

**UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO**

**NORMA IEC 61131-3 PARA PROGRAMAÇÃO DE
CONTROLADORES PROGRAMÁVEIS: ESTUDO E
APLICAÇÃO**

HUGO CASATI FERREIRA GUIMARÃES

**VITÓRIA – ES
SETEMBRO/2005**

HUGO CASATI FERREIRA GUIMARÃES

**NORMA IEC 61131-3 PARA PROGRAMAÇÃO DE
CONTROLADORES PROGRAMÁVEIS: ESTUDO E
APLICAÇÃO**

Parte manuscrita do Projeto de Graduação do aluno Hugo Casati Ferreira Guimarães, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, para obtenção do grau de Engenheiro Eletricista.

**VITÓRIA – ES
SETEMBRO/2005**

HUGO CASATI FERREIRA GUIMARÃES

**NORMA IEC 61131-3 PARA PROGRAMAÇÃO DE
CONTROLADORES PROGRAMÁVEIS: ESTUDO E
APLICAÇÃO**

COMISSÃO EXAMINADORA:

Prof. Dr. Celso José Munaro
Orientador

Prof. Dr. Alessandro Mattedi
Examinador

Prof. Heliomar Guimarães Guzzo
Examinador

Vitória - ES, 16 de Setembro de 2005

DEDICATÓRIA

Aos meus pais e professores que sempre almejaram o melhor estudo para seus filhos.

AGRADECIMENTOS

Agradeço aos colegas, professores, amigos e parentes pelo apoio e incentivo durante todo o curso. Às secretárias Sonia Roubach e Inês Pavan pela paciência e ajuda quando necessário durante todo o curso.

Agradeço ainda ao meu orientador Celso J. Munaro pelas orientações e amizade durante as reuniões durante todo o período do projeto de graduação.

E a minha namorada Fonoaudióloga Fernanda pela compreensão e paciência durante as noites que não pude vela.

LISTA DE FIGURAS

Figura 1 – Componentes de um sistema de controle industrial	13
Figura 2 – Evolução dos sistemas de controle desde o final do século 19	15
Figura 3 – Componentes de um controlador programável.....	16
Figura 4 – Na programação de baixo nível, são usados os programas para edição, montagem e depuração no processo de geração do código de máquina.	17
Figura 5 – Programas escritos em linguagem de alto nível são independentes do hardware e são traduzidos para o código de máquina pelo compilador.	19
Figura 6 – Ciclo de varredura de um Controlador Programável.....	20
Figura 7 – Custo do <i>Software</i> versus Hardware.....	23
Figura 8 – Evolução da norma.	26
Figura 9 – Modelo de <i>Software</i> da norma IEC 61131-3	29
Figura 10 – Comunicação externa (remota) usando caminho de acesso	34
Figura 11 – Partida/parada do fluxo de controle de um <i>Software</i> IEC.	35
Figura 12 – Decomposição hierárquica do Bloco Funcional PID_Fuzzy.	37
Figura 13 – Representação do encapsulamento PID > PID_Auto_Tuning > PID_Fuzzy.....	38
Figura 14 – Exemplo de processamento não-determinístico.	39
Figura 15 – Esquema da codificação de prioridades das tarefas e distribuição dos programas, bem como sua execução.....	39
Figura 16 – Mesmo exemplo no método determinístico.	40
Figura 17 – Operadores padrões do <i>ST</i> no <i>CBF</i>	45
Figura 18 - Biestável SR (algoritmo interno em <i>FBD</i>).....	46
Figura 19 - Detector de Borda de Subida (algoritmo interno em <i>ST</i>).....	47
Figura 20 – Malha de Simulação de Controle de Nível de um Tanque.....	48
Figura 21 – Esquema de uma partida direta de motor no <i>Ladder</i>	49
Figura 22 – Exemplo de programa em <i>FBD</i>	54
Figura 23 – Principais elementos SFC.....	58
Figura 24 – Sistema construído.....	59
Figura 25 – Malha de controle de nível implementada.....	59

Figura 26 – Malha de controle da temperatura do fluido que entra no reservatório principal.....	60
Figura 27 – Malha de controle da temperatura do fluido no reservatório principal. ...	60
Figura 28 – Ambiente <i>Control Builder F</i> com representação hierárquica dos elementos de <i>software</i> IEC.	61
Figura 29 – Bloco Funcional de Controle da Temperatura Interna do Reservatório..	62
Figura 30 – Bloco Funcional de Controle da Temperatura do Fluido de Entrada do Reservatório.	63
Figura 31 – Bloco Funcional de Controle do Nível do Fluido do Reservatório.....	64
Figura 32 – Programa da Lógica de Controle em SFC.....	65
Figura 33 – Editando o programa da transição T1.....	66
Figura 34 e 35 – Editando a transição em Texto Estruturado.....	66
Figura 36 – Parâmetros do passo Tmp_Inte.	67
Figura 37 – Criação do programa em FBD para implementação da malha de controle de temperatura do reservatório.....	67
Figura 38 e 39 – Atribuição do Nome do Programa (Tmp_Inte) e edição do programa.	68
Figura 40 – Inserindo um Bloco Funcional.	68
Figura 41 – Escolha do BF.....	69
Figura 42 – Bloco Funcional de Controle da Temperatura Interna do Reservatório..	69
Figura 43 – Adentrando à lógica do BF.....	70
Figura 44 – Editando o programa em Texto Estruturado da condição de transição em T1.	71
Figura 45 – Programa em <i>ST</i> da condição de transição T4.....	71
Figura 46 – Chamada de Blocos Funcionais na Lista de Instruções.	72

LISTA DE TABELA

Tabela 1 – Primeira letra escrita para representar a variável de representação direta.	33
Tabela 2 – Segunda letra escrita para representar a variável de representação direta.	33
Tabela 3 – Tipos de Unidades de Organização de Programa.	36
Tabela 4 – Linguagens de Programação.	42
Tabela 5 – Representação hierárquica das linguagens.	42
Tabela 6 – Operadores padrões.	43
Tabela 7 – Velocidade 1 e 2 e pressão.	44
Tabela 8 – Operadores Principais	53
Tabela 9 – Operadores IL de comparação e controle de fluxo	54
Tabela 10 – Pilha de operação	54

SIMBOLOGIA

BF – Bloco Funcional

CBF – Control Builder F

CP – Controlador Programável

FÇ - Função

GLOSSÁRIO

CBF – O Control Builder F é o programa utilizado para representar os conceitos da norma IEC 61131-3.

CP – Entende-se por Controlador Programável qualquer equipamento de controle com capacidade de programação, tais como Estação de Engenharia/Operação (Computador de Processo), CLP's, SDCD's, Sistemas Híbridos, etc... Foi utilizado o termo Controlador Programável por desempenhar funções muito mais complexas que o processamento de lógicas discretas (I/O) do Controlador Lógico Programável (CLP) tais como o controle de malhas analógicas, capacidade aritmética, etc...

Depuradores ou *debuggers* – executam o programa passo-a-passo, de forma que possam ser simulados os dados do sistema real, sem a necessidade de conexão com a planta industrial para a depuração de erros.

Elementos de *Software* – a configuração define todos os elementos de *Software* que interagem entre si para desempenhar as funções de controle.

Estação de Engenharia – É o computador que engenheiro interage com o Sistema de Controle (CP + E/S + Transdutores + Atuadores) de Processo com um *Software* de programação e/ou um de supervisão.

PC – Computador Pessoal (*Personal Computer*).

Semântica – Transição sofrida no significado das palavras ou instruções.

Sintaxe – Disposição das palavras nas frases ou das instruções num programa.

SUMÁRIO

DEDICATÓRIA.....	I
AGRADECIMENTOS.....	II
LISTA DE FIGURAS	III
LISTA DE TABELA.....	V
LISTA DE TABELA.....	V
SIMBOLOGIA	VI
GLOSSÁRIO	VII
SUMÁRIO	VIII
RESUMO	XII
1 INTRODUÇÃO – EVOLUÇÃO DOS SISTEMAS DE CONTROLE	13
1.1 Sistemas de Controles	13
1.1.1 Evolução Histórica.....	14
1.1.2 Controladores Programáveis.....	15
1.1.2.1 Métodos de Programação	16
1.1.2.2 Código de Máquina e Assembler	16
1.1.2.3 Compilação e Interpretação	18
1.1.2.4 Execução Cíclica	19
1.1.3 <i>SoftPLC</i>	20
1.2 Motivação para Sistemas Abertos	21
1.2.1 Diferentes Dialeto de Programação	21
1.2.2 Qualidade de <i>Software</i>	21
1.2.3 Custo de <i>Software</i>	22
1.2.4 Portabilidade de Aplicações	23
2 A NORMA IEC 61131.....	25
2.1 Propósito da Norma IEC 61131-3.....	25
2.2 Certificação de Produtos	26
2.3 Introdução.....	28
2.4 Modelo de <i>Software</i> IEC	29
2.4.1 Configurações	30

2.4.2 Recursos.....	30
2.4.3 Programas	30
2.4.4 Tarefas (<i>Tasks</i>).....	30
2.4.5 Blocos Funcionais.....	31
2.4.6 Funções (<i>Functions</i>)	32
2.4.7 Variáveis de Escopo Local e Global.....	32
2.4.8 Variáveis de representação direta	32
2.5 Caminhos de Acesso (<i>Access Paths</i>).....	33
2.6 Fluxo de Controle.....	34
2.6.1 Partida	34
2.6.2 Parada	35
2.7 Unidade de Organização de Programa (POU)	36
2.8 Processamento Multitarefas	38
2.8.1 Escalonamento não-preemptivo	38
2.8.2 Escalonamento preemptivo.....	40
2.8.3 Otimização do uso da CPU do controlador;	41
3 LINGUAGENS DE PROGRAMAÇÃO IEC61131-3	42
3.1 Introdução.....	42
3.2 Texto Estruturado (<i>ST</i>)	43
3.2.1 Operadores	43
3.2.2 Exemplo 1	44
3.2.3 Exemplo 2	44
3.2.4 Comandos da Linguagem Texto Estruturado	44
➤ Cálculos Aritméticos	44
➤ Comando condicional IF THEN ELSE	44
➤ Comando condicional CASE.....	44
➤ Comando de repetição FOR ... DO.....	44
➤ Comando de repetição WHILE ... DO	44
➤ Comando de repetição REPEAT ... UNTIL	44
➤ EXIT	44

➤	RETURN	44
	3.2.5 Chamada de Blocos Funcionais.....	45
	3.2.6 Limites do Sistema	45
	3.2.6.1 Ocupação da Memória.....	45
	3.3 Diagrama de Blocos Funcionais (FDB)	46
	3.3.1 Blocos Funcionais.....	46
	3.3.1.1 Blocos Funcionais Padrões.....	46
	3.3.1.2 Blocos Funcionais Complexos (criados a partir dos padrões) ..	47
	3.3.2 Portabilidade entre <i>ST</i> e <i>FBD</i>	47
	3.3.3 Exemplo de tradução de uma malha de controle do <i>FBD</i> para o <i>ST</i>	48
	3.4 Diagrama Ladder (LD).....	49
	3.4.1 Exemplo de uma Partida Direta de um Motor com contato selo.....	49
	Tradução para o <i>ST</i> da figura 20:.....	49
	3.4.2 Portabilidade para o FBD:	50
	3.4.3 Aplicabilidade de Diagramas Ladder	51
	3.5 Lista de Instruções.....	53
	3.5.1 Semântica.....	53
	3.5.2 Operadores.....	53
	3.5.3 Portabilidade entre IL e outras linguagens IEC.....	54
	3.5.4 Análise de Desempenho;	55
	3.6 Sequenciamento Gráfico de Funções	56
	3.6.1 Estrutura da Rede (Chart)	56
	3.6.2 Análise de Desempenho	58
4	APLICAÇÃO DA NORMA IEC 61131-3 A UM ESTUDO DE CASO ...	59
	4.1 O Ambiente <i>Control Builder F</i>	60
	4.2 Blocos Funcionais	61
	4.2.1 Bloco Funcional MALHA_T_INTE	62
	4.2.2 Bloco Funcional MALHA_T_ENTR	62
	4.2.3 Bloco Funcional MALHA_NIVEL	63
	4.2.4 Programa principal em Sequenciamento Gráfico de Funções.....	64

4.3 Instanciação de Blocos Funcionais	71
4.4 Chamada dos Blocos Funcionais na Lista de Instruções	72
4.5 Portabilidade entre Recursos de programas;	72
5 CONCLUSÃO.....	73
APÊNDICE A.....	74
APÊNDICE B	77
REFERÊNCIAS BIBLIOGRÁFICAS.....	82

RESUMO

Este trabalho apresenta as vantagens práticas da adoção da Norma IEC 61131-3 que padroniza um modelo de *Software* para programação de Controladores Programáveis utilizando cinco linguagens de programação para desenvolvimento de programas para aplicações de controle industrial. Será feita uma introdução com a evolução dos Sistemas de Controle até os dias atuais bem como a motivação para sistemas abertos onde se deseja minimizar os dialetos de programação encontrados no mercado para as cinco linguagens de programação da norma e se portar aplicações existentes de uma plataforma de *hardware* antiga para novos sistemas.

No capítulo 2, tem-se uma introdução da norma com a trajetória da norma ao longo do tempo, as oito partes da Norma IEC 61131 (sendo que a terceira é a que se estuda), sua certificação de produtos (*softwares*) conforme critérios adotados. E a explanação sobre a Norma IEC 61131-3. Com o Modelo de *Software* IEC representando todos os níveis de configuração, as Unidades de Organização de Programas (*POU*) que replicam os programas, blocos funcionais e funções para reutiliza-los em uma nova programação e dois métodos de configuração do tempo de ciclo de varredura de uma tarefa que processa os programas.

No capítulo 3 serão abordadas as cinco linguagens de programação da norma destacando suas particularidades mais importantes respectivamente, como portabilidades entre linguagens e otimização do uso da CPU.

Para analisar os conceitos da Norma IEC 61131-3 foram programadas, no Ambiente *Control Builder F*, três malhas de controle em Diagrama de Blocos Funcionais, para depois inserí-las no Sequenciamento Gráfico de Funções.

1 INTRODUÇÃO – EVOLUÇÃO DOS SISTEMAS DE CONTROLE

1.1 Sistemas de Controles

Todo sistema de controle precisa de algum tipo de controlador para garantir uma operação segura e economicamente viável. No nível mais simples, uma planta pode consistir basicamente de um motor elétrico acionando um ventilador para controlar a temperatura de uma sala. No extremo oposto, uma planta pode ser um reator nuclear para produção de energia elétrica. Assim, todos os sistemas de controle são definidos em três partes: os transdutores, os controladores e os atuadores.

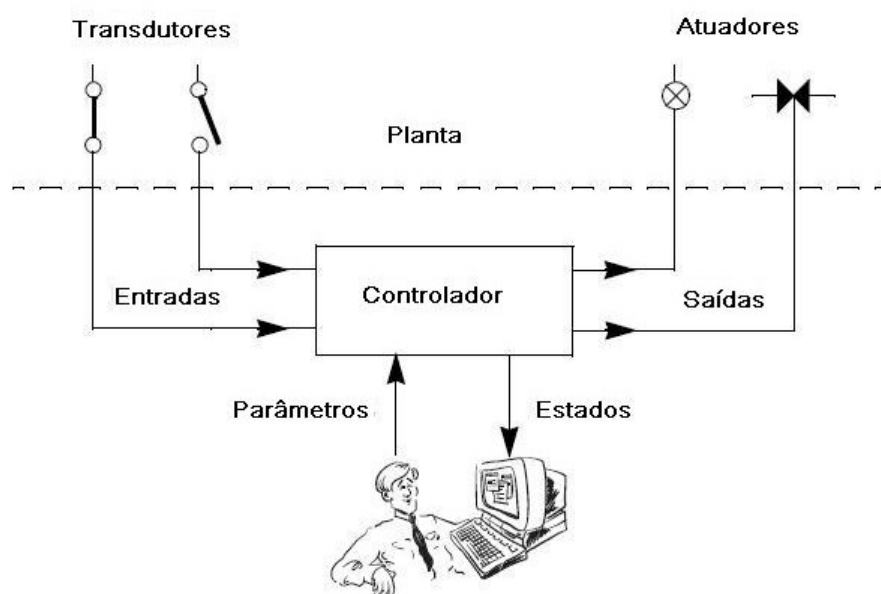


Figura 1 – Componentes de um sistema de controle industrial

O controlador monitora o estado real do processo de uma planta através de um número de transdutores, definido de acordo com a aplicação. Os transdutores convertem as grandezas físicas em sinais normalmente elétricos, os quais são conectados com as entradas dos controladores. Transdutores digitais (discretos) medem variáveis com estados distintos, tais como ligado/desligado ou alto/baixo, enquanto os transdutores analógicos medem variáveis com uma faixa contínua, tais como pressão, temperatura, vazão ou nível.

Com base nos estados das suas entradas (variáveis de processo - *PV*), o controlador utiliza um algoritmo de controle embutido para calcular os estados das suas saídas (variáveis manipuladas - *MV*). Os sinais elétricos das saídas são

convertidos para o processo através dos atuadores. Muitos atuadores geram movimentos como válvulas, motores, bombas e outros e utilizam a energia potencial pneumática para o acionamento.

O operador interage com o controlador através dos parâmetros de controle (ex: *Set Point*, K_p , K_i , K_d). Alguns controladores podem mostrar os estados do processo através de um display ou tela, que é chamado de IHM (Interface Homem Máquina).

1.1.1 Evolução Histórica

Os primeiros sistemas de controle foram desenvolvidos durante a revolução industrial, no final do século 19. As funções de controle eram implementadas através de engenhosos dispositivos mecânicos, os quais automatizam algumas tarefas críticas e repetitivas das linhas de montagem da época. Estes dispositivos tinham de ser desenvolvidos de acordo com a aplicação e devido à natureza mecânica tinham vida útil reduzida.

Nos anos 20, os dispositivos mecânicos foram substituídos pelos relés e contadores. A lógica a relés viabilizou o desenvolvimento de funções de controle digitais mais complexas e sofisticadas mostrando ser uma alternativa de custo viável para automações de pequenas máquinas com um número limitado de transdutores e atuadores.

Atualmente na indústria moderna não existe o uso da lógica a relés. Mesmo porque, o tempo que se perde para entender e alterar uma lógica sequencial com interligações elétricas torna o trabalho muito difícil e com alto custo de mão de obra.

O surgimento na década de 70 do circuito integrado (CI), baseado na tecnologia TTL ou CMOS, proporcionou uma melhora significativa no desempenho e vida útil dos sistemas de controle. Os primeiros computadores eram grandes, caros, difíceis de programar e nada robustos ao ambiente agressivo das indústrias.

Os primeiros CLP's (Controlador Lógico Programável) foram desenvolvidos para atender a demanda na indústria automobilística norte americana com um conjunto de instruções reduzido, processando somente condições lógicas, de controle digital (discreta).

Atualmente, os Controladores Programáveis aplicam-se tanto ao controle discreto (I/O) quanto para o controle de malhas analógicas. Diferentes computadores são conectados via rede local (LAN) a um computador supervisor central, o qual gerencia os alarmes, receitas e relatórios.

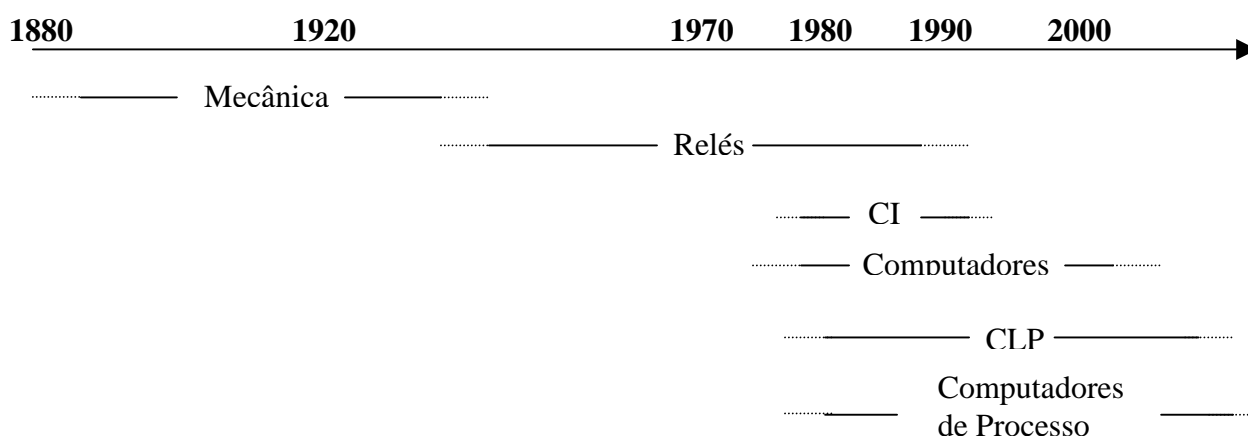


Figura 2 – Evolução dos sistemas de controle desde o final do século 19

1.1.2 Controladores Programáveis

Na década de 70 surgiu o primeiro Controlador Lógico Programável (CLP) com uma linguagem de programação simplificada e limitada para atender a demanda industrial em substituição aos sistemas de controle baseados em relés. Com o passar dos anos várias funções foram incorporadas ao controlador, tais como tratamento de variáveis analógicas e algoritmos aritméticos complexos, não se limitando somente a lógicas discretas (I/O). Assim uma definição mais atual, abrangente e já utilizada pelos fabricantes é a de Controlador Programável.

Todos os CP's possuem três partes físicas (hardware) básicas para o seu funcionamento: CPU (Unidade Central de Processamento), memória e a unidade de Entradas e Saídas (E/S), todas comunicando através de um barramento de comunicação. A CPU coordena todas as tarefas do CP e executa o programa de controle armazenado na memória. Os estados reais do processo são monitorados e amostrados pela unidade de E/S.

Toda programação é feita através de uma estação de engenharia (computador) no qual o programa é compilado e carregado na CPU para ser armazenado na memória

utilizando-se da rede local (LAN). Os CP's permitem a monitoração das unidades de entrada e saída em tempo real utilizando-se a estação de engenharia, enquanto o programa está sendo executado.

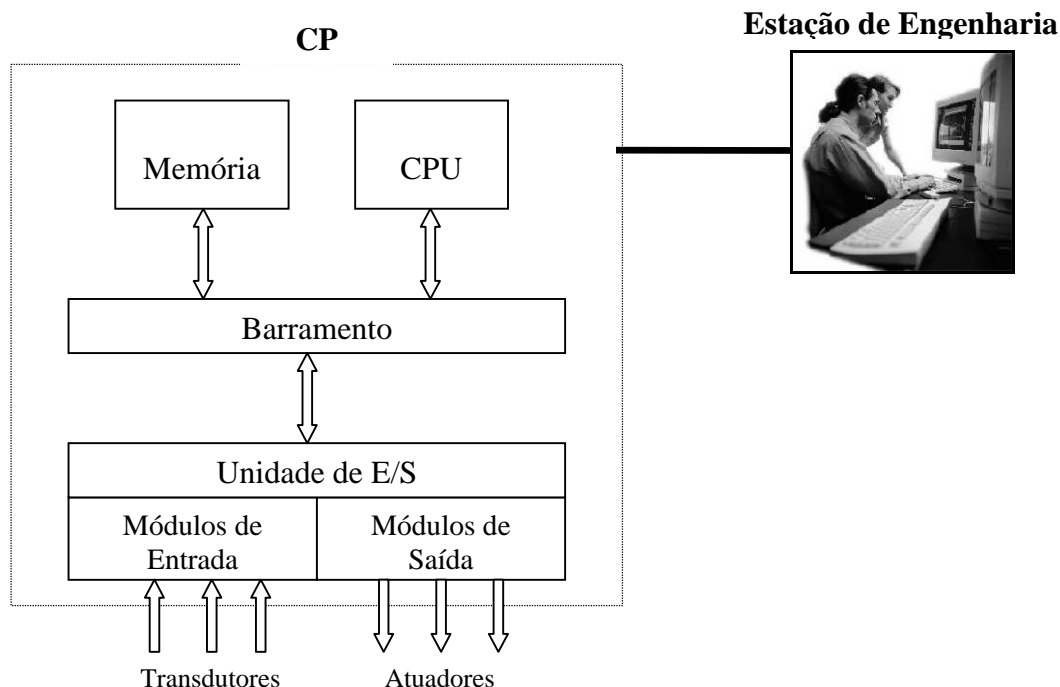


Figura 3 – Componentes de um controlador programável

1.1.2.1 Métodos de Programação

Todo programa de computador consiste em um número de instruções que define para o processador qual o próximo passo a ser executado quando o programa é “rodado”. Como os computadores somente processam informações binárias, e são muito diferentes de nossas próprias maneiras verbais de descrever ações, na programação, existem dispositivos automáticos que são usados para processar e traduzir a descrição verbal (compiladores e interpretador) requerida em uma linguagem própria do computador.

