Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Curso de Graduação em Engenharia Elétrica

Ruy Dantas Nóbrega

Relatório de Estágio Supervisionado

Campina Grande, Paraíba

Março de 2013

Ruy Dantas Nóbrega

Relatório de Estágio Supervisionado

Trabalho de Conclusão de Curso submetido à
Unidade Acadêmica de Engenharia Elétrica
Universidade Federal de Campina Grande
como parte dos requisitos necessários para a
obtenção do grau de Bacharel em Ciências no
Domínio da Engenharia Elétrica

Área de concentração: Automação Industrial, CLPs

Orientador:
Professor George Acioli Júnior, Dr.

Campina Grande, Paraíba
Março de 2013

Ruy Dantas Nóbrega

Relatório de Estágio Supervisionado

Relatório de Estágio submetido à
Unidade Acadêmica de Engenharia Elétrica
Universidade Federal de Campina Grande
como parte dos requisitos necessários para a
obtenção do grau de Bacharel em Ciências no
Domínio da Engenharia Elétrica

Área de concentração: Automação Industrial, CLPs

Aprovado em / /

Ruy Dantas Nóbrega
Universidade Federal de Campina Grande
Aluno

Professor George Acioli Júnior, Dr.
Universidade Federal de Campina Grande
Orientador

Agradecimentos

Agradeço à toda minha família, em especial à minha mãe, pai e irmã, que sempre me apoiaram sobre os momentos mais difíceis passados durante o curso.

À minha namorada e melhor amiga, Dayanne Rocha, por estar sempre ao meu lado, nas horas boas e ruins.

À todos os meus amigos, os antigos, e os novos que encontrei pelo caminho: Victor Loudal, Mário Júnior, Rubem Danilo, Pablo Vinícius, Carlos Ângelo, Lucas Borges, Marconni Freitas, Felipe Lucena, Juliano Leal, João Marcelo e muitos outros.

Finalmente, agradeço à meu orientador, George Acioli Júnior, por dar-me a oportunidade, com este projeto, de retribuir um pouco à universidade tudo que esta me ofereceu durante todo o curso.

"A ciência é uma perversão de si mesma, a menos que tenha como fim último, melhorar a humanidade" - Nikola Tesla

Sumário

1 Introdução

Apresentam-se neste relatório as atividades desenvolvidas no Estágio Supervisionado de 210 horas, realizado no Laboratório de Instrumentação Eletrônica e Controle (LIEC). O LIEC é um laboratório do Departamento de Engenharia Elétrica da Universidade Federal de Campina Grande. Suas pesquisas se concentram nas áreas de controle, automação e instrumentação eletrônica. A área de atuação do estágio foi em automação industrial.

A primeira atividade realizada foi o estudo do kit de treinamento eZTK900. Este kit inclui um Controlador Lógico Programável (CLP) modelo eZAP900 fabricado pela HI Tecnologia® acoplado a um módulo de treinamento com chaves tipo alavanca e LEDs de supervisão para entradas e saídas digitais. Há também suporte à entradas e saídas analógicas que podem ser acessadas através de pinos tipo banana.

A segunda atividade realizada foi o desenvolvimento de uma apostila sobre a linguagem *Ladder*. *Ladder* é uma das cinco linguagens-padrão definidas pela norma IEC 61131-3 e é a linguagem utilizada para a programação dos CLPs da HI Tecnologia®.

A terceira atividade realizada foi o desenvolvimento de guias experimentais para a utilização no laboratório da disciplina de Sistemas de Automação Industrial. Os guias experimentais utilizam os conhecimentos da apostila elaborada na segunda atividade para propor exercícios e práticas experimentais, a fim de familiarizar o aluno com CLPs e sua programação básica.

2 LIEC

O LIEC é um laboratório do Departamento de Engenharia Elétrica da Universidade Federal de Campina Grande, cujos integrantes trabalham nas áreas de Controle, Automação e Eletrônica.

Os projetos de pesquisa contam com a participação dos alunos do curso e do corpo docente. Dessa forma, o embasamento teórico e prático necessário ao futuro Engenheiro Eletricista é desenvolvido e solidificado.

O LIEC está situado no Setor C da Universidade Federal de Campina Grande em Campina Grande - Paraíba. Possui uma área de aproximadamente 600m^2 com oito Laboratórios de desenvolvimento, duas salas de apoio técnico, uma moderna sala de apresentação de trabalhos, salas para Pós-graduação e professores. Modernas instalações e um grupo de pessoas altamente qualificado fazem do LIEC um ambiente bastante agradável para se trabalhar e pesquisar.

As principais áreas de atividade são:

- Instrumentação Eletrônica;
- Controle e Automação Industrial;
- Redes Industriais;
- Sistemas Embarcados;
- Interface Homem-Máquina;
- Modelagem e Simulação Dinâmica;
- Otimização e Controle de Processos;
- Controle Regulatório Multivariável;
- Controle Avançado.

3 Kit de treinamento eZTK900

Na automação industrial, cada fabricante de CLP produz seu próprio *software* de supervisão e programação. Neste estudo será utilizado o kit de treinamento eZTK900, e o *software* que o acompanha, o SPDSW, produzidos pela HI tecnologia.

A HI tecnologia é uma empresa brasileira sediada na cidade Campinas, São Paulo, que atua nas áreas de fabricação de equipamentos e prestação de serviços para automação industrial.

3.1 Apresentação do kit eZTK900

O kit eZTK900 é um kit de treinamento baseado no controlador eZAP900, ambos da HI tecnologia. Este kit proporciona uma fácil interface de uso a partir de chaves, reostatos e LEDs para o eZAP900. A Figura 1 apresenta uma foto ilustrativa do kit eZTK900.

Figura 1: Foto ilustrativa do kit eZTK900

Este kit é alimentado por uma tensão de 85 a 265Vac e consome uma corrente máxima de 200mA, com potência de 1,3 Watts. No total, são 12 entradas digitais, sendo 8 ativadas por chaves tipo alavanca com LEDs de supervisão e 4 entradas via bornes tipo banana para entradas externas. As entradas `A`, `B` e `O` podem ser utilizadas para a implementação de um *encoder*, que é explicado na seção 4.4.3 deste relatório. Para as saídas digitais, conta-se com 8 LEDs de supervisão mais 4 bornes tipo banana para conexão externa. A saída `OO` pode ser utilizada como geradora de frequência e PWM. As figuras 2(a) e 2(b)

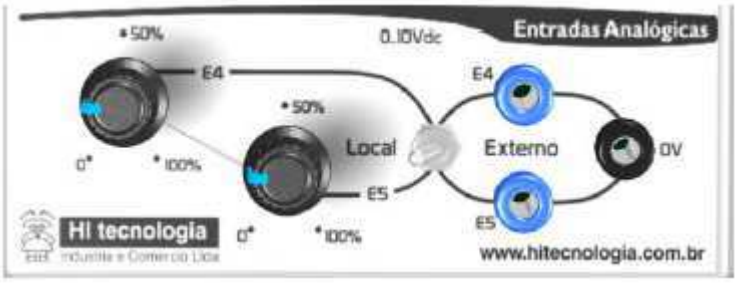
são fotos ilustrativas dos painéis de entradas digitais e saídas digitais.

(a) Painel de entradas digitais

(b) Painel de saídas digitais

Figura 2: Fotos ilustrativas dos painéis de entradas e saídas digitais

O kit ainda possui um painel para duas entradas analógicas que podem ser controladas via reostatos ou sinais externos, de 0 a 5V, de acordo com a posição da chave de seleção. Há também uma saída analógica que pode ser conectada externamente a partir de bornes tipo banana. As figuras 3(a) e 3(b) são fotos ilustrativas dos painéis de entradas e saída analógicas.



(a) Painel de entradas analógicas

(b) Painel de saída analógica

Figura 3: Fotos ilustrativas dos painéis de entradas e saídas analógicas

A interface homem-máquina se dá através do painel do CLP eZAP900, que permite a supervisão de variáveis como também a programação de parâmetros, sinalização de eventos e alarmes. A Figura 4 é uma foto ilustrativa do painel de interface do CLP.

O CLP eZAP900 ainda fornece, em sua parte traseira, uma porta Ethernet e uma porta serial para comunicação com outros dispostivos a partir de protocolos como o MODBUS-RTU, MODBUS-TCP e ASCII. A parte traseira também fornece entradas e saídas extras,

Figura 4: Foto ilustrativa do painel de interface homem-máquina

caso necessário. Na Figura 5 é apresentada uma foto ilustrativa da parte traseira deste CLP.

Figura 5: Foto ilustrativa da parte traseira do CLP eZTK900

3.2 Apresentação e configuração do *software* SPDSW

O SPDSW é um *software* proprietário da HI tecnologia que permite a programação da lógica de controle do CLP através de digramas *Ladder*. Na Figura 6 é mostrado a interface principal do SPDSW.

Para começar a programar para o CLP eZAP900, primeiro é necessário criar um novo projeto. Isto é feito clicando-se em *Projeto* → *Novo...* que abrirá uma janela para a entrada do nome do programa, a descrição, o responsável, a empresa, o número da versão e o número da revisão. Na Figura 7 é mostrado a janela de criação de novos projetos no SPDSW.

Depois de criado o novo projeto, é necessário selecionar o CLP que será utilizado. Para isto, clica-se em *Programa* → *Controlador programável*. Uma janela se abrirá com uma lista de CLPs da HI tecnologia. No caso desta apostila, escolhe-se o CLP eZAP900 e clica-se em *Confirmar*. Na Figura 8 é mostrado a janela de seleção de CLPs.

Figura 6: Interface principal do SPDSW

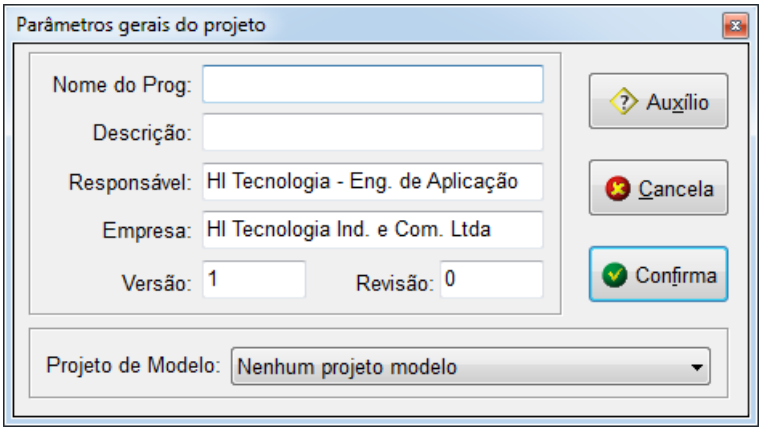


Figura 7: Interface de criação de projetos no SPDSW

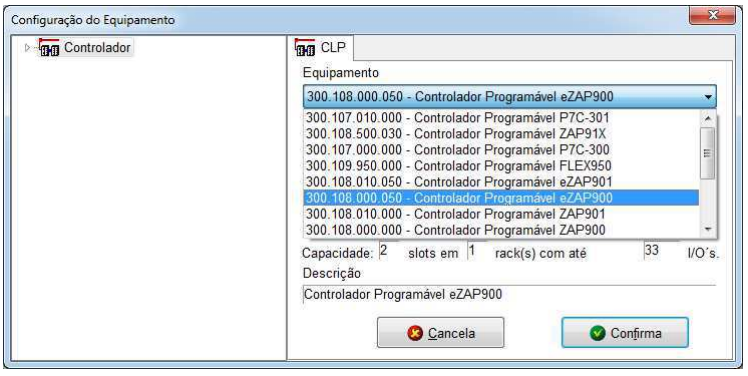


Figura 8: Interface de seleção de CLP

Com o novo projeto criado e configurado para o CLP correto, Deve-se conectar o CLP ao programa através do botão *Conectar*, na interface prinpal do programa (figura 6). Após a conexão, pode-se começar a programar o CLP, clicando em *Editor Ladder*. É possível programar em *Ladder* com o CLP não conectado, mas não será possível testar o programa. Na Figura 9 é mostrado a interface do editor *Ladder*.

O editor *Ladder* é bastante simples e intuitivo. As linhas verticais nas laterais, à

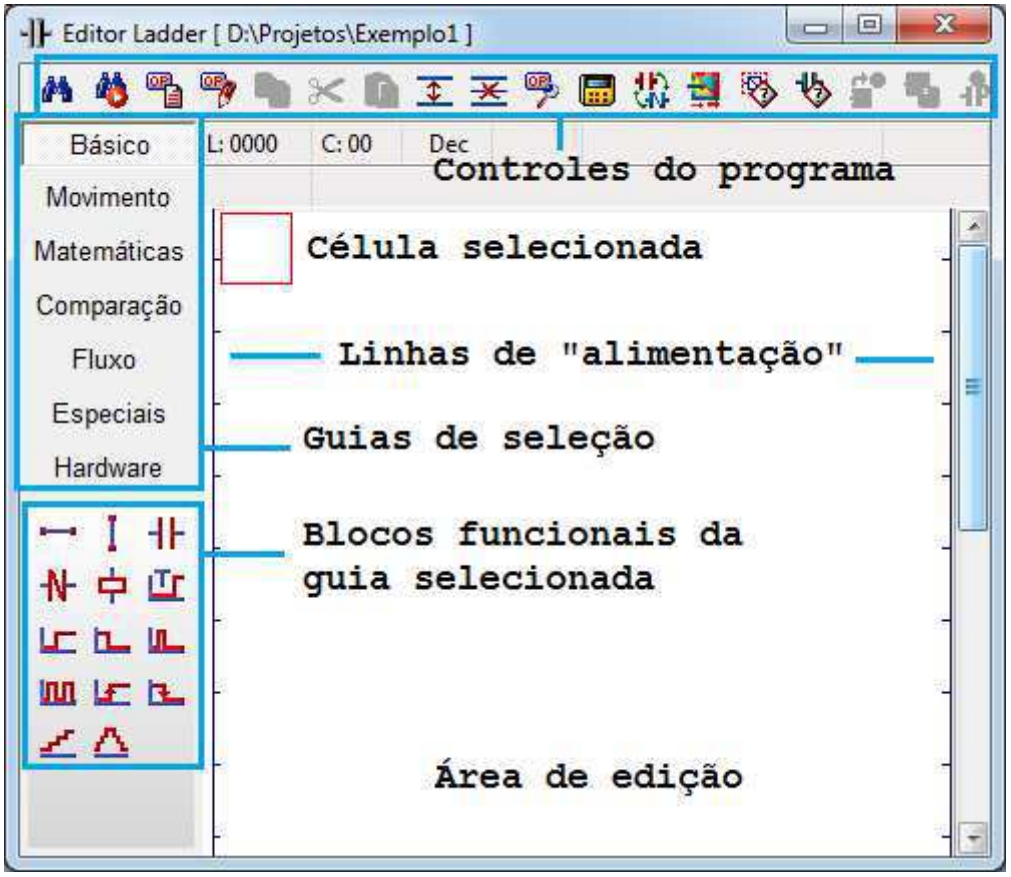


Figura 9: Interface do editor *Ladder*

esquerda e à direita, são as linhas de "alimentação"dos bloco funcionais, sendo a linha esquerda equivalente ao $+V_{cc}$, em um diagrama elétrico, e a linha direita ao terra.

A área de edição é onde os blocos funcionais são dispostos para construir o diagrama lógico. O retângulo vermelho indica qual célula está selecionada no momento. Para inserir um bloco funcional, escolhe-se a célula desejada, e clica-se no bloco funcional desejado à esquerda para inserí-lo no diagrama.

Os botões de controle do programa permitem a inserção e exclusão de linhas, compilação do programa, e programação de interface homem-máquina, caso desejado. Também há um botão dedicado de ajuda que permite ver uma breve e suscinta descrição do bloco funcional selecionado na área de edição.

Os blocos funcionais são organizados em guias, que ficam na parte esquerda da janela, de acordo com suas funcionalidades. Os blocos funcionais de cada guia são mostrados abaixo da seleção de guias. As tabelas 1, 2, 3, 4, 5 e 6 descrevem os blocos funcionais contidos em cada guia.

Guia: Básico			
Ícone	**Descrição (Atalho)**	**Ícone**	**Descrição (Atalho)**
	Linha Horizontal (H)		Linha Vertical (V)
	Chave NA (A)		Chave NF (F)
	Relé (B)		Temporizador (T)
	*SET* (S)		*RESET* (R)
	Pulso (P)		Oscilador (O)
	*SET* de Borda		*RESET* de Borda
	Contador Simples		Contador Bidirecional

Tabela 1: Descrição da guia *Básico*

Guia: Movimento			
Ícone	**Descrição (Atalho)**	**Ícone**	**Descrição (Atalho)**
	Movimentação (M)		Movimentação Indexada (X)
	Mov. Indexada Estendida		Inicialização de Dados (I)
	Conversão de Dados		

Tabela 2: Descrição da guia *Movimento*

Guia: Matemática			
Ícone	**Descrição (Atalho)**	**Ícone**	**Descrição (Atalho)**
	Soma (+)		Subtração (-)
	Multiplicação (*)		Divisão (/)
	Raiz Quadrada		Logaritmo na Base 10
	Exponencial		Potenciação
	Lógica *AND*		Lógica *OR*
	Lógica *XOR*		Deslocamento para Esquerda
	Deslocamento para Direita		

Tabela 3: Descrição da guia *Matemática*

Guia: Comparação			
Ícone	**Descrição (Atalho)**	**Ícone**	**Descrição (Atalho)**
	Igual		Diferente
	Maior Que		Maior ou Igual a
	Menor Que		Menor ou Igual a
	Teste *AND*		

Tabela 4: Descrição da guia *Comparação*

Guia: Fluxo			
Ícone	**Descrição (Atalho)**	**Ícone**	**Descrição (Atalho)**
	Início de Relé Mestre		Fim de Relé Mestre
	Início de Bl. de Lógica		Fim de Bl. de Lógica
	Bloco de Lógica		Fim de Programa

Tabela 5: Descrição da guia *Fluxo*

Guias: Especiais e Hardware			
Ícone	**Descrição (Atalho)**	**Ícone**	**Descrição (Atalho)**
	PID		*Fast Counter*
	Gerador de Frequência		

Tabela 6: Descrição das guias *Especiais* e *Hardware*

4 Apostila - Linguagem *Ladder*

4.1 Conceitos introdutórios

Esta apostila objetiva apresentar a linguagem de programação *Ladder*, que é uma das principais formas de se programar Controladores Lógicos Programáveis (ou CLPs). Esta linguagem é uma das cinco linguagens definidas na norma IEC 61131-3.

4.1.1 Definição - O que é CLP?

Controlador Lógico Programável (CLP ou, em inglês, *PLC, Programmable Logic Controller*) é um dispositivo eletrônico utilizado para a automação de processos industriais, como controle de motores, válvulas, sensores, etc.

CLPs são projetados para trabalhar em ambientes hostis e são capazes de suportar grandes variações de temperaturas, ambientes sujos ou empoeirados, imunidade a ruídos elétricos, e resistência à vibração e impacto. Os programas para o controle das máquinas são tipicamente armazenados em memórias somente de leitura, ou suportadas por baterias.

4.1.2 História e aplicações

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.

Na década de 1920, os dispositivos mecânicos foram substituídos pelos relés e contatores. A lógica a relés viabilizou o desenvolvimento de funções de controle mais complexas e sofisticadas mostrando ser uma alternativa de custo viável para automação de pequenas máquinas com um número limitado de transdutores e atuadores. Mas, para sistemas maiores, se tornava praticamente impossível a automação com lógica a relés por necessitar de uma equipe de técnicos e engenheiros atentos e bem treinados, para implementar, supervisionar e diagnosticar o sistema.

Com o surgimento dos circuitos integrados (CIs) na década de 1970, a lógica a relés começou a ser gradualmente substituída por sistemas eletrônicos, que eram mais compactos que os relés mas ainda eram bastante limitados. Os primeiros CLPs foram desenvolvidos para atender a demanda na indústria automobilística estadunidense com um conjunto de instruções reduzido, processando somente condições lógicas, de controle digital.

A primeira geração de CLPs se caracterizam pela programação intimamente ligada ao hardware do equipamento. A linguagem utilizada era o Assembly que variava de

acordo com o processador utilizado no projeto do CLP, ou seja, para poder programar era necessário conhecer a eletrônica do projeto do CLP. Assim a tarefa de programação era desenvolvida por uma equipe técnica altamente qualificada, gravando-se o programa em memória EPROM, sendo realizada normalmente no laboratório junto com a construção do CLP.

Na segunda geração, aparecem as primeiras linguagens de programação não tão dependentes do hardware do equipamento, possíveis pela inclusão de um programa monitor no CLP, o qual converte o programa escrito em linguagem de máquina. Os terminais de programação (ou maletas, como eram conhecidas) eram na verdade programadores de memória EPROM, que eram externos ao CLP. As memórias depois de programadas eram colocadas no CLP para que o programa do usuário fosse executado.

Na terceira geração, CLPs passam a ter uma entrada de programação, onde um teclado ou programador portátil é conectado, podendo modificar o programa do usuário, além de realizar testes no equipamento e no programa. Neste momento começa a haver uma maior preocupação em padronizar os CLPs, começando pelo seu formato, que tende para a modularização em *Racks*.

A quarta geração traz consigo uma porta de comunicação serial. Com a popularização dos microcomputadores, *softwares* de programação utilizando vários tipos de linguagem começam a surgir. Neste momento, tornou-se possível realizar simulações dos programas escritos e diagnosticá-los antes de serem programados nos CLPs, que começaram a ser programados a partir destes. Os microcomputadores também facilitaram a reutilização de programas, já que estes podiam ser facilmente guardados em disco para posterior uso.

Atualmente existe uma preocupação em padronizar protocolos de comunicação para os CLPss, de modo a proporcionar que o equipamento de um fabricante seja compatível com o equipamento de outro fabricante, não só CLPs, como controladores de processos, sistemas supervisórios, redes internas de comunicação e etc., proporcionando uma integração a fim de facilitar a automação, gerenciamento e desenvolvimento de plantas industriais mais flexíveis e normalizadas.

4.1.3 Princípio de funcionamento

CLPs trabalham de forma cíclica, sempre executando determinados passos de forma sequencial. A figura 10 mostra o ciclo de execução que pode ser dividido em três passos gerais.

O primeiro passo é o de inicialização do CLP, que acontece apenas uma vez, quando este é ligado. Neste passo é verificado as conexões do CLP, a consistência da memória e UCP, e desativa todas as saídas.

O primeiro passo do ciclo é a varredura pelas entradas do CLP. Neste passo, as entradas

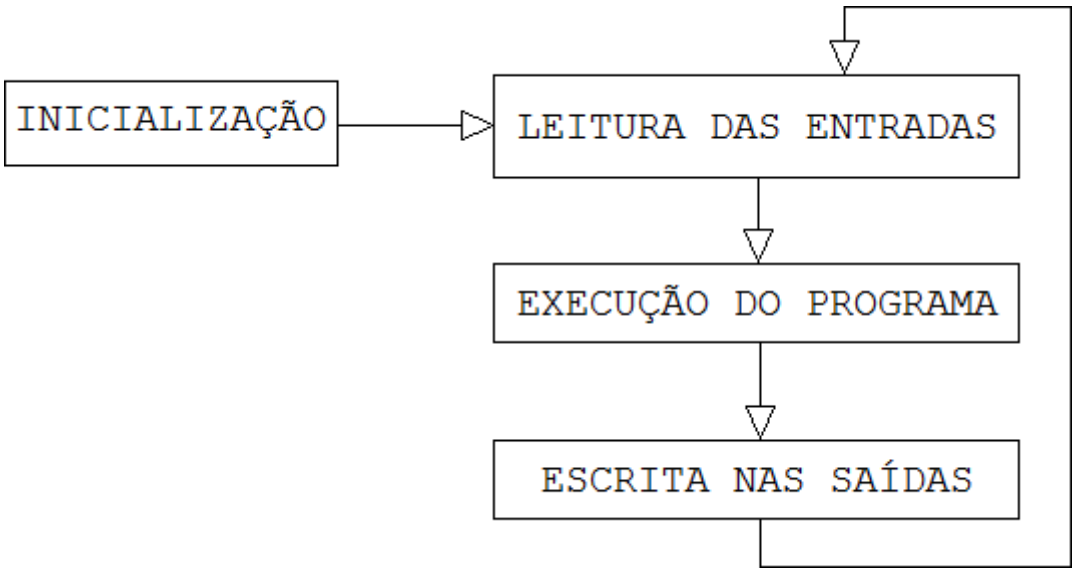


Figura 10: Representação do ciclo de execução de um CLP

(digitais e analógicas) são lidas e seus valores são copiados para a memória do CLP para serem usados durante a execução do programa. Deste modo, os valores das entradas não variam durante o resto da execução do ciclo, evitando inconsistências.

Com os valores das entradas atualizadas na memória, o programa principal é executado sequencialmente, instrução a instrução. Este passo atualiza a memória de acordo com o que é escrito no programa. É importante ressaltar que o programa não interage diretamente com as entradas e saídas do CLP, apenas com sua memória. O programa lê e escreve as entradas e saídas em posições de memória específicas que são depois utilizadas pelo CLP nos outros dois passos do ciclo.

No último passo, o CLP então escreve nas saídas (analógicas e digitais) os valores que são definidos pelo programa.

Após o ultimo passo, o ciclo se repete indefinidamente. O tempo de execução de cada ciclo varia dependendo das configurações e modelo do CLP. Este tempo de execução é referido como o *clock* do CLP. Não confundir com o *clock* da unidade de processamento do CLP.

4.1.4 *Hardware* de um CLP

Um CLP é composto basicamente por 4 partes: fonte de alimentação, unidade de processamento, memória e bateria. Todas são descritas nas próximas seções.

4.1.4.1 Fonte de alimentação

Converte a tensão da rede de 110/220V alternada em +5V, +12V ou +24V contínua para alimentar os circuitos eletrônicos, entradas e saídas e bateria interna.

4.1.4.2 Unidade de processamento

Comumente conhecida por CPU (do inglês, *Central Processing Unit*, ou em português, Unidade Central de Processamento) é composto por microprocessadores ou microcontroladores que executam as instruções do programa. Com mais funções sendo incorporadas aos CLPs e a necessidade corrente de controles mais rápidos, o *clock* das unidades de processamento chegam, hoje em dia, a unidade de Gigahertz.

4.1.4.3 Memória

Existem três tipos de memória em um CLP. Cada uma tem uma função específica para o funcionamento correto do CLP.

A memória do programa supervisor, é onde o programa supervisor do CLP é guardado. O programa supervisor monitora as entradas e saídas, inicializa o CLP, e realiza testes de diagnósticos. Pode ser dito que este programa é para o CLP o que um sistema operacional é para um microcomputador. Esta memória não pode ser modificada pelo usuário e normalmente é do tipo PROM, EPROM ou EEPROM.

O programa do usuário é guardado na memória do usuário. Esta pode ser interna do tipo RAM, EEPROM ou FLASH-EPROM, ou pode ser externa, com cartuchos de memória.

Por fim, há a memória de dados, que guarda todas as variáveis utilizadas pelo programa. Nesta também é guardado a memória-imagem das entradas e saídas, que são atualizadas pelo CLP nos primeiro e último passos do ciclo de execução.

4.1.4.4 Bateria

A bateria interna serve para a alimentação do circuito de relógio em tempo real, retenção de dados em caso de queda de energia e alimentação da memória do usuário, caso esta seja do tipo RAM. Pode ser do tipo Níquel-Cádmio (NiCd) mas, por causa de sua curta vida-útil e menor capacidade de armazenamento, estão sendo gradativamente substituídas por baterias de Íon de Lítio (Li-ion) e ou Níquel-Hidreto Metálico (Ni-MH), que são mais compactas e não possuem (ou no caso da Ni-MH, menor) efeito memória.

4.1.5 Unidades de entrada e saída

CLPs podem trabalhar tanto com variáveis digitais como analógicas. As próximas seções detalham os tipos de entrada e saída e exemplificam dispositivos compatíveis com as mesmas.

4.1.5.1 Entradas digitais

Entradas digitais são entradas que fornecem apenas dois valores, 0 (falso, inativo, desligado ou desabilitado) ou 1 (verdadeiro, ativo, ligado ou habilitado).Alguns exemplos de dispositivos digitais que podem ser ligados às entradas digitais de CLPs são: botões, botoeiras, chaves seletoras, pressostatos e termostatos.

4.1.5.2 Entradas analógicas

Variáveis analógicas são variáveis que podem ter qualquer valor. No caso dos CLPs, esta entradas analógicas podem assumir qualquer valor em uma pré-determinada faixa de tensão, que será convertida para unidade digital por um conversor analógico-digital (A/D). Exemplos de transdutores analógicos que podem ser ligados à essas entradas são: sensores de iluminação, temperatura, pressão e umidade.

4.1.5.3 Saídas digitais

Saídas digitais proporcionam apenas dois valores, 0 ou 1. Este tipo de saída é apropriada para dispositivos que apenas necessitam o controle de ligado e desligado. Exemplos de alguns dispositivos que podem ser utilizados com entradas digitais são as lâmpadas, motores de passo e contatores.

4.1.5.4 Saídas analógicas

As saídas analógicas são geradas pelo CLP através de um conversor digital-analógico (D/A). As saídas analógicas, na verdade, podem assumir apenas certos valores, que dependem da resolução do conversor do CLP. Este tipo de saída pode ser utilizada para controle de velocidade de motores, por exemplo, ou controle de abertura de válvulas, ou intensidade de iluminação de uma lâmpada, entre outros.

4.1.6 Classificação dos CLPs

Os CLPs são classificados por suas capacidades, como número de pontos de E/S (Entrada e Saída) e o máximo de instruções.

CLPs de pequeno porte possuem até 16 pontos de E/S e são normalmente compostos por apenas um módulo com no máximo 512 instruções.

