

# Apostila de Automação Industrial

João Paulo Cerquinho Cajueiro

1 de novembro de 2018

# Sumário

<b>1 Sistemas de automação</b>	<b>4</b>
1.1 História . . . . .	5
1.2 Classificação . . . . .	7
1.3 Pirâmide de automação . . . . .	8
<b>2 Instrumentação Industrial</b>	<b>10</b>
2.1 Sensores discretos . . . . .	11
2.1.1 Sensores de contato . . . . .	11
2.1.2 Sensores de proximidade . . . . .	13
2.1.3 Chaves de processo . . . . .	17
2.2 Sensores Contínuos . . . . .	17
2.2.1 Pressão e força . . . . .	17
2.2.2 Temperatura . . . . .	18
2.3 Características dos Instrumentos . . . . .	20
2.4 Transmissão de dados, aterramento e blindagem em instrumentação.	23
2.5 Diagrama P&I . . . . .	24
2.5.1 Transmissão de dados . . . . .	25
<b>3 Controladores Lógico-Programáveis (CLP)</b>	<b>28</b>
3.1 Dispositivos Eletromecânicos . . . . .	29
3.2 De relês a CLPs . . . . .	30
3.2.1 Diagrama Ladder . . . . .	31
<b>4 Programação - arduino</b>	<b>36</b>
4.1 Piscando um led. . . . .	37
4.2 Sinais analógicos . . . . .	39
4.3 Controle: for e if . . . . .	40
4.4 Comunicação com o computador . . . . .	41
<b>5 CLPs – Definições e termos comuns</b>	<b>43</b>
5.1 Variáveis . . . . .	45

5.2	Programação . . . . .	46
5.3	Ladder e Instruction List . . . . .	47
5.4	Memória . . . . .	48
5.4.1	Subida e descida de uma entrada. . . . .	49
5.5	Contadores . . . . .	52
5.5.1	Acesso às variáveis . . . . .	52
5.6	Temporizadores (timers) . . . . .	54
5.7	plcLib . . . . .	55
5.7.1	Acumulador . . . . .	56
5.7.2	Funções lógicas . . . . .	57
5.7.3	Memória . . . . .	59
5.7.4	Temporizadores . . . . .	60
5.7.5	Contadores . . . . .	61
<b>6</b>	<b>Linguagem grafctet</b>	<b>63</b>
6.1	Divergências e convergências . . . . .	64
6.2	Ações . . . . .	65
6.2.1	JGrafchart . . . . .	65
6.2.2	SFC . . . . .	66
6.3	Níveis . . . . .	68
<b>7</b>	<b>SCADA</b>	<b>73</b>
7.1	Variáveis . . . . .	74
7.2	Sinótico . . . . .	76
7.2.1	Requisitos . . . . .	76
7.3	Alarmes e Eventos . . . . .	78
7.4	Arquitetura de Hardware e Software . . . . .	80
<b>8</b>	<b>Gerenciamento da manufatura: PIMS e MES</b>	<b>82</b>
8.1	PIMS . . . . .	82
8.1.1	Historiador de Processos . . . . .	83
8.1.2	Banco de Dados Temporal . . . . .	85
8.1.3	Interface Gráfica . . . . .	85
8.2	MES . . . . .	86
8.2.1	Redes industriais . . . . .	90
<b>9</b>	<b>Redes Industriais</b>	<b>93</b>
9.1	Redes de Comunicação: Introdução e noções básicas . . . . .	94
9.1.1	Modelo OSI . . . . .	95
9.1.2	Internet . . . . .	98
9.2	Protocolos Industriais . . . . .	100

9.2.1	Modbus . . . . .	100
9.2.2	HART . . . . .	102
9.2.3	CAN . . . . .	103
9.2.4	Profibus DP . . . . .	103
9.2.5	AS-i . . . . .	103
9.2.6	EtherCAT . . . . .	103
9.2.7	OPC-UA . . . . .	103

# Capítulo 1

## Sistemas de automação

A palavra automação vem do latim *automatus* – mover por si mesmo. Logo a automação de uma tarefa consiste em fazer com que tal tarefa seja realizada de modo autônomo, sem envolver trabalho humano. Isto pode ser por diversos motivos: seja por que é uma tarefa perigosa e portanto queremos aumentar a segurança das pessoas, como num processo que envolva alta temperatura, por exemplo; seja para fazer a tarefa de forma mais rápida, seja para melhorar a qualidade do produto final ou seja porque simplesmente o custo do trabalho humano é muito elevado. Logo, podemos definir automação da seguinte forma:

Automação é a substituição do trabalho humano por sistemas autônomos visando melhorar segurança, qualidade, produção e custos.

Neste contexto, automação industrial é nada mais que a automação de um sistema industrial, ou de um sistema de manufatura. Embora manufatura venha de fazer com as mãos, a revolução industrial mudou este conceito, passando a significar a fabricação de praticamente qualquer produto. Do ponto de vista econômico, a manufatura é a transformação de materiais (matéria prima) em itens de maior valor (produto). Isto é conseguido por uma determinada sequência de processos químicos e físicos. De forma mais sucinta:

Manufatura é a transformação de matéria prima em produtos pela aplicação de um ou mais processos.

Logo, a automação industrial consiste em fazer os processos necessários para a manufatura com o mínimo de esforço ou interferência humana, visando melhor segurança, qualidade, produção e custo. Note-se que por esforço humano, queremos dizer tanto esforço físico quanto mental, logo uma máquina bastante complexa mas que funcione a manivela, não se classificaria como automação; da mesma forma uma máquina que não exija esforço físico mas requer atenção constante também não é automatizada (seria apenas mecanizada).

## 1.1 História

É interessante pegar alguns pontos chaves na história da automação. Embora várias máquinas mecânicas de diversas graus de complexidade já existissem, como por exemplo relógios mecânicos desde o século VIII na China e desde o século XIII na Europa e os robôs de Pierre Jaquet-Droz, do século XVIII, considera-se que a revolução industrial iniciou com a invenção do tear mecânico por Cartwright em 1785, que realiza um movimento relativamente complexo de forma automática a partir de uma roda d'água.

Por volta de 1788 houve a invenção do mecanismo de regulagem de fluxo de vapor de James Watt, o que permitiu controlar a potência de caldeiras e outras

máquinas a vapor, manipulando uma potência muito maior que uma roda d'água. Isto foi um grande impulso para a mecanização, mas a verdadeira automatização ainda ficava muito restrita devido a dificuldade de realizar processos complexos de forma automática. Ou seja, retirava-se grande parte do esforço físico do homem, mas ainda era necessário muito esforço mental.

Em 1820 o francês Charles Babbage, trabalhando para automatizar o processo mental do cálculo matemático, começou a desenvolver a sua máquina diferencial, que hoje chamáramos de uma calculadora mecânica. Ela evoluiu até o conceito da máquina analítica, descrita em 1837, que é considerada o primeiro projeto de computador, embora apenas partes dela tenham sido efetivamente construídas.

Em 1880, Herman Hollerith criou um novo método, baseado na utilização de cartões perfurados e calculadoras mecânicas, para automatizar algumas tarefas de tabulação do censo dos EUA que antes duravam 10 anos. Com seu método, o processo era concluído em 6.

Ao longo da primeira metade do século XX foram utilizados muitos sistemas eletromecânicos para o controle de processos industriais. Eram os chamados circuitos chaveados, que utilizavam relés para o controle lógico e para o comando de motores.

Em 1936, Alan Turing descreveu um *computador universal* em seu artigo “On Computable Numbers, with an Application to the Entscheidungsproblem”, o que hoje é conhecido como uma *Máquina de Turing*. Suas idéias foram desenvolvidas em 1944, com a construção do Colossus, considerado como o primeiro computador, embora tivesse a função específica de quebrar o código criptográfico alemão na segunda guerra. Ele consistia de um circuito com 1600 a 2400 válvulas com capacidade de processamento de vinte e cinco mil caracteres por segundo. A título de comparação, os computadores de casa de hoje em dia atingem 10 bilhões de cálculos por segundo.

O avanço da eletrônica fez que a capacidade de processamento dos computadores aumentasse de forma exponencial. Atualmente o mais rápido computador do mundo é o chinês Sunway TaihuLight, que faz 93 quatrilhões de cálculos por segundo, consumindo 15 MW.

Na década de 60 a General Motors fez uma especificação de um CLP – Controlador Lógico Programável, que é um computador voltado para o controle de processos industriais. Em 1968 foi criado o primeiro.

Hoje em dia toda automação está relacionada a sistemas computadorizados, seja em CLPs, CNCs, robôs industriais, automação dos sistemas de apoio a produção, entre outros.

## 1.2 Classificação

Hoje em dia se definem, grosso modo, 3 tipos de automação, tal qual mostra a figura 1.1: fixa, flexível e programável.

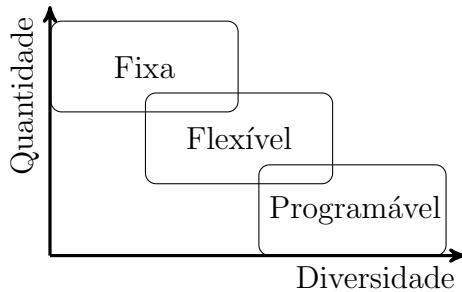


Figura 1.1: Tipos de automação industrial quanto a quantidade e diversidade de produtos.

A automação fixa é aplicada à produção de um único produto (ou com mínimas variações), em grandes quantidades: refinaria de petróleo, parafuso, tampas de garrafa, clips, biscoito, cerveja, etc. Ela utiliza equipamentos feitos sob medida para cada processo, que portanto tem alto custo inicial mas grande produtividade.

A automação flexível é aplicada à produção de produtos parecidos, em que pequenas modificações permitem a alteração do produto, como por exemplo mudança de um perfil a ser prensado ou extrudado ou a mudança das quantidades do mesmo conjunto de matérias primas (mudança de receita). Tipicamente é feita a chamada fabricação em lotes, onde entre um lote e outro se alteram as peças e/ou as sequências a serem seguidas de forma automática para ter o menor tempo parado possível. Exemplos são livros, circuitos integrados, potes de plástico, máquinas de café.

A automação programável é para produção de produtos diferentes mas cujo volume de produção não justifica um processo único. Ela usa máquinas de propósito geral, tais como robôs, ferramentas de controle numérico e impressoras 3d, onde a definição do processo é quase toda feita por *software*, de modo que o custo do maquinário é diluído em diversos produtos.

A tendência é cada vez mais ter a automação flexível e programável aumentando a capacidade de produção, de modo que a flexível vai ocupando nichos da fixa e a programável da flexível. Apesar disso, em vários casos é difícil imaginar alguns produtos deixando de utilizar a automação fixa.

## 1.3 Pirâmide de automação

A automação em larga escala de uma grande indústria envolve muito mais que a mera manufatura e inclui problemas de abastecimento, armazenagem, análise de mercado, exigências ambientais, entre várias outras coisas. Uma forma de se separar os diferentes problemas da automação é através da chamada Pirâmide de Automação, mostrada na figura 1.2.

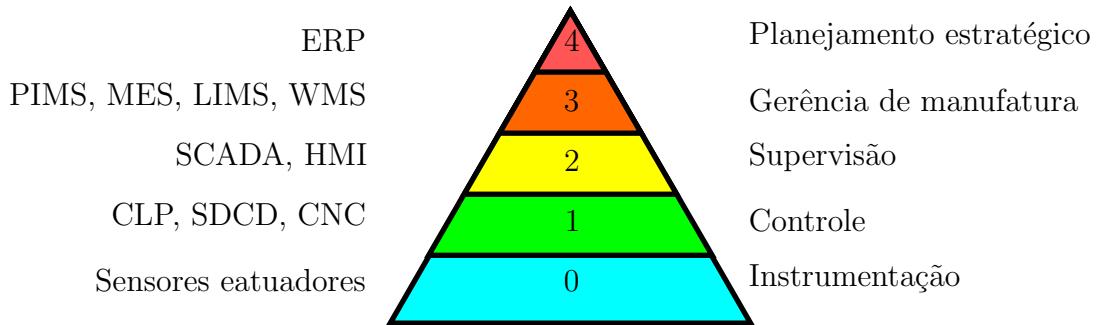


Figura 1.2: Pirâmide de automação.

Note que esta não é a única representação da pirâmide: algumas começam pelo 1, outras têm apenas 4 camadas, e assim por diante, logo mais importante que o número de cada camada é o que tais camadas significam.

**Nível 0: Instrumentação** Camada onde se encontram instrumentos, sensores, motores, máquinas, etc. Consistem nos equipamentos do chamado “*chão de fábrica*”.

**Nível 1: Controle** Controle automático da planta – onde se localizam os Controladores Lógico-Programáveis (CLP), os Sistemas Digitais de Controle Distribuído (SDCD), os Controles Numéricos Computadorizados (CNC) e/ou computadores de controle.

**Nível 2: Supervisão** Supervisão e controle do processo através de Interfaces Homem-Máquina (IHMs) ou SCADA (*Supervisory Control And Data Acquisition*).

**Nível 3: Gerenciamento da Manufatura** Gestão dos recursos da planta e controle da produção. Sistemas PIMS (*Process Information Management System*) e MES (*Manufacturing Execution Systems*).

**Nível 4: Planejamento Estratégico** Gestão dos recursos e produção da empresa como um todo. ERP – *Enterprise Resources Planning*.

A figura 1.3 mostra um diagrama em blocos de um processo automatizado, contendo os blocos representativos dos níveis 0 até o 3 da pirâmide de automação. A instrumentação consiste de sensores para acompanhar as variáveis controladas de um processo e atuadores para alterar as variáveis manipuladas. Como tais atuadores irão operar é definido pelo controle, em função de parâmetros definidos pelo supervisório. Este também monitora estas informações e as repassa ao sistema de gerenciamento, que aglutina dados de diversos processos para a tomada de decisões de mais alto nível.

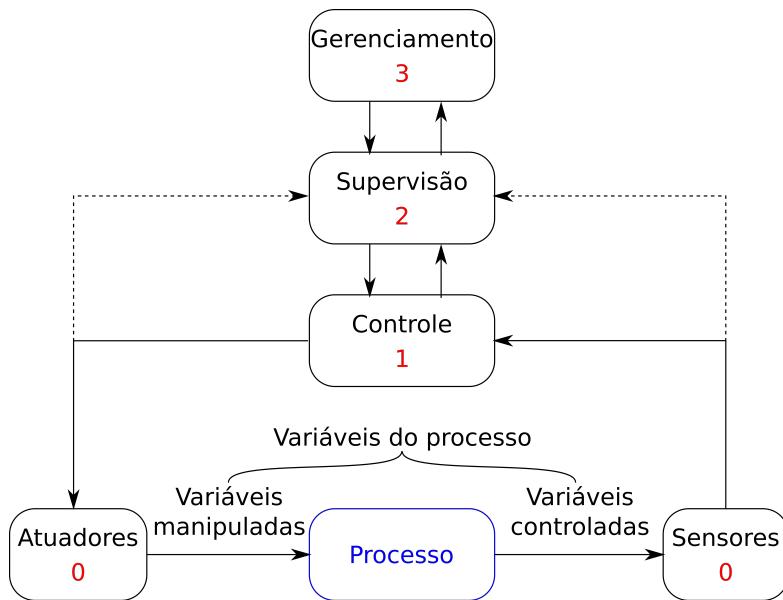


Figura 1.3: Diagrama em blocos da automação de um processo.

Este texto faz um estudo da automação industrial de modo *bottom-up*: começando do nível 0 até o nível 3. O nível 4 é mais importante para um estudo de engenharia de processo ou de produção e portanto não será abordado.

# Capítulo 2

## Instrumentação Industrial

A instrumentação é relativa aos equipamentos que obtém informações acerca do estado de um determinado processo industrial. Os chamados sensores e transdutores.

A definição de sensor e transdutor está longe de ser uma unanimidade. Do ponto de vista da instrumentação, define-se:

**Sensor** é um elemento que gera um sinal padronizado (normalmente elétrico) a partir de uma grandeza física (calor, luz, som pressão etc).

**Transdutor** é um dispositivo que converte um sinal de uma grandeza para outra.

De onde se tira que todo sensor é um transdutor, mas não o contrário. Em geral os sensores se utilizam de transdutores para converter uma grandeza específica para outra mais facilmente manipulável.

Uma classificação bastante útil é a de sensores discretos ou contínuos:

**Sensores discretos** geram uma saída discreta, normalmente binária — 0 ou 1, aberto ou fechado, acionado ou desligado.

**Sensores contínuos** geram uma saída que varia continuamente em função da entrada. Pode ser composto por um único transdutor (um resistor por exemplo).

Um complicador é que alguns textos técnicos apresentam uma confusão de sensores e transdutores com sensores discretos (chamados simplesmente de sensores) ou contínuos (chamados erroneamente de transdutores). Portanto deve-se tomar cuidado com o significado destes termos.

Do ponto de vista do fluxo de energia, pode-se ter sensores passivos ou ativos:

**Sensores ativos** geram um sinal de saída sem a necessidade de alimentação externa. Exemplos: termopar, célula fotoelétrica.

**Sensores passivos** requerem uma entrada de energia para gerar um sinal de saída. Exemplos: Termoresistência, sensor capacitivo.

A maioria dos sensores industriais são passivos.

## 2.1 Sensores discretos

Como citado, a maioria dos sensores discretos industriais são chaves elétricas. Neste caso diferenciam-se os sensores de contato, que são chaves eletromecânicas; os sensores de proximidade, que detectam a presença de algum objeto sem tocá-lo; e as chaves de processo, que atuam em função de uma variável do processo.

### 2.1.1 Sensores de contato

Do ponto de vista da instrumentação, qualquer chave presente no nível 0 da pirâmide que gera sinais lidos no nível 1 realiza logicamente o mesmo tipo de função. Daí que se consideram como sensores de contato:

**Botoeiras** ou botões, acionados pelo operador do processo.

**Chaves de fim de curso** que são acionadas mecanicamente por algo no processo.

As botoeiras podem ter diversos formatos e funções, vide a figura 2.1. Podem ser simples botões que fecham o contato apenas quando pressionados (sem retenção) do tipo liga e desliga (com retenção), interruptores de 3 estados, entre outros. Alguns botões de emergência são acionados quando apertados e só são desligados com o uso de uma chave; também é comum botoeiras que energizam equipamentos poderem ser travadas na posição desligada com um cadeado, o que permite que o manutentor trave o equipamento desenergizado enquanto efetua algum serviço.

Chaves de fim de curso são chaves eletromecânicas feitas para serem acionadas por algum produto ou equipamento. São usadas para detectar, por exemplo, a passagem do material sendo processado por um determinado ponto, a posição final de movimentação de algum equipamento, entre outros. Normalmente é implementado como um botão com uma alavanca ou algum outro mecanismo na parte a ser acionada.

As chaves eletromecânicas são relativamente baratas, de uma tecnologia já bem amadurecida e são imunes a interferências eletromagnéticas, porém o movimento a que são submetidas gera desgaste, o que faz com que sua vida útil seja reduzida. Além disso, necessitam do contato com o alvo para ser acionadas, o que pode ser inviável em alguns casos, e tem um tempo de resposta da ordem de milisegundos, que pode ser lento demais para algumas aplicações.



Figura 2.1: Diversos tipos de botoeiras.



Figura 2.2: Exemplos de chaves de fim de curso.



Figura 2.3: Encapsulamentos comuns para sensores de proximidade.

### 2.1.2 Sensores de proximidade

Existem diversos princípios físicos que podem e são usados para detectar a proximidade de algum material. Estes são chamados de sensores de proximidade e normalmente são aplicados quando se tem algum impedimento ao uso de chaves eletromecânicas.

Para vários casos, é comum o uso de um encapsulamento em formato de rosca, como visto na figura 2.3. Isto faz com que vários sensores de proximidade se pareçam, mesmo que usem princípios físicos diferentes.

#### Indutivo

O sensor indutivo conta com uma bobina na sua extremidade sensora, que gera um campo magnético variável na sua frente. A frequência deste campo magnético depende da própria indutância desta bobina, que por sua vez depende do que estiver na frente do sensor.

Um alvo que esteja na frente deste sensor pode alterar esta indutância por 2 efeitos: se for um condutor, gerará um campo magnético contrário, aumentando a indutância; se for ferroelétrico, concentrará o campo magnético, também aumentando a indutância. Este segundo efeito é maior que o primeiro, o que faz com que este sensor responda melhor a ferro do que a cobre, mesmo com o cobre sendo melhor condutor que o ferro.

A distância que um sensor indutivo consegue detectar um alvo de ferro na sua frente é chamada de **distância sensora nominal**. Para outros materiais esta distância diminui e para materiais não condutoros e sem propriedades magnéticas ela cai a zero. A figura 2.4 mostra a variação da distância sensora de sensores

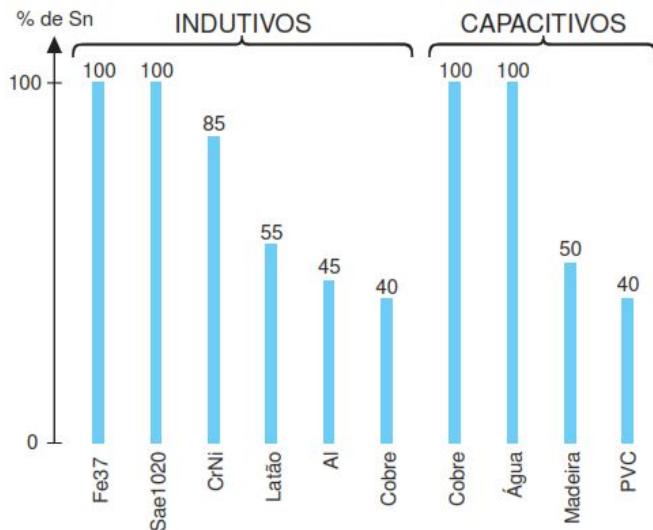


Figura 2.4: Variação da distância sensora de sensores indutivos e capacitivos para diferentes tipos de material.

indutivos e capacitivos para diferentes materiais.

Além da limitação da distância que pode ser usado e da limitação do material do alvo, sensores indutivos estão sujeitos a interferências eletromagnéticas, que podem gerar falsas detecções.

### Capacitivo

Sensores capacitivos usam um circuito oscilador (basicamente o mesmo de sensores indutivos) para detectar a variação da capacitação entre duas placas metálicas na sua ponta. Esta capacitação varia se o alvo tiver uma constante dielétrica diferente que a do ar. Funciona muito bem para água e cobre (que tem uma constante dielétrica 80 vezes maior que o ar) e praticamente não responde para ferro, aço e alumínio.

Pela fraca sensibilidade a papel e plástico, pode detectar a presença de alguns objetos dentro da embalagem. Pela alta sensibilidade à água, é muito usado para detecção de nível. Também é sensível à interferência eletromagnética, embora menos que o indutivo.

### Ultrassônico

Sensores ultrassônicos detectam a presença de um objeto pelo eco de um sinal ultrassônico (da ordem de 40 kHz). Também é muito usado para sensores contínuos de distância, já que o tempo que o eco demora pode ser usado para determinar a

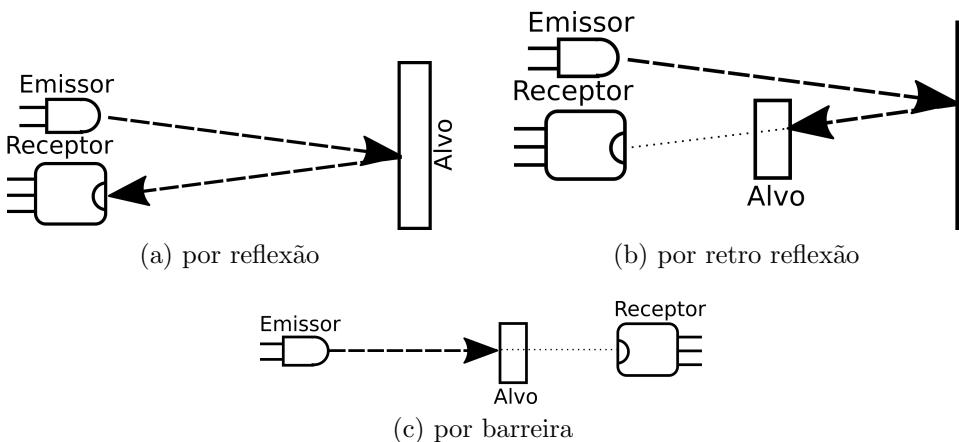


Figura 2.5: Princípio de funcionamento de diferentes configurações de sensor óptico.

distância até o alvo de forma linear.

