DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Desenvolvimento de Firmware para Nova Linha de Inversores de Frequência

Monografia submetida à Universidade Federal de Santa Catarina

como requisito para a aprovação da disciplina:

DAS 5511: Projeto de Fim de Curso

Luísa Faraco Meneghel

Florianópolis, Março de 2013

Desenvolvimento de Firmware para Nova Linha de Inversores de Frequência

Luísa Faraco Meneghel

Esta monografia foi julgada no contexto da disciplina

DAS 5511: Projeto de Fim de Curso

e aprovada na sua forma final pelo

Curso de Engenharia de Controle e Automação

Prof. Rômulo Silva de Oliveira

_____

Agradecimentos

Gostaria primeiramente de agradecer à WEG pela oportunidade, e ao Setor de Desenvolvimento de Inversores de Frequência de Baixa Tensão por me acolher tão bem. Tenho certeza de que meu trabalho nada mais é do que reflexo do quanto me senti confortável em meu ambiente de trabalho.

Aproveito este espaço para agradecer aos meus amigos mais dedicados, e àqueles que nem tanto também. À minha família, por todos os esforços despendidos para que esse momento fosse possível. Aos que mesmo à distância fizeram toda a diferença. E principalmente aos que sempre souberam a importância que tem um abraço.

Finalmente, dedico esse trabalho ao Centro Acadêmico de Engenharia de Controle e Automação, que me ensinou em dois anos e meio mais do que eu jamais pensei que pudesse aprender na minha vida.

Resumo

Este projeto foi desenvolvido na WEG Equipamentos Elétricos, uma das maiores empresas do país, situada em Jaraguá do Sul, fabricante de motores, transformadores, geradores, drives, e diversos outros produtos. A WEG, fundada há mais de 50 anos, atua no mercado mundial nas mais diversas áreas, desde a fabricação de tintas até subestações completas.

Os objetivos principais deste trabalho são o estudo, o desenvolvimento e a documentação de uma nova estrutura de dados para uma nova linha de inversores de frequência da WEG, e o estudo e a análise da linguagem de script Lua para uso também nessa nova linha de inversores da WEG. Esses dois trabalhos, como será explicado posteriormente, ocorrem paralelamente, visto que tem como objetivo seu funcionamento conjunto.

Com foco nessas duas vertentes, este documento explicará como foram estudadas diferentes estruturas de dados, e demonstrará os motivos para escolha de um dos tipos de estruturas para a utilização com a nova linha de inversores, assim como sua implementação. Além disso este documento mostrará as análises que foram feitas sobre a linguagem Lua e as conclusões que podem ser tiradas dessas análises.

Abstract

This project has been developed at WEG Industries, one of the greatest companies in Brazil, located at Jaraguá do Sul, manufacturer of products like motors, transformers, power generators, drives, and a great many more. Founded more than fifty years ago, WEG operates in the global market in several different areas, from the manufacturing of paints to turn-key substations.

The main goals of this work are the study, development and documentation of a new data structure for a new line of WEG frequency drives, and the study and analysis of the Lua scripting language, also to be used in this new line of drives. These two works, as will be explained afterwards, are to be done side by side, seing as they have as a common goal their joint operation.

With these two aspects in mind, this document shall explain how the different data structures were studied, and shall demonstrate the reasons for the choice of data structure made for the new line of drives, as well as its development. Furthermore, this document will show the analysis made about the Lua scripting language, and the conclusions drawn from these analysis.

Sumário

Lista de Figuras

Capítulo 1: Introdução

Neste trabalho busca-se expor o início da construção de firmware para nova linha de inversores de frequência da WEG. A necessidade do desenvolvimento desse projeto vem da observação de que é comum que nas atuais linhas de inversores sejam utilizados firmwares com trechos de código desenvolvidos há anos, em cima dos quais muitos desenvolvedores tem trabalhado. Apesar de demonstrar eficácia por muito tempo, esse tipo de código pode acabar por se tornar oneroso.

Uma característica comum de programas empregados na indústria é a sua *reusabilidade*. Isso está fundamentado na necessidade de aproveitar ao máximo o tempo despendido no seu desenvolvimento, e de evitar que seja gasto tempo desenvolvendo diversas vezes trechos de código com funcionalidades similares.

Porém esse princípio da reutilização de software precisa ser tratado com cautela. Mesmo que bem planejada desde o início do desenvolvimento do código, essa reutilização está sujeita à falha humana, e aos poucos as aplicações começam a apresentar erros que ninguém consegue explicar. Isso é inevitável, principalmente quanto maior o número de pessoas trabalhando no desenvolvimento da aplicação. Tendo isso em mente, podemos observar que com o tempo esse código reutilizado passa a trazer mais inconvenientes que facilidades.

Por esse motivo e muitos outros, se vê necessária uma renovação quase completa do código utilizado para uma aplicação, a cada tantos anos. Para isso deve haver muito planejamento e organização, para que os novos códigos a serem desenvolvidos possam ser aproveitados por uma grande fatia de tempo. Deve-se levar em consideração todos os pontos negativos do código anterior, tudo que causou problemas desde o desenvolvimento até a execução do software, para que seja possível corrigir ou atenuar esses pontos no código que está por vir.

O presente trabalho pode ser representado por duas principais vertentes: o estudo, projeto e desenvolvimento de nova uma estrutura de dados para os parâmetros

Figura 1.1: Exemplos de Parâmetros do CFW11

Parâmetro	Descrição	Faixa de valores	Padrão	Ajuste do usuário	Propriedades	Grupos	Pág.
P0440	Ganho Prop. Id	0.00 a 1.99	0.50		PM	91	21-9
P0441	Ganho Integral Id	0.000 a 1.999	0.005		PM	91	21-9
P0520	Ganho Proporc. PID	0.000 a 7.999	1.000		-	46	20-10
P0521	Ganho Integral PID	0.000 a 7.999	0.043		-	46	20-10
P0522	Ganho Diferencial PID	0.000 a 3.499	0.000		-	46	20-10
P0523	Tempo de Rampa do PID	0.0 a 999.0 s	3.0 s		-	46	20-11
P0524	Sel.Realimentação PID	0 = AI1 (P0231) 1 = AI2 (P0236) 2 = AI3 (P0241) 3 = AI4 (P0246)	1 = AI2 (P0236)		CFG	38, 46	20-12

das novas linhas de inversores de frequência da WEG, e a implementação e teste de um interpretador Lua dentro do código em C em um microcontrolador a ser utilizado nesses inversores.

O estudo da nova estrutura de dados necessita levar em consideração duas coisas: a estrutura já existente nas linhas de inversores atualmente comercializadas, e as restrições e requisitos da nova estrutura. A primeira por haver necessidade de que os inversores da nova linha sejam razoavelmente compatíveis com as linhas de inversores já existentes, além de poder manter o “conforto” do consumidor que já era acostumado com as estruturas de dados anteriores. Isso é importante, pois uma reestruturação de firmware que não leve isso em conta pode inviabilizar a comercialização da nova linha de inversores, por ser necessário que o cliente capacite e treine pessoal para trabalhar com os novos inversores, e também por tornar necessário que se atualize possivelmente todos os inversores de uma fábrica para que eles sejam capazes de estabelecer qualquer tipo de comunicação com o inversor da linha nova. A segunda é apenas natural, pois não se inicia o planejamento de uma nova estrutura sem primeiro pensar no que se quer dela. É nessa etapa que são levados em consideração os pontos positivos e negativos levantados quanto à estrutura de dados já existente.

A estrutura de dados de que tratamos era, até o momento, constituída dos parâmetros dos inversores. Alguns exemplos desses parâmetros podem ser vistos na figura 1.1 [4]. Esses parâmetros servem para configuração do inversor, ditando o seu modo de funcionamento. Eles representam desde o tipo de controle que deve ser utilizado nos inversores, passando pela data atual, e até guardando registros de falhas e alarmes.

Como é possível verificar, o número de parâmetros é muito grande, e até então existia pouca classificação entre eles, o que torna o uso do equipamento uma ta-

refa complicada. Um dos motivos para o desenvolvimento de uma nova estrutura é a separação dos parâmetros em grupos, ou menus, de modo a facilitar a busca e o entendimento dos mesmos. Os maiores problemas da estrutura já existente se encontravam principalmente na incompatibilidade entre diferentes linhas de inversores, e, o que é ainda pior, entre diferentes versões de firmware de cada uma das linhas.

Os dados de cada inversor eram, até então, organizados em uma grande tabela (codificada como um vetor), e cada parâmetro recebia seu índice nessa tabela. A cada nova versão de firmware, ao incluir um parâmetro ou excluir um parâmetro, esse índice viria a variar, tornando absolutamente incompatíveis inversores da mesma linha que tivessem diferentes versões de firmware. Existem maneiras de evitar essas alterações de índices, mas elas não são infalíveis e requerem muita cautela do desenvolvedor, o que nem sempre é fácil de se fazer cumprir. Vê-se então a necessidade de rever a forma como os dados são estruturados.

Por outro lado, temos a linguagem de script Lua. Lua é uma linguagem de programação extremamente útil, leve, e facilmente embarcável. Ela pode ser utilizada para complementar, ou aprimorar, as funcionalidades de uma aplicação de modo simples e eficaz. A linguagem Lua possui muitas características interessantes em diversos pontos de vista. Entre elas podemos citar, de acordo com [6]:

- sua interpretação dinâmica, que é realizada sem muito custo computacional e que permite que sejam executados trechos de código gerados dinamicamente no mesmo ambiente de execução do programa;
- sua gerência automática de memória dinâmica, o que é interessante em sistemas embarcados, pois esses costumam possuir grandes limitações de memória;
- sua grande portabilidade, que a torna muito conveniente, permitindo que ela funcione sobre as mais variadas plataformas (desde os sistemas operacionais proprietários como Windows, passando por sistemas móveis como Android, até sistemas embarcados como é o caso de algumas aplicações para Lego Mindstorms);
- e finalmente, como software livre e distribuído sobre a licença do MIT [1], Lua pode ser usada para qualquer propósito sem custo algum, incluindo propósitos comerciais.

A partir dessas características, podemos antecipar que Lua pode ser muito útil

para configurar o acesso aos dados que serão definidos na nova estrutura a ser desenvolvida. Pode-se usar a linguagem para realizar a troca de dados do inversor com a sua HMI com maior facilidade, e pouco custo computacional. Torna-se, então, claro que o desenvolvimento da nova estrutura de dados deve caminhar paralelamente ao desenvolvimento da aplicação Lua para o novo firmware.

Este projeto pretende, portanto, desenvolver uma estrutura de dados conivente com os requisitos para o firmware da nova linha de inversores da WEG, e que mantenha a capacidade de incorporar a linguagem de script Lua. No próximo capítulo será esclarecido este trabalho no contexto da empresa, além de serem descritas algumas das ferramentas utilizadas no desenvolvimento dele. A seguir, será estabelecida a base teórica utilizada no estudo da estrutura de dados em questão, com uma análise sobre as estruturas estudadas. Nos capítulos 4 e 5 será exposta a implementação dessa estrutura. No capítulo 6 pode ser encontrada a contextualização e análise da linguagem Lua em relação a este projeto. O último capítulo demonstra uma análise final do trabalho feito, e apresenta sugestões para as próximas etapas do desenvolvimento dessa nova linha de inversores.

Capítulo 2: A WEG Equipamentos Elétricos

O trabalho presentemente relatado foi desenvolvido na WEG Equipamentos Elétricos, divisão Drives e Controls. A empresa tem sede situada em Jaraguá do Sul, Santa Catarina, onde a empresa foi criada, porém possui diversas unidades espalhadas pelo mundo. Hoje uma das líderes do mercado no seu ramo, além de uma das maiores empresas do país, a WEG contava em 2011 com um total de 24.580 colaboradores diretos e um faturamento de R$ 6.130 milhões, números que não param de crescer. Com uma imensa gama de produtos comercializados, a WEG atua nos seguimentos Tintas, Automação, Motores, Energia, e Transmissão e Distribuição. A WEG procura oferecer não apenas produtos, e sim soluções, sempre prezando pela simplicidade.

A história da WEG tem início com a produção de motores elétricos em 1961, em um espaço alugado que futuramente se tornaria o Museu WEG. Seus fundadores, Werner Voigt, Eggon da Silva e Geraldo Werninghaus, batizaram a empresa com as iniciais de seus nomes, formando a palavra que em alemão significa “caminho”. Com dificuldade de vendas no início do empreendimento, devido à falta de confiança dos consumidores em uma empresa tão recente e desconhecida, seus fundadores chegaram a viajar a São Paulo levando o motor da WEG nos braços para bater na porta de clientes em potencial. Com o espírito alemão do trabalho árduo, e sempre prezando pelo respeito àqueles que empregava, a WEG hoje se consolida cada vez mais como gigante do mercado. Nas palavras de Eggon da Silva:

> “Se faltam máquinas, você pode comprá-las; se não há dinheiro, você toma emprestado; mas homens você não pode comprar nem pedir emprestado; e homens motivados por uma ideia são a base do êxito.”

A WEG Automação foi criada na metade da década de 80. A divisão de Drives

e Controls (WDC), incorporada na WEG Automação, produz inversores de frequência, CLPs, no-breaks, servoacionamentos, soft-starters, entre outros, além de soluções em software para esses equipamentos. No Setor de Desenvolvimento de Inversores de Frequência de Baixa Tensão são projetados e desenvolvidos os inversores das linhas CFW trabalhando com tensão de alimentação de até 690V, e os motordrives MW. Esse processo passa desde a idealização de uma nova linha, até finalização de protótipos e envio para a produção inicial, mas não para por aí. O setor de desenvolvimento é responsável por auxiliar os colaboradores do setor de Assistência Técnica da WEG a sanar as dúvidas e necessidades dos clientes na manutenção dos produtos comercializados, além de auxiliar os colaboradores de Vendas a se certificar de que o cliente receba o produto que melhor se encaixa às suas necessidades.

