

AUTOMAÇÃO E PROGRAMAÇÃO DE CLPS

 Cursoslivres



Programação de CLPs

Linguagens de Programação de CLPs

Introdução às Linguagens de Programação para CLPs: Ladder, Texto Estruturado e Blocos Funcionais

Os Controladores Lógicos Programáveis (CLPs) utilizam diversas linguagens de programação para desenvolver os programas que controlam processos industriais. As principais linguagens de programação para CLPs são a Ladder (LD), o Texto Estruturado (ST) e os Blocos Funcionais (FBD). Cada uma dessas linguagens oferece diferentes abordagens para resolver problemas de controle, e a escolha da linguagem pode depender da aplicação específica, da experiência do programador e das preferências do setor.

1. Ladder (LD):

- **Descrição:** A linguagem Ladder, também conhecida como Diagrama de Escada, é uma das linguagens mais antigas e amplamente utilizadas para programação de CLPs. Foi desenvolvida para ser similar aos esquemas elétricos tradicionais, facilitando a transição para técnicos e engenheiros eletricitas.

- **Estrutura:** Consiste em uma série de "degraus" que representam instruções de controle, conectados por "trilhos" verticais que simbolizam a alimentação elétrica. Cada degrau contém elementos como contatos, bobinas, temporizadores e contadores.

2. Texto Estruturado (ST):

- **Descrição:** O Texto Estruturado é uma linguagem de alto nível, similar a linguagens de programação convencionais como Pascal ou C. Permite a escrita de código estruturado e lógico utilizando instruções textuais.
- **Estrutura:** Utiliza blocos de código com variáveis, loops, condicionais e funções, permitindo a implementação de lógica complexa de forma clara e eficiente.

3. Blocos Funcionais (FBD):

- **Descrição:** A linguagem de Blocos Funcionais utiliza uma abordagem gráfica para a programação de CLPs, onde a lógica de controle é representada por blocos funcionais conectados por linhas que indicam o fluxo de dados.
- **Estrutura:** Cada bloco funcional realiza uma função específica (por exemplo, operações lógicas, aritméticas ou de temporização) e pode ser interligado a outros blocos para formar a lógica do controle.

Vantagens e Desvantagens de Cada Linguagem

1. Ladder (LD):

○ Vantagens:

- Facilidade de entendimento e uso para técnicos com formação em eletricidade e manutenção.
- Boa visualização de processos sequenciais e lógica combinacional.
- Amplamente suportada por diferentes fabricantes de CLPs.

○ Desvantagens:

- Pode se tornar complexa e difícil de manter em programas grandes e com lógica complexa.
- Menos eficiente para operações matemáticas avançadas e manipulação de dados.

2. Texto Estruturado (ST):

○ Vantagens:

- Flexibilidade e poder para desenvolver lógica complexa e algoritmos sofisticados.
- Facilita a reutilização de código e a modularidade.
- Adequado para programadores com experiência em linguagens de programação convencionais.

○ Desvantagens:

- Pode ser difícil de entender e depurar para técnicos sem formação em programação.

- Menos intuitivo para visualizar processos sequenciais e lógicas simples.

3. Blocos Funcionais (FBD):

○ Vantagens:

- Interface gráfica intuitiva que facilita a visualização e a compreensão da lógica de controle.
- Ideal para processos contínuos e operações paralelas.
- Fácil de modificar e expandir.

○ Desvantagens:

- Pode se tornar confusa em programas muito grandes com muitos blocos interconectados.
- Limitada pela complexidade dos blocos funcionais disponíveis e pela necessidade de criar blocos personalizados para lógica específica.

Exemplos Práticos de Cada Linguagem

1. Ladder (LD):

- **Exemplo:** Controle de um motor utilizando um botão de partida e um botão de parada.

[] ---- [] ---- ()

Start Stop Motor

Texto Estruturado (ST):

- **Exemplo:** Controle de um motor utilizando um botão de partida e um botão de parada.

```
IF StartButton = TRUE THEN
```

```
    Motor := TRUE;
```

```
END_IF;
```

```
IF StopButton = TRUE THEN
```

```
    Motor := FALSE;
```

```
END_IF;
```

Blocos Funcionais (FBD):

- **Exemplo:** Controle de um motor utilizando um botão de partida e um botão de parada.

```
[StartButton] ----| |----| |---- (Motor)
```



Cada linguagem de programação de CLPs tem suas próprias vantagens e desvantagens, e a escolha da linguagem pode depender do contexto da aplicação e da experiência do programador. A linguagem Ladder é amplamente utilizada e intuitiva para técnicos eletricitas, o Texto Estruturado oferece flexibilidade e poder para lógica complexa, enquanto os Blocos Funcionais fornecem uma abordagem gráfica intuitiva para processos contínuos e paralelos.