Sensores ultrassônicos podem ser usados para detectar alvos em distâncias de até alguns metros, muito embora aí deva se ter o cuidado de que não haja outros elementos que possam gerar eco. É sensível não apenas à qualidade do material mas também à geometria do mesmo.

Outra limitação de sensores ultrassônicos é que eles têm uma distância mínima de trabalho. Se o alvo estiver muito próximo o sensor não consegue diferenciar o eco do sinal que ele ainda está gerando.

## Óptico

Sensores ópticos funcionam com um emissor e um receptor de luz. Tipicamente se usa luz infravermelha, pois a eletrônica baseada em silício é mais sensível a este comprimento de onda, o que barateia o custo. Normalmente o sinal luminoso é modulado num trem de pulsos, para diminuir a interferência do sol, cuja luz contém infravermelho mas não é modulada.

As configurações mais comuns deste tipo de sensor são:

**Por reflexão difusa** com o emissor do lado do receptor e detecta-se o alvo pelo seu reflexo. É o método usado na maioria dos smart phones para detectar que o usuário está com o telefone no ouvido e desligar a tela. Figura 2.5a.

**Por retro reflexão** que é o mesmo caso mas com um anteparo reflexivo atrás. Neste caso o alvo impede a passagem da luz. Figura 2.5b.

**Por barreira** que usa o emissor e receptor separados e detecta-se a oclusão do feixe óptico. O uso de um laser como emissor permite alcançar uma grande

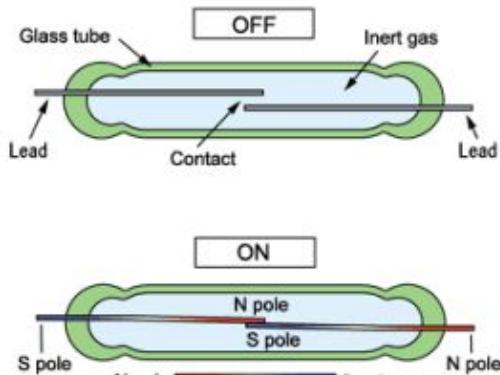


Figura 2.6: *Reed Switch*.

distância. Figura 2.5c.

Um uso interessante do sensor óptico de barreira é a chamada barreira laser, onde um conjunto de lasers passando por espelhos cercam um equipamento mais perigoso, de modo que qualquer pessoa ou coisa que acione a barreira enquanto o equipamento está atuando cause uma parada de emergência, diminuindo o risco do equipamento.

Além da luz ser imune a influência eletromagnética, o uso de fibras ópticas pode separar bastante a eletrônica do processo, permitindo o uso deste tipo de sensor em regiões onde não pode ter equipamentos elétricos.

## Magnético

Há dois tipos básicos de sensores magnéticos: sensores Hall e *reed switches*. Ambos detectam a presença de um campo magnético, normalmente causado pela aproximação de um imã.

O sensor Hall detecta o campo magnético pela influência do mesmo numa corrente elétrica, pelo chamado efeito Hall. É muito usado para medir a rotação em eixos com um imã acoplado ou em motores elétricos. É um sensor barato, de alta durabilidade e rápido tempo de resposta, porém apenas aplicável a alvos magnetizados.

O *reed switch* é composto de dois contatos de material ferromagnético normalmente separados, tal como mostra a figura 2.6. Um campo magnético perpendicular a estes contatos causam a magnetização dos mesmos, o que faz com que eles se atraiam e fechem contato.

O *reed switch* é bem mais barato que o Hall, mas tem pouca durabilidade e elevado tempo de resposta.

### 2.1.3 Chaves de processo

Chaves de processo são chaves elétricas que atuam quando uma grandeza do processo de fabricação (uma variável de processo) passa determinado nível. Por exemplo: temperatura do tanque 1 acima de 100 °C, nível do silo abaixo de 2 m, e assim por diante.

São exemplos de chaves de processo:

**bóias** que acionam um contato elétrico,

**sensor capacitivo** acionado pelo nível de água de um reservatório,

**chaves de fluxo** onde uma lingueta aciona uma chave eletromecânica se o fluxo passar de um determinado limite,

**termostatos** que acionam quando a temperatura passa de determinado nível, entre outros.

É cada vez mais comum trocar as chaves de processo por sensores contínuos e implementar o limite de chaveamento por software no controlador. Isto permite uma maior flexibilidade, pois variar o limite no software é mais simples.

## 2.2 Sensores Contínuos

Sensores contínuos usam algum princípio físico para a medição de alguma grandeza física. As grandes mais comumente medidas são temperatura, comprimento (nível, espessura, posição), vazão e pressão.

### 2.2.1 Pressão e força

Como pressão é força por área, se se consegue medir um, pode-se usar o mesmo princípio físico para medir outro.

Os principais princípios físicos para a medição de pressão ou força são:

**Coluna de líquido** que usa um tubo fino com um líquido. Ao se colocar o tubo de ponta cabeça, o líquido escorre deixando um vácuo. O comprimento do vácuo é proporcional à pressão externa.

Normalmente se usa como um indicador. Não é adequado para interfacear com circuitos elétricos.

**Deformação elástica** que utiliza a deformação de um elemento elástico para determinar a pressão. O elemento elástico em si pode ter vários formatos: formato de C, helicoidal, espiral, diafragma.

Pode ser usado tanto como mero indicador, acoplando o elemento elástico a um ponteiro (Bordon tipo C, por exemplo), ou como um transdutor acoplado a uma fita extensiométrica.

**Piezorresistência** é a variação da resistência com a tração ou compressão usada na fita extensiométrica, também conhecida por *strain gage*. Ao ser colado a um elemento elástico, permite transformar a deformação deste elemento num sinal elétrico. É bastante usada em balanças eletrônicas e também no monitoramento de estruturas metálicas.

**Piezoelétricidade** é o efeito apresentado por alguns cristais (normalmente se usa o quartzo) que geram um sinal elétrico ao serem submetidos a uma força e geram uma força quando lhe são aplicados sinais elétricos. Isto permite a construção de circuitos osciladores que variam a sua frequência de ressonância natural em função da força aplicada.

### 2.2.2 Temperatura

Entre as diversas maneiras de medir temperatura, destacam-se:

**Tubo capilar**, também conhecido como termômetro líquido. Usa um líquido que ao sofrer expansão térmica ocupa um tubo capilar, que permite visualizar a temperatura. Bom para visualização mas não adequado para automação, por não ser fácil gerar um sinal elétrico a partir dele.

**Termopar** Pode-se determinar a energia necessária para retirar elétrons de um determinado camada, o que se chama de *função trabalho* do material. Quando dois materiais diferentes se juntam, a diferença da função trabalho deles gera um potencial elétrico. Normalmente este potencial elétrico não é percebido pois ao se fazer um circuito fechado de diferentes materiais com todas as junções na mesma temperatura, estas diferenças acabam se anulando. Porém, se pegarmos dois fios condutores com diferentes funções trabalhos, unirmos uma ponta e colocarmos esta junção numa temperatura diferente, como num forno, por exemplo, aparece uma tensão que é função dos materiais usados e da diferença de temperatura. Este é o chamado efeito Seebeck:

$$E = (S_B - S_A) \cdot (T_2 - T_1),$$

onde E é a tensão gerada,  $S_A$  e  $S_B$  são os coeficientes de Seebeck dos materiais A e B e  $T_2$  e  $T_1$  são as temperaturas nas diferentes junções.

Logo o termopar mede a diferença de temperatura entre dois lugares e não a temperatura absoluta, precisando de um medidor auxiliar para medir a temperatura ambiente.

Tabela 2.1: Características de diversos tipos de termopar.

Tipo J	Tipo K	Tipo R	Tipo S	Tipo T
0 °C a 760 °C	0 °C a 1370 °C	0 °C a 1000 °C	0 °C a 1750 °C	-160 °C a 400 °C
±0,1 °C	±0,7 °C	±0,5 °C	±1,0 °C	±0,5 °C

Existem diversos tipos de termopar, formados por diferentes junções de metais. Há vários tipos padrões identificados por uma letra: J - ferro e constantan; K - Níquel-Cromo e Níquel-Alumínio; S - Platina e Ródio e Platina. Escolhe-se o tipo pela faixa de operação, precisão e custo. A tabela 2.1 mostra a faixa de operação e precisão de alguns tipos de termopar.

**Delta de tensão de junções** – pode-se usar dois transistores bipolares para gerar uma diferença de tensão ( $\Delta V_{BE}$ ) que é proporcional à temperatura absoluta (em Kelvin). É limitado à faixa de temperatura em que a eletrônica funciona: de aproximadamente -55 °C a 120 °C.

O grande uso da eletrônica fez com que o custo deste tipo de dispositivo caísse bastante, principalmente quando integrado em algum outro circuito. É muito usado em termômetros digitais, no controle de temperatura de processadores de computador e na compensação da temperatura ambiente em circuitos de medição de termopares.

**Termorresistor** utiliza a variação da resistência de um condutor com a temperatura. Usa um polinômio que relaciona a resistência com a temperatura, como:

$$R = R_0[1 + a \cdot T + b \cdot T^2],$$

onde T é a temperatura em Celsius e  $R_0$  é a resistência quando a temperatura é de 0 °C.

São usados metais inertes, tais como o níquel ou a platina. O nome do sensor é o símbolo do elemento químico usado seguido da resistência à 0 °C: Pt100, Ni500, etc.

**Pirômetro** ou termômetro de infravermelho usa a chamada radiação do corpo negro para determinar a temperatura. Todo objeto emite fôtons de comprimento de onda relacionado à temperatura do objeto. O pirômetro mede então o comprimento de onda destes fôtons, permitindo a medição da temperatura à distância.

Apesar da praticidade ainda são bastante caros, embora o custo venha diminuindo rapidamente. Hoje se usam também as câmeras térmicas, baseadas no mesmo princípio.

## 2.3 Características dos Instrumentos

Podemos separar entre características estáticas e dinâmicas. As principais características estáticas são:

**Sensibilidade - S** Relação entre um acréscimo na grandeza medida ( $x$ ) e o acréscimo correspondente na saída do sensor ( $Y$ ).

$$S = \frac{\partial y}{\partial x}$$

Se a curva é uma reta, a sensibilidade é constante. Além disso, se o 0 da entrada gera 0 na saída, pode ser chamado de ganho.

**Faixa de Operação** Faixa de valores de entrada para a qual o dispositivo funciona. Também chamado de Faixa nominal.

Algumas vezes, a Faixa é referida como sendo a diferença entre o valor máximo e mínimo do instrumento.

**Faixa de Operação de Saída** Faixa de valores de saída gerados pelo dispositivo. Também chamado de Fundo de escala.

Se o instrumento for linear, pode-se tirar a sensibilidade a partir da faixa de operação de entrada e de saída.

**Resolução** Menor variação da entrada que gera uma variação na saída.

Para instrumentos digitais, está relacionado ao número de bits do conversor analógico-digital usado.

**Exatidão** O quanto o valor médio dado pelo instrumento corresponde ao valor real.

**Precisão** O quanto as medições variam em torno do valor médio.

**Repetibilidade** Capacidade do instrumento de reproduzir as mesmas saídas quando as mesmas entradas são aplicadas, sob as mesmas condições. É calculado como sendo a amplitude máxima observada nas medições de saída em relação à faixa de operação de saída.

$$\text{Repetibilidade} = \frac{y_{max} - y_{min}}{\text{Faixa de operação de saída}}$$

**Reprodutibilidade** Grau de concordância entre os resultados das medições de uma mesma grandeza, onde as medições individuais são efetuadas variando-se uma ou mais das seguintes condições:

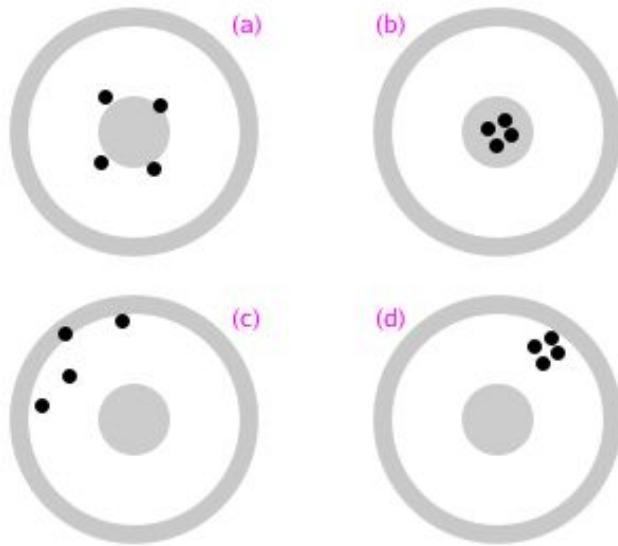


Figura 2.7: Representação de precisão e exatidão.a) Exato e Não Preciso. b) Exato e Preciso. c) Não Exato e Não Preciso. d) Não Exato e Preciso.

- Método de medição.
- Instrumento de medida.
- Observador.
- Outras variáveis (temperatura, pressão, etc.).

**Não linearidade** O quanto o instrumento diverge de uma relação linear entre entrada e saída.

Normalmente é medido como a pior diferença entre a resposta do instrumento e a resposta linear.

**Histerese** Quando há diferentes saídas para uma mesma entrada, conforme o valor da entrada for crescente ou decrescente.

**Banda morta** Região onde não existe variação na saída para uma determinada faixa de valores de entrada.

A resposta dinâmica do instrumento é relacionada a variação da saída ao longo do tempo em função da variação da entrada. Os principais parâmetros são:

**Velocidade de resposta** Medida pelo tempo que o instrumento demora para variar em função de uma mudança em degrau da entrada.

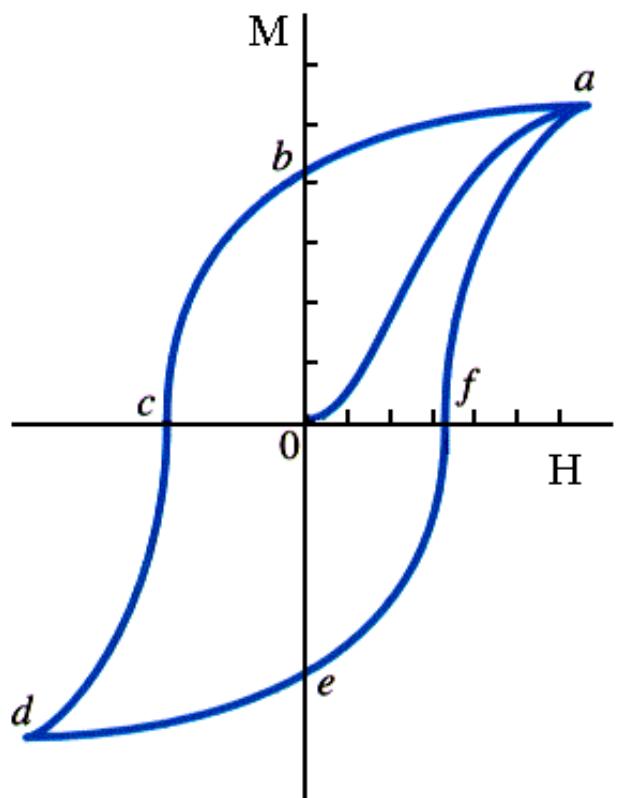


Figura 2.8: Representação de histerese.

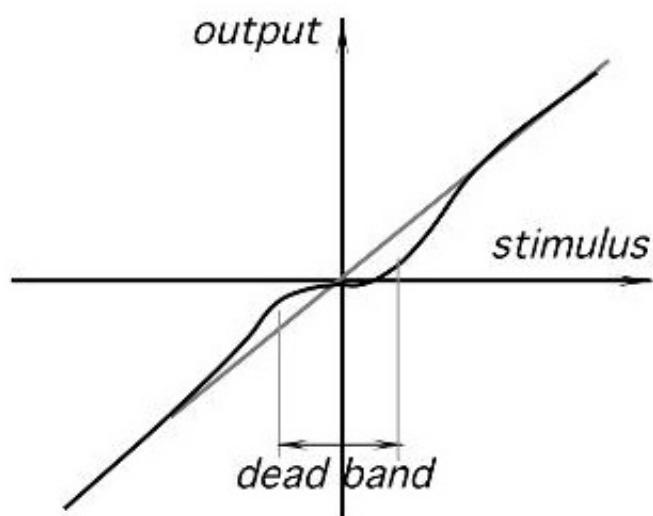


Figura 2.9: Representação de banda morta.

**Atraso** O quanto o instrumento demora para fazer qualquer mudança após uma variação da entrada.

**Overshoot** O quanto a medida passa do valor final ao ter uma alteração da entrada.

**Faixa de frequência** As frequências de entrada às quais o instrumento responde.

**Desvio** O quanto a saída muda ao longo do tempo, mesmo se a entrada estiver parada.

## 2.4 Transmissão de dados, aterramento e blindagem em instrumentação.

## 2.5 Diagrama P&I

Um PI&D (*Piping and Instrumentation Diagram*) é um diagrama que coloca de forma bem compacta várias informações sobre os instrumentos ligados a um processo. Um exemplo pode ser visto na figura 2.10. Neste diagrama, os instrumentos são representados por símbolos padrões, definidos pela norma ISA 51<sup>1</sup>.

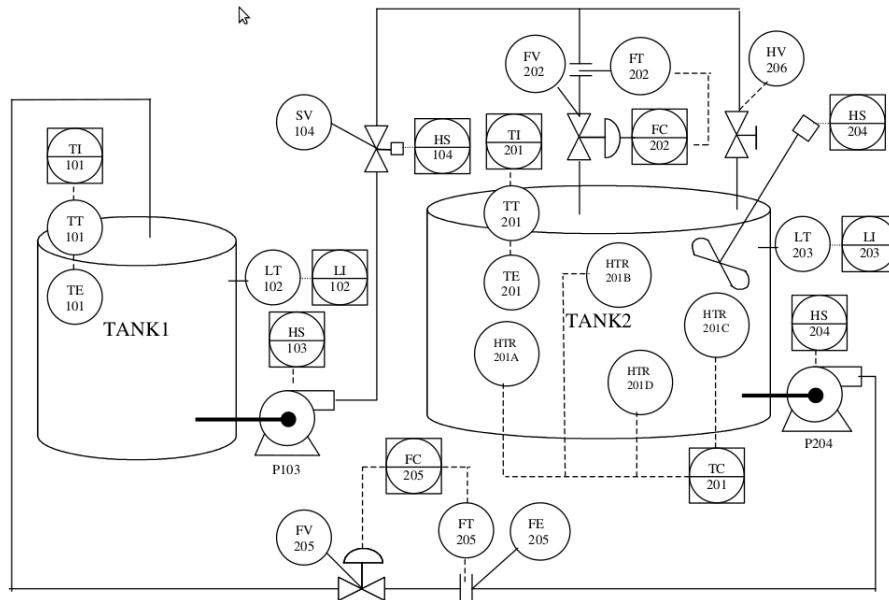


Figura 2.10: Exemplo de um diagrama P&I..

Cada instrumento é definido por um conjunto de letras e um número. A primeira letra indica que grandeza é relacionada àquele instrumento; a segunda letra em diante especifica a função daquele instrumento, com eventuais modificadores. Exemplos:

**TRC** – *Temperature Register and Controller*, controlador e registrador de temperatura.

**PDIC** – *Pressure (differential) Indicator and Controller*, controlador e indicador de pressão diferencial.

**FAH** – *High Flow Alarm*, alarme de vazão alta.

As funções mais comuns de um instrumento são:

**A** – **Alarme**.

---

<sup>1</sup>Vide Símbologia\_ISA.pdf na pasta da disciplina.

**C – Controlador.** Responsável por gerar o sinal de controle do atuador que afeta aquela variável.

**I – Indicador.** O instrumento permite ler o valor da variável nele.

**T – Transmissor.** O instrumento é um transdutor que gera um sinal padrão para outro instrumento.

O número é referente a malha de instrumentação. Normalmente é relacionado com a variável de processo que é controlada por aquele conjunto de instrumentos. Na própria figura 2.10, na parte superior, o conjunto de instrumentos FT202, FC202 e FV202 fazem a malha de controle que mede (FT202), controla (FC202) e atua (FV202) na vazão de água do tanque 1 para o 2.

Vários instrumentos tem símbolos próprios, tais como bombas e válvulas. Outros instrumentos são representados por um símbolo (figura 2.11), contendo o código do mesmo, que indicam o que é o instrumento e onde ele está localizado.

	Montado no campo	Montado no painel principal de controle	Montado atrás do painel principal de controle	Montado em painel local ou do equipamento
Instrumento Discreto	 Diâmetro = 12 mm			
Display compartilhado ( <i>Panel view</i> )				
Função executada no computador				
PLC	 Interface CLP/Campo/CLP	 Interface CLP/Supervisório/CLP	 Interface Interna (lógica)	 Interface CLP/Panel View/CLP

Figura 2.11: Explicação dos símbolos dos instrumentos..

### 2.5.1 Transmissão de dados

Os sinais transmitidos entre os instrumentos seguem um padrão. A linha que liga os instrumentos são decoradas de acordo com a forma com que a informação passa

de um instrumento para o outro, de acordo com a figura 2.12.

1) Conexão do processo, ligação mecânica ou suprimento ao instrumento.	—————
2) Sinal pneumático ou sinal indefinido para diagramas de processo.	// // //
3) Sinal elétrico.	— — — — —
4) Tubo capilar (sistema cheio).	X X X
5) Sinal hidráulico.	L L L
6) Sinal eletromagnético ou sônico (sem fios).	~~~~~
7) Sinal de software	—○—○—○—○—○—○—

Figura 2.12: Explicação das linhas do diagrama P&I..

Sinais comuns são:

**Pneumáticos.** Usam gás comprimido na faixa de 20 a 100 kPa. É usado em áreas que se tenha risco de explosão. Tem limitação com a distância e é difícil de detectar vazamentos.

**Hidráulico.** Usam óleo sobre pressão. São mais caros que sistemas pneumáticos porém são mais rápidos e podem ser usados para acionar equipamentos de grande porte.

**Elétrico.** Usam um sinal de 4 a 20 mA ou de 1 a 5 V. Em geral o de corrente, pois são menos sujeitos a interferências.

Os receptores deste tipo de sinal tem internamente um resistor de  $250\ \Omega$  em paralelo a sua entrada, o que permite que aceitem ambos tipos.

A razão de usar um valor mínimo maior que zero é para facilitar a detecção de falhas no transmissor, que normalmente fazem o sinal ir para 0.

**Sinais digitais.** São sinais elétricos modulados de acordo com um protocolo de comunicação digital. São usados nos chamados sensores inteligentes e estão ficando cada vez mais comuns.

Dependendo do protocolo usado, podem ter um custo total menor que o elétrico analógico por redução de cabeamento. Além disto, a comunicação digital permite passar outras informações, tais como configurações, dados de calibração e diagnósticos.

Existem muitos protocolos diferentes no mercado, com pouca compatibilidade entre eles.

**Rádio.** Normalmente usam sinais digitais modulados em rádio frequência. São muito usados para longas distâncias e para equipamentos móveis. Interferência e problemas de segurança diminuem seu uso mais geral.

# **Capítulo 3**

## **Controladores Lógico-Programáveis (CLP)**

### 3.1 Dispositivos Eletromecânicos

Há vários dispositivos eletromecânicos de interesse para a automação industrial, dos quais destacamos motores elétricos, solenóides, chaves e relês.

Destes, motores e solenóides são atuadores e chaves são sensores. Relês podem ser usados tanto como atuadores como sensores, embora sejam mais usados em atuadores.

Chaves já foram vistas junto com instrumentação, enquanto motores são assuntos de outras disciplinas e fogem do escopo desta disciplina.

Solenóides são basicamente eletroimãs com um eixo ferromagnético preso a uma mola. O eixo pode ter duas posições: uma mantida pela mola, que é a posição que ele assume quando o eletroimã não é acionado, e outra forçando a mola, quando o eletroimã é acionado. Ou seja: são dispositivos de acionamento binários.

Solenóides tipicamente fornecem uma movimentação de no máximo poucos centímetros a cargas de miligramas a dezenas de quilogramas. São muito usados para acionamento de válvulas - válvulas solenóides.