2.1: A Nova Linha de Inversores de Frequência

Os inversores de frequência da WEG são acionamentos de velocidade variável, com tecnologia de última geração, para motores de indução trifásicos. Eles são encontrados em uma variedade de modelos para atender as necessidades específicas de cada aplicação nos mais variados setores industriais.

A solução que a WEG busca oferecer com a nova linha de inversores sobre a qual esse trabalho acontece tem como objetivo apresentar à indústria um novo equipamento que não apenas atende às necessidades do cliente, mas que também apresenta maior conveniência ao usuário do inversor. Sem deixar de ter o desempenho e a confiabilidade que já são parte dos produtos da WEG, procura-se desenvolver um produto mais *user friendly*, e capaz de desempenhar um maior número de tarefas que os inversores atuais.

Do ponto de vista da autora deste trabalho, a WEG está buscando, no projeto dessa nova linha de inversores, não só lançar um novo produto no mercado, mas também revitalizar o seu processo de desenvolvimento de inversores. A empresa mostra uma busca por métodos melhores, e que facilitem o desenvolvimento e o aperfeiçoamento dos produtos, o que só pode trazer benefícios.

Para o desenvolvimento desse trabalho foram utilizadas diversas ferramentas. A mais importante delas foi o *Eclipse IDE for C/C++ Developers*, utilizado tanto para desenvolvimento da estrutura de dados em C que será apresentada, quanto para desenvolvimento de aplicações utilizando a linguagem Lua. Sob orientação do Eng.

Ricardo Wiggers, a autora desse trabalho foi responsável por todos os itens de desenvolvimento nele exposto.

Capítulo 3: Estudo de Estruturas de Dados

Antes de iniciar o estudo de qual tipo estrutura de dados é mais indicada para uso na nova linha de inversores, é necessário fazer um estudo prévio de como os parâmetros serão estruturados, para que se possa buscar o melhor método de implementar essa estrutura de parâmetros. Para isso será explicado primeiramente como são organizados os parâmetros dos inversores de frequência hoje, e depois será mostrado como queremos que a nova estrutura de dados organize seus parâmetros.

3.1: Os parâmetros atuais do CFW11

Até então os inversores da WEG trabalhavam com estruturas de dados onde cada parâmetro possui um número identificador, que também é utilizado como endereço para comunicações em rede entre os inversores. A interface homem-máquina dos inversores atuais permite que se busque os parâmetros em alguns menus, como mostrado na figura 3.1.

Essa classificação apresentada, apesar de eficaz até o momento, se torna insuficiente sob um olhar mais atencioso. Ela é pouco intuitiva ao usuário, e por isso precisa ser melhorada. Atualmente os parâmetros existentes trazem em suas propriedades valores que indicam a quais menus esses parâmetros pertencem.

Se quisermos que a classificação dos parâmetros seja intuitiva o suficiente para que o usuário possa encontrar com facilidade o parâmetro que busca sem precisar procurá-lo em meio a todos os outros parâmetros no manual, precisamos repensar essa estrutura. Tornaram-se então objeto de estudo os parâmetros existentes nos inversores, de modo a projetar uma melhor classificação em menus para eles. No CFW11, tomado como exemplo, existem em torno de 1.000 parâmetros, entre parâmetros de

Figura 3.1: Estrutura de Menus do CFW11

Nível 0	Nível 1		Nível 2		Nível 3	
Monitoração	00	TODOS PARÂMETROS				
	01	GRUPOS PARÂMETROS	20	Rampas		
			21	Refer. Velocidade		
			22	Limites Velocidade		
			23	Controle V/f		
			24	Curva V/f Ajust.		
			25	Controle VVW		
			26	Lim. Corrente V/f		
			27	Lim. Barram. CC V/f		
			28	Frenag. Reostática		
			29	Controle Vetorial	90	Regulador Veloc.
					91	Regulador Corrente
					92	Regulador Fluxo
					93	Controle I/F
					94	Auto-Ajuste
					95	Lim. Corr. Torque
					96	Regulador Barr. CC
			30	HMI		
			31	Comando Local		
			32	Comando Remoto		
			33	Comando a 3 Fios		
			34	Com. Avança/Retorna		
			35	Lógica de Parada		
			36	Multispeed		
			37	Potenc. Eletrônico		
			38	Entradas Analógic.		
			39	Saídas Analógicas		
			40	Entradas Digitais		
			41	Saídas Digitais		
			42	Dados do Inversor		
			43	Dados do Motor		
			44	FlyStart/RideThru		
			45	Proteções		
			46	Regulador PID		
			47	Frenagem CC		
			48	Pular Velocidade		
			49	Comunicação	110	Config. Local/Rem
					111	Estados/Comandos
					112	CANopen/DeviceNet
					113	Serial RS232/485
					114	Anybus
					115	Profibus DP
			50	SoftPLC		
			51	PLC		
			52	Função Trace		
	02	START-UP ORIENTADO				
	03	PARÂM. ALTERADOS				
	04	APLICAÇÃO BÁSICA				
	05	AUTO-AJUSTE				
	06	PARÂMETROS BACKUP				
	07	CONFIGURAÇÃO I/O	38	Entradas Analógic.		
			39	Saídas Analógicas		
			40	Entradas Digitais		
			41	Saídas Digitais		
	08	HISTÓRICO FALHAS				
	09	PARÂMETROS LEITURA				

leitura, parâmetros de escrita, parâmetros de sistema (esses últimos invisíveis ao usuário), etc..

3.2: A nova estrutura de parâmetros

Após longa observação e estudo sobre os parâmetros já existentes, foi sugerida estrutura de menus para a nova linha de inversores demonstrada na figura 3.2.

Essa nova estrutura apresenta uma nova forma de dispor dos parâmetros do inversor. Como uma das principais propostas dessa nova estrutura, podemos citar a criação de um ID estruturado para cada parâmetro, que serve para referenciá-lo facilmente ao seu menu de classificação. Sabemos dessa maneira que, por exemplo, o

Figura 3.2: Sugestão para nova estrutura de menus

Struc. ID	Nível 0	Struc. ID	Nível 1	Struc. ID	Nível 2	Struc. ID	Nível 3	Struc. ID	Nível 4
0.0.0.0.0	**Menu Principal**	1.0.0.0.0	**Status**	1.1.0.0.0	Medições				
				1.2.0.0.0	I/O				
				1.3.0.0.0	CFWXXX				
				1.4.0.0.0	Temperaturas				
				1.5.0.0.0	Comunicações				
				1.6.0.0.0	SoftPLC				
		2.0.0.0.0	**Diagnósticos**	2.1.0.0.0	Falhas				
				2.2.0.0.0	Alarmes				
				2.3.0.0.0	Motor ON				
				2.4.0.0.0	Controle de horas				
				2.5.0.0.0	Parâmetros alterados				
		3.0.0.0.0	**Configurações**	3.1.0.0.0	Local/Remoto				
				3.2.0.0.0	Dados do motor				
				3.3.0.0.0	**Controle**	3.3.1.0.0	Auto-Ajuste		
						3.3.2.0.0	Tipo de controle		
						3.3.3.0.0	**Parâmetros comuns**	3.3.3.1.0	Rampas
								3.3.3.2.0	Referência de velocidade
								3.3.3.3.0	Limites de velocidade
								3.3.3.4.0	Multispeed
								3.3.3.5.0	Lógica de parada
								3.3.3.6.0	Pular velocidade
								3.3.3.7.0	Busca de zero encoder
						3.3.4.0.0	Parâmetros de leitura		
						3.3.5.0.0	**Controle Escalar**	3.3.5.1.0	V/f
								3.3.5.2.0	V/f Ajustável
						3.3.6.0.0	Controle VVW		
						3.3.7.0.0	**Controle Vetorial**	3.3.7.1.0	Sensorless
								3.3.7.2.0	Com encoder
				3.4.0.0.0	**I/O**	3.4.1.0.0	Entradas digitais		
						3.4.2.0.0	Saídas digitais		
						3.4.3.0.0	Entradas analógicas		
						3.4.4.0.0	Saídas analógicas		
				3.5.0.0.0	**Proteções**	3.5.1.0.0	Prot. de tensão		
						3.5.2.0.0	Prot. de corrente		
						3.5.3.0.0	Prot. de tempo		
						3.5.4.0.0	Prot. térmica do motor		
						3.5.5.0.0	Prot. térmica do inversor		
						3.5.6.0.0	Prot. de sobrecarga do motor		
						3.5.7.0.0	Prot. de sobrecarga do inversor		
				3.6.0.0.0	**HMI**	3.6.1.0.0	Senha e idioma		
						3.6.2.0.0	Data e hora		
						3.6.3.0.0	Tela principal		
				3.7.0.0.0	**Funções especiais**	3.7.1.0.0	Regulador PID		
						3.7.2.0.0	Controle Vetorial PM		
						3.7.3.0.0	Trace		
						3.7.4.0.0	Frenagem CC		
						3.7.5.0.0	Frenagem Reostática		
						3.7.6.0.0	Flying Start/Ride-Through		
				3.8.0.0.0	**Comunicações**	3.8.1.0.0	Comandos		
						3.8.2.0.0	Serial RS232/485		
						3.8.3.0.0	CANopen/DeviceNet		
						3.8.4.0.0	Anybus-CC		
						3.8.5.0.0	Profibus-DP		
				3.9.0.0.0	Dados do CFWXXX				
				3.10.0.0.0	SoftPLC				
				3.11.0.0.0	Backup				
		4.0.0.0.0	**Start-up orientado**						
		5.0.0.0.0	**Lista de Parâmetros**						

parâmetro identificado por 1.2.1 indicará o status de uma entrada/saída do inversor. Dessa forma não é necessário consultar o manual ou que o usuário guarde em sua memória o número referente ao parâmetro que é buscado, como é comum nos inversores atuais. Porém não se deseja perder essa possibilidade, visto que muitos clientes antigos dos inversores da WEG já tem o costume de usar essa abordagem.

A estrutura mostrada na figura 3.2, mesmo tendo sido cuidadosamente estudada, constitui apenas uma sugestão, e ainda é passível de alterações. Apesar disso ela satisfaz a necessidade que se tinha de uma estrutura de parâmetros, e é em cima dela que trabalhamos desse ponto em diante.

3.3: Estruturas de Dados

No desenvolvimento da estrutura de dados para o projeto da nova linha de inversores de frequência se mostrou necessária a realização de um estudo do melhor método para armazenamento e acesso aos dados do inversor. A estrutura de dados necessária deveria cumprir os seguintes requisitos:

1. Permitir que seja rápido o acesso aos parâmetros, a partir dos seus respectivos endereços de rede. Esse endereço de rede deve ser compatível com os endereços dos parâmetros já existentes em outras linhas de inversores, de modo a não inviabilizar a comunicação entre eles;
2. Permitir que seja implementada a classificação dos parâmetros sugerida, de forma que eles estejam melhor organizados e dessa maneira que a busca de parâmetros pelo usuário torne-se mais intuitiva;
3. Permitir que seja feito o acesso aos parâmetros por um endereço estruturado, que será atribuído a cada parâmetro de acordo com a classificação em grupos do mesmo;
4. Ao navegar o menu dos parâmetros, a estrutura deve permitir que se memorize a posição da navegação a cada nível;
5. Ser capaz de armazenar as diversas propriedades de cada um dos parâmetros, além dos seus respectivos valores;
6. Permitir que a cada nova versão de firmware posse ser garantida a compatibilidade com as versões anteriores com facilidade.

Diante dessas especificações levantadas para a nova estrutura de dados, o próximo passo é analisar com cautela as possíveis estruturas, para ser feita a escolha de qual delas melhor as acomoda. Utilizando como base o livro escrito por Kyle Loudon [5], são avaliadas então as seguintes estruturas de dados:

- Tabelas
- Listas Encadeadas
- Tabelas Hash
- Árvores Binárias

3.3.1: Tabelas

O primeiro dos métodos de armazenamento de dados estudados foi a utilização de tabelas. *Arrays* com índices inteiros conhecidos são as estruturas de dados utilizadas nas versões anteriores de firmware de inversores da WEG. Ao definir-se um parâmetro seu índice era definido como um número inteiro que seria definido como algo conhecido, como mostrado na figura 3.3.

Figura 3.3: Exemplo de definição dos índices da tabela para cada parâmetro

```
/*****************************************************************************/
/* Defines com os indices dos parametros e seus respectivos dados            */
/* Indice pra tabela kParameterPointers[]                                    */
/*                                                                           */
/*****************************************************************************/
#define P1_1_1          0
#define P1_1_2          1
#define P1_1_3          2
#define P1_1_4          3
#define P1_1_5          4
#define P1_1_6          5
#define P3_8_2_1        6
#define P3_8_2_2        7
#define P3_8_2_3        8
#define P3_8_2_4        9
#define P3_8_2_5        10
```

A implementação dessa estrutura de dados na nova linha de inversores pode ser imaginada da seguinte forma: cada menu seria composto de um vetor de ponteiros nulos compostos por, primeiramente, um ponteiro para uma estrutura contendo as propriedades desse menu, seguido de ponteiros para cada um dos parâmetros desse menu, ou dos submenus que pertençam a esse menu. No caso de o menu ser composto de submenus, os ponteiros desse vetor indicariam os vetores de ponteiros nulos referentes a cada submenu, e no caso de o menu ser composto por parâmetros, os ponteiros desse vetor indicariam os respectivos objetos, que seriam obtidos de uma tabela por meio de índice conhecido. Os ponteiros desses vetores precisam ser de tipo nulo (*void*) para que não haja necessidade de criar diferentes tipos de objetos para serem declarados menus ou parâmetros. Um exemplo de menus utilizando esse método se encontra na figura 3.4.