CLPs de médio porte possuem até 256 pontos de E/S e capacidade de até 2048 instruções.

CLPs de grande porte são modulares, divido entre módulo principal e auxiliares. Permitem até 4096 pontos de E/S e a memória pode ser otimizada para suportar quantas instruções forem necessárias.

4.1.7 Norma IEC 61131-3

A partir da segunda geração de CLPs, os fabricantes começaram a desenvolver seus equipamentos e formas de comunicação com estes das mais diversas formas possíveis, deixando os CLPs de suas marcas cada vez mais únicos. Este tipo de abordagem comercial começou a tornar-se problemática para o mercado por não haver qualquer compatibilidade entre os equipamentos de diferentes fabricantes, deixando os clientes presos a apenas uma fabricante, e as fabricantes menores sem conseguir vender seus produtos.

Baseada nesta situação, a *International ElectroTechnical Commision* (IEC, Comissão Eletrotécnica Internacional, em português), publicou de 1992 a 1997, a norma 61131 (1131, na antiga numeração) que padroniza vários aspectos de CLPs. A norma é divida em oito partes:

1. Informações Gerais (*General Overview, Definitions*)
2. Requisitos de *Hardware* (*Hardware*)
3. Linguagens de Programação (*Programming Languages*)
4. Guia de Orientação ao usuário (*User Guidelines*)
5. Comunicação (*Message Service Specifications*)
6. Comunicação Via *Fieldbus* (*Fieldbus Communication*)
7. Programação Utilizando Lógica *Fuzzy* (*Fuzzy Logic*)
8. Guia para Implementação das Linguagens (*Implementation Guidelines*)

Esta norma estabelece vários critérios e características de CLPs, como requisitos mínimos, condições de trabalho, segurança, linguagens, intercomunicação e várias outras funcionalidades.

Para esta apostila, a parte interessante é a terceira, a IEC 61131-3 de 1993 (revisada em 2003), que trata da padronização da programação para CLPs. Esta parte da norma padrozina vários aspectos da programação como tipos de variáveis, operadores básicos, processamento multitarefas, escalonamento, blocos funcionais, recursos, programas, etc.

Em geral, a norma define 5 linguagens de programação para clps:

- Texto Estruturado (*Structured Text, ST*)

  É uma linguagem textual de alto nível similiar à linguagem Pascal. Útil para controles que utilizam muitos cálculos e expressões aritméticas complexas.

- Lista de Instruções (*Instruction List, IL*)

  Linguagem textual de baixo nível semelhante a *Assembly*. Ideal para pequenos programas, onde existem poucas quebras de fluxo de execução. Pela norma, IL é uma linguagem adicional, menos amigável e flexível que pode ser utilizada para produzir códigos otimizados para trechos de desempenho crítico em um programa.

- Diagram de Blocos Funcionais (*Function Block Diagram, FBD*)

  É uma linguagem gráfica baseada nos diagramas de circuitos, que representa blocos interconectados destacando o fluxo de sinais entre os elemtentos.

- Diagrama *Ladder* (*Ladder Diagram, LD*)

  Estudada nesta apostila, é uma linguagem gráfica baseada nos diagramas elétricos, que representam contatos e bobinas interconectadas destacando a energização entre os elementos.

- Sequenciamento Gráfico de Funções (*Sequential Function Charts, SFC*)

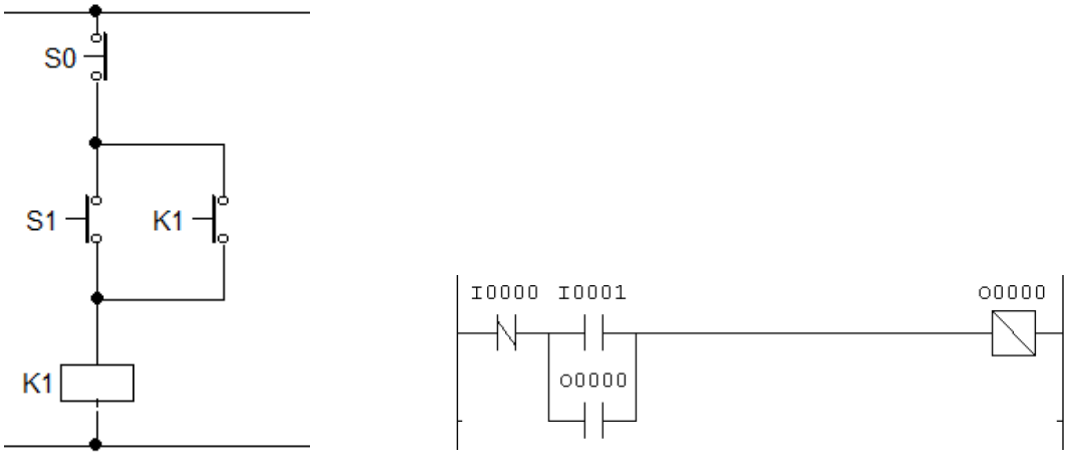
  É uma linguagem baseada em cadeia de eventos, muito similar a uma rede de *Petri*. Permite a visualização gráfica de todos os estados possíveis que o programa pode alcançar durante sua execução, tanto como as condições de transição (*triggers*, do inglês, gatilho) que levam o programa àquele estado.

O uso de qualquer destas linguagens depende de qual foi adotada pelo fabricante do CLP utilizado. Nesta apostila, o estudo é direcionado à linguagem *Ladder*, que é a linguagem mais comum dentre as cinco.

4.2 Programação *Ladder* - Básico

A linguagem *Ladder* foi a primeira linguagem a surgir para a programação de Controladores Lógicos Programáveis (CLPs). Foi desenvolvida juntamente com os primeiros CLPs na década de 70 para substituir a lógica de relés. É uma linguagem gráfica, desenvolvida para assemelhar-se aos diagramas elétricos que eram utilizados naquela época, facilitando a adoção desta nova tecnologia pelos técnicos e engenheiros que utilizavam a antiga lógica.

Assim como em um diagrama elétrico, *Ladder* possui duas barras verticais, uma representando a barra de alimentação (à esquerda) e outra a de neutro (à direita). Os elementos (chaves, relés e blocos funcionais) são dispostos em linhas horizontais, interligando as duas barras e construindo a lógica do programa. Apesar da semelhança com diagramas elétricos, deve-se lembrar que diagramas *Ladder* são executados sequencialmente, da esquerda para a direita, de cima para baixo.



(a) Diagrama elétrico de uma partida direta

(b) Diagrama *Ladder* de uma partida direta

Figura 11: Comparação de um diagrama *Ladder* e um diagrama elétrico

Na Figura 11 é mostrado a implementação da lógica de uma partida de motor direta em diagrama elétrico (lógica de relé) e em diagrama *Ladder*. Como esperado, o mais notável entre os dois diagramas é a semelhança entre eles. As diferenças aparecem apenas na forma de representar os blocos funcionais, que em *Ladder* são bastante intuitivos.

Nesta seção, serão introduzidos os blocos funcionais básicos de *Ladder*, que são as chaves, relés e os contatos auxiliares, e também blocos funcionais baseados em flip-flops, como os blocos *SET* e *RESET*.

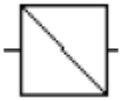
4.2.1 Chaves, relés e contatos auxiliares

Chaves e relés são os elementos básicos de *Ladder* e com eles é possível implementar diversas lógicas para as mais diversas aplicações na indústria tais como partida de motores, controle de nível de tanques, cotnrole de válvulas, etc.

Em *Ladder*, as chaves são representadas por duas barras verticais paralelas, e podem ser Normalmente Abertas (NA) ou Normalmente Fechadas (NF) como ilustrado nas Figuras 12(a) e 12(b) respectivamente. Relés são representados por um quadrado cortado com uma linha diagonal do topo esquerdo à base direita como na Figura 12(c).

(a) Chave Normalmente Aberta

(b) Chave Normalmente Fechada



(c) Relé

Figura 12: Representação *Ladder* de chaves e relés

Com estes três blocos básicos, já é possível implementar diversas lógicas. As seções 4.2.1.1 e 4.2.1.3 mostram implementações básicas em *Ladder* utilizando chaves e relés.

4.2.1.1 Funções lógicas em *Ladder*

A partir de simples chaves NA e NF, podemos montar diversas lógicas. Algumas destas são as lógicas booleanas, *AND*, *OR*, *NAND*, *NOR*, etc. Nesta seção, são mostrados a implementação de algumas destas funções em *Ladder*, e como implemtentar equações lógicas mais complexas.

4.2.1.1.1 Função *AND*

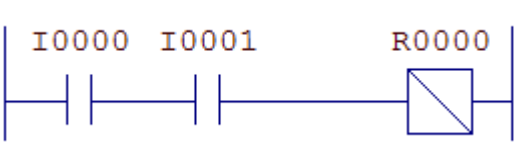


Tabela Verdade		
I0000	I0001	R0000
0	0	0
0	1	0
1	0	0
1	1	1

Figura 13: Implementação da função lógica *AND* em *Ladder*

Na Figura 13 é mostrado a implementação da função lógica *AND* em *Ladder* e sua tabela verdade. Esta função lógica habilita o relé `R0000` apenas se as entradas `I0000` e `I0001` estiverem habilitadas.

4.2.1.1.2 Função *OR*

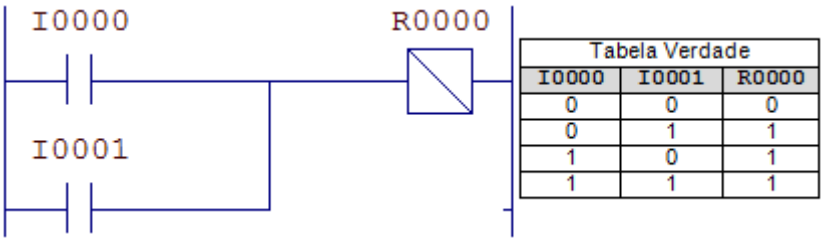


Figura 14: Implementação da função lógica *OR* em *Ladder*

Na Figura 14 é mostrado a implementação da função lógica *OR* em *Ladder* e sua tabela verdade. Esta função lógica habilita o relé `R0000` se pelo menos uma das entradas, `I0000` ou `I0001`, estiverem habilitadas.

4.2.1.1.3 Função *NAND*

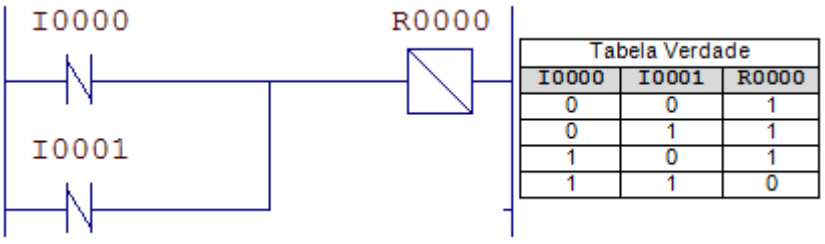


Figura 15: Implementação da função lógica *NAND* em *Ladder*

Na Figura 15 é mostrado a implementação da função lógica *NAND* (*NOT AND*), que é o contrário da função lógica *AND*, em *Ladder* e sua tabela verdade. Esta função lógica desabilita o relé `R0000` se ambas as entradas, `I0000` e `I0001`, estiverem habilitadas. Sua implementação desta forma vem da primeira lei de Morgan: $\overline{A * B} = \overline{A} + \overline{B}$.

É importante lembrar que $\overline{A * B} \neq \overline{A} * \overline{B}$, então apenas trocar as chaves NA por NF no diagrama da Figura 13, não transformará a lógica *AND* numa *NAND*, mas sim numa *NOR*, como é visto na seção 4.2.1.1.4.

4.2.1.1.4 Função *NOR*

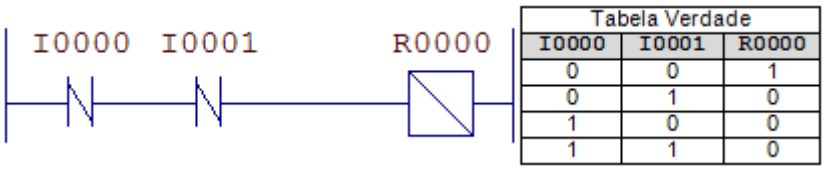


Tabela Verdade		
I0000	I0001	R0000
0	0	1
0	1	0
1	0	0
1	1	0

Figura 16: Implementação da função lógica *NOR* em *Ladder*

Na Figura 16 é mostrado a implementação da função lógica NOR ($NOT\ OR$), que é o contrário da função lógica OR, em *Ladder* e sua tabela verdade. Esta função lógica desabilita o relé `R0000` se uma das entradas, `I0000` ou `I0001`, estiverem habilitadas. Sua implementação desta forma vem da segunda lei de Morgan: $\overline{A+B} = \overline{A} * \overline{B}$.

4.2.1.1.5 Função *XOR*

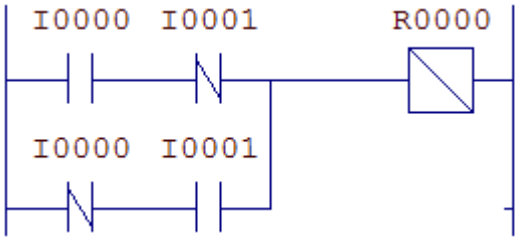


Tabela Verdade		
I0000	I0001	R0000
0	0	0
0	1	1
1	0	1
1	1	0

Figura 17: Implementação da função lógica *XOR* em *Ladder*

A função XOR ($eXclusive\ OR$) é a função lógica de "ou exclusivo", ou seja, o relé `R0000` é habilitado se apenas uma entrada for verdadeira e as demais, falsas. Sua implementação lógica é o conjunto de funções AND e OR de acordo com a equação: $A = B \oplus C = B * \overline{C} + \overline{B} * C$.

Na Figura 17 é mostrado a implementação desta função em *Ladder*, sendo $R_{0000} = I_{0000} * \overline{I_{0001}} + \overline{I_{0000}} * I_{0001}$.

4.2.1.2 Somador completo em *Ladder*

Utilizando as funções lógicas já vistas como base, é possível implementar equações lógicas mais complexas para gerar os mais diversos resultados. Implementaremos aqui um

circuito somador de 2 *bits* para somar dois números binários, A e B. O número A será representado pelas entradas `A1` e `A0`, e o número B, pelas entradas `B1` e `B0`. `A1` e `B1` são os *bits* mais significativos de cada número.

Primeiramente, será implementado o circuito meio-somador de 1 *bit*, ou seja, um circuito somador que não recebe o *carry* (em português, significa o "vai-um") como parâmetro de entrada. Este circuito somará os *bits* menos significativos, `A0` e `B0`, que não precisam de *carry* externo. Para isso, é necessário analisar a tabela verdade deste circuito, que é descrita na tabela 7.

$A_0 + B_0$	Z_0	C_0
$0+0$	0	0
$0+1$	1	0
$1+0$	1	0
$1+1$	0	1

Tabela 7: Tabela verdade de um circuito lógico meio-somador

Analisando a tabela 7, vemos que Z_0 é o resultado da soma, e C_0 é o *carry-out*, o *carry* que será o parâmetro de entrada (*carry-in*) para a soma do próximo algarismo. Pelos resultados da tabela, podemos implementar a variável Z_0 com uma simples porta *XOR*, ou seja, $Z_0 = A_0 \oplus B_0$ e o *carry-out* com uma porta *AND*, $C_0 = A_0 * B_0$. Na Figura 18 é mostrado a implementação da lógica em *Ladder*, sendo `Z0` o relé que representa resultado da soma e `C0`, o relé auxiliar que representa o *carry*.

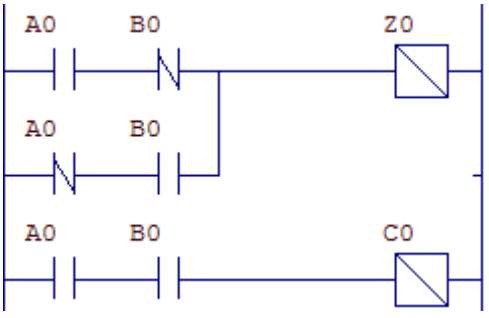


Figura 18: Implementação do circuito meio somador em *Ladder*

Na implementação da soma dos *bits* mais significativos de A e B, o *carry-out* resultante da soma dos *bits* menos significativos, `C0` é levado em consideração, pois a tabela verdade é diferente. Deste modo, implementa-se agora o circuito somador completo, que leva em consideração o *bit carry* no resultado. A tabela verdade para este circuito é apresentada na tabela 8.

Analisando os resultados da tabela 8, deduz-se que Z_1 pode ser implementado por uma porta *XOR* de três entradas. Quando a porta *XOR* tem mais de duas entradas, a

$A_1 + B_1 + Carry - in(C_0)$	Z_1	$Carry - out(C_1)$
$0+0+0$	0	0
$0+0+1$	1	0
$0+1+0$	1	0
$0+1+1$	0	1
$1+0+0$	1	0
$1+0+1$	0	1
$1+1+0$	0	1
$1+1+1$	1	1

Tabela 8: Tabela verdade de um circuito lógico somador completo

saída é verdadeira (ou habilitada) quando o número de entradas verdadeiras é ímpar, ou seja, para uma porta com 3 entradas, a saída é verdadeira quando há 1 ou 3 entradas verdadeiras simultaneamente. Por este motivo, na última linha da tabela, quando as 3 entradas estão ativas, Z_1 é verdadeiro. Em termos matemáticos, pode-se mostrar:

$$\begin{aligned} Z_1 &= A_1 \oplus B_1 \oplus C_0 \\ &= (A_1 \oplus B_1) \oplus C_0, \quad A_1 = B_1 = 1 \rightarrow A_1 \oplus B_1 = 0 \\ &= 0 \oplus C_0, \quad C_0 = 1 \\ &= 1 \end{aligned}$$

Após fazer as reduções possíveis utilizando as regras da álgebra de Boole, pode-se chegar à equação do *carry out* que é $C_1 = A_1 * B_1 + C_0 * (A_1 \oplus B_1)$. Assim, com a equação das saídas, pode-se projetar o diagrama *Ladder* completo para a soma de A e B. Na Figura 19 é mostrado o diagrama *Ladder* do somador de 2 *bits*.

No diagrama da Figura 19, estão o meio somador e o somador completo. O relé `AUX` é um relé auxiliar que recebe o valor da operação $A_1 \oplus B_1$ que é utilizado em outras partes do diagrama. Na tabela 9 é mostrado os possível valores de `A` e `B` e os resultados de cada operação. O *carry out* do somador completo (`C1`) é o bit de *overflow* da operação, mas pode ser interpretado, aqui, como um *bit* a mais para a representação da resposta, por isso, `C1Z1Z0` representa o resultado da soma de `A` e `B`.

4.2.1.3 Circuitos de selo

Na Figura 20 é mostrado o diagrama *Ladder* de uma partida direta de um motor. Analisando o diagrama, `I0000` e `I0001` representam entradas e `O0000` a saída. Quando a chave `I0001` é habilitada, o relé `O0000` é energizado e, assim, a chave `O0000` também é habilitada. Como o relé `O0000` está energizado, a chave `O0000` sempre estará ligada, fazendo o relé permanecer ativo, independente do estado de `I0001`. O relé só é desligado

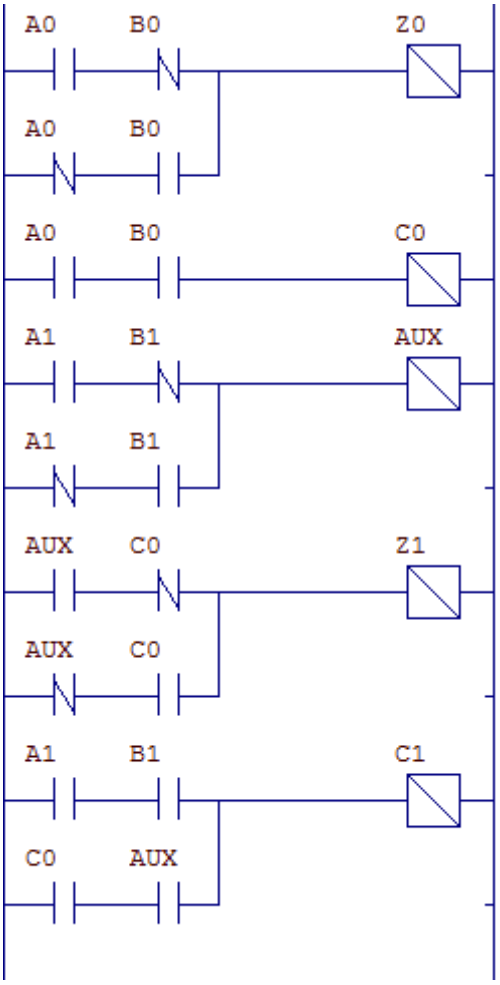


Figura 19: Implementação do circuito somador de 2 *bits*, em *Ladder*

$A_1A_0 + B_1B_0$	C_0	$C_1Z_1Z_0$
$00 + 00$	0	000
$00 + 01$	0	001
$00 + 10$	0	010
$00 + 11$	0	011
$01 + 01$	1	010
$01 + 10$	0	011
$01 + 11$	1	100
$10 + 10$	0	100
$10 + 11$	0	101
$11 + 11$	1	110

Tabela 9: Tabela verdade do circuito somador representado em *Ladder*

se pressionada a chave `I0000`. Este tipo de circuito é chamado de circuito de selo com prioridade no desligamento, pois se as duas chaves forem habilitadas ao mesmo tempo, o circuito sempre estará desligado.

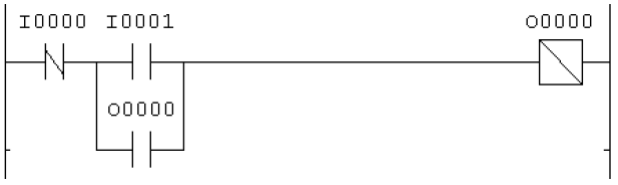


Figura 20: Circuito de uma partida direta de um motor em *Ladder*

Há também o circuito com prioridade no ligamento, que trabalha da mesma maneira que o selo com prioridade no desligamento, exceto que quando as duas entradas estão habilitadas, o relé estará sempre energizado. O diagrama do circuito selo com prioridade no ligamento é mostrado na Figura 21. Por razões de segurança, o selo com prioridade no desligamento é o mais utilizado.

Figura 21: Diagrama *Ladder* de uma partida direta com prioridade no ligamento

As chaves de entrada utilizadas nos diagramas das Figuras 20 e 21, (`I0000` e `I0001`), são do tipo `I` (de ***Input***, do inglês, Entrada) que são variáveis de entrada digital (0 ou 1, apenas). Já a saída `O0000` é do tipo `O` (de ***Output***, do inglês, Saída) que são variáveis de saída digital. Estes dois tipos de variáveis são ligadas logicamente a pinos do CLP e são o meio de comunicação deste com o meio externo. O número de entradas e saídas digitais depende do modelo e fabricante do CLP.

4.2.1.4 Contatos auxiliares

Muitas vezes, é necessário a utilização de relés temporários ou auxiliares. Para este tipo de situação, usa-se relés internos do tipo `R` que também são chamados de contatos auxiliares[1]. Na Figura 22 é mostrado a implementação em *Ladder* de uma porta *XOR* de 4 entradas que se utiliza de contatos auxiliares.

A saída deste diagrama segue a equação $O_0 = I_0 \oplus I_1 \oplus I_2 \oplus I_3 = (I_0 \oplus I_1) \oplus (I_2 \oplus I_3) = R_0 \oplus R_1$. Neste caso, é necessário a utilização dos contatos auxiliares `R0000` e `R0001` para diminuir o tamanho do diagrama e facilitar a leitura do mesmo. Caso não fosse utilizado

[1] o nome relé interno é apenas figurativo. Na verdade, os relés internos são dados guardados na memória interna do CLP, como variáveis. Este e outros tipos de dados são explicados na seção 4.3.1

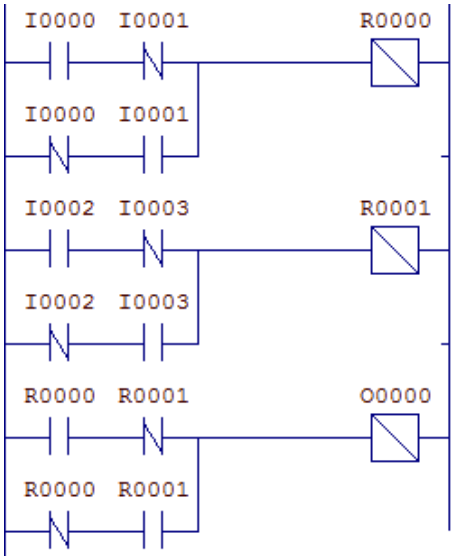


Figura 22: Diagrama *Ladder* de uma porta *XOR* de 4 entradas

contatos auxiliares, esta simples função se tornaria grande demais, aumentando as chances de erros na lógica.

Em outro exemplo, Na Figura 23 é mostrado o diagrama de uma partida sequencial, ou seja, uma ação só pode ser executada se outra determinada ação já tiver sido executada. No diagrama existem dois circuitos selo, sendo o primeiro selo para o contato auxiliar `R0000` a partir da entrada `I0001`, que liga a saída `O0000` na linha imediatamente depois. No segundo selo, só é possível ligar `O0001` se `R0000` estiver ligado. A entrada `I0000` desliga todo o circuito.

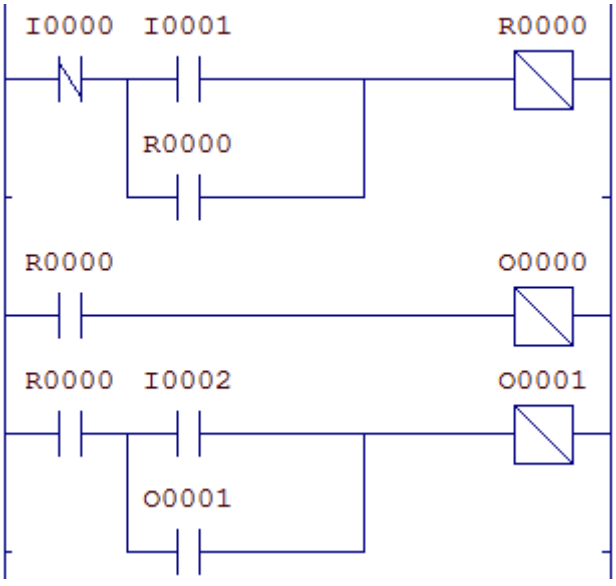


Figura 23: Diagrama *Ladder* de uma partida sequencial

Como CLPs são máquinas de memória relativamente limitada, uma boa prática de programação é evitar o uso desnecessário de recursos. Nota-se no exemplo da Figura 23 que o contato `R0000` não é realmente necessário neste caso, podendo ser substituído por `00000`, fazendo uso de menos recursos do CLP e eliminando uma linha do programa. Este tipo de prática facilita também a leitura do diagrama por outras pessoas posteriormente.

4.2.2 Blocos flip-flop

Flip-flops são circuitos digitais biestáveis com capacidade de armazenamento de memória de 1 *bit* cada. Este tipo de circuito é muito particular por ser causal, como o flip-flop tipo J-K que é mostrado na figura 24.

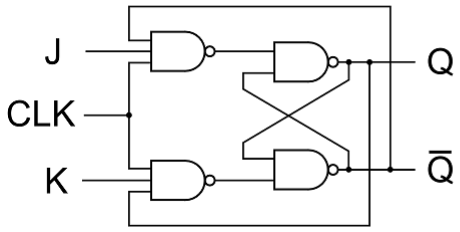
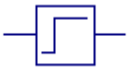


Tabela Verdade		
J	K	Q_t
0	0	Q_{t-1}
0	1	0
1	0	1
1	1	$\overline{Q}_{t-1}$

Figura 24: Circuito de um flip-flop do tipo J-K e sua tabela verdade

Na seção 4.2.1.3 viu-se como é possivel manter o estado da saída ativo, mesmo quando a entrada é desativada através de um circuito de selo. Porém, pode-se chegar ao mesmo resultado utilizando os blocos funcionais do tipo *SET* e *RESET*. Estes blocos em conjunto desempenham a mesma função de um flip-flop do tipo S-R (do inglês, *Set and Reset*)[2].



(a) Bloco *SET*

(b) Bloco *RESET*

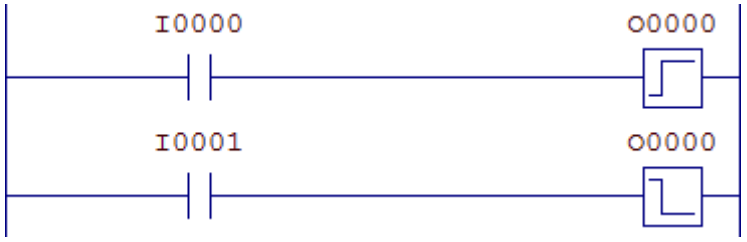
Figura 25: Representação *Ladder* dos blocos *SET* e *RESET*

O bloco *SET* verifica a entrada, e se esta estiver ativa, a saída é então ativada, mantendo este estado até que seja modificado por outros blocos ou chaves. É importante ressaltar que o bloco *SET* não desativa a saída, mesmo quando a entrada é desativada.

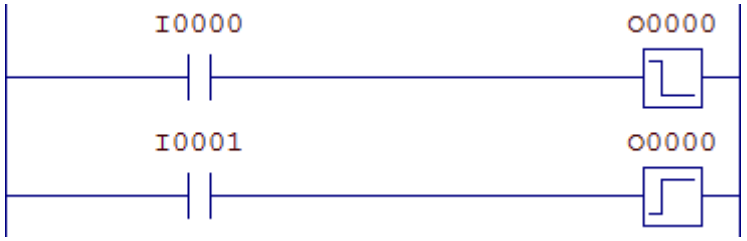
O bloco *RESET* é o oposto: quando a entrada está ativa, a saída é desativada, mantendo-se neste estado também indefinidamente.