Relês eletromecânicos são, por construção, solenóides no qual o eixo faz abrir ou fechar contatos elétricos. Eles são chaves elétricas acionadas eletricamente. Hoje em dia existem relês baseados em eletrônica, chamados de relês de estado sólido, que tem a mesma função mas por princípios de funcionamento bem diferentes. Eletricamente pode-se visualizar um relê apenas como um indutor (a bobina do eletroimã) e um contato, e muitas vezes é desta forma que ele é representado em um diagrama esquemático (vide exemplo na Figura 3.1b).

O estado de repouso dos cantatos define o tipo básico do relê: normalmente aberto ou normalmente fechado. É comum relês que tenham 2 ou mais contatos acionadas por uma mesma bobina, inclusive podendo ser uma normalmente aberta e outra normalmente fechada, tal como o relê da figura 3.1a.

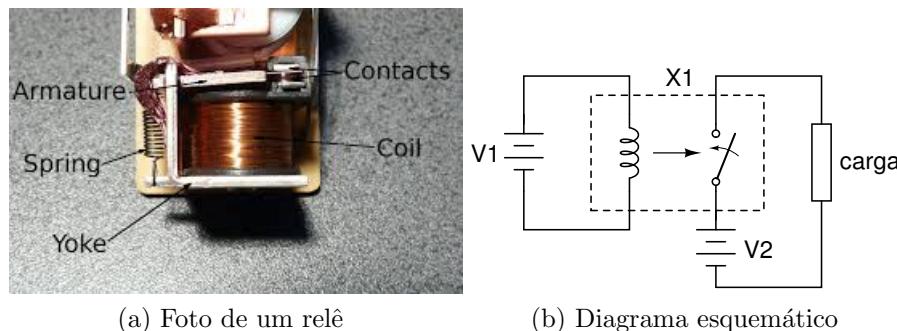


Figura 3.1: Foto de um relê e representação esquemática.

Uma grande utilidade dos relês vem do fato que a bobina de acionamento é eletricamente isolada dos cantatos, o que permite usar relês para:

- Controle de alta tensão e/ou potência através de baixa tensão e/ou potência  
– um sinal de mW pode comandar kW de potência através de um relê.
- Isolação e proteção do circuito de controle.
- Acionamento trifásico – um contator é um relê com três conjuntos isolados de contatos acionados pela mesma bobina.
- Implementação de lógica de controle – são os chamados circuitos chaveados.

### 3.2 De relês a CLPs

Um exemplo de aplicação de relê é mostrado na figura 3.2, onde 2 relês c1 e c2 são usados para o acionamento de um motor trifásico comandado pelas botoeiras b0, b1 e b2.

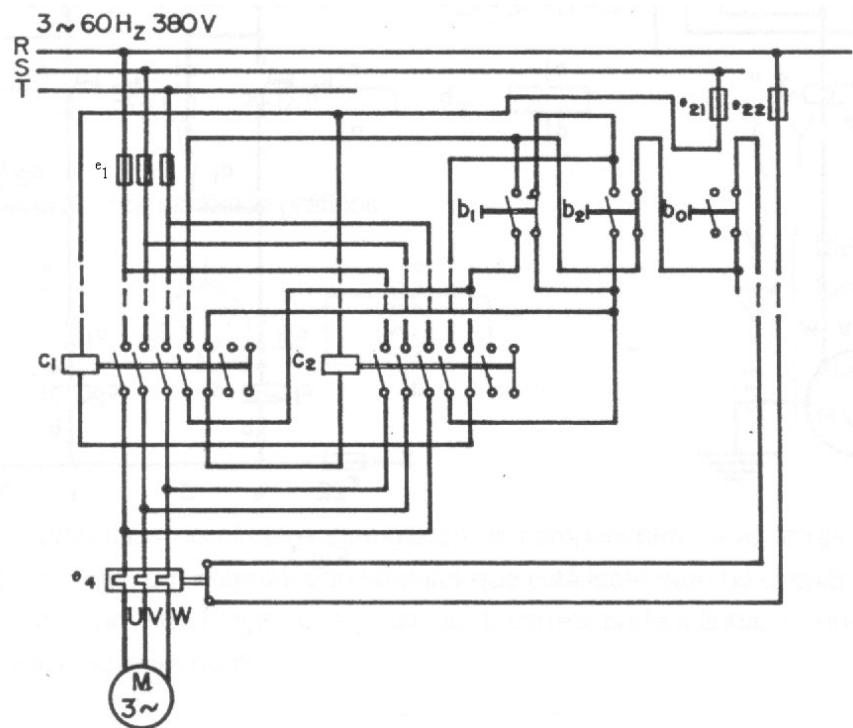


Figura 3.2: Circuito de acionamento de um motor trifásico por relê.

Em geral é mais fácil analisar um circuito do ponto de vista lógico separando a parte de acionamento da parte de controle, então é comum que o símbolo de um relê seja separado em duas partes: a bobina (o eletroímã) e o contato (a chave),

interligados pelo mesmo nome. Isto é exemplificado na figura 3.3, que redesenha o mesmo circuito da figura 3.2 separando a parte de controle da de acionamento, tornando o circuito bem menos convoluto. A ligação entre os diversos contatos e bobinas dos relês permite a realização de diversas funções interessantes para o controle de circuito. Tais circuitos ficaram conhecidos por *circuitos chaveados*.

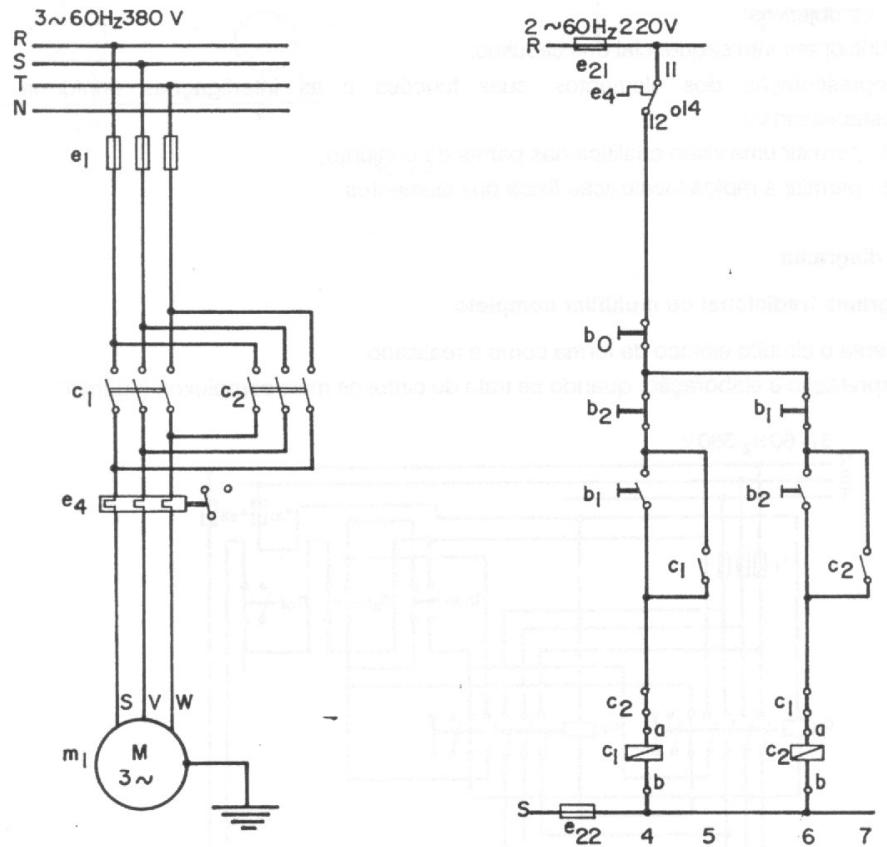


Figura 3.3: Mesmo circuito da figura 3.2, separando a parte de controle da de acionamento.

### 3.2.1 Diagrama Ladder

Os diagramas ladder são muito utilizados para representar circuitos com relês enfatizando a lógica da ligação. Neste tipo de diagrama as bobinas tem o símbolo  $\text{---}\text{O}$ , os contatos normalmente abertos são simbolizados por  $\text{---}\text{|}$  e os normalmente fechados por  $\text{---}\text{\|}$ .

A Figura 3.4 mostra o mesmo circuito de controle da Figura 3.3, só que agora

descrito em ladder. Com os circuitos conectados desta maneira, o diagrama fica parecendo uma escada; daí o nome<sup>1</sup>.

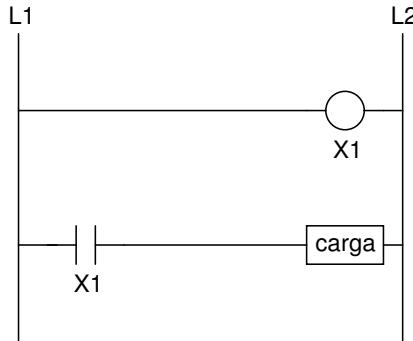


Figura 3.4: Exemplo de circuito de controle em ladder.

Logo de cara nota-se que não existem neste diagrama as tensões V1 e V2. Isto se dá pois neste diagrama considera-se que as tensões estão entre as duas barras L1 e L2 e abstrai-se a fonte de tensão. Isto lembra, de certo modo, a ligação real, com os elementos conectados entre os cabos de fase e o neutro.

Como exemplo, a figura 3.5 mostra diagramas ladder que acendem uma lâmpada apenas quando 2 relês são acionados, se qualquer um dos relês for acionado ou quando nenhum dos relês é acionado.

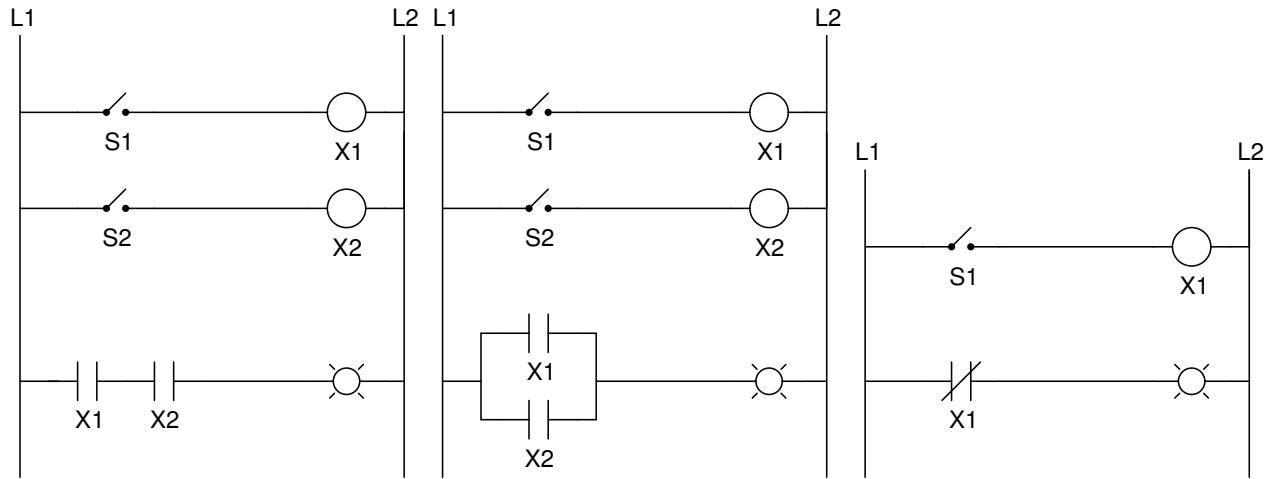
O engenheiro Claude Shannon descobriu em 1934 a relação entre os circuitos chaveados e a álgebra de Boole, que é um mapeamento da lógica em uma álgebra. Uma ligação em série realiza um AND lógico (uma multiplicação booleana), uma ligação paralela realiza um OR lógico (uma soma na álgebra de Boole) e o uso de um contato normalmente fechado realiza a inversão lógica, ou o NOT (representado por uma barra sobre a variável). A álgebra de Boole mostra que a combinação destas três operações, e portanto a combinação destes três circuitos, permite realizar qualquer condição lógica para o acionamento do que quer que seja, muito embora a aplicação direta dos postulados e teoremas desta álgebra não seja muito intuitiva.

Porém, apesar da facilidade deste diagrama, a montagem física ainda era bastante complexa, uma vez que o acionamento e os contatos de um relê são obrigatoriamente parte de um único dispositivo físico, e que para lógicas mais complexas essa fiação se tornava bastante complicada. Isto é ainda mais preocupante quando se precisa alterar a lógica de funcionamento, pois se tem que mexer no circuito como um todo.

Além disso, por ser um dispositivo eletromecânico, relês não tem um elevado tempo de vida: pode chavear talvez menos que 10 000 vezes, dependendo da tensão e corrente que suporta.

---

<sup>1</sup>em inglês *ladder* significa escada.



(a) Aciona apenas se ambos relês estiverem acionados.  
(b) Aciona se um ou outro relê estiver acionado.  
(c) Aciona se o relê não estiver acionado.

Figura 3.5: Exemplos de circuitos lógicos chaveados em ladder.

Em 1947, foi inventado o transistor, que age, como o relê, como uma chave elétrica controlada eletricamente. Principalmente pelo fato de não ter partes móveis, um transistor tem um tempo de vida muito mais longo que um relê, além de ter um chavemaneto mais rápido. Isto permitiu a construção de computadores eletrônicos muito mais rápidos, confiáveis e robustos que os a relê. O problema para o uso na automação é que os transistores não aguentam tensões e correntes tão elevadas quanto os relês.

Em 1968, a GM lançou uma especificação de um elemento de controle que pudesse substituir relês. Esta especificação pedia por:

- Facilidade de programação;
- Facilidade na manutenção (encaixe de módulos);
- Confiabilidade maior;
- Mais compacto;
- Possibilitar o envio de dados à central de processamento;
- Economicamente competitivo;
- Entradas em tensão alternada 115 V;
- Saídas em 115 V C.A. com mais que 2 A (operação das válvulas solenóides, comando para partida de motores e outros);

- Possibilitar a ampliação com um mínimo de alteração;
- Dotado de memória;

A partir desta especificação, foi desenvolvido pela Bedford Associates o primeiro PLC, posteriormente vendido pela marca MODICON – *M*ODular *D*Igital *C*ONtrol*l*er. Ele usa um microcontrolador para realizar a lógica e conta com dispositivos modulares de interface de entrada e saída. A figura 3.6 mostra a arquitetura típica de um CLP.

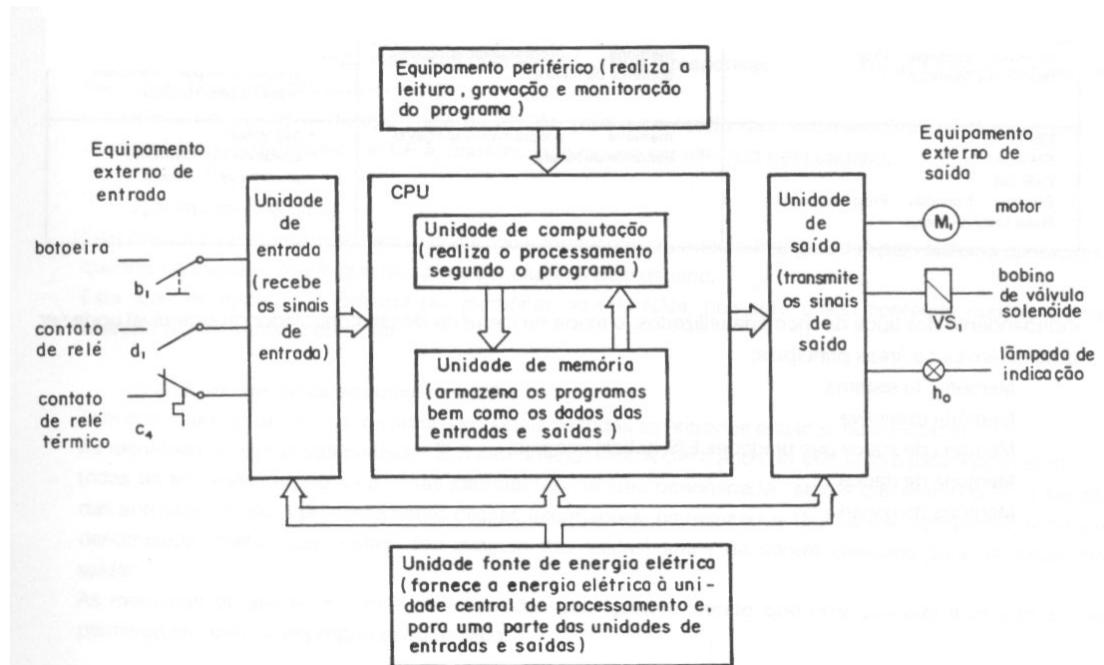
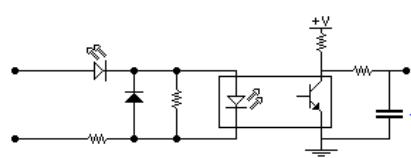
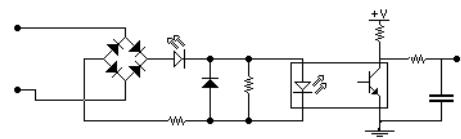


Figura 3.6: Arquitetura interna de um CLP.

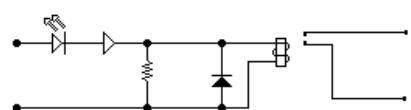
As entradas e saídas dos CLPs são isoladas eletricamente, uma vez que o microcontrolador não suporta altas tensões ou correntes. Tipicamente ainda são usados relês para a implementação das saídas digitais, mas cada vez mais se tem soluções totalmente eletrônicas. A Figura 3.7 mostra circuitos típicos de entrada e saída.



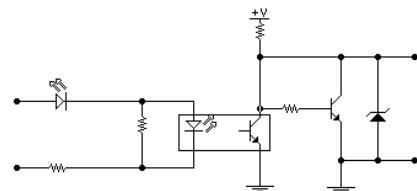
(a) Entrada de tensão contínua.



(b) Entrada de tensão alternada.



(c) Saída a relê.



(d) Saída de tensão contínua.

Figura 3.7: Entradas e saídas típicas de CLPs.

# Capítulo 4

## Programação - arduino

O arduino é uma plataforma de microcontrolador simplificada. O nome arduino refere-se a: uma placa com um microcontrolador Atmel, uma linguagem de programação e um ambiente de desenvolvimento para esta linguagem. E também um auto-proclamado rei da Itália, mas este último não importa para nós.

A placa Arduino tem um conector USB para se ligar ao computador. Isto serve tanto para programar o microcontrolador quanto para comunicação entre os 2. Existem várias versões do arduino, pois já que é um sistema *open-source* quem quiser pode fazer sua versão diferente da placa. Vamos nos referenciar aos arduinos UNO ou outras placas compatíveis com ele.

O arduino UNO tem 4 barras de pinos fêmeas para conexão com outros dispositivos: uma com tensões de alimentação (POWER), um com 6 entradas analógicas (ANALOG IN) e 2 com um total de 14 entradas e saídas digitais (DIGITAL).

A linguagem de programação arduino é basicamente a linguagem C++ para microcontroladores ATME, mas com algumas funções e definições facilitadoras. A principal diferença entre C++ e a linguagem arduino é que não existe a função main(), mas sim as funções (ou rotinas) setup() e loop(). A função setup() é executado apenas uma vez no momento que o arduino é ligado (ou resetado) e depois o código dentro da função loop() é executado repetidamente. Com isto o esqueleto de um programa arduino fica:

Código 4.1: Esqueleto de um programa arduino.

---

```
void setup() {
    //codigo a ser executado no inicio
}

void loop() {
    //codigo a ser executado repetidamente
}
```

---

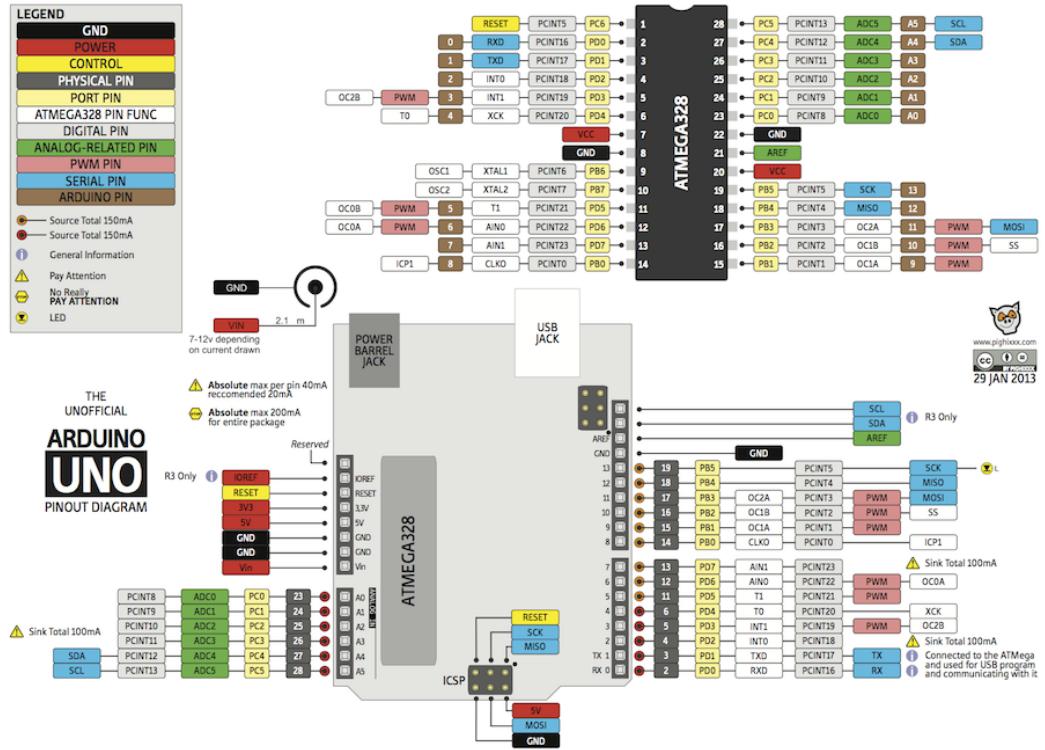


Figura 4.1: Pinagem do Arduino Uno

Lembrando que no arduino, como no C, tudo que tiver depois de // é comentário e é ignorado pelo compilador.

## 4.1 Piscando um led.

Vamos passar logo a um exemplo para analisar um programa arduino. As placas de arduino UNO já tem um led ligado ao pino 13, identificado por um L na placa. Podemos fazer um programa que faça este led piscar.

Código 4.2: Programa para piscar led.

```
void setup() {
    //codigo a ser executado no inicio
    pinMode(13,OUTPUT); //define o pino 13 como uma saida (led)
}

void loop() {
    //codigo a ser executado repetidamente
```

```

digitalWrite(13,LOW); //apaga o led
delay(500); //espera meio segundo
digitalWrite(13,HIGH); //acende o led
delay(500); //espera meio segundo
}

```

---

A chamada `pinMode(13, OUTPUT);` serve para definir que o pino 13 será uma saída. Obviamente isto só precisa ser feito no início do programa, logo está dentro de `setup()`. Se quiséssemos ter uma entrada digital, usariámos a mesma função, mas trocando `OUTPUT` por `INPUT`: `pinMode(pino,INPUT);`.

A função que define o valor de um pino digital é a `digitalWrite(pino,valor)`. Ela é chamada duas vezes no código 4.2 dentro de `loop()`, uma para apagar o led (gravando `LOW`) e outra para acendê-lo (gravando `HIGH`). `LOW` e `HIGH` são duas constantes, de valor 0 e 1, referentes ao zero e um lógico, respectivamente. Na prática, no sistema arduino, o `LOW` é uma tensão próxima a 0 V e o `HIGH` uma tensão próxima a 5 V.

Um detalhe é que o microcontrolador do arduino funciona numa velocidade de 8 ou 16 MHz (dependendo da versão), logo se colocássemos apenas as duas chamadas à função `digitalWrite` não veríamos o led piscar, mas teríamos a impressão que ele está aceso com metade da intensidade. Para vermos o led piscar é necessário colocar um atraso, que é justamente obtido pela função `delay(x)`, que gera um tempo morto de `x` milisegundos.

Se quiséssemos saber o valor de um pino digital que tivesse sido definido como entrada, a função seria `digitalRead(pino)`, que retornaria o valor digital naquele pino. Pode-se usar isto por exemplo, para fazer com que uma saída digital seja a cópia de uma entrada digital, como no código 4.3.