Como se pode ver na figura 3.4, o menu Status é composto de submenus (Measurements, I/O, etc). Desse modo, seu vetor de ponteiros nulos aponta primeiro para uma estrutura que contém informações sobre suas propriedades, e depois para os vetores de ponteiros nulos que correspondem a cada um de seus submenus. Já

Figura 3.4: Sugestão de implementação de tabela como estrutura de dados

```
775  const TMENUPROPERTIES kPropertiesMenuStatus =
776  {
777          {1, 0, 0, 0}
778  };
779  const void* kObjListMenuStatus[] =
780  {
781      &kPropertiesMenuStatus,                 //menu properties, followed by objects list
782      &kObjListMenuMeasurements,
783      &kObjListMenuStatusIO,
784      &kObjListMenuCFWXXX,
785      &kObjListMenuTemperatures,
786      &kObjListMenuStatusCommunications,
787      &kObjListMenuStatusSoftPLC
788  };
789
790
791  const TMENUPROPERTIES kPropertiesMenuMeasurements =
792  {
793          {1, 1, 0, 0}
794  };
795  const void* kObjListMenuMeasurements[] =
796  {
797      &kPropertiesMenuMeasurements,           //menu properties, followed by objects list
798      &kParameterPointers[P1_1_1],
799      &kParameterPointers[P1_1_2],
800      &kParameterPointers[P1_1_3],
801      &kParameterPointers[P1_1_4],
802      &kParameterPointers[P1_1_5],
803      &kParameterPointers[P1_1_6]
```

o menu Measurements é composto de parâmetros, e portanto seu vetor de ponteiros nulos é composto por um ponteiro para uma estrutura que contém informações sobre suas propriedades, e depois para cada os ponteiros referentes a cada um de seus parâmetros dentro da tabela de parâmetros do inversor. Os valores “P_1_1_1”, “P_1_1_2”, etc, são definidos como índices da tabela, como já mostrado anteriormente.

Para utilizar essa estrutura, é necessário que cada menu possua informação sobre os endereços de seus respectivos parâmetros dentro da tabela no momento de escrita do código (armazenada na definição de *P_X_X_X*). Isso torna difícil a interação entre diferentes versões de firmware. Caso seja necessária a remoção, inserção ou alteração de algum parâmetro na tabela de parâmetros, a alteração nos índices de cada parâmetro na tabela pode tornar incompatíveis as diferentes versões. Desse modo não seria possível ser feita, por exemplo, a cópia dos parâmetros de um inversor para outro de mesmo modelo, porém de diferente versão de firmware. Esse método, portanto, torna difícil o cumprimento da sexta restrição citada, que é das mais cruciais. Passamos então ao estudo do próximo método.

3.3.2: Listas Encadeadas

Outro método estudado de estruturação dos dados do inversor foi o método das listas encadeadas. Esse tipo de estrutura é um dos mais fundamentais que existem, e consiste em elementos ligados, ou encadeados, em uma ordem específica. Ele apresenta vantagem sobre as tabelas por serem muito mais eficientes ao criar e remover elementos. As listas encadeadas são, na prática, um conjunto de elementos que apresenta duas partes: uma que possui seus dados, e uma que aponta, pelo menos, para o próximo elemento da lista. Existem também as listas duplamente encadeadas, onde além de um elemento apontar para o próximo elemento da lista, ele aponta também para o anterior. Esse tipo de lista é interessante quando se precisa otimizar buscas dentro da mesma, pois permite que a lista também seja percorrida de trás pra frente.

Antes de pensar em qual tipo de lista seria interessante trabalharmos, vamos pensar nos modos que usaríamos para armazenar os dados referentes aos parâmetros dos inversores em listas encadeadas. Existem duas maneiras principais que poderiam ser utilizadas para implementar esse tipo de estrutura.

Uma delas seria a criação de uma única lista, contendo todos os parâmetros do inversor. Como já mencionado, os inversores tem um número muito grande de parâmetros. Essa lista poderia trazer grandes problemas de performance ao inversor. Independente da maneira que a lista fosse organizada, ela seria muito grande. A busca por parâmetros poderia ser extremamente lenta, visto que seria necessário percorrer toda a lista até o parâmetro desejado. Esse tipo de lista, portanto, fere a primeira restrição dada à estrutura de dados para a nova linha de inversores.

Outra implementação é a de criar uma lista encadeada para os parâmetros de cada menu. Isso diminuiria significativamente o tamanho das listas, porém seriam necessárias em torno de 60 listas encadeadas. Nesse caso, cada menu teria um ponteiro apenas para sua respectiva lista de parâmetros, e ao inicializar o software do inversor seriam inseridos cada um dos parâmetros na sua respectiva lista.

Essa implementação permitiria que fossem inseridos parâmetros no fim de cada lista, no início, ou em uma posição específica a partir de funções previamente definidas. Cada objeto da lista teria as seguintes informações: endereço de rede, endereço estruturado, tipo do dado, ponteiro para o próximo objeto da lista, propriedades do objeto e dados. Poderiam ser feitas buscas em cada uma das listas a partir de qualquer uma dessas informações. Continuaria sendo trabalhoso, porém, procurar por um

parâmetro se não é conhecida a lista em que ele se encontra, pois seria necessário percorrer todas as listas existentes para encontrar um parâmetro que estivesse no final da última lista em que fosse feita a busca. Voltamos portanto ao problema levantado na primeira interpretação sugerida para listas encadeadas.

Pode-se concluir, portanto, que não seria possível garantir as especificações do início desta seção utilizando listas encadeadas.

3.3.3: Tabelas Hash

Um conceito que levanta o interesse quando se fala em estruturas de dados relativamente grandes onde serão realizadas buscas de elementos é o das tabelas hash. Vamos então fazer um breve estudo delas. De acordo com Loudon, tabelas hash consistem de vetores onde os dados são acessados por meio de um índice especial chamado “chave”, e a idéia principal é de estabelecer um mapeamento entre um conjunto de possíveis chaves e posições em um vetor utilizando uma função hash. Enquanto os tipos de chaves utilizados podem variar, os valores retornados pela função hash serão sempre inteiros.

A maior preocupação quando se trabalha com tabelas hash é a busca da melhor função hash para cada aplicação. A situação é ideal quando uma função hash garante que duas chaves distintas não resultarão no mesmo valor hash. Porém nesse caso a memória ocupada pela tabela hash seria muito maior que o que é realmente necessário. Existem, porém, maneiras de se lidar com as chamadas “colisões”, nome que é dado quando duas chaves distintas retornam o mesmo valor hash. Uma dessas maneiras é colocar em cada elemento da tabela hash uma lista encadeada, onde se buscaria então o dado desejado.

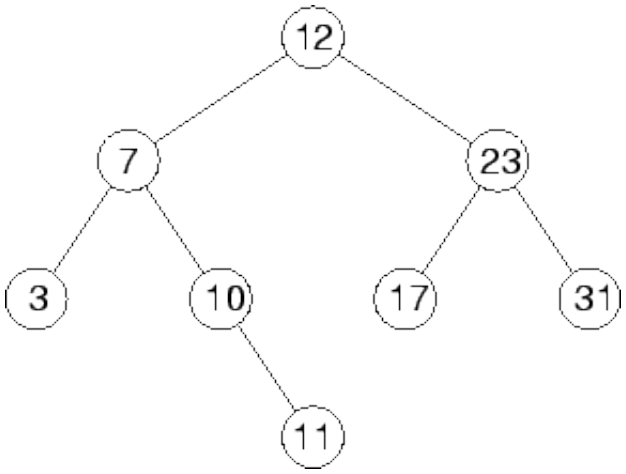
As tabelas hash fornecem então um método rápido de acesso aos dados, apesar de ser necessário (como nas tabelas com índices definidos) que sejam armazenados os valores de chave de cada elemento da tabela. Mas antes de sair em busca da função hash ideal para essa situação, tarefa que pode se mostrar complexa, vamos analisar o próximo tipo de estruturas de dados, que é relativamente simples.

3.3.4: Árvores Binárias

Árvores binárias são estruturas de dados particularmente eficientes quando se deseja fazer buscas sobre os dados. Cada nó da árvore possui uma parte constituída

por seus dados, um valor usado como ID do objeto, um ponteiro indicando o nó à direita, e outro indicando o nó à esquerda. O nó à direita de cada elemento sempre terá ID de valor maior que o ID do elemento, enquanto que o nó da esquerda sempre terá ID de valor menor. O nó localizado no topo da árvore é chamado de raiz, enquanto que os nós ligados à direita e esquerda de cada nó são chamados de filhos. Cada nó, com exceção da raiz, possui um nó pai, que é aquele localizado diretamente acima dele. A performance de uma árvore binária está intrinsecamente relacionada à sua altura, que é definida como o número de níveis em que existem nós. A figura 3.5 exemplifica uma árvore binária, de altura 3, onde o nó com ID 12 é a raiz e os nós 7 e 23 são seus filhos.

Figura 3.5: Exemplo de Árvore Binária



A busca de elementos na árvore deve ser feita sempre focada no valor de ID do elemento. A partir deste valor de identificação pode-se fazer buscas em árvores com muitos elementos utilizando um pequeno número de iterações. A altura de uma árvore binária representa a distância do a raiz ao nó mais distante dela, sendo 0 a altura de uma árvore cujo único elemento é o nó raiz. O número máximo de elementos de uma árvore binária de altura h segue a fórmula:

$$n = 2^{h+1} - 1 \tag{3.1}$$

Portanto sabemos que uma árvore de altura 9 pode chegar a 1023 elementos, e que no pior caso ao fazer uma busca nessa árvore demoraremos apenas 9 iterações para chegar ao elemento desejado. Porém isso só acontece quando a árvore encontra-se corretamente balanceada. Isso significa que os elementos devem estar bem distribuídos sobre a árvore para que as buscas sobre ela sejam o mais eficiente possível.

A figura 3.6 representa uma árvore onde os elementos não estão bem distribuídos, e dessa maneira a árvore se comporta da mesma forma que uma lista encadeada.

Figura 3.6: Exemplo de árvore não-balanceada

Uma árvore balanceada, segundo Loudon, é aquela em que todos os nós sem filhos encontram-se no último nível, ou então nos últimos dois níveis e o penúltimo nível se encontra cheio. Por exemplo, a árvore da figura 3.5 encontra-se balanceada pois todos os nós sem filhos se encontram nos últimos dois níveis, e o penúltimo se encontra cheio. Para garantir que uma árvore binária esteja sempre balanceada existem diversos métodos. Um tipo relativamente simples de árvore auto-balanceada é a Árvore AVL, que será explicada mais adiante neste documento.

3.4: Conclusão do estudo

Neste capítulo foram estudados alguns tipos de estruturas de dados, de forma a avaliar qual delas deveria ser empregada na nova linha de inversores da WEG. Em vista das especificações lançadas na terceira seção deste capítulo, podemos ver que as árvores binárias (com auto-balanceamento) alcançam com simplicidade o nosso objetivo, enquanto as tabelas hash podem acabar não sendo tão eficazes se a função hash escolhida não for adequada. Abaixo se encontram as maneiras encontradas de satisfazer cada um dos seis objetivos traçados anteriormente.

1. O acesso aos parâmetros é suficientemente rápido se garantimos o balanceamento da árvore, e utilizamos o endereço de rede de cada parâmetro como ID do objeto.

2. Podemos criar objetos para representar cada um dos menus e submenus sugeridos, e ligá-los a cada um dos objetos a eles pertencentes.

3. É possível que se crie uma estrutura para objeto, na qual podemos adicionar um segundo valor de identificação, sendo esse um ID estruturado, que identifica o objeto perante a classificação de menus sugerida.

4. Quando da navegação no menu, é possível que o sistema armazene informações sobre os objetos pelos quais está passando.

5. A árvore binária terá em cada nó um objeto, e cada objeto terá em sua estrutura as propriedades e valores referentes aos parâmetros a que eles se referem.

6. As novas versões de firmware precisam apenas não alterar os endereços de rede de cada parâmetro para manter a compatibilidade entre si. Como a árvore usará deste endereço para fazer buscas em seus objetos, qualquer um dos outros dados do elemento que vier a ser alterado não influenciará na capacidade de acesso ao elemento a partir do endereço de rede.

Capítulo 4: Implementação da Árvore Binária AVL

Depois de escolhido o tipo de estrutura que será utilizada, precisamos partir para a fase de desenvolvimento dessa estrutura. Em primeiro lugar precisa ser definida qual será a forma do objeto que será colocado na árvore. O objeto precisará armazenar diversas propriedades e informações que o parâmetro do inversor precisa ter. Vejamos então o que queremos que esse objeto indique:

- Um endereço de rede, a ser utilizado como ID para a árvore binária;
- Um endereço estruturado, a ser utilizado para navegação em menus pela HMI;
- O dado, ou valor, a ser armazenado no objeto;
- É necessário que cada objeto descrito tenha um campo indicando seu tipo, para que não seja necessário lidar com diversos tipos diferentes de objetos, mas que ainda se possa ter o conhecimento do tipo de objeto com o qual estamos tratando;
- Cada objeto deve apresentar ponteiros para as funções que farão sua leitura e escrita. Como existem grandes diferenças entre os parâmetros do inversor, é necessário implementar diversos métodos para que sejam feitas leitura e escrita sobre os parâmetros, então a melhor forma de garantir a integridade dessas operações é que cada objeto indique como elas devem ser feitas;
- As propriedades particulares de cada parâmetro. Essas propriedades incluem uma vasta gama de informações, como por exemplo se o parâmetro é somente de escrita, se o parâmetro pode ser acessado via rede, quantas casas decimais devem ser utilizadas na sua visualização na HMI, a unidade do parâmetro, entre muitas outras. Isso será discutido melhor adiante.