1.1.2.2 Código de Máquina e Assembler

Através da combinação das instruções de operações simples, coletar um dado, armazenar o dado, executar uma operação, etc..., pode-se obter funções complexas, utilizando os códigos de máquina.

Como os códigos de máquina são binários ou hexadecimais, a tarefa de programação é simplificada com o uso de instruções assembler (código mnemônico, ex.: ADD). Os programas assembler são escritos através de um editor, e antes que sejam executados, os códigos mnemônicos são traduzidos (assemblador) para os códigos de máquina em hexadecimal. Além da tradução do programa, o assemblador também verifica a sintaxe e calcula os saltos lógicos dentro de um programa.

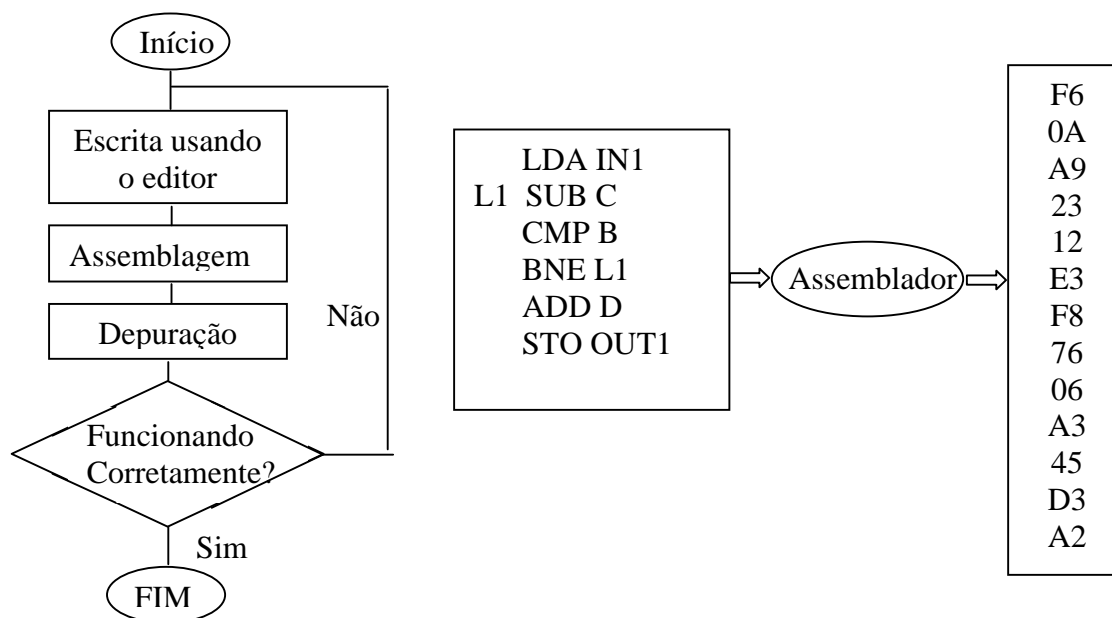


Figura 4 – Na programação de baixo nível, são usados os programas para edição, assemblagem e depuração no processo de geração do código de máquina.

Algumas desvantagens da programação em assembler consistem basicamente:

- Faz-se necessário o conhecimento do funcionamento de computadores para facilitar a programação.
- O problema deve ser devidamente estruturado para facilitar a utilização do conjunto de instruções de um determinado computador.
- O programa final é destinado a um tipo de sistema operacional, não sendo facilmente portátil para outros.

Apesar desses entraves, a linguagem assembler permite um ótimo desempenho e uma melhor utilização da memória do computador. Vantagens essas, que podem ser

determinantes em função dos recursos de sistema de controle. Por exemplo, a quantidade de instruções a serem executadas num programa pode ser limitada de acordo com a configuração da memória, ou do processador, ou do barramento, ou meio físico a ser transmitido, etc...

Assim, o assembler é chamado de linguagem de baixo nível, em função do mesmo ser próximo à forma como os computadores processam os dados.

1.1.2.3 Compilação e Interpretação

A programação é facilitada significativamente quando é feita utilizando-se de uma linguagem de alto nível, a qual é traduzida para o código de máquina através de um programa chamado de interpretador ou compilador.

O compilador primeiramente traduz todo o programa para o código de máquina antes mesmo de ser executado. O interpretador traduz o programa instrução por instrução durante a execução do programa. Isto significa que os programas compilados (C, PASCAL) têm um processamento mais rápido que os interpretados (BASIC).

As instruções das linguagens de alto nível são semelhantes às funções matemáticas, sendo, portanto mais intuitivas e fáceis de usar.

As vantagens da utilização são:

- O programador não necessita conhecer configurações físicas (*hardware*) do computador (memória, processador, etc...).
- O programa é portátil para outro computador desde que exista o compilador adequado.

As desvantagens dos programas escritos em linguagens de alto nível são o maior consumo de memória e desempenho inferior em relação às linguagens de máquina.

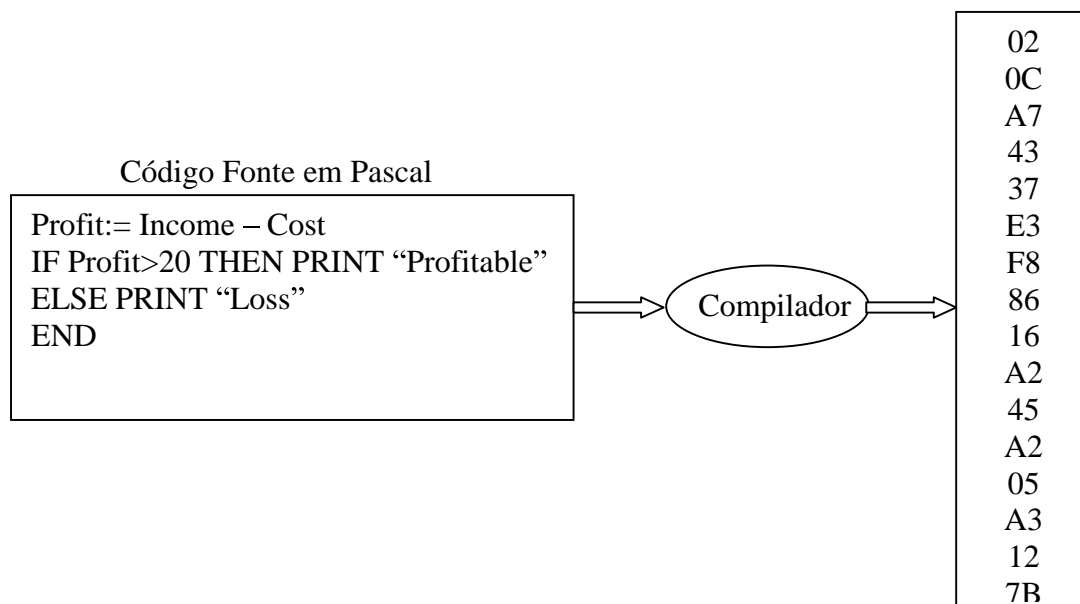


Figura 5 – Programas escritos em linguagem de alto nível são independentes do hardware e são traduzidos para o código de máquina pelo compilador.

1.1.2.4 Execução Cíclica

Sistemas de controle industrial são sistemas de tempo real, o que significa que alterações nos sinais de entrada exigem uma imediata ação no sinal de saída correspondente. Ou seja, o tempo de reação do sistema de controle deve ser condizente com as necessidades do processo controlado. Por exemplo, na Realcafé tem-se dois Torradores de Café em que as portas de proteção das 4 correias que acionam o motor do cilindro contém uma chave ON/OFF, que na lógica de controle age como uma interrupção de segurança. Quando alguém abrir as portas, o sistema irá desenergizar o motor do cilindro do Torrador. Se o controlador não processar no tempo necessário o operador poderá ficar sem os dedos ou as mãos. Consequências inaceitáveis, prejuízos imensos para a empresa.

Para garantir o atendimento às exigências de um sistema de tempo real, o programa de controle deve monitorar constantemente os sinais de entrada e responder com os sinais de saída no tempo estabelecido de acordo com a aplicação. Para possibilitar isso, o programa compilado é executado ciclicamente numa frequência específica, ou varredura (*scan*). Alterações nos sinais de entrada somente afetarão os

sinais de saída no fim de cada ciclo completo de programa. O tempo de ciclo necessário para o programa é determinado pelo atraso máximo permitido para o processo controlado.

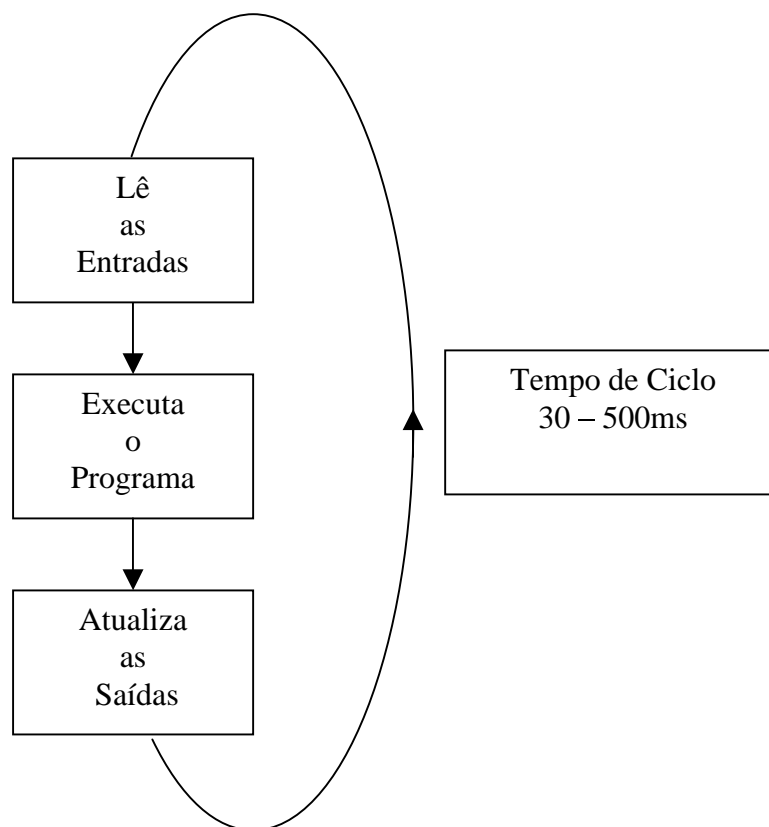


Figura 6 – Ciclo de varredura de um Controlador Programável

1.1.3 *SoftPLC*

Um problema relacionado aos CP's, é que os fabricantes utilizam *hardware* proprietário para o controlador, com uma linguagem de programação também proprietária. Apesar das funções básicas serem praticamente idênticas, as instruções possuem nomes diferentes e existem regras diferentes para sintaxe dos programas.

Isto torna impossível a comunicação e intercâmbio de programas entre sistemas de fabricantes diferentes. Infelizmente, os fabricantes ainda não estabeleceram o *SoftPLC* como um padrão industrial significando que os programas de controle desenvolvidos com um *SoftPLC* não podem ser transferidos de um para outro fornecedor. Mas tudo indica que caminham para uma solução única e otimizada, onde

os recursos serão aprendidos entre os fornecedores e o produto final será um só: Sistemas de Controle Industrial em Tempo Real.

1.2 Motivação para Sistemas Abertos

1.2.1 Diferentes Dialectos de Programação

O CP é um dos componentes mais críticos da indústria atual. Com a utilização dos sistemas de controle na maioria das indústrias, incluindo aplicações que exigem segurança, é muito importante que os programas possam ser facilmente entendidos por uma grande parte dos profissionais do chão de fábrica. Além do programador, o programa de controle deve ser de fácil entendimento para todos os técnicos, engenheiros e gerentes de processo.

Por quase duas décadas o mercado tem sido dominado por poucos fabricantes que oferecem soluções muito parecidas, porém com particularidades nos dialectos de programação. Muitos usuários de CP's têm decidido eleger no mínimo três fornecedores, com o objetivo principal de minimizar o risco. Em aplicações reais, isto implica em um maior custo devido ao retrabalho e problemas de comunicação entre produtos de diferentes fabricantes.

1.2.2 Qualidade de *Software*

Quanto mais tarefas da indústria de manufatura e de processos são automatizadas, maiores e mais complexos se tornam os programas, dificultando o gerenciamento dos mesmos. Em muitos casos, são necessários mais de um programador para o desenvolvimento do programa para automação industrial.

A experiência da ATAN [4] mostra que o risco devido a erros de programação cresce exponencialmente com o número de programadores envolvidos e com o tamanho do programa. E que as novas plantas industriais apresentam problemas por um longo período após a instalação. As falhas podem parar a linha de produção ou causar danos aos equipamentos e produtos e acidentes aos operadores.

Um *Software* de controle de boa qualidade representa um custo maior, e geralmente é desenvolvido pela própria indústria ou por empresas integradoras de

sistemas dedicados ao ambiente industrial. Em ambos os casos, a produção de *software* e seu custo, competem no mercado aberto, logo os fornecedores de *software* são motivados a buscar ferramentas e métodos de desenvolvimento mais eficientes.

A grande maioria dos programas de controle é escrita utilizando pacotes de *Software* proprietário dos fabricantes de produtos de controle, que não dispõem de recursos adequados para trabalhar com módulos para reutilização de código e para documentação. Portanto a qualidade de *software* é dependente da capacidade intelectual do programador.

Antes da norma IEC 61131-3 uma boa engenharia de *Software* era a meta principal das aplicações de controle.

1.2.3 Custo de *Software*

Na última década, a padronização dos pacotes de *Software* para os computadores pessoais, tais como processadores de texto e planilhas eletrônicas, tornou-se muito popular, vendendo milhões de cópias e proporcionando aos fornecedores a redução drástica dos preços de produtos.

A distribuição pela Internet ampliou ainda estes limites e hoje são muitas as aplicações padrões disponíveis como licenças *freeware* e *shareware*, quase sem custo.

De forma contrária, os programas para aplicações de controle são adaptados para características específicas de uma determinada planta industrial significando que o custo total do desenvolvimento recorre sobre um único usuário.

Muitos usuários (empresas que contratam outras especializadas para o desenvolvimento do *Software* de controle industrial) consideram difícil absorver o custo total do desenvolvimento do programa de controle. Um usuário sem experiência no desenvolvimento de programas consegue apresentar apenas uma pobre descrição funcional para o desenvolvedor, atendendo parcialmente aos requisitos do usuário.

O crescente aumento da velocidade de desenvolvimento do *hardware* tem levado a uma constante queda nos preços dos computadores pessoais que tem desempenho comparável aos antigos modelos de processamento centralizados (*mainframes*) nas quais as interações interfaces homem-máquina eram acessíveis apenas àqueles que possuíam algum conhecimento específico. Assim o custo total da

instalação é mais determinado pelo tempo de desenvolvimento do programa de controle.

O maior peso dos projetos está na padronização e reutilização dos programas do que na busca de uma ótima configuração de um controlador.

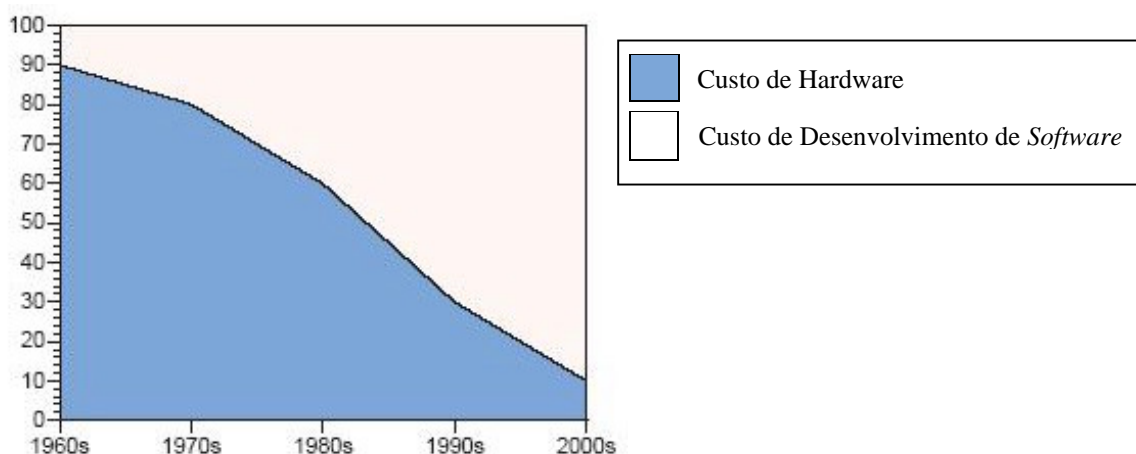


Figura 7 – Custo do *Software* versus Hardware

A automação de plantas e máquinas pode ser perigosa para o operador ou para os produtos se o *software* de controle apresentar erros críticos. Portanto, o *software* depende de um procedimento rigoroso para teste e validação, e bastante demorado para aplicações reais se o trabalho tem que ser feito com o processo em funcionamento.

1.2.4 Portabilidade de Aplicações

O computador pessoal juntamente com o sistema operacional *Windows* é um padrão de fato para as aplicações de escritório de todo o mundo. A principal razão para esta grande difusão do uso dos PC's é a compatibilidade de *Software*. Os programas aplicativos desenvolvidos para o *Windows* podem ser usados em quase todos os PC's espalhados do mundo.

Os usuários dos produtos (*softwares*) não querem um mercado monopolista onde uma empresa fornece um produto que somente ela detém da tecnologia de informação. Mas querem, um mercado onde todos os fabricantes possam apropriar-se dos conceitos estabelecidos e padronizados e a partir daí estabelecer o seu produto que poderá ter recursos mais aprofundados diferenciando do seu concorrente. Resultando

assim, numa economia para as empresas que adquirem seus produtos e aumentando o ramo de aplicações industriais.

Mais de 25 anos após a introdução dos controladores programáveis, este mercado ainda carece de uma padronização internacional similar ao que ocorreu com os PC's. Muitos fornecedores usam um dialeto de programação proprietário que funciona apenas com o seu *hardware* proprietário.

Do lado oposto, a quase totalidade das indústrias usuárias de controladores programáveis tem uma alta demanda pela portabilidade do *software* do sistema de controle. Desde que o custo de desenvolvimento de um *software* bem testado é muito maior que o custo do *hardware*, é cada vez maior a necessidade de se portar as aplicações existentes de uma plataforma de *hardware* antiga para novos sistemas.

Muitos observadores consideram um mistério o fato de serem necessários mais de 25 anos para o mercado de controladores programáveis promover o estabelecimento de uma padronização de controladores proposta pela norma IEC 61131-3.

2 A NORMA IEC 61131

2.1 Propósito da Norma IEC 61131-3

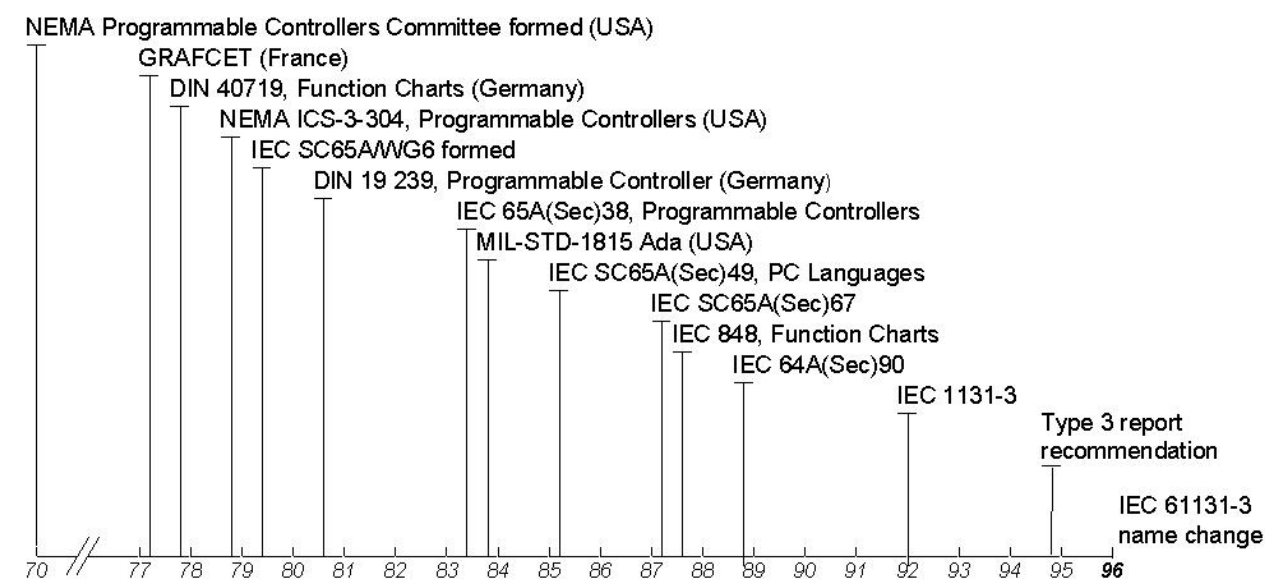
Durante os últimos dez a quinze anos uma escala larga de diferentes técnicas de programação foi utilizada para escrever programas para aplicações de controle industriais e para Controladores Lógicos Programáveis (CLPs). As aplicações de controle foram desenvolvidas em BASIC, FORTH, em C, no Inglês Estruturado, na Lista de Instruções e em outras numerosas outras “linguagens proprietárias” incluindo vários dialetos da programação LADDER. Infelizmente, a única coisa que pode ser dita de todas estas linguagens de programação é que são todas diferentes. Para as pessoas envolvidas com tais sistemas desde técnicos, o pessoal da manutenção, projetistas de sistemas aos gerentes de planta, isso resulta no uso ineficiente do tempo e do dinheiro. Há claramente um desperdício dos recursos humanos envolvidos no treinamento da equipe de funcionários para habilitação em muitas linguagens de controle.

Felizmente a comunidade industrial internacional reconheceu que um novo padrão para controladores lógicos programáveis foi requerido. Um grupo de trabalho do *International Electrotechnical Commission* (IEC – Comissão Eletrotécnica Internacional) foi organizado em 1979 para estudar e avaliar o projeto completo dos controladores lógico programáveis, incluindo o projeto do *hardware*, a instalação, os testes, a documentação, a programação e as comunicações. O IEC como uma organização irmã da *International Standardisation Organization* (ISO – Organização Internacional de Normatização) fundada em Genebra na Suíça, tem comitês e grupos de trabalho formados a partir de representantes da maioria de países industriais do mundo que põem a frente procedimentos de padronização.

