "." <funcoesoupropriedades> | î ;
<funcoesoupropriedades> ::= <funcoes> | <propriedade>;
<propriedade> ::= IDENTIFICADOR #21;
<funcoes> ::= <subliteral> | <indice> | <ultimoindice> |
              <tamanho> | <parainteiro> | <paradecimal> | <paraliteral> |
<caractere> | <tirarespacos>;
<procedimentos> ::= <concatenar> | <sub> | <div> | <mul> | <add>;
<concatenar> ::= CONCATENAR #7 "(" <variavelliteral> ")";
<subliteral>::=SUBLITERAL#8"("<variavelinteiro>","<variavelinteiro>")" #11;
<indice> ::= INDICE #8"(" <variavelliteral> ")" #12;
<ultimoindice> ::= ULTIMOINDICE #8 "(" <variavelliteral> ")" #12;
<tamanho> ::= TAMANHO #8 "(" ")" #12;
<parainteiro> ::= #27 PARAINTEIRO "(" ")" #12;
<paradecimal> ::= #29 PARADECIMAL "(" ")" #14;
<paraliteral> ::= #28 PARALITERAL "(" ")" #11;
```

Quadro 14 – Primeiro quadro BNF

O quadro 15 completa as regras da BNF como é mostrado abaixo.

```
<caractere> ::= CARACTERE #8 "(" <variavelinteiro> ")" #11;
<tirarespacos> ::= #8 TIRARESPACOS "(" ")" #11;
<sub> ::= SUB #9 "(" <variavelintdec> ")";
<div> ::= DIV #9 "(" <variavelintdec> ")";
<mul> ::= MUL #9 "(" <variavelintdec> ")";
<add> ::= ADD #9 "(" <variavelintdec> ")";
<funcoeslogicas> ::= <naoigual> | <igual> | <maior> | <menor>;
<naoigual> ::=  NAOIGUAL #10 "(" <variaveltodas> ")";
<igual> ::=  IGUAL #10 "(" <variaveltodas> ")";
<maior> ::=  MAIOR #9 "(" <variavelintdec> ")";
<menor> ::=  MENOR #9 "(" <variavelintdec> ")";
<comandoslogicos> ::= <se> | <enquanto>;
<se> ::= SE "(" <condicoes> ")" "{" <comandos> "}" <senao> <comandos>;
<senao> ::= SENAO "{" <comandos> "}" | î;
<enquanto> ::= ENQUANTO "(" <condicoes> ")" "{" <comandos> "}" <comandos>;
<condicoes>::="("IDENTIFICADOR#5"."<funcoeslogicas>")"<condicoesextensao> ;
<condicoesextensao> ::= OU <condicoes> | E <condicoes> | î;
```

Quadro 15 – Segundo quadro BNF

3.3 IMPLEMENTAÇÃO

A seguir são apresentados detalhes sobre a implementação da ferramenta, explanando as técnicas e ferramentas utilizadas no desenvolvimento, bem como os passos necessários para atender os objetivos do presente estudo. Em seguida são apresentados a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

No desenvolvimento da ferramenta foi utilizado o ambiente de programação Microsoft Visual Studio 2008 fazendo uso da linguagem de programação C# e aplicando o paradigma de orientação a objetos.

3.3.1.1 Implementação da criação de projetos embarcados

Para o desenvolvimento da criação de projetos embarcados, utilizaram-se dois conceitos distintos, que são a criação de projetos para a ferramenta do presente estudo e a criação de projetos do software Mikroc. A ferramenta Mikroc necessita possuir um conjunto de arquivos separados, pois a compilação do software embarcado se dá através dele, sendo assim, quando um projeto é criado um conjunto de arquivos para ferramenta é criado a fim de armazenar esta configuração e, em paralelo a isto, um espelho destes arquivos é formatado na estrutura que o software Mikroc reconhece. A figura 39 mostra a estrutura do diretório de um projeto criado.

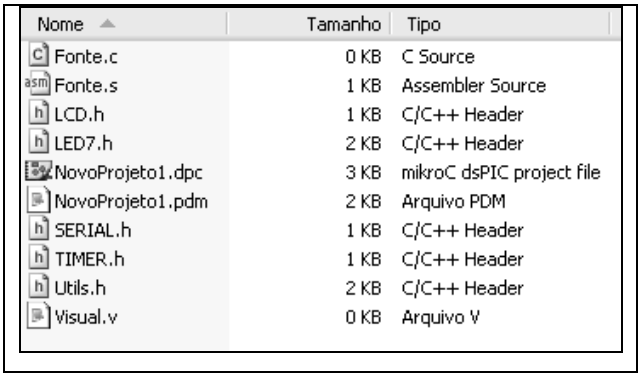


Figura 39 – Estrutura do diretório do projeto

Como demonstra a figura 39, nota-se a presença de um arquivo `NovoProjeto1.dpc` que é o arquivo de projeto do Mikroc e o arquivo `NovoProjeto1.pdm`, que é o arquivo que contém as configurações de projeto da ferramenta. O quadro 16 mostra a estrutura de um arquivo `pdm` omitindo os *flags*.

```
nome=Novo Projeto 1
descricao=Digite aqui a descrição do Projeto
familia=0
microcontrolador=0
clock=8
memoria=20
caminho=C:\
arquivoFonte=C:\Novo Projeto 1\Fonte.s
arquivoVisual=C:\Novo Projeto 1\Visual.v
arquivoProjetoMikroC=C:\Novo Projeto 1\NovoProjeto1.dpc
arquivoFonteMikroC=C:\Novo Projeto 1\Fonte.c
```

Quadro 16 – Estrutura do arquivo de projeto da ferramenta

O quadro 17 mostra a estrutura do arquivo de projeto do software Mikroc.

```
[DeviceName]
value=P30F4011
[DeviceClock]
value=8
[MainUnit]
value=Fonte.c
[DeviceFlags]
Count=76
value0=_CSW_FSCM_OFF = $FFFF
value1=_CSW_ON_FSCM_OFF = $7FFF
value2=_CSW_FSCM_ON = $3FFF
value3=_EC = $FFFB
value4=_ECIO = $FFFC
value5=_EC_PLL4 = $FFFD
value6=_EC_PLL8 = $FFFE
value7=_EC_PLL16 = $FFFF
value8=_ERC = $FFF9
value9=_ERCIO = $FFF8
value10=_XT = $FFF4
value11=_XT_PLL4 = $FFF5
value12=_XT_PLL8 = $FFF6
[BuildType]
bType=0
[ProjectFiles]
Count=1
Value0=Fonte.c
[ObjLibFiles]
Count=0
[EEPROMinfo]
isused=0
[MemoryUsage]
Val=20
[SearchPath]
Count=3
Value0=C:\Arquivos de programas\Mikroelektronika\mikroC dsPIC\Defs\
Value1=C:\Arquivos de programas\Mikroelektronika\mikroC dsPIC\Uses\
Value2=C:\Novo Projeto 1\
[IncludePath]
Count=0
```

Quadro 16 – Estrutura do arquivo de projeto do software Mikroc

O arquivo `Visual.v` também faz parte desta estrutura de projetos, mas não possui nenhum vínculo com o software Mikroc. Este arquivo contém as informações pertinentes aos componentes visuais da ferramenta armazenando os valores das propriedades como nome, posição e outras características. O quadro 17 apresenta esta estrutura.

O quadro 17 mostra a estrutura do arquivo de projeto do software Mikroc.

```
LCD:(PosicaoX=109; PosicaoY=70; Nome=lcd1; Linha1=; Linha2=; Cursor=0)
SERIAL:(PosicaoX=327; PosicaoY=243; Nome=serial2; Bits=9600; Dados=)
LED7:(PosicaoX=461; PosicaoY=116; Nome=led73; Valor=0)
TIMER:(PosicaoX=555; PosicaoY=284; Nome=timer4; Intervalo=1000; Ativo=0)
```

Quadro 17 – Estrutura do arquivo de projeto Visual.v

A criação e abertura destes arquivos são realizadas de forma simples e não possuem nenhum padrão envolvido no processo, exceto pelo fato de que o arquivo `Visual.v` por lidar com componentes físicos necessita utilizar a técnica de programação reflexiva que realiza a instância do componente correspondente ao tipo gravado no arquivo. O quadro 18 mostra em código fonte a técnica.