Figura 4.1: Estrutura dos objetos

```
typedef struct _TObject TObject;

struct _TObject
{
        int                             NetworkID;
        unsigned char   StructuredID[5];
        TObjectType     Type;
        int                     (*Read)(TObject* self, void* data, int element, TInterfaceFormat format, int modifiers);
        int                     (*Write)(TObject* self, void* data, int element, TInterfaceFormat format, int modifiers);
        int                     (*ReadProperty)(TObject* self, void* data, int element);
        int                     (*WriteProperty)(TObject* self, void* data, int element);
        void**                  Properties;
        void**                  Data;
};
```

Tendo em mente esses itens, sugerimos a estrutura da figura 4.1 para os objetos com que trabalharemos desse ponto em diante.

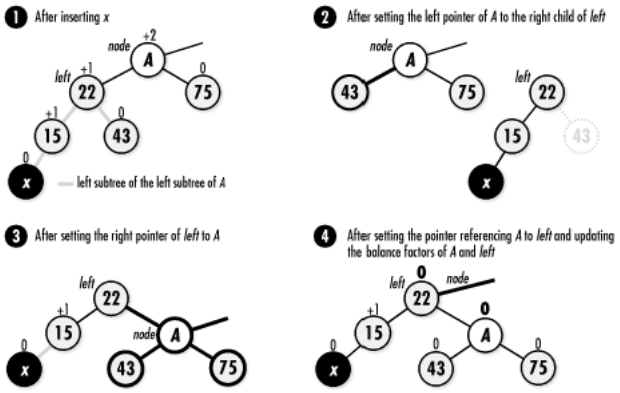
Após designar a forma que o objeto terá, precisamos começar a pensar na árvore em si. Um elemento qualquer de uma árvore binária precisa de três informações: o número identificador do objeto, uma referência ao objeto da esquerda e uma referencia ao objeto da direita. Como precisamos, além disso, que a árvore esteja sempre balanceada, precisamos verificar de que forma isso será feito. Como já mencionado, vamos fazer agora uma breve contextualização sobre Árvores AVL.

4.1: Árvore AVL

A árvore AVL mantém apenas um valor a mais em cada nó[5]. Esse valor é chamado de “Fator AVL”, e representa um índice de balanceamento da árvore. Quando a subárvore à direita de um nó apresenta a mesma altura que a subárvore à esquerda dele, dizemos que a árvore deste nó está balanceada. O fator de balanceamento pode variar entre -1 e 1, onde -1 indica que a árvore está “mais pesada” à direita, 1 indica que árvore está “mais pesada” à esquerda, e 0 indica que a árvore está balanceada. Qualquer valor diferente desses indica que a árvore não está balanceada, e é necessário corrigir isso.

Para fazer essa correção são utilizadas operações chamadas rotações. Cada inserção de um novo membro na árvore atualiza o fator AVL dos nós envolvidos na inserção, e são realizadas as rotações necessárias para manter o balanceamento da árvore. Existem 4 tipos de rotações: as rotações direita-direita, esquerda-esquerda, direita-esquerda e esquerda-direita. Vamos explicar as rotações esquerda-direita e esquerda-esquerda, visto que as rotações direita-esquerda e direita-direita são, res-

Figura 4.2: Rotação Esquerda-Esquerda



pectivamente, simétricas a elas.

4.1.1: Rotação Esquerda-Esquerda

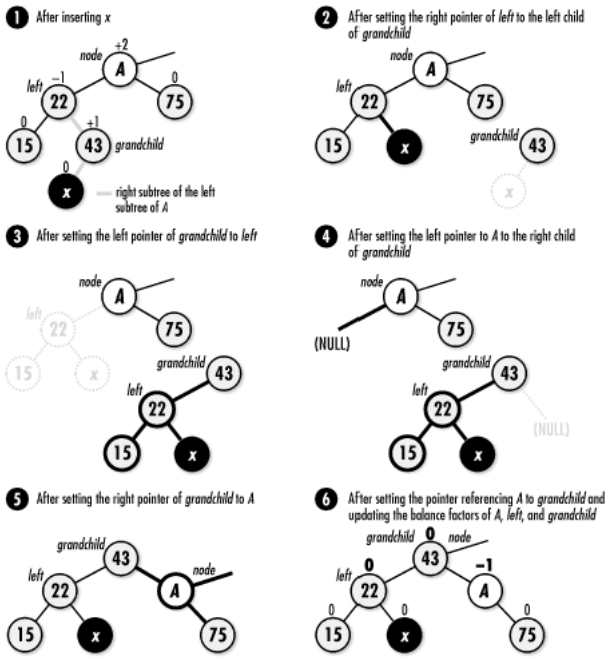
Vamos considerar a inserção de um nó x em uma árvore. Seja um nó A o antecessor mais próximo de x cujo fator AVL tenha se tornado 2. A rotação esquerda-esquerda acontece quando x se encontra na subárvore à esquerda da subárvore esquerda de A. Seja *left* a subárvore esquerda de A. Para realizar a rotação colocamos no braço esquerdo de A o braço direito de *left*, colocamos no braço direito de *left* o elemento A, e colocamos no ponteiro que referenciava A o elemento *left*. Após a rotação os fatores AVL de A e de *left* serão ambos 0, e os fatores AVL dos outros elementos permanecerão inalterados. A figura 4.2 mostra uma representação gráfica da rotação.

4.1.2: Rotação Esquerda-Direita

Essa rotação é feita quando tentamos inserir o nó x na subárvore direita da subárvore esquerda de A. Mais uma vez seja *left* a subárvore esquerda de A, e seja *grandchild* a subárvore direita de *left*. A rotação é feita colocando no braço direito de *left* o elemento à esquerda de *grandchild*, colocando no braço esquerdo de *grandchild* o elemento *left*, colocando no braço esquerdo de A o elemento à direita de *grandchild*, colocando no braço direito de *grandchild* o elemento A, e colocando no ponteiro que referenciava A o elemento *grandchild*, como mostrado na figura 4.3

No caso da rotação esquerda-direita, calcular os novos fatores AVL é um pouco mais complexo, pois depende dos fatores AVL anteriores dos nós. Se o fator AVL

Figura 4.3: Rotação Esquerda-Direita



original de *grandchild* era 1, o novo fator AVL de A é -1 e de *left* é 0. Se o fator AVL original de *grandchild* era 0, os novos fatores AVL de A e de *left* também serão 0. E se o fator AVL original de *grandchild* era -1, o novo fator AVL de A é 0 e de *left* é 1. Em todos os casos o novo fator AVL de *granchild* será 0, e os fatores AVL dos outros elementos são mantidos.

4.2: Construção da Árvore AVL

4.2.1: Definição do Nó

Depois de definido o tipo de árvore binária a ser utilizado no projeto da estrutura de dados, podemos trabalhar na construção da árvore em si. Primeiramente definimos o que queremos que tenha na estrutura do elemento da árvore:

- O objeto em questão;
- Ponteiros para os nós à direita e esquerda;
- Para facilitar a busca de elementos dentro da árvore, vamos incluir em cada nó um ponteiro para seu nó-pai (que será nulo na raiz da árvore);

Figura 4.4: Estrutura do Nó da Árvore AVL

```
typedef struct TreeNode *SearchTree;
struct TreeNode
{
        TObject         Object;
        SearchTree  Left;
        SearchTree  Right;
        SearchTree      ParentNode;
        SearchTree      ParentMenu;
        int                     AvlFactor;
        int                     inUse;
}Nodes[MAX_NODES];
```

- O fator AVL do nó;
- Para fins de facilitar a navegação em menus na HMI, vamos incluir também um ponteiro para o nó cujo objeto é o menu-pai do objeto contido em cada nó (que será nulo quando o objeto em questão for o menu-raiz da HMI).

Além disso, precisamos definir como a memória para cada nó será alocada. Precisamos levar em consideração que essa estrutura de dados será utilizada no firmware do inversor, por meio de um microcontrolador. Em sistemas embarcados, principalmente que continuam em funcionamento por grandes espaços de tempo, alocação dinâmica de memória pode causar problemas muito sérios. O mais relevante deles é o da fragmentação de memória, que pode fazer com que o sistema fique cada vez mais lento, inviabilizando o funcionamento do inversor. Existem métodos de trabalhar com alocação dinâmica de memória de modo a evitar esses problemas, mas a solução mais simples ainda é a de se trabalhar com alocação estática de memória.

Decidimos então utilizar de um vetor para alocar cada um dos nós da árvore. O vetor de nós é criado com um número de elementos grande o suficiente para garantir que não faltará espaço para a alocação de novos nós na árvore, mas sem gerar grande desperdício de memória. Para garantir também que um novo elemento não sobrescreverá um elemento existente quando de sua alocação, adiciona-se um campo a mais na estrutura do elemento de árvore. Esse campo indicará se a posição do array está em uso ou não. Desse modo, no momento da alocação basta verificar esse elemento do nó para saber se a posição do array pode ser ocupada pelo novo elemento.

A forma final, portanto, da estrutura dos nós da árvore é mostrada na figura 4.4.

4.2.2: Função de Inserção

O próximo passo é definir a função que fará a inserção de cada elemento na árvore. Ela deve fazer o ajuste do fator AVL dos nós quando necessário, e deve solicitar que sejam feitas rotações sempre que necessário. Ela deve verificar, como já foi mencionado, que a posição do vetor de nós está livre antes de alocar o objeto naquele nó. Mas principalmente, a função deve cumprir o seguinte algoritmo:

1. Seja uma árvore T, de nó raiz R, cujo nó direito é D e nó esquerdo é E, e um nó N que desejamos inserir em T.
2. Se R for nulo, faça aloque N em R, fim.
3. Se o nó N possui ID menor que o ID de R, redefina R como E, e retorne ao início. Se o ID de N for maior que o ID de R, redefina R como D, e retorne ao início. Se o nó N possui ID igual ao ID de R, fim (o nó já está na árvore).

Esse algoritmo é facilmente satisfeito com uma função recursiva, que recebe como argumento a árvore de parâmetros do inversor, ou simplesmente o nó raiz dessa árvore. Note que a chamada dessa função deve ser feita inicialmente passando como argumento a árvore primária dos argumentos, e não alguma subárvore dela. Além disso, para garantir que cada nó inserido conheça seu nó-pai, a função também deve receber como argumento o ponteiro para o nó-pai, que será nulo quando o argumento for o nó raiz da árvore primária.

Como mencionado anteriormente, queremos que cada nó conheça também o seu menu-pai. Para realizar esse objetivo, precisamos utilizar do ID estruturado do objeto. De posse desse ID estruturado, que é constituído por um array de 5 posições, o que fazemos é ajustar para zero a última das posições do array cujo valor não for zero, e procurar pelo objeto que possua esse valor como seu ID estruturado (essa busca será abordada adiante neste documento). Isso significa que o objeto referente ao menu-pai já deve estar na árvore no momento da inserção de cada objeto. Esta é uma restrição que o desenvolvedor não pode negligenciar, para manter a integridade da árvore projetada.

A função de inserção então foi dividida em duas:

- `SearchTree SInsert( TObject Obj, SearchTree T )`: Uma função a ser chamada quando for inserido cada um dos objetos, que recebe como parâmetros o

objeto a ser inserido, e a árvore dos parâmetros do inversor. Essa função primeiramente checa se o objeto a ser inserido é um menu ou um parâmetro, e se for um parâmetro a função atribui a ele o valor padrão que ele deve ter. Esse valor é uma propriedade de cada parâmetro, e está contido no objeto quando ele é criado. Depois disso, a função chama uma função interna.

- `SearchTree SInsertAvl(TObject Obj, SearchTree T, SearchTree ParentNode)`: Uma função interna, que recebe como parâmetro o objeto já com o valor inicial, a árvore onde ele deve ser inserido, e um ponteiro para o nó-pai do objeto a ser inserido. Essa função interna é recursiva, e quando chamada da função externa recebe como parâmetro para o nó-pai um ponteiro de valor nulo. Essa função primeiro checa se a árvore que recebeu como argumento é nula. Se não for, ela compara o endereço de rede do nó no topo da árvore com o do objeto a ser inserido, e chama a própria função novamente, mandando como argumento a subárvore do lado apropriado, e como nó-pai o nó atual. Isso se repete até que a função encontre com uma árvore de nó raiz nulo. Quando isso acontece, a função insere o objeto naquele ponto. Depois de conseguir inserir o objeto, a função atualiza dos fatores AVL de cada nó pelo qual passou, sinaliza que a árvore precisa checar se está balanceada, e faz com que sejam feitas rotações sempre que necessário.

4.2.3: Funções de Busca

4.2.3.1: Busca por Endereço de Rede

A busca mais intuitiva quando se trata de uma árvore binária é pelo seu próprio valor de identificação, no caso o endereço de rede de cada parâmetro. A busca é feita da maneira que já foi mencionada: se buscamos um elemento de ID x, comparamos ao nó raiz da árvore, e vamos para a direita se x é maior que o ID do nó raiz, e para a esquerda no caso contrário. Quando o nó consultado tiver o mesmo valor de ID que procuramos, a busca termina. Se chegarmos a um nó nulo, sabemos que o elemento procurado não se encontra na árvore.

A função desenvolvida, assim como a função de inserção, foi dividida em duas partes. (`SearchTree SFindByNID( int NetID, SearchTree T )`) recebe um ID e uma árvore qualquer, e então percorre essa árvore procurando o nó mais alto dela. Ao encontrar, é chamada a função interna (`SearchTree SFindByNIDWholeTree ( int Ne-`

tID, SearchTree T)), que recursivamente procura o objeto com o endereço de rede indicado. Se a função chega em um ponto em que o objeto não foi encontrado, ela retorna uma árvore nula, indicando que o objeto não está na árvore. Caso contrário, ela retorna o objeto encontrado.