Durante os anos 1990, O IEC publicou várias partes do padrão IEC 61131 que cobre o ciclo completo dos CLP's, que são:

- Parte 1 – Definição da informação geral, da terminologia básica e dos conceitos; *Publicado em 1992.*
- Parte 2 – Exigências de equipamento e testes eletrônicos e testes mecânicos de construção e verificação; *Publicado em 1992.*

- **Parte 3 – Estrutura do *Software* do CLP, execução do programa e linguagens de programação; Publicado em 1993.**
- Parte 4 – Guia de orientação ao usuário na seleção, instalação e manutenção de CLP's; *Publicado em 1995.*
- Parte 5 – Facilidade do *Software* em especificação de mensagens de serviços a comunicar-se com outros dispositivos usando as comunicações baseadas em MAP (*Manufacturing Messaging Services*); *Publicado em 1998.*
- Parte 6 – Comunicação via facilidade do *Software* fieldbus para comunicação de PLC's utilizando IEC fieldbus; *Aguardando fechamento do padrão fieldbus.*
- Parte 7 – Programação utilizando Lógica Nebulosa (Fuzzy); *Publicada em 1997.*
- Parte 8 – Guia para implementação das linguagens;



Source: Dr. J. Christensen

Figura 8 – Evolução da norma.

2.2 Certificação de Produtos


A associação independente PLCOpen [7] existe desde 1992 com o intuito destinada a promover e suportar o uso da norma IEC 61131-3. A norma IEC apresenta uma tabela de características padrões e exige apenas que os fabricantes indiquem como

os produtos são aderentes a estas características para serem considerados como controladores IEC. As atividades desenvolvidas pela PLCOpen:


- Especificação de elementos mandatórios para certificação de produtos em diferentes níveis de conformidade com a norma IEC 61131-3;
- Desenvolvimento de um *software* de teste baseado nas especificações de certificação;
- Realização de testes e certificação de produtos conforme a norma IEC 61131-3;
- Divulgação da norma e realização de eventos para estimular a adoção da norma.
- Elaboração de bibliotecas de funções e blocos funcionais padronizados para atendimento às necessidades dos usuários da norma;
- Atuação junto às entidades normatizadoras e usuários para definição de melhorias e sugestões para revisão da norma junto ao IEC;
- Complementação dos aspectos não cobertos pela norma IEC 61131-3.

Os produtos testados e aprovados pela PLCOpen recebem um selo de conformidade de acordo com o nível de certificação obtido:


➤ Nível Básico (*Base Level*):

	<p>Define as características mínimas a serem obedecidas para que os produtos sejam considerados como aderentes à norma IEC 61131-3 e que utilizam a sintaxe padrão;</p>
---	---

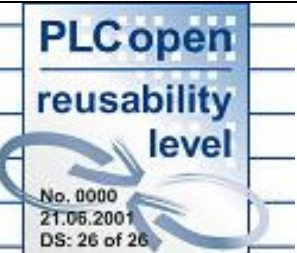
➤ Nível de Portabilidade (*Portability Level*):

	<p>Define os requisitos para que alguns elementos de <i>Software</i> (blocos) possam ser portados entre diferentes produtos compatíveis com este nível;</p>
---	---


➤ Nível de Conformidade (*Conformity Level*):

 <p>No. 0000 ST 21.06.2001 Datatypes supported: 26 of 26</p>	<p>Define a conformidade quanto a quantidade de tipos de dados declarados baseado na IEC 61131-3;</p>
---	---

➤ **Nível de Re-utilização (*Reusability Level*):**

 <p>No. 0000 21.06.2001 DS: 26 of 26</p>	<p>Define o quanto o produto é reutilizável baseado na IEC 61131-3;</p>
---	---

➤ **Nível de Controle de Movimento (*Motion Control*):**

	<p>Certifica Blocos Funcionais de acordo com a especificação do Controle de Movimento;</p>
--	--

➤ **Full Compliance Level:** Define os requisitos para que todos os elementos de *Software*, incluindo toda uma configuração, sejam conformes e compatíveis entre diferentes produtos certificados neste nível.

2.3 Introdução

A norma proporciona que sejam desenvolvidos ambientes de programação capazes de decompor programas complexos em diferentes elementos de *software*, os quais possuem uma interface padronizada e bem definida entre os mesmos. São definidas 5 linguagens de programação para o desenvolvimento de módulos ou componentes de *software*.

A programação é orientada para o desenvolvimento de programas a partir da abordagem de cima para baixo (*top-down*) e de baixo para cima (*bottom-up*), baseada em três princípios:

- Modularização – decomposição de qualquer sistema, complexo ou não, em partes menores capazes de serem gerenciáveis;
- Estruturação – forma hierárquica utilizada para a programação em níveis facilitando a modularização e reutilização de blocos;
- Reutilização – de funções, de blocos funcionais ou programas.

2.4 Modelo de Software IEC

Quando um Controlador Programável está processando o programa carregado na memória são necessárias as seguintes interfaces:

- **Interface de Entrada/Saída:** Permite o acesso aos dispositivos para leitura dos sinais do processo como pressões, temperaturas, etc..., assim como fazer o comando dos dispositivos de campo, motores, atuadores e outros.
- **Interfaces de Comunicação:** Troca de informações com outros CP's. IHM's, etc...
- **Interfaces de Sistema:** Consiste na interface entre o programa do CP e o *hardware*, para garantir o seu correto funcionamento. São utilizados os serviços do sistema, os quais são uma combinação do *hardware* e *firmware* do CP.

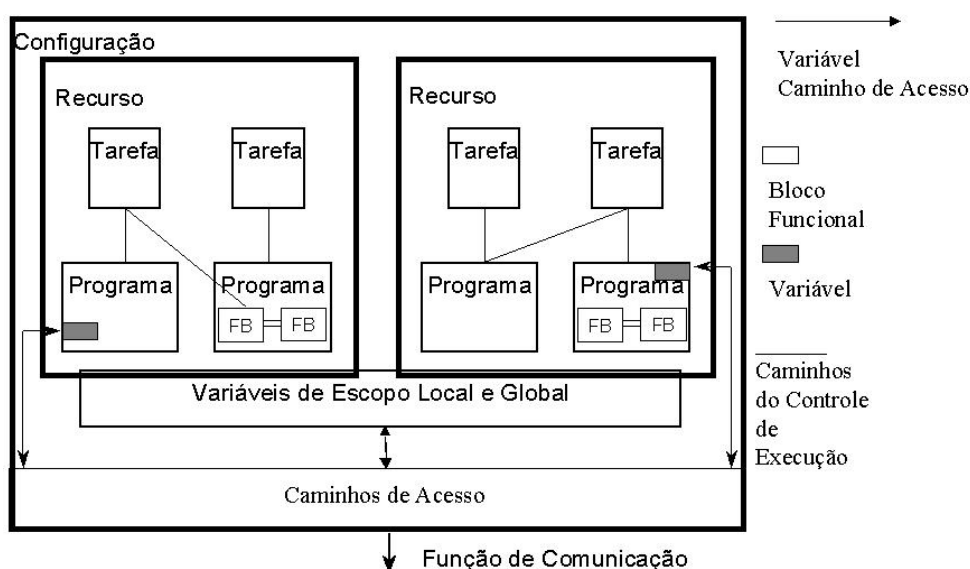


Figura 9 – Modelo de Software da norma IEC 61131-3

2.4.1 Configurações

No nível mais alto, o *software* para um sistema de controle está contido numa configuração. Estes elementos podem constituir de CP's, IHM's, dispositivos de comunicação, etc..., desde que os mesmos possuam interfaces compatíveis com a norma.

2.4.2 Recursos

Dentro de cada configuração podem existir um ou mais recursos que basicamente são quaisquer elementos com capacidade de processamento, responsáveis pela execução dos programas. Um recurso pode existir fisicamente ou ser apenas uma máquina virtual. Neste último caso, mais de um recurso pode compartilhar independentemente a capacidade de processamento de um computador, ou seja, podem existir uma IHM e um *SoftPLC* no mesmo computador.

2.4.3 Programas

Um programa IEC pode ser construído a partir de diferentes elementos de *Software*, cada qual escrito em qualquer uma das diferentes linguagens da norma. Um programa consiste em um agrupamento lógico dos elementos necessários à todas as linguagens de programação, para o processamento dos sinais desejados. Um programa pode acessar as variáveis de E/S e comunicar com outros programas.

2.4.4 Tarefas (*Tasks*)

Uma tarefa pode ser configurada para controlar a execução de programas ou blocos funcionais, de forma periódica ou engatilhada por eventos (*triggers*). Uma simples execução de um programa ou de um bloco funcional implica que todos os elementos de *software* ^[1] dentro dos mesmos serão processados de uma só vez (ex: Se

dentro de um Bloco Funcional Derivativo (veja nas figuras 12 e 13), tivermos uma programação em Ladder. Tudo que está dentro das POU's (programa e bloco funcional, nesse caso), será executado também). A norma IEC não define nenhum mecanismo implícito para execução de programas. Ou seja, um programa ou bloco funcional ficará aguardando a sua execução até que seja associado a uma determinada tarefa e esta seja ativada por uma execução periódica ou por um determinado evento.

A necessidade de se executar programas em períodos diferentes tem por objetivo atender as exigências de controle de tempo de resposta do processo e de otimizar o uso da capacidade de processamento do CP.

Por exemplo, um torrador de café faz todo ciclo de torra a cada 8 minutos. Já a porta de segurança do acoplamento polia + correias do cilindro do torrador tem a lógica de controle executada a cada 200ms para não expor perigo de acidentes aos funcionários. Assim uma interrupção é programada na lógica de funcionamento, fazendo com que o motor do cilindro pare na ocorrência citada.

A cada tarefa podemos atribuir um período de execução e uma prioridade (em ordem decrescente), onde 0 (zero) é a maior prioridade.

2.4.5 Blocos Funcionais

O conceito de bloco funcional é um dos mais importantes da norma IEC 61131-3, para permitir o projeto de *software* de forma hierárquica e estruturada. Blocos funcionais podem ser utilizados para a criação de elementos de *software* ^[1] totalmente reutilizáveis, até programas complexos.

As principais características dos blocos funcionais são que estes possuem um conjunto de dados, os quais podem ser alterados por um algoritmo interno. Somente o conjunto de dados, é mantido na memória para uma determinada instância do bloco funcional. Os dados possuem persistência, ou seja, possuem estados internos que são mantidos entre uma execução e outra. Blocos funcionais podem ser utilizados para a criação de outros blocos funcionais (blocos derivativos), aumentando ainda mais a capacidade de reutilização do *software*.

2.4.6 Funções (*Functions*)

Funções são elementos de *software* que não aparecem no modelo de *software*. Funções não possuem persistência, existindo apenas em tempo de execução, assim como sub-rotinas. Portanto, não possuem estados internos, ou seja, sempre produzem o mesmo resultado para o mesmo conjunto de entradas.

No apêndice B, tem-se a declaração de todas as Funções e os Blocos Funcionais classificados no *Control Builder F*.

2.4.7 Variáveis de Escopo Local e Global

A norma exige a declaração de variáveis dentro de diferentes elementos de *Software*, tais como programas e blocos funcionais. As variáveis podem utilizar nomes com significado abrangente (simbólicos) e serem de diferentes tipos de dados. As variáveis podem ser de alocação dinâmica e associadas a posições de memória (representação direta). O escopo das variáveis é local aos elementos de *software* que as declara, permitindo acesso dentro do próprio elemento que pode ser uma configuração, recurso, programa, bloco funcional ou função. Variáveis também podem ser de escopo global, sendo acessadas por todos os elementos de *software*. Variáveis globais podem ser declaradas dentro de uma configuração ou recurso.

2.4.8 Variáveis de representação direta

Posições de memória do CP podem ser acessadas usando variáveis de representação direta. A representação direta do endereço de memória permite a leitura e escrita de dados em posições conhecidas de memória, tais como entradas, saídas e endereços internos (o *CBF* não possui esse endereçamento). As variáveis de representação direta (variáveis locais) têm seu uso restrito aos programas. A notação utilizada é padronizada para permitir a portabilidade.

Todas começam com o caractere % seguido de uma ou duas letras, onde a memória do CP pode ser dividida em três regiões lógicas:

Primeira Letra	Interpretação
I	Input: Recebe os valores das variáveis analógicas e discretas dos módulos de entrada.
Q	Output: Para armazenar os valores a serem escritos nos dispositivos externos.
M	Memória Interna: Armazena valores intermediários.

Tabela 1 – Primeira letra escrita para representar a variável de representação direta.

Segunda Letra	Interpretação
X	Bit
B	Byte (8 bits)
W	Word (16 bits)
D	Double word (32 bits)
L	Long word (64 bits)

Tabela 2 – Segunda letra escrita para representar a variável de representação direta.

Se a segunda letra não é mencionada, é considerada como bit (X).

Exemplos:

%IX100 (*Memória de endereço 100 e entrada bit*)

%IW122 (*Memória de endereço 122 e entrada palavra de 16 bits*)

%MW132 (*Palavra de endereço de memória 132*)

%IW10.1.21 (*Poderia representar uma remota no campo: Rack 10, Módulo 1, Canal 21*)

%QL188 (*Memória de endereço 188 e entrada palavra longa de 32 bits*)

2.5 Caminhos de Acesso (*Access Paths*)

Os caminhos de acesso permitem a transferência de dados entre diferentes configurações. Cada configuração pode definir um número de variáveis (leitura, escritas ou ambos) para acesso por configurações remotas. A norma assume que estarão disponíveis mecanismos de comunicação para a troca de informação, não abordando a forma a ser adotada. A parte 5 da norma (IEC 61131-5) define os blocos funcionais que proverão os serviços para leitura e escrita de variáveis em configurações remotas.

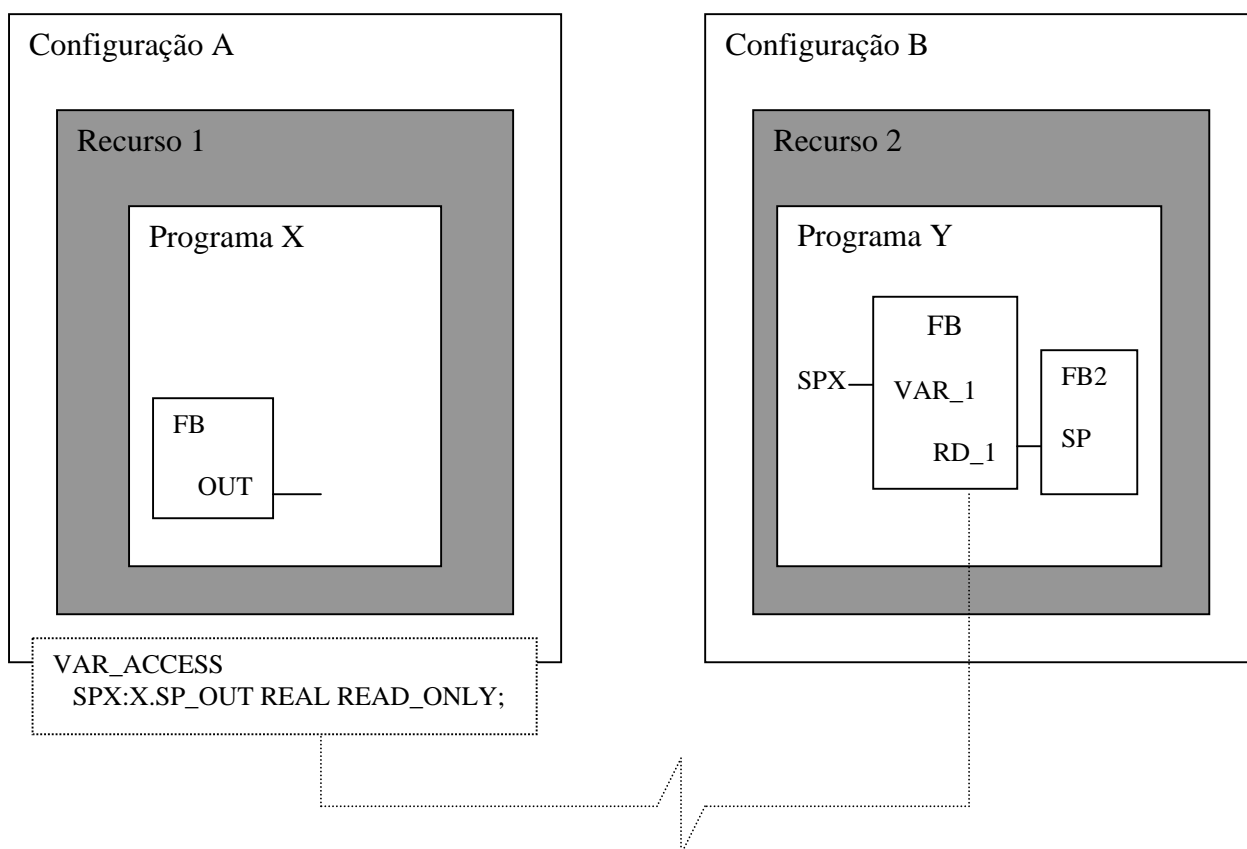


Figura 10 – Comunicação externa (remota) usando caminho de acesso

2.6 Fluxo de Controle

A norma IEC não define os mecanismos para controle de execução dos elementos de *Software*, os quais são dependentes da implementação. Entretanto, são definidos os comportamentos na partida (*start-up*) e parada (*shut-down*) do sistema:

2.6.1 Partida

1. Quando uma configuração parte, todas as variáveis globais são inicializadas e todos os recursos são ativados;
2. Quando um recurso parte, todas as variáveis dentro do recurso são inicializadas e todas as tarefas são habilitadas;
3. Uma vez habilitadas as tarefas, todos os programas e blocos funcionais associados serão executados, quando a tarefa estiver ativa.

2.6.2 Parada

1. Quando uma configuração pára, todos os recursos da mesma também param;
2. Quando um recurso pára, todas as tarefas são desabilitadas, interrompendo a execução dos programas e blocos funcionais.

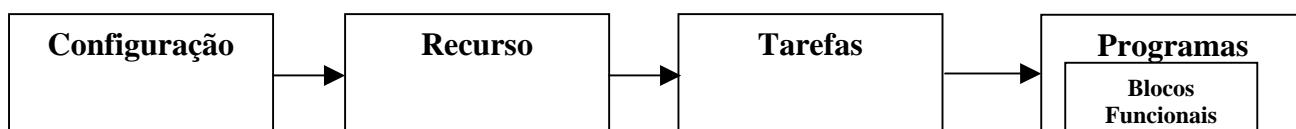


Figura 11 – Partida/parada do fluxo de controle de um *Software* IEC.

2.7 Unidade de Organização de Programa (POU)

A norma IEC descreve programas, blocos funcionais e funções como Unidades de Organização de Programas (*POU – Program Organization Units*). A principal característica destes elementos de *Software* é que os mesmos podem ser reutilizados no desenvolvimento de uma aplicação. A reutilização através de cópias ou réplicas de programas e blocos funcionais é feita criando-se instâncias, as quais são declaradas pelo tipo e possuem uma área de dados separada na memória do CP. Funções não precisam ser instanciadas, porque só existem em tempo de execução.

A norma IEC estimula a reutilização desde o nível mais macro, com programas, até o nível mais micro, com blocos funcionais e funções. **A recursividade não é permitida dentro de um POU, para garantir a estabilidade da aplicação.**

Tipo de POU	Replicado como	Comentário
Programa	Instância de Programa	Permite reutilização no nível macro, como programas para reatores, transportadores, caldeiras, etc...
Bloco Funcional	Instância de Bloco Funcional	Possibilita a reutilização desde simples, a complexas estratégias de controle e algoritmos, como controle PID, filtros, motores, etc...
Função	Função	Usada para tratamento comum de dados, como lógica E, OU, seno, cosseno, soma, etc...

Tabela 3 – Tipos de Unidades de Organização de Programa.

Uma importante característica da norma é a ênfase dada ao projeto de forma hierárquica. Isto significa que um sistema de controle pode ser dividido em níveis de complexidade de cima para baixo e estruturado de baixo para cima em função das necessidades de controle. Isto implica na utilização de blocos funcionais e funções padrões, através de bibliotecas que podem ser fornecidas pelo fabricante do CP, ou pela criação de blocos específicos definidos pelo usuário para tender às necessidades do processo. O uso de bibliotecas nos blocos padrões permite a portabilidade de

soluções para diferentes sistemas de controle. Na figura 12 temos a extratificação do Bloco Funcional PID_Fuzzy.

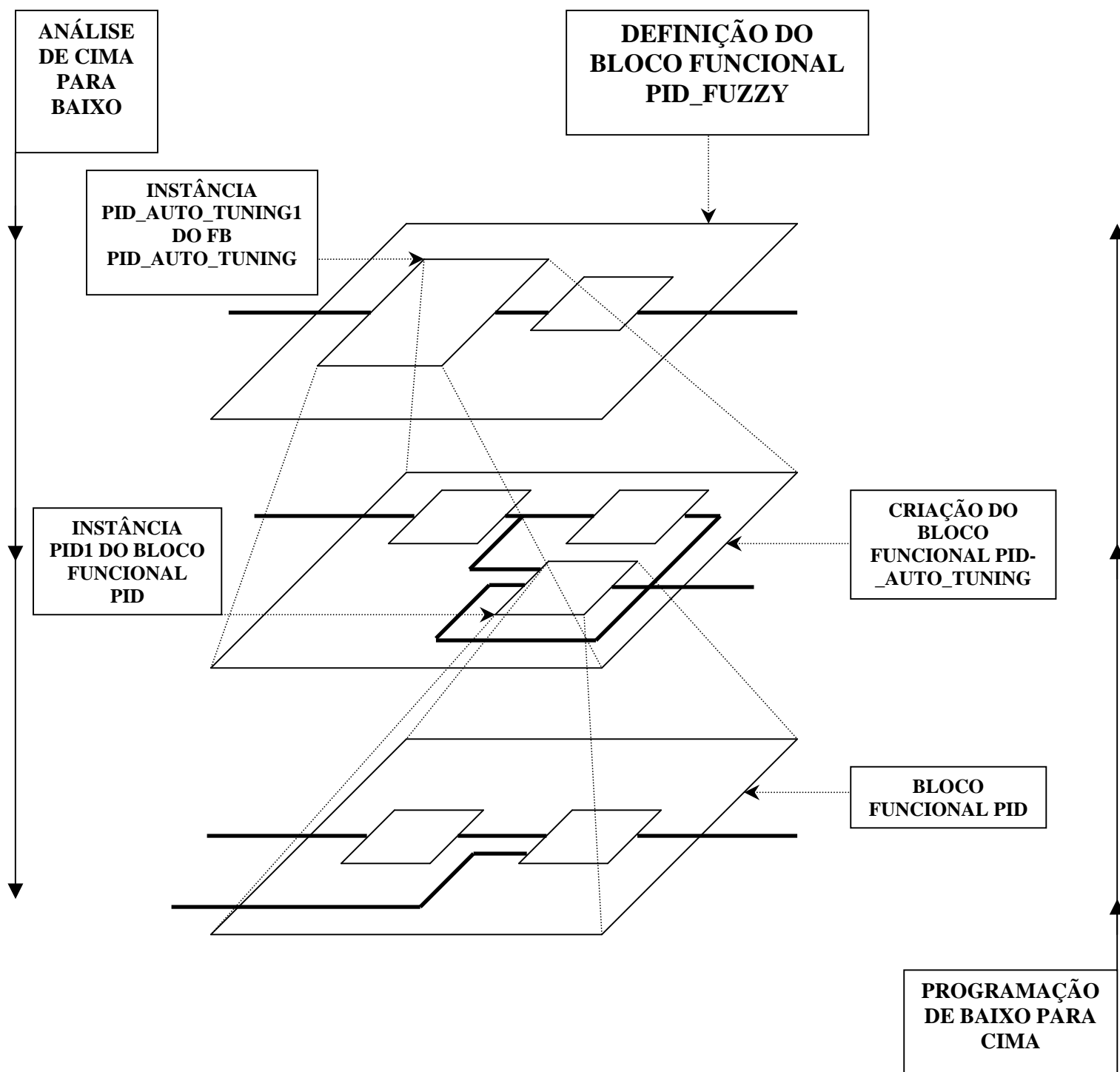


Figura 12 – Decomposição hierárquica do Bloco Funcional PID_Fuzzy.