Código 4.3: Programa para acender um led em função de uma entrada digital.

```

const int pinoSaida = 13;
const int pinoEntrada = 10;

int valor;

void setup() {
    //codigo a ser executado no inicio
    pinMode(pinoSaida,OUTPUT); //define a saida (led)
    pinMode(pinoEntrada,INPUT); //define a entrada
}

void loop() {
    //codigo a ser executado repetidamente
    valor = digitalRead(pinoEntrada); //le a entrada
}

```

---

```

    digitalWrite(pinoSaida, valor); //e escreve na saída
}

```

---

No código 4.3 acrescentamos também algumas variáveis. Duas são constantes com os pinos usados. Elas facilitam a leitura do código e também facilitam caso posteriormente quisermos mudar os pinos utilizados. A outra variável armazena o valor lido da entrada, que depois é escrito na saída.

## 4.2 Sinais analógicos

Em contraste com os sinais digitais, os sinais analógicos são aqueles que podem assumir qualquer valor de tensão. No contexto do arduino, vamos por enquanto assumir que os sinais analógicos estão entre 0 V e 5 V.

Para valores analógicos, usamos as funções `analogWrite(pino,valor)` e `analogRead(pino)`, que, ao contrário das equivalentes digitais, são restritas a alguns pinos específicos. As entradas analógicas são identificadas pelos pinos ANALOG IN (A0 a A5 no arduino) e são ligadas a um conversor analógico/digital (A/D) do microcontrolador, que transforma estes sinais numa palavra binária de 10 bits. Como  $2^{10} = 1024$ , isto significa que a função `analogRead` retorna um valor entre 0 (para uma entrada de 0 V) e 1023 (para uma entrada de 5 V).

O arduino não tem um conversor D/A, logo a função `analogWrite` não gera um sinal analógico verdadeiro no pino. O que esta função faz é gerar um sinal modulado por largura de pulso - PWM (*Pulse Width Modulation*).

O sinal PWM é um trem de pulsos digital, com frequência da ordem de 500 Hz (no caso do arduino) cuja razão entre o tempo em alto e o período (conhecida como *duty cycle*) pode ser alterada pelo parâmetro passado, como mostra a figura 4.2. Se um sinal PWM é enviado a um pino com um led, ele piscará 500 vezes por segundo, o que é muito rápido para o olho humano, de modo que na prática o que se vê quando se varia o duty cycle de um sinal PWM que aciona um led é uma variação de sua intensidade. Logo um sinal PWM funciona, para muitas aplicações, como um sinal analógico.

Novamente, não são todos os pinos do arduino que conseguem gerar este sinal PWM, logo a função `analogWrite` está restrita aos pinos 3, 5, 6, 9, 10 e 11. Um outro detalhe que vale a pena ressaltar é que enquanto a função `analogRead` gera um valor entre 0 1023, a `analogWrite` recebe como parâmetro um valor entre 0 e 255 apenas.

Uma função útil do arduino para lidar com este tipo de situação é a função `map(valor, minIn, maxIn, minOut, maxOut)`, que faz uma transformação linear de valor de acordo com a seguinte equação:

$$(valor - minIn) \times \frac{maxOut - minOut}{maxIn - minIn} + minOut$$

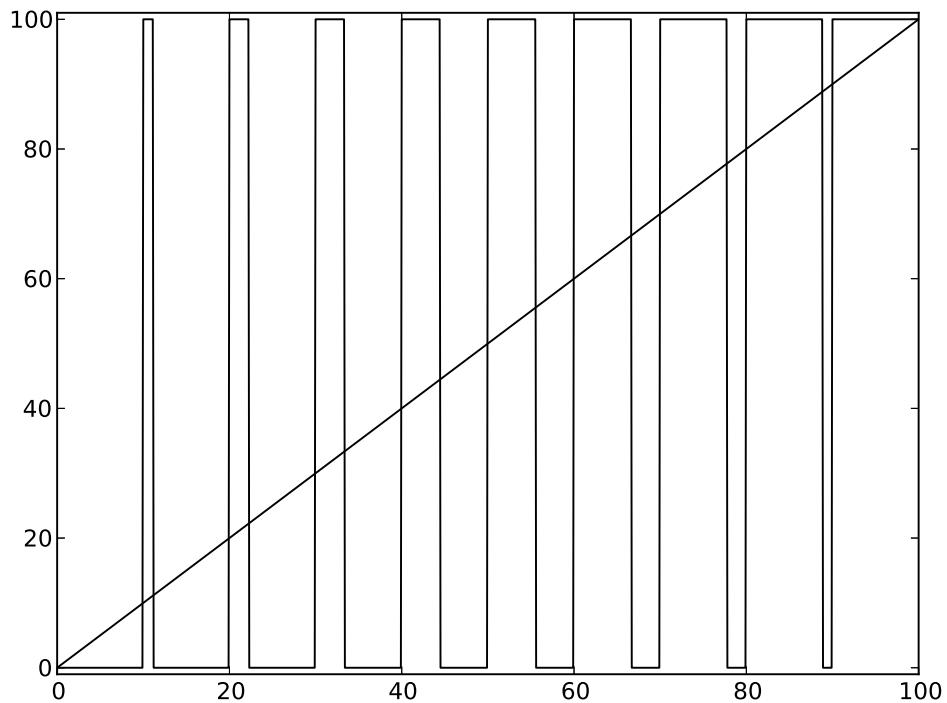


Figura 4.2: Reta modulada em largura de pulso (PWM).

A partir desta função, um código que leia o valor gerado pelo potenciômetro (em A0) e controle a intensidade do led no pino 5 poderia ser simplesmente:

---

```
analogWrite(5, map(analogRead(A0), 0, 1024, 0, 256));
```

---

Note que o valor lido pelo analogRead não precisa ser usado apenas na função analogWrite mas pode ser usado para outra finalidade, como por exemplo alterar um atraso.

### 4.3 Controle: for e if

Até aqui foram feitos programas puramente sequenciais, porém em vários momentos é interessante realizar operações repetidas vezes ou realizar algumas tarefas apenas em situações específicas. Para estes casos existem comandos como o **for** e o **if**. O comando for serve para tarefas repetidas. Por exemplo, se quiséssemos inicializar os pinos de 2 a 10 como saídas com valor LOW, poderíamos usar o seguinte código:

---

```
for (int i = 2; i < 11; i++) {
    pinMode(i, OUTPUT);
    digitalWrite(i, LOW);
}
```

---

O que este código faz é definir uma variável local *i* com valor inicial 2 depois ele checa se *i* é menor que 11, se for ele executa os comandos que estão entre as chaves “{” e “}”, incrementa a variável *i* (*i++*) e checa novamente. Quando *i* < 11 for falso, ele sai do laço.

O comando **if** executa um determinado código apenas se determinada condição for verdadeira. Por exemplo, para acender um led apenas se um sinal analógico for maior que a metade da escala ( $512 = 1024/2$ ) pode-se escrever:

---

```
if (analogRead(A0) >= 512) {
    digitalWrite(13, HIGH);
}
```

---

Note que este código apenas fará alguma coisa se a condição for verdadeira. Para fazer uma coisa OU outra usa-se o comando **else** após o **if**:

---

```
if (analogRead(A0) >= 512) {
    digitalWrite(13, HIGH);
} else {
    digitalWrite(13, LOW);
}
```

---

## 4.4 Comunicação com o computador

Como já dito, a conexão USB do arduino serve também para a comunicação do mesmo com o computador. Do lado do computador o arduino aparece como uma porta serial e o próprio programa contém um terminal serial pelo qual é possível se comunicar com o arduino. Do lado do arduino, os pinos 0 e 1 são os pinos transmissor e receptor ligados ao USB .

A programação é feita através do objeto **Serial**. Este objeto tem vários comandos, porém para nós os que interessam neste momento são:

**Serial.begin(baud)** Inicializa a comunicação serial na velocidade (baud rate) indicada.

**Serial.read()** Retorna o valor de um byte recebido ou -1 caso não tenha sido recebido nenhum byte.

**Serial.print(dado) e Serial.println(dado)** Se dado for um char ou uma string (texto), envia dado. Se dado for um número, envia este número como uma string. No caso de println, é acrescentada uma quebra de linha após dado.

**Serial.available()** Retorna o número de bytes recebidos que ainda não foram lidos.

# Capítulo 5

## CLPs – Definições e termos comuns

A IEC61131, antes conhecida por IEC1131 é uma norma técnica do *International Electrotechnical Commission*, que define um padrão para CLPs. Esta norma define vários requisitos de hardware e software para os CLPs, desde aspectos de segurança até guias de uso.

No CLP há um programa principal que executa a sequência mostrada na figura 5.1. Em condições normais, o CLP fica num loop executando as seguintes operações:

1. Lê as entradas, armazenando uma cópia das entradas em posições de memória.
2. Executa o código programado.
3. Atualiza as saídas, em função das entradas e do código do usuário.
4. Se comunica com outros dispositivos, se houverem. Estes outros dispositivos podem ser: computador, outro CLP, IHM, etc.
5. Faz um auto-teste.

A este loop se chama ciclo de execução, ciclo de *scan* ou *scan cycle*. O tempo que o CLP demora executando esta sequência é conhecido por *scan time*, ou tempo de execução. O ideal é que este tempo seja tão pequeno a ponto de poder ser desprezado em comparação aos tempos do processo. Neste contexto o código do usuário deve ser visto como tendo uma execução imediata. Obviamente isto não é verdade, já que o processador que executa este código é uma máquina de Turing – que é inherentemente sequencial. Em processos simples, o *scan time* pode ser da ordem de milissegundos ou dezenas de milissegundos, o que é bem rápido para

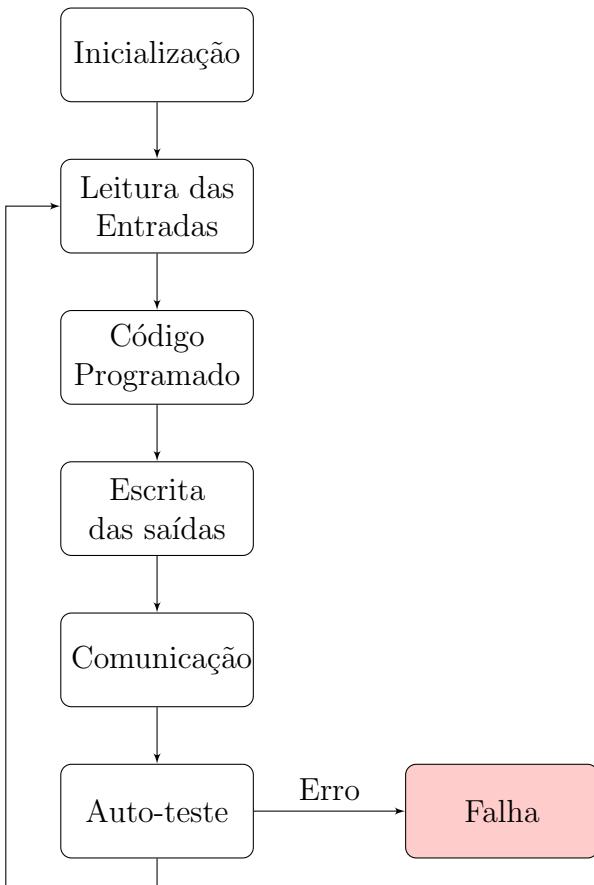


Figura 5.1: Sequência do programa principal de um CLP.

uma grande quantidade de processos. Isto é especialmente importante quando se trabalha com a linguagem ladder, cuja estrutura remete a um circuito elétrico que é inherentemente paralelo, porém o *scan time* depende do número de variáveis e do tamanho do código, podendo chegar a algumas centenas de milissegundos, onde a abstração de execução imediata pode deixar de funcionar.

Há porém, mesmo em processos simples, situações em que é necessário que o CLP atue mais rápido que no tempo do *scan time*, como por exemplo na leitura de um encoder, cujo sinal pode facilmente passar de kiloHertz. Para estes casos se usa um recurso dos microprocessadores e microcontroladores chamado interrupção, que permite interromper momentaneamente a execução do programa normal para executar uma tarefa mais prioritária, como por exemplo a leitura do encoder.

Outro uso de interrupção é um recurso de segurança chamado *watchdog timer*: um temporizador que é resetado uma vez a cada ciclo de execução. Se houver algum problema ou no código ou no hardware do microcontrolador que faça com que o ciclo de execução demore mais que o tempo ajustado no *watchdog timer*

(tipicamente da ordem de 200 ms), este causa uma interrupção que envia o CLP ao estado de Falha.

CLPs mais modernos ainda permitem um outro modo de funcionamento, que permite aplicar redundância para aumentar a robustez do sistema de controle: colocam-se 2 CLPs ligados aos mesmos módulos de entrada e saída e com o mesmo programa gravado. Os dois executam o mesmo programa, só que apenas um deles aciona as saídas. Se este falhar, o segundo assume seu lugar sem interrupção do controle. A isto se chama *hot swap*.

## 5.1 Variáveis

As variáveis dos CLPs e de seus módulos de controle são segmentadas de acordo com seu uso. Elas podem se referenciar a um valor dos seguintes tipos: entrada (I), saída (Q), memória interna (M) ou constante (K). O formato desta variável pode ser bit (X), byte (B) palavra de 16 bits, ou *word* (W), palavra de 32 bits, *double word* (D), de 64 bits, *long word* (L), ou número real em ponto flutuante (F). É possível – e aconselhável – definir símbolos que representem as memórias utilizadas num determinado programa, mas o símbolo deve apontar para uma memória descrita da seguinte forma:

**%{tipo}{formato}{pos}**

**pos** vem de posição, que pode ser simplesmente o endereço daquela variável dentro daquele segmento. Porém, ao se ter módulos, precisa-se colocar qual módulo está sendo endereçado, o que normalmente é feito pela posição do módulo no barramento (*rack*) em que está montado. Neste caso é comum o **pos** ser da forma:

**{rack}.{módulo}.{canal}**

Por exemplo:

**%IX100** – Bit de memória de entrada na posição 100.

**%I100** – Idem, pois para bit pode-se deixar o formato implícito.

**%QW122** – Word de memória de saída na posição 122.

**%IW10.1.21** – Word de memória de entrada no rack 10, módulo 1, canal 21.

**%IX0.1.0** – entrada binária 0 no módulo 1.

Obviamente, o módulo precisa ter fisicamente aquele tipo de variável para ser endereçado.

## 5.2 Programação

O código programado é definido pelo usuário e é composto por um conjunto de blocos em 5 possíveis linguagens, descritas na norma:

**IL – Instruction List** Uma linguagem de baixo nível de abstração, parecida com assembly.

**ST – Structured Text** Uma linguagem textual de alto nível, tal como C ou Pascal.

**LD – Ladder** Baseado no diagrama de circuitos a relê.

**FBD – Function Block Diagram** Parecido com um diagrama de portas lógicas: blocos com entradas e saídas.

**SFC – Sequential Function Chart** Baseado em Grafcet, uma extensão de máquina de estados.

Os blocos podem ser independentes, e neste caso eles são executados um após o outro na sequência em que estão na memória, ou podem haver blocos de subrotinas, que são executados quando chamados por outro bloco. É possível ter um bloco de uma linguagem chamando blocos de outras linguagens, o que aumenta bastante o poder descritivo destas linguagens. Além disso as variáveis são geralmente globais e acessíveis de todos os blocos, em qualquer linguagem.

Na prática, a maior parte da programação de um CLP é feita em Ladder, pois é a linguagem mais conhecidas pelos técnicos, responsáveis primeiros da manutenção dos equipamentos. Há também um grande uso de SFC devido a facilidade de usar esta linguagem para resolver problemas sequenciais. Além disso, o Grafcet (no qual é baseado o SFC) pode ser usado como uma ferramenta de desenvolvimento de um programa em ladder. Por conta disto este texto tratará principalmente destas duas linguagens. Porém antes será feita uma breve análise de Ladder e Instruction List, pois permite termos uma melhor ideia de como é o funcionamento interno de um CLP.

### 5.3 Ladder e Instruction List

Apesar de parecer com um circuito, um diagrama ladder é a descrição de um código, executado da esquerda para a direita e de cima para baixo. Na prática, o ladder tem uma estrutura muti parecida com o Instruction List, portanto é interessante compararmos os 2.

Para o funcionamento do ladder ou de IL, o CLP trabalha com uma estrutura chamada pilha, que é uma memória do tipo LIFO – *last in, first out*, que pode ser visualizada como uma pilha de papel, onde o último papel colocado é o primeiro retirado. A pilha tem duas funções básicas de acesso: a load (LD), que carrega um dado de alguma posição de memória para o topo da pilha, e a store (ST), que armazena a informação no topo da pilha em alguma posição de memória. Estas posições de memória podem ser memórias internas, entradas ou saídas de um CLP. Pos simplicidade, chamemos o topo da pilha de acumulador.

Na linguagem IL, as instruções lidam diretamente com o acumulador, que passa a funcionar como uma variável implícita. Por exemplo: um código IL que define uma saída como o E de duas entradas é simplesmente:

Código 5.1: Código IL de um E lógico

---

```
LD %I0
AND %I1
ST %Q0
```

---

No código 5.1, a instrução LD %I0 carrega a entrada 0 no acumulador. Logo depois a instrução AND %I1 faz o E lógico do valor na entrada 1 com o valor no acumulador (que foi copiado da entrada 0) e guarda o resultado no próprio acumulador. É como se retirasse o valor da pilha, fizesse a operação, e guardasse de volta na pilha. Agora o acumulador contém o resultado do E lógico entre %I0 e %I1. Por último, a instrução ST %Q0 passa o valor no acumulador para a saída 0.

Este código é equivalente ao ladder da figura 5.2. Neste caso, o primeiro contato carrega a entrada 0 no acumulador. Como o segundo contato está em série, ele equivale à AND do IL. A bobina então salva o valor presente no acumulador na saída 0. Note-se então a relação que há entre a bobina do ladder e o comando ST do IL e entre o contato do ladder e os comandos LD, AND e OR do IL. Não é então de se admirar que se tenha o equivalente aos contatos normalmente fechados em IL – basta acrescentar um N aos comandos IL: LDN, STN, ANDN, ORN.

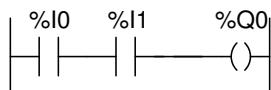


Figura 5.2: Ladder equivalente ao IL do código 5.1.

A pilha acaba sendo usada quando se tem caminhos paralelos no ladder. Analise-se por exemplo o ladder da figura 5.3. Um compilador ladder, analisando o diagrama da esquerda para a direita, vai carregar A, fazer o AND com B e, chegando à junção, checar de onde ela se origina, ele então vai carregar C, no topo da pilha acima do resultado do AND entre A e B que já está lá, fazer o AND com D, então fazer o OR entre os dois valores no topo da pilha e salvar o resultado em X. Um código IL equivalente é mostrado ao lado do ladder. Observe no IL que neste caso o comando OR não tem nenhum argumento; o que faz com que ele execute a operação com os dois últimos valores da pilha.

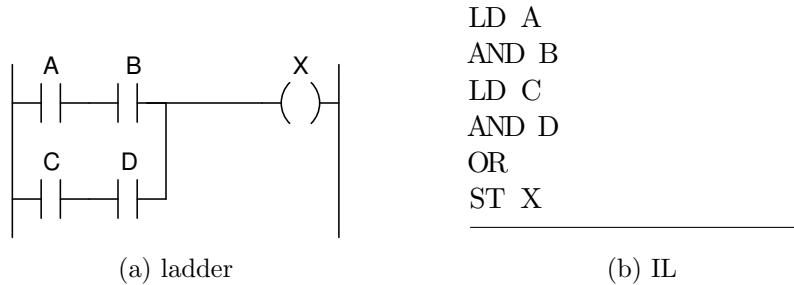


Figura 5.3: Exemplo de ladder e IL que usa a pilha para armazenar um valor intermediário.

## 5.4 Memória

É possível no ladder, assim como também o é usando relês, acionar uma bobina X com uma lógica que usa o contato X. Isto gera uma realimentação que tem umas propriedades interessantes. Veja por exemplo o ladder e equivalente IL da figura 5.4.

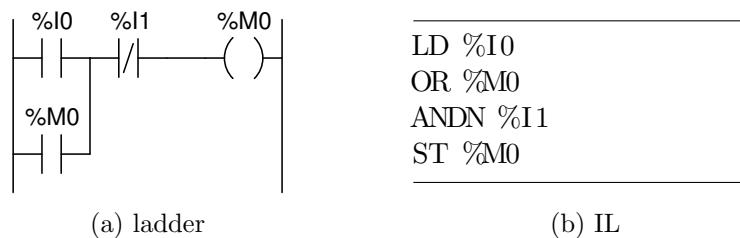


Figura 5.4: Ladder e IL implementando um latch.

Esta estrutura faz com que, se %I0 for acionado, acione-se %M0, que fica acionado daí por diante independente de %I0. %M0 só deixa de ser acionado se for acionado %I1. Isto faz com que %M0 implemente uma memória de 1 bit. Esta estrutura é a

base inicial das primeiras memórias de computadores eletromecânicos e é chamada de latch. Neste caso %I0 funciona como o sinal de SET e %I1 como o reset deste latch.

Por ser muito útil, a funcionalidade de latch acaba sendo criado diretamente, pelos comandos de set (S em IL e  $\neg(S)$  em ladder) e reset, (R em IL e  $\neg(R)$  em ladder). Estes comandos fazem com que o exemplo da figura 5.4 possa ser implementado mais facilmente, como mostra a figura 5.5.

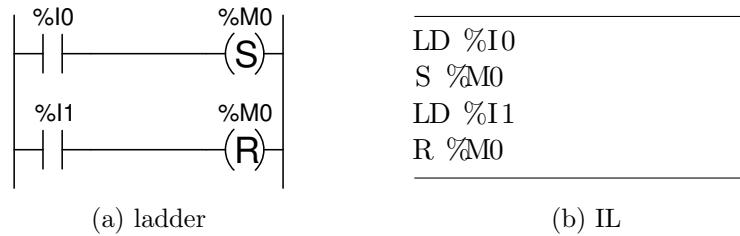


Figura 5.5: Ladder e IL implementando um latch com os comandos de set e reset.

#### 5.4.1 Subida e descida de uma entrada.

Considere que se queira inverter uma saída cada vez que se aperte um botão. A princípio bastaria fazer o ladder da figura 5.6, que ao apertar o botão, seta a saída se ela estiver 0 ou reseta se estiver 1. Porém esta solução não funciona, pois a cada ciclo a situação da saída estará trocada e ela vai acabar oscilando enquanto a chave estiver pressionada.

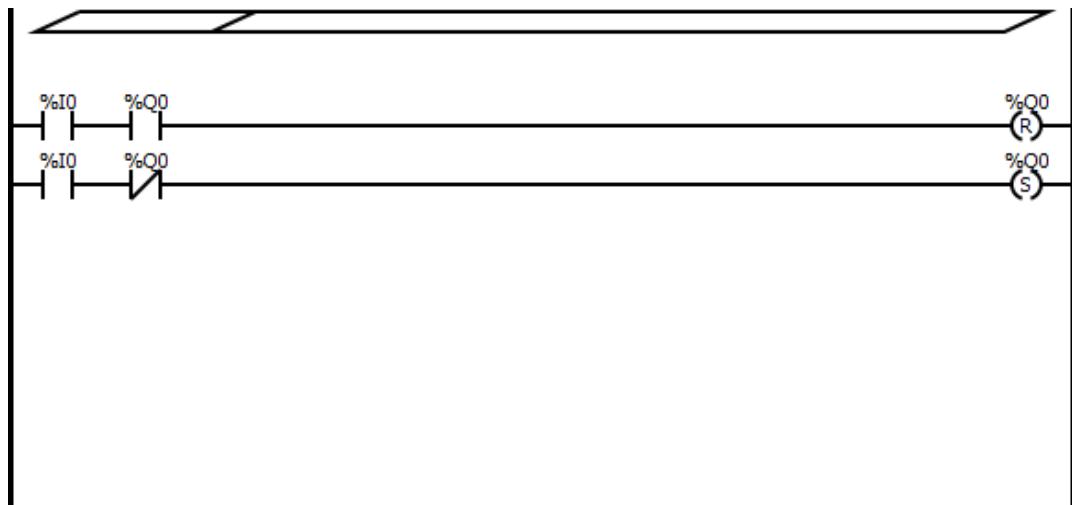


Figura 5.6: Troca de saída ao apertar um botão – solução errada.