4.2.3.2: Busca por Endereço Estruturado

A função de busca por endereço estruturado também é dividida em duas partes. Na primeira (SearchTree SFindBySID(unsigned char ID[5], SearchTree T)) busca-se o nó mais alto da árvore, e chama-se a função interna de busca. A função interna de busca (SearchTree SFindBySIDWholeTree(unsigned char ID[5], SearchTree T)), que precisa ser chamada com a árvore inteira de parâmetros como argumento, é recursiva, e passa por cada um dos elementos da árvore à procura daquele cujo ID estruturado é o procurado, indo primeiro sempre ao ramo da direita, depois o da esquerda.

Ao contrário da busca por endereço de rede, essa busca não possui um ID para se guiar sobre a árvore. Ela não sabe se o elemento que procura está mais à direita ou mais à esquerda. Ela precisa procurar em cada um dos elementos existentes até achar. Essa busca, portanto, será mais lenta, porém como a especificação do problema permite uma menor velocidade nesse aspecto da estrutura, ela é válida. A função retorna um ponteiro para o objeto desejado se ele existe na árvore, e retorna um ponteiro de valor nulo se o objeto não existe na árvore.

4.2.3.3: Busca de Máximo e Mínimo

Por definição, o nó com o menor valor de ID de uma árvore binária será sempre o elemento mais à esquerda. Analogamente, o nó com maior valor de ID é aquele que se encontra mais à direita. Desse modo, basta percorrer a árvore até a sua base para encontrar os elementos máximos e mínimos.

Para isso foram desenvolvidas as funções SearchTree SFindMin(SearchTree T) e SearchTree SFindMax(SearchTree T). A função SFindMin é utilizada quando se deleta um objeto da árvore. A função SFindMax não é utilizada na estrutura desenvolvida, porém antecipando que ela pode vir a ser útil ela foi criada.

4.2.3.4: Busca por Objeto

Para facilitar a navegação nos menus, criou-se a função `SearchTree SFindByObject ( TObject Obj, SearchTree T )`, que faz busca por objeto. Essa busca basicamente faz o mesmo que a busca por endereço de rede, porém ela recebe como argumento um objeto, e não um valor de ID. Como a função de navegação nos menus foi desenvolvida para se lembrar do objeto selecionado no momento, a busca por objeto serve apenas como facilitadora.

4.2.3.5: Busca por Nó

Apesar de não fazer exatamente uma busca, a função `TObject TRetrieve( SearchTree S )` é utilizada para obter o objeto contido no nó da árvore passado como parâmetro para a função. Essa função permite que trechos de código possam obter o objeto a partir do nó sem conhecer a estrutura dos elementos da árvore.

4.2.4: Função de retirada

A princípio o inversor não precisaria de funções para remover parâmetros da memória. Os parâmetros seriam criados no momento da inicialização do inversor, e não seriam mais alterados durante o funcionamento dele. Caso o desenvolvedor deseje excluir um parâmetro da memória do inversor em versões posteriores de firmware, ele poderia simplesmente excluir do código a instrução que cria o parâmetro. Porém, em aplicações futuras, operações de retirada de parâmetros podem ser necessárias. Portanto, afim de exercer cautela, é criada uma função de retirada.

A função `SearchTree SDelete( TObject Obj, SearchTree T )` procura o objeto na árvore por seu endereço de rede, e ao encontrá-lo verifica primeiramente se o objeto que se está tentando excluir possui nós filhos. Caso ele não possua, a função apenas o substitui por um valor nulo. Caso ele possua um filho, a função substitui o objeto na árvore pelo filho dele. Caso ele possua dois filhos, ele substitui o objeto na árvore pelo menor objeto da àrvore à direita.

Essa função pode acarretar a perda do balanceamento da árvore.

4.2.5: Funções de Relação entre Objetos-Parâmetros e Objetos-Menus

Como já foi apresentado, todos os objetos do inversor foram criados a partir da mesma estrutura, sejam eles parâmetros ou menus. Cada objeto que representa um parâmetro terá suas propriedades, seus métodos de leitura/escrita e seus dados, referente ao parâmetro a que se refere. Um objeto que procura representar um menu não necessita de propriedades ou métodos de leitura/escrita, e portanto esses elementos da estrutura podem ser nulos. Porém é necessário que seu campo de dados estabeleça uma relação com os parâmetros, ou submenus, que queremos encontrar nesse menu. É necessário, portanto, que sejam criadas funções para estabelecer essas relações.

4.2.5.1: Inserção de Parâmetros em um Menu

Primeiramente vamos explicar a função que insere parâmetros em um menu. A função `TObject * TInsertMenuParameters(TObject Obj, SearchTree T)` recebe um objeto, que deve obrigatoriamente ser um menu, e insere em um vetor todos os objetos que devem estar dentro desse menu. É esse vetor, então, que o ponteiro de dados do menu referenciará. A função deve ser chamada para cada um dos menus e submenus do inversor. Essa função se utiliza de uma outra função para buscar cada um dos parâmetros do menu na árvore. Esse processo é repetido até que a função auxiliar retorne um valor nulo, e então é retornado o vetor com os objetos encontrados.

A função auxiliar que busca os parâmetros do menu em questão na árvore é `TObject TGetMenuParameter(TObject Obj, SearchTree T, int index)`. Ela recebe o objeto que deve conter o menu, a árvore onde procurar os objetos a inserir no menu, e um índice que se refere à posição do objeto no menu. A partir dessas informações a função monta o endereço estruturado do objeto, e procura por ele na árvore. Caso ele não seja encontrado, a função retorna um valor nulo.

Além dessas funções, foram criadas mais duas para um caso especial. Um dos menus que a HMI deve mostrar é o menu “Todos os Parâmetros”. Esse menu deve listar todos os parâmetros existentes no inversor, em ordem crescente de endereço de rede. É necessário, portanto, que o campo de dados do objeto que representa esse menu possua um vetor composto de todos os parâmetros do inversor na ordem desejada. A função `TObject * TFillAllParametersMenu(SearchTree T)` é chamada para realizar essa tarefa. Ela certifica-se de que a árvore passada como argumento

possui como nó raiz o nó mais alto da árvore de parâmetros, chama uma função interna para realizar a busca de todos os parâmetros do inversor, organiza eles na ordem desejada, e retorna o endereço do vetor que contém os parâmetros ordenados.

A função interna `void vFillAllParametersMenuInternal(SearchTree T)` é responsável por fazer a busca na árvore de parâmetros por todos os objetos da árvore cujo tipo não é menu, e os armazena em um vetor global na ordem em que são encontrados.

Com o auxílio dessas funções, no momento da inicialização do software do inversor todos os parâmetros dos menus são inseridos neles com facilidade.

4.3: Síntese

Este capítulo mostrou as rotinas criadas para a manipulação da árvore binária onde estarão inseridos os dados dos inversores da nova linha da WEG. Com as funções aqui demonstradas imagina-se estarem cobertas todas as operações que o desenvolvedor pode desejar realizar sobre elas. Estão garantidas as conexões entre menus e submenus ou parâmetros, o balanceamento razoável da árvore binária (através de rotações durante as inserções dos elementos), e consequentemente a esse balanceamento está garantida também a rapidez da busca de parâmetros por endereço de rede sobre a árvore. Mesmo algumas operações que em um primeiro momento não parecem necessárias estão implementadas, e desse modo se um futuro desenvolvedor realmente não encontrar uso para elas, elas podem ser suprimidas.

É necessário ressaltar as diversas rotinas de busca implementadas sobre a árvore. O desenvolvedor tem à sua disposição buscas muito rápidas, e buscas um pouco mais lentas. As buscas mais rápidas terão como número máximo de recursões a altura da árvore. Entre elas figuram as buscas por endereço de rede, por objeto, e de máximo e mínimo. Já a busca por endereço estruturado terá uma velocidade reduzida, visto que ela percorre a árvore toda, sempre fazendo o mesmo percurso até encontrar o objeto procurado. Além disso se a informação que o usuário possui é sobre um determinado nó da árvore, ele através de uma função de busca obter também o objeto contido nesse nó. Em suma, estão previstas rotinas o suficiente sobre todos os tipos de busca que vemos necessárias até o presente momento.

Capítulo 5: Teste e Análise da Árvore

5.1: Criação de parâmetros e menus

A partir do momento que temos as funções referentes à árvore de parâmetros criadas, precisamos populá-la. Para isso foram escolhidos pelo menos um parâmetro referente a cada menu ou submenu definido na estrutura demonstrada na figura 3.2. Os parâmetros escolhidos são mostrados na figura 5.1. Para cada um desses é necessário criar um objeto `TObject`, com todas as informações que esse objeto deve ter. As informações necessárias para a criação desses objetos foram obtidas nas tabelas de parâmetros dos inversores atuais da WEG.

Para cada objeto precisamos conhecer, como mostrado na figura 4.1, seus endereços estruturado e de rede, seu tipo, e suas propriedades. Dos endereços de rede e estruturado já falamos, mas o tipo e as propriedades de cada parâmetro ainda precisam ser explicados. O tipo do objeto precisa ser conhecido para que possamos saber como tratar do valor do objeto no momento de sua manipulação, ou exibição. Os tipos com que esse projeto trabalha são:

- *float*
- *int 16 bits*
- *int 32 bits*
- *unsigned int 16 bits*
- *unsigned int 32 bits*
- Date
- IP Address
- MACAddress

Figura 5.1: Lista de parâmetros escolhidos para realizar testes na árvore

Net ID	Struc. ID	Parâmetro Exemplo
0003	1.1.1.0.0	Corrente no motor (P0003)
0012	1.2.1.0.0	Estado de DI8 a DI1 (P0012)
0006	1.3.1.0.0	Estado do Inversor (P0006)
0034	1.4.1.0.0	Temperatura Ar Interno (P0034)
0316	1.5.1.0.0	Estado da Interface Serial (P0316)
1000	1.6.1.0.0	Estado da SoftPLC (P1000)
0049	2.1.1.0.0	Falha Atual (P0049)
0048	2.2.1.0.0	Alarme Atual (P0048)
0044	2.3.1.0.0	Contador kWh (P0044)
0042	2.4.1.0.0	Horas Energizado (P0042)
0220	3.1.1.0.0	Seleção Fonte Local/Remoto (P0220)
0400	3.2.1.0.0	Tensão Nominal Motor (P0400)
0408	3.3.1.1.0	Fazer Auto-Ajuste (P0408)
0202	3.3.2.1.0	Tipo de Controle (P0202)
0100	3.3.3.1.1	Tempo de Aceleração (P0100)
0120	3.3.3.2.1	Backup da Referência de Velocidade (P0120)
0132	3.3.3.3.1	Nível Máximo de Sobrevelocidade (P0132)
0124	3.3.3.4.1	Referência 1 Multispeed (P0124)
0125	3.3.3.4.2	Referência 2 Multispeed (P0125)
0217	3.3.3.5.1	Bloqueio por Velocidade Nula (P0217)
0303	3.3.3.6.1	Velocidade Evitada 1 (P0303)
0191	3.3.3.7.1	Busca de Zero do Encoder (P0191)
0001	3.3.4.1	Referência de Velocidade (P0001)
0136	3.3.5.1.1	Boost de Torque Manual (P0136)
0142	3.3.5.2.1	Tensão de Saída Máxima (P0142)
0397	3.3.6.1.0	Compensação de Escorregamento Durante a Regeneração (P0397)
0372	3.3.7.1.1	Corrente de Frenagem CC Sensorless (P0372)
0405	3.3.7.2.1	Número de Pulsos do Encoder (P0405)
0263	3.4.1.1.0	Função da Entrada DI1 (P0263)
0275	3.4.2.1.0	Função da Saída DO1 (P0275)
0231	3.4.3.1.0	Função do Sinal AI1 (P0231)
0251	3.4.4.1.0	Função da Saída AO1 (P0251)
0357	3.5.1.1.0	Tempo de Falta de Fase da Rede (P0357)
0342	3.5.2.1.0	Configuração da Detecção de Corrente Desequilibrada no Motor (P0342)
0340	3.5.3.1.0	Tempo de Auto-Reset (P0340)
0159	3.5.4.1.0	Classe Térmica do Motor (P0159)
0353	3.5.5.1.0	Configuração da Proteção de Sobretemperatura nos IGBTs e no Ar Interno (P0353)
0348	3.5.6.1.0	Configuração da Proteção de Sobrecarga do Motor (P0348)
0350	3.5.7.1.0	Configuração da Proteção de Sobrecarga do Inversor (IGBTs) (P0350)
0200	3.6.1.1.0	Senha (P0200)
0193	3.6.2.1.0	Dia da Semana (P0193)
0205	3.6.3.1.0	Seleção Parâmetro de Leitura 1 (P205)
0040	3.7.1.1.0	Variável de Processo PID (P0040)
0438	3.7.2.1.0	Ganho Proporcional do Regulador de Corrente de Iq (P0438)
0550	3.7.3.1.0	Fonte de Trigger para o Trace (P0550)
0299	3.7.4.1.0	Tempo de Frenagem CC na Partida (P0299)
0153	3.7.5.1.0	Nível de Frenagem Reostática (P0153)
0320	3.7.6.1.0	Flying Start/Ride-Through (P0320)
0682	3.8.1.1.0	Controle Serial/USB (P0682)
0308	3.8.2.1.0	Endereço Serial (P0308)
0684	3.8.3.1.0	Palavra de Controle via CANopen/DeviceNet (P0684)
0686	3.8.4.1.0	Controle Anybus-CC (P0686)
3000	3.8.4.2.0	Endereço IP do Inversor (P3000)
0741	3.8.5.1.0	Perfil Dados Profibus (P0741)
0295	3.9.1.0.0	Corrente Nominal de ND/HD do Inversor (P0295)
3001	3.9.2.0.0	Endereço MAC do Inversor (P3001)
1001	3.10.1.0.0	Comando para SoftPLC (P1001)
0204	3.11.1.0.0	Carrega/Salva Parâmetros (P0204)
0317	4.1.0.0.0	Start-Up Orientado (P0317)

- Motor Data
- Drive Data
- MAX
- Menu

Já as propriedades dos parâmetros são diversas (aqui tratamos de parâmetros porque os objetos que se referem a menus não tem necessidade dessas propriedades). Essas propriedades se dividem em seis partes:

- uma estrutura constituída por um grupo de *flags* do parâmetro;
- a unidade do parâmetro (podendo ser Ohm, rpm, Hz, porcentagem, entre outros);
- o número de casas decimais que deve ser utilizado quando for mostrado o valor do parâmetro;
- o valor máximo que o parâmetro pode assumir;
- o valor mínimo que o parâmetro pode assumir;
- o valor padrão para esse parâmetro, que é constituído de dois valores: um para quando o inversor opera em frequência padrão 60Hz, e um para quando o inversor opera em frequência padrão 50Hz. A frequência padrão de operação do inversor é selecionada pelo usuário.