```
public override void Abrir()
{
   base.Abrir();
   string path = "ADDM30F.ComponentesGerais.Editor.";

   for (int index = 0; index < this.linhas.Count; index++)
   {
      string[] linha = this.linhas[index].Split(':');

      Type type = Type.GetType(path + linha[0]);
      Componente componente = (Componente)Activator.CreateInstance(type);
      componente.Indice = index;
      SetPropriedades(componente,linha[1].Substring(1,linha[1].Length-2));
      this.componentes.Add(componente);
   }
}
```

Quadro 18 – Programação reflexiva no arquivo Visual.v

3.3.1.2 Implementação do editor visual com *drag and drop*

Para realizar o desenvolvimento da técnica de *drag and drop*, o componente `EditorVisual` foi utilizado como container recebendo as requisições de desenho sendo responsável pelo movimento e manipulação dos componentes. Para desenhar o componente, uma instancia do mesmo é criada ao clicar sobre o item correspondente na paleta de componentes e desenhado ao disparar o método `MouseClick`. O quadro 19 mostra o código

fonte do método supracitado.

```
private void editorVisual_MouseClick(object sender, MouseEventArgs e)
{
   this.cbComponentes.SelectedIndex = -1;
   this.dgvPropriedades.Rows.Clear();

   DesselecionarComponentes();

   if (this.componenteDesenho != null)
   {
      try
      {
         editorVisual.DesenharComponente(this.componenteDesenho, e.X,e.Y);
         editorVisual.MontarComboComponentes();
         this.projetoSelecionado.ProjetoSalvo = false;
      }
      catch (Exception exc)
      {
          MessageBox.Show(exc.Message, "Adicionar",
          MessageBoxButtons.OK, MessageBoxIcon.Error);
      }

      this.Cursor = Cursors.Default;
      this.componenteDesenho = null;
   }
}
```

Quadro 19 – Evento para *drag and drop*

3.3.1.3 Implementação da paleta de componentes

A paleta de componentes é concebida de forma visualmente simples e conta com os botões que correspondem aos componentes visuais da ferramenta. Os botões que compõem a paleta e realizam a instanciação dos itens são:

a) cursor: representa o cursor do *mouse*;
b) remover: remove o componente selecionado;
c) lcd gráfico: representa o componente LCD Gráfico;
d) porta serial: representa o componente porta serial;
e) led 7 segmentos: representa o componente LED 7 Segmentos;
f) *timer*: representa o componente *timer*.

A figura 40 demonstra a paleta de componentes.

Figura 40 – Paleta de componentes

O quadro 20 demonstra o código empregado nos itens da paleta de componentes que realizam a instanciação. O exemplo a seguir mostra o evento do botão `LCD Gráfico` que instancia um componente LCD na variável `componenteDesenho` que é utilizado pelo evento `MouseClick` demonstrado no item anterior.

```
private void btLcd_Click(object sender, EventArgs e)
{
   DesselecionarComponentes();
   this.componenteDesenho = new LCD();
   this.Cursor = Cursors.Cross;
}
```

Quadro 20 – Evento paleta de componentes

3.3.1.4 Implementação dos analisadores léxico, sintático e semântico

O desenvolvimento dos analisadores léxico e sintático é realizado através da geração do código fonte com a ferramenta GALS contando com a especificação da BNF. A geração é realizada em linguagem Java e traduzida manualmente para linguagem C#. A figura 41 mostra os arquivos já traduzidos.

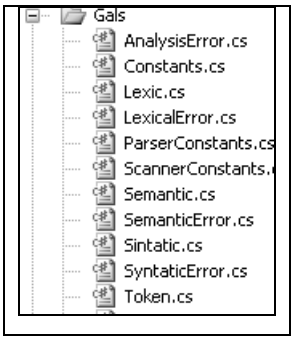


Figura 41 – Arquivos GALS

O analisador semântico possui uma implementação a parte com o auxílio que a ferramenta GALS proporciona através dos símbolos que podem ser introduzidos na gramática da linguagem e possibilita um desenvolvimento de maior controle dos *tokens*, pois no momento em que o *parser* identifica um desses símbolos, a regra semântica pode ser interpretada e aplicada.

O quadro 21 demonstra um trecho da gramática para exemplificar a aplicação dos símbolos semânticos.

```
<declaracao> ::= IDENTIFICADOR #2 ":" <variavel> #3 ";" <declaração> | î ;
```

Quadro 21 – Símbolos semânticos

É possível notar no quadro 21 dois números precedidos por um sustenido, que são `#2` e `#3`, e correspondem aos símbolos previamente explicados. No exemplo de código a seguir será usado o símbolo semântico `#3`, que realiza a função de declarar determinada variável e notificar caso haja duplicidade de declarações conforme quadro 22 mostrando o trecho de código da classe `Semantic`, gerada previamente pela ferramenta GALS.

```
case 3:
{
   if (variavelDeclaracao != "")
   {
      if (componentes.GetComponente(variavelDeclaracao) != null)
      {
         throw new SemanticError("Variavel " + variavelDeclaracao +
                                 " já  declarada como componente!");
      }

      if (!variaveis.JaDeclarada(variavelDeclaracao))
      {
         variaveis.DeclararVariavel(new  Variavel(variavelDeclaracao,
                                    token.getLexeme()));
         variavelDeclaracao = "";
      }
    }
    else
    {
        throw new SemanticError("Variavel " + variavelDeclaracao +
                                " já declarada!");
    }
}
```

Quadro 22 – Símbolos semânticos

3.3.1.5 Implementação da interpretação e tradução da linguagem

A interpretação e tradução da linguagem são realizadas obtendo a lista de *tokens* advinda do analisador léxico na classe `Lexic` e organizados em uma estrutura de dados do

tipo árvore, que tem por finalidade estruturar os *tokens* de forma hierárquica possibilitando um maior poder de manipulação, uma vez que ao interpretar um comando lógico sendo verificado em um comando condicional, por exemplo, os tokens que estão contidos dentro deste mesmo comando podem ser ignorados. Para que se faça entender tal implementação, a figura 42 é apresentada.

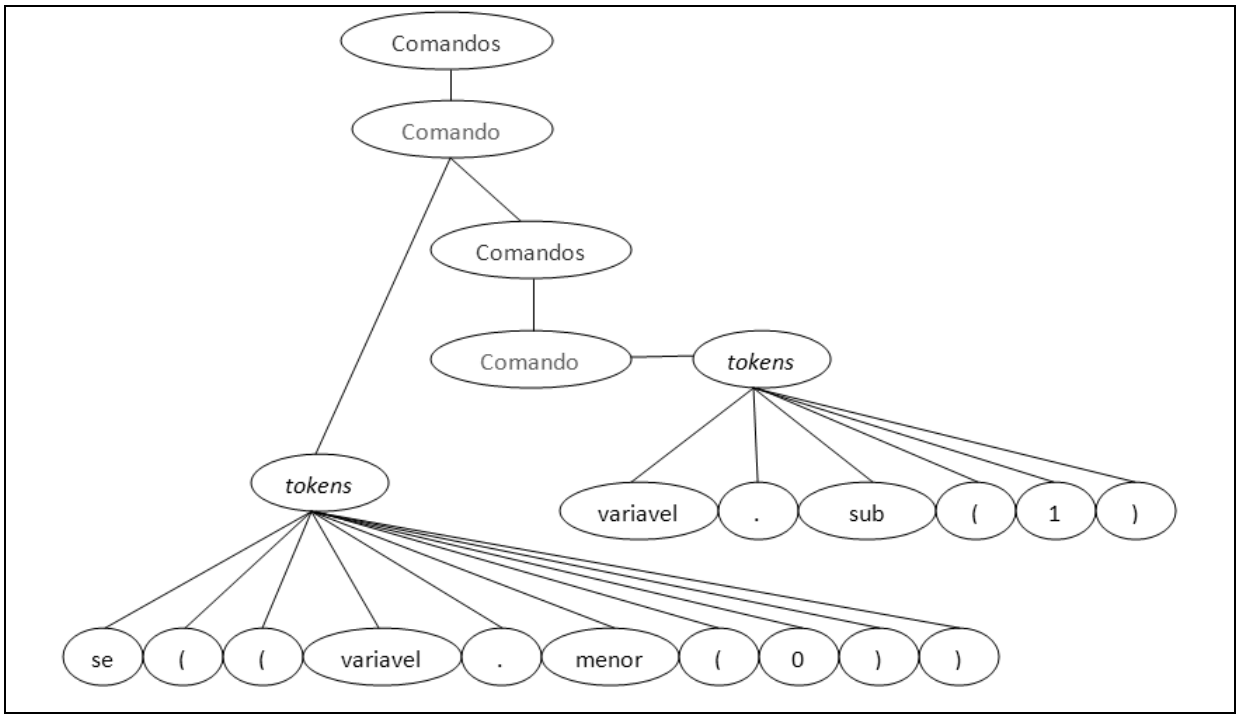


Figura 42 – Estrutura de tradução e interpretação

As rotinas de tradução e interpretação estão baseadas na estrutura apresentada na figura 42. Através do nó principal da arvore de comandos é executada uma rotina recursiva que chega a todos os nós da arvore.

3.3.1.5.1 Tradução

A tradução dos comandos é feita de forma literal para a linguagem C, ou seja, percorre-se a árvore de comandos e a cada *token* processado é substituído pelo comando ou sintaxe correspondente na linguagem C. Os quadros 23 e 24 respectivamente mostram a execução desta rotina pertencente à classe `Gerador`.