[2]A tabela verdade de um flip-flop S-R e de um selo com *SET* e *RESET* como na Figura 26 diferem em um ponto: nos flip-flops tipo S-R, quando as duas entradas estão ativas, a saída é indefinida; já no selo com *SET* e *RESET*, a saída dependerá da ordem dos blocos no diagrama.

Na Figura 26(a) é mostrado o circuito selo com prioridade no desligamento e na Figura 26(b) é mostrado o circuito selo com prioridade no ligamento. A diferença dos dois circuitos está na ordem dos blocos *SET* e *RESET*. Como foi dito no início da seção, programas escritos em *Ladder* são executados de cima para baixo, da esquerda para direta. Assim, no circuito da Figura 26(a), caso as duas entradas estiverem ativas, a prioridade será do último bloco executado, o bloco *RESET*, mantendo a saída inativa. O mesmo ocorre no circuito da Figura 26(b), com o bloco *SET*, mantendo a saída ativa.



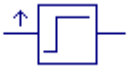
(a) Circuito selo com prioridade no desligamento



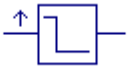
(b) Circuito selo com prioridade no ligamento

Figura 26: Circuitos selo compostos por flip-flops

Há também os blocos *SET* e *RESET* de borda, que funcionam do mesmo modo que o *SET* e *RESET* explicados, mudando apenas que são ativados na borda de subida (quando a entrada sai de inativa para ativa) do sinal de entrada, e não durante todo o tempo em que a entrada estiver ativa. Esta é uma diferença pequena mas importante: um selo feito com estes blocos, por exemplo, não tem prioridade no ligamento ou desligamento. Isto dependerá apenas de qual entrada foi ativada por último. Portanto, não é aconselhável selos com estes blocos.



(a) Bloco *SET* de borda

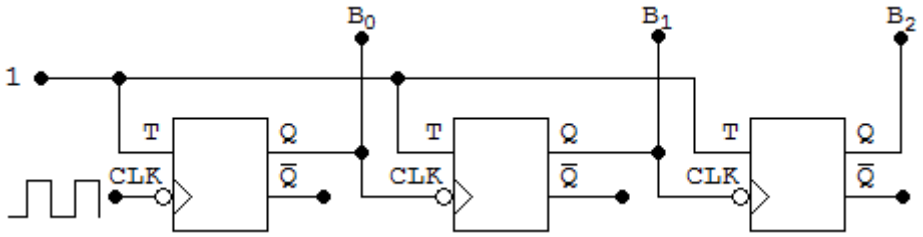


(b) Bloco *RESET* de borda

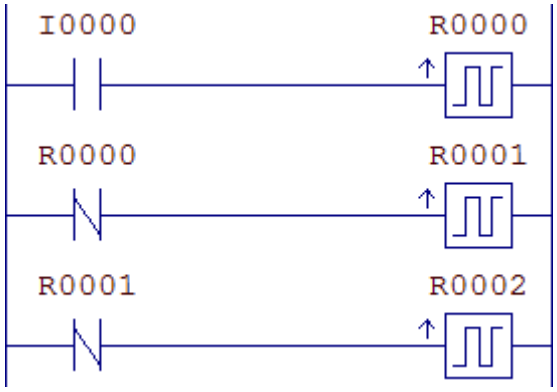
Figura 27: Representação *Ladder* dos blocos *SET* e *RESET* de borda

Em *Ladder* também é definido o bloco funcional oscilador. Este bloco funciona da mesma maneira que um flip-flop do tipo T (do inglês, *Toggle*, que significa Alternar): quando o sinal de entrada varia de inativo para ativo, a saída é invertida. Ou seja, caso

a saída esteja inativa, ela tornar-se-á ativa, e se estiver ativa ela será desativada. Com este flip-flop é possível, por exemplo, fazer um contador binário como mostrado na Figura 28(a). Na Figura 28(b) é mostrado um diagrama *Ladder* feito com osciladores e contatos auxiliares e na Figura 29 é mostrado o gráfico de onda das saídas para quando a entrada é uma série de pulsos[3].



(a) Contador 3 *bits* composto de flip-flops tipo T



(b) Contador 3 *bits* descrito em *Ladder*

Figura 28: Comparação de um contador de 3 *bits* composto por flip-flops tipo T e um composto por osciladores em *Ladder*

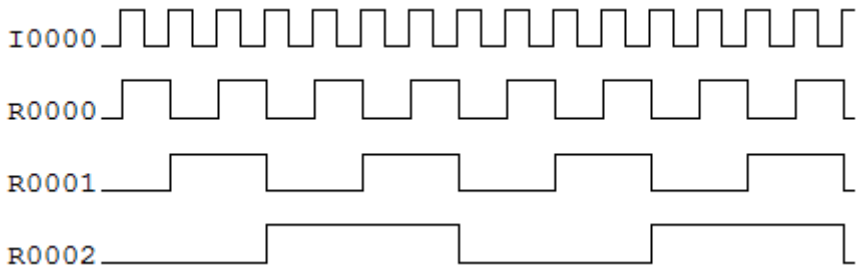


Figura 29: Gráfico de onda da entrada e das saídas do contador descrito em *Ladder* na Figura 28(b)

[3]Este contador é apenas didático. Seu uso na prática é desencorajado por ser altamente custoso em termos de memória e design de projeto. Na prática, utiliza-se blocos contadores que são apresentados na seção 4.3.2

4.3 Programação *Ladder* - Intermediário

Na seção 4.2 foram apresentados os blocos funcionais básicos de *Ladder*, são estes as chaves, relés, e blocos funcionais mais simples compostos de flip-flops. Com estes já é possível projetar algumas aplicações, por exemplo: partida de motores, controle de válvulas, acionamento de bombas, etc.

Contudo, apesar da semelhança com os diagramas elétricos, *Ladder* é uma linguagem de programação, e os CLPs são capazes de muito mais. Nesta seção serão apresentados blocos funcionais que realizam funções mais complexas, como os blocos temporizadores e contadores. Também será visto como ler e escrever em sensores e atuadores analógicos, e como realizar operações matemáticas e de comparação.

4.3.1 Tipos de dados

Em *Ladder*, existem vários tipos de dados. Já vimos anteriormente os dados do tipo `I`, `O`, e `R`, que representam entrada digital, saída digital e contatos auxiliares. Na tabela 10 são mostrados todos os tipos de dados disponíveis em *Ladder*[4].

Tipo	Valores	Descrição
`I`	0 ou 1	Entrada lógica, aceita apenas valores binários.
`O`		Saída lógica, aceita apenas valores binários.
`R`		Variável interna lógica, aceita apenas valores binários.
`E`	-32e3 a +32e3	Entrada analógica inteira de 16 *bits*.
`S`		Saída analógica inteira de 16 *bits*.
`M`		Variável interna inteira de 16 *bits*.
`K`		Constante interna inteira de 16 *bits*.
`D`	-34e37 a +34e37	Variável interna real de 32 *bits*.
`Q`		Constante interna real de 32 *bits*.
`X`	Texto ASCII	Variável interna de texto de até 48 *bytes*.
`W`		Constante interna de texto de até 48 *bytes*.
`T`	-	Identificador de blocos e sub-rotinas do programa.

Tabela 10: Tipos de dados disponíveis em *Ladder*

Analisando a tabela 10, vê-se que há apenas 5 tipos de dados no geral: lógico (ou

[4]Os tipos `X` e `W` dependem da existência destes no CLP para qual está sendo desenvolvido.

booleano), inteiro, real, texto e identificador. Todos os outros tipos são derivações destes 5 "básicos" como, por exemplo, um inteiro pode ser variável (`M`), constante (`K`) ou representação de uma saída ou entrada analógica (`S` e `E`).

Os dados em *Ladder* são exibidos na forma `YXXXX` onde `Y` é o tipo do dado e `XXXX` é o seu número de identificação na memória. Por exemplo, `M0001` é uma variável inteira com número de identificação `0001`.

4.3.2 Contadores

Na seção 4.2.2 é apresentado um contador de 3 *bits* composto inteiramente de flip-flops do tipo T. Apesar de serem de simples implementação, este tipo de contador não é muito utilizado pois, *Ladder* define blocos funcionais contadores que são mais versáteis e simples de utilizar. Na Figura 30 é mostrado a representação deste bloco.

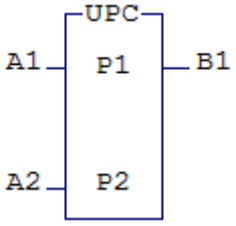
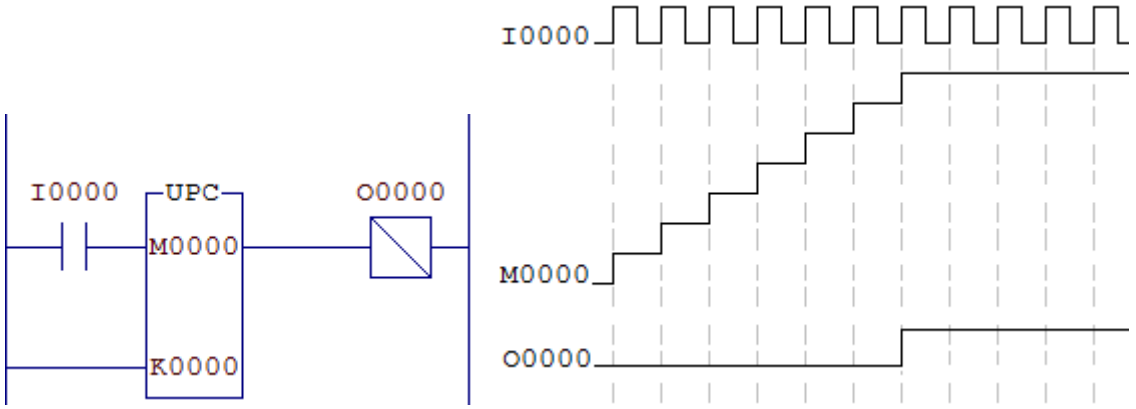


Figura 30: Representação *Ladder* de um bloco contador

O bloco contador possui duas entradas e leva dois parâmetros. A entrada `A1` determina a contagem do bloco: quando esta passa do estado inativo para o ativo, o contador é incrementado de hum. A entrada `A2` habilita ou zera o contador: quando A2 está inativo, a contagem é zerada, independente de `A1`, e quando está ativo, permite a contagem. O parâmetro `P1` guarda o estado da contagem e deve ser do tipo `M`(variável do tipo inteiro). É esta variável que é incrementada por `A1`, ou zerada por `A2`. O parâmetro `P2` é o máximo valor que o contador pode contar, ou seja, quando este valor é atingido (`P1=P2`), o contador para de contar. Este pode ser do tipo `M` ou `K`(inteiro constante), ou seja, o valor máximo da contagem pode ser variável ou não. Para sinalizar o término da contagem, a saída `B1` é ativada, caso contrário, permanece inativa. Na Figura 31(b) é mostrado o gráfico de `I0000`, `M0000` e `O0000` definidos no diagrama da Figura 31(a). Considere `K0000=7`.

Observando o comportamento de `M0000` e `O0000`, nota-se que o contador apenas conta uma vez e para: quando o bloco contador conta até o valor máximo (`M0000=K0000`), a saída `O0000` é ativada, permanecendo assim indefinidamente. Isto ocorre para que a saída do contador seja devidamente tratada pelo programador para então, se necessário, zerar o contador. Imaginemos o caso em que o contador permanecesse contando e zerando quando chegasse ao limite indefinidamente: dependendo do diagrama, o programa poderia não



(a) Diagrama *Ladder* (b) Gráfico de onda

Figura 31: Diagrama e gráfico de onda das variáveis de um bloco contador

detectar que a contagem chegou ao limite e zerou, perdendo o evento ocorrido, fazendo o programa não funcionar corretamente.

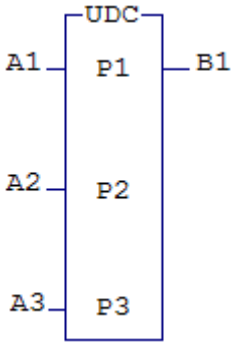
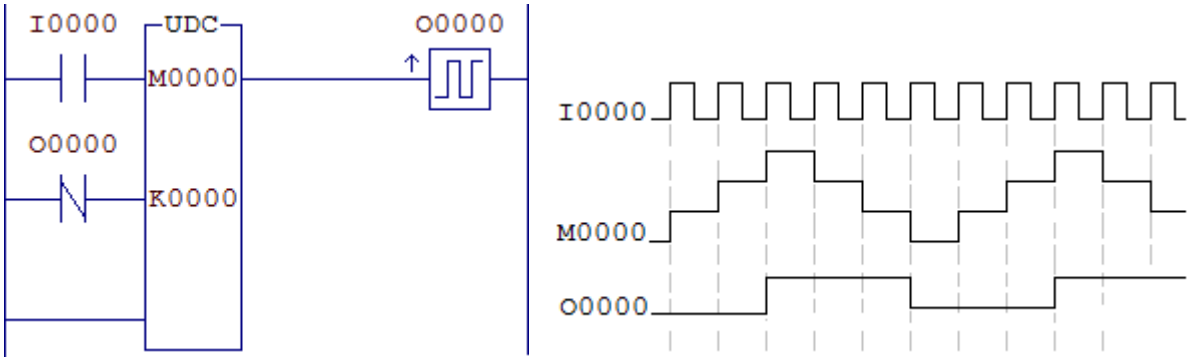


Figura 32: Representação *Ladder* de um bloco contador *UP/DOWN*

4.3.2.1 Contador bidirecional

Também há um outro tipo de contador, o contador bidirecional, que é capaz de contar para frente e para trás. A representação deste bloco está na Figura 32. Este bloco funciona de maneira similar ao bloco contador normal, apenas com uma entrada a mais. A entrada `A1` incrementa ou decrementa a contagem do bloco, dependendo do estado de `A2`. `A2` determina o sentido da contagem: se ativa, o sentido da contagem é direto (incrementa o contador) e, se inativa, o sentido da contagem é inverso (decrementa o contador). `A3` tem a função de habilitar ou reiniciar a contagem. Como a contagem é reiniciada depende do estado de `A2`: se `A2` estiver ativa, o contador é zerado, e se estiver inativa, o contador recebe o valor máximo definido pelo parâmetro `P2`. Um exemplo de

contador bidirecional contando indefinidamente é mostrado no diagrama *Ladder* da Figura 33. Considere `K0000=3`.



(a) Diagrama *Ladder* (b) Gráfico de onda

Figura 33: Diagrama e gráfico de onda das variáveis de um bloco contador bidirecional

4.3.3 Temporizadores

Temporizadores são muito utilizados em *Ladder*. De partida de motores, até controle de atuação de válvulas e eventos em geral. Sua representação *Ladder* é mostrada na Figura 34.

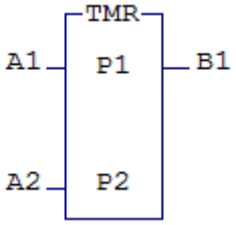


Figura 34: Representação *Ladder* de um bloco temporizador

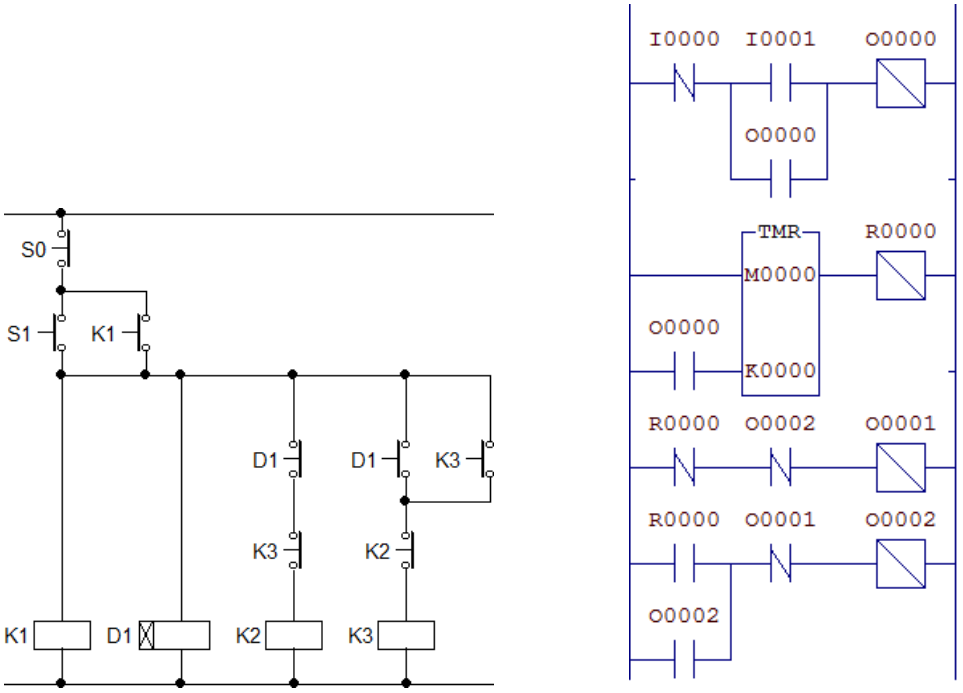
Com duas entradas e dois parâmetros, o funcionamento do bloco temporizador é semelhante ao bloco contador. O parâmetro `P1` é o tempo atual da contagem, significando quanto tempo falta para o término, e deve ser do tipo inteiro(`M`). O parâmetro `P2` é o tempo inicial da contagem e deve ser um inteiro variável ou constante (`K`). A entrada `A1` habilita ou pausa a contagem, e a entrada `A2` habilita ou reinicia a contagem. A tabela 11 detalha as ações possíveis a partir da combinação das entradas `A1` e `A2`.

O bloco temporizador conta em passos de 10ms (ou 0,01s), ou seja, deve-se seguir a equação $P_2 = 100 * t_{desejado}$ para que o temporizador conte corretamente. Por exemplo, caso seja requerido um tempo de 3 segundos, `P2` deve ser igual a 300. Deste modo, a cada 10ms, `P1` é decrementado de hum e quando este atinge zero, a saída `B1` é ativada. Como no contador, a saída permanecerá ativa até que o temporizador seja reiniciado.

A1	A2	Ação
0	0	Reinicia o temporizador e pausa a contagem
0	1	Contador pausado
1	0	Reinicia o temporizador
1	1	Habilita a temporização

Tabela 11: Opções de controle do bloco temporizador

Um exemplo de uso de temporizadores, é na partida de motores Δ-Y com temporizador, onde o operador não precisa comutar manualmente a ligação do motor de Y para Δ. Na Figura 35(a) é mostrado o diagrama elétrico de controle de uma partida Δ-Y e Na Figura 35(b) é mostrado o diagrama *Ladder*. Nota-se a semelhança de implementação quase direta dos dois diagramas, exceto pelas variáveis utilizadas em *Ladder*.



(a) Diagrama elétrico (b) Diagrama *Ladder*

Figura 35: Diagramas de partida Δ-Y com temporizador

4.3.4 Manipulação de dados

Nesta seção será apresentado como manipular dados em *Ladder*. Para tal, será apresentado o bloco funcional *MOV*, que permite a manipulação dos dados da memória do CLP.

Este bloco permite que dados sejam copiados entre posições de memória sendo, assim, capaz de ler e escrever dados nas entradas e saídas analógicas e digitais do CLP. Por isso, é importante seu estudo e entendimento para uma boa prática de programação *Ladder*.

4.3.4.1 Leitura e escrita analógica

O uso de sensores analógicos é de extrema importância para o controle de processos. Com eles é possível obter dados importantes de um processo como temperatura, pressão, luminosidade, pH, etc. A leitura de sensores analógicos em *Ladder* é muito simples e requer apenas o entendimento do bloco funcional *MOV*.

O bloco funcional *MOV* tem seu nome emprestado da linguagem de máquina *Assembly*, onde *MOV* é uma instrução para movimentação de dados entre os registradores e a memória da máquina. Em *Ladder*, o bloco tem função semelhante, movimentando dados dentro do CLP entre memória, saídas e entradas. O bloco é representado como mostrado na Figura 36.

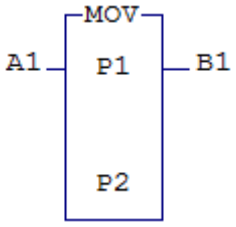


Figura 36: Representação *Ladder* de um bloco *MOV*

O bloco *MOV* tem apenas uma entrada. Quando `A1` está ativo, a operação de movimentação será efetuda e, se bem sucedida, a saída `B1` será ativada. O parâmetro `P1` é o valor que será atribuido à `P2`, ou seja, basicamente este bloco tem a função de fazer `P2 = P1`.

Como visto na seção 4.3.1, entradas analógicas são representadas pelo tipo de dado `E`, que é um inteiro variável. É neste tipo de dado que é guardado o valor lido pelo CLP de sensores que estejam ligados à ele. Considere o exemplo de um sistema de controle de temperatura da Figura 37.

O sensor LM35 é um sensor de temperatura. A saída deste está ligada à uma entrada analógica do CLP, a entrada `E0000`. Assim, para ler esta entrada analógica basta "mover"[5] o valor de `E0000` para onde deseja-se. No caso da Figura 38, para a variável inteira `M0000`.

A escrita nas saídas analógicas do CLP também é muito simples, e também utiliza-se apenas o bloco *MOV*. Neste caso, o parâmetro `P2` deve ser do tipo `S`, que é o tipo de

[5]O valor não é exatamente movido, é apenas copiado. O dado "movido" permanece o mesmo depois de copiado.

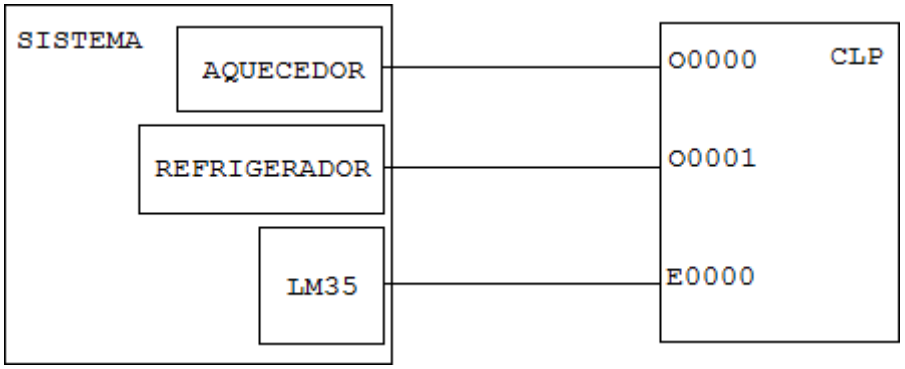


Figura 37: Diagrama de ligação entre um sistema de controle de temperatura e um CLP

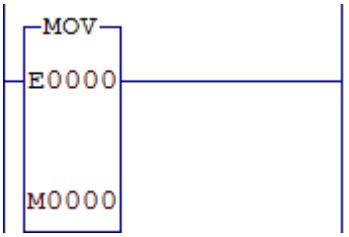


Figura 38: Diagrama *Ladder* para leitura da variável analógica `E0000`

dado referente à saída analógica. Um exemplo de diagrama de como escrever numa saída analógica está na Figura 39.

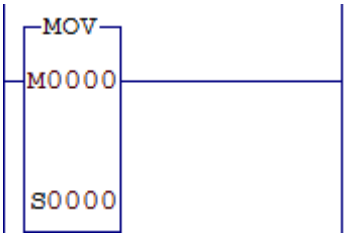
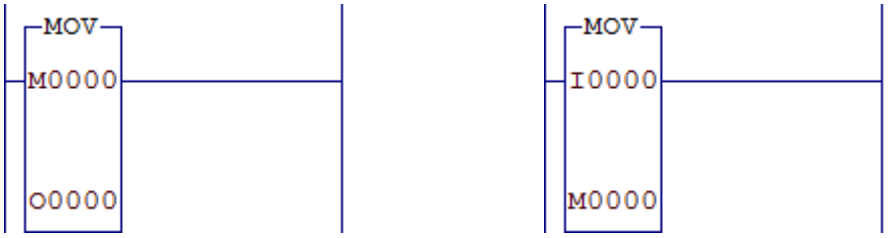


Figura 39: Exemplo de escrita em variáveis analógicas

4.3.4.2 Leitura/escrita digitais em bloco

Em alguns CLPs, também é possivel escrever em bloco nas saídas digitais. Isso também é feito utilizando o bloco *MOV*, com o parâmetro `P2` sendo do tipo `O` e seu índice deve ser múltiplo de 16 (0, 16, 32...). Neste modo, o CLP escreve o parâmetro `P1` (que pode ser um inteiro variável ou constante) *bit* a *bit* nas saídas digitais. O diagrama da Figura 40(a) é um exemplo de como escrever em bloco, escrevendo o valor de `M0000` nas saídas digitais de `O0000` a `O0015`.



(a) Diagrama *Ladder* para escrita digital em bloco

(b) Diagrama *Ladder* para leitura digital em bloco

Figura 40

Assim, se `M0000=157`, as saídas digitais seriam `O0015..O0000=0000000010011101`$_b$ que é exatamente a representação binária de 157 em 16 *bits*, sendo `O0015` o *bit* mais significativo e `O0000` o *bit* menos significativo.

Também é possível fazer a leitura em bloco de entradas digitais, utilizando o bloco *MOV*. Neste caso o parâmetro `P1` deve ser do tipo `I` e seu índice deve ser múltiplo de 16 e o parâmetro `P2` deve ser do tipo `M`. No diagrama da Figura 40(b) é mostrado um exemplo de como ler em bloco de entradas analógicas. Neste cenário, se as entradas digitais fossem `I0015..0000=1110110010110011` o valor lido seria `M0000=-4941`, de acordo com a representação numérica de complemento para 2.

4.3.5 Operações lógicas e matemáticas

Na seção 4.3.4.1 foi visto como ler e escrever dados em entradas e saídas analógicas. Estes dados muitas vezes são dados brutos (*raw data*, em inglês), ou seja, os dados lidos/escritos não foram tratados. Para isso, *Ladder* oferece blocos de funções matemáticas para o tratamento destes dados. Como exemplo, serão tratados os dados brutos gerados pelo sensor LM35 do circuito da Figura 37.

Segundo o *datasheet* do sensor LM35, a saída V_{out} é linearmente proporcional à temperatura, com proporção de $10mV/^oC$, seguindo a equação (1).

$$V_{out} = 0,01 * T \tag{1}$$

Ainda, considere que a entrada analógica do CLP tenha resolução de 12 *bits* e trabalhe na faixa de 0 a +5V[6]. Neste caso, o valor máximo lido (o máximo valor que `E0000` pode ter) é de 4095, ou $2^{12} - 1$. Assim, dividindo-se a faixa de valores $\Delta V = 5V$ pela resolução $R = 4096$ encontra-se a sensibilidade S da entrada analógica.

[6]A resolução das entradas e saídas analógicas, e a faixa de trabalho depende de cada modelo e fabricante de CLP, o manual deve ser consultado.

$$\begin{aligned} S &= \Delta V/R \\ &= 5/4096 \\ &\approx 1,22mV/unidade \end{aligned} \tag{2}$$

Com as equações (1) e (2), pode-se fazer uma ligação direta entre o valor lido na entrada analógica, e o valor da temperatura em ^oC. A equação (3) é usada para transformar um valor E, lido de uma entrada analógica, em temperatura T_{real}.

$$\begin{aligned} T_{real} &= \frac{V_{out}}{0,01} \\ &= 100 * V_{out} \\ &= 100 * S * E \\ &\approx 100 * 1,22 * 10^{-3} * E \\ &\approx 0,122 * E \end{aligned} \tag{3}$$

Deste modo, obtêm-se uma equação linear que transforma diretamente o valor lido de uma entrada analógica em temperatura. Quando o valor lido E for 220, por exemplo, a temperatura real será de $0,122 * 220 = 26,84^oC$.

Para realizar a implementação desta equação em *Ladder*, teremos que usar um bloco multiplicador, que é representado como na Figura 41. Este bloco leva três parâmetros, onde `P1` e `P2` são os operandos e `P3` guarda o resultado da multiplicação, ou seja, `P3 = P2*P1`. A entrada `A1` ativa a operação e `B1` é ativado quando a operação é bem sucedida.

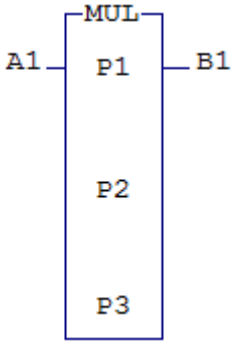


Figura 41: Representação *Ladder* de um bloco multiplicador

A Figura 42 é o diagrama *Ladder* da implementação da equação (3) utilizando bloco multiplicador. Neste diagrama, `Q0000` é uma constante real de valor igual 0,122 e `D0000` guarda o resultado da multiplicação.

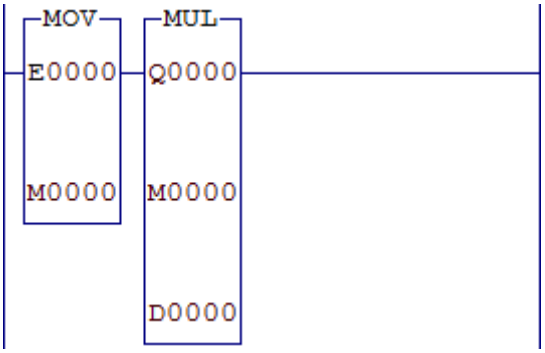


Figura 42: Implementação da equação 3 em *Ladder*

Uma nota importante a lembrar é que a ordem dos operandos importa. Neste caso, fizemos a multiplicação `Q0000*M0000`. Como o primeiro operando (`P1=Q0000`) é um número real, o segundo operando (`P2=M0000`) será convertido de inteiro para real, realizando a multiplicação, e guardando o valor em uma variável real. Mas caso inverta-se os operandos, `D0000=M0000*Q0000`, o segundo operando `Q0000` será convertido em inteiro pois o primeiro operando é um inteiro, resultando a multiplicação sempre em zero. Por exemplo, se o valor lido for 220, a multiplicação na primeira situação seria `D0000 = 0,122*220,0 = 26,84`. Já na segunda situação, `D0000 = 220*trunc(0,122) = 220*0 = 0`. Podia-se evitar esta preocupação utilizando uma variável real para guardar o valor lido de `E0000`.