É necessário então um modo de garantir a inversão da saída apenas da primeira vez que se detecta o botão apertado. Isto pode ser conseguindo armazenando o botão numa variável auxiliar *apenas ao final* do ciclo. Deste modo a variável auxiliar vai armazenar a entrada *antiga*, do ciclo anterior. Com isto podemos detectar a subida da entrada se a entrada estiver acionada mas sua cópia antiga não estiver. Isto é feito na figura 5.7.

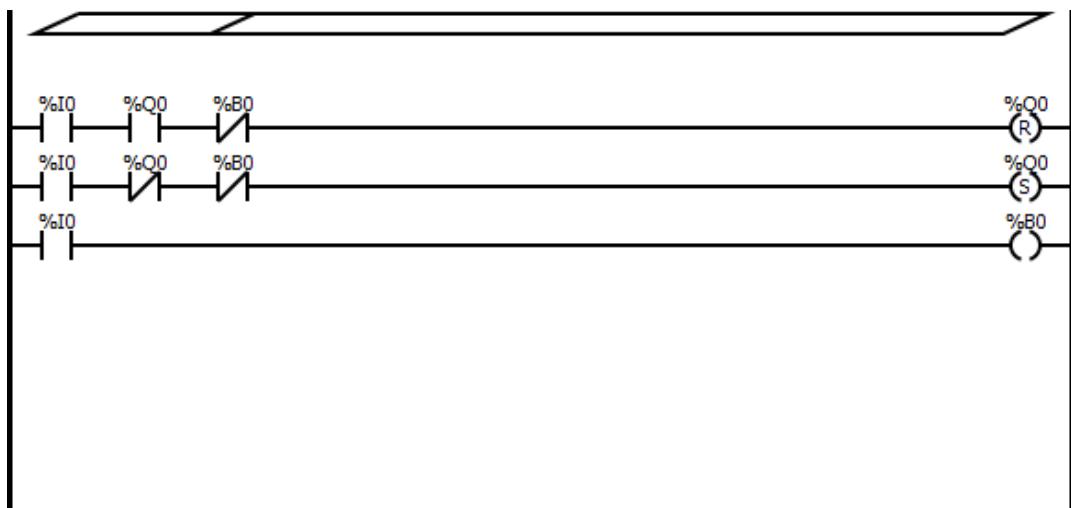


Figura 5.7: Troca de saída ao apertar um botão – solução correta.

A norma define um comando específico para ladder nesta situação, que é acionado apenas no primeiro ciclo que a variável é acionada. Este comando é identificado por um contato com um P no meio, muito embora o classicladder use um chapeuzinho, identificando a subida do sinal. Também há um comando específico para determinar a descida de um sinal, indicado ou por um N no meio do contato ou por um chapeuzinho descendo, no caso do classicladder.

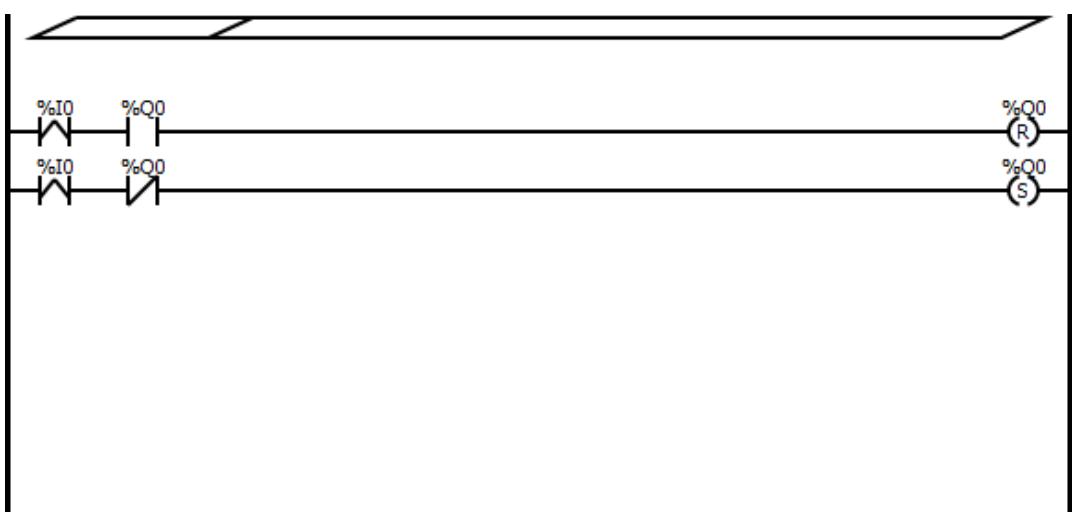


Figura 5.8: Troca de saída ao apertar um botão usando o detetor de subida.

## 5.5 Contadores

Contadores crescentes e decrescentes. Tem o nome %Cxx e conta com dois valores numéricos: %Cxx.V, o valor de contagem e %Cxx.P, o valor de preset. Na implementação do classicladder, o contador pode contar de 0 a 9999. O contador tem as seguintes entradas que modificam o valor de contagem V:

**R – reset** leva o contador para 0 quando acionado.

**P – preset** leva o contador para o valor de preset quando acionado.

**U – up** incrementa o contador na subida.

**D – down** decrementa o contador na subida.

O contador tem as seguintes saídas:

**D – done** acionado quando a contagem alcança o valor de preset (%Cxx.V = %Cxx.P).

**E – empty** acionado quando o contador está em zero e é decrementado (V vai para 9999). Também chamado de underflow.

**F – overflow** acionado quando o contador está em 9999 e é incrementado (V vai para 0).

As saídas também são disponíveis como variáveis, no formato %Cxx.X, onde xx é o número do contador e X a variável (V, P, D, E ou F). O limite de contagem pode ser resolvido com o uso destas variáveis. Por exemplo, pode-se fazer um contador que conta até 34357 da forma mostrada na figura 5.9.

### 5.5.1 Acesso às variáveis

As variáveis do contador, assim como outras variáveis numéricas, podem ser lidas ou definidas usando os blocos COMPARE ou OPERATE.

O bloco COMPARE retorna um valor booleano (acionado ou não acionado) definido por uma expressão matemática. Esta expressão pode conter:

- variáveis numéricas: %W0, %W3 ou símbolos definidos;
- operadores matemáticos padrões, +, -, \*, /, ^ (potência), % (módulo);
- comparações, =, <, >, <=, >=, <> (diferente);
- operadores lógicos, & (and), | (or);

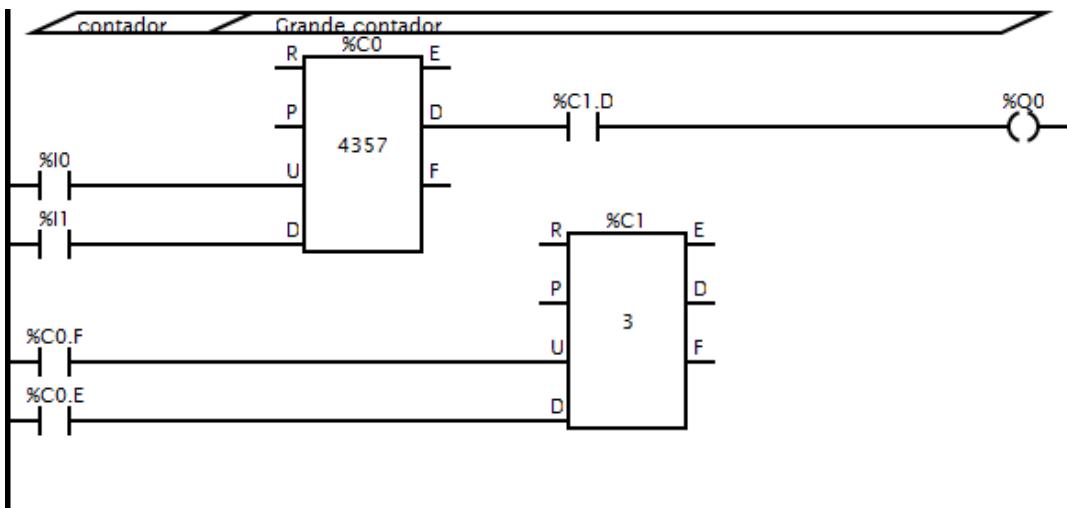


Figura 5.9: Contador com preset de 34357 feito com dois contadores.

- Funções: ABS (valor absoluta de um valor), MOY (média, em francês)/AVG (média de valores, separados por vírgula).

O bloco OPERATE é posicionado como uma bobina e recebe uma equação da forma: <variável>=valor. Se acionado acionada, como uma bobina, faz com que a variável receba o valor descrito.

Pode-se, por exemplo, definir que, quando o contador 2 contar 5, o seu valor de preset mude para 12, como mostra a figura 5.10.

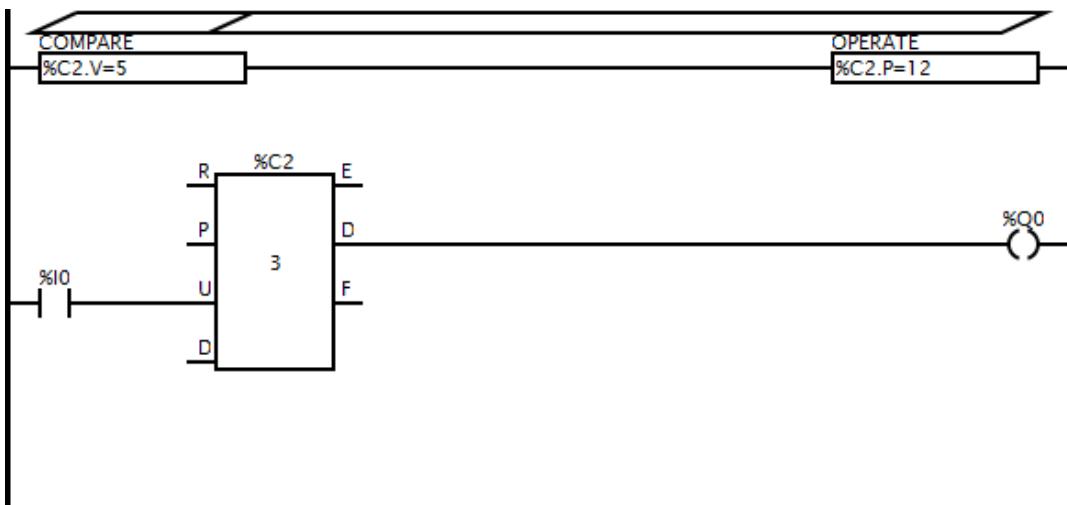


Figura 5.10: Contador tendo seu valor de preset mudado por um bloco OPERATE.

## 5.6 Temporizadores (timers)

O timer definido no padrão IEC61131, seguido pelo classicladder, pode ser de 3 tipos: TON, TOF ou TP. A figura 5.11 mostra um exemplo de cada um destes tipos.

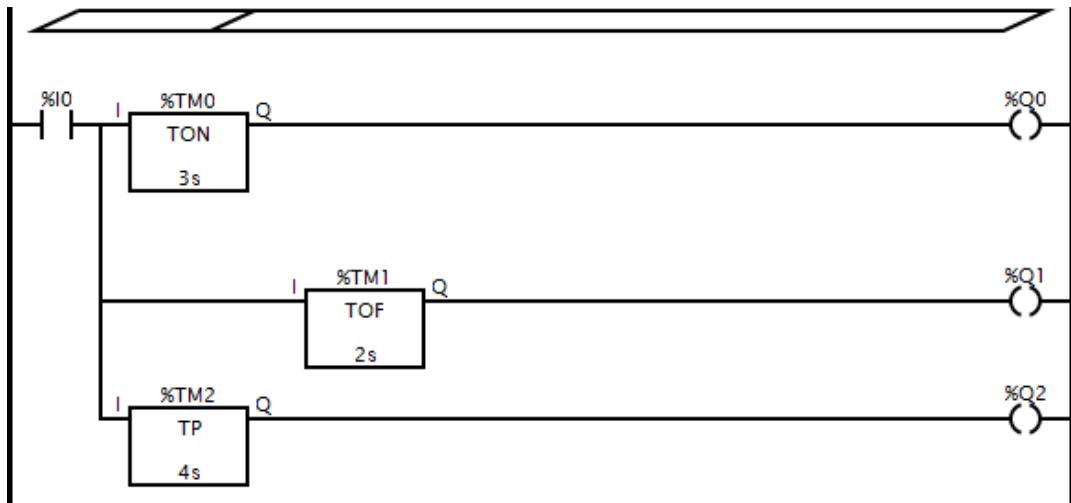


Figura 5.11: Timers TON, TOFF e TP.

O temporizador tipo TON causa um atraso no tempo de subida do sinal. O TOF causa um atraso no tempo de descida do sinal. O TP causa um pulso de tempo determinado iniciado pela subida do sinal de entrada. Isto pode ser visto na figura 5.12.

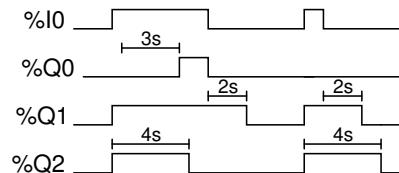


Figura 5.12: Diagrama temporal com os sinais em relação ao ladder da figura 5.11.

## 5.7 plcLib

A biblioteca plcLib (<http://www.electronics-micros.com/software-hardware/plclib-arduino/>) define um conjunto de funções muito parecidas com as da linguagem *Instruction List*, e que podem ser facilmente transcritas para a linguagem Ladder. Na configuração padrão, chamada pela função setupPLC() este sistema define 4 entradas (X0 a X3, usando A0 a A3) e quatro saídas (Y0 a Y3, usando 3, 5, 6 e 9).

Vamos usar um shield de arduino que já tem 3 botões de entrada, 4 leds, uma busina e um potenciômetro: o Multi-function shield (MFS). Usando o MFS, as entradas correspondem ao potenciômetro (A0) e aos 3 botões, logo estão bem adequadas. As saídas correspondem ao buzzer (3) e a três conectores de servo, porém não temos nenhuma saída ligada aos leds. Para definir todas entradas e saídas de uma vez, fazemos:

Código 5.2: Definição de função e constantes para usar o o Multi-function Shield com o plcLib.

---

```
// Definicao das constantes dos pinos
const int VR = A0;
const int S1 = A1;
const int S2 = A2;
const int S3 = A3;
const int D1 = 13;
const int D2 = 12;
const int D3 = 11;
const int D4 = 10;
const int LS1 = 3;
const int Q1 = 5;
const int Q2 = 6;
const int Q3 = 9;
const int Q4 = A5;

void setupShield(){ // Funcao para inicializar
    pinMode(VR,INPUT);
    pinMode(S1,INPUT);
    pinMode(S2,INPUT);
    pinMode(S3,INPUT);
    pinMode(D1, OUTPUT);
    pinMode(D2, OUTPUT);
    pinMode(D3, OUTPUT);
    pinMode(D4, OUTPUT);
    pinMode(Q1, OUTPUT);
    pinMode(Q2, OUTPUT);
```

```

pinMode(Q3, OUTPUT);
pinMode(Q4, OUTPUT);
pinMode(LS1, OUTPUT);
// Valores iniciais para apagar os leds e nao soar a buzina
digitalWrite(D1,HIGH);
digitalWrite(D2,HIGH);
digitalWrite(D3,HIGH);
digitalWrite(D4,HIGH);
digitalWrite(LS1,HIGH);
}

```

---

Deste modo basta chamarmos setupShield() no setup que configuramos nossa entradas e saídas. Infelizmente neste shield, os botões geram 0 quando apertados e 1 quando ligados e tanto os leds quanto a buzina acionam com nível lógico baixo, o que faz com que nossas entradas e saídas tenham que ser invertidas. Isto porém não impede o funcionamento do circuito.

### 5.7.1 Acumulador

Os CLPs trabalham com o conceito de acumulador (que na implementação do plcLib tem o nome scanValue), que é uma variável implícita. As funções pegam implicitamente o valor a ser trabalhado do acumulador e salvam o resultado de volta nele. O plcLib define as funções in(entrada), inNot(entrada), out(saida) e outNot(saida), equivalentes às LD, LDN, ST e STN, de Instruction List, respectivamente. As duas primeiras pegam o valor de uma entrada e guardam no acumulador, enquanto que as 2 últimas pegam o valor do acumulador e colocam na saída. O Not (N) no final indica que estes valores são invertidos. Vejamos como exemplo o código 5.3:

Código 5.3: Código simples de leitura de cópia de entradas para saídas.

---

```

void loop() {
    in(S1);      // Le chave 1
    out(D1);     // manda para led 1

    inNot(S2);   // Le chave 2 (invertida)
    out(D2);     // manda para led 2

    inNot(S3);   // Le chave 3 (invertida)
    outNot(D3);  // manda para led 2 (invertido)
}

```

---

Neste caso, D1 fica sendo uma cópia de S1, D2 o valor invertido de S2. Dá para imaginar o que acontece com D3.

É bom lembrar que no multi-function shield as chaves são conectadas ao terra, enquanto que os leds são conectados à 5V. Isto faz com que quando apertada, a chave gera um 0 lógico (não um 1) e que o led acende quando se envia um 0 lógico (e não um 1). Por isto é interessante para trabalharmos com este shield fazermos `inNot(Sn)`, que resulta em 1 se a chave estiver apertada, e `outNot(Ln)`, que acende o led quando o valor no acumulador for 1.

### 5.7.2 Funções lógicas

Mas apenas carregar um único valor não é tão interessante. Bem mais útil é montar uma relação lógica entre valores. Para isto servem as funções `andBit`, `orBit`, `xorBit`, `andNotBit` e `orNotBit`. Cada uma destas funções faz uma operação lógica entre o acumulador e a variável indicada, salvando o resultado no acumulador. A tabela 5.1 mostra as tabelas verdadeiras de cada uma delas.

Tabela 5.1: Tabela verdade das funções lógicas definidas em `plcLib`.

scanValue	ent	scanValue (após operação)				
		andBit	orBit	xorBit	andNotBit	orNotBit
0	0	0	0	0	0	1
0	1	0	1	1	0	0
1	0	0	1	1	1	1
1	1	1	1	0	0	1

Colocando estas funções em cascata é possível criar lógicas bem interessantes, como por exemplo, soar o alarme se S1 ou S2 forem pressionados ao mesmo tempo que S3 for pressionado.

Código 5.4: Aciona a buzina em função das chaves.

---

```
inNot(S1); // Le se chave 1 apertada
orNotBit(S2); // Le se chave 2 apertada e faz um OU logico
andNotBit(S3); // Le se chave 3 apertada e faz um E logico
outNot(LS1); // Se condicao satisfeita, aciona a buzina
```

---

O problema do uso do acumulador é que às vezes uma situação exige uma lógica mais complexa do que o acumulador permite. Por exemplo: como acender D2 se a maioria dos botões estiver pressionado? (Problema do voto majoritário) Existem 2 formas de resolver este problema: pilha ou variáveis;

A pilha é como a maioria dos CLPs trabalham: ao invés de ter um único acumulador, ele tem um número finito de posições de memória que ele usa e todo comando usa implicitamente o topo da pilha. No caso da implementação do `plcLib`,

a pilha pode ser definida separadamente como um objeto da classe Stack. Logo, se fizermos Stack pilha; definimos uma pilha de nome pilha.

Os principais comandos de acesso à pilha são o push(), que passam o valor do acumulador para o topo da pilha, e o pop(), que tira o valor do topo da pilha e passa para o acumulador. Outros comandos úteis para a lógica são o andBlock(), que faz o E do valor no topo da pilha com o acumulador, salvando no acumulador e o orBlock(), que faz a mesma coisa com o OU lógico. Usando a pilha é possível resolver o problema da maioria dos botões da seguinte forma:

---

Código 5.5: Solução do voto majoritário com uso de pilha.

```
void loop(){\lstinline|
    inNot(S1);
    andNotBit(S2);
    andBit(S3);
    pilha.push();
    inNot(S1);
    andBit(S2);
    andNotBit(S3);
    pilha.push();
    in(S1);
    andNotBit(S2);
    andNotBit(S3);
    pilha.push();
    inNot(S1);
    andNotBit(S2);
    andNotBit(S3);
    pilha.orBlock();
    pilha.orBlock();
    pilha.orBlock();
    outNot(D2);
}|
```

---

Outra possibilidade é simplesmente definir variáveis auxiliares para receberem os valores intermediários. Neste caso as variáveis precisam ser do tipo **unsigned int**.

---

Código 5.6: Solução do voto majoritário com uso de variável.

```
int temp1;
void loop(){
    inNot(S1);
    andNotBit(S2);
    andBit(S3);
    out(temp1);
    inNot(S1);
```

```

    andBit(S2);
    andNotBit(S3);
    orBit(temp1);
    out(temp1);
    in(S1);
    andNotBit(S2);
    andNotBit(S3);
    orBit(temp1);
    out(temp1);
    inNot(S1);
    andNotBit(S2);
    andNotBit(S3);
    pilha.orBlock();
    pilha.orBlock();
    pilha.orBlock();
    orBit(temp1);
    outNot(D1);
}

```

---

Note-se que não se usou a melhor estratégia lógica para ambas soluções apresentadas. Fica como exercício resolver este problema de forma mais compacta.

### 5.7.3 Memória

Até o momento usamos apenas a chamada lógica combinacional, onde a saída depende apenas da entrada. Porém é bem comum a situação de querermos que a saída fique num determinado estado em função da sua história pregressa.

Um exemplo simples: Como acionar LS1 apertando S1 e pará-lo apertando S2? O estado de LS1 depende então de qual foi o último botão apertado.

É possível implementar este tipo de memória a partir de lógica combinacional usando realimentação. Ou seja, devemos ler a saída do sistema como sendo entrada, o que, apesar de estranho, é perfeitamente possível. O código 5.7 mostra justamente esta implementação.

Código 5.7: Implementação de latch com lógica combinacional.

---

```

inNot(S1);      // Se chave S1 apertada
orNotBit(LS1);  // ou se LS1 ja esta acionado
andBit(S2);     // mas apenas se S2 nao estiver apertada
outNot(LS1);    // aciona LS1

```

---

Como esta função é bastante utilizada, ela recebeu o nome de *latch* (tranca, ferrolho) e tem funções específicas, para deixar uma variável em 1 (função `set()`) ou 0 (`reset`) dali em diante, como mostrado no código 5.8.

---

Código 5.8: Funções para uso de latch.

---

```
inNot(S1);      // Se chave S1 apertada
reset(LS1);     // Seta LS1 (aciona dai em diante)
andBit(S2);     // Se S2 estiver apertada
set(LS1);       // Reseta LS1 (desliga dai em diante)
```

---

### 5.7.4 Temporizadores

O arduino já tem uma função padrão `delay`, que causa a paralisação da execução por um determinado tempo. O problema de `delay` é que ele para o programa todo, o que pode ocasionar problemas até de segurança.

Os temporizadores definidos na `plcLib` permitem gerarmos atrasos em sinais específicos, sem mexer nos demais. O `timerOn` atrasa a subida de um sinal, enquanto que o `timerOff` atrasa a descida de um sinal, como pode ser visto no exemplo abaixo. Outro tipo de temporizador é o `timerPulse`, que na subida do sinal de entrada gera um pulso por um tempo determinado. `timerPulse` e `timerOff` são bem parecidos, com a diferença que o último começa a medir o tempo a partir da descida da entrada, enquanto que o primeiro mede a partir da subida da entrada. Cada função destas recebe como parâmetro a variável (do tipo `unsigned long`) que contará o tempo e o tempo final a ser contado. Logo para cada temporizador é necessário ter uma variável para armazenar o tempo utilizado.

---

Código 5.9: Exemplos de uso de temporizadores.

---

```
unsigned long TIMER0 = 0;
unsigned long TIMER1 = 0;
unsigned long TIMEa = 0;
void loop(){
    inNot(S1);      // Se chave S1 apertada
    timerOn(TIMER0,2000); // atrasa subida por 2 segundos
    outNot(D1);
    inNot(S2);      // Se chave S2 apertada
    timerOff(TIMER1,2000); // atrasa descida por 2 segundos
    outNot(D2);
    inNot(S3);      // Se chave S3 apertada
    timerPulse(TIMERa, 2000); // gera pulso de 2 segundos
    outNot(D3);
}
```

---

Um outro temporizador interessante é o `timerCycle`, que enquanto o acumulador for 1, gera um sinal alternado definido por 2 tempos: o tempo em alto e o tempo em baixo. Muito útil para alarmes.