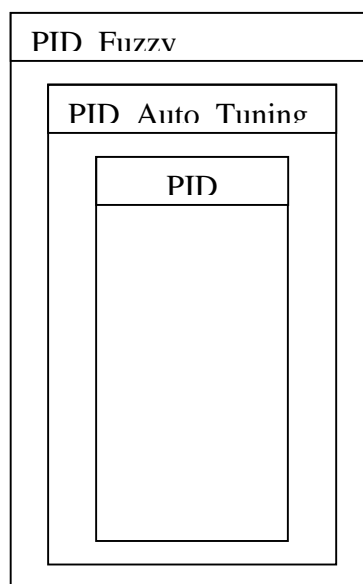


Figura 13 – Representação do encapsulamento PID > PID_Auto_Tuning > PID_Fuzzy.

A despreocupação do entendimento de um algoritmo interno de um Bloco Funcional e sim o entendimento da aplicação e utilização daquele bloco com as variáveis de entrada e saída para a utilização no programa conforme exemplificado nas figuras 13 e 14 do PID_Fuzzy, é uma vantagem muito importante da norma.

2.8 Processamento Multitarefas

Quando existem tarefas múltiplas são atribuídos diferentes intervalos e prioridades de execução utilizando-se os métodos de **escalonamento preemptivo e não-preemptivo**. O método a ser adotado e os parâmetros de tempo definidos podem alterar significativamente o desempenho de processamento do CP e o comportamento do sistema.

2.8.1 Escalonamento não-preemptivo

É quando uma tarefa sempre é executada do início ao fim, sem interrupções. Ao término desta, a próxima com maior prioridade à espera do processador será escalonada. Caso haja empate na prioridade, a tarefa com maior tempo de espera é escalonada.

O intervalo entre a execução de tarefas pode variar muito neste tipo de escalonamento. Uma tarefa que demore um pouco mais que o ciclo normal irá atrasar todas as demais tarefas, tornando-se impossível prever com exatidão quando uma das

demaís tarefas será executada e caracteriza o sistema como não-determinístico. Esse método apresenta piores respostas de sistemas de controle.

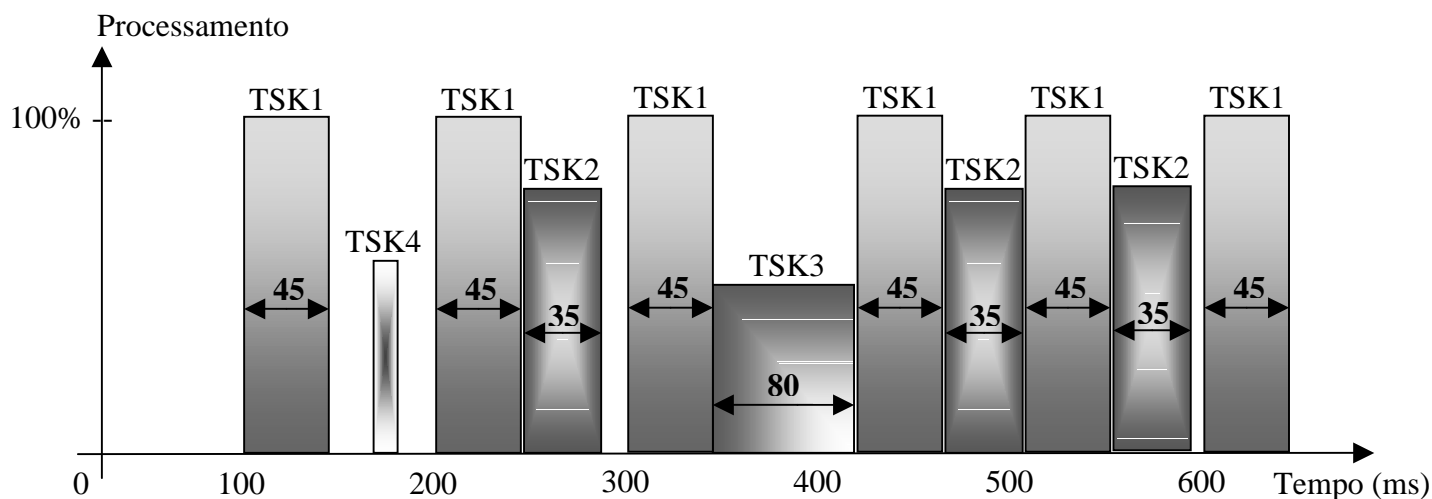


Figura 14 – Exemplo de processamento não-determinístico.

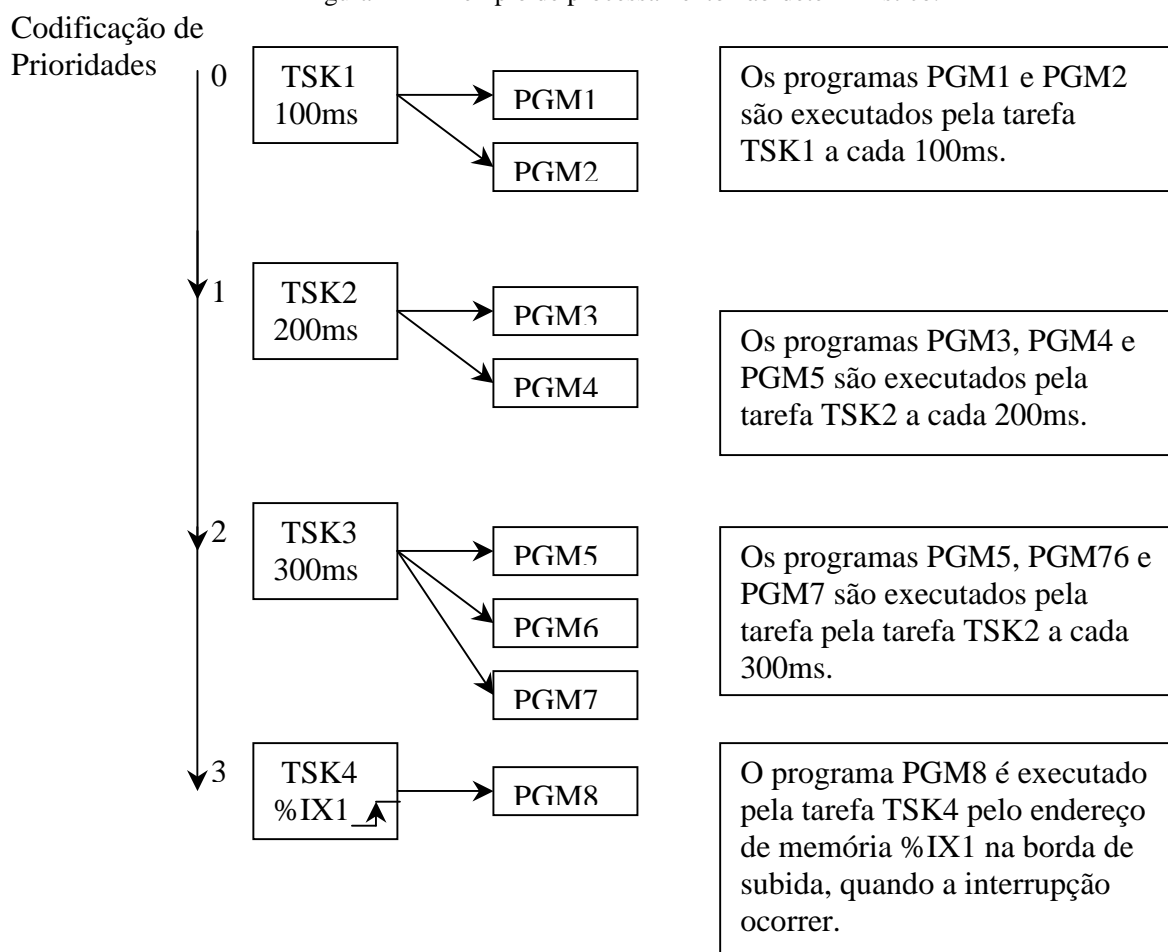


Figura 15 – Esquema da codificação de prioridades das tarefas e distribuição dos programas, bem como sua execução.

A tarefa 3 (TSK3) atrapalha a execução da tarefa 1 (TSK1) e conseqüentemente das demais em 400ms e adianta TSK1 para o final de TSK3.

2.8.2 Escalonamento preemptivo

É quando o intervalo de uma tarefa de maior prioridade vence, a tarefa em execução sofre preempção (é suspensa) e a tarefa de maior prioridade passa a ser executada instantaneamente. Quando a tarefa de maior prioridade termina, a tarefa suspensa volta sua execução de onde parou.

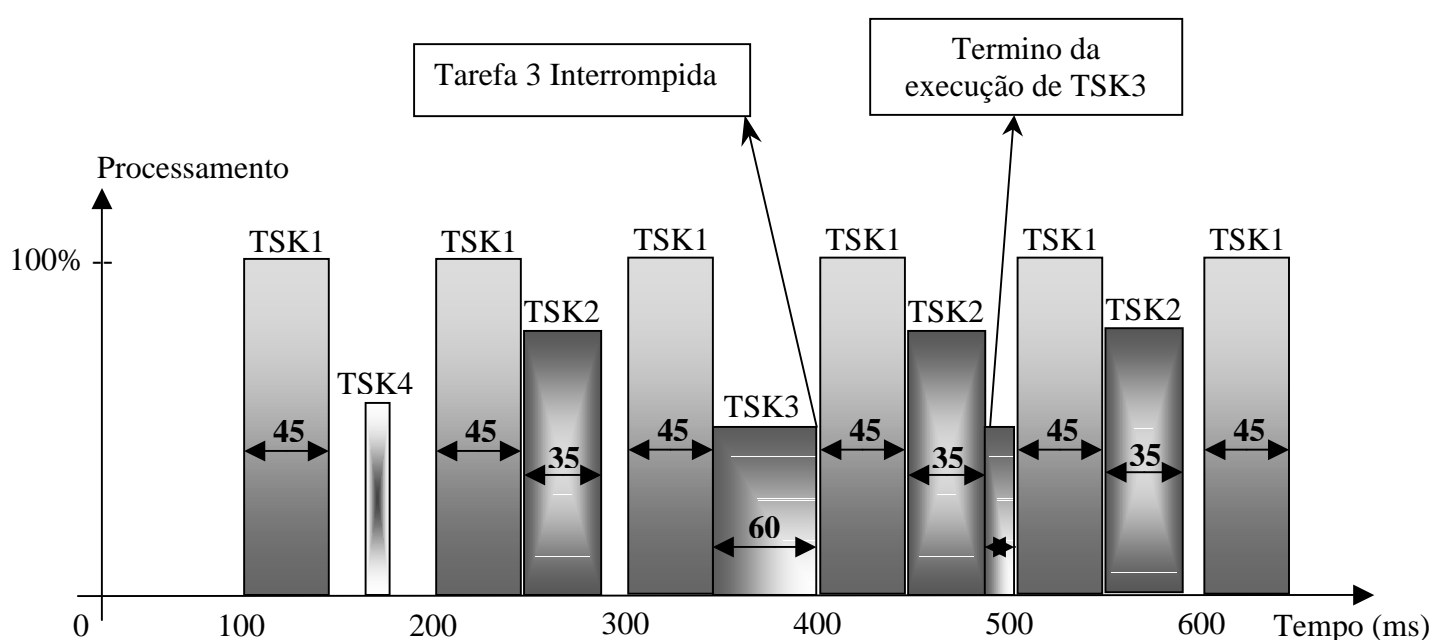


Figura 16 – Mesmo exemplo no método determinístico.

A tarefa 3 foi interrompida em 400ms e terminada sua execução quando atender a codificação de prioridades das demais tarefas.

A parametrização do intervalo de execução das tarefas deve acontecer de acordo com a aplicação do programa em questão. Para o melhor desempenho do processador deve-se escolher uma tolerância de tempo mínimo de execução tal que não afete os resultados empíricos de campo.

2.8.3 Otimização do uso da CPU do controlador;

Através da configuração ideal do tempo de ciclo de varredura de uma tarefa (limite de tempo estabelecido de acordo com a aplicação dos programas vinculados à tarefa) pelo escalonamento em método preemptivo (figura 22) do processamento multitarefas, consegue-se estabelecer um processamento ótimo e rápido da Unidade Central de Processamento.

3 LINGUAGENS DE PROGRAMAÇÃO IEC61131-3

3.1 Introdução

A norma IEC 61131-3 define 5 linguagens de programação:

Texto Estruturado (ST)	Textuais
Lista de Instruções (IL)	
Diagrama de Blocos Funcionais (FBD)	Gráficas
Diagrama Ladder (LD)	
Sequenciamento Gráfico de Funções (SFC)	

Tabela 4 – Linguagens de Programação.

SFC			
ST	IL	LD	FBD
TEXTUAIS		GRÁFICAS	

Tabela 5 – Representação hierárquica das linguagens.

As linguagens ST, IL, FBD e SFC podem ser usadas dentro de blocos de ação e em transições da programação SFC.

A norma IEC 61131-3 define elementos comuns às 5 linguagens de programação com o intuito de padronizar o correto entendimento como variáveis e tipos de dados para permitir a utilização de qualquer linguagem de programação.

3.2 Texto Estruturado (ST)

É uma linguagem de alto nível com sintaxe similar ao Pascal (ISSO 7185), desenvolvida especificamente para controle industrial usada para descrever o comportamento de:

- Funções;
- Blocos Funcionais;
- Programas;
- Passos, ações e transições da linguagem SFC.

É uma linguagem de fácil assimilação para os desenvolvedores de programas, pois permite uma fácil interpretação pelo uso de identificadores de fácil entendimento, associados a comentários. É muito útil para o desenvolvimento de cálculos aritméticos complexos, pois é só digitar a fórmula em questão.

3.2.1 Operadores

A norma IEC define uma faixa de operadores padrões para operações aritméticas e booleanas.

Operador	Descrição	Precedência
(...)	Expressão com parêntesis	Maior
Função (...)	Lista de parâmetros de uma função	:
**	Exponenciação	
-	Negação	
NOT	Complemento booleano	
*	Multiplicação	
/	Divisão	
MOD	Operador de módulo	
+	Soma	
-	Subtração	
<,>,<=,>=	Comparação	
=	Igualdade	
<>	Desigualdade	
AND,&	E booleano	
XOR	OU Exclusivo booleano	
OR	OU booleano	
		Menor

Tabela 6 – Operadores padrões.

Quando os operadores têm a mesma precedência, eles são avaliados da esquerda para a direita. Expressões entre parêntesis têm a maior precedência, ou seja, devem ser avaliadas antes das demais, de dentro pra fora.

3.2.2 Exemplo 1

Considere os valores de velocidade e pressão:

Velocidade1	50.0
Velocidade2	60.0
Pressao	30.0

Tabela 7 – Velocidade 1 e 2 e pressão.

Taxa:=Velocidade1/10.0 + Velocidade2/20.0 – SQRT(Pressao + 6.0);
Taxa:=2.0;

3.2.3 Exemplo 2

StartUp:=A AND B AND C AND D;

Se A=FALSE, então o valor FALSE é atribuído à variável StartUp e a avaliação da expressão é interrompida.

3.2.4 Comandos da Linguagem Texto Estruturado

- **Cálculos Aritméticos**
- **Comando condicional IF THEN ELSE**
- **Comando condicional CASE**
- **Comando de repetição FOR ... DO**
- **Comando de repetição WHILE ... DO**
- **Comando de repetição REPEAT ... UNTIL**
- **EXIT**
- **RETURN**

No *Control Builder F*, todos os operadores do Texto Estruturado podem ser inseridos apertando o botão direito do *mouse*, conforme a figura 38.

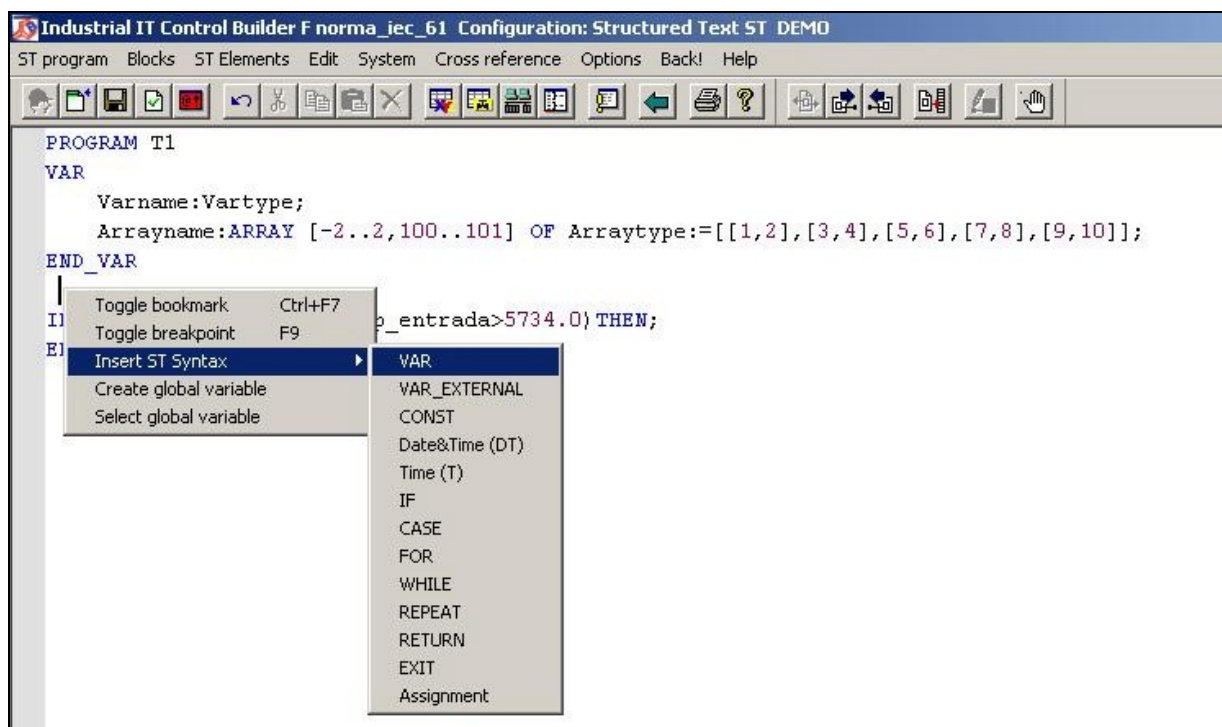


Figura 17 – Operadores padrões do ST no CBF.

3.2.5 Chamada de Blocos Funcionais

Podemos chamar ou invocar um bloco funcional através de seu nome, tendo como parâmetros, valores válidos para variáveis de entrada e saída.

3.2.6 Limites do Sistema

O número de variáveis locais é limitado a 65526 para cada programa do ST no *Control Builder F*.

3.2.6.1 Ocupação da Memória

O Texto Estruturado é uma linguagem de alto nível que quando traduzida no código de máquina é substancialmente mais longo do que o código de fonte. Em disposições multidimensionais o número de elementos individuais aumenta substancialmente. A matriz [1.. 100] X [1.. 100] tipo de dado REAL contém 10.000 elementos e requer aproximadamente 39 kBytes do espaço de armazenamento.

3.3 Diagrama de Blocos Funcionais (FDB)

É uma linguagem gráfica baseada nos diagramas de circuitos, que representa blocos interconectados destacando o fluxo de sinais entre os elementos. É coerente com a Norma IEC 617-12. É usada para descrever o comportamento de:

- Funções;
- Blocos Funcionais;
- Programas;
- Passos, ações e transições da linguagem SFC.

Cada função também tem uma saída digital extra, denominada ENO (*Enable Output*) que é definida verdadeira (TRUE) quando a execução da função é completada com sucesso.

Assim é comum se encadear a saída ENO de uma função com a entrada EN da outra para garantir que a cadeia só produzirá um resultado correto, quando todas as etapas estiverem corretas.

3.3.1 Blocos Funcionais

A norma IEC 617-12 (*Graphical symbols for diagrams*) define um conjunto básico de blocos funcionais que são freqüentemente usados na definição de blocos mais complexos.

3.3.1.1 Blocos Funcionais Padrões

- ❖ Biestáveis SR e RS;

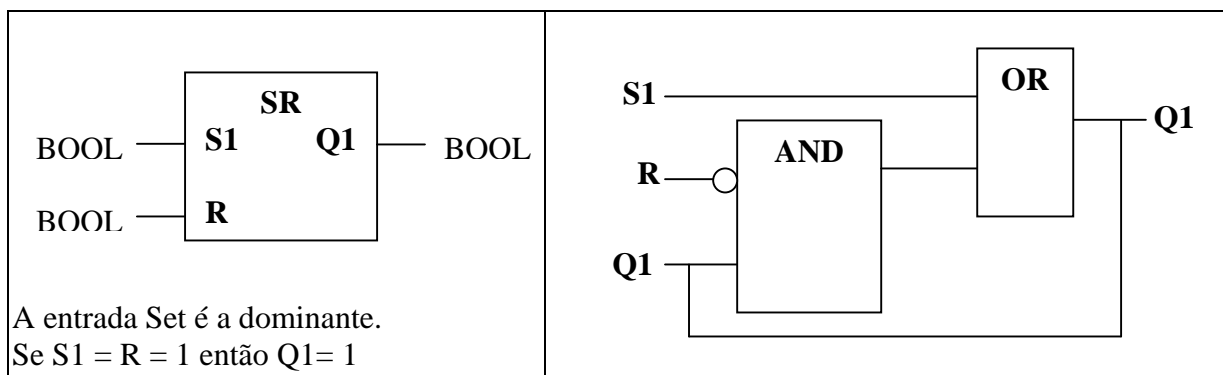


Figura 18 - Biestável SR (algoritmo interno em FBD).

- ❖ Detectores de Bordas de Subida, Descida e Atributo sensível à borda;

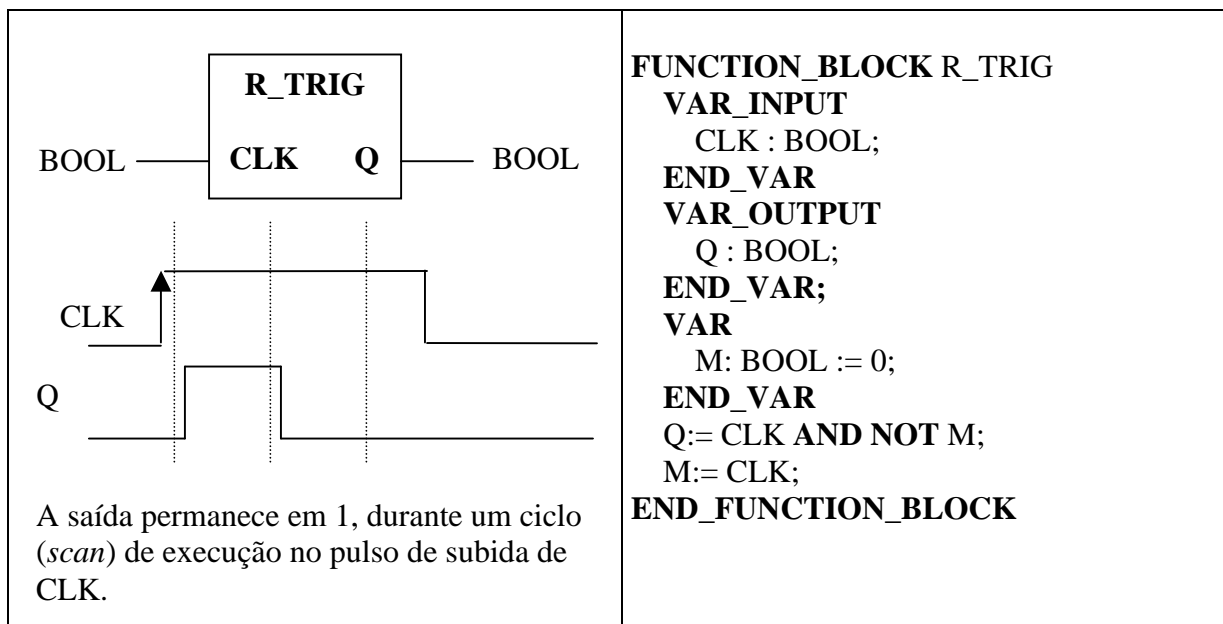


Figura 19 - Detector de Borda de Subida (algoritmo interno em ST).