Estrutura de um Programa em Ladder

Elementos Básicos da Linguagem Ladder

A linguagem Ladder (Ladder Logic ou Diagrama de Escada) é uma representação gráfica usada para desenvolver programas de controle para CLPs. Sua aparência e estrutura são semelhantes aos esquemas elétricos de circuitos de relés, facilitando a compreensão para técnicos e engenheiros. Os elementos básicos da linguagem Ladder incluem:

1. Trilhos Verticais:

- Representam a alimentação elétrica, com o trilho esquerdo como o "positivo" e o trilho direito como o "negativo" ou o retorno.

2. Contatos:

- **Contatos normalmente abertos (NO):** Representados por dois parênteses verticais (| |), eles permitem a passagem do sinal quando ativados.
- **Contatos normalmente fechados (NC):** Representados por dois parênteses com uma linha transversal (|/|), eles bloqueiam a passagem do sinal quando ativados e permitem quando desativados.

3. Bobinas:

- **Bobinas (Coils):** Representadas por um par de parênteses horizontais (()), são usadas para ativar ou desativar saídas como motores, válvulas e lâmpadas.

4. Temporizadores e Contadores:

- **Temporizadores (Timers):** Utilizados para criar atrasos temporais. Podem ser temporizadores de retardo na energização (TON) ou de retardo na desenergização (TOF).
- **Contadores (Counters):** Utilizados para contar eventos ou pulsos. Podem ser contadores ascendentes (CTU) ou descendentes (CTD).

5. Funções Lógicas:

- **AND, OR, NOT:** Implementadas pela combinação de contatos e bobinas para criar lógica de controle.

Criação de Diagramas Ladder

A criação de um diagrama Ladder envolve a disposição dos elementos mencionados acima para representar a lógica de controle desejada. O processo pode ser dividido em várias etapas:

1. Definição dos Requisitos:

- Identifique as entradas (sensores, botões) e saídas (motores, lâmpadas) do sistema.
- Determine a lógica de controle necessária para atender aos requisitos do processo.

2. Desenho do Diagrama:

- Inicie com os trilhos verticais representando a alimentação.
- Adicione contatos e bobinas conforme a lógica de controle definida.
- Utilize temporizadores e contadores conforme necessário para implementar atrasos e contagens.

- Combine elementos lógicos para criar a funcionalidade desejada.

3. Exemplo Prático:

- Controle de um motor com partida e parada:

|---[StartButton]---| |---[StopButton]---|/|---(Motor)---|

Simulação e Teste de Programas em Ladder

A simulação e o teste são etapas cruciais no desenvolvimento de programas em Ladder para garantir que a lógica funcione conforme o esperado antes de implementar no hardware real.

1. Simulação:

- Utilize o software de programação do CLP para simular o programa Ladder.
- Verifique o comportamento do programa em um ambiente virtual, observando as mudanças de estado das entradas e saídas.
- Realize ajustes conforme necessário para corrigir qualquer erro lógico identificado durante a simulação.

2. Teste no Hardware:

- Após a simulação bem-sucedida, transfira o programa para o CLP real.
- Conecte os dispositivos de entrada e saída ao CLP.
- Teste o programa no ambiente de operação real, verificando o funcionamento correto de todo o sistema.

- Realize ajustes finais no programa para otimizar a performance e garantir a robustez do controle.

Exemplos de Testes e Verificações:

- **Teste de Segurança:**

- Verifique se todas as condições de segurança são atendidas, como desligamento de emergência e intertravamentos.

- **Teste de Condições Extremas:**

- Teste o programa sob diferentes condições de operação, como sobrecarga de entradas e falhas de componentes.

- **Teste de Continuidade:**

- Garanta que o programa continue a operar corretamente após reinicializações ou falhas momentâneas de energia.