Código 5.10: Exemplos de uso timerCycle.

---

```
unsigned long TempoLOW = 0;
unsigned long TempoHIGH = 0;
void loop(){
    inNot(S1);
    timerCycle(TempoLOW,1300,TempoHIGH,100);
    outNot(LS1);
}
```

---

### 5.7.5 Contadores

Um contador incrementa ou decremente uma variável de acordo com o número de eventos que recebe. Na implementação do plcLib, estes eventos são subidas (ir de 0 para 1) em suas entradas.

Para os contadores, cria-se um objeto do tipo Counter, que tem um valor interno de preset presetValue, uma variável de contagem count, 4 entradas countUp, countDown, preset, clear e 2 saídas upperQ e lowerQ. As regras do contador são:

- Uma subida em countUp incrementa o valor de count, se for menor que preset.
- Uma subida em countDown decrementa o valor de count, se for maior que 0.
- Uma subida em clear faz count igual a 0.
- Uma subida em preset faz count igual a presetValue.
- Se count for igual a 0, lowerQ retorna 1.
- se count for igual a preset, upperQ retorna 1.

Código 5.11: Exemplo de uso do contador.

---

```
Counter ctr(10);           // Contador na faixa 0-10, começando em zero
unsigned long TIMER0 = 0;   //
unsigned long TIMER1 = 0;   //

void loop() {
    inNot(S1);             // Se S1 apertada
    timerOn(TIMER0, 10);    // 10 ms de atraso para debounce
    ctr.countUp();          // incrementa ctr.
    inNot(S2);              // Se S2 apertada
    timerOn(TIMER1, 10);    // 10 ms de atraso para debounce
```

```
ctr.countDown();           // decrementa ctr.  
inNot(S3);               // Se S3 apertada  
ctr.clear();              // zera o contador.  
ctr.lowerQ();             // Se zero,  
outNot(D1);  
ctr.upperQ();             // Se 10,  
outNot(D2);  
}
```

---

É possível ainda criar um contador que inicializa no valor de preset. Tomando o exemplo do código 5.11, ao invés de fazer Counter ctr(10);, faria-se Counter ctr(10,1);

# Capítulo 6

## Linguagem grafset

GRAphe Fonctionnel de Commande Etape/Transition - Grafo funcional de comando etapa/transição, é um formalismo matemático tal como uma máquina de etapas. Tem origem na chamada máquina de etapas, mas com mais funcionalidades, tais como a possibilidade de ter etapas em paralelo e de ter macro-etapas.

Um diagrama grafset é composto dos seguintes elementos: etapas, transições, ações, divergências e convergências E e divergências e convergências OU.

O básico do grafset é a sequência etapa-transição-etapa, como mostra na figura 6.1. Ela apresenta um diagrama grafset com 3 etapas (os quadrados) e três transições (os retângulos verde e vermelhos). Etapas e transições podem estar ativas ou não. Etapas ativas são representadas com uma ficha dentro delas (vide Etapa 1 na figura), enquanto que transições ativas são em geral representadas por uma mudança de cor. Neste exemplo uma transição ativa é representada pela cor verde, enquanto uma inativa é representada pela cor vermelha.

A ativação de uma transição é diretamente relacionada a uma condição lógica associada a mesma: quando verdadeiras, as ativam. Já a ativação das etapas depende da dinâmica do sistema. No exemplo da figura 6.1, A Etapa 0 é uma etapa inicial (representado pela linha dupla), significando que o sistema inicia com a Etapa 0 ativa, mas não as demais. No momento mostrado na figura 6.1, a Etapa 1 é que está ativa, indicado pela ficha.

A evolução do sistema descrito em grafset é dada pelo disparo das transições. Um transição dispara quando:

- ela está ativa (ou seja, a condição ligada a ela é verdadeira) e
- todos as etapas anteriores a ela estão ativas.

Quando uma transição dispara ela desativa todos os etapas anteriores a ela (as etapas acima) e ativa todos as posteriores (as abaixo). Por exemplo, se na figura 6.1 a condição A se tornasse verdadeira, como a única etapa anterior (a Etapa 1)

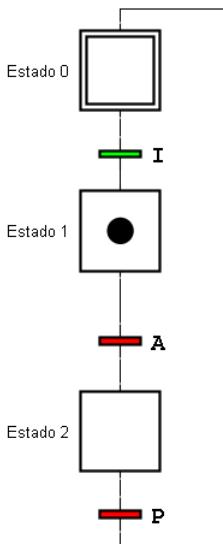


Figura 6.1: Exemplo de um diagrama grafcel simples.

está ativa, esta transição dispararia, desativando a Etapa 1 e ativando a Etapa 2. Esta sequência é mostrada na figura 6.2.

## 6.1 Divergências e convergências

No grafcel, pode-se ter uma etapa com mais de uma transição de saída, que pode levar para etapas diferentes de acordo com qual transição dispare primeiro. A isto se chama uma *divergência OU*, pois separa em dois caminhos excludentes. Da mesma forma, duas (ou mais) transições podem levar para o mesmo etapa, o que se chama de convergência OU. A figura 6.3 mostra um exemplo em que o sistema pode evoluir por dois caminhos diferentes, com o uso de divergência e convergência OU.

Note-se na figura 6.3 que a Etapa 1 é obrigatoriamente transitória, pois ou A é verdadeiro e o sistema evolui imediatamente para o etapa 2 ou A é falso e o sistema ativa o etapa 3. É possível definir transições que estejam ativas ao mesmo tempo, neste caso o sistema evolui para a transição com maior prioridade – ou a mais a esquerda ou se tiver uma prioridade explícita.

Diferente de uma máquina de estados, no grafcel é permitido que uma transição acione mais que uma etapa. Neste caso o sistema se divide em caminhos paralelos. Isto é chamado de divergência E e indicado por duas linhas horizontais, para diferenciar da divergência OU. Da mesma forma, uma convergência E liga várias etapas a uma única transição, transformando caminhos paralelos num único caminho. A figura 6.4 mostra um exemplo de divergência e convergência E.

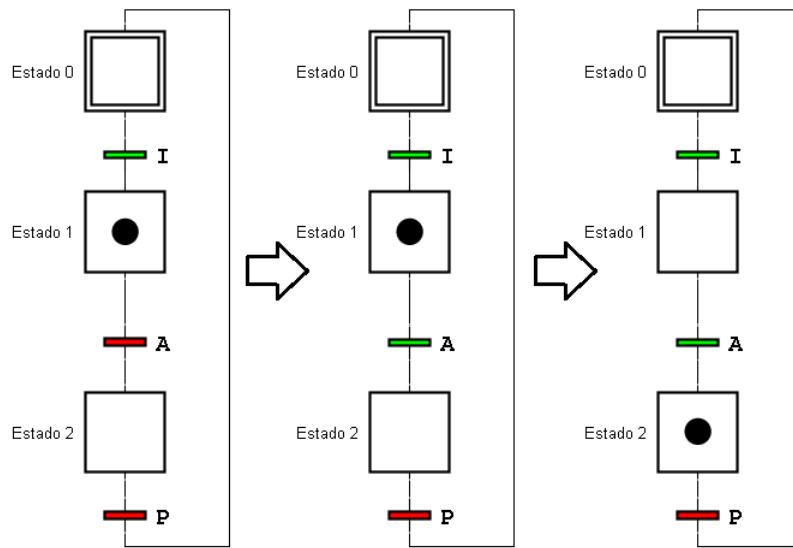


Figura 6.2: Sequência de disparo da transição A

Neste caso, mesmo com P ativo, o sistema não desativa a Etapa 2a nem ativa a Etapa 0, pois como a Etapa 2b não está ativa, a transição P não pode disparar.

## 6.2 Ações

Outra característica do grafcet é que podem ser definidas ações, ligadas a etapas. O que exatamente compõem uma ação depende da implementação do grafcet. Peguemos dois exemplos: o JGrafchart e a SFC, que é a descrição de grafcet definida pela norma IEC61131.

### 6.2.1 JGrafchart

No JGrafchart, uma ação pode ser uma definição de valor (*um assignment*), da forma variavel = expressao, uma chamada de subrotina (*call*) ou uma variável booleana (*bool*), que fica verdadeira enquanto a etapa estiver ativa. Um qualificador de uma letra define o momento em que a ação é executada:

**S assignment|call;** Executa apenas uma vez, quando a etapa é ativada.

**X assignment|call;** Executa apenas uma vez, quando a etapa é desativada.

**P assignment|call;** Executa a cada ciclo enquanto a ação estiver acionada.

**N bool;** Torna verdadeira a variável booleana enquanto a etapa estiver acionada.

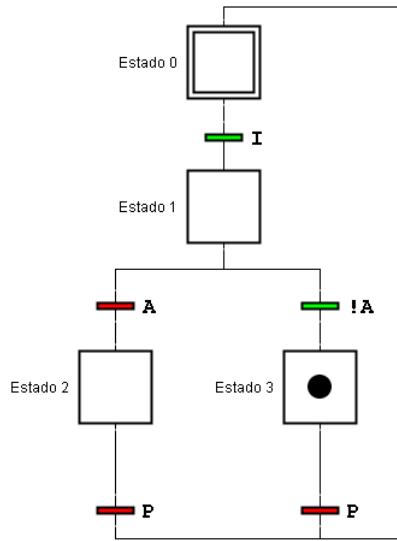


Figura 6.3: Exemplo de grafcet com divergência e convergência OU. O  $\text{!}A$  indica que a transição está ativa quando  $A$  for falso.

**A assignment|call;** Executa apenas uma vez, se a etapa for desativada por uma transição de exceção (apenas definida no JGrafchart).

Note-se que a ação **N bool;** é equivalente a **S bool=1; X bool=0;** e é presente para facilitar a escrita, já que tal operação é feita muitas vezes.

Toda etapa tem uma variável booleana **x** (`<nome_da_etapa>.x`), que é verdadeira quando a etapa é acionada, e uma variável de tempo em décimos de segundo **t** (`<nome_da_etapa>.t`) e em segundos **s** (`<nome_da_etapa>.s`) que contam o tempo desde que a etapa iniciou.

### 6.2.2 SFC

As ações do SFC são do formato **qualificador funcao|variavel finalizado . finalizado** uma variável opcional no caso de se usar uma função, que aciona quando a função termina, o que a torna útil para ser utilizada na transição de saída da etapa.

O qualificador pode ser qualquer um dos descritos na tabela 6.1. Pode-se não especificar o qualificador, onde o comportamento fica o mesmo de se usar o qualificador **N**. No caso da ação ser uma função, ele é executada sempre que ficar ativa. Todos os qualificadores que tem tempo acompanham um valor de tempo dado ou por uma variável ou por uma constante da forma **T#4k**, onde **k** pode ser **t** (décimo de segundo), **s** (segundo), **m**, (minuto) ou **h** (hora).

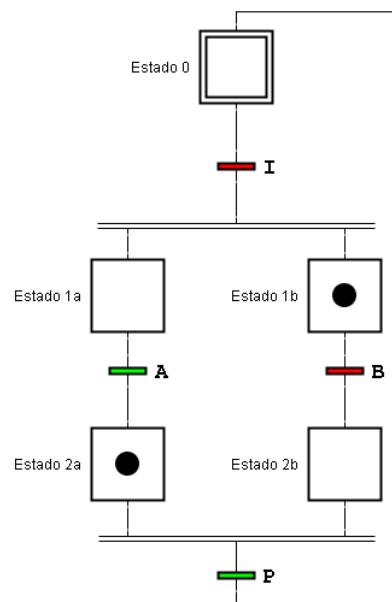


Figura 6.4: Exemplo de grafcet com divergência e convergência E.

Tabela 6.1: Qualificadores do SFC

Qualificador	Descrição
N	Ativa a etapa estiver acionada.
S	Set – ativa e mantém ativado mesmo depois de sair da etapa.
R	Reset – desativa.
P	Ativa apenas no primeiro ciclo que a etapa estiver ativa.
L tempo	Ativa pelo tempo determinado enquanto a etapa estiver acionada.
SL tempo	Ativa pelo tempo determinado independente da etapa ainda estar acionada.
D tempo	Ativa após o tempo determinado enquanto a etapa estiver acionada.
SD tempo	Seta após o tempo determinado independente da etapa ainda estar acionada.
DS tempo	Seta após o tempo determinado se a etapa ainda estiver acionada.

### 6.3 Níveis

O Grafcet pode ser definido em 2 níveis de abstração: o 1 e o 2. O nível 1 serve como uma ferramenta de desenvolvimento, para analisar a partir do problema como um todo a sequência de etapas, as ações a serem realizadas em cada etapa e as condições de transição de uma etapa para outra. No nível 1 as ações e transições são descritas de forma ampla, para permitirem uma melhor visualização do processo pelo desenvolvedor. O resultado final é uma descrição dos requisitos daquelas etapas e transições e não serve como uma linguagem para programar um sistema.

Já o Grafcet nível 2 é propriamente uma linguagem de programação, com diferentes implementações. Uma delas é o SFC - *Sequential Function Chart*, descrita no padrão IEC1131-3 para programação de CLPs. No SFC, cada ação corresponde a uma atuação nas saídas ou variáveis internas do CLP e cada transição corresponde a um valor binário obtido no CLP seja de entradas, seja de comparações de valores analógicos ou de tempo.

Desta forma o principal uso do grafcet é num projeto top-down, onde a partir do problema se desenvolve a sequência a ser seguida no grafcet nível 1, a partir deste se definem todas as entradas, saídas e condições necessárias para o controle automático daquela sequência e então programa-se o sistema usando grafcet nível 2.

Tomemos como exemplo uma tarefa relativamente simples: cozinhar um ovo. Considere-se o sistema da figura 6.5, consistindo de uma panela, uma entrada de água, uma boca de fogão a gás e um mecanismo que solte o ovo na água.

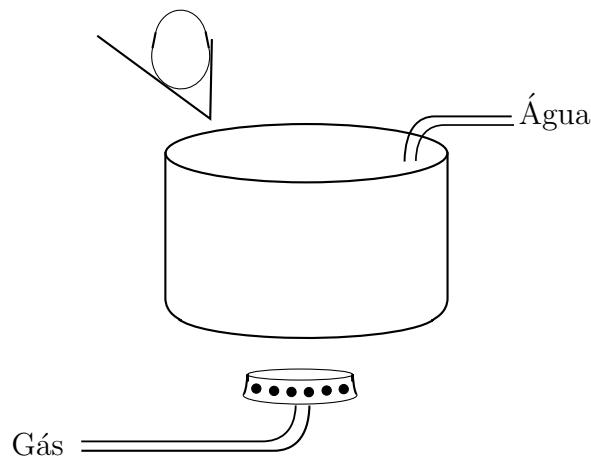


Figura 6.5: Sistema para cozinhar ovo sem sensores ou atuadores.

Podemos inicialmente fazer um grafcet considerando que tudo dá sempre certo: ao iniciarmos o sistema, a panela enche de água, o ovo é colocado, o fogo acendido

e aguarda-se até o ovo ficar pronto, de onde se retira a panela para tirar o ovo. Esta sequência é descrita pelo grafctet da figura 6.6.

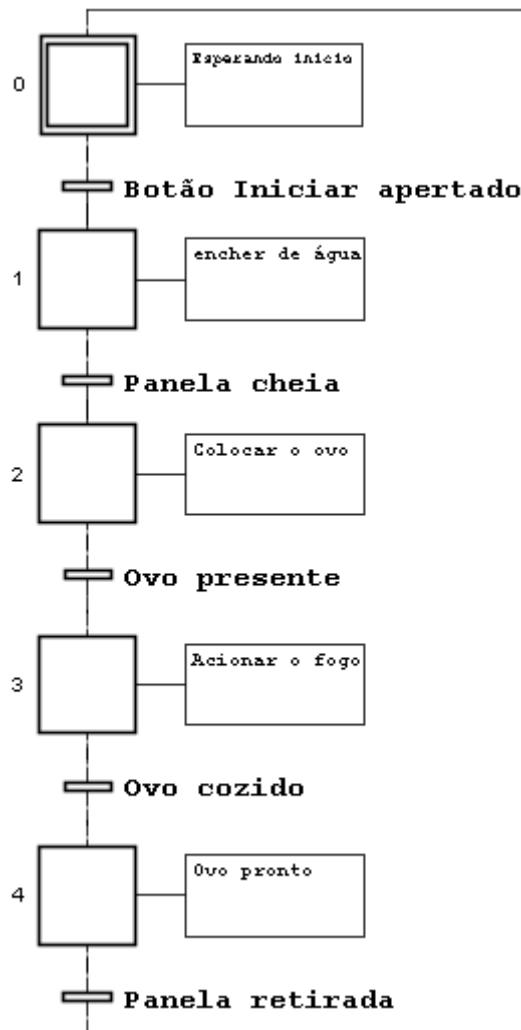


Figura 6.6: Diagrama grafcet nível 1 para cozinar um ovo de forma otimista.

Porém, problemas acontecem. Pode-se, por exemplo, esquecer de colocar a panela de volta antes de iniciar, faltar fogo ou acabar o ovo. Colocando todas estas condições a mais e definindo o que fazer se elas acontecerem o grafcet cresce para o apresentado na figura 6.7.

Com base neste diagrama, pode-se analisar agora quais são os sensores e atuadores necessários para esta tarefa, mostrados na figura 6.8. São eles:

**S\_PP** Sensor de presença para detectar a panela.

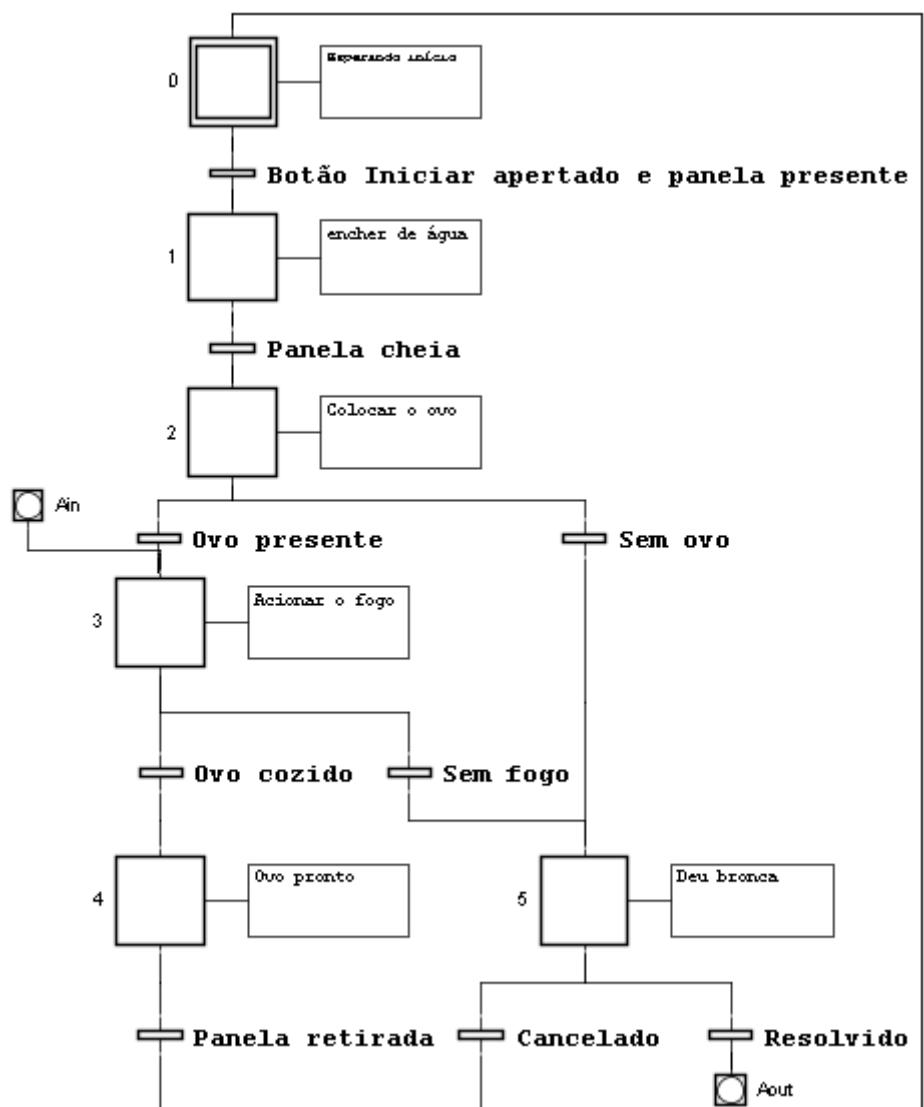


Figura 6.7: Diagrama grafcet nível 1 para cozinhar um ovo.

**S\_N** Sensor de distância para obter o nível de água.

**S\_G** Sensor de gás.

**B\_I** Botão de início.

**B\_OK** Botão de OK, para continuar se der bronca.

**B\_C** Botão de cancelamento.

**PO** Solenoíde para abrir a portinhola do ovo.

**VA** Válvula de controle da água.

**VG** Válvula de controle do gás.

**CENTELHA** para ligar o fogo.

**ALARME** para indicar que deu errado.

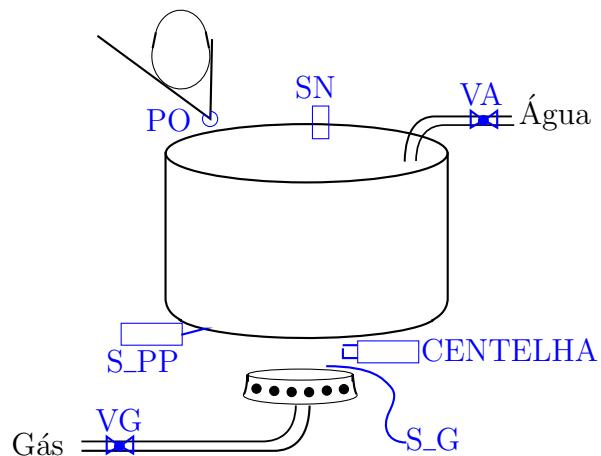


Figura 6.8: Sistema para cozinhar ovo com sensores e atuadores.

Então com base nos sensores e atuadores presentes, podemos fazer o grafctet nível 2 (figura 6.9, descrevendo as ações e transições de acordo com estas entradas e saídas.

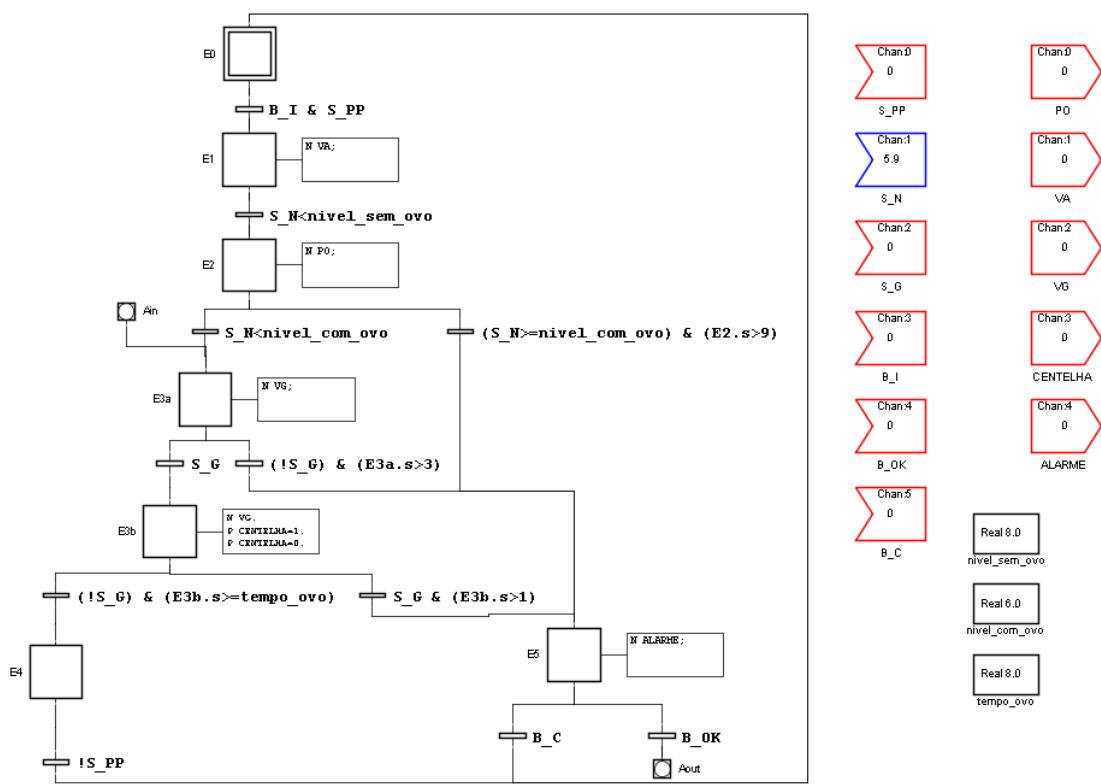


Figura 6.9: Diagrama grafcet nível 2 para cozinhar um ovo.