- ❖ Contadores Incremental, Decremental e Incremental/Decremental;
- ❖ Temporizadores de Pulso com atraso na subida e atraso na descida;
- ❖ Relógio de Tempo Real

3.3.1.2 Blocos Funcionais Complexos (criados a partir dos padrões)

- ❖ Integral;
- ❖ Derivada;
- ❖ PID (Proporcional Integral e Derivativo);
- ❖ Rampa;
- ❖ Histerese;
- ❖ *Ratio Monitor*;

3.3.2 Portabilidade entre ST e FBD

Funções que controlam sua execução explicitamente com o uso de EN (*Enable*) e ENO (*Enable Output*), não podem ser traduzidas do FBD para o ST, pois não existe sintaxe no Texto Estruturado para endereçar a saída ENO.

Já no Diagrama de Blocos Funcionais fica difícil a tradução dos operadores condicionais IF...THEN, CASE, FOR, WHILE, REPEAT e o acesso de elementos em um vetor:

```

FOR I:=1 TO 100 DO
    rate[I]:= 100;
END_FOR;

```

3.3.3 Exemplo de tradução de uma malha de controle do *FBD* para o *ST*.

- ❖ O FBD difere na melhor representação e conseqüente entendimento mais rápido do sistema por se tratar numa linguagem gráfica.

Em *FBD*:

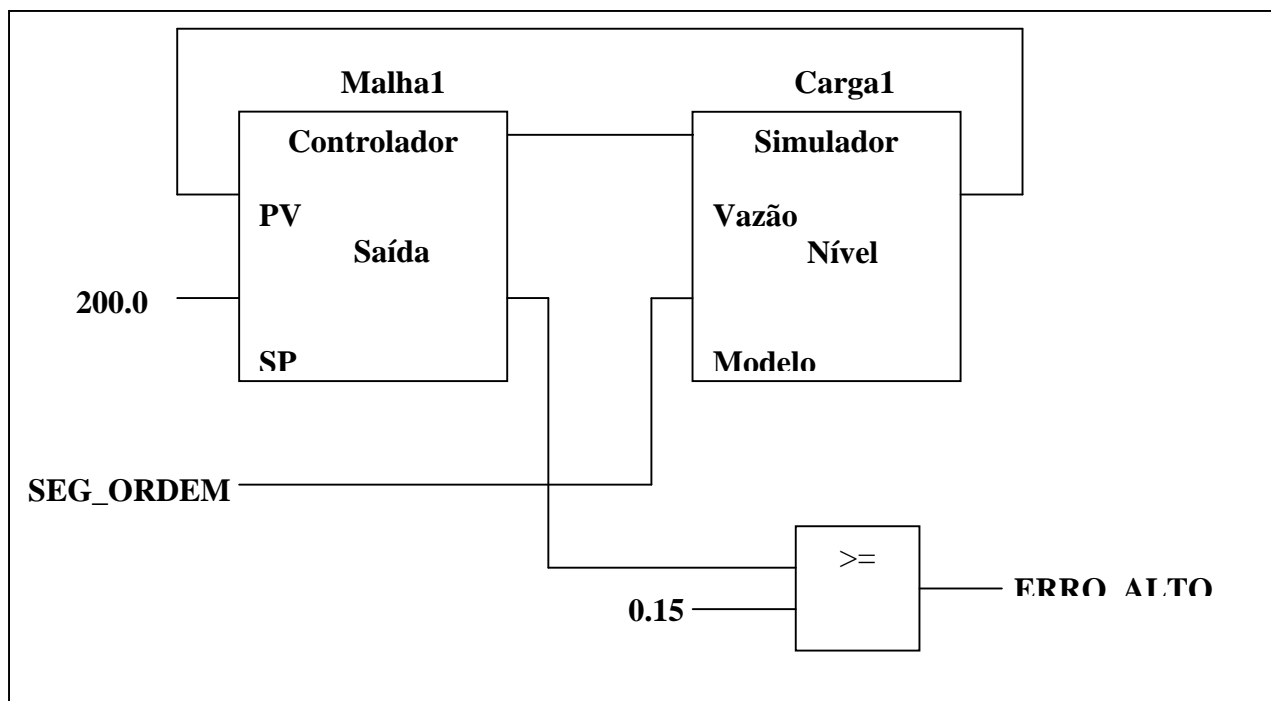


Figura 20 – Malha de Simulação de Controle de Nível de um Tanque.

Em *ST*:

VAR

Malha1: Controlador;

Carga1: Simulador;

END_VAR

(*Chamada dos blocos funcionais com conexão entre parâmetros*)

Malha1(PV:= Carga1.Nível, SP:=200.0);

Carga1(Vazão:= Malha1.Saída, Modelo:= SEG_ORDEM);

ERRO_ALTO:= Malha1.Erro>=0.15;

Para o projeto de blocos funcionais, em determinados casos pode ser necessário o desenvolvimento em linguagens não definidas pela norma, tais como o C e Pascal, quando se faz necessário o acesso a recursos internos do sistema operacional do CP.

3.4 Diagrama Ladder (LD)

É uma linguagem gráfica baseada nos diagramas elétricos, que representa contatos e bobinas interconectados destacando a energização entre os elementos. É usada para descrever o comportamento de:

- Funções
- Blocos Funcionais
- Programas
- Passos, ações e transições em SFC.

Uma linha vertical à esquerda representa um barramento energizado e à direita uma barra de terra, com o fluxo de potência sempre da esquerda para a direita. A função de controle é definida pela forma como os contatos (abertos ou fechados) são associados para comandar a bobina do relé (serial ou paralelo).

3.4.1 Exemplo de uma Partida Direta de um Motor com contato selo

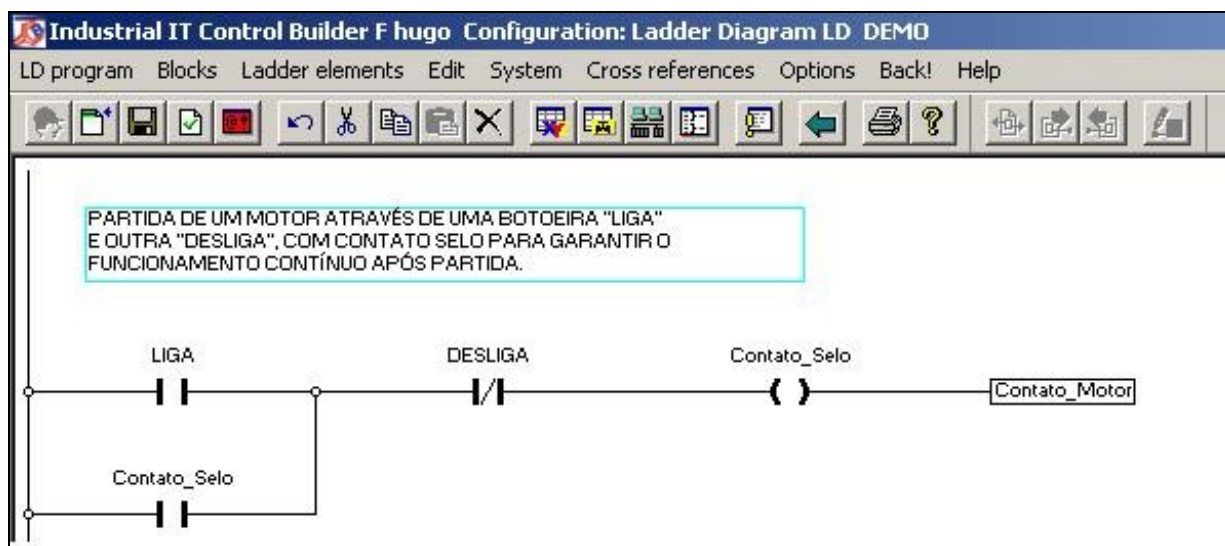


Figura 21 – Esquema de uma partida direta de motor no *Ladder*.

Cada contato está associado ao estado de uma variável lógica (booleana).

Tradução para o ST da figura 20:

Contato_Selo := (Liga **OR** Contato_Selo) **AND** Desliga;

Atualmente os Diagramas *Ladder* existentes nos mais diferentes fornecedores possuem Funções e Blocos Funcionais além dos padrões definidos pela norma para

aumentarem o ramo de aplicações da linguagem e diferenciarem seus produtos dos demais. Por exemplo, o Diagrama *Ladder* da *RSLogix 500* da *Rockwell* possui uma adaptação para a utilização da Função *MOV* no *Ladder* (Apêndice A). Será exemplificado mais adiante uma aplicação real de processo para controle de temperatura e nível utilizando o *RSLogix 500* da *Automation Rockwell*.

3.4.2 Portabilidade para o FBD:

Em LD, temos:

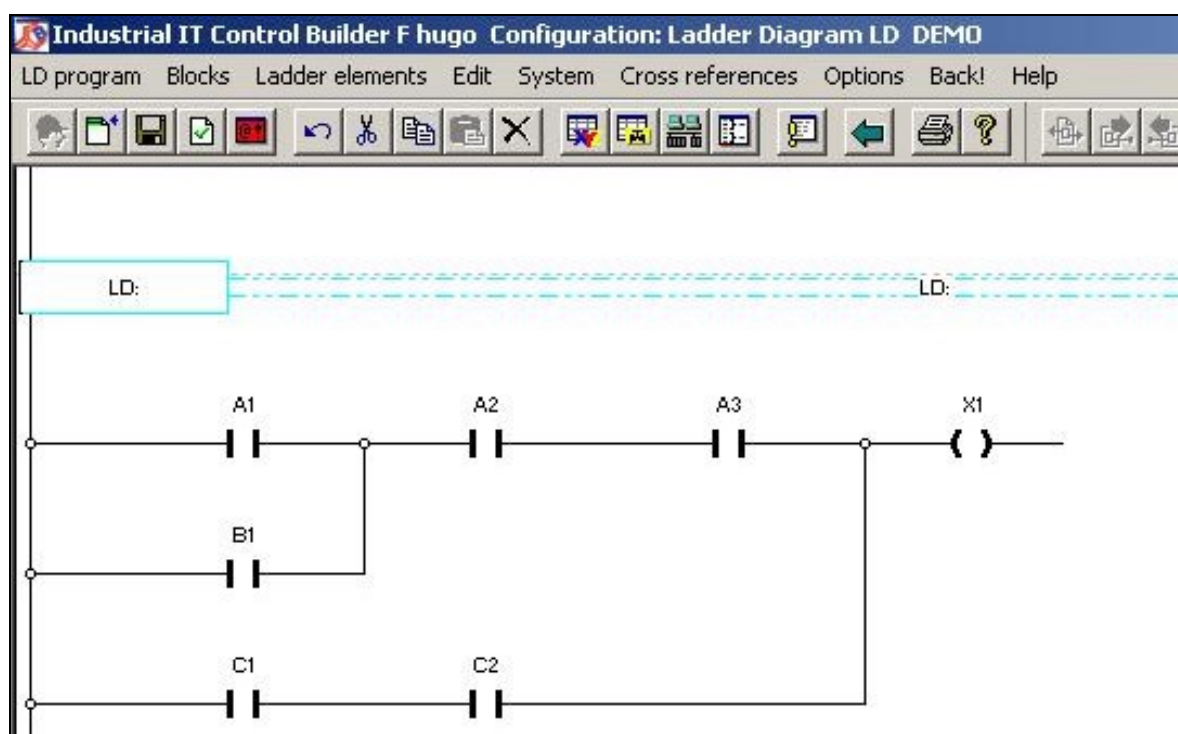


Figura 25 – Lógica booleana simples em LD.

Em FBD, temos:

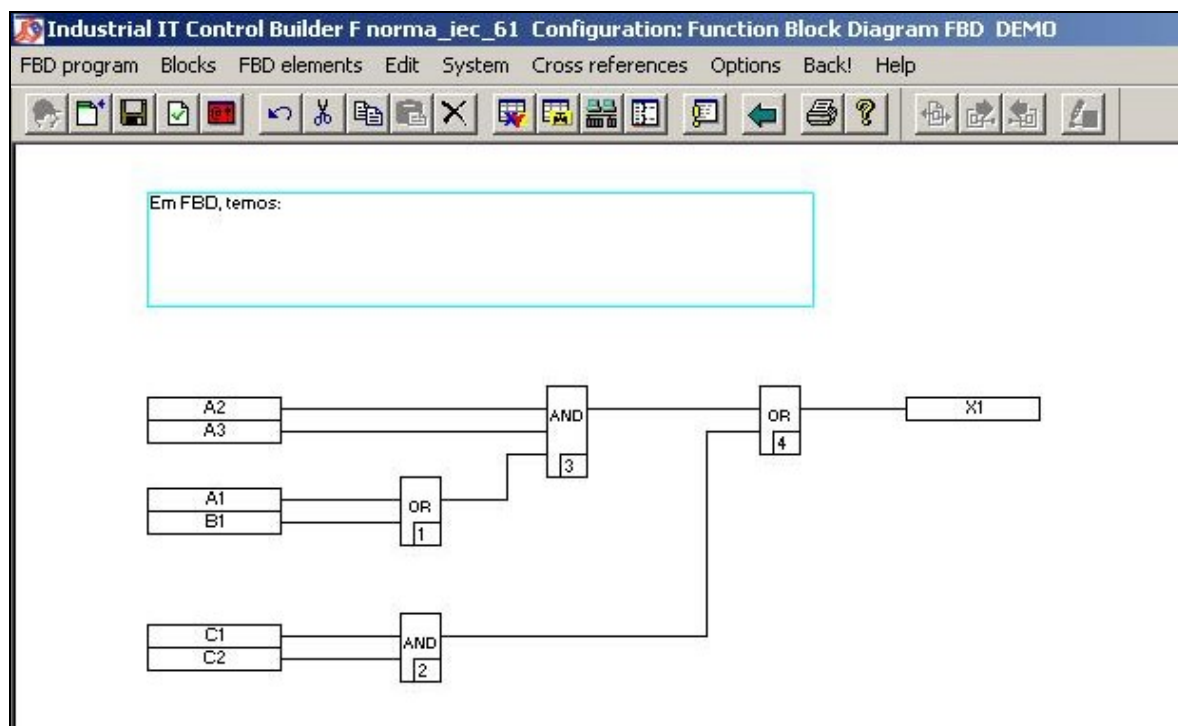


Figura 26 – Lógica booleana simples em FBD.

Também fica difícil a tradução para o *FBD*, dos operadores condicionais IF...THEN, CASE, FOR, WHILE, REPEAT e o acesso de elementos internos em um vetor conforme declarados na linguagem ST.

3.4.3 Aplicabilidade de Diagramas Ladder

O diagrama ladder é muito bom para representação de lógica booleana combinacional simples, pela sua facilidade de programação e entendimento. Entretanto, o uso de ladder se torna complicado devido ao tamanho e a complexidade do programa, quando consideramos cálculos, controle em malha fechada e sequenciamento de operações.

O diagrama ladder deve ser evitado quando o programa torna-se extenso e complexo, pois isto vai contra a estruturação de *software* proposto pela norma IEC 61131-3. Quando se tem um programa em Diagrama Ladder muito extenso, mais de 30 linhas de código, por exemplo, um programador para alterar uma linha da lógica, deve percorrer todas as linhas do início ao fim para

entender qual a linha/lugar que deve ser alterada. Já nas linguagens gráficas o mesmo não acontece, pois possuem melhor representação em blocos representativos (SFC, FBD e LD).

3.5 Lista de Instruções

A Lista de Instruções (IL) é uma linguagem textual, de baixo nível, com estrutura semelhante ao assembler, usada para descrever o comportamento de:

- Funções
- Blocos Funcionais
- Programas
- Passos, ações e transições em SFC.

A linguagem é ideal para resolver problemas simples e pequenos, onde existem poucas quebras no fluxo de execução do programa.

Pela norma, IL é uma linguagem adicional, menos amigável e flexível e que pode ser usada para produzir código otimizado para trechos de desempenho crítico em um programa.

3.5.1 Semântica

Cada instrução ocupa uma linha formada por:

Instrução = Operador + Operandos

3.5.2 Operadores

Operadores principais, de comparação e controle de fluxo:

Operador	Modificador	Operando	Comentários
LD	N	Qualquer	Carrega operando no acumulador
ST	N	Qualquer	Armazena acumulador no operando
ST		BOOL	Reset operando para TRUE
R		BOOL	Reset operando para FALSE
AND	N,(BOOL	E booleano
&	N,(BOOL	equivalente a E
OR	N,(BOOL	OU booleano
XOR	N,(BOOL	OU exclusivo
ADD	(Qualquer	Adição
SUB	(Qualquer	Subtração
MUL	(Qualquer	Multiplificação
DIV	(Qualquer	Divisão

Tabela 8 – Operadores Principais

Operador	Modificador	Operando	Comentários
GT	(Qualquer	Compara maior que
GE	(Qualquer	Compara maior ou igual
EQ	(Qualquer	Comparação igual
NE	(Qualquer	Comparação diferente
LE	(Qualquer	Comparação menor ou igual
LT	(Qualquer	Compoaração menor
JMP	C,N	Label	Salta para label
CAL	C,N	Nome	Chamada de bloco funcional
RET	C,N		Retorno de função ou bloco funcional
)			Executa o último operador adiado

Tabela 9 – Operadores IL de comparação e controle de fluxo

3.5.3 Portabilidade entre IL e outras linguagens IEC

A tradução de outras linguagens para IL é mais fácil que de IL para outras linguagens.

Programa em FBD:

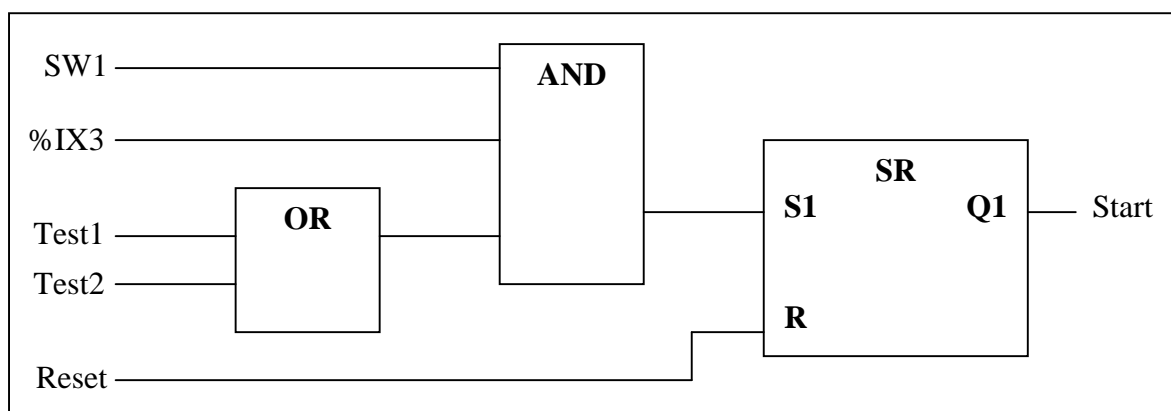


Figura 22 – Exemplo de programa em FBD.

Tradução do programa em IL:

Operador	Operando	Comentários
LD	Test1	(*Acumulador = Test1 *)
OR	Test2	(*Acc. = Test1 OR Test2 *)
AND	SW1	(*AND SW1 *)
AND	%IX3	(* AND entrada 3 *)
ST	StartSR.S1	(* Seta entrada de StartSR *)
LD	Reset	(* Carrega valor de Reset *)
ST	StartSR.R	(* Armazena na entrada Reset *)
CAL	StartSR	(* Chama bloco funcional StartSR *)
LD	StartSR.Q1	(* Carrega saída Q1 *)
ST	Start	(* e armazena em Start *)

Tabela 10 – Pilha de operação

3.5.4 Análise de Desempenho;

Quando for de conhecimento intelectual e capacidade do programador é vantajosa a tradução do código de alto nível, se possível, para o de baixo nível ou a programação direta na Lista de Instruções, pois em partes de programa onde se necessita de melhor desempenho de processamento e alocação de dados na memória, limitados por equipamentos de hardware ou não, a linguagem de baixo nível é a indicada.

Toda vez que o programa de baixo nível for executado a etapa de compilação será economizada aumentando a performance da CPU.

3.6 Sequenciamento Gráfico de Funções

A linguagem Sequenciamento Gráfico de Funções (SFC) é baseada nas técnicas correntes para descrever o comportamento seqüencial, sendo usada para:

- Descrever o comportamento seqüencial de um sistema;
- Como linguagem de estruturação de ações de um programa segundo um modelo *top-down*.
- Para descrever o comportamento de baixo nível de um processo seqüencial (ex.: Partida, Bombeando, Esvaziando, etc...)
- Para representar as fases de um processo por batelada.

Universidades francesas desenvolveram uma linguagem de representação de processos seqüenciais baseada nas Redes de Petri, o Grafcet (*Graphe Fonctionnel de Command Etape-Transition*). Grafcet se tornou um padrão europeu com a introdução do padrão IEC 848: *Preparation of function charts for control system*. Muitos fabricantes europeus de CP's oferecem *Grafcet* como uma linguagem gráfica.

A norma IEC 61131-3 introduziu algumas modificações no padrão IEC 848 visando integrar esta quinta linguagem às demais linguagens da norma. O SFC é adotado pela ISA SP 88 para programação de sistemas para controle de bateladas (*batch*).

São mostrados todos os estados de um sistema, todas as possíveis mudanças de estado e as respectivas causas. Particionando um problema de controle de forma que todos os aspectos relevantes sejam considerados e executados.

Como exemplo, na Realcafé a linguagem *SFC* é usada para descrever toda a linha de produção desde o armazenamento/beneficiamento do café verde até a embalagem e expedição do café solúvel. Cada passo é programado em *FBD* e as transições em Texto Estruturado. Logo, o Delta V não possui o Diagrama *Ladder* e a Lista de Instruções, mas no supervisorio trabalha-se com o *BASIC*.

3.6.1 Estrutura da Rede (Chart)

Uma seqüência em SFC é composta por uma série de passos (Steps) mostrados como retângulos conectados por linhas verticais, onde cada passo representa um estado

particular programado em qualquer uma das demais quatro linguagens IEC (ST, FBD, LD e IL) e até em SFC.

Cada linha de conexão possui uma barra horizontal representando uma transição, a qual é associada a uma condição que, quando verdadeira (TRUE), causa a desativação do passo anterior e a ativação do passo seguinte. Uma transição recebe um nome (T1, T2, T3, etc...) e está programada nas linguagens ST, FBD, LD, IL no *CBF*.

Cada passo pode ter uma ou mais ações associadas que é representada por um ou mais programas e podem ser descritos utilizando-se uma das quatro linguagens IEC: ST, FBD, LD ou IL.

O fluxo é de cima para baixo (*top-down*), mas ramos podem ser usados para retornar para passos anteriores.