```
private string TraduzSe(List<string> tokens)
{
   return "if ( " + TraduzComandosLogicos(tokens) + " )";
}
```

Quadro 23 – Rotina de tradução comando `se`

A função `TraduzSe` é executada quando um *token* é identificado como sendo um comando condicional `se` e a lista de *tokens* é passada para a chamada mais especializada `TraduzComandosLogicos` que está detalhada de forma mais resumida para que se faça entender.

```
case "igual":
{

   switch (tipo)
   {
      case Variavel.LITERAL:
      {

         comando += "strcmp(" + variavelToken + ", " +
                                parametroToken + ") == 0";
      }

      break;

      default:

         comando += variavelToken + " == " + parametroToken;

      break;

   }

}
```

Quadro 24 – Rotina de tradução `TraduzComandosLogicos`

O código apresentado no quadro 24 demonstra o *token* `igual` sendo identificado na iteração da lista de *tokens* e, de acordo com o tipo da variável presente nesta verificação, o comando correspondente é traduzido. Neste exemplo, se a variável for do tipo `literal` a tradução remete ao comando `strcmp` utilizado na linguagem C para realizar a verificação da igualdade de duas cadeias de caracteres passadas. Caso a variável seja de qualquer outro tipo, os operadores de igualdade `==` utilizados na linguagem C para este mesmo fim são utilizados.

3.3.1.5.2 Interpretação

A interpretação dos comandos é feita para apresentar os resultados visuais da execução do código em tela através da classe `Executor` e segue a mesma lógica da tradução, porém os comandos são executados imediatamente apresentando os resultados que se esperam. O quadro 25 demonstra a execução resumida da função `ExecutarComandoCondicional` e as variações de cada comando.

```
switch (tokens[2])
{
   case "igual":
   {
      return variavel.Valor == parametro;
   }

   case "naoigual":
   {
      return variavel.Valor != parametro;
   }

   case "menor":
   {
       return double.Parse(variavel.Valor) < double.Parse(parametro);
   }

   case "maior":
   {
       return double.Parse(variavel.Valor) > double.Parse(parametro);
   }
}
```

Quadro 25 – Rotina resumida de execução

Como demonstra o quadro 25, os *tokens* identificados são diretamente executados conforme finalidade atribuída a cada um e seu resultado é diretamente enviada à função que o chamou.

3.3.1.5.3 Bibliotecas prontas

As bibliotecas prontas são os arquivos criados em linguagem C conhecidos como *header files (.h)* a fim de atender pequenas limitações que o desenvolvimento para microcontroladores utilizando a ferramenta Mikroc possui. Estes arquivos são adicionados ao projeto no momento da criação e utilizados no código fonte traduzido e devidamente requisitados conforme a demanda. Os arquivos criados são `Utils.h`, `LCD.h`, `LED7.h`, `TIMER.h` e `SERIAL.h`.

O arquivo `Utils.h` contém as funções auxiliares no desenvolvimento de código que não existem na biblioteca interna do software Mikroc. O quadro 26 contém as assinaturas das funções e suas descrições.

Função	Descrição
`Trim`	Função que retira os espaços à direita e a esquerda da cadeia de caracteres.
`Substring`	Captura um subtexto dentro de outro texto passado através de um índice inicial e o tamanho do subtexto desejado.
`indexOf`	Busca o índice de um subtexto dentro de um texto passado.
`lastIndexOf`	Busca o ultimo índice de um subtexto dentro de um texto passado.
`charAt`	Busca um caractere através de um índice presente dentro do texto passado.

Quadro 26 – Descritivo das funções `Utils.h`

Os arquivos `LCD.h`, `LED7.h`, `TIMER.h` e `SERIAL.h` são responsáveis por contemplar as funcionalidades dos componentes da ferramenta do presente estudo, contendo um tipo de dados que armazena as propriedades internas de cada um e as funções responsáveis por instalar e invocar estas propriedades. O quadro 27 demonstra a estrutura padrão do arquivo.

```
typedef struct
{
	//propriedades do componente

} COMPONENTE;

void installCOMPONENTE(COMPONENTE componente)
{
	//função que realiza a instalação do componente
}