Ladder também oferece outros blocos com funções matemáticas, com o último parâmetro sempre a receber o resultado da operação. É importante lembrar que a mesma regra para a ordem dos operandos na multiplicação vale para todos os bloco funcionais matemáticos. As seguintes operações são definidas:

- Soma (bloco `ADD`): $P3 = P1 + P2$
- Subtração (bloco `SUB`): $P3 = P1 - P2$
- Divisão (bloco `DIV`): $P3 = P1/P2$
- Potenciação (bloco `POW`): $P3 = P1^{P2}$
- Lógica *AND* (bloco `AND`): $P3 = P1 \& P2$
- Lógica *OR* (bloco `OR`): $P3 = P1 \| P2$
- Lógica *XOR* (bloco `XOR`): $P3 = P1 \oplus P2$
- Deslocamento binário para a esquerda (bloco `<<`): $P3 = P1 << P2$
- Deslocamento binário para a direita (bloco `>>`): $P3 = P1 >> P2$

- Raiz Quadrada (bloco `SQR`): $P2 = \sqrt{P1}$
- Exponencial (bloco `EXP`): $P2 = \exp P1$
- Logaritmo base 10 (bloco `LOG`): $P2 = \log P1$

Como é visto, *Ladder* não oferece um método fácil de resolver uma equação, com os blocos funcionais realizando apenas uma operação por vez. Ainda assim, os blocos matemáticos podem ser postos em uma única linha em série, facilitando um pouco a programação. O diagrama da Figura 43 é um meio de implementar a equação $Y = 2 * X^2 + 3 * Z$ em uma única linha, sendo `Y=D0000`, `X=D0001`, `Z=D0002` e as constantes `Q0000=2` e `Q0001=3`.

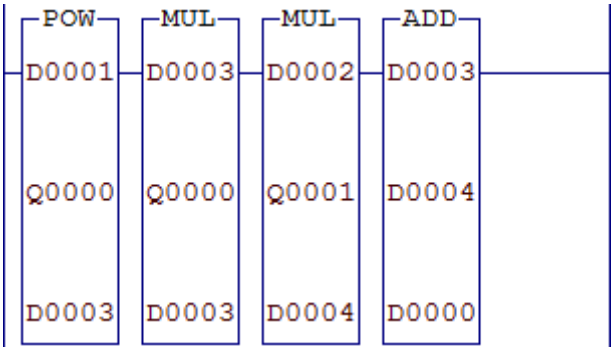


Figura 43: Implementação da equação $Y = 2 * X^2 + 3 * Z$ em *Ladder*

4.3.6 Operações de comparação

No circuito da Figura 37, é visto que o CLP está ligado ainda a um sistema de resfriamento e um sistema de aquecimento. O sistema de resfriamento pode ser um ventilador, por exemplo, e o de aquecimento uma simples lâmpada incandescente. Nesta seção é mostrado um simples projeto de controle de temperatura para este sistema, que se utiliza de blocos de comparação. Na Figura 44 é mostrado os blocos de comparação disponíveis em *Ladder*.

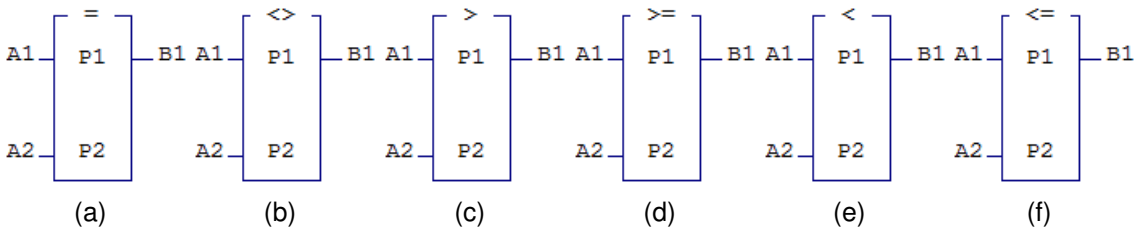


Figura 44: Blocos funcionais de comparação

Blocos de comparação são blocos funcionais que comparam dois valores através de um teste lógico. Estes blocos são ativados por uma única entrada e possuem dois parâmetros, `P1` e `P2`. O teste é sempre realizado de `P1` em relação a `P2`. Caso o teste lógico realizado com os dois parâmetros for verdadeiro, a saída `B1` será ativada, caso contrário, será destivada.

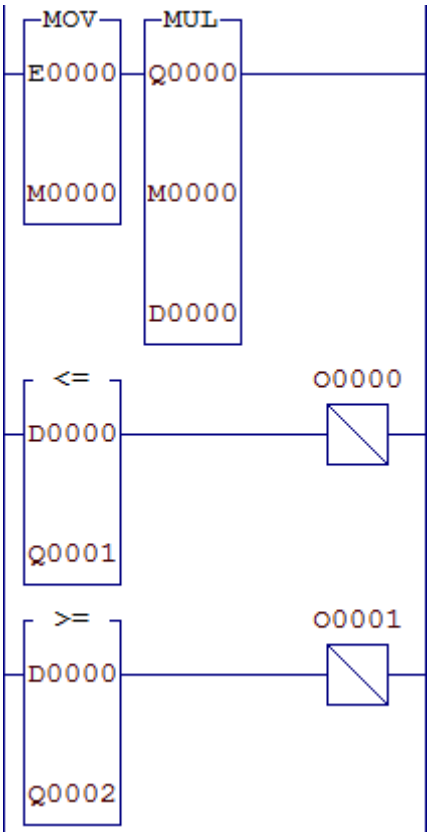


Figura 45: Diagrama *Ladder* de um projeto de controle de temperatura

No diagrama da Figura 42, o valor da entrada anlógica `E0000` é lido e guardado na variável `M0000` para ser mutiplicada pela constante real Q0000 de valor igual 0,122. Como é visto na seção 4.3.5, esta é a parte da transformação dos dados brutos lidos da porta analógica em valores de temperatura em ^{o}C. Agora, para controlar esta temperatura, deve-se controlar os sistemas de aquecimento e resfriamento. O diagrama da Figura 45 apresenta como este controle é feito.

Na parte de controle, há dois blocos de comparação. O primeiro bloco compara se a temperatura, `D0000`, é menor ou igual a constante real `Q0001`, se sim, a saída `O0000` é ligada. Já o segundo bloco compara se a temperatura, `D0000`, é maior ou igual a constante real `Q0002`, se sim, a saída `O0001` é ligada. As saídas `O0000` e `O0001` são ligadas ao aquecedor e refrigerador, respectivamente, de acordo com a Figura 37.

Nesta conFiguração, podemos utilizar as constantes `Q0001` e `Q0002` para determinar uma faixa de operação para um determinado processo. Por exemplo, se `Q0001=27` e

`Q0002=30`, o CLP tentará manter a temperatura do sistema entre 27^oC e 30^oC. Quando o sistema estiver abaixo de 27^oC o aquecedor é ligado, e quando a temperatura estiver acima de 30^oC, o refrigerador é ligado. Quando a temperatura estiver dentro da faixa, o aquecedor e refrigerador são desligados.

Este sistema de controle é bastante simples, pois não há controle de potência do aquecedor nem do refrigerador. Em modelos mais avançados, pode-se controlar mais eficientemente, utilizando as saídas analógicas ou *PWM* (do inglês, *Pulse Width Modulation*), a potência do aquecedor e refrigerador, de acordo com o gradiente de temperatura do sistema.

4.4 Programação *Ladder* - Avançado

Nas seções anteriores, foram apresentados e exemplificados vários tipos de blocos funcionais que, em conjuto, conseguem suprir a grande maioria das necessidades de controle da indústria moderna. No entanto, em alguns casos, é necessário o uso de controles mais complexos e elaborados.

Nesta seção, serão apresentados formas adicionais de controle que requerem conhecimentos extras como noções de programação procedural, PWM, *encoders* de rotação, filtros de primeira ordem e malhas PID. Cada assunto será superficialmente explicado, focando os esforços na implementação em *Ladder*.

4.4.1 Controle de fluxo de execução

Em *Ladder*, a implementação de diagramas torna-se, algumas vezes, cansativa e repetitiva. Por isso, *Ladder* define alguns blocos funcionais que ajudam a acelerar a programação e melhorar a reutilização de código. Aqui serão vistos como estes blocos funcionam e quais suas principais vantagens.

4.4.1.1 Relés mestres

Relés mestres são blocos funcionais importantes para uma boa prática de programação *Ladder*. Eles permitem delimitar uma região de linhas, com os blocos Início de Relé Mestre e Fim de Relé Mestre, que serão executadas somente se o bloco Início de Relé Mestre for ativado. Na Figura 46 é mostrado a representação *Ladder* destes blocos.

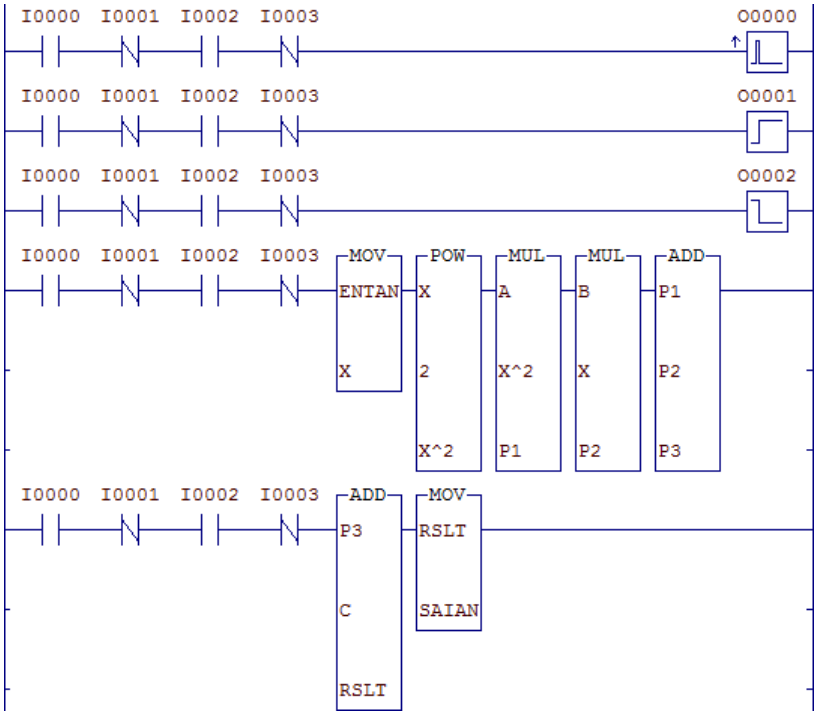
(a) (b)

Figura 46: Representação *Ladder* dos blocos início de relé mestre (a) e fim de relé mestre (b)

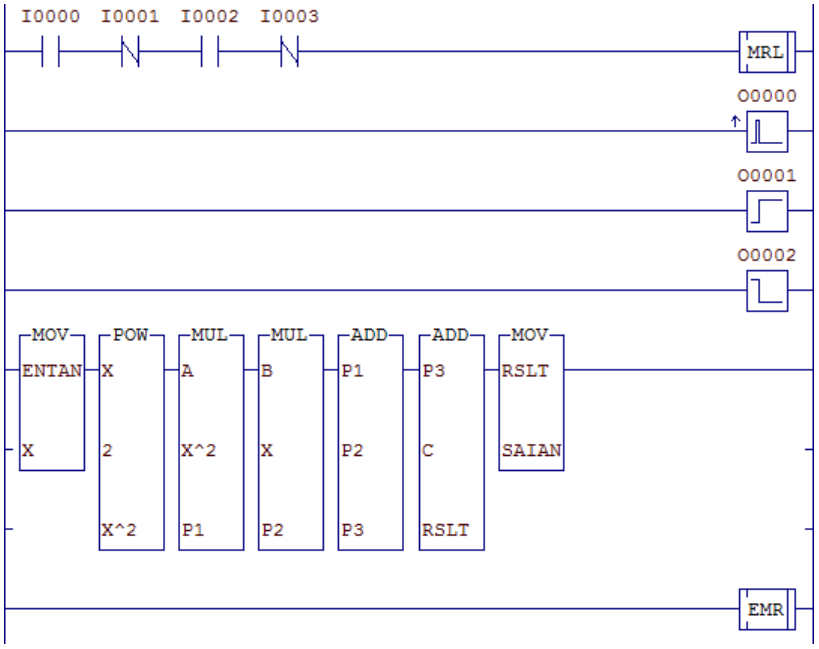
Estes blocos funcionais são extremamente úteis, pois permitem que sejam programadas várias linhas para uma mesma condição lógica. Na Figura 47 é mostrado a comparação de dois diagramas *Ladder*, com e sem o uso de relé mestre.

Os dois diagramas implementam a mesma lógica e funcionam da mesma forma. Quando a lógica $I_0 * \overline{I_1} * I_2 * \overline{I_3}$ é verdadeira, um pulso é enviado a saída `O0000`, `O0001` é ativada, `O0002` é desativada, e a saída analógica `SAIAN` recebe o resultado da equação $A * x^2 + B * x + C$, onde x é a leitura da entrada analógica `ENTAN`.

A diferença reside no modo de como a lógica foi implementada. No diagrama da Figura 47(a), a cada linha implementada a mais, o teste lógico com as entradas digitais



(a)



(b)

Figura 47: Comparação de dois diagramas *Ladder* que implementam a mesma lógica, sem relé mestre (a) e com relé mestre (b)

tem que ser refeito. Neste caso, o teste precisou ser refeito seis vezes para se conseguir implementar a lógica desejada. Este método polui a interface gráfica e dificulta a posterior interpretação do código, além de carregar o CLP com operações que podem ser evitadas.

Já no diagrama da Figura 47(b), quando o bloco Início de Relé Mestre é ativado, as

linhas entre ele e o bloco Fim de Relé Mestre são executadas de uma vez, sem a necessidade de retestar a lógica das entradas digitais a cada linha, tornando o diagrama muito mais legível.

Assim, o uso de relés mestres vai além da estética visual, melhorando a perfomance do programa, o que é crucial para CLPs, que são máquinas com capacidades de memória e processamento limitadas.

4.4.1.2 Blocos de lógica

Em programação, um dos conceitos mais importantes é o de funções. Pode-se definir funções como chamadas a rotinas ou "micro-programas"que recebem certos parâmetros de entrada, e retornam valores de acordo com sua lógica. Na linguagem C, por exemplo, a função `y = sin(x);` é uma função que recebe como parâmetro o ângulo `x` e retorna o valor do seno deste ângulo, que é guardado na variável `y`.

Em *Ladder*, pode-se implementar um conceito similar, utilizando-se os blocos de lógica. Na Figura 48 é mostrado a representação *Ladder* dos três blocos funcionais que definem um bloco de lógica.

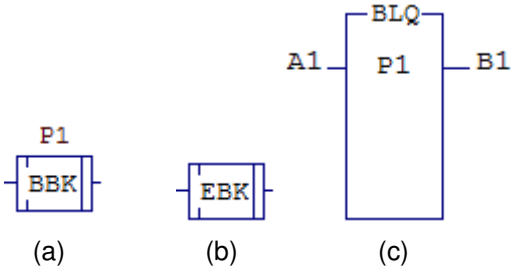


Figura 48: Representação *Ladder* dos blocos Início BBK (a) e Fim EBK (b) de Bloco Lógico, e o Bloco de Chamada de blocos lógicos BLQ (c)

Os blocos funcionais Início de Bloco Lógico (BBK) e Fim de Bloco Lógico (EBK) operam de maneira similar aos blocos Início de Relé Mestre e Fim de Relé Mestre. Os blocos BBK e EBK delimitam uma região de linhas que serão executadas quando o Bloco de Chamada (BLQ) for ativado.

Os blocos BBK e BLQ recebem apenas um parâmetro, `P1`, que deve ser do tipo `T`, que é o identificador do bloco. O identificador do bloco é utilizado para referenciar qual bloco de lógica é utilizado. Por exemplo, caso seja programado um bloco de lógica com o Início de Bloco Lógico (BBK) com o parâmetro `T0001`, deve-se utilizar o Bloco de Chamada (BLQ) com o mesmo parâmetro `T0001` para que este bloco de lógica seja executado.

A diferença entre os dois tipos de bloco, Relé Mestre e Bloco Lógico, é que o bloco lógico é ativado por um terceiro bloco, o Bloco de Chamada, enquanto o Relé Mestre é

executado se o bloco Início de Relé Mestre for ativado. Na Figura 49 é mostrado um diagrama *Ladder* que implementa uma mesma equação para parâmetros diferentes. É importante ressaltar que os blocos lógicos devem ser definidos após o bloco de fim de programa.

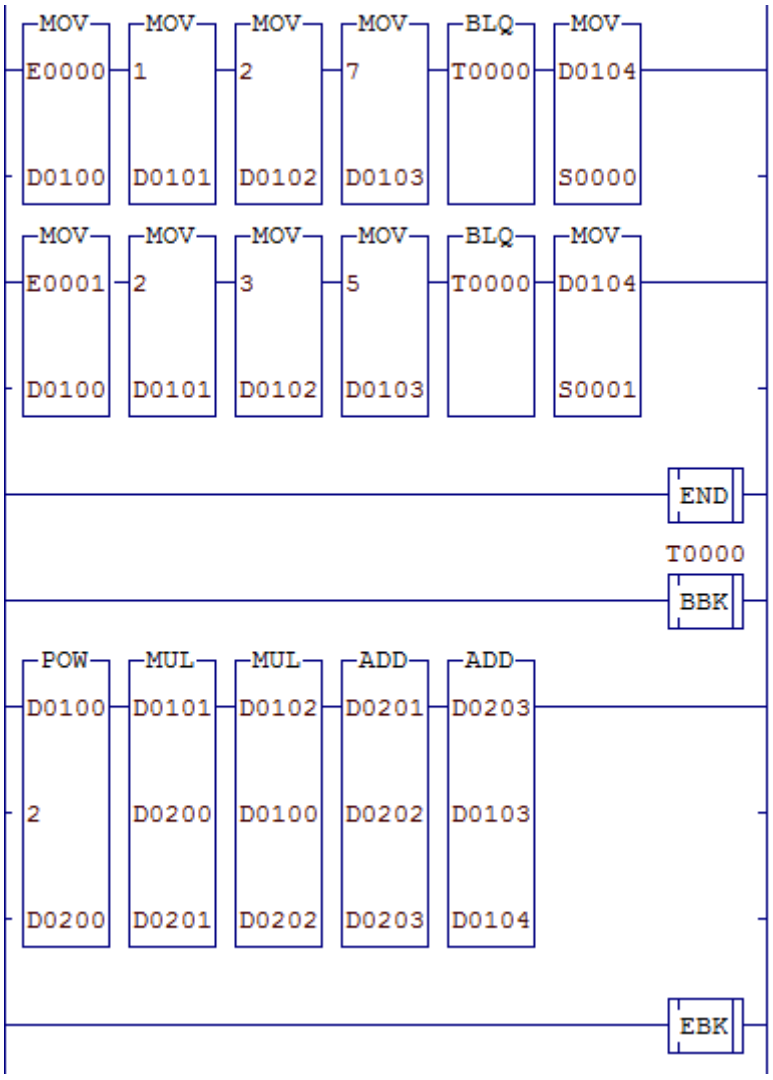


Figura 49: Diagrama *Ladder* da implementação de uma equação utilizando bloco lógico

O diagrama *Ladder* da Figura 49 implementa a equação de segundo grau $A * x^2 + B * x + C$ para duas saídas analógicas, `S0000` e `S0001`. Pode-se começar a entender o diagrama, analisando a linha entre os blocos BBK e EBK.

Na implementação da equação de segundo grau, `D0100`, `D0101`, `D0102` e `D0103` são os parâmetros de entrada x, A, B e C, respectivamente. As variáveis `D0200` a `D0203` são variáveis temporárias utilizadas para calcular o resultado. O resultado do cálculo é guardado em `D0104`.

Uma boa prática de programação *Ladder* é definir regiões de memória. Aqui, definiu-

se certa região de memória para ser utilizada apenas para parâmetros de entrada e saída de blocos lógicos, e outra apenas como variáveis temporárias. A região de memória `D0100-0199` é utilizado como a região onde são passados os parâmetros de entradas e saídas, e `D0200-0299` como a região de variáveis temporárias. A escolha dessas regiões foi arbitrária e pode ser definida de qualquer maneira.

No bloco BBK, é passado um parâmetro do tipo `T` que é o identificador do bloco lógico. Caso se tenha mais de um bloco lógico, cada um deve ter um identificador diferente. Este parâmetro `T` é o mesmo que é passado ao bloco BLQ, quando se deseja executar o bloco lógico.

Nas duas primeiras linhas do diagrama é onde acontecem as chamadas para o bloco lógico definido. Na primeira linha, utilizamos os blocos `MOV` para definir os parâmetros utilizados pelo bloco lógico, `D0100-0103`, como sendo a entrada analógica `E0000`, e as constantes reais `1`, `2` e`7`, que serão o x, A, B e C da equação, respectivamente. Depois, o bloco lógico `T0000` é executado e o resultado, `D104`, é escrito na saída analógica `S0000`. Em outras palavras, esta linha se resume a execução da equação $S_{0000} = 1 * E_{0000}^2 + 2 * E_{0000} + 7$. Seguindo a mesma lógica, a segunda linha executa a equação $S_{0001} = 2 * E_{0001}^2 + 3 * E_{0001} + 5$.

Percebe-se neste exemplo a economia de tempo que pode ser feita utilizando blocos lógicos. Caso este diagrama fosse escrito sem o uso de blocos lógicos, o cálculo da equação deveria ser implementada duas vezes, deixando o programa mais suscetível a erros de programação. A facilidade na modificação da lógica do bloco é também uma das principais vantagens, já que é apenas necessário modificar em um lugar.

4.4.2 PWM

No passado, quando era preciso controlar a potência fornecida de uma fonte a uma carga, eram-se utilizados reostatos (também conhecidos como potenciômetros) em série, que funcionava como um divisor de tensão e consumia parte da potência fornecida pela fonte. Na Figura 50 é mostrado um exemplo de como funcionava o sistema.

A utilização deste método implica em uma eficiência energética baixa, pois parte da potência fornecida pela fonte é sempre perdida no reostato em forma de calor.

Mas com o surgimento da eletrônica, e o desenvolvimento da eletrônica de potência, novas técnicas foram desenvolvidas para minimizar estas perdas, e obter mais controle sobre a potência fornecida. Uma destas técnicas é o chaveamento da alimentação da carga utilizando um sinal PWM. Na Figura 51 é mostrado um simples circuito com uma chave, que liga ou desliga o circuito.

Considere que a chave S1 seja controlada por um sinal que pode abri-la ou fechá-la, e que a carga seja resistiva. Determinando por quanto tempo a chave fica aberta e quanto tempo fica fechada, podemos controlar a potência média fornecida à carga. Caso queira-

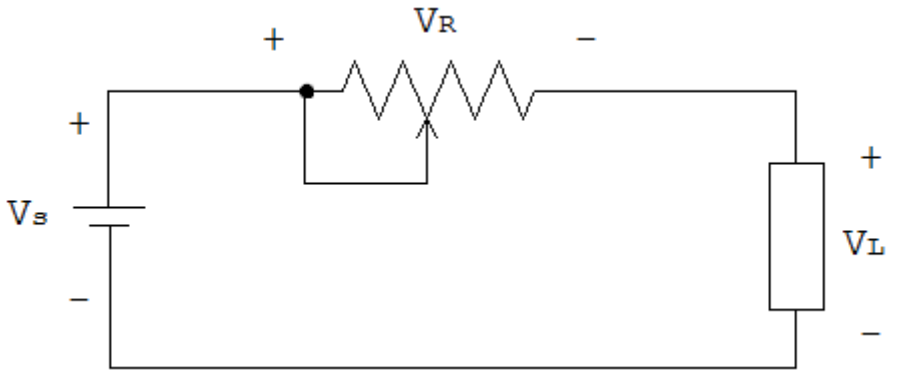


Figura 50: Circuito de uma carga alimentada por uma fonte em série com um reostato

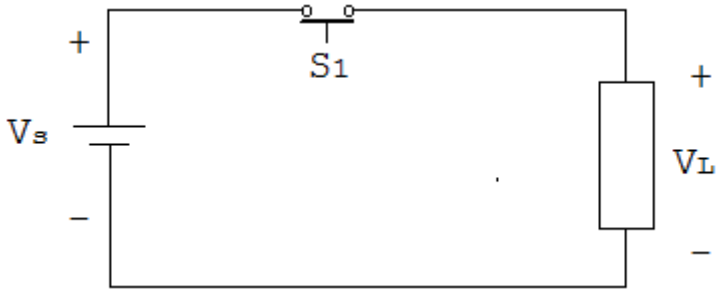


Figura 51: Circuito de uma carga alimentada por uma fonte em série com uma chave

se que a fonte forneça apenas 60% de sua potência total, por exemplo, a chave S1 deve permanecer ligada por 60% do tempo e desligada por 40%. Deste modo, a potência média fornecida pela fonte será 60% da potência máxima. Assim, durante o tempo em que a chave permanecer fechada, a fonte fornecerá a máxima potência possível, e quando estiver aberta, a fonte não fornecerá nenhuma potência. Na Figura 52 é mostrado o gráfico da potência média e instantânea fornecida para a carga de acordo com o chaveamento de S1.

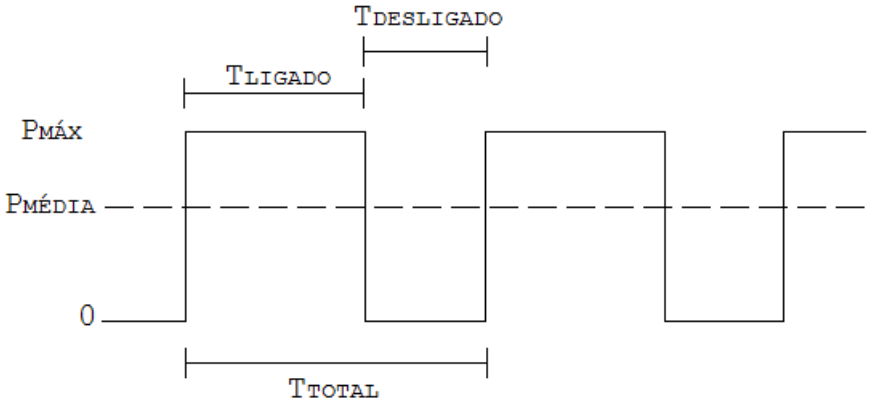
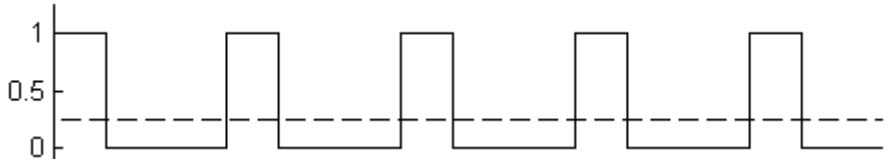


Figura 52: Gráfico da potência média e instantânea fornecida para a carga

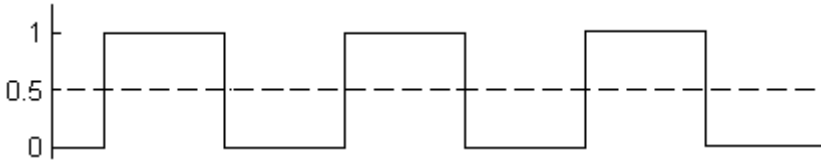
O sinal descrito na Figura 52 é um sinal PWM. Em outras palavras, PWM é um sinal retangular que tem a largura do pulso variável. A relação entre o tempo que a chave permanece ligada e o período do sinal é chamado de *duty cicle*, que significa a razão cíclica de trabalho, e é definida pela equação:

$$D = \frac{T_{ON}}{T_{TOTAL}} * 100\% \tag{4}$$

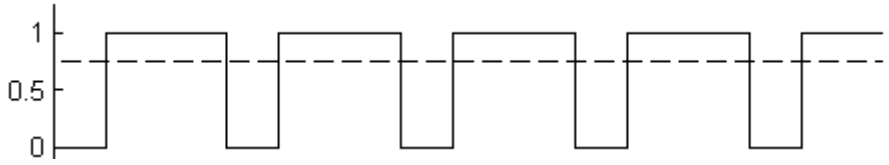
No caso do exemplo considerado, a razão cíclica é de 60%. Na Figura 53 é mostrado exemplos de sinais PWMs com diferentes razões cíclicas.



(a) 25% de razão cíclica



(b) 50% de razão cíclica



(c) 75% de razão cíclica

Figura 53: Exemplos de sinais PWM

Em *Ladder*, podemos gerar um sinal PWM a partir do bloco funcional gerador de frequência. Na Figura 54 é mostrado a representação *Ladder* deste bloco.

O parâmetro `P1` é do tipo `T` e indica em qual canal será gerado o sinal. O parâmetro `P2` é a frequência (em Hertz) do sinal, e o parâmetro `P3` é a razão cíclica. A razão cíclica é definida em décimos de porcentagem (0,1%), ou seja, se o sinal PWM tem razão cíclica de 80%, por exemplo, `P3` deve ser igual a 800. As entradas `A1` e `A2` ativam ou bloqueiam o gerador de frequência.

Na Figura 55 é mostrado um exemplo de diagrama *Ladder* que permite a permutação da razão cíclica entre 25% e 75% do bloco gerador de frequência. Considere `K0000=250`, `K0001=750` e `M0000=2000`. Neste exemplo, o gerador de frequência opera a 2kHz com razão cíclica de 25% ou 75%, dependendo das entradas `I0000` e `I0001`.