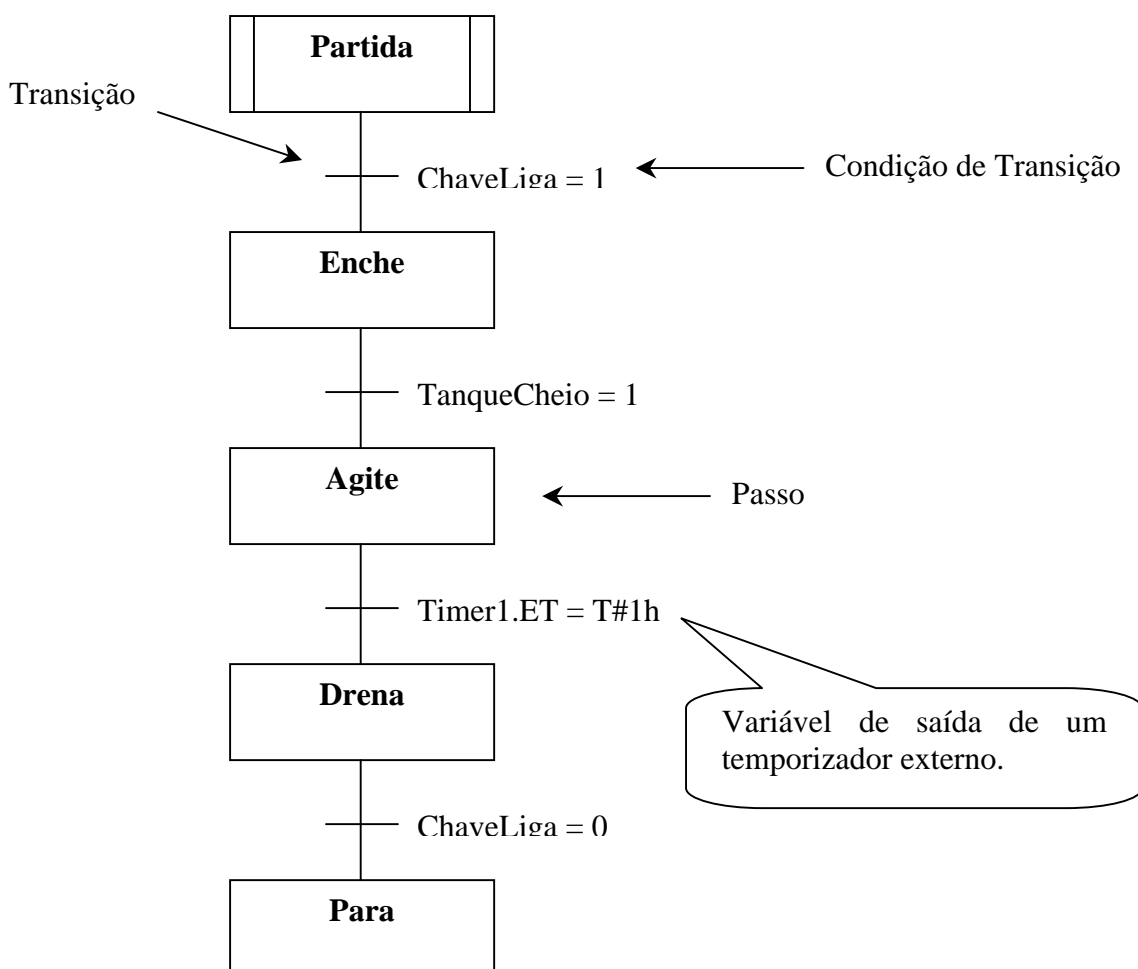


Figura 23 – Principais elementos SFC

3.6.2 Análise de Desempenho

Com o particionamento dos problemas, a linguagem SFC traz ganhos de desempenho porque apenas o código relativo aos passos ativos é executado.

4 APLICAÇÃO DA NORMA IEC 61131-3 A UM ESTUDO DE CASO

Em [3] foi construído um sistema de controle de nível do reservatório, controle de temperatura do fluido de entrada e controle de temperatura interna do reservatório todos programados na linguagem Diagrama *Ladder*, mostrado no Apêndice A e de acordo com [7], com o programa *RSLogix 500 da Rockwell*.



Figura 24 – Sistema construído

Foi utilizada uma malha de controle (figura 25) para manter o nível do reservatório de acordo com um valor pré-determinado:

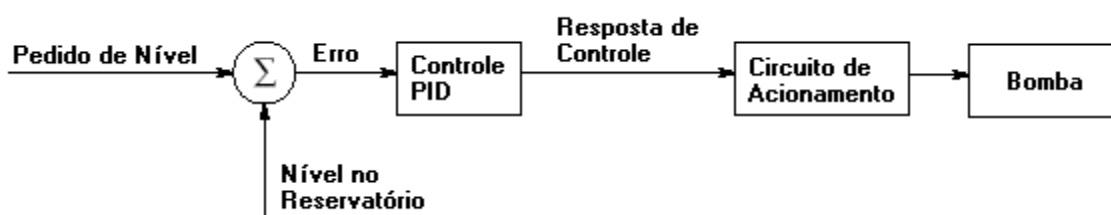


Figura 25 – Malha de controle de nível implementada.

Uma malha para controlar a temperatura do fluido que entra no reservatório (figura 26) foi desenvolvida com a utilização de um sinal *Feedforward* para “supressão, antes mesmo que ocorram, das perturbações na temperatura do fluido em

questão ocasionadas pela constante variação da vazão de entrada em resposta aos diversos pedidos de vazão solicitados pelo processo”.

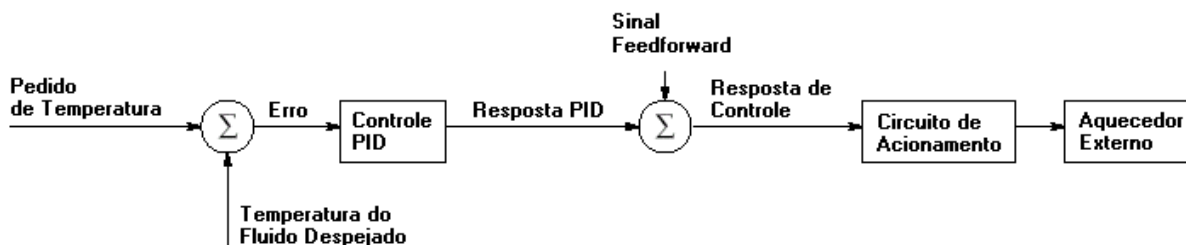


Figura 26 – Malha de controle da temperatura do fluido que entra no reservatório principal.

Por fim, uma malha de controle para manter a temperatura do fluido no reservatório (figura 27) foi implementada com um aquecedor interno.

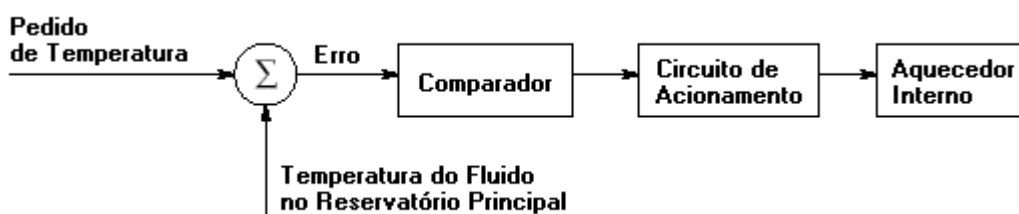


Figura 27 – Malha de controle da temperatura do fluido no reservatório principal.

4.1 O Ambiente *Control Builder F*

O ambiente de programação (configuração) do *CBF* foi projetado de forma que se tenha uma representação hierárquica dos elementos de software e áreas para programação de blocos funcionais.

Na figura 28 a Configuração foi representada no nível mais alto (CONF) de acordo com a norma. Logo abaixo se tem uma área para programação de Blocos Funcionais na linguagem preferida. No mesmo nível, o Recurso, a Estação de Processo (D-PS) tem todas as tarefas (Tarefa1 e Tarefa2) para a execução dos programas criados a partir do programador e as tarefas intrínsecas a um sistema de controle que o CBF exige (TASKLIST). Dentro de cada tarefa programada podemos ter ou uma programação em uma das quatro linguagens IEC, LD, FBD, IL e ST ou então em SFC.

Um rascunho (POOL) é disponibilizado para a inserção das transições e passos no SFC, ou qualquer programação que ache necessário, pois é só arrastar e soltar a programação onde desejar, e for conveniente, na representação hierárquica.

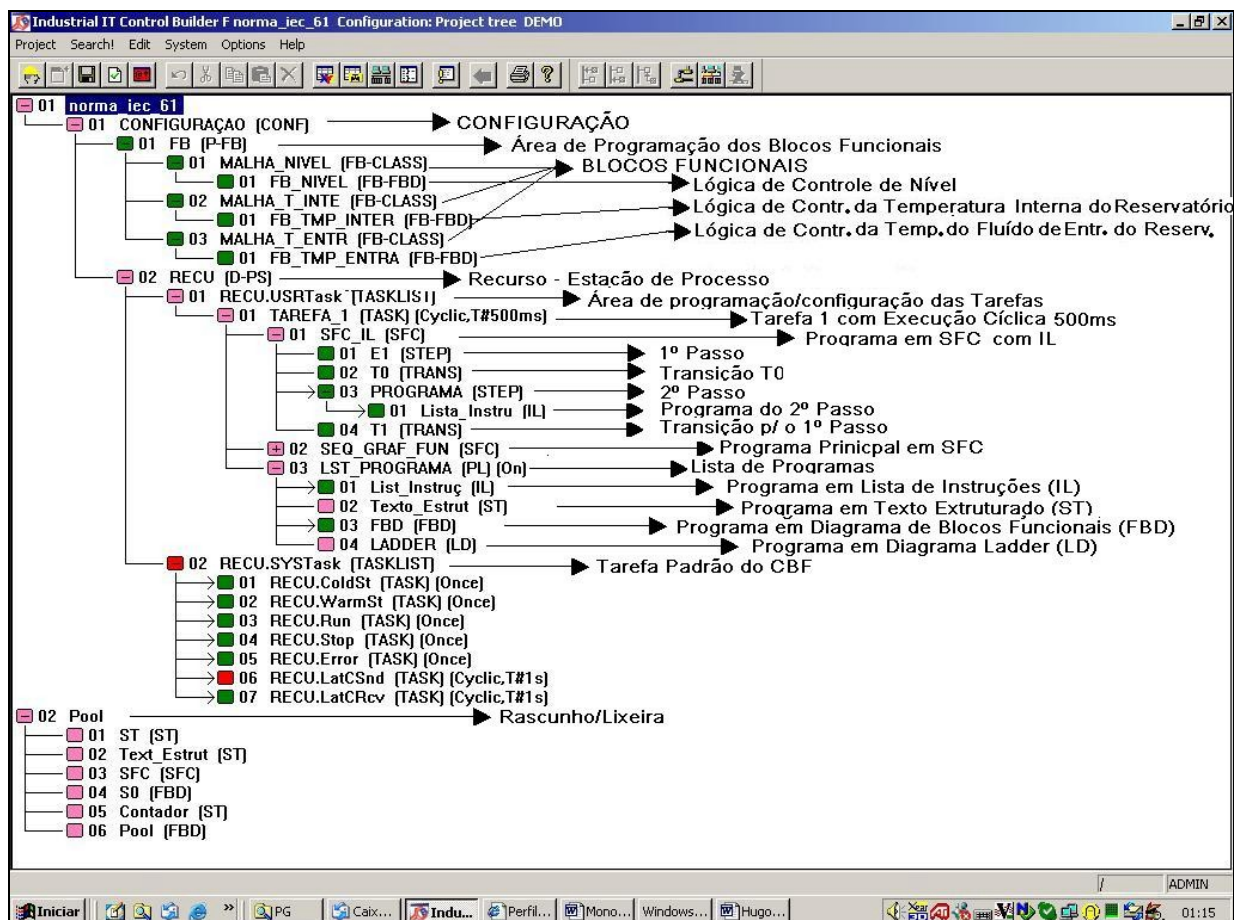


Figura 28 – Ambiente *Control Builder F* com representação hierárquica dos elementos de *software* IEC.

4.2 Blocos Funcionais

Foram utilizadas as três malhas de controle nas cinco linguagens de programação no *CBF*.

No ambiente *CBF* a **reutilização de linhas de código** foi comprovada quando programadas as malhas de controle na Área de Programação dos Blocos Funcionais (figura 25, 26 e 27). Assim, foi possível reutilizar os Blocos Funcionais (*FB-CLASS*) programados pelo usuário na Lista de Programas (*PL*), ou no Rascunho (*POOL*), economizando tempo de re-programação minimizando ou eliminando erros, reduzindo o custo do programa final.

Os Blocos Funcionais, MALHA_T_INTE, MALHA_T_ENTR e MALHA_NIVEL (figura 34), foram programados na linguagem Diagrama de Blocos Funcionais, a qual é indicada para representar malhas de controle, pela melhor representatividade e interpretação da programação.

O detalhamento com comentários (*comentários entre asterisco e parêntesis*) em linhas de código, faz com que o novo programador ou o mantenedor não perca a linha de pensamento, não precisando sempre retornar ao início do programa para o entendimento das linhas de código, o que acontece com o diagrama *Ladder* em proporções muito grandes, no qual foi requerido quando se fez necessário a portabilidade do *RSLogix 500* para o *CBF*.

4.2.1 Bloco Funcional MALHA_T_INTE

A lógica de controle de temperatura interna do reservatório foi programada de acordo com as linhas 2, 11, 12, 13, 14 e 16 do Apêndice A.

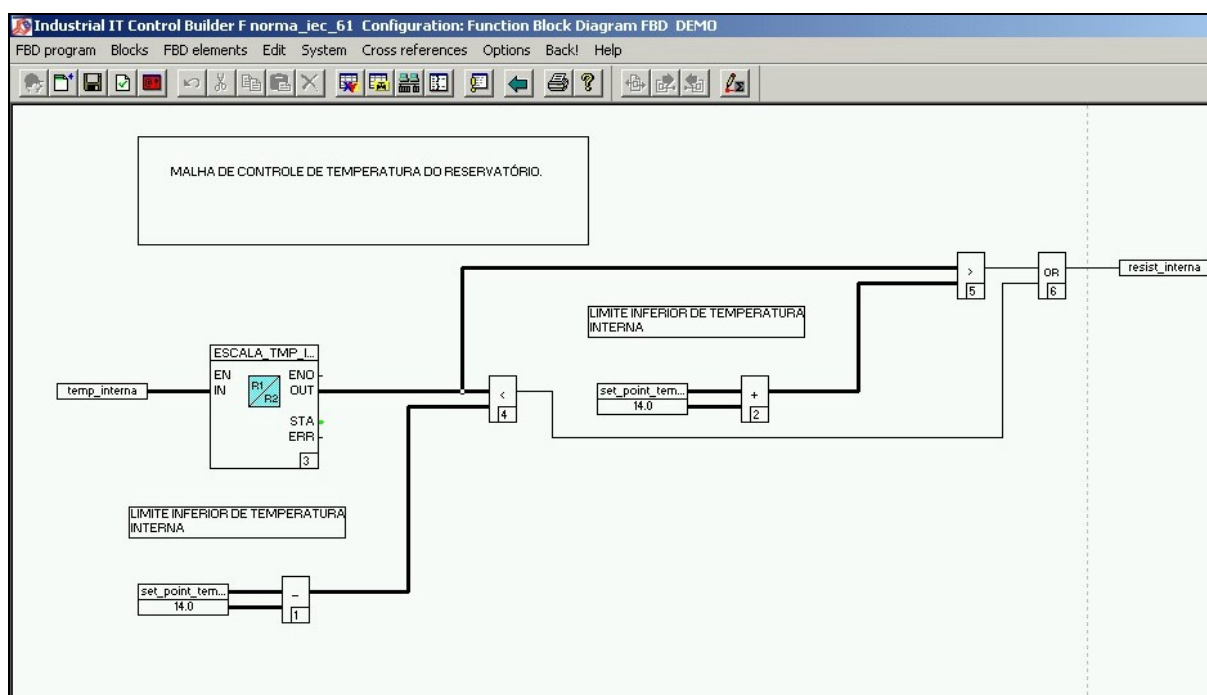


Figura 29 – Bloco Funcional de Controle da Temperatura Interna do Reservatório.

4.2.2 Bloco Funcional MALHA_T_ENTR

A lógica de controle de temperatura do fluido de entrada do reservatório foi programada de acordo com as linhas 1, 5, 6, 7, 8, 9 e 10 do Apêndice A.

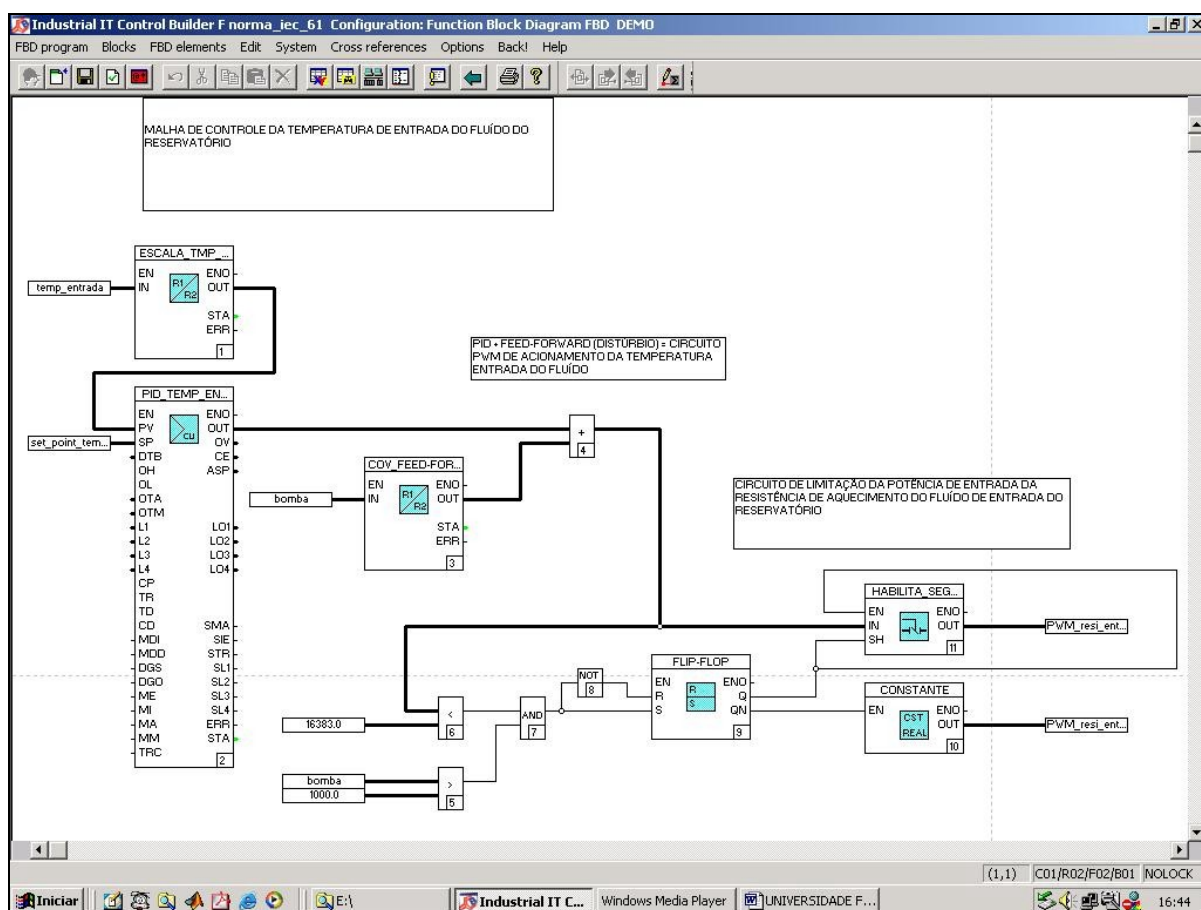


Figura 30 – Bloco Funcional de Controle da Temperatura do Fluido de Entrada do Reservatório.

4.2.3 Bloco Funcional MALHA_NIVEL

A lógica de controle de nível do reservatório foi programada de acordo com as linhas 0, 3, 4 e 15 do Apêndice A.

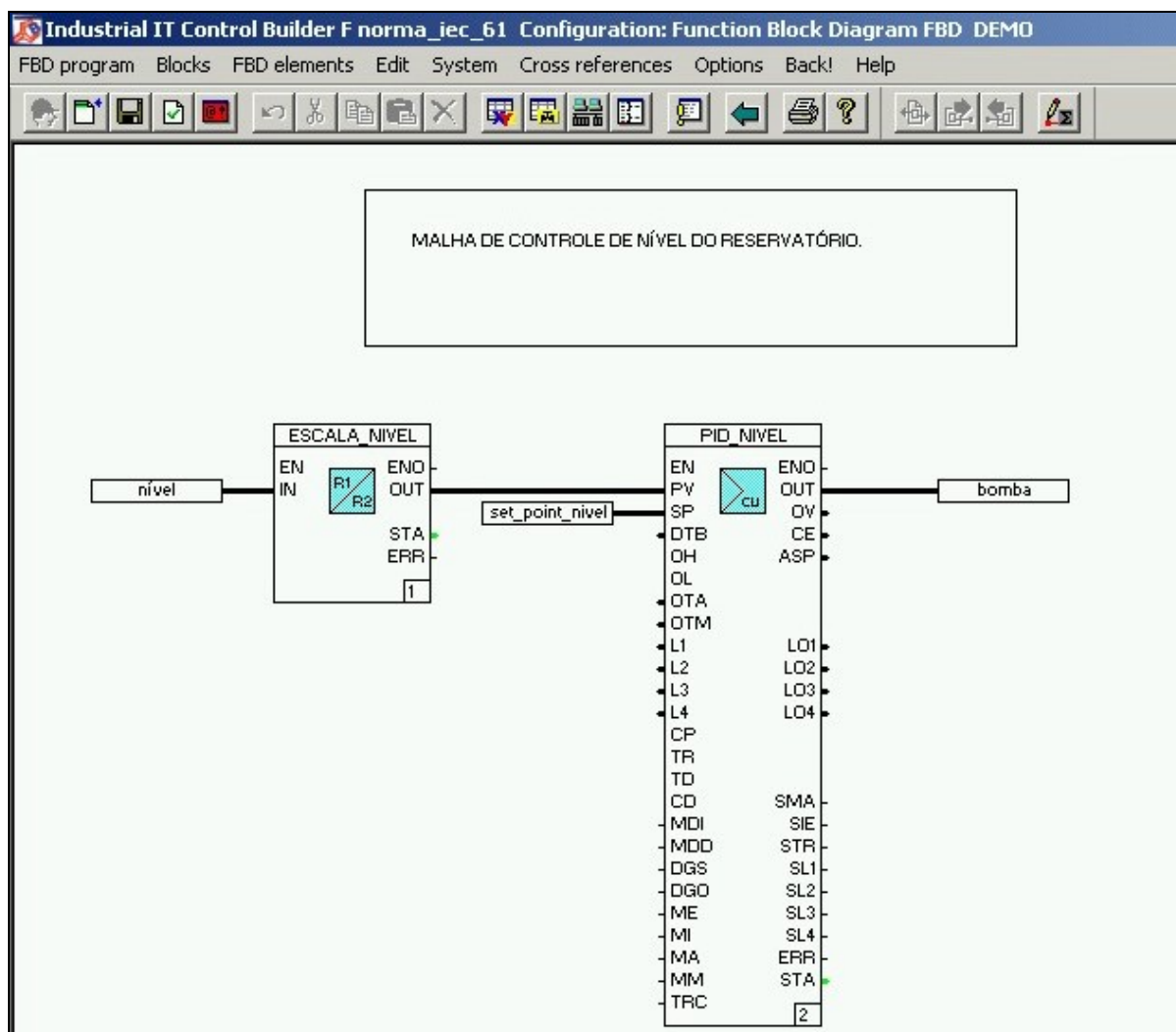


Figura 31 – Bloco Funcional de Controle do Nível do Fluido do Reservatório.

4.2.4 Programa principal em Sequenciamento Gráfico de Funções

Para efeito demonstrativo, utilizei a linguagem seqüencial representando as três malhas de controle serão executadas em paralelo, ou seja, ao mesmo tempo em três passos diferentes conforme a figura 32.