void handleCOMPONENTE(COMPONENTE componente)
{
	//função que invoca a chamada das propriedades
}
```

Quadro 27– Estrutura padrão do *header file*

É importante salientar que esta estrutura deve ser seguida para todo novo componente implementado na ferramenta, uma vez que a geração de código utiliza esta padronização para trabalhar com os componentes de entrada e saída do microcontrolador. A geração de código substitui a constante `COMPONENTE` evidenciada no quadro 27 pelo tipo do componente. O quadro 28 exemplifica a explanação.

Componente	Funções
`LCD`	`void installLCD(LCD lcd);` `void handleLCD(LCD lcd);`
`LED7`	`void installLED7(LED7 led7);` `void handleLED7(LED7 led7);`
`SERIAL`	`void installSERIAL(SERIAL serial);` `void handleSERIAL(SERIAL serial);`
`TIMER`	`void installTIMER(TIMER timer);` `void handleTIMER(TIMER timer);`

Quadro 28– Exemplo de funções de componentes

Nos quadros abaixo serão feitos os detalhamentos de cada componente descrevendo as propriedades e especificando as assinaturas das funções de cada arquivo.

As propriedades do componente LCD são especificadas no quadro 29.

Propriedade	Descrição
`Linha1`	Escreve o texto na primeira linha e primeira coluna do LCD.
`Linha2`	Escreve o texto na segunda linha e primeira coluna do LCD.
`Cursor`	Configura a opção de mostrar ou não o cursor no LCD.

Quadro 29– Especificação das propriedades do componente LCD

As funções do componente LCD são especificadas no quadro 30 e o código fonte está contido no apêndice a.

Funções	Descrição
`installLCD`	Função responsável por configurar o componente LCD nas portas corretas em que está ligado ao microcontrolador.
`handleLCD`	Função que transpõe o conteúdo das propriedades do LCD nas APIs correspondentes a escrita no LCD bem como a desativação do cursor.

Quadro 30– Especificação das funções do componente LCD

As propriedades do componente LED7 são especificadas no quadro 31 .

Propriedade	Descrição
`Valor`	Valor inteiro que é correspondente ao número decimal que será escrito no LED de sete segmentos.

Quadro 31– Especificação das propriedades do componente LED7

As funções do componente LED7 são especificadas no quadro 32 e o código fonte está contido no apêndice b.

Funções	Descrição
`installLCD`	Função responsável por configurar o microcontrolador para escrever os valores no LED de sete segmentos.
`convert`	Função que recebe um número inteiro e retorna o código hexadecimal correspondente aos segmentos de LED a serem acesos para aquele número em especifico.
`handleLCD`	Função que captura o número inteiro da propriedade `Valor` e realiza a quebra do mesmo de forma a conter quatro algarismos distintos onde cada um deles é escrito em seu LED de sete segmentos correspondente.

Quadro 32– Especificação das funções do componente LED7

As propriedades do componente SERIAL são especificadas no quadro 33.

Propriedade	Descrição
`Dados`	Propriedade que recebe o conteúdo a ser transmitido via porta serial.
`Bits`	Quantidade de *bits* por segundo.

Quadro 33– Especificação das propriedades do componente SERIAL

As funções do componente SERIAL são especificadas no quadro 34 e o código fonte está contido no apêndice c.

Funções	Descrição
`installLCD`	Função que inicializa a porta serial com os bits por segundo presente na propriedade `Bits`.
`handleLCD`	Função que realiza o envio do conteúdo presente na propriedade `Dados`.

Quadro 34– Especificação das funções do componente SERIAL

As propriedades do componente TIMER são especificadas no quadro 35.

Propriedade	Descrição
`Intervalo`	Propriedade que recebe um número inteiro representando a quantidade de milissegundos.
`Ativo`	Propriedade que recebe uma informação lógica representando a atividade de intervalo.

Quadro 35– Especificação das propriedades do componente TIMER

As funções do componente TIMER são especificadas no quadro 36 e o código fonte está contido no apêndice d.

Funções	Descrição
`installLCD`	Função que inicializa as propriedades do componente.
`handleLCD`	Função que realiza a chamada da API responsável pelo *delay* de software.

Quadro 36– Especificação das funções do componente TIMER

3.3.1.5.4 Código produzido para código gerado

No presente item é feito um comparativo entre o código produzido pela ferramenta e o código gerado em linguagem C. No quadro 37 é apresentado um exemplo de código produzido seguido pela sua tradução onde um determinado texto é escrito em um componente LCD. No código fonte contém uma serie de comentários a fim de descrever cada passo do código.

```
variaveis
{
      // declaração de variáveis
      texto : literal;
}

programa principal
{
      // armazena o conteúdo na variavel texto
      texto = "Teste TCC II";
      // escreve a variavel texto na linha 1 do LCD
      lcd1.Linha1 = texto;
}
```

Quadro 37– Código produzido

O quadro 38 contém o código traduzido que é baseado no código produzido presente no quadro 37.

```
// arquivos incluídos no código fonte
#include "Utils.h"
#include "LCD.h"
#include "LED7.h"
#include "SERIAL.h"
#include "TIMER.h"

// criação dos tipos boolean e string
typedef unsigned short boolean;
typedef char string[20];

// criação dos valores true e false não presentes na linguagem C
#define true 0
#define false 1

// declaração do component LCD
LCD lcd1;

// declaração da variável texto
string texto;

void main()
{
	// instalação do componente LCD
	installLCD(lcd1);

	// inicialização dos dados do componente LCD
	strcpy(lcd1.Linha1, "");
	strcpy(lcd1.Linha2, "");
	lcd1.Cursor = false;

	// instalação do componente LCD
	handleLCD(lcd1);

	// atribui o conteúdo a variável texto
	strcpy(texto, "Teste TCC II");

	// copia o conteúdo da variável para a propriedade
	strcpy(lcd1.Linha1, texto);

	// escreve o conteúdo das propriedades no componente LCD
	handleLCD(lcd1);

}
```

Quadro 38– Código traduzido

É possível notar a diferença de tamanho de um código para o outro ressaltando a quantidade de trabalho que foi economizada para escrever um pequeno texto em um componente LCD.

3.3.1.6 Implementação da compilação da linguagem criada

A implementação da compilação da linguagem traduzida é feita através da ferramenta Mikroc e é realizada passando o caminho absoluto do projeto Mikroc para o aplicativo.

Apesar da ferramenta Mikroc não possuir chamada externa através de parâmetros nesta versão 4.0 utilizou-se a técnica de abrir-lo e enviar a mensagem de compilação (Ctrl+F9) utilizando-se do *handle* da janela. Um pequeno aplicativo foi criado para simplificar a chamada do Mikroc pela ferramenta do presente estudo, este aplicativo é chamado de CallApp e se encontra presente no diretório corrente desta ferramenta. O quadro 39 demonstra o comando utilizado na chamada do aplicativo CallApp.

```
private void CompilarMikroC()
{
   SetMensagem("Compilando MikroC...");
   ArquivoParametros parametros = new ArquivoParametros();
   parametros.Abrir();
   string caminhoProjeto = "\"" + this.projeto.Caminho + "\\" +
                                  this.projeto.Nome + "\\" +
                                  Regex.Replace(this.projeto.Nome, " ",
                                  "") + ".dpc" + "\"";

   Process processo = new Process();
   processo.StartInfo = new ProcessStartInfo(Application.StartupPath +
                        "\\CallApp.exe", "\"" + parametros.CaminhoMikroC +
                        "\\mikroCdsPic.exe " + "\" " + caminhoProjeto);
   processo.Start();
   processo.StartInfo.WindowStyle = ProcessWindowStyle.Minimized;
   processo.WaitForExit();
}
```

Quadro 39– Chamada do aplicativo CallApp

O código resumido do aplicativo CallApp que realiza o envio do comando de compilação (Ctrl+F9) é apresentado no quadro 40.

```
string keys = "^{F9}";

APIWin32.ShowWindow(handle, 5);

APIWin32.SetForegroundWindow(handle);

SendKeys.SendWait(keys);