# **Capítulo 7**

## **SCADA**

SCADA – *Supervisory Control And Data Acquisition*, são os sistemas do nível 2 da pirâmide de automação. Eles são a interface com os operadores com os sistemas de controle do nível 1. Suas funções são:

**Supervisão** , mostrando de forma prática o estado do processo.

**Operação** , substituindo os painéis de controle. Permite ligar e desligar os equipamentos, definir setpoints, etc.

**Controle** de operações simples e que não tenham restrições temporais grandes.

Hoje em dia os sistemas SCADA também tem grande integração com os sistemas acima da pirâmide, e são responsáveis por fornecer dados a estes sistemas e receber informações adicionais, tais como setpoints pré-fixados e receitas. Basicamente o SCADA serve como interface principal dos usuários com os sistemas de automação.

Tipicamente, um sistema SCADA possui:

**Sinóticos** - Telas representativas do processo.

**Alarmes** - Seja definidos no próprio SCADA seja definidos num elemento de controle (que é o mais comum).

**Gráficos de tendência** - Mostram a variação de variáveis do processo ao longo do tempo.

**Gerador de relatórios** - tipicamente contendo os alarmes e eventos de um determinado período e vários gráficos de tendência. Podem ser gerados devido a uma condição de alarme.

## 7.1 Variáveis

O sistema SCADA trabalha com informações advindas do nível 1 da pirâmide de automação (CLPs, SDCDs), do nível 3 (PIMS e MES) ou geradas no próprio SCADA. Estas informações são armazenadas em variáveis na base de dados do SCADA, também chamadas de TAGs. Esta variáveis tem associadas a elas um conjunto de informações, dos quais os mais comuns são:

**Tag** O nome *oficial* da variável. Único para cada uma delas.

**Endereço** A origem da variável — que memória de qual CLP, etc.

**Descrição** Uma descrição sucinta da variável.

**Valor** Último valor medido.

**Time Stamp** Data e hora em que foi medido o último valor.

São variáveis típicas:

**Variáveis Analógicas** Tipicamente obtidas dos CLPs em formato bruto e convertidas para valores de engenharia no SCADA. Muitas vezes são também submetidas a um filtro digital no próprio SCADA, para diminuir ruídos.

É comum que se adicionem limites a uma variável deste tipo relacionados a alarmes da mesma, tais como:

- Lim inferior: valor em UE ser atribuído ao valor 0% da variável.
- Lim superior: valor em UE a ser atribuído ao valor 100% da variável.
- Limite HH: valor em UE para alarme Muito Alto.
- Limite H: valor em UE para alarme Alto.
- Limite L: valor em UE para alarme Baixo.
- Limite LL: valor em UE para alarme Muito Baixo.

**Variáveis Discretas** Tipicamente valores binários.

Em geral são associadas uma descrição dos estados 0 e 1, tais como: aberto/fechado, desligado/ligado, parado/funcionando, e assim por diante.

**Totalizadores** Pode ser um contador de pulsos ou um integrador de um valor analógico (de vazão, por exemplo). Em alguns casos o valor totalizado é calculado no CLP e em outros no próprio SCADA.

Um totalizador pode ser referente a um período (hora, turno, dia, mês, ano), a um equipamento, a um operador, a um produto ou a alguma outra informação.

**Controladores PID** Um controle do tipo PID (Proporcional, Integral e Derivativo) é muito usado para garantir que uma determinada grandeza observada (variável de processo) acompanhe o valor desejado (setpoint) pelo ajuste contínuo de uma grandeza controlada (variável manipulada). CLPs mais modernos já contam com algoritmos de controle PID que dependem das constantes definidas no controlador (setpoint, Kp, Ki e Kd).

No sistema SCADA, uma variável do tipo PID armazena os valores destas constantes, que podem então ser ajustados pelo operador ou por um algoritmo de ajuste. Tipicamente o CLP e o SCADA definem formas de sobrepassar o controlador PID permitindo o controle da variável manipulada diretamente pelo operador.

**Equipamentos** Corresponde a um equipamento de processo qualquer: motor, transportador de correia, bomba, reator, etc. Conta com valores discretos que informam o status do equipamento, bem como com um horímetro, que fornece o total de horas de operação do equipamento.

Normalmente permite sobrepassar o controle automático do equipamento e realizar ações de comando, tais como ligar e desligar.

## 7.2 Sinótico

Boa parte do uso de um sistema SCADA é como uma IHM – Interface Homem Máquina. Neste ponto ele substitui os painéis ou quadros sinóticos (vide figura 7.1), usados ainda em algumas aplicações simples mas muito usados antigamente.



Figura 7.1: Exemplo de um antigo painel sinótico.

Um sinótico é uma representação gráfica simplificada de um sistema – uma síntese ótica. Num sistema SCADA, um sinótico representa uma área do processo em um certo nível de detalhe.

Tipicamente se tem um sinótico geral para uma planta, do qual se podem abrir sinóticos específicos de uma determinada área (numa hierarquia de sinóticos) ou a uma visão de uma outra camada do mesmo sinótico (por exemplo, um mostrando características elétricas e outro características termodinâmicas de um processo).

### 7.2.1 Requisitos

O projeto de um sinótico é bastante complexo e foge um tanto das competências típicas de um engenheiro. Ao projetar uma IHM o projetista deve trabalhar com

vários requisitos ergonômicos e de comportamento humano, procurando:

- Diminuir a chance de erro do operador, principalmente nos momentos de maior demanda operacional, coincidentes com o aumento do stress.
- Evitar as situações de monotonia que levam à desconcentração do operador. Sinóticos pouco representativos do processo e sem atrações de animação ou com muitos dados tabulares levem ao desinteresse.
- Evitar situações que acarretam cansaço.
- Manter o operador sempre atento ao que realmente interessa. Sinóticos muito cheios trazem excesso de informações que o operador não é capaz de processar. Os alarmes e informações devem ser mostrados apenas em casos de exceção para evitar avalanches de alarmes.
- Evitar consulta a referências externas ao sistema. Se necessário, o operador deve poder tirar dúvidas quanto a operação do sistema no próprio SCADA.

Estudos ergonômicos voltados para a atenção de pessoas a IHMs mostram que os olhos tendem a se focar primeiro em uma imagem grande ao invés de uma pequena, em cores mais saturadas (brilhantes), em formas simétricas ao invés de formas assimétricas e em algo que se move e pisca em contraponto a uma imagem estática. Estes conhecimentos devem ser utilizados para a construção do sinótico bem balanceado.

São bons costumes em um sinótico:

- Utilizar representação gráfica dinâmica (animações) para evitar a monotonia, mas.
- Evitar objetos grandes piscantes, que cansam e distraem.
- Representar equipamentos por desenhos de acordo com sua forma e tamanhos reais, porém a representação fotográfica com excesso de detalhes, sombra, etc. é desaconselhável.
- Deixar o mais óbvio e intuitivo possível a seqüência para ligar ou desligar equipamentos ou realizar ações de controle similares. Deve-se utilizar de metáforas visuais com equipamentos físicos.
- Utilizar mensagens devem ser claras, explícitas e auto suficientes. Um contra

Um determinado valor pode ser mostrado num sinótico de diversas formas. As mais comuns são:

**Texto** – Para uma variável analógica, exibe seu valor em unidades de engenharia.

A cor do texto pode servir para codificar o status da variável (por exemplo, se valor muito alto ou muito baixo). Para um variável binária é comum atribuir o significado: ABERTO/FECHADO, LOCAL/REMOTO ou LIGADO/DESLIGADO.

**Barras horizontais e verticais** – Fornecem uma representação percentual do valor da variável. Podem ser utilizados para mostrar o enchimento de um silo, tanque, reator, etc.

**Deslocamento e rotação** – Podem ser pelo valor de uma variável analógica ou relacionados a valores discretos, tais como sensores de fim de curso, por exemplo.

**Mostradores Circulares** – Evocam os mostradores de ponteiro de instrumentos indicadores analógicos.

**Tendência** – Exibe o gráfico dos últimos valores da variável em função do tempo.

**Associação a cor (ou outro atributo) de um objeto** – A cor do objeto muda de acordo com o status da variável associada.

**Associação a um par de objetos complementares** – Os dois objetos ocupam fisicamente a mesma posição no sinótico. Quando a variável está em 0 o objeto chave aberta, por exemplo, é exibido, quando está em 1 a chave é mostrada na posição fechada.

Deve-se sempre buscar a representação mais natural para cada variável. Por exemplo, enchimento para tanques e silos, rotação para um forno de cimento ou britador de martelos, etc.

## 7.3 Alarmes e Eventos

Outra função do SCADA é mostrar e armazenar todos os eventos significativos que ocorreram no processo, que são classificados em simples eventos a serem registrados (por exemplo: troca de operador, fim de fabricação de lote, etc) e alarmes, que são situações indesejadas e que requerem ações corretivas.

Os alarmes são classificados por prioridade, definida por dois fatores: severidade das consequências e tempo requerido para tomar ações corretivas. Tipicamente os sistemas SCADA usam uma classificação numérica para a prioridade dos alarmes, mas uma classificação mais intuitiva é a de prioridade baixa, média, alta e crítica. Tome-se por exemplo o sinal de temperatura de um forno. Para um determinado processo, pode-se considerar que:

- Uma temperatura baixa (L) durante a operação do forno indica um problema no controle, mas que a princípio não causará dano imediato. Atribui-se um alarme de baixa prioridade a este evento.
- Uma temperatura muito baixa (LL) indica que o processo já não mais funcionará e talvez vá se perder aquele lote. Atribui-se um alarme de prioridade média.
- Uma temperatura alta (H) já indica perda de material e pode levar a problemas piores se não for corrigido rápido. Atribui-se um alarme de alta prioridade.
- Uma temperatura muito alta (HH) já é um risco iminente de incêndio. Atribui-se um alarme de prioridade crítica.

Estes alarmes e eventos podem ser definidos no próprio SCADA ou no CLP, enviados ao SCADA como uma variável booleana.

Um alarme deve ser mostrado no painel de controle do SCADA para chamar atenção do operador. A partir do momento que um alarme ocorre, ele precisa ser normalizado (a situação que o gerou deve ser corrigida, seja por atuação do operador ou pelo controle automático do processo) e o operador precisa marcar no supervisório que identificou o alarme (chama-se reconhecer o alarme). A ideia por trás disso é que uma vez que o operador reconheceu o alarme, o supervisório não mais precisa ficar chamando a atenção do operador para aquele problema.

Com base nisto, pode-se apresentar o alarme na tela da seguinte forma (entre outras possibilidades):

**Vermelho Piscante** Alarme não reconhecido.

**Vermelho** Alarme já reconhecido.

**Amarelo** Normalizado e não reconhecido.

**Verde** Normalizado e reconhecido.

Os alarmes e eventos são armazenados em uma base de dados com os seguintes dados:

- Alarme ou evento disparado
- Valor no momento do alarme
- Descrição do evento
- Data e hora do evento

A normalização ou reconhecimento de um alarme conta como eventos que também são armazenados nesta base de dados.

Esta base de dados pode ser consultada no próprio SCADA, pode ser inserida num relatório automático e/ou enviada para sistemas acima na pirâmide automação.

## 7.4 Arquitetura de Hardware e Software

Sistemas SCADA modernos dependem bastante da estrutura da rede de computadores. Estes sistemas permitem dividir as tarefas do SCADA em vários computadores: Um servidor de entrada e saída, para comunicação com os CLPs, um servidor de banco de dados, para armazenar e distribuir os dados obtidos, um gerenciador de alarmes e várias estações de acesso, que permitem acessar o sistema de vários pontos. Cada um destes sistemas pode, além disto, ter uma redundância, de modo que se um computador falhar outro assume na mesma hora, sem perda de funcionalidade. Ao mesmo tempo, para sistemas simples ainda é possível ter todas estas funções rodando em um único computador.

Cada vez mais comum é ter um servidor web, que permite disponibilizar toda a ferramenta SCADA num navegador de internet. Para tanto deve-se ter um cuidado com o acesso a esta rede, o que normalmente é feito através de VPNs - *Virtual Private Networks*, que criam uma rede privada virtual sobre a internet usando criptografia para prevenir acessos indesejados.

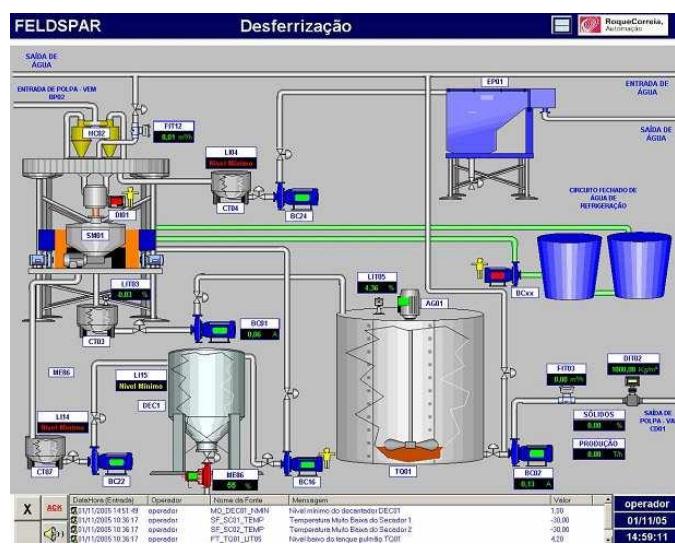


Figura 7.2: Exemplo de um sinótico em um SCADA.

# Capítulo 8

## Gerenciamento da manufatura: PIMS e MES

PIMS – *Process Information Management System* e MES – *Manufacturing Execution System* são sistemas da camada 3 da pirâmide de automação, responsáveis pelo armazenamento e tratamento de dados do nível 2, concentrando os dados de diversos processos separados em um único ponto. São os chamados *middleware*, pois ficam a meio caminho entre os sistemas de gerenciamento de empresa e os supervisórios, às vezes combinando funções de um ou de outro.

De forma geral, no terceiro nível da pirâmide a preocupação é em consolidar os dados brutos do processo (*data*), para com eles gerar informações (*information*) e conhecimento (*knowledge*) sobre o processo, aumentando o valor destes valores, como mostra a figura 8.1. Os dados são obtidos ou do controlador ou do supervisório de um determinado processo. A relação entre dados ou a variação destes dados no tempo geram informação sobre a planta. A relação entre informações ou a variação de informações no tempo geram conhecimento.

Um exemplo desta relação é mostrado na figura 8.2. Nesta figura, a partir dos dados de temperatura e vazão de um fluido é gerada a informação do calor removido em determinado trocador de calor. A comparação dos calores removidos de diversos trocadores gera o conhecimento de qual trocador é mais eficiente.

### 8.1 PIMS

Para a finalidade de gerar informação e conhecimento, o ponto de partida é obter os dados brutos. Esta é a tarefa principal do PIMS: adquirir, armazenar e apresentar diversos dados de uma planta. O PIMS foi criado e ainda é principalmente usado para processos contínuos, tais como uma refinaria ou siderúrgica, e portanto tem um enfoque muito grande em variáveis analógicas e na relação delas com o tempo.

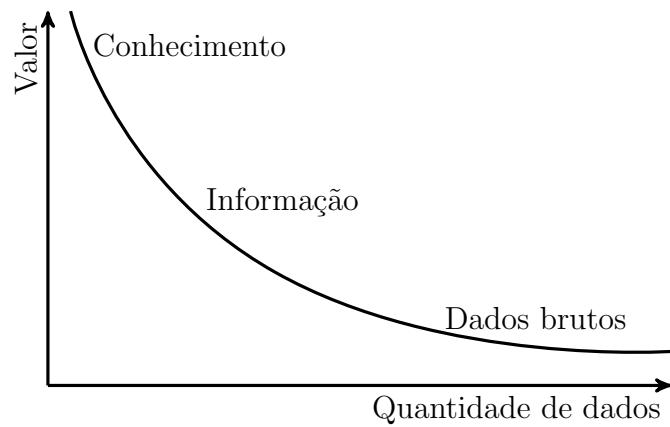


Figura 8.1: Relação entre dados, informações e conhecimentos.

Do ponto de vista do PIMS podemos esquematizar os sistemas de uma fábrica como mostra a figura 8.3, onde se vê que o PIMS tem 4 partes principais: historiador de processos, banco de dados temporal, interface gráfica e aplicações clientes (variadas funções, desde análise dos dados a interface com outros sistemas).

### 8.1.1 Historiador de Processos

O historiador do processo se comunica com vários sistemas do nível 1 (CLP, CNC) ou 2 (supervisório) ou ainda de outros sistemas nível 3, tais como um LIMS – *Laboratory Information Management System* ou MES para obter dados brutos dos diversos processos e acumula-os no banco de dados. Tal sistema fornece as seguintes funcionalidades:

**Registro histórico** para análise de incidentes, controle de qualidade, métricas de performance, entre outros.

**Adequação a normas** como por exemplo para controle ambiental.

**Monitoração de equipamentos** para controle de vida útil e apoio à manutenção.

**Análise de processo** facilita a visualização de dados e detecção de correlações.

Os dados coletados são principalmente os valores das variáveis do processo, sejam discretos ou contínuos, mas também abarcam outras informações, tais como a ocorrência de alarmes, a marcação de que operador está presente, o período que um equipamento está ligado, entre outros. Cada uma destas informações é identificada por um marcador único - a chamada *tag*, ao qual também está associado o

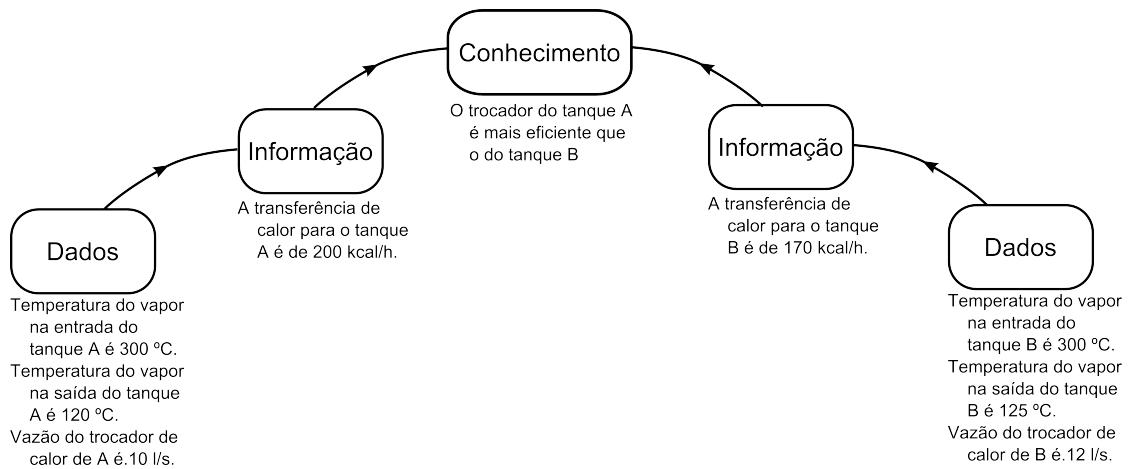


Figura 8.2: Exemplo da transformação de dados para informação e de informação para conhecimento.

endereço lógico de onde se obtém tal informação e o tempo em que tal dado foi gerado (*time stamp*). Em alguns casos se associa também uma métrica da qualidade do dado, referente a confiabilidade daquele dado, tal como se o instrumento de medida está calibrado ou não.

Estas informações podem ser obtidas tanto de sistemas SCADA (nível 2) ou de CLPs (nível 1). Algumas vantagens de pegar informação dos sistemas nível 2 são que:

- o SCADA já converteu os dados para unidades de engenharia enquanto que em alguns CLPs os dados estão em valor bruto (de 0 a 4095);
- muitas variáveis são definidas apenas no sistema SCADA, não existindo nos CLPs, tais como o motivo de alarmes ou qual operador está monitorando a operação;
- interface com os sistemas SCADA costuma ser padrão, o que facilita a comunicação.

Vantagens de obter as informações do CLP são:

- busca dos eventos com menor atraso temporal;
- pode-se coletar os dados em um ponto único, se todas as redes de CLPs estiverem interligadas;
- CLPs são mais confiáveis e apresentam menor suscetibilidade a falhas que os sistemas SCADA.

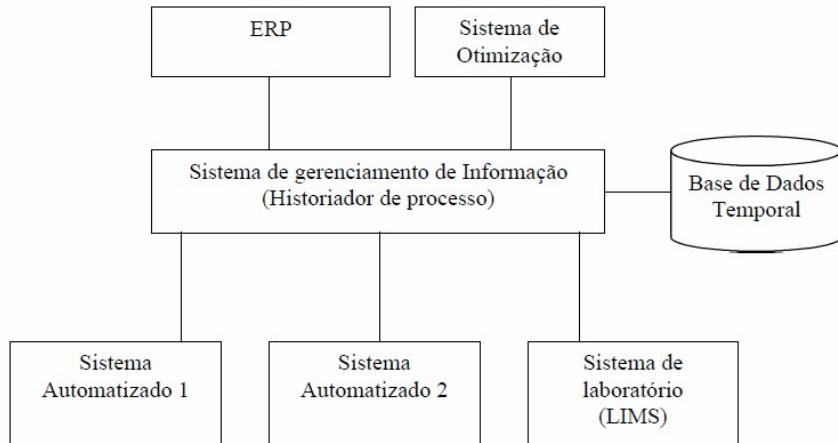


Figura 8.3: Sistema PIMS.

### 8.1.2 Banco de Dados Temporal

A maioria das análises realizadas nos dados de um sistema PIMS são em função do tempo, logo é comum ele usar um banco de dados que indexa a informação pelo tempo. Basicamente é uma tabela, relacionando *time stamp*, *tag*, tipo de dado (análogico, booleano, texto), valor e qualidade (se houver).

Um problema do uso deste tipo de banco de dados, ao invés dos chamados bancos de dados relacionais, tipo SQL, é que a busca por informação pode ter uma baixa performance quando a quantidade de dados aumenta muito. Esta é a principal razão para que estes sistemas façam uma compressão de dados, que basicamente se resume a não armazenar dados que não tragam muita informação nova. A figura 8.4 mostra a idéia por trás da compressão de dados.

Algoritmos comuns para a compressão de dados no PIMS são *banda morta*, onde os dados são apenas armazenados se variarem mais do que um mínimo especificado; o *SDCA – Swinging Doors Compression Algorithm*, onde para cada valor recebido é definida uma reta entre ele e o último valor armazenado, descartando valores que possam ser definidos por esta reta e mais um erro; e o *boxcar/backslope*, que usa a banda morta e mais uma reta definida pelo último valor armazenado.

### 8.1.3 Interface Gráfica

A interface gráfica de um sistema PIMS é, em muitos aspectos, muito parecida com a de um sistema SCADA, contendo representações pictóricas do processo (sinóticos) com os valores de várias variáveis e gráficos de tendência. Tais elementos são melhor vistos no contexto de um sistema SCADA.