Os *flags* contidos nas propriedades dos parâmetros não precisam ser todos explicados aqui. Eles indicam por exemplo se o parâmetro só pode ser alterado quando o inversor não está acionando um motor, se o parâmetro é somente de leitura, se ele pode ser acessado via rede, entre outros. A figura 5.2 mostra a estrutura de propriedades de um parâmetro.

Definidos os parâmetros e menus a serem incluídos, a estrutura do objeto, e toda a estrutura de propriedades dos parâmetros, só resta fazer a declaração deles. A figura 5.3 mostra a construção de um objeto parâmetro, e a figura 5.4 mostra a construção de um objeto menu.

Figura 5.2: Estrutura de Propriedades de um Parâmetro

```
typedef struct _TObjectProperties TObjectProperties;
struct _TObjectProperties
{
    objflags    flags;
    char*       unit;
    char*       decimal;
    void*       LimitMax;
    void*       LimitMin;
    void*       StdValue[2];
};
```

Figura 5.3: Declaração de um objeto parâmetro

```
TObjectProperties ObjectMotorCurrentAmpProperties = {
                {0, 0, 0,
                   0, 1, 0,
                   0, 0, 0,
                   0, 0, 0,
                   0, 1, 0, 0},
                UNIT_A,
                1,
                45000,
                0,
                {0, 0}
};

const TObject ObjectMotorCurrentAmp = {
        3,
        { 1, 1, 1, 0, 0 },
        ObjectTypeUInt16,
        StandardObjectReader,
        StandardObjectWriter,
        StandardObjectPropertiesReader,
        StandardObjectPropertiesWriter,
        &ObjectMotorCurrentAmpProperties,
        &ObjectMotorCurrentAmpData };
```

Figura 5.4: Declaração de um objeto menu

```
const TObject ObjectMeasurementsMenu = {
        2006,
        { 1, 1, 0, 0, 0 },
        ObjectTypeMenu,
        StandardObjectReader,
        StandardObjectWriter,
        StandardObjectPropertiesReader,
        StandardObjectPropertiesWriter,
        NULL,
        &ObjectMeasurementsMenuData };
```

5.2: Inclusão na Árvore

Declarados os objetos, precisamos agora fazer a inserção de cada um deles na árvore AVL. Como definido na subseção 4.2.2, os objetos devem ser inseridos em ordem de nível, referente à estrutura de menus. Significa dizer que o primeiro objeto a ser inserido na árvore deve ser o Menu Principal, seguido dos menus do nível 1 (Status, Diagnósticos, Configurações, Startup Orientado e Lista de Parâmetros), e assim em diante. Isso deve ser feito para que seja possível que cada objeto inserido na árvore possa saber quem é o menu que leva à ele, de modo a facilitar a navegação em menus.

É declarada então uma árvore (`SearchTree`) nula, e são feitas sucessivas chamadas da função de inserção de objetos, para cada um dos objetos declarados, em ordem de nível. Após todas essas inserções, é chamada sucessivamente a função `TInsertMenuParameters`, para cada um dos menus que deve receber objetos. A ordem de chamada dessa função já não importa, visto que ela procura para cada objeto menu todos os seus parâmetros ou submenus que já estão inseridos na árvore. E por fim é chamada uma única vez a função `TFillAllParametersMenu`, para preencher o menu Lista de Parâmetros.

Feito isso está criada a árvore AVL, e nos resta fazer testes para garantir que ela opera da maneira esperada.

5.3: Testes

5.3.1: Impressão da Árvore

O primeiro teste a ser feito é a impressão na tela da árvore criada, para certificar de que todos os objetos foram inseridos da maneira correta, e que a árvore encontra-se razoavelmente balanceada. São criadas portanto três funções: `void vPrintTree ( SearchTree T )`, `void vPrintTreeLvl ( SearchTree T, int level )` e `void vPrintPadding ( char ch, int n )`. A primeira é a função principal, que deve ser chamada pelo desenvolvedor quando deseja imprimir a árvore, e recebe apenas ela como argumento. A segunda é uma função interna, recursiva, que inicialmente recebe a árvore a ser impressa como argumento, e um valor 0. Este valor inteiro é utilizado pela função para que ela saiba em que nível da árvore ela se encontra no momento. Já a terceira função é uma função auxiliar, que usa do valor do nível atual em que a árvore se en-

Figura 5.5: Árvore de objetos - topo

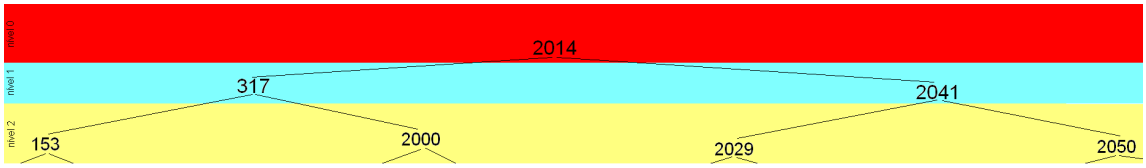
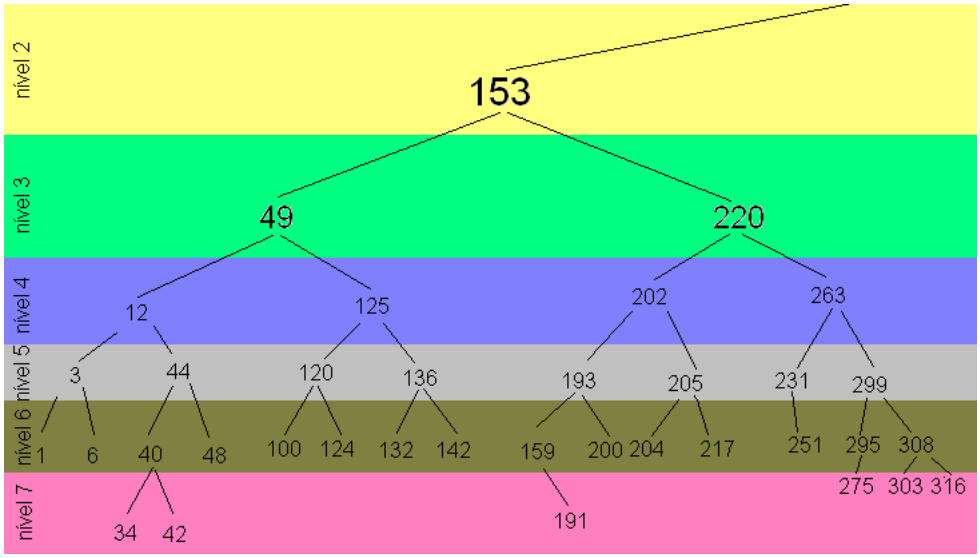


Figura 5.6: Árvore de objetos - ramo da extrema esquerda



contra para imprimir um caractere de “preenchimento”, para facilitar a visualização da árvore impressa.

A árvore é impressa com cada objeto mostrando primeiro o seu ID estruturado, e depois seu ID de rede. A função percorre a árvore inteira primeiro indo para a direita, e depois indo para a esquerda. E enfim a árvore é impressa na tela do console do ambiente de programação. Como a árvore criada é muito grande, fica inviável anexar a este documento uma imagem mostrando sua impressão no console do Eclipse. As figuras 5.5, 5.6, 5.7, 5.8 e 5.9 são melhores para visualizar a árvore final.

Como podemos ver, a árvore alcançada não se encontra perfeitamente balanceada segundo a definição citada por Loudon [5], mesmo que nenhum dos nós possua fator AVL fora da faixa designada. Existem nós sem filhos nos níveis 5, 6, e 7, o que fere a definição que diz que esses nós devem estar todos nos dois últimos níveis. Porém podemos considerar o balanceamento da árvore como razoavelmente aceitável, visto que não existem grandes discrepâncias na árvore a ponto de o balan-

Figura 5.7: Árvore de objetos - ramo da esquerda-centro

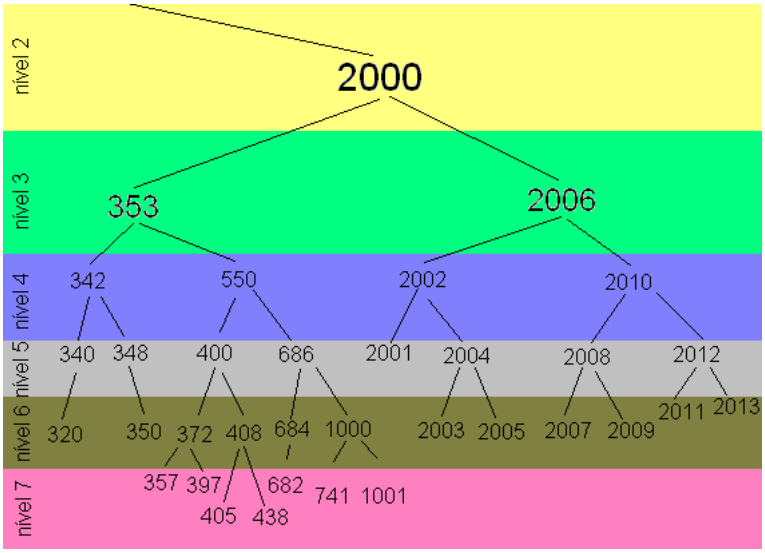


Figura 5.8: Árvore de objetos - ramo da direita-centro

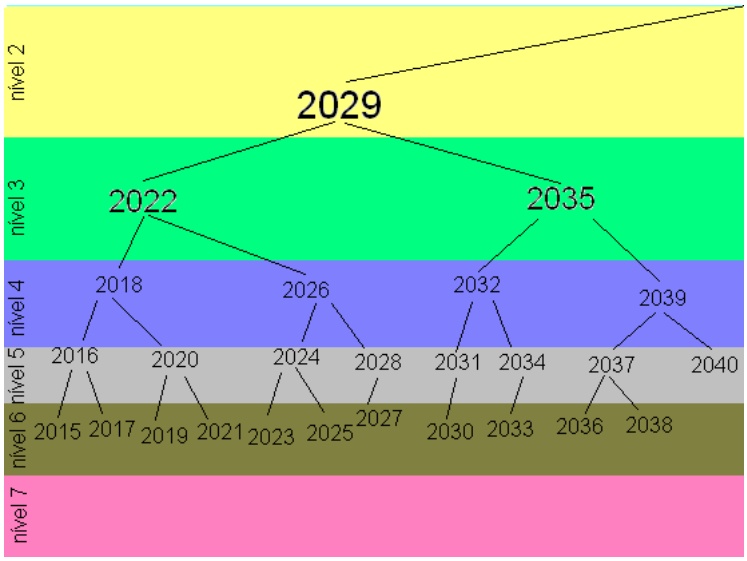
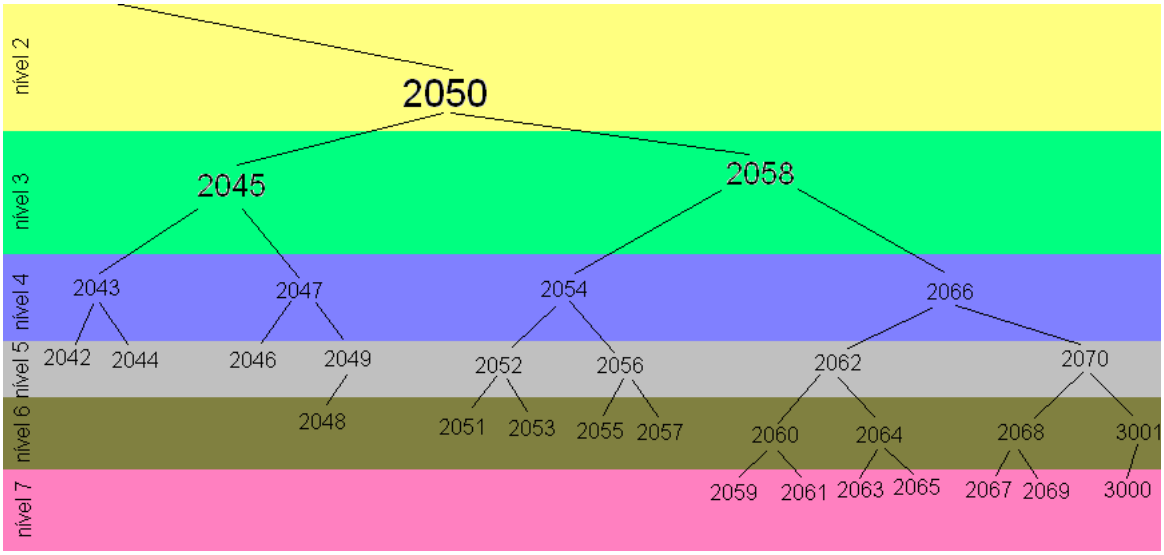


Figura 5.9: Árvore de objetos - ramo da extrema direita



ceamento atual causar significante queda de desempenho. Apesar de não garantir perfeito balanceamento, a árvore AVL se mostrou adequada, e podemos acreditar que ela se manterá adequada mesmo quando forem inseridos todos os parâmetros do inversor, e não só a amostra selecionada.