Thread.Sleep(1000);
```

Quadro 40– Código resumido do aplicativo CallApp

Ao realizar a execução das rotinas apresentadas no quadro 40 o Mikroc será aberto e a

compilação será iniciada após o envio das teclas através do comando `System.Windows.Forms.SendKeys.SendWait` e tem como saída o arquivo hexadecimal após a compilação. A figura 43 mostra o progresso da compilação pela ferramenta do presente estudo e pelo software Mikroc.

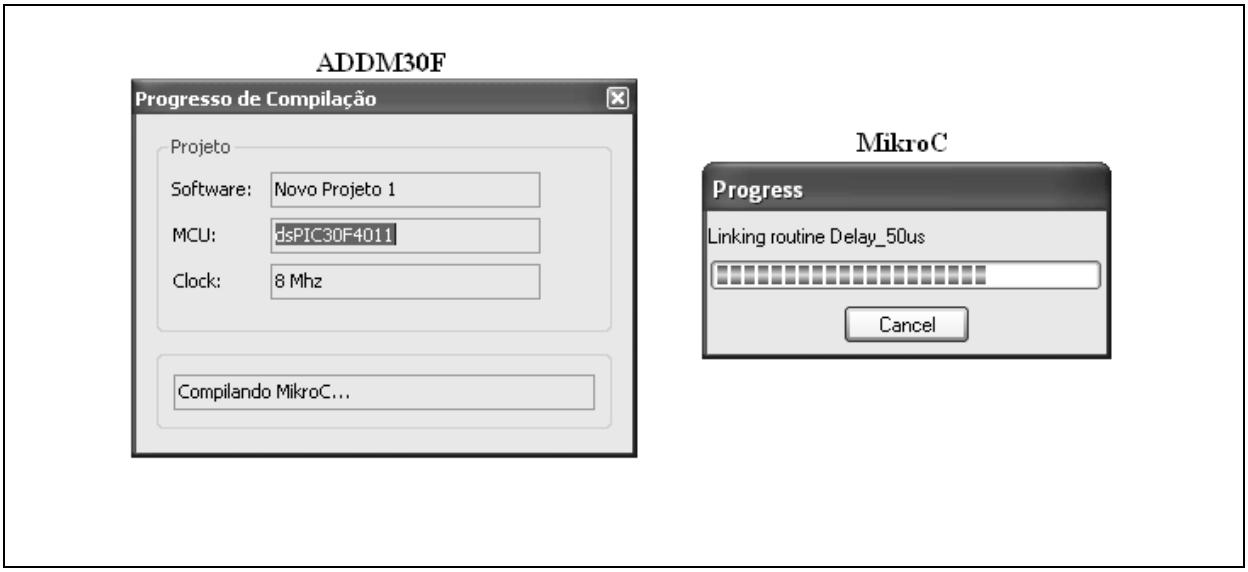


Figura 43 – Barra de progresso da compilação

3.3.1.7 Implementação da gravação do código da linguagem compilada

A gravação de código da ferramenta do presente estudo é realizada o com auxílio da ferramenta Winpic800, passando o caminho do arquivo hexadecimal via parâmetro. Uma tela de detecção é apresentada e em seguida o microcontrolador é apagado e gravado com o novo arquivo hexadecimal. O quadro 41 mostra a rotina de execução do software Winpic800, onde são apresentados os parâmetros corretos da chamada.

```
string comando = "-p -s -a \"" + caminho + "\"";

Process.Start(parametros.CaminhoWinPic800 + "\\WinPic800.exe", comando);
```

Quadro 41– Execução do software Winpic800

A figura 44 demonstra a tela de gravação do arquivo hexadecimal.

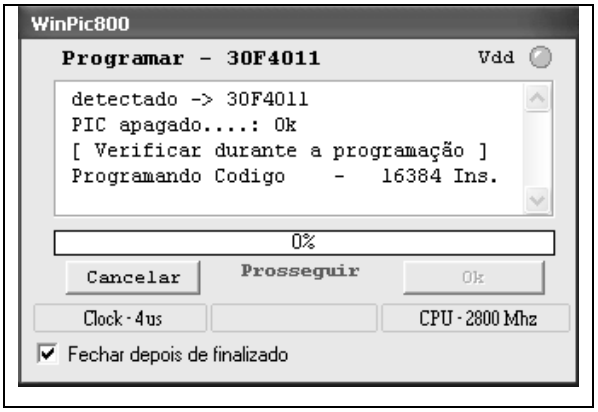


Figura 44 – Tela de gravação do arquivo hexadecimal

3.3.1.8 Implementação do módulo de depuração

O modo de depuração foi desenvolvido apenas em modo visual a fim de mostrar a execução do código fonte, e conta também com uma listagem de variáveis e seus valores, que são atualizados conforme a execução transcorre.

A forma que a ferramenta apresenta as informações em tempo real nos componentes vem da implementação do método `EventoAlterarPropriedade` que pertence a classe `Componente` e é estendido por todos os componentes da ferramenta. Este método é chamado no momento em que a simulação do código altera alguma propriedade de determinado componente.

O quadro 42 exemplifica a implementação deste método para o componente LCD.

```
public override void EventoAlterarPropriedade(
string nome, string valor, Propriedade propriedade, bool debug){
   base.EventoAlterarPropriedade(nome, valor, propriedade, debug);

   switch (nome)
   {
      case "Linha1":
      {
         this.tbLinha1.Text = valor;
      }

      break;