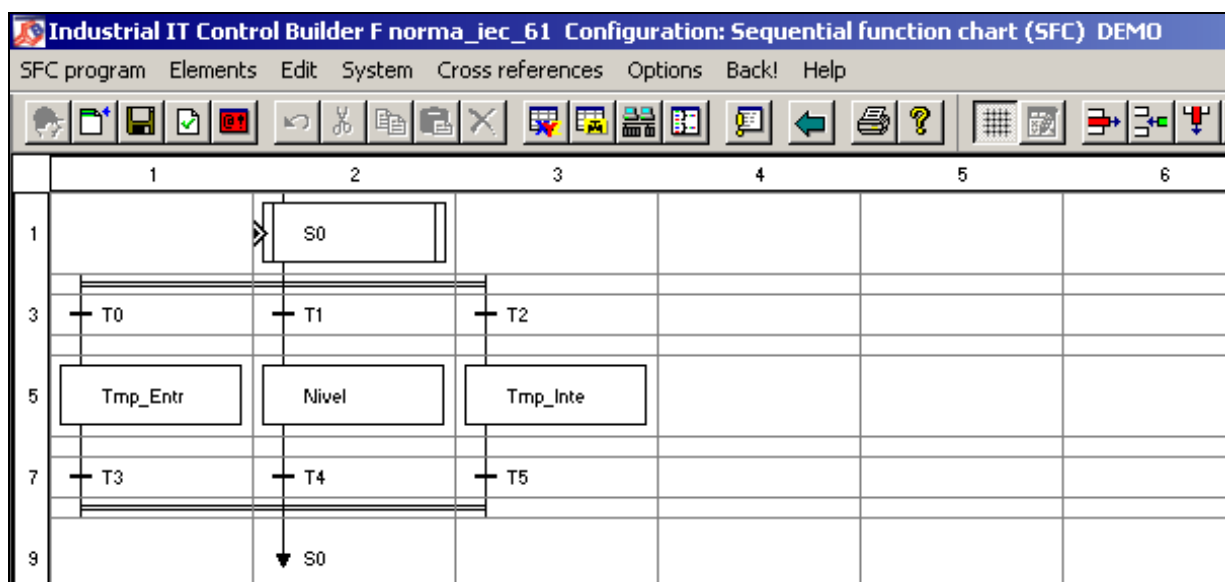


Figura 32 – Programa da Lógica de Controle em SFC.

Para se inserir uma *POU* no *SFC*, deve-se criá-la primeiramente no Rascunho (*POOL*), compilá-la (*Check*), para depois inseri-la no programa principal.

Uma vez o programa em **S0**, cada transição criada (**T0**, **T1** e **T2**) só irá estar satisfeita se: em **T0** a Temperatura do Fluido de entrada estiver abaixo de 5374.0 (*set point* - temp_entrada), em **T1** se o Nível estiver acima de 11122.0 (*set point* - nivel), e em **T2** se a Temperatura Interna do Fluido estiver acima de 16383.0 (*set point* – temp_interna), para obedecer a condição de transição entre **S0** e os demais passos, conforme as figuras 33, 34 e 35, representando a transição **T0**.

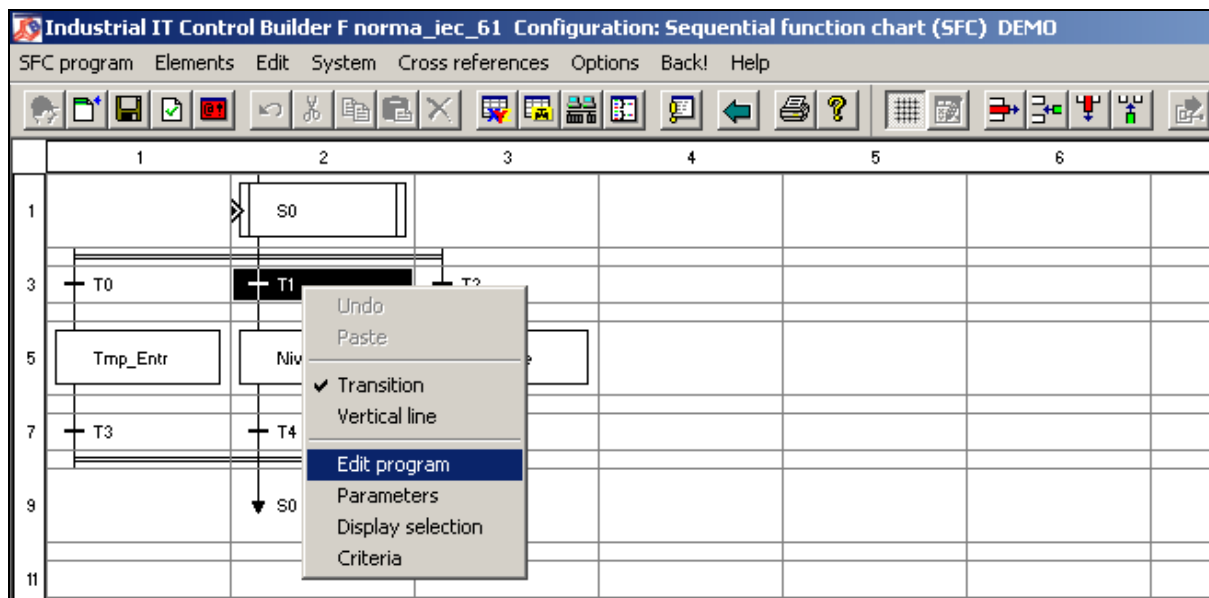


Figura 33 – Editando o programa da transição T1.

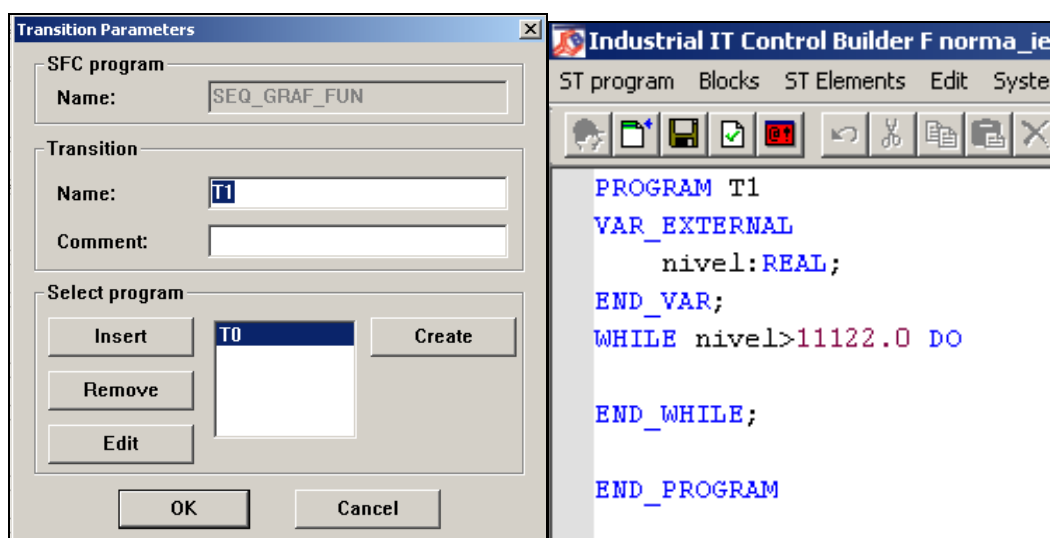


Figura 34 e 35 – Editando a transição em Texto Estruturado

Após obedecidas as transições (T0, T1 e T2), todos os três passos respectivos (MALHA_T_ENTR, MALHA_NIVEL e MALHA_INTE) serão iniciados desde cada condição respectiva seja ativada. Cada passo foi implementado a partir de um programa criado no Rascunho (*POOL*) para que seja inserido no mesmo, conforme as figuras 36, 37, 38 e 39.

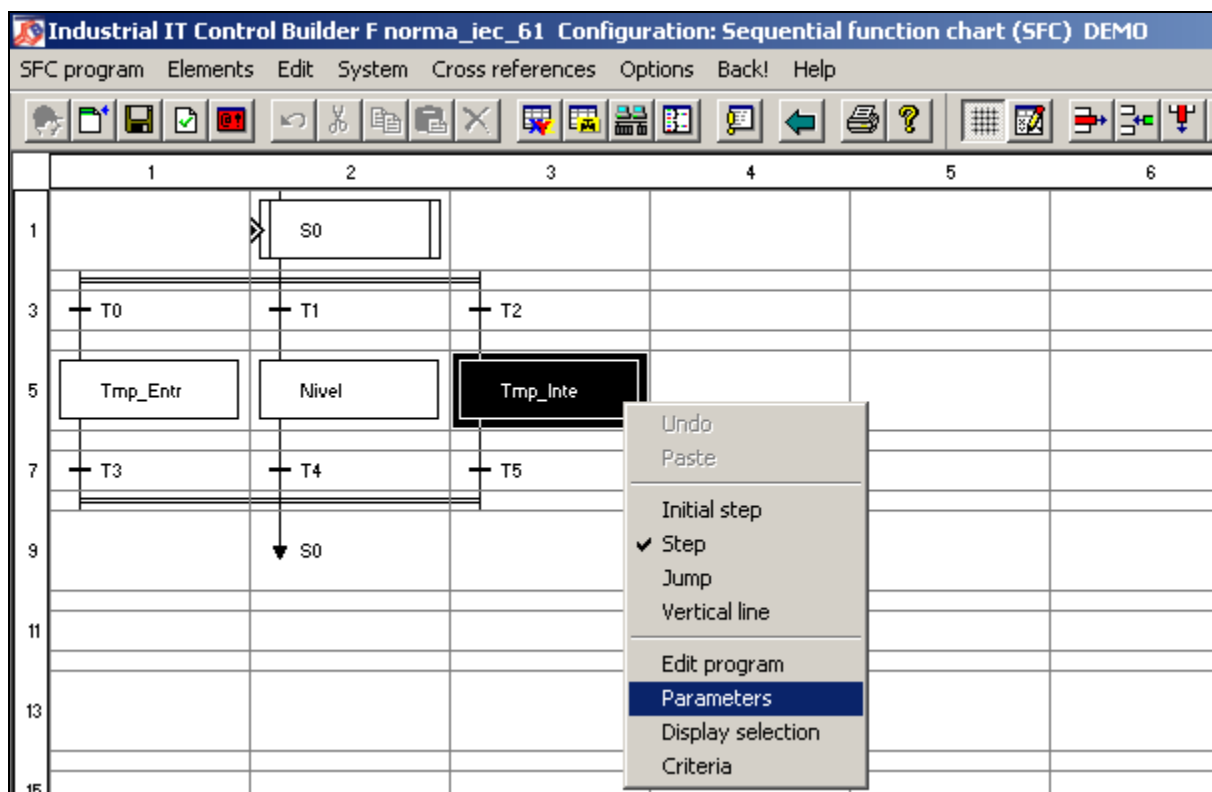


Figura 36 – Parâmetros do passo Tmp_Inte.

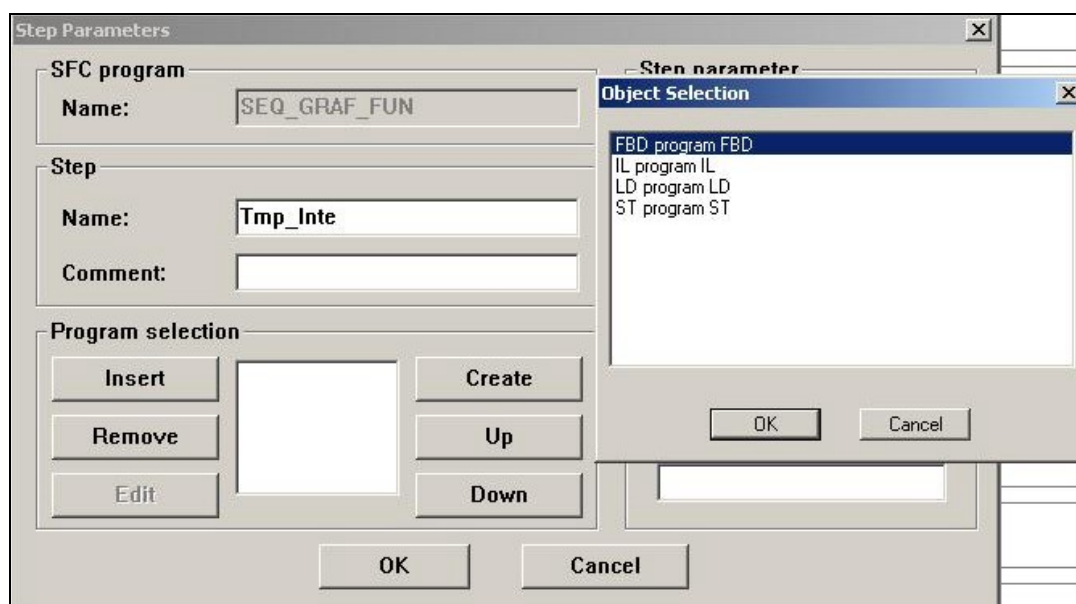


Figura 37 – Criação do programa em FBD para implementação da malha de controle de temperatura do reservatório.

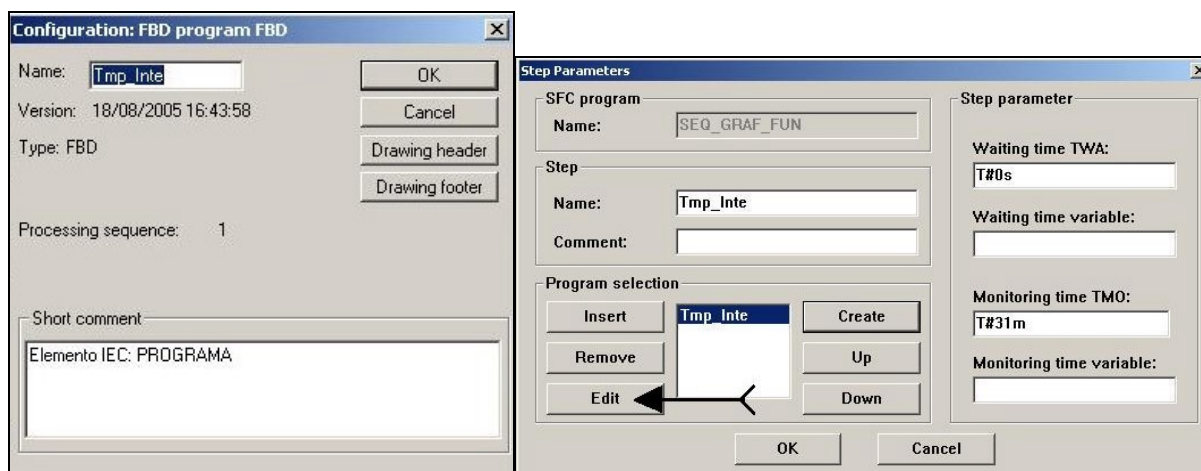


Figura 38 e 39 – Atribuição do Nome do Programa (Tmp_Inte) e edição do programa.

Para a edição do programa em *FBD*, foi inserido o Bloco Funcional MALHA_T_INTE.

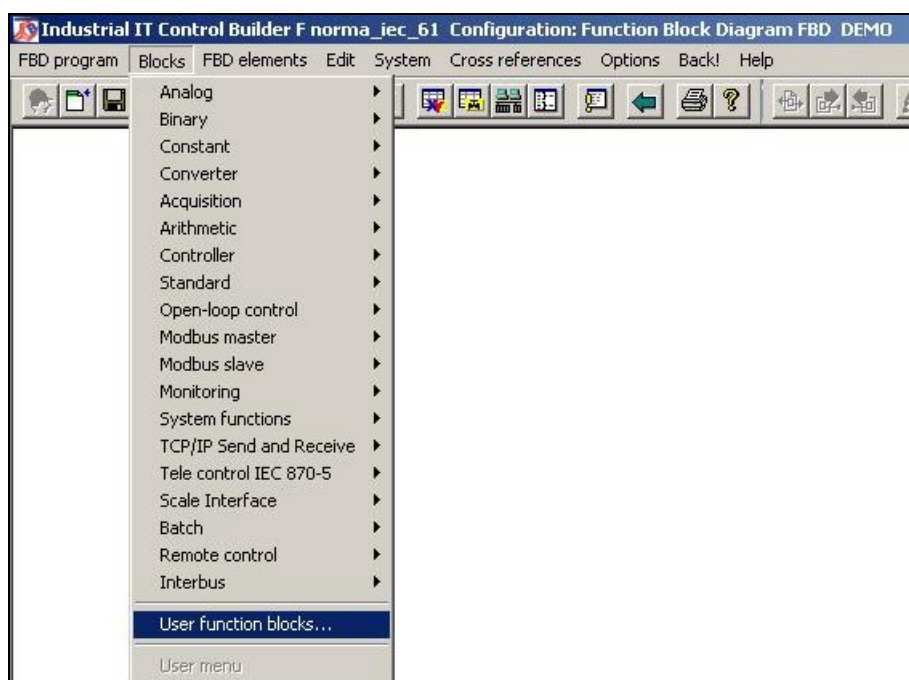


Figura 40 – Inserindo um Bloco Funcional.

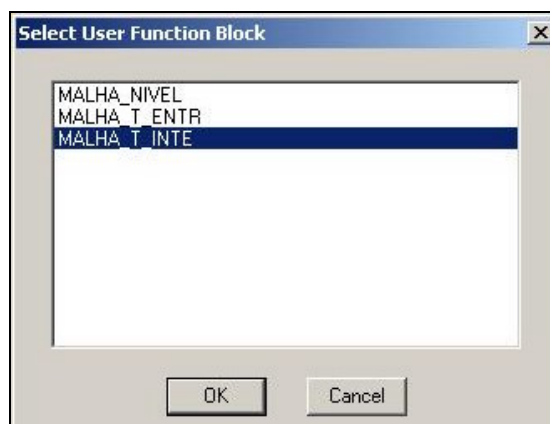


Figura 41 – Escolha do BF.

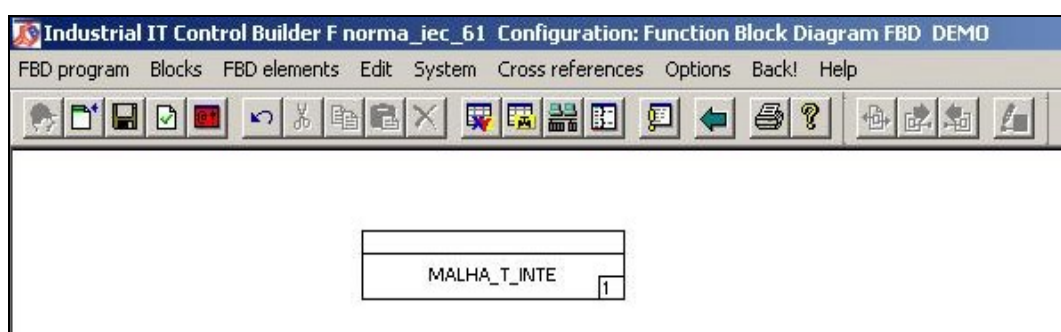


Figura 42 – Bloco Funcional de Controle da Temperatura Interna do Reservatório.

Se adentrarmos ao BF MALHA_T_INTE (figura 48), veremos toda a malha implementada em *FBD*, conforme a figura 35.

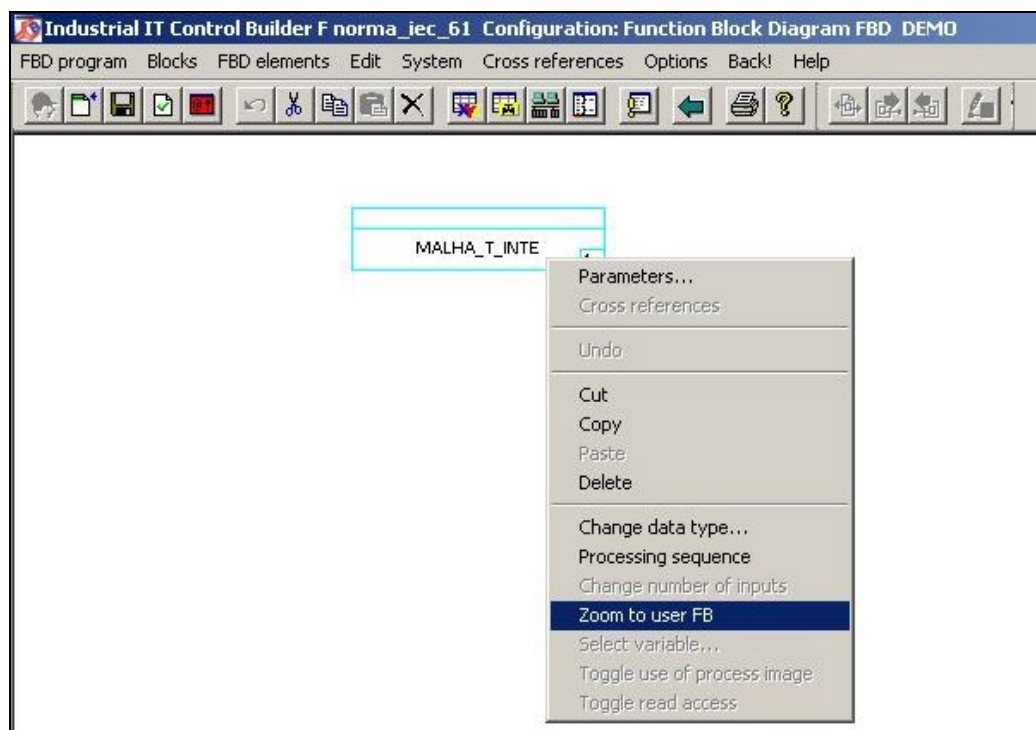


Figura 43 – Adentrando à lógica do BF.

Foram implementadas condições de transição em T3, T4 e T5, onde os passos somente serão interrompidos se o sensor de temperatura do fluido de entrada for maior que 5734,0 (temp_entrada), o sensor de nível for maior que 11122.0 (nível) e o sensor de temperatura interna (temp_interna) for maior 16383.0. Assim, o SFC irá terminar a execução somente se as três condições forme satisfeitas e retornará para S0, iniciando um novo ciclo.

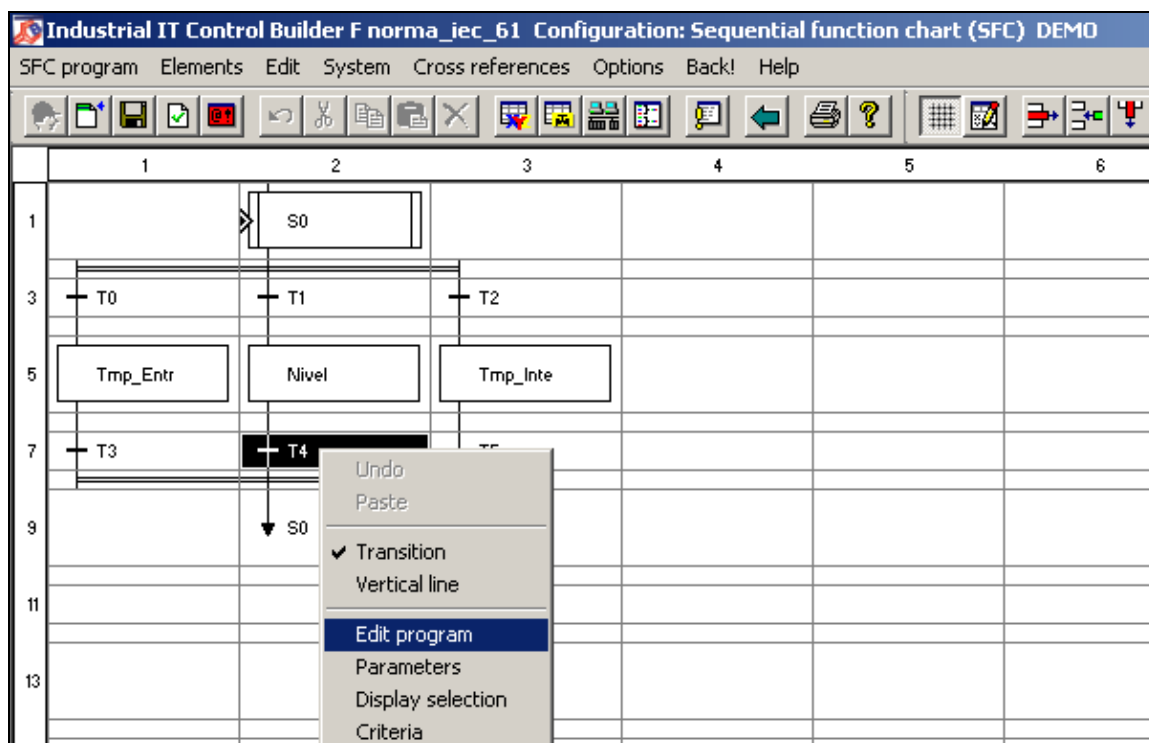


Figura 44 – Editando o programa em Texto Estruturado da condição de transição em T1.