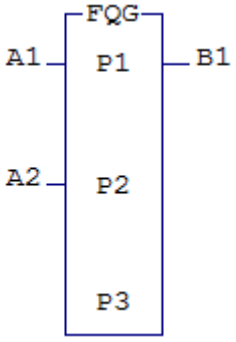


Figura 54: Representação *Ladder* de um bloco gerador de frequência

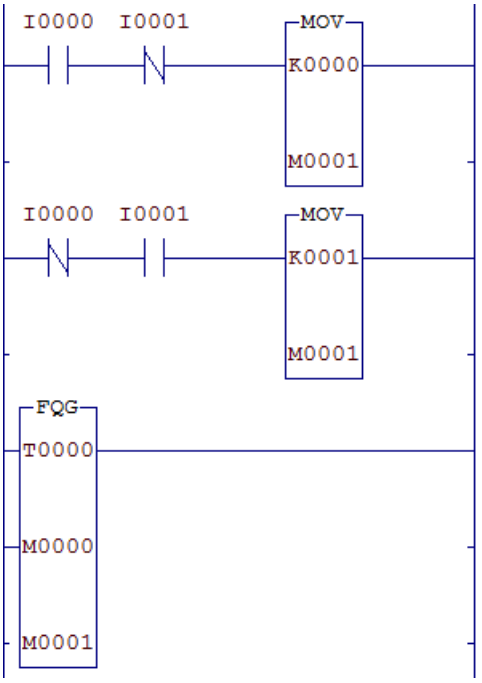


Figura 55: Diagrama *Ladder* de um gerador de frequência com razão cíclica mutável

4.4.3 *Encoders* de rotação

Encoder de rotação (ou simplemente *encoder*) é um sensor de posição angular que gera sinais elétricos mediante a rotação de seu eixo, podendo indicar de maneira precisa uma posição ou ângulo. Conectado ao eixo de um motor, por exemplo, será submetido a uma rotação a qual fará com que, internamente, um disco perfurado gire, interrompendo um feixe de luz que chega até um sensor ótico. Este é ligado a uma placa eletrônica que converte o sinal do sensor em pulsos (*encoder* incremental) ou em código binário (*encoder* absoluto). Na Figura 56 é mostrado o esquema de funcionamento de um *encoder*.

O disco utilizado pode ser de metal com furos ou de material transparente com partes escuras. Os dois tipos de *encoder* têm o mesmo esquema de funcionamento, diferenciando-

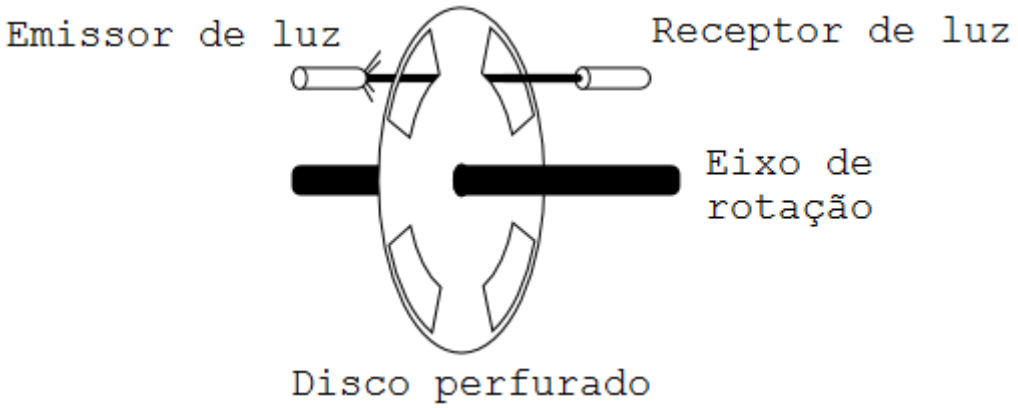


Figura 56: Esquemático de um *encoder* de rotação

se apenas na codificação contida no disco utilizado. As seções 4.4.3.1 e 4.4.3.2 explicam o funcionamento de cada tipo de *encoder*.

4.4.3.1 *Encoder* absoluto

Encoders absolutos possuem codificados em seu disco, números em formato binário que indicam a posição angular, o sentido da rotação (horário ou anti-horário) e a velocidade angular. Um exemplo de disco de *encoder* absoluto de 3 *bits* está nNa Figura 57.

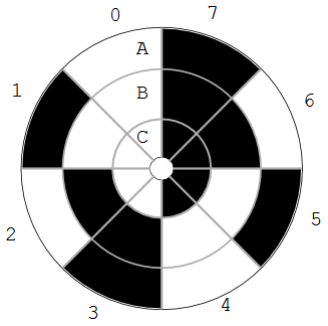


Figura 57: Disco perfurado de um *encoder* abstoluto de 3 *bits*

Cada trilha do disco (A, B e C) representa 1 *bit* do número, sendo o *bit* C o mais significativo e o *bit* A o menos significativo. Neste disco, os números codificados estão arranjados em ordem crescente (0, 1, 2, ..., 7) mas nada impede que seja usado outro tipo de arranjo, como o código Gray (0, 1, 3, 2, 6, 7, 5, 4) por exemplo.

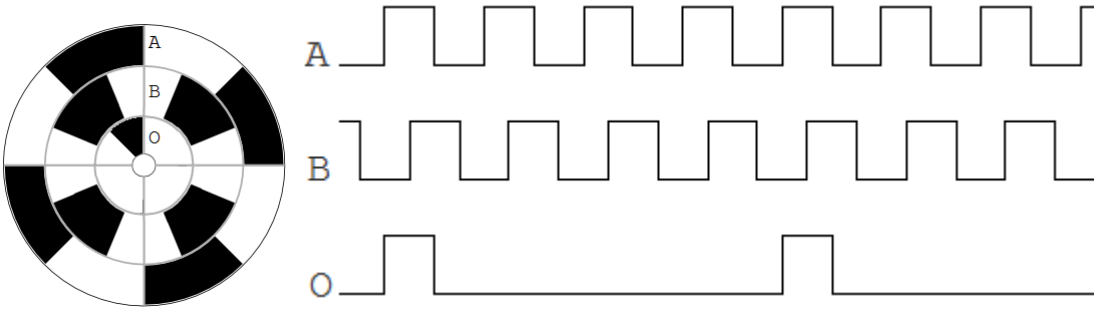
Deste modo, pode-se determinar a ordem de rotação do eixo, sabendo se a contagem é progressiva ou regressiva. Pode-se também determinar qual a posição angular do eixo baseado na numeração, com uma precisão de 45º.

Neste tipo de *encoder*, A precisão da posição angular do motor é determinada pelo

número de *bits* disponíveis. Isto pode se tornar um problema pois, caso se necessite de uma precisão de menos de 2º, por exemplo, o *encoder* deve ser de 8 *bits* ($360^o/2^8 \simeq 1,4^o$). Isto aumenta o número de trilhas, marcações e sensores utilizados. Para esse tipo de aplicação, o melhor *encoder* seria o incremental.

4.4.3.2 *Encoder* incremental

Neste tipo de *encoder*, o disco possui apenas três trilhas. Nas duas primeiras trilhas (A e B), as marcações são defasadas de 90º uma da outra, e a terceira (O) marca o ângulo zero do eixo, também chamada de trilha de sincronismo. Nas Figuras 58(a) e 58(b) são mostrados um disco de *encoder* incremental de 4 pulsos e os sinais de saída para a rotação no sentido horário.



(a) Disco perfurado de um *encoder* incremental

(b) Saída dos sinais A, B e O para rotação no sentido horário

Figura 58

Uma diferença do *encoder* incremental para o absoluto, é que todo o tratamento dos dados deve ser feito via *software*. O sentido da rotação é determinado pelas transições dos *bits* A e B. Se as transições forem $AB = 10 \rightarrow 11 \rightarrow 01 \rightarrow 00$, o eixo está girando em sentido horário mas, se forem $AB = 00 \rightarrow 01 \rightarrow 11 \rightarrow 10$ a rotação está no sentido anti-horário.

Um problema a ser considerado neste tipo de *encoder* é o posicionamento do eixo em determinado ângulo pois, deve-se saber a posição angular atual para determinar quantos pulsos[7] devem ser lidos para alcançar o ângulo desejado. Por exemplo, considerando o disco da Figura 58(a), se a posição angular for de 30º e se deseja posicionar o eixo em 120º, deve-se contar 1 pulso no sentido horário ou 3 pulsos no sentido anti-horário.

4.4.3.3 Implementação em *Ladder*

[7]Cada pulso do *encoder* incremental equivale a duas transições de AB.

Em *Ladder*, é possível utilizar, nativamente, apenas o *encoder* incremental. A escolha deste se dá pelo fato da padronização do número *bits* que não varia e é bastante reduzido (apenas três), se comparado ao *encoder* absoluto que pode ter o número de *bits* variável.

Para utilizar *encoders* incrementais, *Ladder* proporciona o bloco funcional *Fast Counter*, ou contador rápido, que pode ser configurado para intepretar sinais do *encoder*. Na Figura 59 é mostrado a representação *Ladder* do bloco funcional.

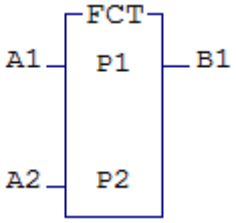


Figura 59: Representação *Ladder* de um bloco contador rápido

Este bloco é de muito simples implementação[8] e entendimento. O primeiro parâmetro deste bloco, `P1`, é o identificador, do tipo `T`, de qual canal será utilizado para ler os dados obtidos do *encoder*. O parâmetro `P2` é onde a contagem dos pulsos é armazenada e deve ser uma variável inteira. Caso a rotação seja horária, `P2` é incrementado, caso contrário, `P2` é decrementado. A entrada `A1` bloqueia a operação do bloco funcional quando inativa, e `A2` reinicia a contagem dos pulsos (`P2=0`) quando inativa. O bloco estará funcionando normalmente quando as duas entradas estiverem ativas. A saída `B1` indica se o bloco está ou não bloqueado.

Como exemplo, No diagrama *Ladder* da Figura 60 é mostrado a implementação de um leitor de velocidade a partir dos dados obtidos de um *encoder* incremental. Considere o *encoder* utilizado sendo de 128 pulsos e que os *bits* das trilhas A, B e O estão ligados às entradas `I0000`, `I0001` e `I0002`[9], respectivamente.

No diagrama da Figura 60, o temporizador está programado para contar 1 segundo, com a constante inteira `1SEG=100` e o tempo restante da temporização sendo guardado na variável inteira `TMR`. Enquanto a temporização ainda não está concluída, na terceira e quarta linhas, o bloco *Fast Counter* guarda o número total de pulsos lidos em `CONT`, variável inteira, que é copiado para `PULSO`, também variável inteira. Quando a contagem de tempo termina, o relé auxiliar `RESET` é ativado, reiniciando e bloqueando a temporização. O bloco *Fast Counter* também é parado e a variável `CONT` é zerada. Por fim, ainda enquanto o relé `RESET` está ativo, acontece a transformação do valor lido de pulsos, `PULSO`,

[8]Cada CLP determina como a ligação física ao *encoder* deve ser feita e por isso, este tópico não será abordado. Consulte o manual do CLP utilizado.

[9]É importante notar que, dentro do programa, não é necessário configurar as entradas `I0000`, `I0001` e `I0002` pois elas são configuradas fisicamente nos CLPs para serem utilizadas automaticamente pelo bloco *Fast Counter*.

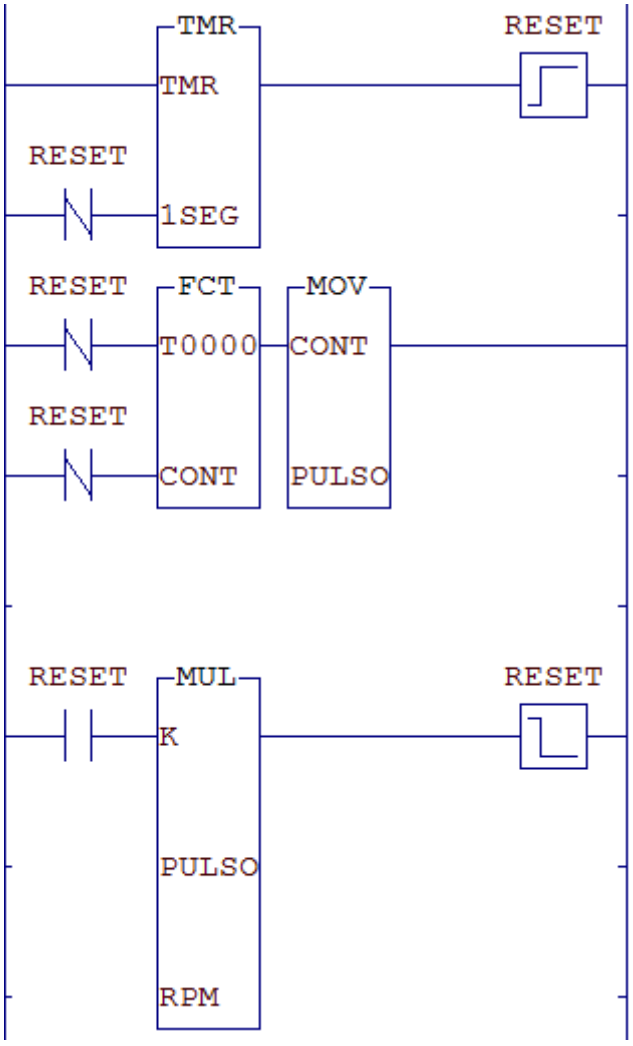


Figura 60: Diagrama *Ladder* para a leitura de velocidade de um *encoder*

para RPM, guardada em `RPM`, e a desativação do relé `RESET`. É necessário lembrar que a ordem dos operandos no bloco de multiplicação afeta o resultado. No caso, `K` é uma constante real e por isso o resultado da multiplicação será real. A constante `K` é calculada da seguinte forma:

$$\begin{aligned} K &= \frac{60}{K_P} \\ &= \frac{60}{128} \\ &= 0,46875 \end{aligned} \tag{5}$$

onde K_P que é o número de pulsos por rotação do *encoder*, e 60 é a transformação de segundo para minuto. Por exemplo, se o número de pulsos lidos, `PULSO`, for 3840,

tem-se que $RPM = 0,46875 * 3840$ e assim determina-se que o eixo está a 1800 rotações por minuto no sentido horário. Em outro exemplo, caso o número de pulsos lido seja de -2560, encontra-se que o eixo está a 1200 rotações por minuto no sentido anti-horário.

4.4.4 Filtros

No tratamento de sinais analógicos, muitas vezes o ambiente onde os sensores estão inseridos são propícios à disturbios e ruídos, fazendo com que o sinal lido sofra interferências e divirja do valor real. Nestes casos, podemos utilizar filtros passa-baixo ou passa-alto para permitir a passagem das frequências desejadas, e atenuar ou anular as outras.

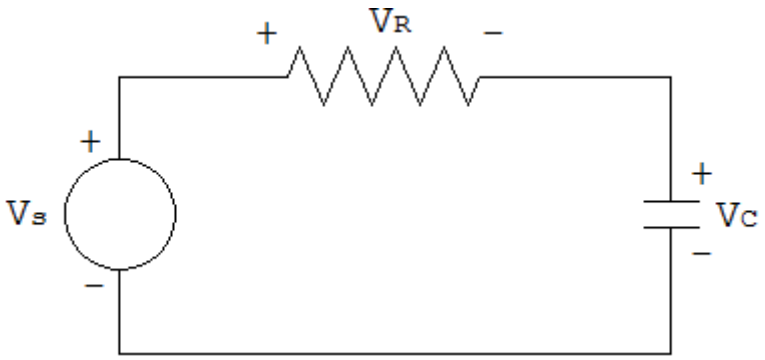


Figura 61: Circuito RC

Circuitos RC, como o da Figura 61, podem operar como filtro passa-baixo ou passa-alto. Para o filtro ser passa-baixo, o sinal de saída deve ser lido sobre o capacitor, então, pode-se deduzir a equação do filtro a partir da relação entre o sinal de saída, V_C, e o sinal de entrada, V_S:

$$\begin{aligned}
V_S(t) - V_C(t) &= R * i(t) \\
V_S(t) - V_C(t) &= R * C * \frac{dV_C(t)}{dt} \\
V_S(t) - V_C(t) &= R * C * \frac{V_C(t) - V_C(t - \Delta T)}{\Delta T} \\
V_C(t) * (1 + \frac{RC}{\Delta T}) &= V_S(t) + R * C * \frac{V_C(t - \Delta T)}{\Delta T} \\
V_C(t) &= V_S(t) * \frac{\Delta T}{\Delta T + RC} + V_C(t - \Delta T)\frac{RC}{\Delta T + RC}
\end{aligned}$$

Discretizando,

$$V_C(k) = V_S(k) * (1 - \alpha) + V_C(k - 1) * \alpha, \quad \alpha = \frac{RC}{\Delta T + RC} \tag{6}$$

Generalizando a equação (6), pode-se encontrar a equação geral de filtros passa-baixo:

$$V_{out}(k) = V_{in}(k) * (1 - \alpha) + V_{out}(k - 1) * \alpha, \quad \alpha = \frac{\tau}{\Delta T + \tau} \tag{7}$$

Sendo τ a constante de tempo, podemos calcular a frequência de corte do filtro pela equação:

$$f = \frac{1}{2\pi * \tau} \tag{8}$$

Resolvendo o circuito novamente, desta vez para o sinal de saída V_R, pode-se encontrar a equação geral de filtros passa-alto:

$$V_{out}(k) = [V_{in}(k) - V_{in}(k - 1)] * \alpha + V_{out}(k - 1) * \alpha, \quad \alpha = \frac{\tau}{\Delta T + \tau} \tag{9}$$

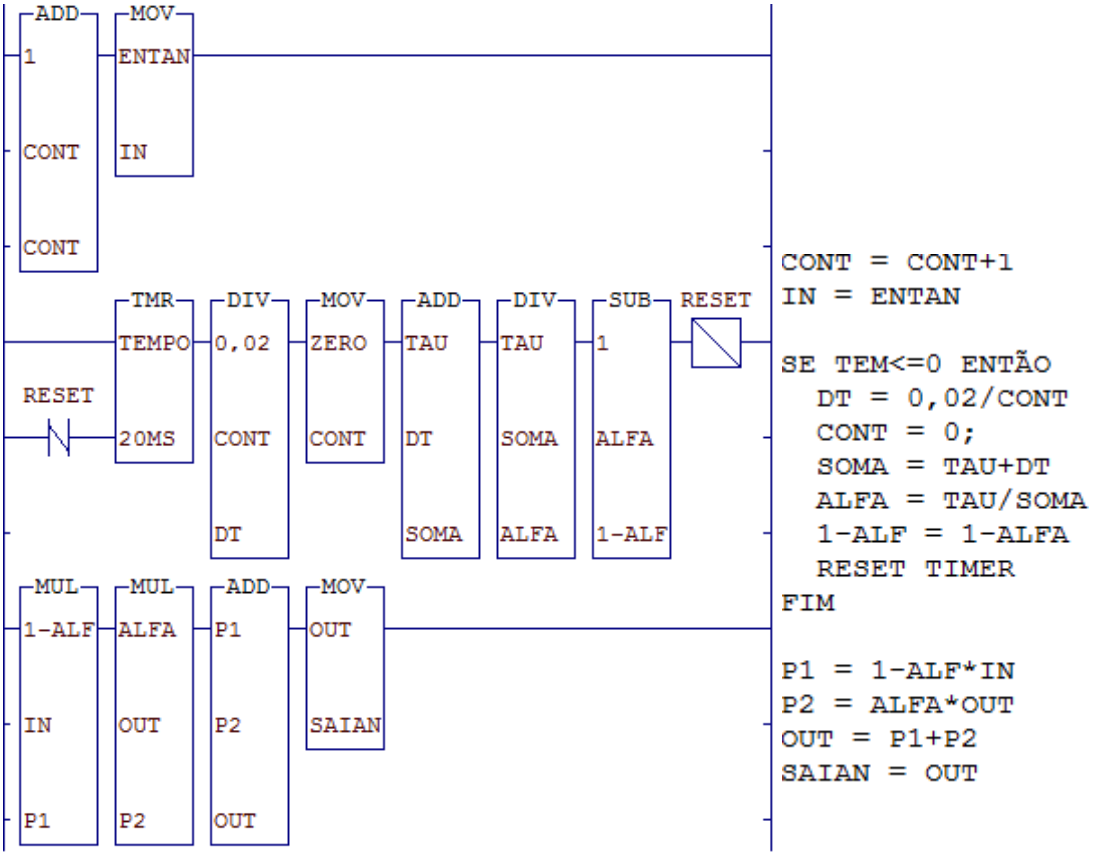
Assim, a dificuldade para a implementação em *Ladder* destes filtros se resume apenas a definir a variação de tempo, ΔT. Há duas maneiras de se definir a variação de tempo: utilizando-se blocos temporizadores ou estimando o tempo de varredura do sistema.

Como já foi visto, os blocos temporizadores tem o passo mínimo de 10ms (0,01s), o que equivale a uma frequência máxima de 100Hz. Então, segundo o teorema da amostragem, a frequência máxima para o sinal de entrada deve ser de 50Hz, que pode ser baixa, dependendo da aplicação.

O segundo método é a estimação do tempo de varredura do sistema, ou seja, quanto tempo é gasto para o CLP executar o programa novamente. Pode-se estimar o tempo de varredura de várias maneiras, inclusive, alguns editores *Ladder* dão a estimativa de quanto tempo é necessário para o programar ser executado. Este método tem a vantagem de conseguir trabalhar em maiores frequências mas peca na precisão do cálculo da variação de tempo. Na Figura 62 é mostrado a implementação *Ladder* deste segundo método para a implementação de um filtro passa-baixo, e o algoritmo utilizado.

A cada execução do programa, a variável `CONT` é incrementada e o valor da entrada analógica `ENTAN` é guardado na variável real `IN`. Quando o temporizador, que está programado para 20ms, ativar a saída, a variável `DT` guarda o valor da média do tempo gasto para o número `CONT` de execuções, é calculado o valor de α e $1 - \alpha$ baseado no novo valo de `DT`, e então o temporizador é reiniciado. A sétima linha é apenas a implementação da equação (7) do filtro passa-baixo. Após o cálculo do novo valor da variável real `OUT`, a saída analógica `SAIAN` recebe este valor. O valor de 20ms para o temporizador foi escolhido arbitrariamente. Este valor pode ser modificado de acordo com a necessidade, lembrando-se apenas de modificar o cálculo de `DT`.

Neste exemplo, o valor de `DT` é estimado com uma média simples de valores passados e não garante que as próximas execuções terão essa média de tempo. O cálculo pode ser



(a) Diagrama *Ladder* (b) Algoritmo utilizado

Figura 62: Implementação de um filtro passa-baixo com estimação do tempo de varredura

modificado para melhorar a estimativa, utilizando-se médias de valores passados de ΔT ou estimativa de valores futuros, por exemplo.

Alguns CLPs possuem filtros nativos em suas entradas e saídas analógicas que funcionam de variadas formas, de acordo com cada fabricante. *Ladder* possui também um bloco funcional que implementa um controlador PID, e sistemas de supervisão para a malha, tornando a implementação mais rápida e fácil. A seção 4.4.5 explica o funcionamento deste bloco.

4.4.5 Controladores PID

Controladores Proporcional-Integral-Derivativo, ou simplesmente PID, são uma combinação dos controladores proporcional, integral e derivativo. O objetivo desta junção é o melhor aproveitamento destas três técnicas de controle, a fim de otimizar a resposta de um processo. Na Figura 63 é ilustrado um diagrama de blocos de um controlador PID paralelo com realimentação.

Onde $r(t)$, $y(t)$, $e(t)$ e $u(t)$ são o sinal de referência, a saída do processo, o erro da saída, e a entrada do processo, respectivamente. A equação que representa o controlador

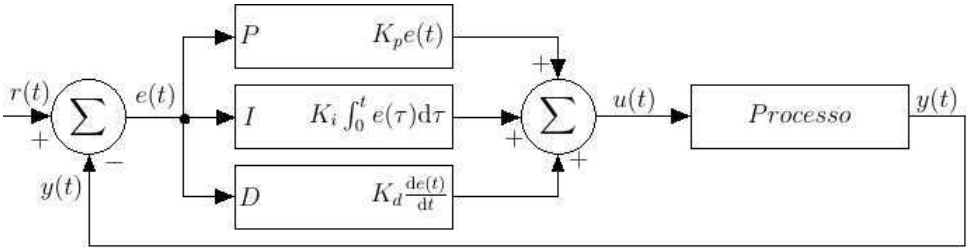


Figura 63: Diagrama de blocos de um controlador PID com realimentação

é dado abaixo:

$$u(t) = P(t) + I(t) + D(t) = K[\quad e(t) + \frac{1}{T_i}\int_0^t e(\tau)\mathrm{d}\tau + T_d\frac{\mathrm{d}e(t)}{\mathrm{d}t}\quad] \tag{10}$$

Para implementar em *Ladder*, deve-se antes dicretizar a equação. Discretizando por partes, tem-se a discretização de $P(t)$:

$$\begin{aligned} P(k) &= K[r(k) - y(k)] \\ P(k) &= Ke(k) \end{aligned} \tag{11}$$

Discretizando $I(t)$:

$$\begin{aligned} I(t) &= \frac{K}{T_i}\int_0^t e(\tau)\mathrm{d}\tau \\ \frac{\mathrm{d}I(t)}{\mathrm{d}t} &= \frac{K}{T_i}e(t) \\ \frac{I(k) - I(k-1)}{\Delta T} &= \frac{K}{T_i}e(k) \\ I(k) &= I(k-1) + \frac{K\Delta T}{T_i}e(k) \end{aligned} \tag{12}$$

E para $D(t)$:

$$\begin{aligned} D(t) &= KT_d\frac{\mathrm{d}e(t)}{\mathrm{d}t} \\ D(k) &= \frac{KT_d}{\Delta T}[e(k) - e(k-1)] \end{aligned} \tag{13}$$

Ao fim, temos a equação (10) discretizada[10] na forma:

[10]As discretizações aqui apresentadas podem ser realizadas de outras formas, como pela aproximação de Tustin, por exemplo. Deve-se utilizar o método que melhor responde às necessida-

$$u(k) = Ke(k) + [I(k-1) + \frac{K\Delta T}{T_i}e(k)] + \frac{KT_d}{\Delta T}[e(k) - e(k-1)] \quad (14)$$

A implementação desta equação em *Ladder* pode ser realizada de duas maneiras: implementação manual da equação, ou a utilização do bloco funcional PID.

Para a implementação manual, deve-se calcular toda a equação (14) a partir de blocos funcionais matemáticos e de movimentação. Para tal, pode-se utilizar os conhecimentos adquiridos das seções 4.4.1.2 e 4.4.4. A desvantagem desse método é o tempo gasto para a implementação e a falta de um sistema supervisório para acompanhar a evolução do sistema.

O método alternativo, e mais fácil, é a utilização do bloco funcional PID, que implementa a equação (14) automaticamente. Na Figura 64 é mostrado a representação *Ladder* do bloco PID.

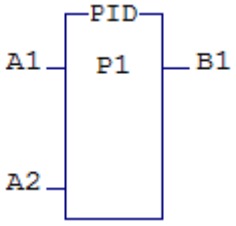


Figura 64: Representação *Ladder* do bloco funcional PID

Este bloco funcional tem apenas um parâmetro do tipo `T` que é o identificador do bloco. As entradas `A1` e `A2` habilitam ou desbilitam o funcionamento do bloco. A saída `B1` é ativada quando o bloco funcional está operacional.

Para a programação do bloco PID, cada editor *Ladder* implementa uma forma de edição diferente, em janelas específicas. Apesar disto, o bloco PID têm sempre os mesmos parâmetros. São estes:

- `PV` (*Process Variable*): Posição na memória onde é guardado a variável do processo, $y(t)$;
- `SP` (*Set Point*): Posição na memória onde é guardado a entrada de referência, $r(t)$;
- `OV` (*Output Variable*): Posição na memória onde é guardado a saída do controlador, $u(t)$;
- `GP`: Posição na memória onde é definido o ganho proporcional, K;
- `TI`: Posição na memória onde é definido o tempo integral, T_i;

des de controle.

- `TD`: Posição na memória onde é definido o tempo derivativo, T_d;
- `OV Máx`: Valor máximo que pode ser assumido por `OV`, em %;
- `OV Min`: Valor mínimo que pode ser assumido por `OV`, em %;
- `OV Ini`: Valor inicial de `OV`, em %.

É importante notar que as entradas e saídas, `PV`, `SP` e `OV`, deste bloco controlador não estão em unidades de engenharia. São programados de acordo com a resolução de n *bits* das portas analógicas, geralmente $n = 12$ *bits*, ou seja, estas variáveis assumem apenas os valores inteiros de 0 a $2^n - 1$. A conversão destes valores para unidades de engenharia deve ser feita externamente ao bloco.

Alguns editores *Ladder* também incluem sistemas supervisórios para os blocos PID. Assim, é possível acompanhar graficamente a resposta do sistema ao controlador, facilitando sua sintonização.

Este método é somente utilizado quando há o bloco funcional PID implementado no CLP, o que depende de cada fabricante. Para os CLPs que não possuem este bloco, o controlador PID deve ser implementado manualmente.

5 Guias para experimentos

Este capítulo contém guias com exercícios de fixação de aprendizagem da linguagem *Ladder* e experimentos propostos a serem realizados utilizando o kit de treinamento eZTK900.

5.1 Guia Experimental: Chaves e Relés

Objetivo

O objetivo deste guia é familiarizar o aluno com o ambiente de programação SPDSW a partir da programação de funções simples como a implementação de portas lógicas e circuitos de selo.