      case "Linha2":
      {
         this.tbLinha2.Text = valor;
      }

      break;
   }
}
```

Quadro 42 – Código do método `EventoAlterarPropriedade`

3.3.2 Operacionalidade da implementação

A presente seção aborda a operacionalidade da implementação em nível do usuário, são apresentadas as formas de operar a ferramenta dispondo de pequenas explicações de forma seqüencial sobre o funcionamento da mesma.

3.3.2.1 Configurando parâmetros

Para realizar a configuração dos parâmetros da ferramenta, o menu `Editar/Parametros` deve ser executado e a tela de configuração será aberta. A figura 45 mostra a tela de configuração.

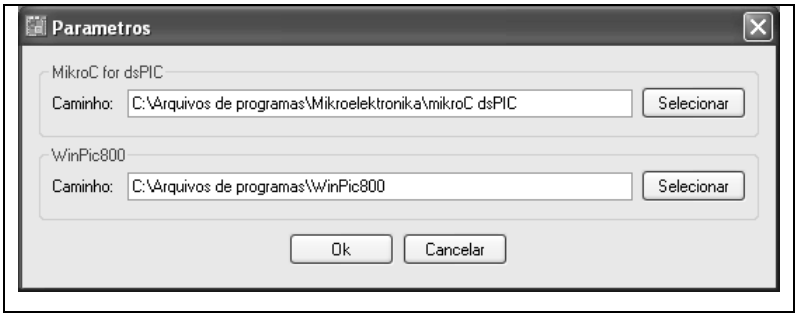


Figura 45 – Tela de parâmetros

Os botões selecionar devem ser executados para realizar a escolha dos diretórios para cada aplicativo em especifico e o botão `Ok` depois de pressionado concluirá o processo criando um arquivo chamado `parâmetros.cfg` que armazenará esta configuração.

3.3.2.2 Criando e editando projetos

A criação de projetos é invocada através do menu `Projeto/Novo Projeto` e permite a inserção de dados relevantes no processo de criação de um sistema embarcado. A figura 46 apresenta a tela de criação de projetos.

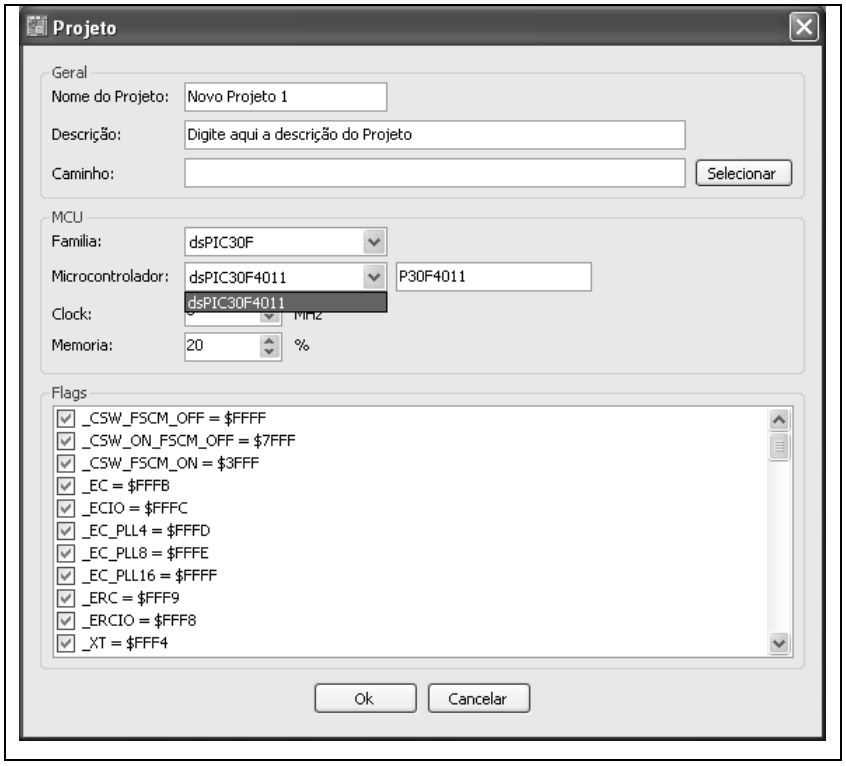


Figura 46 – Tela de criação e edição de projetos

A tela possibilita a seleção da família de microcontroladores, modelo, *clock* de trabalho, quantidade de memória interna reservada para ROM e os *flags*. Após clicar no botão `Ok` o projeto será salvo no diretório especificado e o projeto será apresentado na listagem de projetos da ferramenta. A figura 47 mostra esta funcionalidade.

Figura 47 – Listagem de projetos

3.3.2.3 Componentes e *drag and drop*

Após realizar o duplo clique no projeto presente na listagem a área de componentes é habilitada para uso. A figura 48 mostra a paleta de componentes sendo utilizada com *drag and drop*.

Figura 48 – Paleta de componentes *drag and drop*

Ao clicar no botão correspondente ao componente basta realizar um novo clique na área visual que o componente escolhido será construído e possibilitará a manipulação. A figura 49 mostra o componente LCD construído.

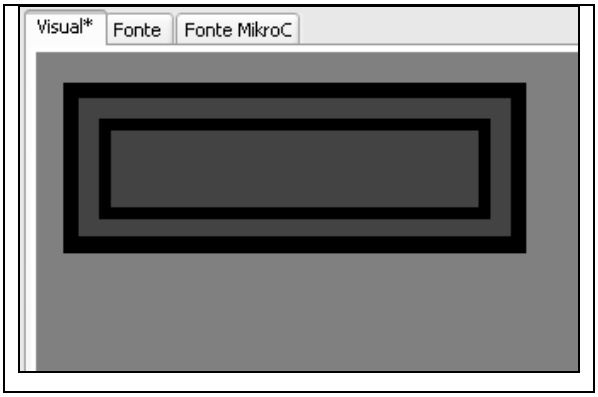


Figura 49 – Componente LCD construído

3.3.2.4 Propriedades dos componentes

As propriedades dos componentes permitem a manipulação visual e de características especificas dos componentes. A figura 50 demonstra esta funcionalidade para o componente LCD.

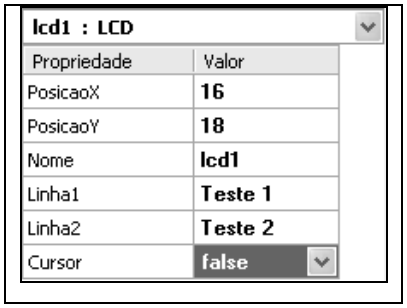


Figura 50 – Propriedades dos componentes

3.3.2.5 Receptor serial

O receptor serial é um pequeno módulo criado a fim de monitorar as recepções via porta serial, permitindo realizar a escolha da porta a ser monitorada. É executada através do menu `Acessórios/Receptor Serial`. A figura 51 demonstra a funcionalidade.

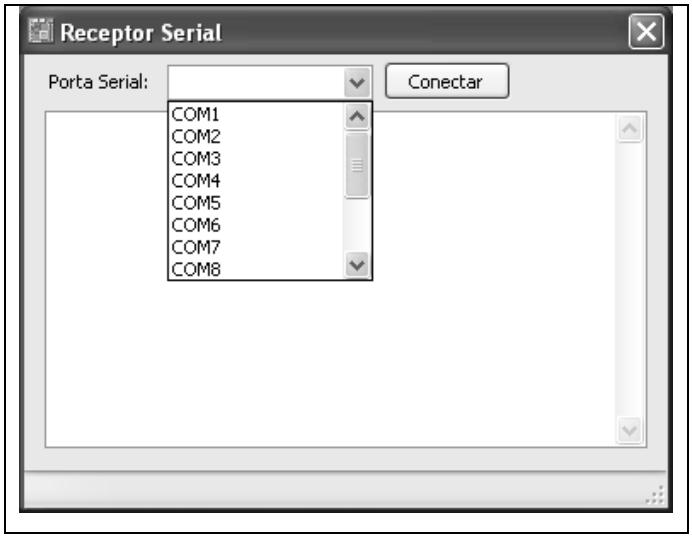


Figura 51 – Receptor serial

Ao escolher determinada porta e clicar no botão conectar, caso a conexão seja bem sucedida, os dados serão inseridos na área especifica que é a caixa de texto presente, facilitando assim os testes envolvendo recepções via porta serial.

3.3.2.6 Menu construir e exibir

O menu construir contém as funcionalidades responsáveis por construir, gravar e executar os sistemas embarcados criados com a ferramenta. A figura 52 mostra a imagem do menu.

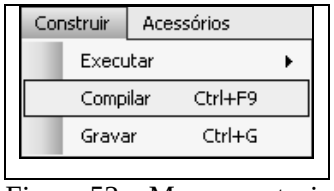


Figura 52 – Menu construir

O menu exibir é uma simples funcionalidade criada para tornar as áreas da ferramenta visíveis ou invisíveis para melhor visualização da mesma. A figura 53 mostra o menu.

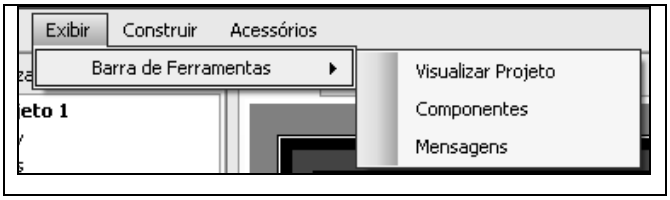


Figura 53 – Menu exibir

3.3.2.7 Protótipo 1

O primeiro protótipo demonstra o desenvolvimento de um pequeno sistema embarcado que simula a utilização do componente LCD escrevendo uma informação no *display*. Um componente LCD será disposto na área visual com as propriedades `Linha1` e `Linha2` possuindo as informações `TCC II` e `Protótipo 1`, respectivamente. A figura 54 demonstra o componente LCD preenchido com as informações alteradas via propriedade.

Figura 54 – Protótipo 1 LCD

Uma pequena alteração via código será realizada na `Linha1` concatenando o texto `TCC II` ao texto `LCD`. O quadro 43 mostra o código proposto.

```
variaveis
{
	texto: literal;
}


programa principal
{
	texto = lcd1.Linha1;
	texto.concatenar(" LCD");
	lcd1.Linha1 = texto;
}
```

Quadro 43 – Código proposto protótipo 1

É possível notar a utilização do procedimento `concatenar` que recebe como parâmetro o texto `LCD` e é aplicado ao conteúdo presente na `Linha1`. A figura 55 demonstra a alteração após a simulação.

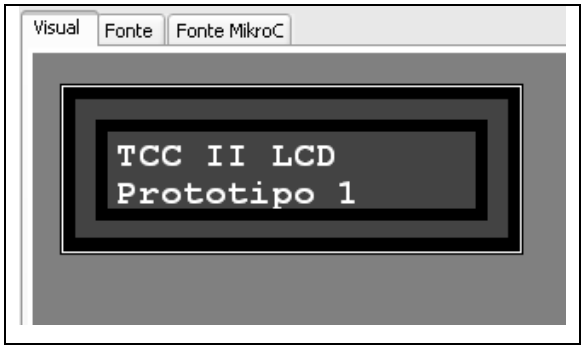


Figura 55 – Protótipo 1 LCD alterado

O quadro 44 demonstra o código traduzido para linguagem C.