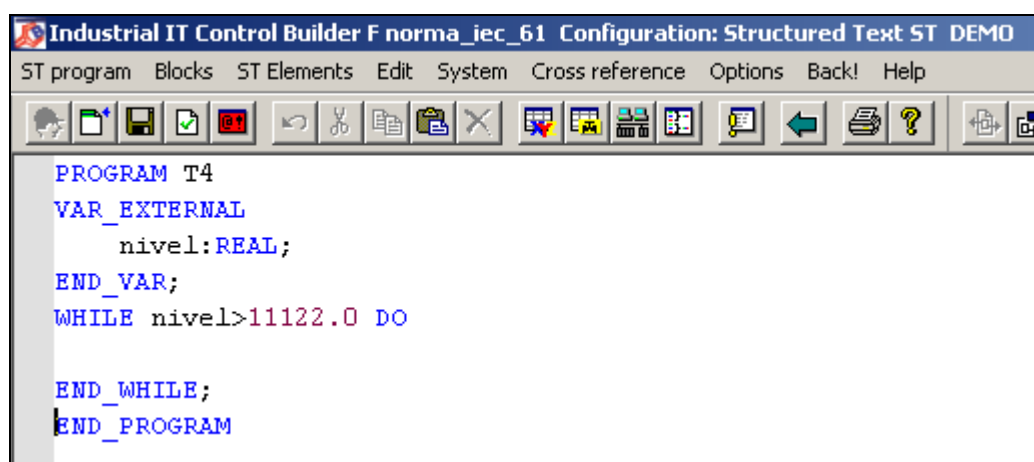


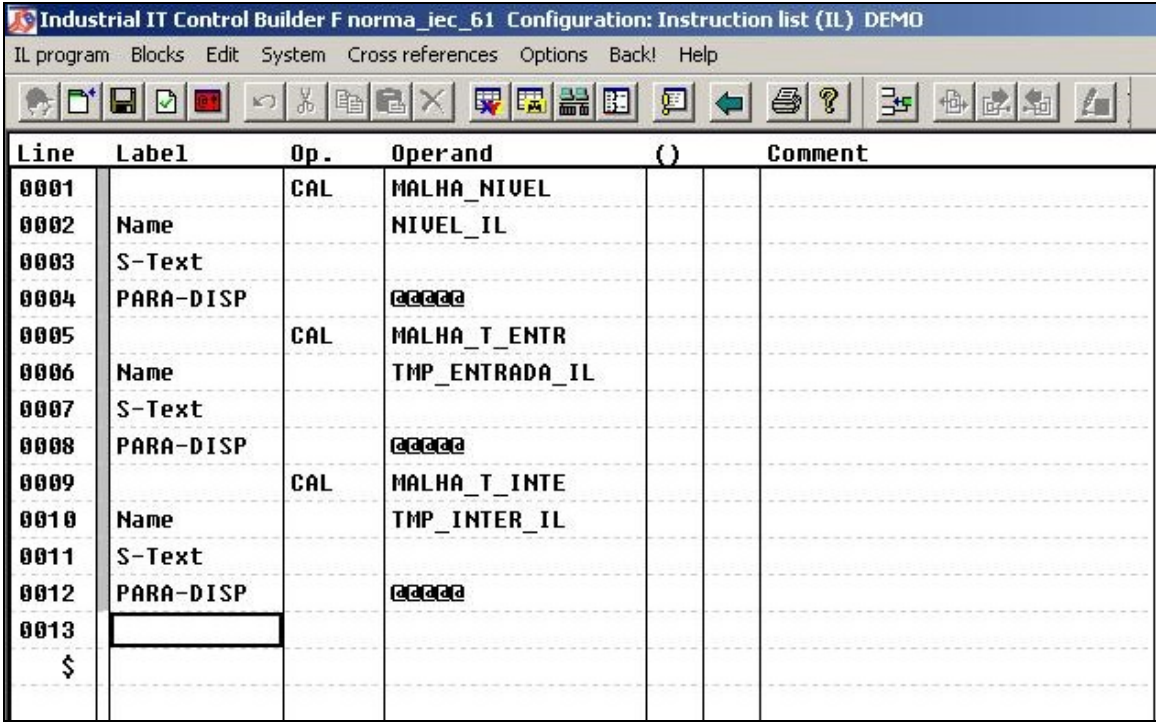
Figura 45 – Programa em ST da condição de transição T4.

4.3 Instanciação de Blocos Funcionais

Quando se cria um Bloco Funcional, pode-se inserir uma cópia do mesmo em outro Bloco Funcional ou Programa criando-se um elemento de software com a mesma função do original (que está na biblioteca do *CBF*). Assim uma nova variável será declarada do tipo do Bloco Funcional em questão.

4.4 Chamada dos Blocos Funcionais na Lista de Instruções

Cada passo do *SFC* pode ser escrito também na Lista de Instruções. Clicando em **Blocks > User function blocks...** Assim é criada uma instância de cada BF, para inserção no programa.



Line	Label	Op.	Operand	()	Comment
0001		CAL	MALHA_NIVEL		
0002	Name		NIVEL_IL		
0003	S-Text				
0004	PARA-DISP		00000		
0005		CAL	MALHA_T_ENTR		
0006	Name		TMP_ENTRADA_IL		
0007	S-Text				
0008	PARA-DISP		00000		
0009		CAL	MALHA_T_INTE		
0010	Name		TMP_INTER_IL		
0011	S-Text				
0012	PARA-DISP		00000		
0013					
\$					

Figura 46 – Chamada de Blocos Funcionais na Lista de Instruções.

4.5 Portabilidade entre Recursos de programas;

O programa *CBF* possui a opção de exportar e importar lógicas implementadas e Blocos Funcionais.

O programa *Freelance 2000 Digtool* é uma versão mais antiga do *CBF* e não contém a linguagem *ST* para programação. Um programa desenvolvido com extensão *.PRO no *Freelance 2000 Digtool*, não é portátil para o *CBF* com a mesma extensão, mas é exportável / importável de um programa para o outro.

5 CONCLUSÃO

A norma foi de grande importância para a padronização das linguagens de programação que existiam no mercado e portabilidade entre elas, da modularização ou encapsulamento do elemento de *software* que se quer reutilizar e pela estruturação do modo como o Modelo de *Software* IEC organiza seus elementos.

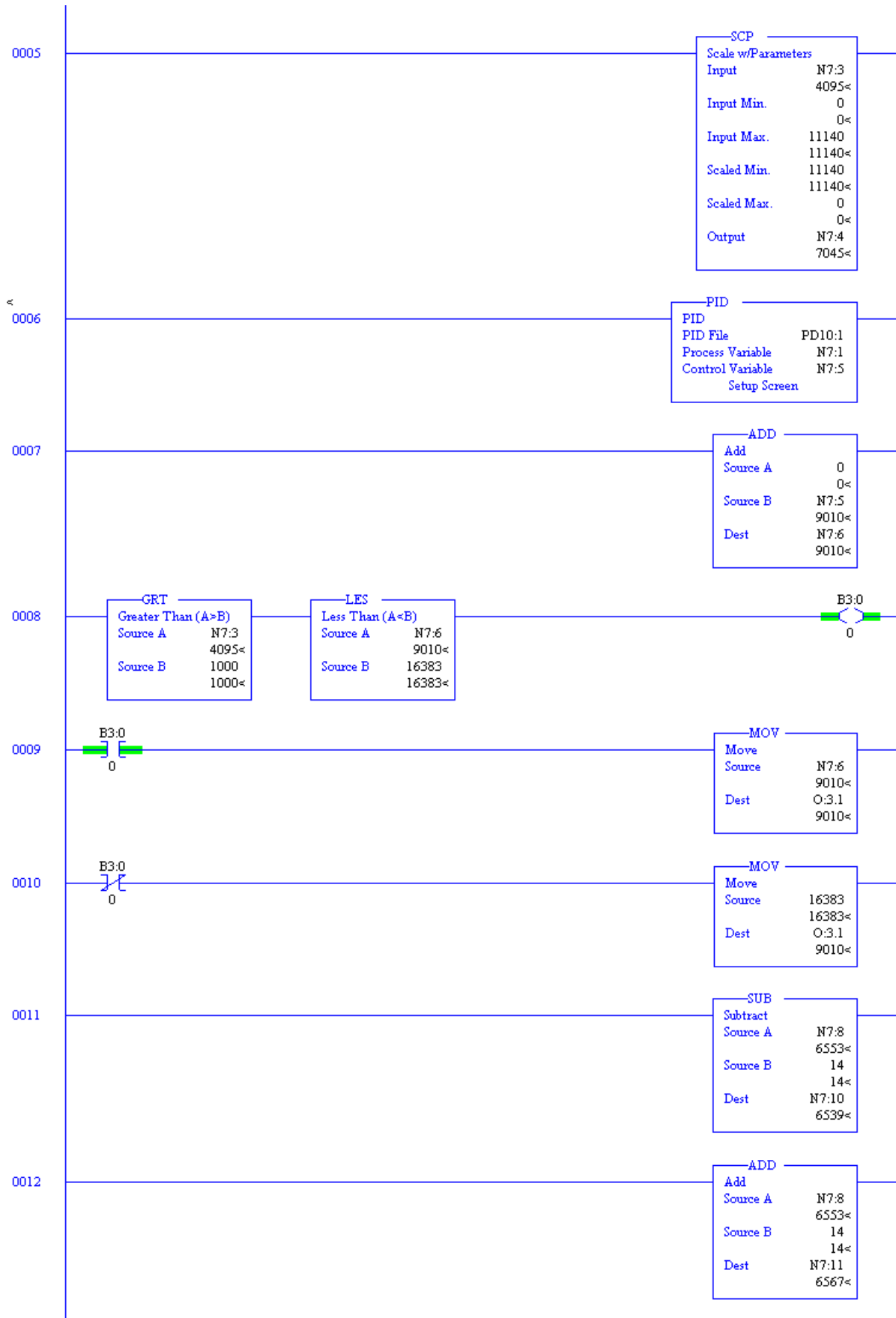
Ideal seria se todos os fornecedores industriais padronizassem seus programas para uma solução de *hardware* única, pois quando se quisesse migrar de uma plataforma para outra, se poderia portar a aplicação. Já que a necessidade de se portar aplicações de plataformas antigas para sistemas atuais é eminente.

Um produto é diferenciado do seu concorrente, de acordo com o grau de adequação à norma, de acordo com o item 2.2, e a quantidade de recursos a mais que é disponibilizado. Assim o mercado absorverá o produto com melhor valor agregado.

O programa *Control Builder F* da ABB é o produto que mais se aproxima dos conceitos da norma IEC 61131-3, de acordo com [4] e representa hierarquicamente o Modelo de *Software* IEC (figura 9) representado na figura 28.

APÊNDICE A

0000	<div>SCP</div> <div>Scale w/Parameters</div> <div>Input1:2.0167<</div> <div>Input Min.00<</div> <div>Input Max.300300<</div> <div>Scaled Min.00<</div> <div>Scaled Max.1638316383<</div> <div>OutputN7:09120<</div>
0001	<div>SCP</div> <div>Scale w/Parameters</div> <div>Input1:2.13070<</div> <div>Input Min.27302730<</div> <div>Input Max.37303730<</div> <div>Scaled Min.00<</div> <div>Scaled Max.1638316383<</div> <div>OutputN7:15570<</div>
0002	<div>SCP</div> <div>Scale w/Parameters</div> <div>Input1:2.23079<</div> <div>Input Min.27302730<</div> <div>Input Max.37303730<</div> <div>Scaled Min.00<</div> <div>Scaled Max.1638316383<</div> <div>OutputN7:25718<</div>
0003	<div>PID</div> <div>PID</div> <div>PID FilePD10:0</div> <div>Process VariableN7:0</div> <div>Control VariableN7:3</div> <div>Setup Screen</div>
0004	<div>MOV</div> <div>Move</div> <div>SourceN7:34095<</div> <div>DestO:3:04095<</div>



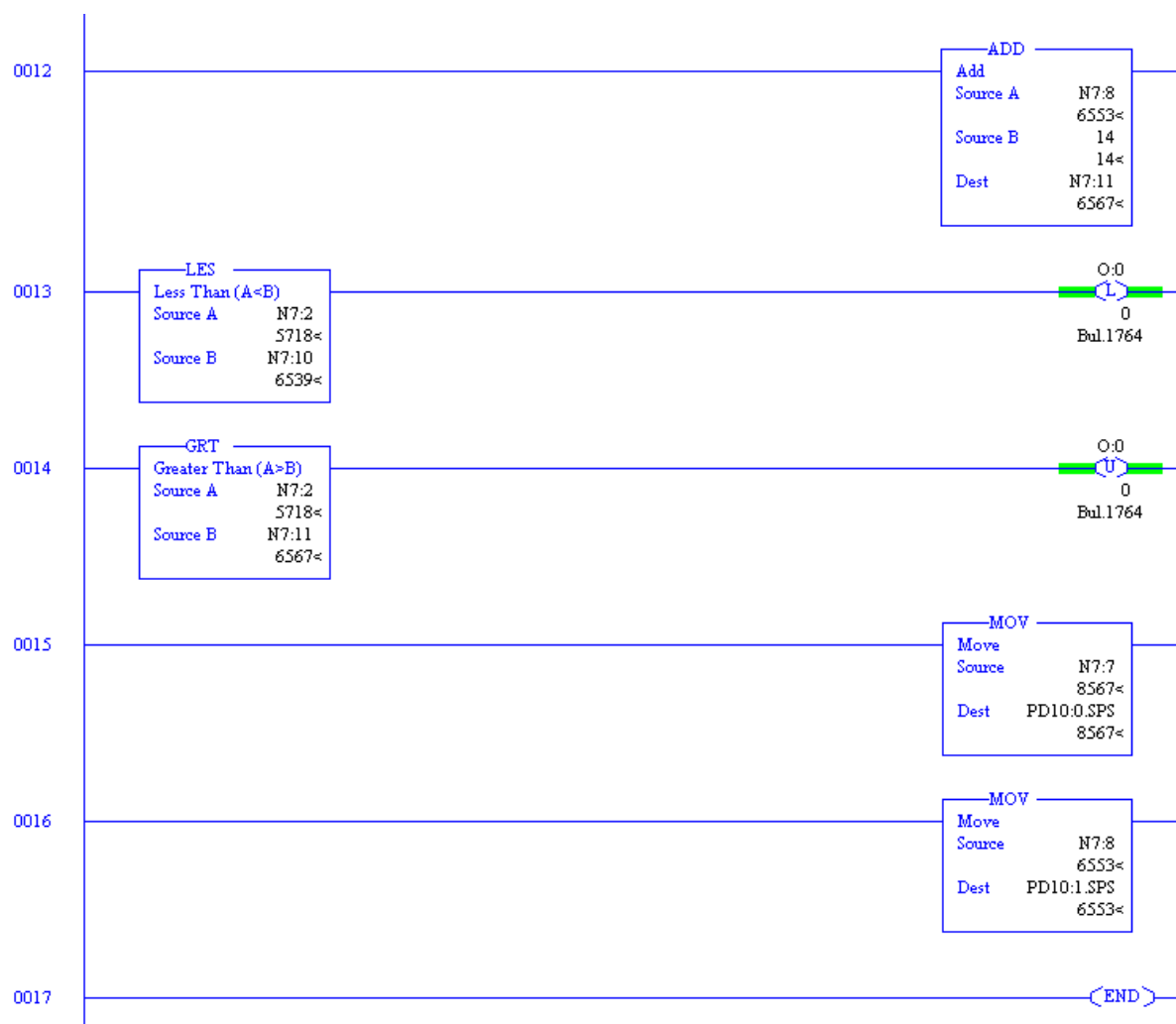


Figura 34 – Diagrama Ladder implementado em [4] no RSLogix 500 Industrial Programming Software for Allen-Bradley SLC 500 Family of Small Programmable Controllers.

**ROCKWELL
SOFTWARE**



APENDICE B

Funções e Blocos Funcionais

As designações dos blocos são idênticas no Diagrama de Blocos Funcionais, Diagrama *Ladder* e na Lista de Instruções. Temos abaixo a relação e designação das Funções e Blocos Funcionais descritos no Control Builder F.

Bloco	Função/Bloco Funcional	Designação	Tipo*
Análogica	Counter with analog input	CT_ANA	BF
	Linearization	LIN	BF
	Set point controller	C_ANA	BF
	Analog input transformation	AI_TR	BF
	Analog input transformation transient	AI_TRT	BF
	Analog output transformation	AO_TR	BF
	Scale change	SCAL	BF
	Delay function	DELAY	BF
	Lead/lag	LD_LG	BF
	Dead time function	TDEAD	BF
	Average value in time	TAVER	BF
	Maximum value in time	TMAX	BF
	Minimum value in time	TMIN	BF
	Analog filter in time	TFLILT	BF
	Time scheduler	TS	BF
Binária	Binary output	M_BOUT	BF
	Monoflop	MONO_F	BF
	Timer, switch-on/switch-off delay	TONOF	BF
	Timer, switch-on delay	TON	BF
	Timer, switch-off delay	TOF	BF
	Timer with extern time	TIMER	BF
	Time counter	CTT	BF
	Pulse counter	CT_P	BF
	Up/down counter	CTUD	BF
	Operating time counter	CT_LT	BF
	Frequency/analog converter	FAC_D	BF
	Touch button	TOUCH	BF
Controlador	Continuous controller, standard	C_CS	BF
	Continuous controller, universal	C_CU	BF
	Continuous controller, ratio	C_CR	BF
	Step controller, standard	C_SS	BF
	Step controller, universal	C_SU	BF

	Step controller, ratio Two-position controller, standard Two-position controller, universal Three-position controller, standard Three-position controller, universal Controller self-tuning	C_SR C_OS C_OU C_PS C_PU TUNE	BF BF BF BF BF
Aquisição	Disturbance course log acquisition Trend acquisition	DISLOG TREND	BF BF
Monitorando	Analog monitoring Binary monitoring Binary monitoring of antivalence General monitoring Connection monitoring Event message Horn	M_ANA M_BIN M_BAV M_GEN M_CONN EVENT HORN	BF BF BF BF BF BF BF
Controle em Malha Aberta	IDF for unidirectional units IDF for bi-directional units IDF for actuators Dosing circuit Dosing circuit with analog input Extended dosing circuit	IDF_1 IDF_2 IDF_A DOS DOS_A DOS_E	BF BF BF BF BF BF
Padrão/Lógico	And Or Xor Not	AND OR EXOR NOT	FÇ FÇ FÇ FÇ
Padrão/ Operação Bit shift	Shift left Shift right Rotate left Rotate right	SHL SHR ROL ROR	FÇ FÇ FÇ FÇ
Padrão/ Estatístico	Minimum Maximum Average	MIN MAX AVER	FÇ FÇ FÇ
Padrão/ Comparador	Equal Greater equal Greater than Less than Less than equal Not equal	EQ GE GT LT LE NE	FÇ FÇ FÇ FÇ FÇ FÇ

Padrão/ Inversão	Binary Multiplexer Flip-flop Trigger	SEL MUX FF TR	FÇ FÇ BF BF
Padrão/ Detecção de Borda	Rising Falling	R_TRIG F_TRIG	BF BF
Padrão/ Funções Básicas	Integrator Differentiation Dead band Split-range Hysteresis Sample & Hold Selection of three	INTEG DIFF DEADB SPLIT HYST S_H OF3	BF BF BF BF BF BF BF
Padrão / Funções Texto	String length Left string part Right string part Mid string part Concatenate string Insert string Delete string part Replace string part Find string	S_LEN S_LEFT S_RIGHT S_MID S_CONCAT S_INS S_DEL S_REPL S_FIND	FÇ FÇ FÇ FÇ FÇ FÇ FÇ FÇ FÇ
Aritmética/ Aritmética Básica	Addition Multiplication Subtraction Division Modulo Power	ADD MUL SUB DIV MOD EXPT (XY)	FÇ FÇ FÇ FÇ FÇ FÇ
Aritmética / Numerica	Absolute value Square root N-th root Sign	ABS SQRT NRT SGN	FÇ FÇ FÇ FÇ
Aritmética / Logartimo	Natural logarithm Common logarithm Exponent	LN LOG EXP(eX)	FÇ FÇ FÇ
Aritmética / Trigonometria	Sine Cosine Tangent Arc sine Arc cosine Arc tangent	SIN COS TAN ASIN ACOS ATAN	FÇ FÇ FÇ FÇ FÇ FÇ
Aritmética / Limitador	Analog limiter Time limiter	LIMIT TLIMIT	BF BF

	Rate of change limiter	RLIMIT	BF
Conversor / Conversor de Tipos de Dados	Data type to BOOL Data type to BYTE Data type to INT Data type to UINT Data type to DINT Data type to UDINT Data type to WORD Data type to DWORD Data type to REAL Data type to TIME Data type to DT	TO_BO TO_BY TO_IN TO_UI TO_DI TO_UD TO_WO TO_DW TO_RE TO_TI TO_DT	FÇ FÇ FÇ FÇ FÇ FÇ FÇ FÇ FÇ FÇ FÇ
	Data type to STRING8 Data type to STRING16 Data type to STRING32 Data type to STRING64 Data type to STRING128 Data type to STRING256 Real to integer	TO_STR8 TO_STR16 TO_STR32 TO_STR64 TO_STR128 TO_STR256 TRUNC	FÇ FÇ FÇ FÇ FÇ FÇ FÇ
Conversor / Conversão Binária	Int to code* Code* to Int * BCD, GRAY, Aiken	INT TO CO CO TO INT	BF BF
	Pack BOOL to BYTE Pack BOOL to WORD Pack BOOL to DWORD Pack BYTE to WORD Pack BYTE to DWORD Pack WORD to DWORD Extract type to type Unpack BYTE to BOOL Unpack WORD to BOOL Unpack WORD to BYTE Unpack DWORD to BYTE Unpack DWORD to WORD	PBOBY PBOWO PBODW PBYWO PBYDW PWODW EXTCT UPBYBO UPWOBO UPWOBY UPDWBY UPDWWO	FÇ FÇ FÇ FÇ FÇ FÇ FÇ BF BF BF BF BF
Conversor Conversão DT	Pack to DT Unpack DT	P_DT UP_DT	BF BF
	Local time to summer time Summer time to local time	LT_TO_DST DST_TO_LT	BF BF
Constantes	BOOL constant BYTE constant	CSTBO CSTBY	BF BF

	DINT constant DT constant DWORD constant INT constant REAL constant TIME constant UDINT constant UINT constant WORD constant STRING8 constant STRING16 constant STRING32 constant STRING64 constant STRING128 constant STRING256 constant	CSTDI CSTDT CSTDW CSTIN CSTRE CSTTI CSTUD CSTUI CSTWO CS8 CS16 CS32 CS64 CS128 CS256	BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF
Funções de Sistema	Force cold start Force redundancy toggle	FCS PRIM_SEC	FÇ FÇ

*Tipo do Bloco: BF – Bloco Funcional; FÇ - Função



ABB Automation Products GmbH

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] Lewis, R. W. – IEC 61131-3 Programming Standard. Disponível no site <http://www.searcheng.co.uk/selection/control/Articles/IEC61131/main.htm> e acessado em 6 de Agosto de 2005.
- [2] *Help* (Ajuda) do Control Builder F.
- [3] Paulo Roberto Paixão Aguiar, Projeto de Graduação – Período 2004/02.
- [4] Marcos de Oliveira Fonseca, M. Sc. / ATAN – Curso Norma IEC 61131-3 para Programação de Controladores / ISA, 10 e 11 de Maio de 2005.
- [5] ABB, Control IT – IEC 61131 Control Languages Introduction. Development Version.
- [6] Hugh Jack – Automating Manufacturing Systems with PLCs, Version 4.7, April 14, 2005. Disponível no site http://claymore.engineer.gvsu.edu/~jackh/books/plcs/pdf/plcbook4_7.pdf e acessado em 06 de Agosto de 2005.
- [7] PLCopen - www.plcopen.org/

This document was created with Win2PDF available at <http://www.daneprairie.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.