Para a realiazação deste experimento, é necessário a leitura prévia da seções 3 e 4.2.1 da apostila.

Introdução

Chaves e relés são os elementos básicos da programação *Ladder*. Pode-se dizer que estes elementos, entre outros, foram "importados" dos diagramas elétricos, já que a principal inspiração do *design* de *Ladder* foram os diagramas elétricos, e que o funcionamento destes elementos refletem o comportamento físico de seus similares.

Como em um diagrama elétrico, *Ladder* possui duas barras verticais que se equivalem a uma fonte de alimentação, como as duas barras horizontais em diagramas elétricos. A barra à esquerda, é a "barra de alimentação", e a barra à direita é a "barra de terra". Na figura 65 são indicadas as barras de alimentação do editor SPDSW.

A seguir, será descrito o mais simples programa em *Ladder* , que apenas liga uma saída a partir de uma entrada.

Com o programa SPDSW configurado[11] e o editor *Ladder* aberto, seleciona-se a célula no editor que irá receber a chave. Neste exemplo, será a célula [1,1] (coluna 1, linha 1). Após isto, seleciona-se a aba *Básico* à esquerda e logo abaixo, aparecerá diversos ícones de blocos funcionais. Clica-se no terceiro ícone, *Chave NA*, e a chave normalmente aberta aparecerá na área de edição na célula selecionada, como é mostrado na figura 66.

Agora, com a chave inserida no editor, deve-se dizer o que esta chave está representando. Para tal, deve-se editar o bloco selecionando-o e em seguida clicando no botão *Edita operando selecionado* (4º botão da barra de ferramentas), ou apenas pressionando **F6**. Assim, digita-se o valor `I0000` e pressiona-se *Enter*. O valor `I0000` significa que o

[11] Para se começar a programar em *Ladder* , primeiro é necessário criar um novo projeto e configurá-lo de acordo com os conhecimentos do apêndice 3.

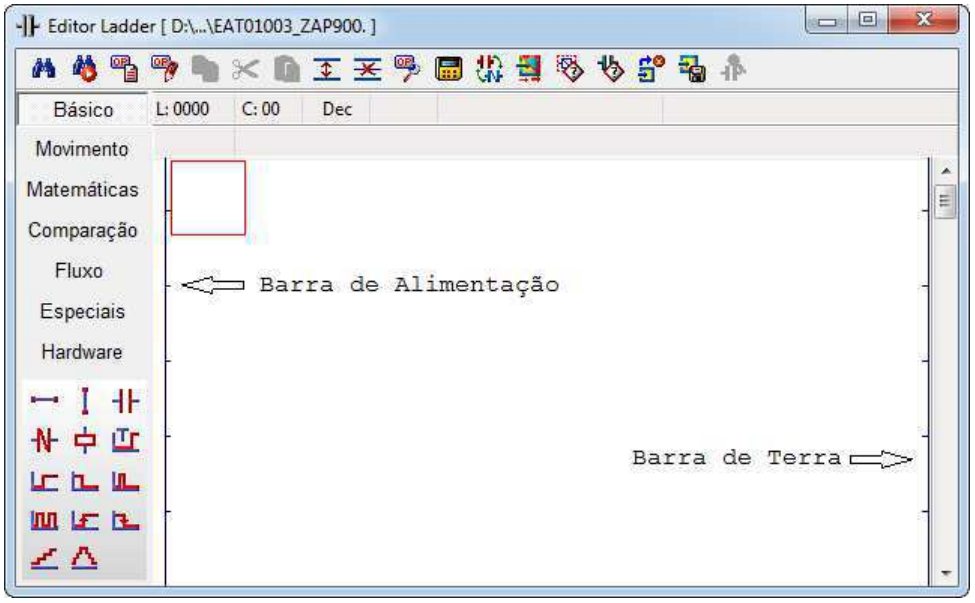


Figura 65: Visualização das barras de alimentação e terra do SPDSW.

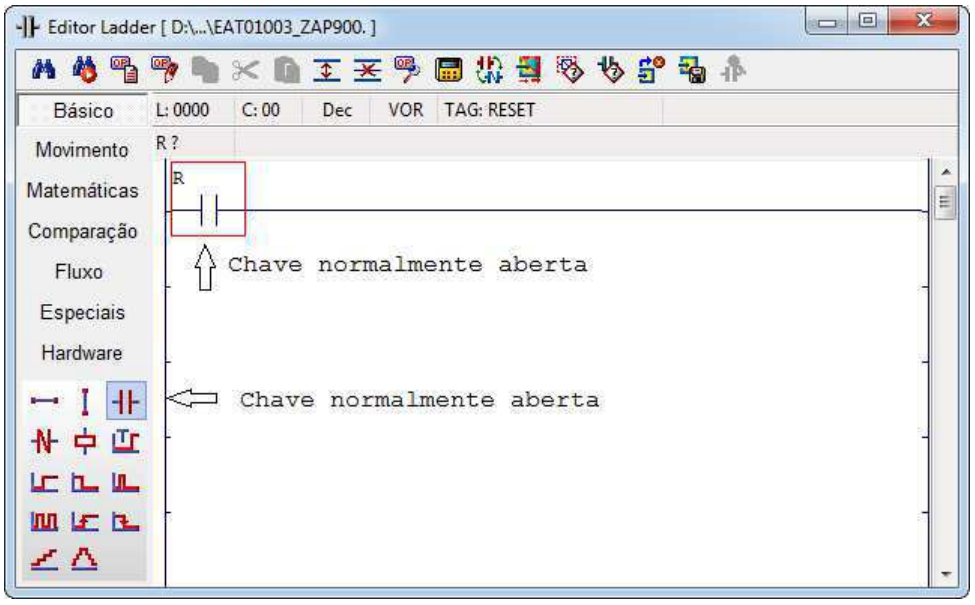


Figura 66: Chave normalmente aberta colocada na área de edição.

programa relaciona o estado desta chave virtual com o estado do pino de entrada digital `I0000` do CLP.

No kit de treinamento eZTK9000, a chave frontal `I0000` é ligada ao pino de entrada digital `I0000` do CLP. Isso significa que, se a chave frontal do kit for ligada, a chave virtual também é ligada, e se a chave frontal do kit for desligada, a chave virtual é desligada.

Com a chave configurada, será colocado agora o relé que será ativado pela chave (veja figura 67). Primeiro, seleciona-se a célula [2,1] e, na aba *Básico*, clica-se no quinto ícone,

Relé. Nota-se aqui que o relé não foi posto na célula [2,1], mas sim na [10,1]. Isto é um comportamento normal do SPDSW, que permite que relés sejam inseridos apenas na última coluna, pois, o SPDSW não permite que dois relés sejam colocados na mesma linha. Seguindo os mesmos passos realizados para a chave, edita-se o relé para o valor `O0000`. Este valor indica que o programa relaciona este relé virtual com o estado do pino de saída digital `O0000` do CLP.

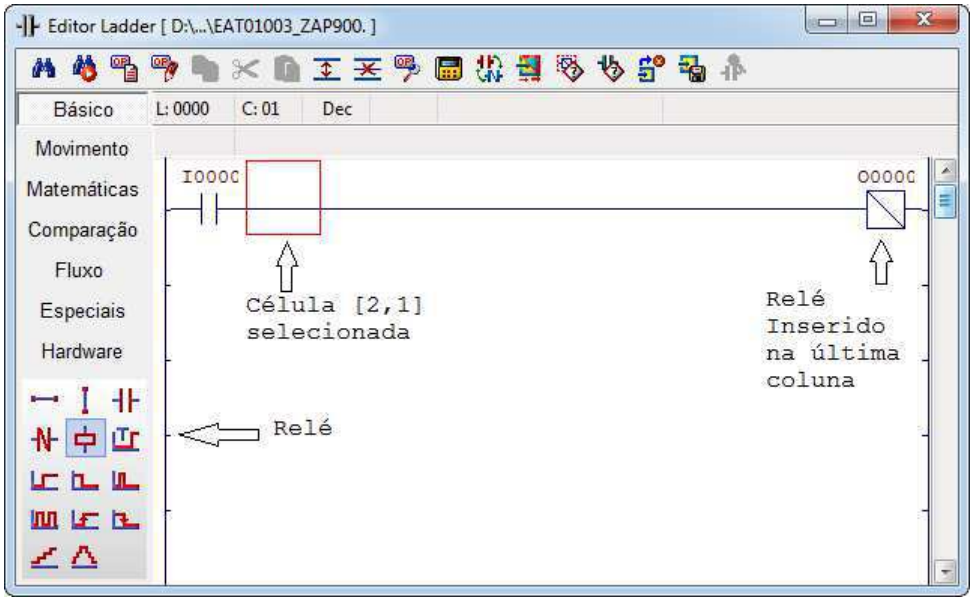


Figura 67: Relé inserido no diagrama e já editado.

Para determinar o fim do programa, deve-se indicar ao SPDSW que o diagrama já foi concluído. Isso é feito utilizando o bloco `END`, que é o último botão da aba *Fluxo*, *Fim de Programa*. Assim, seleciona-se a segunda linha do diagrama, não importa qual coluna, e se insere este bloco. Na figura 68 é mostrado o diagrama completo, com o bloco `END` já inserido.

Agora pode-se testar o programa clicando-se no botão *Compila/Carrega/Depura* (13º botão da barra de ferramentas) ou apenas pressionando **Ctrl+Z**. Assim, pode-se ver que este programa tem a simples função de ligar o LED `O0000` do kit de treinamento eZTK900 a partir da chave `I0000`.

A programação em *Ladder* também permite, como nos diagramas elétricos, que vários relés sejam energizados pela mesma entrada. Para tal, será editado o programa anterior para incluir esta função.

Primeiramente, será incluída uma nova linha no programa já elaborado. Para isso, seleciona-se a segunda linha (a linha onde está o bloco `END`) e clica-se em *Insere Linha* (8º botão da barra de ferramentas) ou pressiona-se **Ctrl+Ins**. Esta ação incluirá uma linha entre a primeira e segunda linha, como mostrado na figura 69.

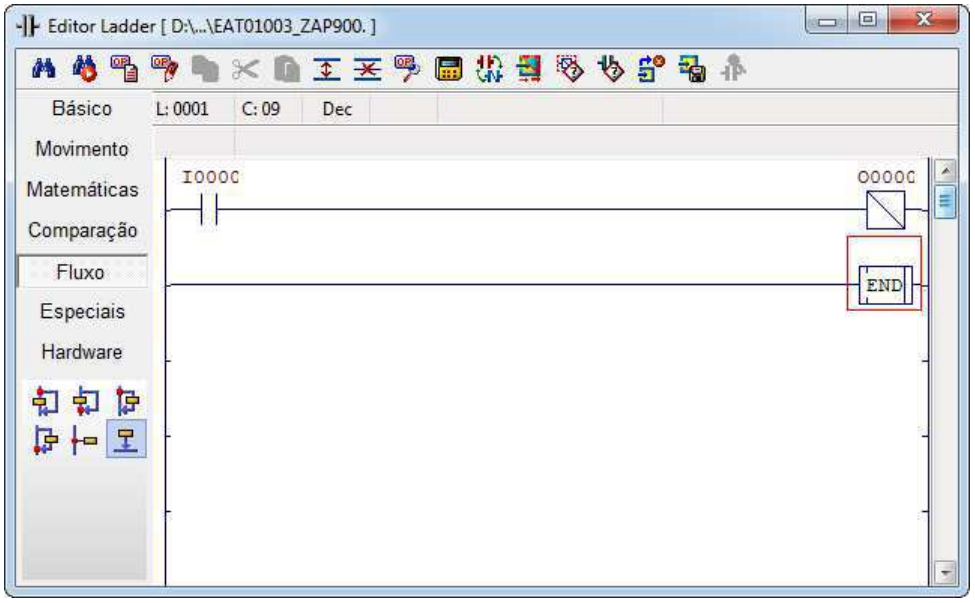


Figura 68: Diagrama completo com o bloco `END` inserido.

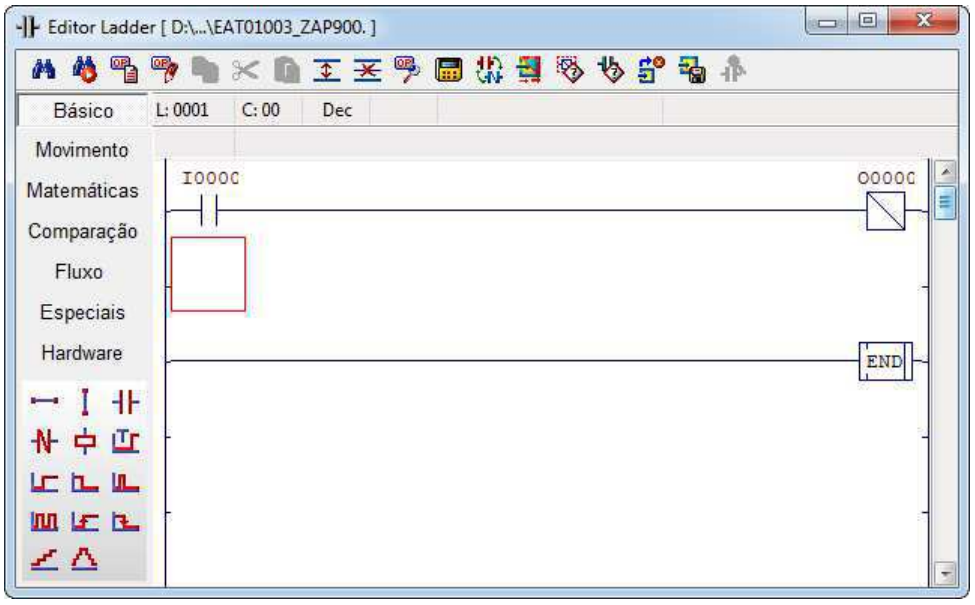


Figura 69: Diagrama com uma linha a mais inserida.

Na nova linha, uma chave na primeira coluna com o valor `I0000` é inserida igual à chave da primeira linha, e um relé de valor `O0001`. Na figura 70 é mostrado o diagrama com a nova linha inserida.

Testando este novo programa, percebe-se que utilizando apenas uma chave, `I0000`, duas saídas são ativadas, `O0000` e `O0001`.

Utilizando chaves e relés, pode-se implementar todas as lógicas que são possíveis com a lógica de relés, como a lógica booleana. Na figura 71 é mostrado a implementação de

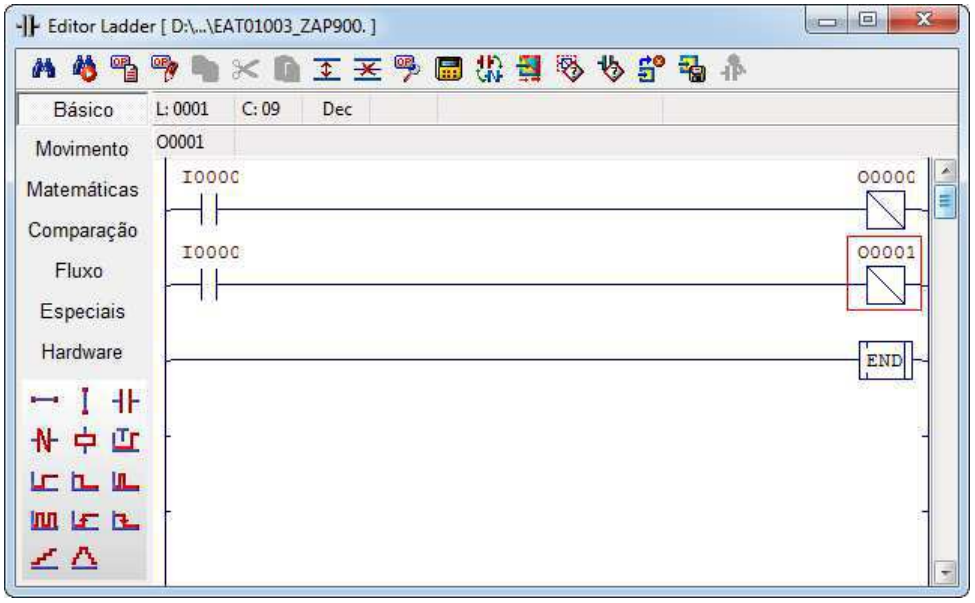


Figura 70: Diagrama com uma chave ligando dois relés ao mesmo tempo.

uma porta lógica *AND*, utilizando duas chaves em série para ativar uma saída.

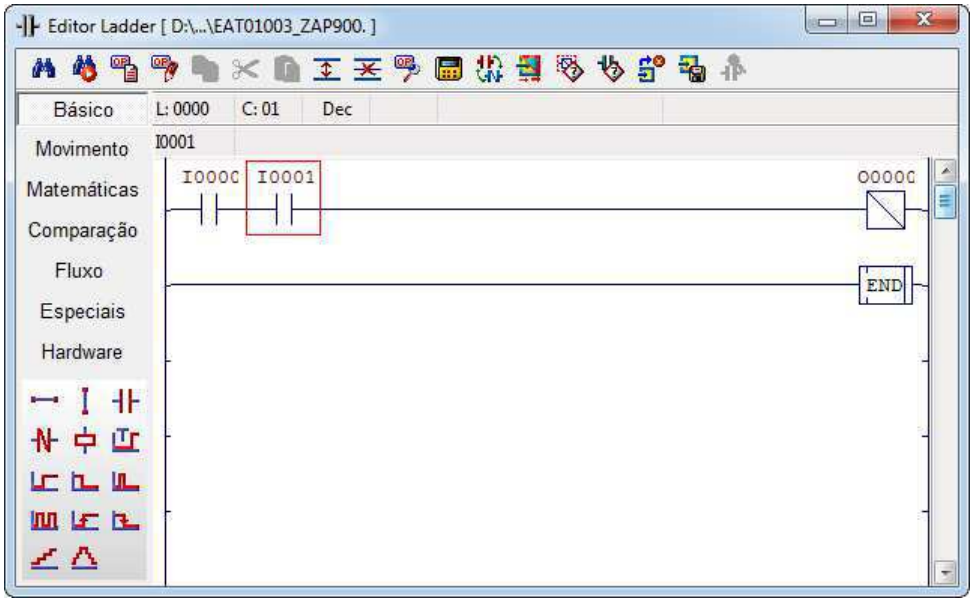


Figura 71: Implementação de uma porta lógica *AND* em *Ladder*

Assim como a porta lógica, o relé do diagrama só será energizado quando as duas entradas forem ligadas, caso contrário, estará desenergizada. Mais informações sobre funções lógicas em *Ladder* podem ser encontradas na seção 4.2.1.

Em diagramas elétricos, pode-se haver a necessidade de energizar um relé com uma lógica dependente do estado de outro relé, ligando a saída do segundo relé na lógica do

primeiro. *Ladder* permite implementar este tipo de situação, trocando-se na chave a entrada pela saída. O diagrama da figura 72 ilustra este caso.

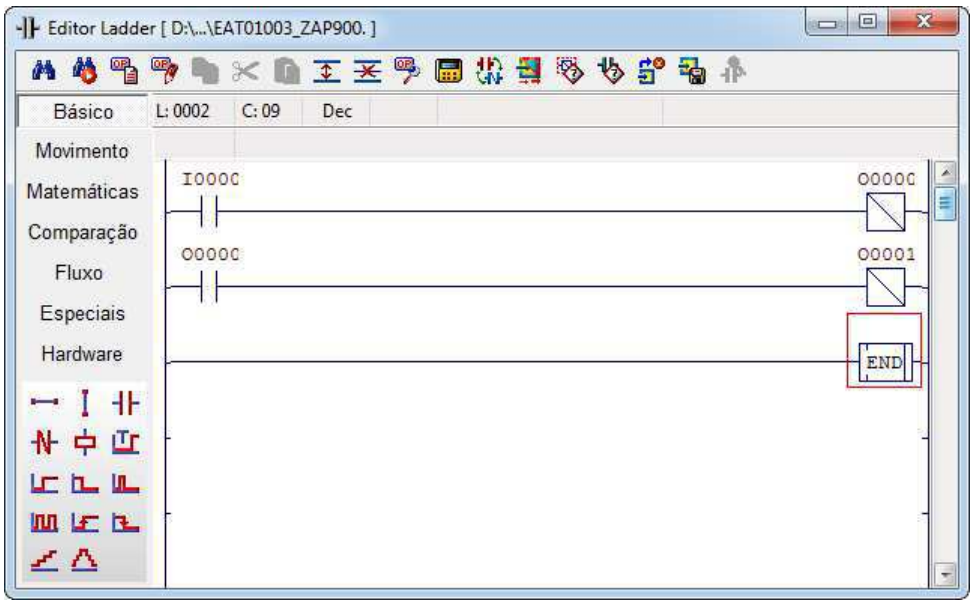


Figura 72: Diagrama *Ladder* de um relé dependente do estado de outro

Quando a chave `I0000` é ligada, o relé `O0000` é energizado. Por consequência, a chave `O0000` também é ligada e, assim, o relé `O0001` é energizado.

Exercícios de Preparação

1. Projete diagramas *Ladder* para executar as equações lógicas seguintes e descreva a tabela verdade de cada uma.

   a) $Y_1 = A + B + \overline{C}$

   b) $Y_2 = A * B + \overline{A} * C$

   c) $Y_3 = \overline{A} * (B + Y_3)$

   d) $Y_4 = B + \overline{A} * Y_4$

2. A função lógica *XNOR* (*NOT eXclusive OR*) é a negação da função *XOR*. Também chamada de função coincidência, sua saída é verdadeira somente se as duas entradas forem iguais e é representada da forma $Z = A \odot B = A * B + \overline{A} * \overline{B}$. Implemente esta função em *Ladder* e descreva a tabela verdade.

3. Um certo motor deve ser ligado apenas quando duas chaves são ativadas ao mesmo tempo. Implemente o diagrama ladder deste circuito. Depois, utilizando mais uma chave, implemente um selo com prioridade no desligamento para o mesmo motor.

4. Implemente em ladder o diagrama elétrico da figura 73, onde Sx são entradas e Kx saídas.

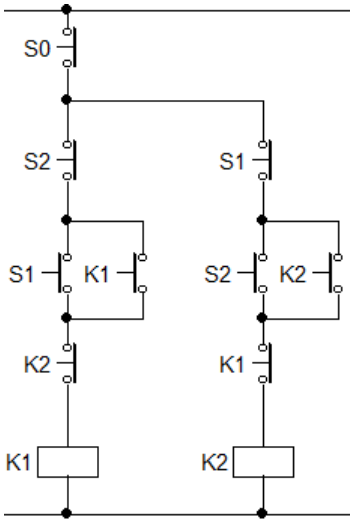


Figura 73: Diagrama elétrico de uma partida com reversão

5. Um tanque deve ter seu volume mantido numa certa faixa. O controle de volume do tanque é feito a partir de duas válvulas e dois sensores de nível, como mostrado na figura 74. As válvulas são controladas pelas saídas digitais `Q0000` e `Q0001`, que determinam se a válvula será aberta ou fechada. Os sensores de nível 1 e 2 são ligados nas entradas digitais `I0000` e `I0001`, respectivamente, e detectam se o nível do tanque está acima deles. Quando o nível do tanque estiver acima de um sensor de nível, este sensor é ativado. Com base nesta informações, projete o controle do tanque de modo que:

   - A válvula 1 é aberta quando o nível do tanque estiver menor que o nível do sensor 1, e fechada quando o nível estiver maior que o nível do sensor 2;
   - A válvula 2 é aberta quando o nível do tanque estiver maior que o nível 1; fechada caso contrário.

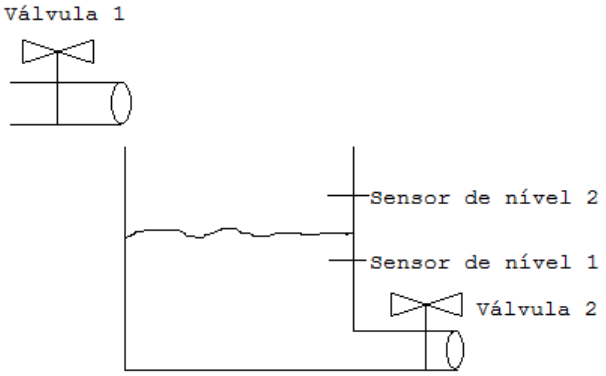


Figura 74: Tanque controlado por duas válvulas.

Prática Experimental

1. Implemente o diagrama *Ladder* da figura 72 e teste seu funcionamento.

2. Implemente duas das funções lógicas do exercício 1 em *Ladder*. Compare o resultado de cada função com sua correspondente tabela verdade.

3. Implemente a função lógica *XNOR* em *Ladder*. Compare o resultado com a tabela verdade da porta lógica *XOR*.

4. Implemente o resultado obtido do exercício 3. Depois, modifique o diagrama para um selo com prioridade no ligamento. Compare o resultados experimentais obtidos dos dois diagramas.

5. Implemente o resultado obtido do exercício 4. Compare o diagrama obtido com o diagrama elétrico. Quais as semelhanças e diferenças?

6. Implemente o resultado do exercício 5 e descreva a lógica das válvulas em álgebra de Boole. Compare o resultado obtido com o requerido da questão. O que acontece quando o nível do líquido está maior que o nível 1 e menor que o nível 2?

5.2 Guia Experimental: Blocos Flip-flop e Contadores

Objetivo

O objetivo deste guia é permitir ao aluno adquirir prática experimental com blocos flip-flops *SET* e *RESET*, blocos osciladores e blocos contadores.

Para este experimento, é necessário a leitura prévia das seções 4.2.2, 4.3.1 e 4.3.2 da apostila.

Introdução

Os blocos *SET* e *RESET* em conjunto desempenham a mesma função de um flip-flop do tipo S-R (do inglês, *Set and Reset*). Na seção 4.2.1.3 mostrou-se como é possivel manter o estado da saída ativo, mesmo quando a entrada é desativada através de circuitos de selo. Porém, pode-se chegar ao mesmo resultado utilizando os blocos funcionais do tipo *SET* e *RESET*, como será mostrado a seguir.

Para a programação de um selo com prioridade no desligamento, deve-se primeiramente configurar o programa SPDSW e abrir o editor *Ladder*.

Os blocos *SET* e *RESET* são encontrados na aba *Básico*, sendo o sexto e sétimo botões, respectivamente (veja figura 75). Assim, será programado um selo com prioridade no desligamento a partir das chaves `I0000` e `I0001`.

No digrama da figura 75, nota-se primeiramente que os dois blocos, *SET* e *RESET*, possuem o mesmo atributo, `O0000`. Assim, quando a chave `I0001` é ativada, a saída `O0000` é ativada ("setada") e permanecerá neste estado, mesmo se `I0001` for desativado. A saída será desativada somente se a chave `I0000` for ativada, ativando o bloco *RESET* e desativando ("resetando") a saída `O0000`. Este diagrama é equivalente a um selo com prioridade no desligamento porque, se as duas chaves estiverem ativas ao mesmo tempo, o bloco *RESET* sempre será o último a ser executado pelo CLP, mantendo a saída `O0000` desativada.

Para implementar um selo com prioridade no ligamento o diagrama é bastante similar, trocando-se apenas a ordem das linhas como mostrado na figura 76.

No caso do selo com prioridade no ligamento, o último bloco a ser executado pelo CLP será o bloco *SET* então, caso as duas entradas estiverem ativas ao mesmo tempo, a saída `O0000` sempre estará ativa.

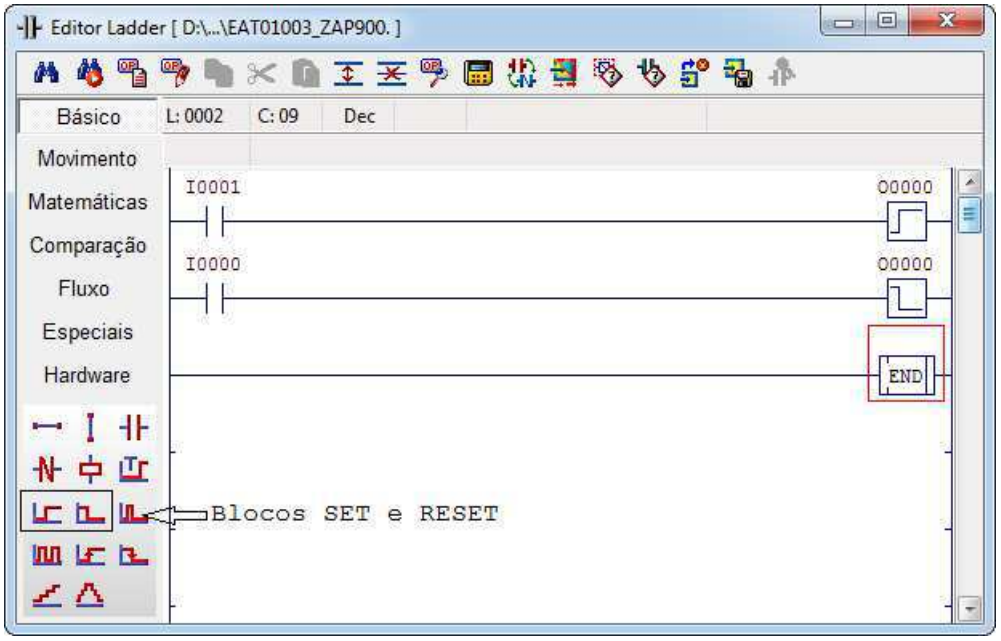


Figura 75: Diagrama *Ladder* de um selo com prioridade no desligamento feito com blocos *SET* e *RESET*.