A estrutura de um programa em Ladder permite a criação de lógica de controle robusta e eficiente para aplicações industriais. A familiaridade com os elementos básicos, a habilidade de criar diagramas claros e a realização de simulações e testes rigorosos são fundamentais para o sucesso na automação de processos com CLPs.

Funções e Instruções Avançadas em Ladder

Utilização de Temporizadores e Contadores

Temporizadores e contadores são elementos essenciais em programas Ladder, permitindo o controle preciso de eventos baseados em tempo e em contagem de ocorrências.

1. Temporizadores (Timers):

- **Temporizador de Retardo na Energização (TON):** Atraso no acionamento de uma saída após uma entrada ser ativada.

```
|---[StartButton]---(TON, T1, 5s)---(Motor)---
```

Nesse exemplo, o motor (Motor) será ativado 5 segundos após o botão de partida (StartButton) ser pressionado.

- **Temporizador de Retardo na Desenergização (TOF):** Atraso na desativação de uma saída após uma entrada ser desativada.

```
|---[StartButton]---(TOF, T2, 10s)---(Light)---
```

Aqui, a luz (Light) permanecerá acesa por 10 segundos após o botão de partida (StartButton) ser liberado.

- **Temporizador de Pulso (TP):** Gera um pulso de duração fixa quando a entrada é ativada.

```
|---[StartButton]---(TP, T3, 3s)---(Buzzer)---
```

O alarme (Buzzer) será acionado por 3 segundos quando o botão de partida (StartButton) for pressionado.

1. Contadores (Counters):

- **Contador Ascendente (CTU):** Conta o número de eventos de uma entrada.

|---[PulseInput]---(CTU, C1, 10)---(Output)---|

A saída (Output) será ativada após 10 pulsos do (PulseInput).

- **Contador Descendente (CTD):** Conta regressivamente até zero, a partir de um valor inicial.

|---[PulseInput]---(CTD, C2, 5)---(Output)---|

A saída (Output) será ativada quando o contador (C2) atingir zero após 5 pulsos.

Instruções de Controle e Comparações

Além de temporizadores e contadores, os programas Ladder utilizam instruções de controle e comparações para tomar decisões baseadas em condições específicas.

1. Instruções de Controle:

- **Set (S):** Liga uma saída independentemente das condições subsequentes.

|---[Condition]---(S, Motor)---|

- **Reset (R):** Desliga uma saída independentemente das condições subsequentes.

|---[Condition]---(R, Motor)---|

Instruções de Comparação:

- **Maior que (GT):** Verifica se um valor é maior que outro.

|---[Value1 > Value2]---(Output)---|

- **Menor que (LT):** Verifica se um valor é menor que outro.

|---[Value1 < Value2]---(Output)---|

- **Igual a (EQ):** Verifica se dois valores são iguais.

|---[Value1 == Value2]---(Output)---|

Exemplos de Programas Complexos Utilizando Funções Avançadas

1. Controle de Esteira Transportadora com Temporizador e Contador:

- Objetivo: Acionar uma esteira transportadora que liga após 5 segundos quando um sensor detecta uma peça e desliga após 10 peças passarem pelo sensor.

|---[Sensor]---(TON, T1, 5s)---(CTU, C1, 10)---(Motor)---|

|---[C1 >= 10]---(R, Motor)---|

Sistema de Iluminação Automática com Temporizador:

- Objetivo: Acionar as luzes de um corredor por 30 segundos quando um sensor de movimento é ativado.

|---[MotionSensor]---(TP, T1, 30s)---(Lights)---|

Controle de Nível de Tanque com Comparação:

- Objetivo: Ligar uma bomba quando o nível do tanque estiver abaixo de um valor mínimo e desligar quando atingir um valor máximo.

|---[Level < MinLevel]---(S, Pump)---|

|---[Level >= MaxLevel]---(R, Pump)---|

Sistema de Alarme de Segurança:

- Objetivo: Acionar um alarme se uma porta for aberta por mais de 10 segundos.

|---[DoorOpen]---(TON, T1, 10s)---(Alarm)---|

Esses exemplos ilustram como as funções e instruções avançadas em Ladder podem ser utilizadas para criar programas de controle eficientes e robustos. A utilização adequada de temporizadores, contadores, instruções de controle e comparações permite a automação de processos complexos e a implementação de lógicas de controle sofisticadas em ambientes industriais.