```
#include "Utils.h"
#include "LCD.h"
#include "LED7.h"
#include "SERIAL.h"
#include "TIMER.h"

typedef unsigned short boolean;
typedef char string[20];

#define true 0
#define false 1

LCD lcd1;
string texto;

void main()
{
      installLCD(lcd1);
      strcpy(lcd1.Linha1, "TCC II");
      strcpy(lcd1.Linha2, "Prototipo 1");
      lcd1.Cursor = false;
      handleLCD(lcd1);

      strcpy(texto, lcd1.Linha1);

      strcat(texto, " LCD");

      strcpy(lcd1.Linha1, texto);
      handleLCD(lcd1);

}
```

Quadro 44 – Código proposto protótipo 1 traduzido

3.3.2.8 Protótipo 2

O segundo protótipo demonstra o desenvolvimento de um pequeno sistema embarcado que simula a utilização do componente LED7 escrevendo uma informação numérica no

display. Um componente LED7 será disposto na área visual com a propriedade `Valor` possuindo a informação `2222`. A figura 56 demonstra o componente LED7 preenchido com a informação.

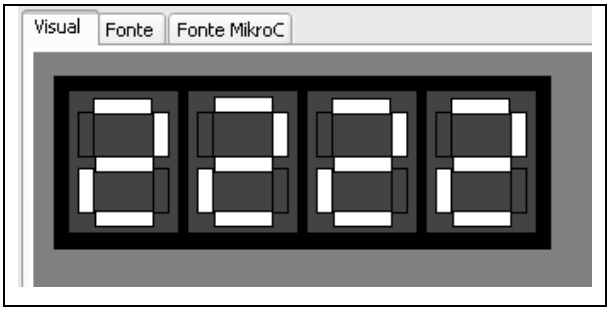


Figura 56 – Protótipo 2 LED7

A alteração que será realizada é o incremento de dez unidades ao número 2222 já presente na propriedade do componente. O quadro 45 demonstra o código proposto.

```
variaveis
{

       numero : inteiro;

}


programa principal
{

       numero = led71.Valor;
       numero.add(10);

       led71.Valor = numero;

}
```

Quadro 45 – Código proposto protótipo 2

O código proposto apresentado demonstra a captura do valor da propriedade `Valor` e o incremento deste número utilizando o procedimento `add` tendo como parâmetro o algarismo dez. A figura 57 demonstra a alteração de forma visual.

Figura 57 – Protótipo 2 LED7 alterado

O quadro 46 demonstra o código traduzido para linguagem C.

```
#include "Utils.h"
#include "LCD.h"
#include "LED7.h"
#include "SERIAL.h"
#include "TIMER.h"

typedef unsigned short boolean;
typedef char string[20];

#define true 0
#define false 1

LED7 led71;
int numero;

void main()
{
      installLED7(led71);
      led71.Valor = 2222;
      handleLED7(led71);


      numero = led71.Valor;

      numero = numero + 10;

      led71.Valor = numero;
      handleLED7(led71);

}
```

Quadro 46 – Código proposto protótipo 2 traduzido

3.4 RESULTADOS E DISCUSSÃO

Os resultados obtidos tendo como base os requisitos em relação aos componentes de entrada e saída do microcontrolador, apresentaram resultados na depuração da ferramenta idênticos ao ambiente de testes Kit Dspicgenios. Uma característica observada em relação ao LED de sete segmentos é o fato da operação deste componente ser por varredura de LEDs. É possível perceber o componente apagando a informação caso a mesma não seja freqüentemente atualizada, devido ao fato deste mesmo componente não possuir memória interna para manter os dados.

O componente TIMER possui as mesmas características encontradas no ambiente de testes, sendo possível utilizar as mesmas parametrizações de tempo, tanto na simulação como

na execução do código no microcontrolador, sendo assim o *delay* ocasionado pela utilização deste componente, pode ser percebido de forma idêntica nos dois ambientes.

O componente LCD possui funções muito específicas na API do software Mikroc, sendo possível englobar todas as características possíveis para este componente, apenas uma limitação causa certo entrave no desenvolvimento, devido ao fato de que existe uma API para desligar o cursor do LCD e não para ligá-lo.

O componente SERIAL, que é relacionado às operações com a porta serial RS232, possui também funções na API do Mikroc sendo possível realizar o envio de dados bem como configurar os *bits* de tamanho.

Todas estas características apresentadas foram transformadas conceitualmente na forma de propriedades de componente na linguagem da ferramenta, simplificando o manuseio destes componentes.

Em relação à tradução de código fonte, procurou-se deixar o código traduzido o mais limpo e simples possível, aliando à endentação de código a simplicidade do mesmo. Os componentes de entrada e saída do microcontrolador escolhidos para englobar a paleta de componentes são adequados, mas não suficientes, ressaltando que para pequenos projetos de testes utilizando microcontroladores a ferramenta atende plenamente.

Em relação aos trabalhos correlatos, as funcionalidades contidas na ferramenta Zexus++ se assemelham muito a ferramenta do presente estudo, já a ferramenta Flowchart Editor possui poucas similaridades. O quadro 47 mostra o comparativo.

Funcionalidade	ADDM30F	Zexus++	Flowchart Editor
Editor visual	Sim	Sim	Sim
Edição de código fonte	Sim	Sim	Não
Paleta de componentes	Sim	Sim	Sim
Editor de propriedades	Sim	Sim	Não
Lista de projetos	Sim	Não	Não
Tradução de código	Sim	Sim	Sim
Compilação	Sim	Sim	Não
Gravação no dispositivo	Sim	Não	Não
Endentação de código	Sim	Sim	Não
Sintaxe colorida	Sim	Não	Não

Quadro 47– Comparativo de funcionalidades

4 CONCLUSÕES

O processo de desenvolvimento para microcontroladores tem seu foco acentuado onde a automatização de processos necessita reduzir o trabalho empregado na conclusão de projetos específicos, que demandam uma carga consideravelmente grande de tempo e de complexidade, o que invariavelmente elevam os custos de tais projetos.

Tendo em vista as situações acima descritas, a ferramenta do presente trabalho vai ao encontro desta importante necessidade e possibilita um grau maior de agilidade ao desenvolvedor, desprendendo o mesmo de certos detalhes que demandam uma curva de aprendizado morosa.

O desenvolvimento da ferramenta ADDM30F cumpriu o que se esperava em relação ao que foi inicialmente proposto, contando apenas com alguns detalhes negativos durante o desenvolvimento da mesma.

A ferramenta Mikroc dificultou o desenvolvimento em alguns aspectos como a falta de algumas funções auxiliares que tiveram de ser criadas, bem como a falta da funcionalidade de chamada externa através de parâmetros, sendo assim a compilação do código fonte ficou aquém do que se realmente esperava, porém com o requisito atendido.

Já a ferramenta Winpic800 atendeu plenamente por possuir uma chamada externa simples de ser executada.

Em relação à criação da linguagem de programação, a utilização da ferramenta GALS foi de suma importância no desenvolvimento deste trabalho, por tornar a especificação dos analisadores léxicos e sintáticos muito simples e intuitiva, produzindo um código totalmente limpo e fácil de ser portado para outras linguagens, como foi o caso do presente estudo onde o código Java foi transformado em C#.

A ferramenta possui uma linguagem de programação limitada contando apenas com poucas funções de manipulação e um conjunto de componentes pequeno, porém de uma utilização simples e clara capaz de realizar o desenvolvimento para microcontroladores de forma intuitiva e simples.

4.1 EXTENSÕES

Como sugestão de trabalhos futuros sugere-se desenvolver um número maior de componentes de entrada e saída do microcontrolador para que a ferramenta tenha uma maior capacidade em relação a projetos de sistemas embarcados, indica-se também a criação de mais funções auxiliares na linguagem de programação e a implementação da depuração com a utilização de *breakpoints*.

Recomenda-se a substituição ou atualização da ferramenta de compilação Mikroc por não disponibilizar chamada externa e por fim inserir componentes que aproveitem a tecnologia digital do microcontrolador DSPIC30F4011.

REFERÊNCIAS BIBLIOGRÁFICAS

AHO, Alfred V.; SETHI, Ravi; ULLMAN, Jeffrey D. **Compiladores**: princípios, técnicas e ferramentas. Tradução Daniel de Ariosto Pinto. Rio de Janeiro: LTC, 1995.

ATMEL. **Microcontrollers**. [S.l.]. 2011. Disponível em: <http://www.atmel.com/products/overview_mcu.asp?source=cms&category_id=163&source=super_footer>. Acesso em: 18 out. 2011.

FONT, Sisco B. **WinPic800**: software for PIC programming. [S.l.], [2010?]. Disponível em: <http://www.winpic800.com/>. Acesso em: 18 mar. 2011.

FONTANIVE, Drayton R. **Protótipo de editor fluxoprogramático com interface visual para geração de código para o microcontrolador PIC16C84 da Microchip Technology**. 1999. 79 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

GESSER, Carlos E. **GALS**: gerador de analisadores léxicos e sintáticos. 2003. 71 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis.

MATIC, Nebojsa; ANDRIC, Dragan. **Microcontroladores PIC**: para iniciantes também. Tradução Alberto Jerônimo. [S.l.]: eBook, 2000.

MICROCHIP. **DSPIC30F4011**. [S.l.]. 2005. Disponível em: <http://ww1.microchip.com/downloads/en/devicedoc/70135c.pdf>. Acesso em: 05 apr. 2011.

_____. **PIC16C84**. [S.l.]. 1997. Disponível em: <http://ww1.microchip.com/downloads/en/devicedoc/30445c.pdf>. Acesso em: 05 apr. 2011.

MICROGENIOS. **Kit dsPICgenios**: kit de desenvolvimento profissional família dsPIC30F4011. São Paulo, 2009. Disponível em: <http://www.microgenios.com.br/news/manuais_kits/kitdspicgenios_manual.pdf>. Acesso em: 18 out. 2011.

MIKROELEKTRONIKA. **MikroC**: C compiler for Microchip dsPIC 30/33 and PIC 24 microcontrollers. [S.l.], 1998. Disponível em: <http://www.mikroe.com/eng/products/view/7/mikroc-pro-for-pic/>. Acesso em: 18 mar. 2011.

PARMA, Gustavo G. **Porta serial**: pratica 14. Minas Gerais, 2006. Disponível em: <http://www.cpdee.ufmg.br/~parma/spp/Prat14.pdf> Acesso em: 18 out. 2011.

PEREIRA, Fabio. **Microcontroladores PIC**: técnicas avançadas. 3. ed. São Paulo: Érica, 2002.

PISKE, Otavio R.; SEIDEL, Fábio A. **Rapid application development**. Curitiba, 2006. Disponível em: <http://www.angusyoung.org/arquivos/artigos/rad.pdf>. Acesso em: 19 mar. 2011.

PRICE, Ana M. A.; TOSCANI, Simão S. **Implementação de linguagens de programação**: compiladores. Porto Alegre: Sagra Luzzato, 2001.

RIBEIRO, Marco A. **Automação industrial**. 4. ed. Salvador: Tek Treinamento & Consultoria Ltda, 2001.

SILVA, Renato A. **Programando microcontroladores PIC**: linguagem C. São Paulo: Ensino Profissional, 2006.

SOUZA NETO, Oscar N. **Análise comparativa das metodologias de desenvolvimento de softwares tradicionais e ágeis**. 2004. 60 f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação) – Centro de Ciências Exatas e Tecnologia, Universidade da Amazônia, Belém. Disponível em: <http://www.cci.unama.br/margalho/portaltcc/tcc2004/oscar.PDF>. Acesso em: 01 abr. 2011.

ZIMERMANN, Jean C. **Ferramenta para geração de código C++ para a plataforma Zexus**. 2005. 72 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

APÊNDICE A – Código fonte da biblioteca `LCD.h`

O quadro 48 demonstra o código fonte da biblioteca `LCD.h` que é anexado ao código fonte traduzido.