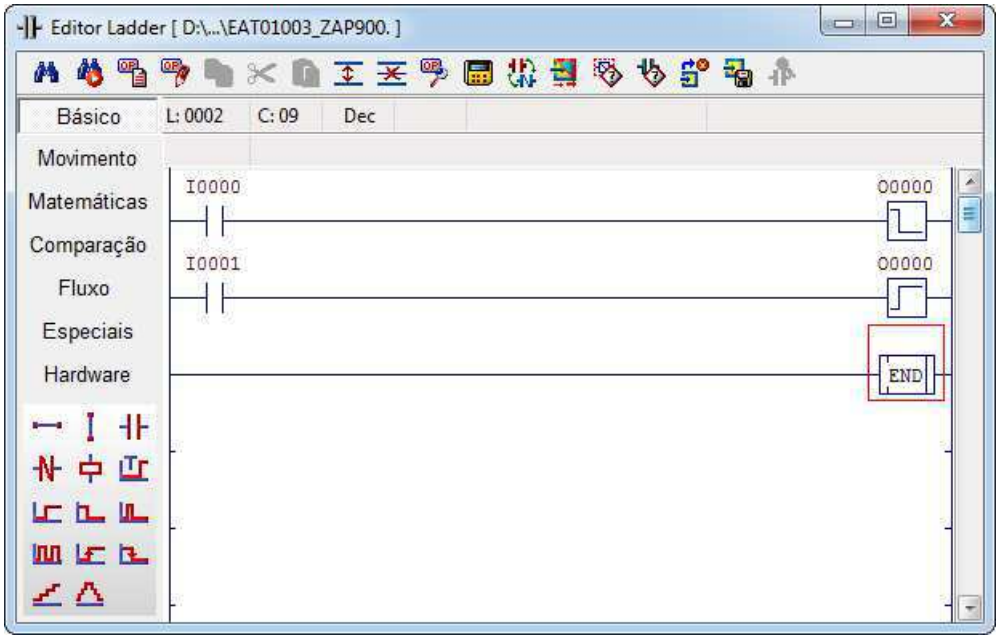


Figura 76: Diagrama *Ladder* de um selo com prioridade no ligamento feito com blocos *SET* e *RESET*.

Outro bloco flip-flop definido em *Ladder* é o bloco oscilador. Este bloco é equivalente a um flip-flop do T (do inglês, *Toggle*).

Em *Ladder* também é definido o bloco funcional oscilador. Este bloco funciona da mesma maneira que um flip-flop do tipo T (do inglês, *Toggle*, que significa Alternar): quando o sinal de entrada varia de inativo para ativo, a saída é invertida. Ou seja, caso a

saída esteja inativa, ela tornar-se-á ativa, e se estiver ativa ela será desativada. Com este flip-flop é possível, por exemplo, fazer um contador binário como mostrado na figura 77.

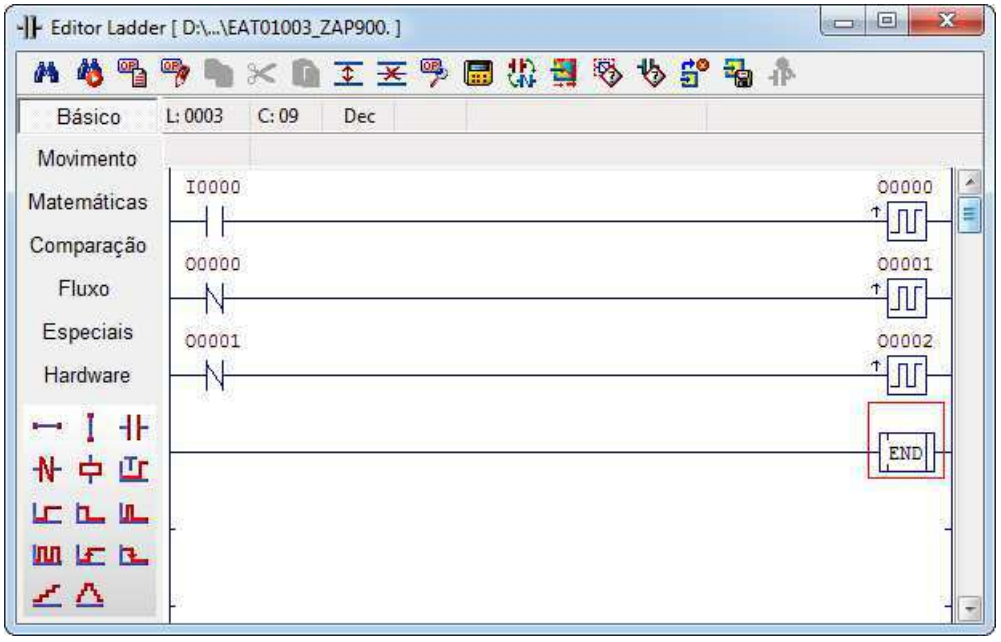


Figura 77: Diagrama *Ladder* de um contador de 3 *bits* composto de blocos osciladores.

Considerando a entrada `I0000` sendo uma série de pulsos, as saídas `O0000`, `O0001` e `O0002` registrarão a contagem destes pulsos, sendo `O0002` o *bit* mais significativo.

Mas, apesar de contadores feitos de blocos osciladores serem de simples implementação, este tipo de contador não é muito utilizado pois, *Ladder* define blocos funcionais contadores que são mais versáteis e simples de utilizar. O bloco contador possui dois parâmetros: o primeiro parâmetro (`P1`) é a contagem total de pulsos recebidos na entrada `A1`, e o segundo parâmetro (`P2`) é o valor máximo que é contado. Este bloco é encontrado na aba *Básico* (13º elemento) com o nome de *Contador UP*. Na figura 78 é mostrado o bloco contador, com uma chave na entrada `A1` para contar e outra chave na entrada `A2` para reiniciar a contagem, inserido no editor com os parâmetros `M0000` e `K0000`.

No diagrama da figura 78, quando a chave `I0000` muda de inativa para ativa, a variável `M0000` é incrementada de hum. Isto se repete até que `M0000=K0000`, onde `M0000` não é mais incrementado independentemente de `I0000`. Quando a contagem chega ao limite, a saída do contador é ativada, indicando o término da contagem e ativando a saída `O0000`. A contagem só é reiniciada quando a entrada `I0001` é ativada, o que faz com que a chave dependente desta seja aberta, reiniciando a contagem.

Ainda para o mesmo diagrama da figura 78, deve-se configurar o valor da constante inteira `K0000`. Para tal, seleciona-se a célula na qual está `K0000` e clica-se em *Lista identificadores*, ou **F7**, que abrirá uma janela onde pode-se atribuir uma *TAG* (mnemônico),

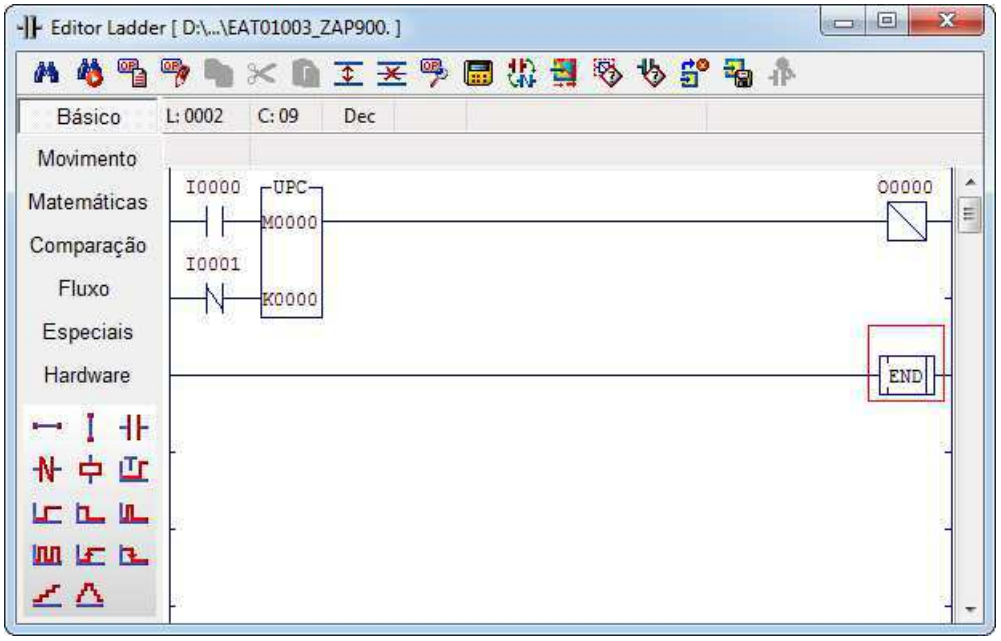


Figura 78: Diagrama *Ladder* de um bloco contador controlador por duas chaves.

uma descrição e um valor. Na figura 79 é mostrado esta janela com o K0000 igual a 7.

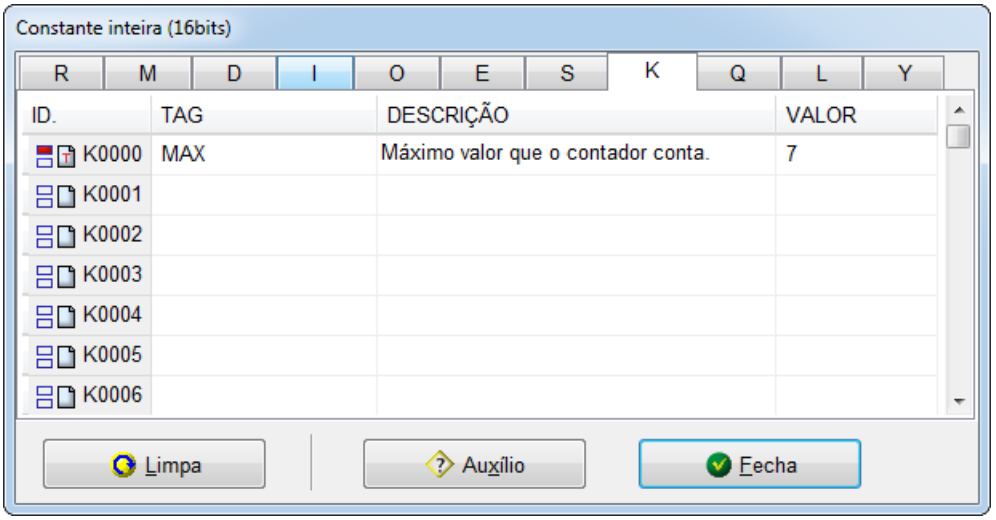


Figura 79: Janela para edição dos identificadores de cada variável.

Nesta janela da figura 79, é possível editar todas as variáveis do programa. Por exemplo, para editar M0000, clica-se na aba superior *M* e edita-se a entrada *M0000*.

Para visualizar as *TAG's* ao invés dos números de identificação das variáveis, pressiona-se **F2** no editor. Na figura 80 são mostradas as variáveis como *TAG's* no blocos.

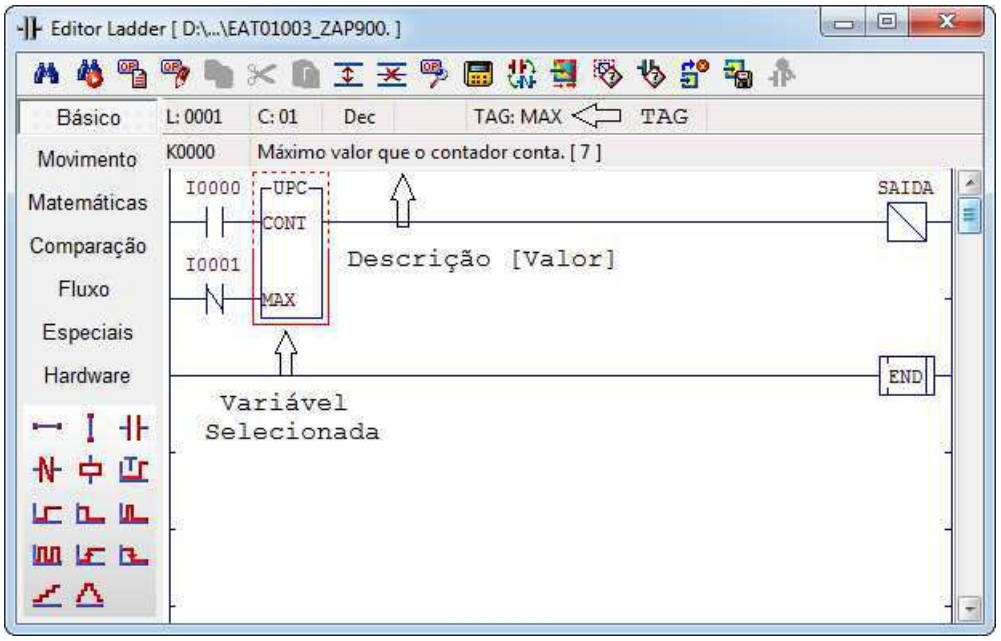


Figura 80: Editor *Ladder* com *TAG's* a mostra.

Exercícios de Preparação

1. Elabore diagramas *Ladder* para o selo com prioridade no desligamento e para o selo com prioridade no ligamento utilizando os blocos *SET* e *RESET*.

2. Para o contador composto de blocos osciladores no diagrama da figura 77, elabore um modo de reiniciar a contagem utilizando blocos *RESET*.

3. Utilizando blocos *RESET*, modifique o diagrama da figura 77 para que contador conte apenas até 5 e então reinicie a contagem.

4. Em uma linha de produção, uma esteira possui um sensor que é ativado quando um objeto cruza certo ponto. Após 10 objetos cruzarem esta esteira, o sistema de controle deve parar a esteira e enviar um sinal para que seja iniciado o processo de empacotamento. A empacotadeira, por sua vez, envia um sinal de volta ao sistema de controle, indicando que o processo de empacotamento foi finalizado e que a esteira pode voltar a funcionar. Considere que `I0000` é o sinal recebido do sensor, `O0000` controla o movimento da esteira, e `I0001` e `O0001` são os sinais recebidos e enviados à empacotadeira, respectivamente. Com base nesta descrição, elabore um diagrama *Ladder* para o sistema de controle.

5. Considerando a mesma esteira do exercicio 4, os objetos que passam sobre ela são de dois tipos: A e B. Um sensor identifica qual o tipo do objeto que está passando pela esteira e envia dois sinais para um segundo sistema de controle: o primeiro sinal indica que há um objeto passando; o segundo sinal indica o tipo, A ou B. É sabido que a ordem de passagem dos objetos deve ser sempre alternada: primeiro A depois B depois A depois B e assim por diante. Deste modo, o sistema de controle verifica a ordem dos objetos e caso a ordem esteja incorreta, envia um sinal de erro que permanece ligado. Considere que os sinais do sensor estão ligados aos pinos `I0000` e `I0001` do CLP, e que o sinal de erro é ligado ao pino `O0000`. Projete este sistema de controle em *Ladder* utilizando um contador bidirecional.

Prática Experimental

1. Implemente em *Ladder* os resultados do exercício 1. Há diferença entres os dois diagramas? E em relação aos selos feitos com relés?

2. Implemente o resultado dos exercícios 2 e 3. Qual a diferença entre os dois diagramas?

3. Implemente os resultados obtidos do exercício 4. Verifique se o diagrama atende às necessidades de controle.

4. Implemente os resultados obtidos do exercício 5 e verifique se atende às necessidades de controle. Há algum cenário possível em que o sistema falhe?

5.3 Guia Experimental: Temporizadores

Objetivo

O objetivo deste guia é familiarizar o aluno com os blocos temporizadores e suas aplicações mais comuns.

Para a realiazação deste experimento, é necessário a leitura prévia da seção 4.3.3 da apostila.

Introdução

O funcionamento do bloco temporizador é similar ao bloco contador. O parâmetro `P1` é o tempo atual da contagem, significando quanto tempo falta para o término, e deve ser do tipo inteiro(`M`). O parâmetro `P2` é o tempo inicial da contagem e deve ser um inteiro variável ou constante (`K`). A entrada `A1` habilita ou pausa a contagem, e a entrada `A2` habilita ou reinicia a contagem.

Deve-se lembrar que os blocos temporizadores contam com um passo de 10ms(ou 0,01s), ou seja, o tempo requerido deve ser multiplicado por 100 para que o temporizador funcione adequadamente. Por exemplo, caso queira-se temporizar 2 segundos, `P2` deve ser igual 200.

Assim como o bloco contador, a saída do bloco temporizador se mantém ativa após a temporização, até que seja reiniciada. Na figura 81 é mostrado o diagrama de um temporizador que ativa a saída `O0000` após a temporização e é reiniciado pela chave `I0001`. Considere `K0000=100`.

No diagrama da figura 81, o temporizador está configurado para temporizar 1 segundo, com `K0000=100`, e quando este tempo acaba, o relé `O0000` é ativado indefinidamente, até que a chave `I0001` seja ativada. Quando a chave é ativada, `M0000` recebe o valor de `K0000`, e quando é desativada, `M0000` começa a ser decrementado a cada 10ms, até ser igual a zero e ativar o relé `O0000` novamente.

Ligado à entrada de um contador, o bloco temporizador pode servir como um multiplicador de tempo. Na figura 82 é ilustrado este caso.

Considere no diagrama da figura 82, que `K000=50` ou seja, o temporizador é de 0,5s. Assim, o relé auxiliar `R0000`, que é ativado quando a temporização está completa, reinicia o temporizador e dá um pulso no oscilador `O0001`, fazendo-o alternar de valor. Para `O0002`,

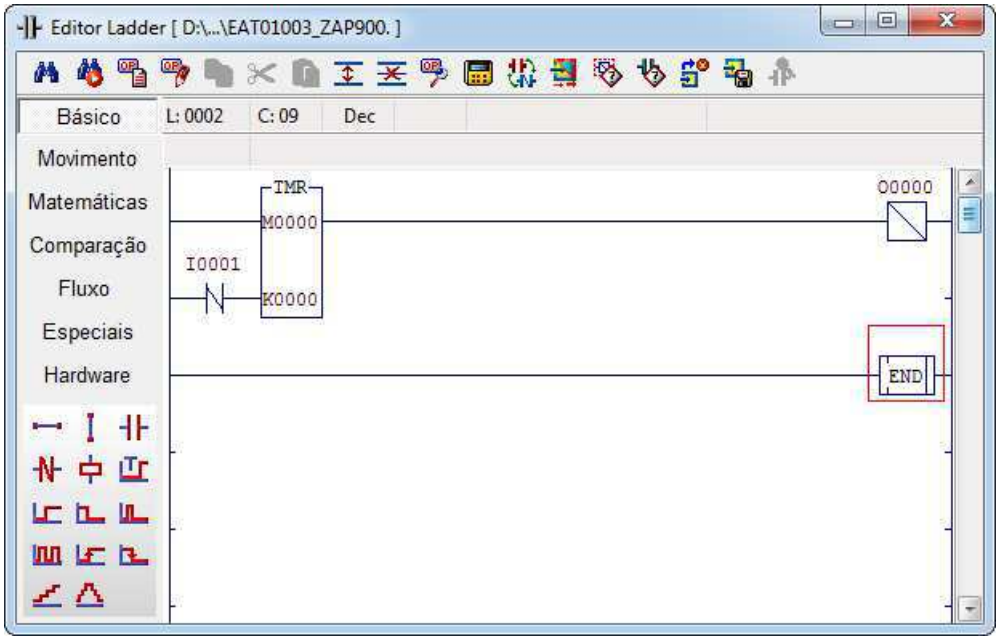


Figura 81: Diagrama *Ladder* de um bloco temporizador reiniciado pela chave `I0000`.

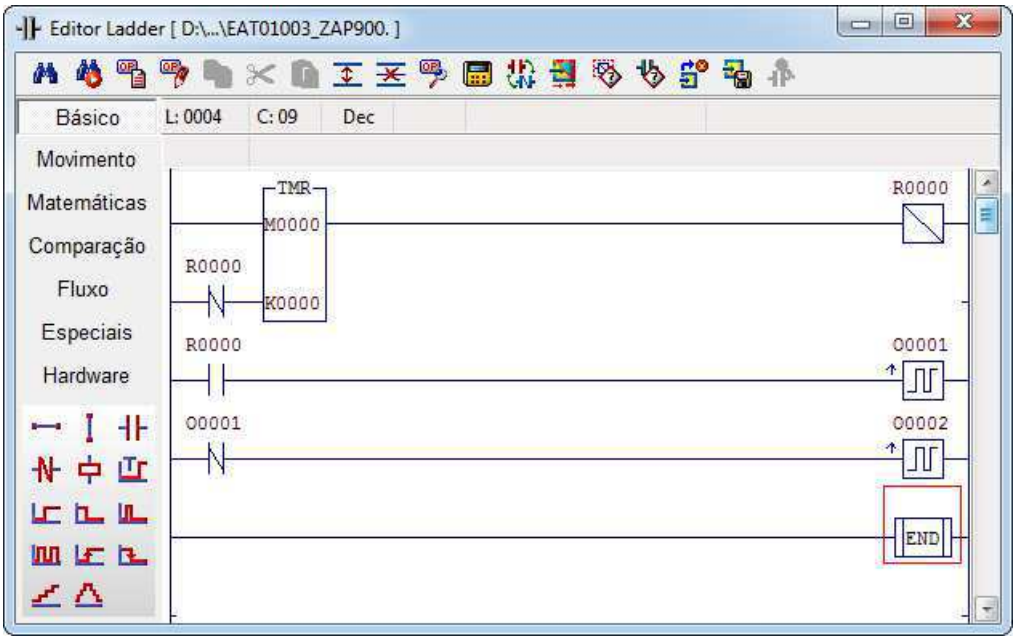


Figura 82: Diagrama *Ladder* de um bloco temporizador ligado à entrada de um contador composto de osciladores.

este alterna de valor quando `O0001` é desativado no próximo ciclo de temporização.

Analisando-se o diagrama da figura 82, a saída `O0001` terá um período de 1 segundo, pois necessita de dois pulsos de `R0000` para que complete o ciclo. Já a saída `O0002` terá o dobro do período de `O0001`, 2 segundos, já que este depende de dois pulsos de `O0001`. Assim, para aumentar a multiplicação do período, pode-se adicionar mais bloco

osciladores, ou adicionar um bloco contador para monitorar quantas transições foram sofridas por R0000. Na figura 83 é ilustrado uma multiplicação de período utilizando o bloco contador.

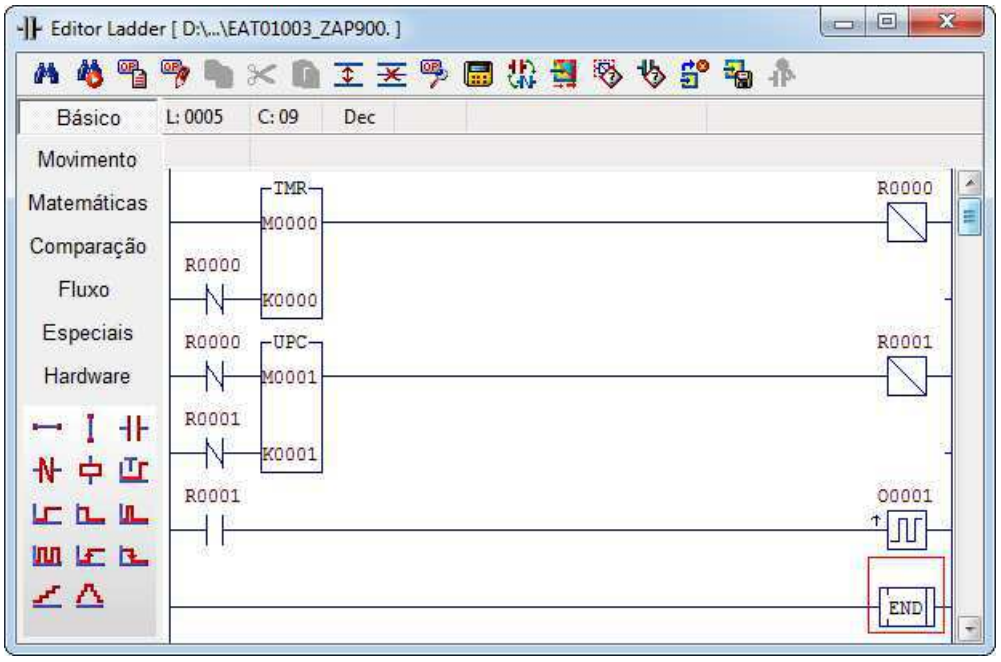


Figura 83: Diagrama *Ladder* de um bloco temporizador ligado à um bloco contador

Analisando o diagrama da figura 83, nota-se que o período de O0001 será de $T = 0,02 * K_{0000} * K_{0001}[s]$. Assim, se considerado K0000=50 e K0001=4, calcula-se o período de O0001 sendo de 4 segundos.

Exercícios de Preparação

1. Elabore um diagrama *Ladder* para gerar um sinal quadrado com período de 1 segundo.

2. Elabore um diagrama *Ladder* que ligue quatro saídas em ordem sequencial, sendo apenas uma ativa por vez. Os estados das saídas devem seguir a ordem $1000 \rightarrow 0100 \rightarrow 0010 \rightarrow 0001 \rightarrow 1000 \rightarrow ...$.

3. Lâmpadas vermelhas no topo de altas torres e edificações são requeridas pela aeronáutica para cidades que possuem tráfego aéreo. Elas ajudam na visibilidade em dias nublados para aeronaves e são geralmente projetadas juntamente com o pára-raios da torre/edifício. O sistema de controle destas lâmpadas é bastante simples: a luz é ligada por um curto período de tempo, e desligada por um outro período, muito maior, de tempo. Assim elabore um diagrama *Ladder* para este sistemas de controle, considerando que a lâmpada é acesa por 200 milisegundos e desligada por 5 segundos.

4. Em indústrias, a partida do motor era dada a partir de dois botões sem retenção colocados em série. Assim, a partida do motor era feito da seguinte forma: segurando o primeiro botão com uma mão, apertava-se o segundo com a outra mão. Mas, após algum tempo, percebeu-se que os operadores deixavam o primeiro botão sempre pressionado por um objeto, tendo apenas que apertar o segundo botão quando fosse necessário ligar o motor. Para se evitar essa situação, foi projetado um sistema no qual, após o primeiro botão ser pressionado e liberado, o operador tem até cinco segundos para apertar o segundo e ligar o motor. Se o tempo for excedido, o primeiro botão é desativado e deve ser pressionado novamente. Elabore um diagrama *Ladder* para este sistema.

Prática Experimental

1. Implemente o resultado do exercício 1 e verifique o sinal de saída.

2. Implemente o resultado do exercício 2. O que seria necessário modificar no diagrama para ligar mais saídas?

3. Implemente e teste o resultado do exercício 3. Como se pode modificar o diagrama atual para aumentar o tempo ligado de 200ms para 1s?

4. Implemente o diagrama obtido no exercício 4 e teste se o problema descrito foi superado.

5.4 Guia Experimental: Sensores e Atuadores

Objetivo

O objetivo deste guia é familiarizar o aluno com o funcionamento de sensores e atuadores com CLP, e introduzir operações matemáticas e comparativas.

Para a realiazação deste experimento, é necessário a leitura prévia das seções 4.1.3, 4.3.4, 4.3.5, 4.3.6 e 4.4.2 da apostila.

Introdução

O uso de sensores e atuadores analógicos é de extrema importância para o controle de processos. Com eles é possível obter dados importantes de um processo tal como escrever dados para o controle. A leitura de sensores e a escrita em atuadores analógicos em *Ladder* é simples, feita simplesmente utilizando o bloco funcional *MOV*.

O bloco funcional *MOV* tem a função de manipular os dados na memória do CLP. Este bloco leva dois parâmetros, `P1` e `P2`, e tem a simples função de fazer `P2=P1`. Assim, o dado que está em `P1` é copiado para `P2`.

De acordo com o princípio de funcionamento do CLP, antes do programa ser executado, as entradas (analógicas e digitais) são copiadas para uma porção de memória do CLP, criando uma "imagem" do que realmente está nas portas. Para as saídas (analógicas e digitais) acontece coisa similiar: durante a execução do programa, as saídas são escritas em uma "imagem", que só depois são escritas nas saídas.

Assim, como as entradas e saídas analógicas são escritas na memória do CLP, pode-se manipulá-las utilizando o bloco *MOV*. Na figura 84 é mostrado um exemplo de leitura e escrita analógica.

O diagrama da figura 84 funciona de maneira simples: o valor da entrada analógica `E0000` é copiado para `M0000`, que é copiada para a saída analógica `S0000`. Como é visto, as leituras e escritas analógicas em *Ladder* ocorrem de maneira bastante direta.

É importante observar que os dados lidos/escritos utilizando os bloco *MOV* são dados brutos, ou seja, não foram tratados. Estes dados variam de 0 a $2^n - 1$, onde n é a resolução do conversor A/D do CLP. Para transformar estes dados lidos/escritos em unidades de engenharia, é necessário utilizar os blocos funcionais de operações matemáticas. Estes blocos funcionais matemáticos estão sob a guia *Matemática*, e implementam as 13

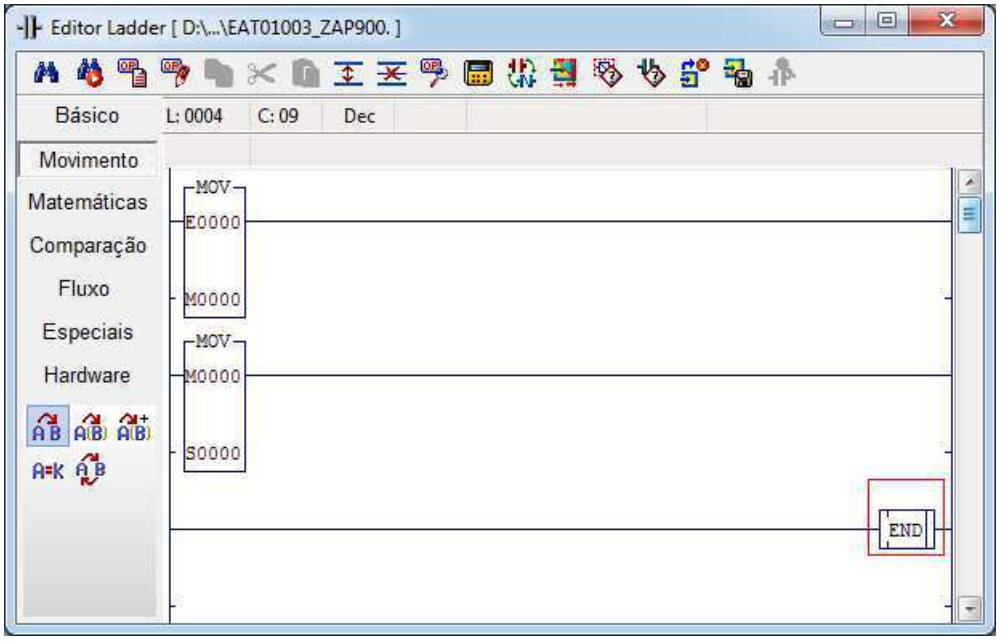


Figura 84: Diagrama *Ladder* para leitura e escrita analógica.

operações matemáticas mais comuns. Considere como exemplo, o sensor LM35 da figura 85, cuja a equação 3 transforma esse dado bruto lido em temperatura em ^oC.