5.3.2: Navegação nos Menus

Apenas para motivo de teste da estrutura de dados criada, é desenvolvida uma função de simulação de navegação entre os menus que utiliza os objetos criados. A navegação na HMI, ou por algum outro tipo de interface, será desenvolvida em um momento posterior do projeto da nova linha de inversores, mas é necessário que se simule essa navegação no escopo desse projeto para nos certificarmos de que a estrutura criada atende os requisitos de navegação especificados anteriormente nesse documento.

A função de simulação de navegação possui um ponteiro (`TObject * TSelectedObject`) que referencia o objeto atualmente selecionado. Esse ponteiro inicialmente referencia o menu raiz da estrutura (Menu Principal). Criado esse ponteiro, a função passa a funcionar dentro de um *loop*, que inicialmente recebe informação de que tecla do teclado foi pressionada, e de acordo com essa informação é selecionado um objeto diferente ou não.

Com base em um protótipo de HMI da nova linha de inversores, sabemos que

para navegar no menu o usuário terá seis botões à sua disposição:

- Uma seta para cima
- Uma seta para baixo
- Uma seta para a esquerda
- Uma seta para a direita
- Uma tecla “Enter”
- Uma tecla “ESC”

As setas para cima e para baixo devem servir para selecionar o objeto anterior e o próximo, respectivamente, dentro do menu onde o usuário já se encontra. Elas portanto não realizam troca de nível. Já as setas para a esquerda e para a direita devem servir para retornar ao nível anterior ou para avançar de nível (“entrar no menu”), respectivamente. A tecla “Enter” além ter a mesma função da seta da direita deve ser capaz de entrar em um modo de edição de valor, quando o objeto selecionado for um parâmetro. A tecla “ESC” deve possuir a mesma função da seta para a esquerda, e além disso deve retornar ao modo de monitoração da HMI quando o objeto selecionado for o Menu Principal.

As operações descritas acima são desenvolvidas com facilidade a partir das operações com ponteiros que C possibilita. Quando a navegação avança de nível, é consultado o vetor de dados do objeto em questão, que faz parte da estrutura do mesmo. Esse objeto necessariamente deve ser um menu, ou o programa simplesmente ignora a operação. O vetor de dados do objeto menu possui a lista dos objetos que tem esse menu como pai, organizados já na ordem em que devem ser exibidos. É necessário então apenas que o ponteiro de objeto selecionado passe a apontar para o primeiro objeto nesse vetor. A partir disso, sempre que o usuário navega para o objeto anterior ou posterior da lista (pelas setas para cima e para baixo), o ponteiro apenas é incrementado ou decrementado, quando existir objetos suficientes no vetor que permitam essas operações.

A operação de retorno de nível usa do campo na estrutura do objeto que indica quem é seu menu-pai. Sempre que é solicitado o retorno ao nível anterior, o ponteiro do objeto selecionado passa a apontar para o objeto que consta como menu-pai do objeto selecionado atualmente. Porém precisamos prever uma exceção para essa

regra. Por causa do menu “Lista de Parâmetros”, que contém todos os parâmetros do inversor, precisamos criar um *flag* que indique quando a navegação passa ao próximo nível a partir do objeto que representa esse menu. Desse modo, quando acontece o retorno de nível, consulta-se esse *flag*, e se ele indicar que o usuário havia entrado nesse menu, o novo objeto selecionado deve ser o próprio menu “Lista de Parâmetros”.

Como para a realização desses testes não é necessário a troca do modo de operação da HMI (para modo de edição de valor ou modo de monitoração), eles não são implementados nesse momento. A execução dessa função de simulação de navegação permite chegar a conclusão de que a estrutura definida comporta o sistema de navegação necessário para interação dos dados com a HMI.

5.4: Síntese

Neste capítulo tratamos de adicionar à árvore uma parte dos parâmetros atuais dos inversores e os menus desenvolvidos para a nova linha de inversores, para que fossem feitas análises sobre a árvore. Foi feita então a impressão da árvore obtida, de modo a poder visualizar a correta inserção dos dados e o balanceamento dela. Depois disso foi feita uma simulação da navegação nos menus da HMI, usando a árvore construída.

Essa etapa constitui a última do desenvolvimento de uma estrutura de dados para a nova linha de inversores. Depois de realizado um estudo aprofundado sobre estruturas de dados, escolhida uma estrutura, e construídas as rotinas necessárias para a criação dessa estrutura, restava apenas a realização de testes e posteriores análises para finalizar esse desenvolvimento. As rotinas criadas apresentaram os resultados esperados em todas as situações, e a realização de uma navegação simulada foi suficiente para concluir que a estrutura construída é bastante eficaz para a aplicação desejada.

Capítulo 6: Estudo e Análise da Linguagem de Script Lua

Como já levantado, a implementação e teste do interpretador Lua embarcado em um microcontrolador e o desenvolvimento de uma estrutura de dados para o inversor precisavam caminhar paralelamente, visto que o projeto visa a interação destes em um momento futuro. A partir desse momento será descrito o trabalho feito com a linguagem Lua, e sempre que possível estará traçada neste documento a relação entre este trabalho e o que foi descrito antes desse capítulo.

Lua é uma linguagem de script cujo emprego em inúmeros tipos de aplicação não para de crescer, desde grandes aplicações para *desktops* até softwares embarcados. Atualmente Lua é a linguagem mais utilizada para *scripting* em jogos, é encontrada no Sistema Brasileiro de Televisão Digital, e faz parte inclusive de sistemas de segurança, como a ferramenta Wireshark.

De acordo com [2] e [6], estão entre suas características:

- Robustez: Lua foi lançada em 1993, possui um extenso manual de referência, e é utilizada em diversas aplicações industriais e de larga escala, que confiam na sua robustez e estabilidade;
- Velocidade: Lua tem uma merecida reputação de apresentar bom desempenho. Lua é marca de referência como a linguagem mais rápida entre as linguagens de script interpretadas;
- Portabilidade: Lua é distribuída em um pacote que pode ser compilado e executado em qualquer sistema que tenha um compilador C padrão. Ela pode ser utilizada em Unix, Windows, dispositivos móveis (rodando sistemas Android, iOS, BREW, Symbian, Windows Phone), microprocessadores embarcados, mainframes IBM, etc.;

- *Embarcabilidade*: Lua possui uma API simples e muito bem documentada, que pode ser incluída facilmente em programas escritos nas mais diversas linguagens;
- Tamanho Reduzido: Adicionar Lua à sua aplicação não faz com que ela fique imensa. Seu código-fonte contém em torno de 20.000 linhas de C;
- *Open-Source*: Lua é um software livre distribuído sob a licença do MIT [1], e sob seus termos pode ser usada para qualquer propósito, seja ele comercial ou não, sem custo algum.

Mas acima de tudo precisamos reforçar que Lua é uma linguagem dinâmica, e como tal possui:

- Interpretação dinâmica: linguagens dinâmicas são capazes de executar trechos de código criados em tempo de execução;
- Tipagem dinâmica forte: Tipagem dinâmica significa que as verificações de tipo ocorrem em tempo de execução. Isso implica que não existem declarações de tipo dentro do código. Tipagem forte significa que nenhuma operação é aplicada a um tipo incorreto;
- Gerência automática de memória dinâmica: significa que a linguagem gerencia automaticamente a memória que está em uso pelo programa, garantindo que não hajam desperdícios de memória ou descontinuidades, sem a necessidade de interferência do desenvolvedor;

Essas características fazem com que Lua seja particularmente atrativo para estabelecer a relação entre a HMI do inversor de frequência e seus dados, com os quais trabalhamos para criar uma estrutura nos capítulos anteriores. Isso é evidenciado pelas vantagens que Lua oferece para ser embarcada em microprocessadores, dispositivos que costumam trabalhar com recursos reduzidos de memória e por grandes períodos de tempo (inversores de frequência podem ficar em operação por períodos de anos).

6.1: Interpretador Lua

Em um primeiro momento, foi necessário um estudo para familiarização com a linguagem Lua. Afinal é necessário adquirir conhecimento sobre uma linguagem antes

de embarcá-la em outra. Nesse estágio, foram utilizados como recursos o livro “Programming in Lua”[2], o manual de referência de Lua [3], e o texto “Uma Introdução à Programação em Lua” [7]. Para o estudo feito foi utilizado o interpretador Lua fornecido pelo projeto *luaforwindows*.

6.2: Lua embarcado

Depois de estudada a linguagem Lua o suficiente para o desenvolvimento de alguns programas simples, a próxima etapa era embarcá-lo em outros programas simples em C para microcontrolador. Nessa etapa já se torna importante utilizar o microcontrolador, ou a família de microcontroladores que se estuda colocar na nova linha de inversores. Foi utilizado, então, um microcontrolador da família RX600 do fabricante Renesas. Os microcontroladores da família RX600 tem processadores de 32 bits funcionando a 100MHz, possuem até 2MB de memória flash, e tem a capacidade de estabelecer diversos tipos de comunicação (Ethernet, USB, CAN, I2C, entre outros). Esses microcontroladores devem utilizar um sistema operacional em tempo real (RTOS), que será utilizado na nova linha de inversores.. Para a programação do microcontrolador foi utilizado o *debugger J-Link GDB Server* da SEGGER.

Para utilização da bilioteca Lua embarcada é necessário apenas obter suas bibliotecas e incluí-las no projeto em desenvolvimento. A partir disso o desenvolvedor deve criar um novo estado Lua, se necessário abrir as bibliotecas auxiliares que Lua fornece, e começar a chamar funções da API. A API Lua gira em torno de duas estruturas: o estado Lua, e a pilha. O estado Lua é uma estrutura que armazena todas as informações necessárias para o interpretador Lua, visto que o código C de Lua não possui nenhuma variável global. A pilha é a maneira como a aplicação C e a API Lua conversam. Todos os valores que um deseja passar ao outro deve ser feito por meio da pilha.

Após mais uma etapa de estudo, dessa vez sobre a API Lua, podemos partir para implementação do programa no microcontrolador. Levando em consideração que, apesar de pequena, a API precisa de um considerável espaço na pilha do microcontrolador, sobre o qual está rodando também um RTOS, foram necessários consideráveis ajustes nas configurações de pilha e memória tanto do microcontrolador quanto do RTOS para conseguir ter total funcionamento da API embarcada.

6.2.1: Testes de desempenho

Começando com rotinas simples, foi feito primeiramente um teste para piscar leds utilizando C puro, e utilizando C com o auxílio do Lua. Esse teste mostrou que a função utilizando a API Lua precisou de pelo menos 180 vezes mais espaço na pilha que a função escrita em C. Apesar disso a API se mostra, até agora, viável.

É necessário então fazer mais testes de desempenho do Lua embarcado, para garantir seu funcionamento em operações mais complexas. Para testar o tempo de processamento do Lua, implementaremos o algoritmo da sequência de Fibonacci, utilizando três diferentes métodos:

1. função com implementação em código Lua, chamada pela API Lua C;
2. função com implementação em código C;
3. função com implementação em código C, chamado via código Lua a ser chamado pela API Lua C;

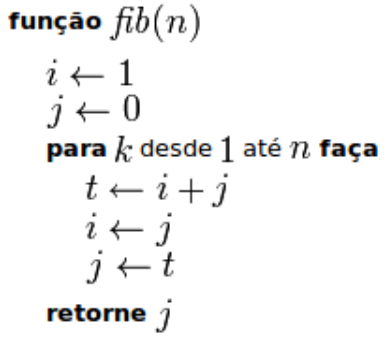
Para isso são utilizadas as seguintes funções da API Lua C:

- *luaL_dostring*: função que interpreta e roda um string de código Lua;
- *lua_register*: seta uma função escrita em C como o valor de uma variável global.

Os códigos para o cálculo da sequência de Fibonacci seguem o algoritmo da figura 6.1. O teste é feito calculando 100 vezes para cada método os primeiros 5000 números da sequência. Na implementação em Lua o string que contém o trecho de código é simplesmente chamado pela função *luaL_dostring* a cada iteração. Na implementação em C chamada pelo código Lua, a função em C é registrada como valor da variável ”f_fibonacci()”, e a cada iteração é utilizada a função *luaL_dostring* para chamar essa função. É utilizado então um dos timers do microcontrolador para registrar quantos pulsos de clock são necessários para completar as 100 iterações. Os resultados encontram-se na figura 6.2, e representam o valor do contador do timer para cada operação.

Espera-se que o segundo método seja o mais rápido deles, pois a função puramente em C é traduzida com maior simplicidade para as instruções do microcontrolador. O terceiro método espera-se que seja ligeiramente mais lento, visto que é

Figura 6.1: Algoritmo usado para cálculo da sequência de Fibonacci



necessário fazer a chamada via Lua antes de rodar as instruções do C. Já o primeiro método deve requerer um tempo muito maior de processamento, de valor pelo menos duas ordens maior do que o segundo método, visto que são necessárias diversas chamadas da API Lua para concluir a operação.

É necessário ressaltar que as amostras 19 e 69 apresentaram comportamento anormal, como se pode ver na figura 6.2. Essas anormalidades se apresentaram em todas as repetições feitas dos testes, e aparentemente tudo ocorreu normalmente durante essas iterações. Não foi encontrado nenhum erro ou comportamento fora do esperado durante elas, e no entanto não se sabe explicar porque a contagem do timer do microcontrolador não ficou próxima das outras amostras.

Além da sequência de Fibonacci foi feita uma tentativa de realizar uma multiplicação de matrizes 10x10 utilizando os mesmos três métodos, porém essa tentativa não obteve sucesso. Enquanto que nos métodos 2 e 3 não houve problemas na implementação e execução, o método 1 não pode ser realizado. Apesar de diversas tentativas e diferentes implementações utilizadas o teste se provou inviável ainda no ponto da alocação de memória para as matrizes a serem multiplicadas pelo Lua. Apesar de não haver problema na alocação de memória para as matrizes em C, no ambiente Lua não foi possível fazê-lo, com os recursos disponíveis.