```
typedef struct
{
   char Linha1[20];
   char Linha2[20];
   unsigned short Cursor;
} LCD;

void installLCD(LCD lcd)
{
   ADPCFG = 0xFFFF;
   TRISE = 0;
   TRISB = 0;
    Lcd_Custom_Config(&PORTB, 3,2,1,0, &PORTE, 4,0,5);
    Lcd_custom_Cmd(Lcd_CLEAR);
}

void handleLCD(LCD lcd)
{
   Lcd_custom_Cmd(Lcd_CLEAR
   if (lcd.Cursor == 1)
   {
      Lcd_custom_Cmd(Lcd_CURSOR_OFF);
   }
   Lcd_Custom_Out(1,1, lcd.Linha1);
   Lcd_Custom_Out(2,1, lcd.Linha2);
}
```

Quadro 48– Código fonte da biblioteca `LCD.h`

APÊNDICE B – Código fonte da biblioteca `LED7.h`

O quadro 49 demonstra o código fonte da biblioteca `LED7.h` que é anexado ao código fonte traduzido.

```
typedef struct
{
   int Valor;
} LED7;

void installLED7(LED7 led7)
{
   ADPCFG = 0xFFFF;
   TRISD  = 0;
   PORTD  = 0;
   TRISB  = 0;
   PORTF  = 0;
   PORTD = 0x0;
   PORTB = 0x0;
   led7.Valor = 0;
}
unsigned short convert(unsigned short number)
{
   switch (number)
   {
      case 0 : return 0x3F;
      case 1 : return 0x06;
      case 2 : return 0x5B;
      case 3 : return 0x4F;
      case 4 : return 0x66;
      case 5 : return 0x6D;
      case 6 : return 0x7D;
      case 7 : return 0x07;
      case 8 : return 0x7F;
      case 9 : return 0x6F;
    }
}
```

```
void handleLED7(LED7 led7)
{
   unsigned int num = led7.Valor;
   int calc = 0;
   calc = num / 1000;
   PORTD.F0 = 1;
   PORTB = convert(calc);
   Delay_ms(2);
   PORTD.F0 = 0;
   calc = num % 1000;
   calc = calc / 100;
   PORTD.F1 = 1;
   PORTB = convert(calc);
   Delay_ms(2);
   PORTD.F1 = 0;
   calc = num % 100;
   calc = calc / 10;
   PORTB = convert(calc);
   PORTD.F2 = 1;
   Delay_ms(2);
   PORTD.F2 = 0;
   calc = num % 10;
   PORTB = convert(calc);
   PORTD.F3 = 1;
   Delay_ms(2);
   PORTD.F3 = 0;
}
```

Quadro 49– Código fonte da biblioteca `LED7.h`

APÊNDICE C – Código fonte da biblioteca `SERIAL.h`

O quadro 50 demonstra o código fonte da biblioteca `SERIAL.h` que é anexado ao código fonte traduzido.

```
typedef struct
{
   int Bits;
   char Dados[20];
} SERIAL;

void installSERIAL(SERIAL serial)
{
   Uart2_Init(9600);
}

void handleSERIAL(SERIAL serial)
{
   int index;
   Uart2_Init(serial.Bits);

   if (strcmp(serial.Dados, "") != 0)
   {
      for (index = 0; index < strlen(serial.Dados); index++)
      {
         Uart2_Write_Char(serial.Dados[index]);
      }

      strcpy(serial.Dados, "");
   }
}
```

Quadro 50– Código fonte da biblioteca `SERIAL.h`

APÊNDICE D – Código fonte da biblioteca `TIMER.h`

O quadro 51 demonstra o código fonte da biblioteca `TIMER.h` que é anexado ao código fonte traduzido.

```
typedef struct
{
   int Intervalo;
   unsigned short Ativo;
} TIMER;

void installTIMER(TIMER timer)
{

}

void handleTIMER(TIMER timer)
{
   int index;
   int interval;

   if (timer.Ativo == 0)
   {
      interval = timer.Intervalo / 1000;

      for (index = 0; index < interval; index++)
      {
         Delay_1sec();
      }
   }
}
```

Quadro 51– Código fonte da biblioteca `TIMER.h`