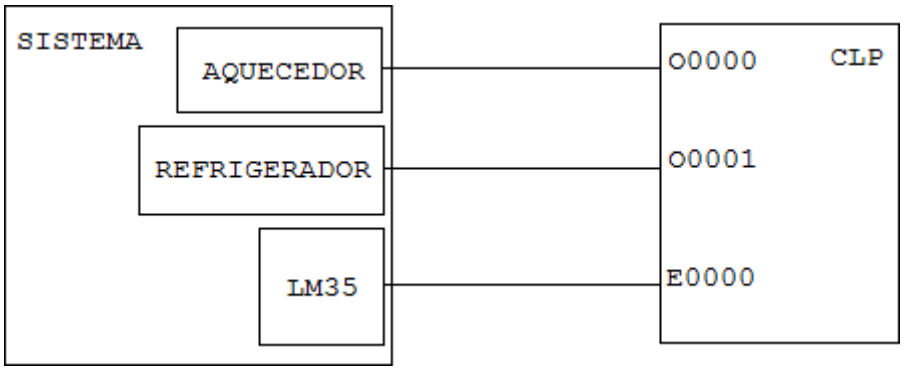


Figura 85: Diagrama de ligação entre um sistema de controle de temperatura e um CLP

Utilizando o bloco multiplicador, pode-se implementar esta equação de acordo com o diagrama da figura 86. Considere `D0000=0,122`.

No diagrama da figura 86, o valor lido da entrada analógica, `M0000`, é multiplicado pela variável real `D0000` e o resultado desta multiplicação é guardado em `D0001`. Assim, `D0001` armazena o verdadeiro valor da temperatura lida pelo sensor LM35.

Ladder também possui blocos comparativos. Estes blocos realizam testes lógicos entre dois parâmetros e, caso o teste lógico seja verdadeiro, a saída é ativada. São disponíveis 7 blocos comparativos sob a aba *Comparação*. Os testes realizados são sempre do parâmetro

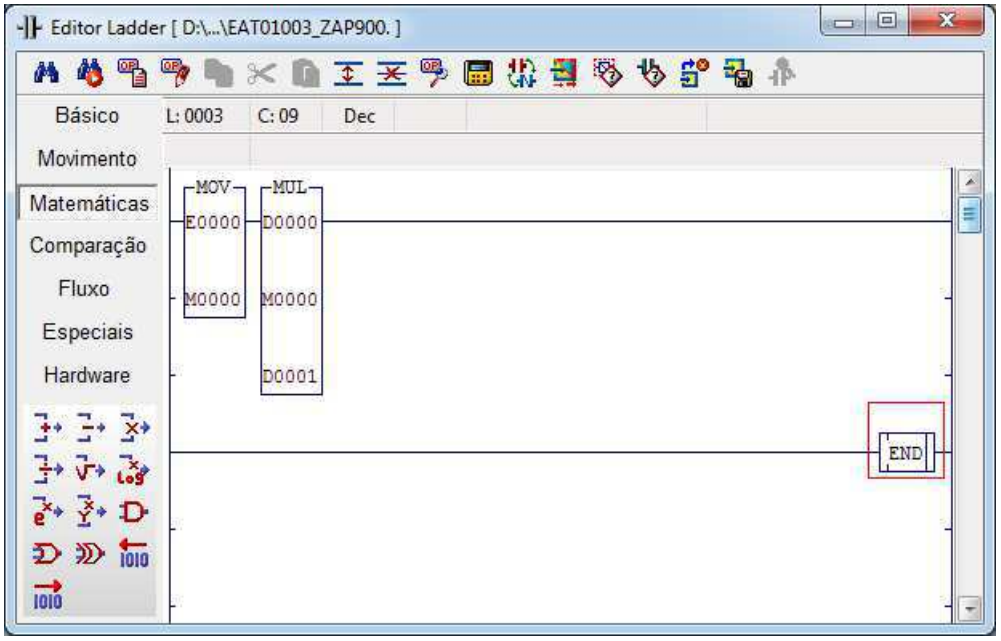


Figura 86: Diagrama *Ladder* para implementação da transformação de dados brutos em temperatura.

P1 em relação ao parâmetro P2. No diagrama da figura 87 é mostrado um pequeno sistema para aviso de temperatura.

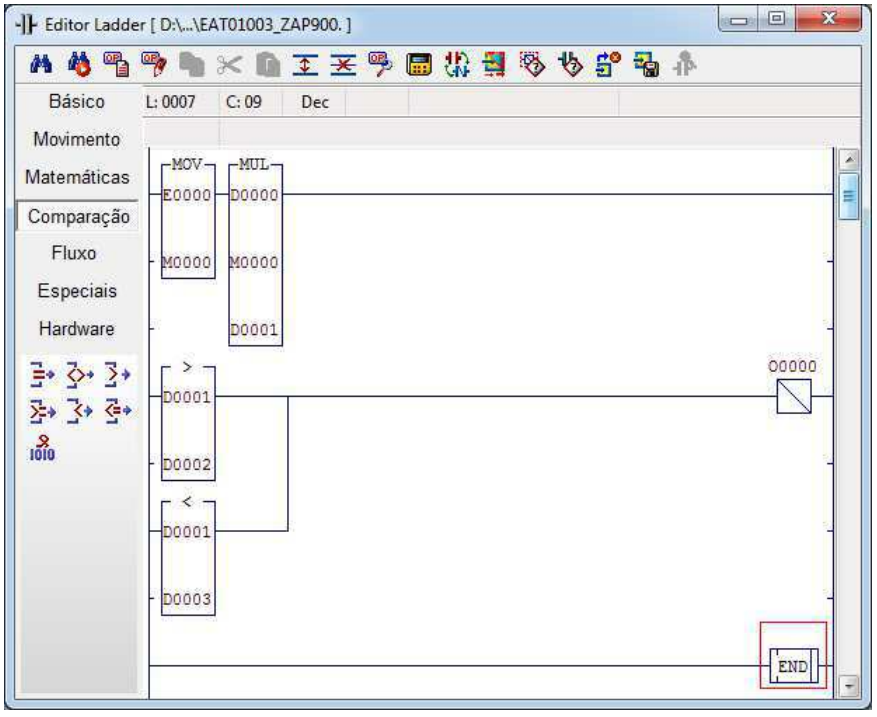


Figura 87: Diagrama *Ladder* para teste de uma faixa de temperatura.

O diagrama da figura 87 lê a entrada analógica E0000 e a trata, guardando o valor da

temperatura em `D0000`. Na parte de comparação, os dois bloco comparativos, maior que e menor que, comparam se `D0000` é maior que `D0002` ou menor que `D0003` e caso algum dos testes for verdadeiro, a saída `O0000` é ativada.

Outro tipo de sinal muito utilizado é o sinal *PWM*. O *PWM* no SPDSW é gerado pelo bloco gerador de frequência que fica sob a aba *Hardware*. O PWM é comumente utilizado em inversores para controlar a potência fornecida para uma carga. O bloco leva três parâmetros: o primero é o identificador do canal que será utilizado para gerar o PWM, no caso do kit eZTK900, existe apenas um canal gerador identificado por `T0000`; o segundo parâmetro é a frequência do sinal, que pode variar de 57Hz a 4kHz; o terceiro parâmetro define o *duty cycle* do PWM, variando de 0 (0%) a 999 (99,9%). No diagrama da figura 88 é mostrado a geração de um sinal PWM com *duty cycle* de 70,5% com frequência de 2kHz. Considere `M000=2000` e `M0001=705`.

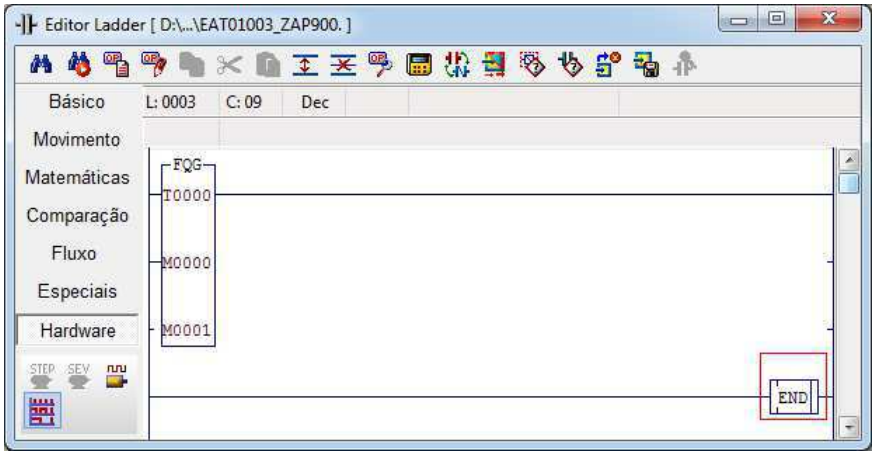


Figura 88: Diagrama *Ladder* para geração de um sinal *PWM*.

Exercícios de Preparação

1. Elabore um diagrama que permita comutar entre a leitura de duas entradas anlógicas, `E0004` e `E0005`, e escrever a entrada selecionada na saída analógica `S0000`.

2. Elabore um diagrama que implemente a equação $Y = 10^{-4} * (X - 2048)^2$. Considere X como sendo o valor lido da entrada `E0004`, e Y o valor que será escrito na saída `S0000`.

3. Elabore um diagrama *Ladder* para gerar um sinal PWM com o *duty cycle* que varia linearmente com a entrada `E0004`. Quando `E0004=4095`, o *duty cycle* deve ser de 99,9%.

4. No controle de velocidade de um motor CC, usa-se um sinal PWM para controlar a potência fornecida para este. Um sensor mede a velocidade deste motor e envia um sinal analógico à entrada `E0004` do CLP. Este sensor é calibrado de forma que o valor lido já é dado em RPM, ou seja, se o valor lido for `E0004=1200` significa que o motor está a 1200rpm. Elabore um diagrama que controle o *duty cycle* do PWM gerado de modo que, se o motor estiver acima da velocidade desejada, o valor do *duty cycle* deve ser diminuído linearmente até que o motor esteja na velocidade desejada. O mesmo se o motor estiver abaixo da velocidade desejada: o *duty cycle* deve ser incrementado linearmente até que o motor atinja a velocidade desejada.

5. O jogo de "mais ou menos" tem regras bastantes simples: é escolhido um número aleatório e o jogador deve adivinhar este número chutando valores; se o valor escolhido pelo jogador for maior que o número correto, o jogador é avisado que deve chutar um número menor, e se o valor do jogador for menor que o número correto, o jogador é instruído a chutar um número maior. O jogador vence quando o número chutado é idêntico ao número escolhido. Utilizando a entrada analógica `E0004` para escolher o número aleatório e as entradas `O0000..0007` para chutar o número em binário, elabore um diagrama *Ladder* para este jogo. Use a leitura digital em bloco. (Dica: Divida o número lido de `E0004` por 16 para ficar na faixa de 0 a 255, que é o máximo valor que as entradas `O0000..0007` podem atingir)

Prática Experimental

Para simular as entradas analógicas, utilize os reostatos do kit eZTK900 que são ligados às entradas `E0004` e `E0005` do CLP.

1. Implemente e teste o resultado obtido do exercício 1.
2. Implemente o diagrama obtido do exercício 2. Qual o valor mínimo escrito na saída e em que valor de entrada
3. Implemente e teste o resultado obtido do exercício 3.
4. Implemente e teste o resultado obtido do exercício 4. Considere que a velocidade desejada seja de 1800rpm.
5. Implemente e teste o resultado obtido do exercício 5.

5.5 Guia Experimental: Filtros e controle PID

Objetivo

O objetivo deste guia é familiarizar o aluno com a implementação de filtros digitais de primeira ordem e controles PID.

Para a realiazação deste experimento, é necessário a leitura prévia das seções 4.4.4 e 4.4.5 da apostila.

Introdução

No tratamento de sinais analógicos, muitas vezes o ambiente onde os sensores estão inseridos são propícios à disturbios e ruídos, fazendo com que o sinal lido sofra interferências e divirja do valor real. Nestes casos, podemos utilizar filtros passa-baixo ou passa-alto para permitir a passagem das frequências desejadas, e atenuar ou anular as outras.

A implementação da equação destes filtros no SPDSW é bastante direta. O único problema na implementação é saber a variação de tempo. Por isso, pode-se utilizar dois métodos para descobrir a variação de tempo. O primeiro método consiste em utilizar um temporizador para ativar a leitura do sinal e a implementação da equação. Este método é bom para sinais de frequências baixas, dado que a mínima temporização é de 10 milisegundos. Na figura 89 é mostrada a implementação de um passa-baixo utilizando este primeiro método.

No diagrama da figura 89, `Q0000` e `Q0001` são constantes iguais a $1-\alpha$ e α, respectivamente. Como a variação de tempo é constante e igual a `K0000`, α também será constante e não precisa ser recalculado. O valor filtrado é guardado na variável `D0004`.

O segundo método de implementação é calcular a variação de tempo baseado no número de vezes que o programa foi executado em um intervalo de tempo pré-determinado. Deste modo, pode-se calcular a variação de tempo como sendo o tempo pré-determinado dividido pelo número de execuções. Na figura 90 é mostrada a implementação deste método com um filtro passa-baixo.

A implementação do diagrama da figura 90 está apresentando as *TAGs* das variáveis. Toda vez que o programa é executado, `CONT` é adicionado de 1, e a entrada analógica `ENTAN` é lida. O temporizador está programado para 20ms, e quando este tempo é alcançado, os 20ms são divididos pelo número de execuções `CONT` e assim sabemos quanto tempo foi

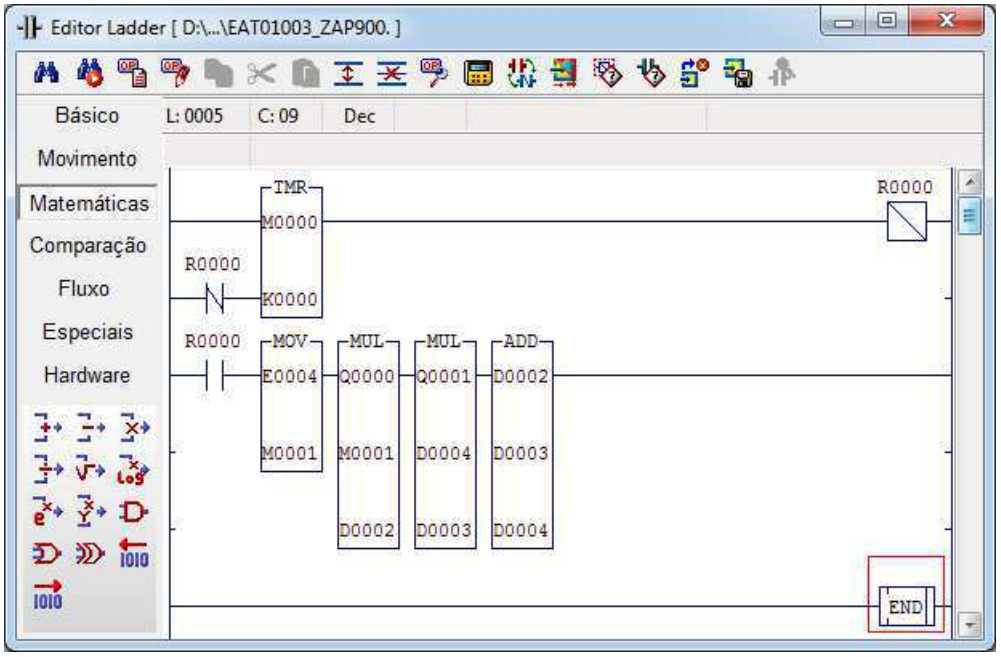


Figura 89: Diagrama *Ladder* para implementação de um filtro passa-baixo.

gasto, em média, em cada execução do programa. Após calculado `DT`, `CONT` é zerado e é calculado o valor de α, `ALFA`, e $1-\alpha$, `1-ALF`. A quarta linha é a implementação da equação do filtro.

Este método tem a vantagem de ser mais rápido, mas peca na precisão do cálculo da variação de tempo. A escolha do método utilizado depende das necessidades de cada sistema.

Para a implementação do controle PID, O SPDSW define o bloco funcional *PID*, que é o primeiro bloco sob a aba *Especiais*. À mostra, este bloco leva apenas um parâmetro do tipo identificador, `T`, que para o kit eZTK900 pode variar de `T0000` a `T0007`. Na figura 91 é mostrado o bloco *PID* inserido no programa.

A entrada `A1` controla se o bloco está ativo ou não, podendo pausar o controle a qualquer momento. A entrada `A2` escolhe se o bloco está em automático ou manual, que será explicado mais a frente. Para editar as variáveis que o bloco PID controla, seleciona-se o bloco e então pressiona-se **Ctrl+T**. Este comando abrirá a janela de edição do bloco PID, como mostrado na figura 92.

Na janela de edição, pode-se definir os seguintes parâmetros:

PV (*Process Variable*) Posição na memória onde é guardado a variável do processo, $y(t)$;

SP (*Set Point*) Posição na memória onde é guardado a entrada de referência, $r(t)$;

OV (*Output Variable*) Posição na memória onde é guardado a saída do controlador, $u(t)$;

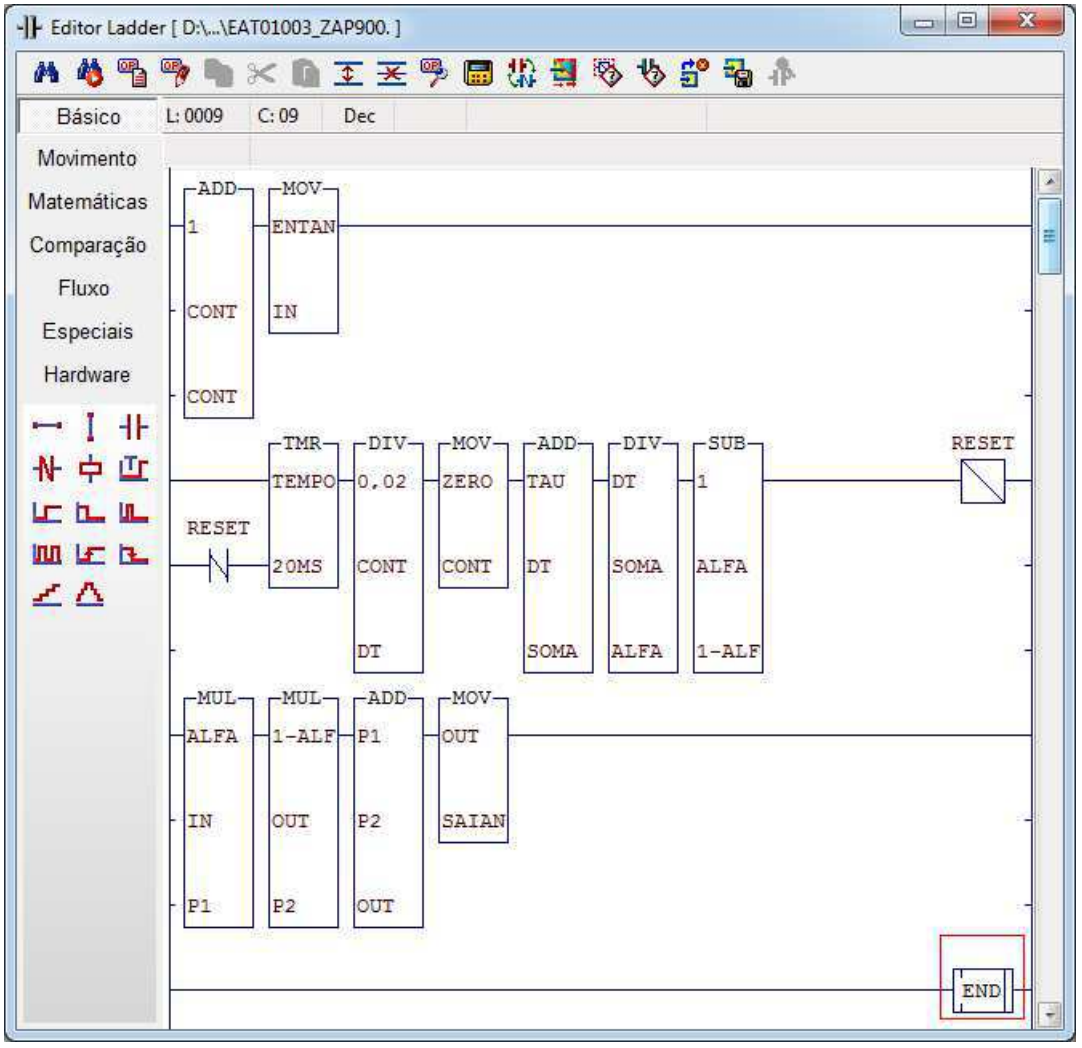


Figura 90: Diagrama *Ladder* para implementação de um filtro passa-baixo com variação de tempo mutável.

GP Posição na memória onde é definido o ganho proporcional, K;

TI Posição na memória onde é definido o tempo integral, T_i;

TD Posição na memória onde é definido o tempo derivativo, T_d;

OV Máx Valor máximo que pode ser assumido por `OV`, em %;

OV Min Valor mínimo que pode ser assumido por `OV`, em %;

OV Ini Valor inicial de `OV`, em %.

O SPDSW também permite que o bloco PID seja monitorado enquanto o programa está sendo executado. Para isso, na janela principal do SPDSW, clica-se em *Supervisão* → *Supervisão PID*. A figura 93 mostr a janela de supervisão para PID.

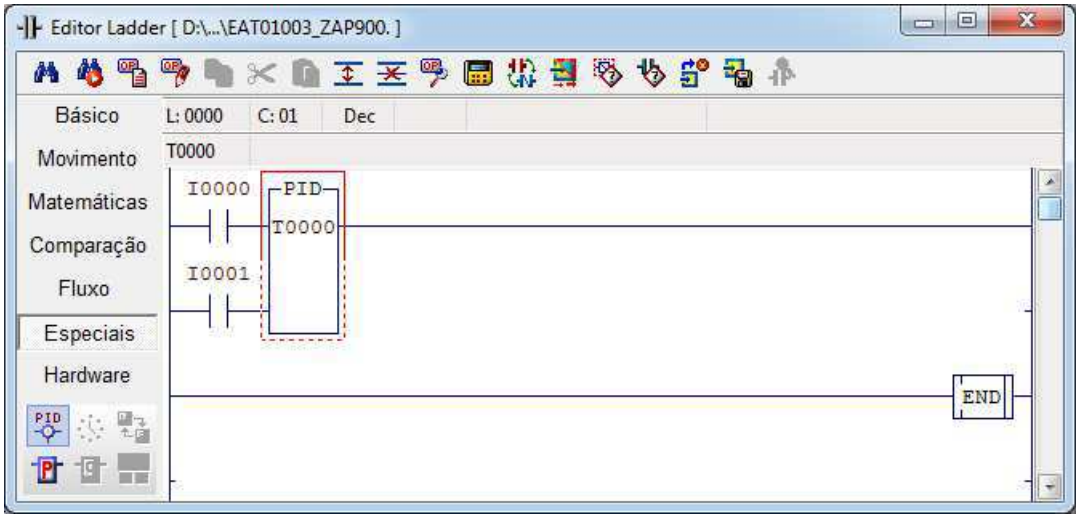


Figura 91: Diagrama *Ladder* com um bloco PID controlado por duas chaves.

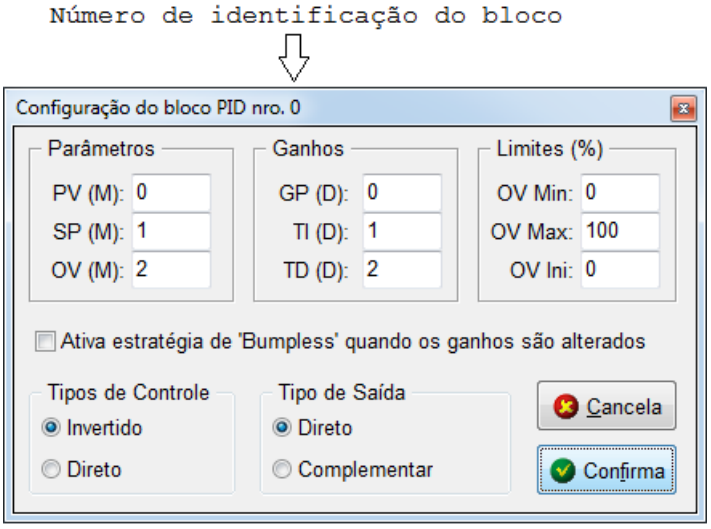


Figura 92: Janela de edição do bloco PID.

Nesta janela, pode-se ver os gráficos da variável de entrada, da variável de saída, e do *Set Point*. Ao lado, é possível ver os valores selecionados para ganho proporcional (GP), tempo integrativo (TI) e tempo derivativo (DT) e os valores do *Set Point* (SP), da variável de processo (PV) e da variável de saída (OV).

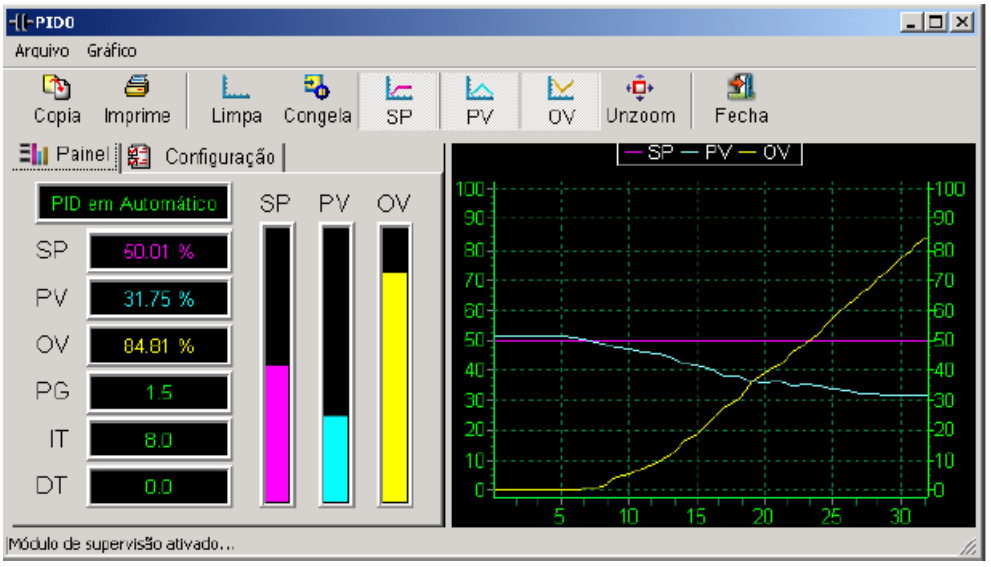


Figura 93: Exemplo da janela de supervisão do bloco PID.

Exercícios de Preparação

1. Elabore o diagrama *Ladder* de um filtro passa-alto com frequência de corte de 60Hz, utilizando o método de variação de tempo constante.

2. Elabore o diagrama *Ladder* de um filtro passa-alto com frequência de corte de 60Hz, utilizando o método de variação de tempo mutável.

Prática Experimental

1. Implemente os resultados dos exercícios 1 e 2, com `E0004` sendo a variável de entrada e o resultado sendo escrito em `S0000`. Existe diferença entre os dois filtros?

6 Considerações Finais

O estágio supervisionado realizado no LIEC foi de grande proveito. Este permitiu que os conhecimentos sobre CLPs e a linguagem *Ladder* adquiridos durante o curso de graduação fossem expandidos e refinados.

O estudo da linguagem *Ladder* neste estágio permitiu adquirir certa expertise no assunto, chegando ao desenvolvimento de uma apostila e guias experimentais sobre a linguagem, o que foi bastante gratificante.

Neste estágio foi possível retribuir um pouco de conhecimento à Universidade Federal de Campina Grande, com a apostila e os guias experimentais produzidos.

7 Bibliografia

BRYAN, L. A.; BRYAN, L. A. *Programmable Controllers:* Theory and Implementation. 2 Ed. Atlanta: Industrial Text, 1997.

PARR, E. A. *Programmable Controllers:* An Engineer's Guide. 3 Ed. Burlington: Newnes, 2003.

NETO, Rogério C. P. *Desenvolvimento de um ambiente para Programação de um Micro-CLP*. Vitória: 2003.

GUIMARÃES, Hugo C. F. *Norma IEC 61131-3 para Programação de Controladores Lógicos Programáveis:* Estudo e Aplicação. Vitória: 2005.

SILVA, Marcelo E. da. *Controladores Lógico Programáveis* - Ladder. Piracicaba: 2007.

HI TECNOLOGIA. Kit de Treinamento com o controlador eZAP900: Minas Gerais. Disponível em: http://www.hitecnologia.com.br/produtos/hardware/kit-de-treinamento/kit-familia-zap900/eztk900. Acesso em: 7 de Janeiro de 2013.