6.3: Análise dos resultados dos testes

A partir dos resultados da figura 6.2 podemos ver que, conforme era esperado, para aplicações CPU-bound o código com Lua embarcado exige do sistema uma capacidade de processamento muito maior que os códigos puramente em C, e em

Figura 6.2: Tabela de resultado dos testes realizados com a sequência de Fibonacci

Iteração	Lua	C	Lua + C
1	3866325	45250	49344
2	3864230	45249	48905
3	3864455	45250	48954
4	3864225	45250	48904
5	3864458	45250	48904
6	3864230	45250	48903
7	3864455	45250	48904
8	3864236	45250	48904
9	3864466	45020	48905
10	3864228	45250	48904
11	3864459	45250	48904
12	3864468	45250	48904
13	3864214	45250	48904
14	3864461	45249	51141
15	3864226	45250	48898
16	3864463	45250	48898
17	3864236	45250	48898
18	3864461	45250	48946
19	3864235	45250	14538
20	3864458	45250	48890
21	3864231	45249	49512
22	3864464	45250	48797
23	3864230	45250	48806
24	3864462	45250	48806
25	3864229	45250	48805
26	3864459	45250	48805
27	3864224	45020	48805
28	3864461	45250	48805
29	3864229	45250	48805
30	3864460	45250	48805
31	3864231	45250	48805
32	3864455	45250	48805
33	3864231	45249	48805
34	3864451	45250	48805
35	3864224	45250	49011
36	3864466	45250	48897
37	3864231	45250	48898
38	3864466	45250	48898
39	3864232	45250	48898
40	3864466	45249	48898
41	3864228	45250	48898
42	3864460	45250	48898
43	3864236	45250	48898
44	3864461	45250	48898
45	3864230	45020	48898
46	3864455	45250	48898
47	3864224	45250	48948
48	3864459	45250	48898
49	3864229	45250	48897
50	3864463	45250	48898
51	3864237	45250	48898

Iteração	Lua	C	Lua + C
52	3864466	45249	48898
53	3864231	45250	48898
54	3864458	45250	48898
55	3864233	45250	48898
56	3864458	45250	48898
57	3864235	45250	48898
58	3864462	45250	48898
59	3864228	45249	48898
60	3864459	45250	51212
61	3864230	45250	48969
62	3864460	45020	48939
63	3864231	45250	48891
64	3864467	45250	51274
65	3864451	45250	49420
66	3864232	45250	48805
67	3864461	45250	48805
68	3864234	45250	48805
69	3864466	45250	3459
70	3864236	45250	48787
71	3864457	45249	48807
72	3864236	45250	48883
73	3864457	45250	48806
74	3864236	45250	48806
75	3864470	45250	48806
76	3864221	45250	48806
77	3864467	45250	48806
78	3864233	45249	48807
79	3864467	45250	48806
80	3864241	45020	48806
81	3864462	45249	48806
82	3864236	45250	48807
83	3864457	45250	48806
84	3864230	45250	48806
85	3864465	45250	48806
86	3864231	45250	48805
87	3864469	45250	48805
88	3864226	45249	48805
89	3864458	45250	48805
90	3864231	45250	48805
91	3864451	45250	48805
92	3864224	45250	48805
93	3864466	45250	48805
94	3864234	45250	48805
95	3864461	45249	48805
96	3864230	45250	48805
97	3864469	45250	48805
98	3864229	45020	48805
99	3864465	45250	48805
100	3864231	45250	48805
Média	**3864367**	**45236**	**48146**

aplicações com processamento limitado a velocidade da operação será prejudicada. O método que utiliza o Lua para chamar uma função em C mostra pouca diferença de tempo de processamento frente ao tempo de processamento do método que utiliza C puro, o que é esperado, visto que, passado o *overhead* da chamada do Lua, a função executada é a mesma. Apesar disso, o funcionamento da API Lua até então não apresentou erros.

Já no teste da sequência Fibonacci, a grande necessidade de recursos impossibilitou o uso do Lua embarcado no microcontrolador utilizado. Será necessário, portanto, um estudo mais detalhado da linguagem e dos recursos disponíveis para validar sua viabilidade de uso embarcada.

Apesar da grande limitação encontrada para aplicações CPU-bound, seria necessário fazer também testes com aplicações I/O-bound, visto que esse tipo de operação é o que mais seria utilizado na interação da HMI com o inversor, que afinal é onde queremos aplicar a linguagem Lua. Porém não houve tempo hábil durante esse projeto para o desenvolvimento de uma aplicação I/O-bound para realização de testes.

Capítulo 7: Resultados Obtidos

Este documento apresentou a elaboração de uma proposta de solução para uma estrutura de dados a ser utilizada em nova linha de inversores de frequência da WEG e sua implementação, e a análise da utilização de um interpretador Lua embarcado para aplicação nesta linha. Esses dois temas do trabalho caminham paralelamente em direção ao mesmo objetivo, que é a criação de um novo firmware a ser incorporado à nova linha de inversores de frequência da WEG.

A partir das especificações traçadas para a estrutura de dados desenvolvida, foi feito uma minuciosa pesquisa sobre as estruturas com as quais podia ser implementada a estrutura desejada. De todas as estruturas estudadas, a que conseguia atingir as especificações com maior simplicidade era a estrutura em árvore binária. Esse tipo de estrutura, porém, só seria capaz de cumprir com as especificações implementando a simples modificação que a transforma em árvore AVL. Apesar de as árvores AVL não garantirem que a árvore binária estará perfeitamente balanceada, elas garantem um nível de balanceamento bom o suficiente para cumprir as especificações designadas para esse projeto.

Foi feito então um estudo sobre os parâmetros já existentes nos inversores de frequência, para possibilitar que fosse proposta uma estrutura de objeto condizente com a necessidade para cada um dos parâmetros do inversor. A estrutura de parâmetro proposta ao fim do estudo consegue contemplar todos os objetos que o inversor precisa conhecer. Conhecida então a estrutura do objeto, foi dado início o trabalho de criação da estrutura da árvore.

Como principal rotina de construção da árvore binária, a função de inserção conta com diversas funcionalidades. Ela não faz apenas alocar espaço na árvore para aquele elemento. Ela também determina que objeto corresponde ao menu no qual é encontrado o objeto sendo inserido; ela atribui ao objeto seu valor-padrão quando o objeto é um parâmetro; ela cuida da alocação de memória dos objetos; e ela cuida do

balanceamento da árvore, por meio de chamadas às funções de rotação sempre que necessário.

As rotinas de busca sobre a árvore binária compreendem, além da busca pelo ID do objeto (no caso seu endereço de rede), rotinas de busca de objetos por outros elementos, como o endereço estruturado e o próprio objeto. Essas buscas alternativas são úteis para facilitar a navegação pelos menus do inversor. As funções de retirada de objetos da árvore, apesar de terem sido criadas, a princípio não serão utilizadas.

Para melhor testar a estrutura criada, foi criada uma rotina de simulação de navegação nos menus do inversor. Essa rotina consegue com facilidade percorrer os menus utilizando a estrutura proposta, o que mostra que o trabalho desenvolvido terá sua aplicação garantida no firmware da nova linha de inversores.

Ao mesmo tempo foi sendo estudada uma forma de embarcar um interpretador Lua em um microcontrolador, para avaliar sua possibilidade de aplicação para a interação entre inversor e HMI. A princípio os testes foram fornecendo resultados positivos, mas não puderam ser todos concluídos. As rotinas que utilizavam da API Lua requeriam mais memória e maior capacidade de processamento que as rotinas escritas em C, e por fim a memória disponível no microcontrolador utilizado acabou não sendo suficiente para um dos testes.

A API Lua é poderosa, e quando completamente funcional é um recurso valioso para diversos tipos de aplicação. Porém, com os recursos disponíveis, ela não se mostrou viável a esse projeto. É necessário um estudo mais aprofundado da linguagem para garantir que se possa utilizá-la na nova linha de inversores da WEG. Devem ser feitos, no futuro, testes de aplicações I/O-bound com Lua, para uma análise mais completa da linguagem. Acompanhada desse estudo deve ser feita uma análise sobre a possibilidade de ampliação dos recursos de memória do equipamento. Ou então, fica como sugestão que quando as exigências de memória e processamento de uma certa rotina forem muito grandes para serem tratadas com Lua, que seja feito com que essa rotina seja realizada com C.

Capítulo 8: Conclusões

Este relatório é fruto do trabalho realizado pela autora durante seu estágio realizado na WEG Equipamentos Elétricos, unidade de Jaraguá do Sul. Este trabalho foi capaz de mostrar a aplicação dos conhecimentos adquiridos em diversas das disciplinas ministradas no curso de graduação de Eng. de Controle e Automação.

Uma das mais significativas das disciplinas que se relacionam com esse trabalho é a disciplina Fundamentos da Estrutura da Informação. No escopo dessa disciplina trabalhamos com tipos abstratos de dados, e são abordadas diversas estruturas de dados, como as estudadas nesse trabalho. Essa disciplina proporciona ao graduando uma base teórica, e alguns exemplos de aplicação, das estruturas de dados abordadas. Essa base foi essencial para o desenvolvimento do estudo em questão, que permitiu à autora que fossem aprofundados seus conhecimentos, e seu entendimento das estruturas de dados trabalhadas.

Além de Fundamentos da Estrutura da Informação, outra disciplina fortemente relacionada ao trabalho desenvolvido foi Microprocessadores (hoje substituída pela disciplina Arquitetura e Programação de Sistemas Microcontrolados). Nesse curso trabalhamos primeiramente com microprocessadores, e em seguida com microcontroladores. A disciplina requeria a realização de um trabalho com um microcontrolador PIC, que proporcionou uma experiência essencial para o trabalho desenvolvido nesse estágio. Essa experiência pode ser ainda mais aprofundada na disciplina Informática Industrial I, que, buscando demonstrar alguns dos equipamentos utilizados em sistemas de produção automatizados, também solicitava o desenvolvimento de um sistema utilizando um microcontrolador PIC.

Os conhecimentos que Microprocessadores e Informática Industrial I forneceram sobre microcontroladores foram absolutamente necessários na realização do estudo sobre a linguagem de script Lua. Como já foi dito foram necessários diversos ajustes nas configurações de pilha e memória do microcontrolador, que não poderiam

ter sido feitos sem o conhecimento prático do uso de microcontroladores. Além disso a utilização dos recursos do microcontrolador disponível (como interrupções, timer, entre outros) teria sido um trabalho de grande dificuldade, não fossem esses conhecimentos prévios.

Apesar de este trabalho não se aprofundar neste quesito, é importante lembrar do emprego de um sistema operacional de tempo real com o microcontrolador utilizado. Como não era essencial que fosse coberto esse ponto no presente relatório, o RTOS utilizado foi apenas mencionado, mas efetivamente precisou ser feito longo estudo sobre o mesmo durante o estágio. Esse tipo de ferramenta foi trazido na disciplina de Informática Industrial II, apesar de não ter sido realizado nenhum trabalho prático com ela durante as aulas. Fica aqui a sugestão da autora para que se inclua essa ferramenta em projetos durante a graduação, como por exemplo o projeto relacionado a microcontroladores da disciplina Informática Industrial I, visto que os RTOS são largamente utilizados em aplicações industriais e comerciais e o conhecimento prático sobre ele pode ser muito favorável.

É necessário ressaltar também a importância da disciplina Estágio em Controle e Automação Industrial. Nessa disciplina também precisamos desenvolver um estágio, e ao final dele apresentar um relatório similar a este. Essa disciplina oferece a preparação necessária para realização do Projeto de Fim de Curso, de modo que o estudante cursando o último possui uma melhor idéia dos métodos que deve adotar durante a realização do seu estágio.

Do ponto de vista pessoal da autora desse trabalho, o estágio realizado foi capaz de proporcionar uma experiência em engenharia que o ambiente da universidade jamais poderia. A convivência dentro do ambiente de uma empresa mostrou como funcionam as coisas dentro dela, como as etapas de desenvolvimento de um produto, a hierarquia dentro da empresa, e os métodos utilizados para alcançar os objetivos. Esse tipo de experiência os professores de alguma determinada disciplina podem tentar expressar aos alunos, porém a compreensão jamais poderia ser completa sem realmente vivê-la. Tendo passado por diversas atividades de diferentes naturezas durante a graduação, claramente o projeto de fim de curso contribuiu fortemente para o futuro da autora como engenheira.

A WEG teve um impacto muito positivo, pois se mostrou sempre como uma empresa que prioriza os seus colaboradores, sempre respeitando eles, e garantindo um ambiente de trabalho amigável e confortável. Através da interação com os colegas de

trabalho, do convívio em um ambiente de desenvolvimento, e da mudança de cenário da universidade para a indústria, foi possível moldar a visão final da engenharia nos olhos da autora.

Referências

[1] The MIT License, http://opensource.org/licenses/mit-license.html. Acessado em Janeiro/2013.

[2] IERUSALIMSCHY, R. Programming in Lua. Ierusalimschy, Roberto, 2003.

[3] IERUSALIMSCHY, R.; FIGUEIREDO, L. H.; CELES, W. Lua 5.2 Reference Manual. Lua.org, PUC-Rio, 2011.

[4] WEG Equipamentos Elétricos S. A. Inversor de Frequência CFW-11 V3.1X Manual de Programação, 12/2011.

[5] LOUDON, K. Mastering Algorithms with C. O'Reilly, 1999.

[6] Lua: About, http://www.lua.org/about.html. Acessado em Janeiro/2013.

[7] IERUSALIMSCHY, R. Uma Introdução à Programação em Lua. XXVIII Jornadas de Atualização em Informática, 2009.