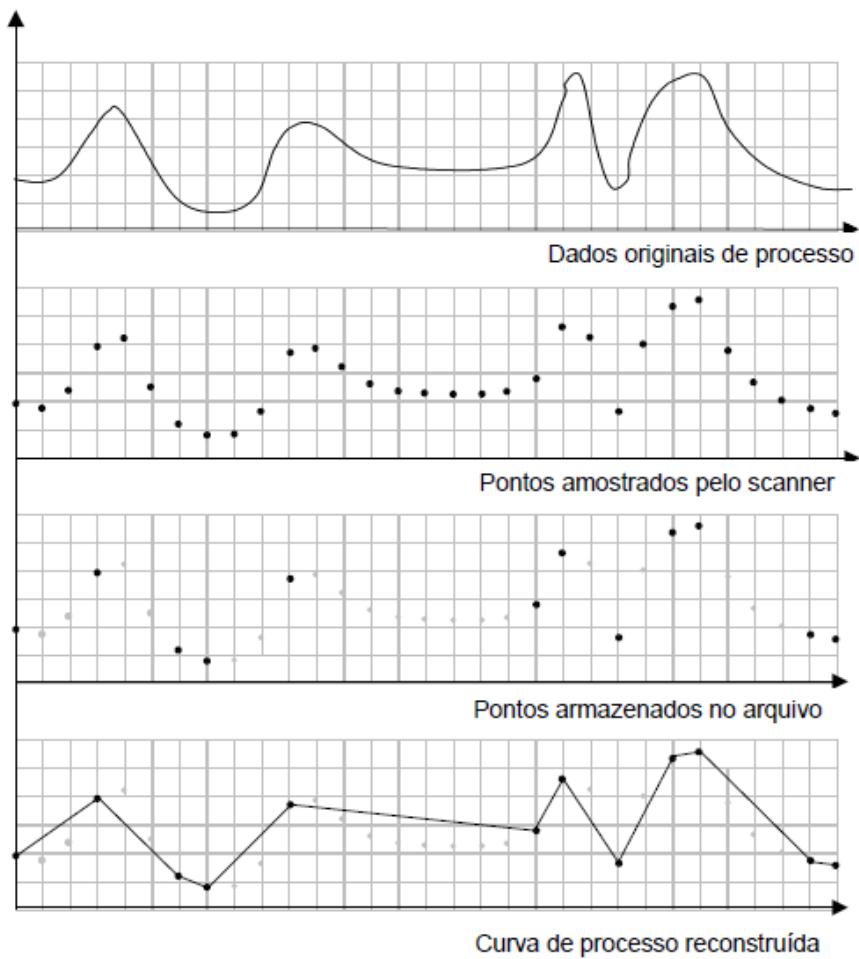


Figura 8.4: Processo de amostragem, compressão e reconstrução dos dados.

## 8.2 MES

Assim como o PIMS, um sistema MES também adquire dados do processo com o objetivo de apresentar uma visão geral do processo. Porém, enquanto o PIMS é focado mais no armazenamento dos dados, o MES tem uma gama maior de funções e um papel mais ativo. Historicamente, os MES são usados principalmente em processos discretos, muitas vezes em sistemas de automação flexível ou programada. Isto faz com que o MES tenha que lidar com o sequenciamento dos processos, o que ocorre menos em sistemas contínuos.

Os sistemas MES agregam diversas funções de sistemas anteriores mais simples e específicos e são definidos por terem as seguintes funções:

**Gerenciamento das definições de produto.** Todas informações necessárias para a fabricação do produto. Isto inclue lista de materiais e insumos, set-points do processo e receitas.

Por receita, entenda-se uma variação do processo para, na mesma máquina, produzir produtos diferentes ou variações dele. Por exemplo, num processo de pintura a receita diria quais as tintas usadas, em que sequência, em que volume e em que velocidade de aplicação. É muito importante na automação flexível e programável.

Sistemas de automação fixa também se utilizam de receitas. Podemos citar dois casos mais importantes: para a melhoria do processo e do produto, testando diferentes receitas e checando os resultados e para casos de variações de matéria prima ou de condições ambientais, onde se buscaria a melhor receita para cada caso.

**Gerenciamento de insumos.** Permite preparar e executar ordens de produção com garantia de disponibilidade dos insumos. Esta função cuida do controle de estoques e dos pedidos aos fornecedores, principalmente quando se usa a metodologia de *just in time*, que minimiza os estoques.

**Agendamento de produção.** Permite determinar a ordem que a produção será feita, para alcançar os requerimentos de produção definidos pela ERP (camada 4 da pirâmide) utilizando otimamente os recursos. Também chamado de *scheduler*.

A figura 8.5 mostra o exemplo de um *scheduler*, onde se vê o uso de um diagrama de Gantt para definir o sequenciamento das operações e máquinas.

Tipicamente inclui ferramentas de simulação, que permitem comparar diversas opções de ordens e estimar efeitos quando de mudanças imprevistas na sequência de produção (em geral por conta de uma parada não programada).

**Envio de ordens de produção.** Em função do agendamento feito, o MES cuida de enviar as ordens de produção para os diversos postos da planta. A informação vai para os supervisórios e, em alguns casos mais avançados, para os controladores.

**Acompanhamento da execução de ordens de produção.** Também se comunica com sistemas níveis 1 e 2 para garantir a execução das ordens. Inclui também o registro de paradas.

O registro de paradas é feito automaticamente quando o equipamento para, seja por alguma condição espúria detectada no nível 1 ou por ação do operador no nível 2. Em ambos os casos fica registrado uma parada em aberto, que

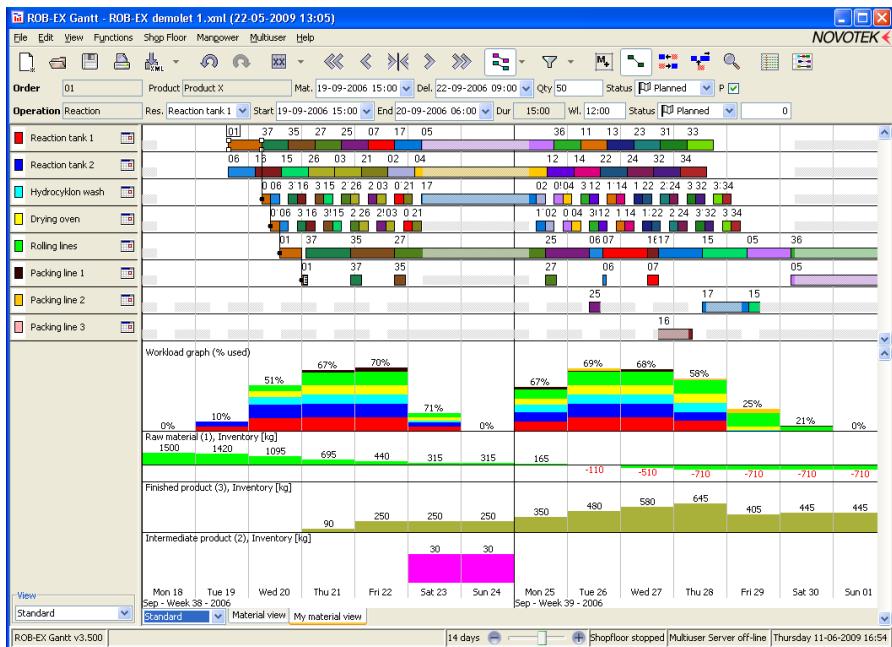


Figura 8.5: Exemplo de um *scheduler*.

só finaliza quando o operador complementar certas informações que auxiliam no diagnóstico da parada e o equipamento voltar a funcionar.

**Coleção dos dados de produção.** Equivalente ao historiador de processo do PIMS.

**Análise da performance da produção.** Cálculo dos chamados índices de produção – *KPI, Key Performance Indicators*, tais quais na figura 8.6. É a geração de informação a partir dos dados da produção.

O OEE – *Overall Equipment Effectiveness* é um exemplo de KPI não diretamente ligado ao produto, mas à produção. O OEE aponta a efetividade de um determinado equipamento ou célula de produção. Este índice é dado por:

$$\text{OEE} = \text{disponibilidade} \times \text{performance} \times \text{qualidade}, \quad (8.1)$$

onde por disponibilidade entende-se a razão entre o quanto de tempo o equipamento funcionou e o quanto de tempo ele deveria ter funcionado, ou seja, descontando-se as paradas não programadas; performance é a razão entre a produção do equipamento enquanto funcionava e a capacidade de produção

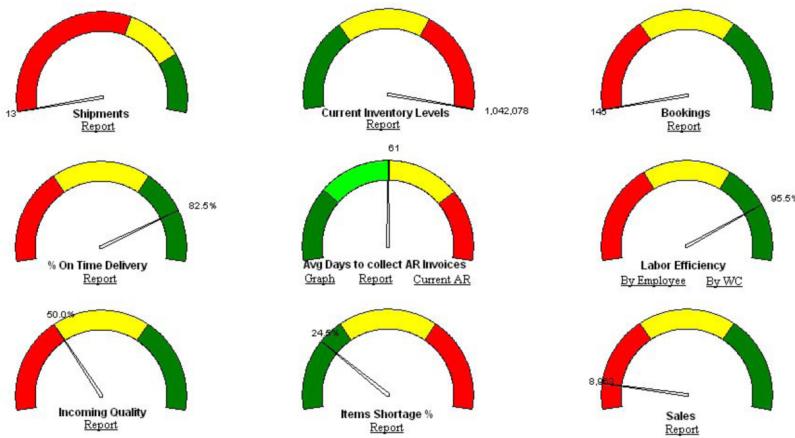


Figura 8.6: Exemplo de visualização de KPIs.

de que o equipamento é capaz, ou seja, descontando a ociosidade do equipamento; e qualidade é o valor do que o equipamento produziu em relação ao valor se não tivesse havido nenhum descarte ou geração de produtos de menor valor. Esta relação é melhor visualizada na figura 8.7 e exemplificada na figura 8.8.

■ **OEE = Disponibilidade \* Performance \* Qualidade**



Figura 8.7: Overall Equipment Effectiveness.

**Rastreamento da produção.** Permite levantar que produto ou lote foi feito quando e em qual equipamento. Útil para melhoria da produção e imprescindível para remédios e produtos alimentícios.



Figura 8.8: Overall Equipment Effectiveness.

**Armazenamento dos logs de produção.** Hoje em dia tais logs são inseridos pelo operador no próprio sistema supervisório e realcionados às variáveis de produção.

**Interface de auditoria.** Permite a análise dos diversos dados e informações armazenados e o cruzamento destes dados com outras bases de dados.

### 8.2.1 Redes industriais

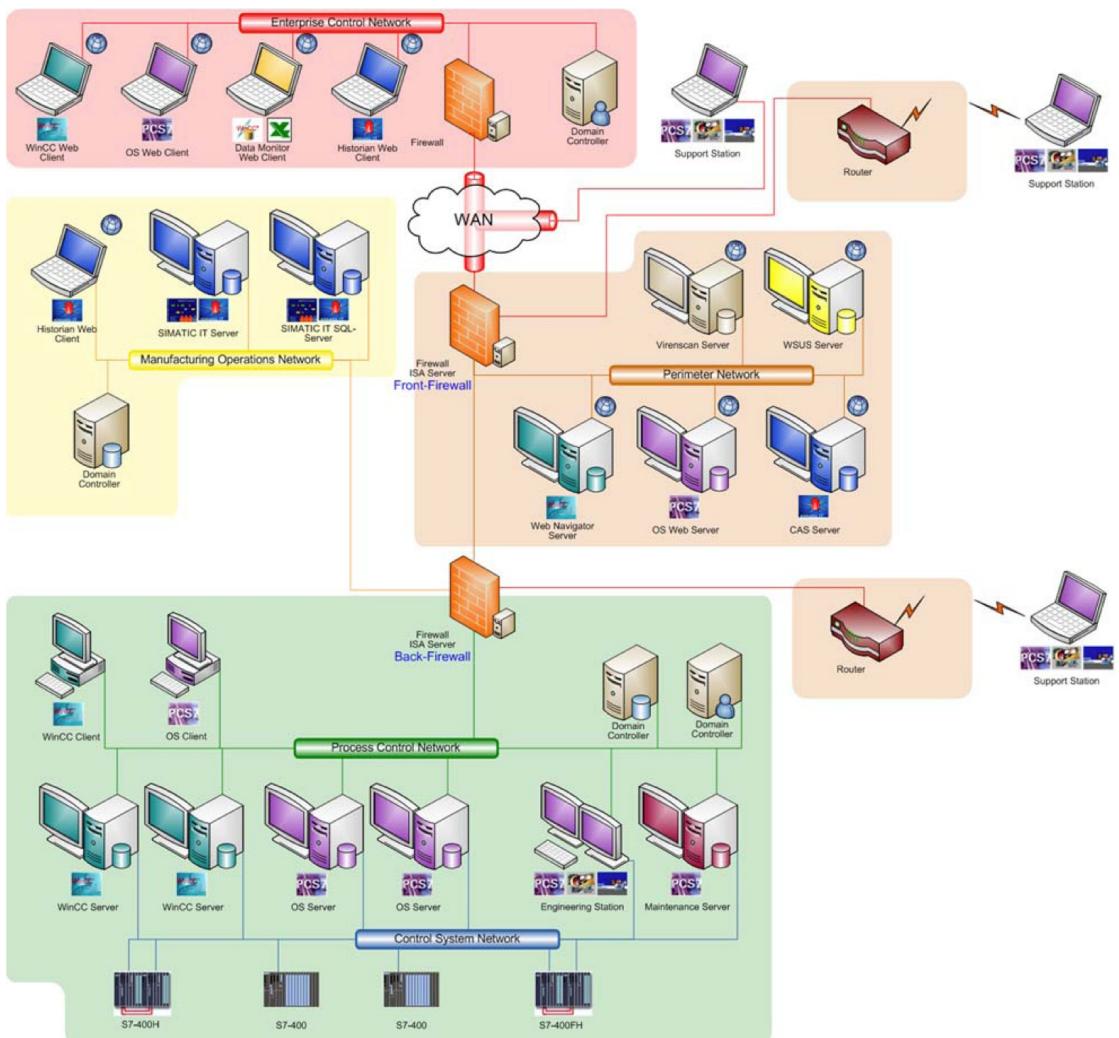


Figura 8.9: Arquitetura de rede de dados industrial.

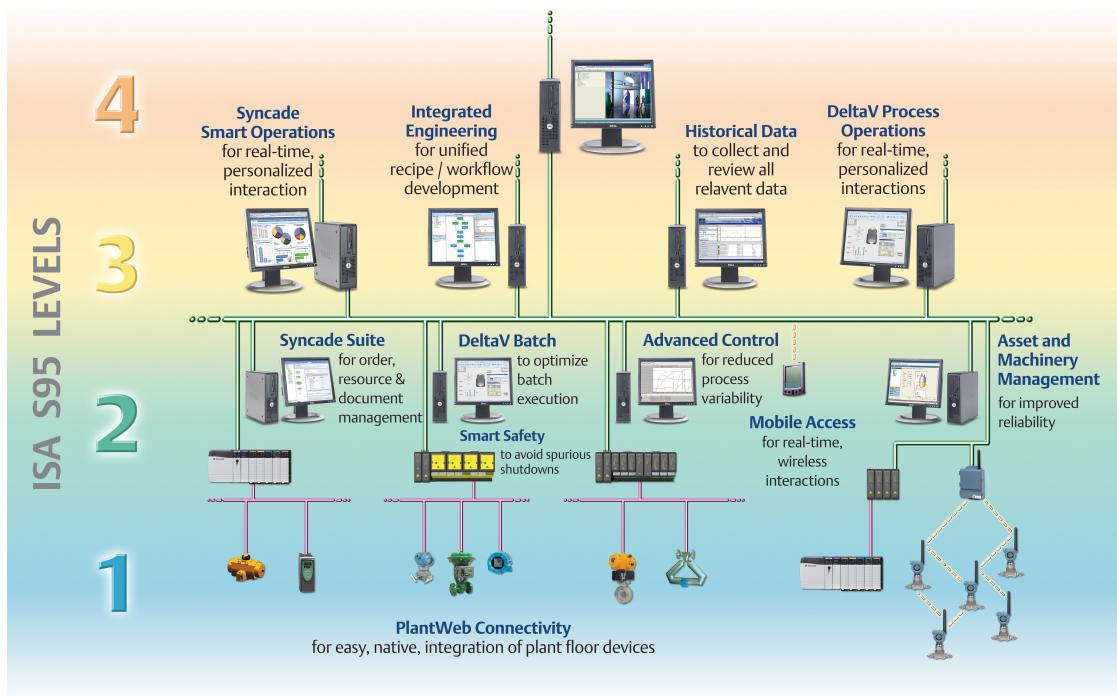


Figura 8.10: Arquitetura de rede de dados industrial.

# Capítulo 9

## Redes Industriais

## 9.1 Redes de Comunicação: Introdução e noções básicas

Na pirâmide de automação, a partir da camada 2, os sistemas utilizados são softwares em computadores, sejam servidores, terminais, desktops, notebooks ou mesmo tablets e smartphones (para o caso do supervisório). No nível de controle, o dispositivo utilizado pode ser um computador, um CLP – que não deixa de ser um computador – ou outro dispositivo que incorpora um processador. Hoje em dia até no nível 0 estão ficando cada vez mais comuns os dispositivos ditos *inteligentes*, que incorporam também um microprocessador. A comunicação entre eles é feita por uma rede de dados, tais como a ethernet.

O problema da comunicação de dados automática entre diversos dispositivos é bastante complexo. Ele envolve a:

1. transferência de dados de um dispositivo a outro,
2. em alta velocidade,
3. sem perda de informação,
4. eventualmente passando por dispositivos intermediários,
5. talvez envolvendo sistemas de comunicação diferentes,
6. com privacidade,
7. sem envolver outros dispositivos desnecessariamente.

Peguemos como exemplo pagar uma conta via internet. Logo é necessária a comunicação do dispositivo caseiro (que seja um tablet) e o computador central do banco (1). Quanto mais rápida for esta comunicação, mais agradável é para o usuário e mais barato é para o banco (2). Obviamente nenhuma das partes envolvidas querem que seja pago o valor errado ou a conta errada (3). Entre o tablet e o servidor do banco estão: o modem wifi, a central telefônica, servidores da companhia telefônica e talvez de outras fornecedoras de serviço (4), e em geral não é do interesse do usuário que a companhia telefônica ou outro elemento saiba de sua senha do banco (6). A comunicação do tablet ao modem é por wifi. Do modem para a central telefônica é pelo fio telefônico. Da central em diante pode ser por cabos de cobre ou, mais provavelmente, por fibra óptica (5). Também, por questão de custos e privacidade, não é desejável que outros elementos recebam aquela informação, mesmo que criptografada.

Para resolver este problema, foi desenvolvida a idéia de quebrá-lo em várias partes, ou camadas, onde cada camada é responsável por uma determinada parte

do problema. A informação fica então passando de camada a camada para resolver todas as questões relativas à comunicação. Tal esquema é também chamado de pilha de protocolos.

### 9.1.1 Modelo OSI

Diferentes tipos de rede de dados tem problemas diferentes. O modelo OSI procura definir camadas o mais genéricas possíveis, de modo a abranger qualquer tipo de comunicação de dados. Este modelo define 7 camadas: física, enlace, rede, transporte, sessão, apresentação e aplicação.

#### Física

A camada física é a única que não pode ser implementada em software. Ela lida com as definições eletro-mecânicas necessárias para a comunicação. Ou seja, a camada física define:

- a forma como os dados são transmitidos (sinais elétricos por cabo, sinais de rádio, luz, etc);
- as características do meio de transmissão, tais como o tipo de cabo, os níveis de tensão, o formato dos conectores, etc;
- a taxa de transmissão.

Note-se que a taxa de transmissão acaba sendo limitada justamente pelos diversos parâmetros físicos descritos pela camada física de um determinado protocolo de comunicação.

Outro parâmetro definido na camada física é se a rede é *simplex*, *half-duplex* ou *full-duplex*. Simplex significa que a comunicação no meio de transmissão é apenas num sentido (exemplo: televisão), half-duplex é uma comunicação em dois sentidos, mas apenas um por vez (exemplo: walkie-talkie) e full-duplex é uma comunicação em ambos os sentidos ao mesmo tempo (exemplo: telefonia).

#### Enlace

A camada de enlace cuida da comunicação de um dispositivo a seu vizinho, que não necessariamente são os pontos finais.

No exemplo acima, esta seria a comunicação entre o tablet e o modem wifi; entre o modem e a central; entre a central e o servidor da companhia telefônica, e assim por diante.

A camada de enlace de um protocolo define quem acessa o meio de transmissão e quando, que sinais indicam o início e o fim de uma transmissão, mecanismos que

detectem e/ou corrijam erros e, se necessário, para qual dentre vários dispositivos é aquela comunicação específica.

Alguns protocolos de comunicação, tais como USB e ethernet, fazem a conexão ponto a ponto. Neste caso apenas 2 dispositivos podem acessar o meio. Outros sistemas, como o wifi, podem ter vários (em alguns casos, centenas) de dispositivos no mesmo meio. O que torna necessário a definição de quem pode acessar o meio em cada instante. Várias possibilidades existem:

**mestre-escravo** Neste sistema, a comunicação sempre é iniciada pelo mestre, e um escravo só ocupa o meio quando o mestre requisita uma informação. Ex.: USB, Modbus.

**token ring** É passada uma ficha (token) virtual entre os dispositivos. Quem tem a ficha pode falar.

**Multiplexação por tempo** Cada dispositivo tem uma hora específica para controlar o barramento. Um sinal periódico (*NUT- Network Update Time*) sincroniza os dispositivos.

**CSMA-CD – *Carrier Sense Multiple Access with Collision Detection***  
Sempre que um dispositivo quer se comunicar com outro, ele checa antes se o meio está livre. Ex.: wi-fi, bluetooth, CAN.

Tipicamente os protocolos da camada de enlace acrescentam uma certa redundância ao dado transmitido, o que permite detectar e até corrigir erros de transmissão. São os chamados códigos corretores de erro, dentre os quais se utiliza bastante o CRC - *Cyclic Redundancy Check*.

## Rede

Camada responsável pelo roteamento da informação. Ou seja: Se A quer se comunicar com D, mas A apenas tem ligação direta com B e C, para quem A deve mandar a informação?

Tipicamente este roteamento é feito através de uma tabela que mapeia o endereço lógico de um dispositivo (o endereço usado nas etapas acima, tais como o IP) e o endereço físico, usado pela camada de enlace (muitos sistemas usam o *MAC address*). Esta é a última camada a que uma comunicação chega num dispositivo que não o inicial ou final: o programa desta camada checa se o endereço lógico da mensagem é o mesmo daquele dispositivo; não sendo, ele busca na tabela qual o endereço físico para o qual reenviar aquela mensagem e reenvia, tal qual mostra a figura 9.1. Isto serve também para fazer a conexão entre diferentes redes.

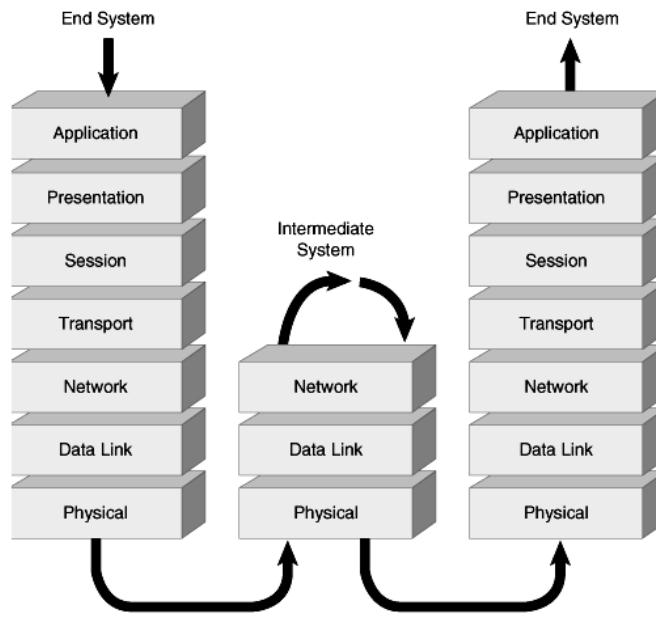


Figura 9.1: Comunicação passando por dispositivo intermediário.

## Transporte

A camada de transporte cuida da comunicação entre o ponto inicial e final. Usa apenas o endereço lógico. Esta camada pode resolver quebrar uma mensagem grande em vários pacotes, mandando um pacote de cada vez para a camada de rede e remontando a mensagem no recebimento.

Um conceito interessante utilizado na camada de transporte é o de *Quality of Service – QoS*, qualidade de serviço. Um sistema com alto QoS garante a chegada de todos os pacotes, na ordem em que foram enviados.

A internet foi originalmente pensada como um sistema de baixo QoS, voltado para a transmissão de arquivos. A ordem dos pacotes não interessa e nem o atraso de um pacote. Caso um pacote se perca, simplesmente pede-se que seja reenviado. Uma rede de telefonia, por outro lado, busca garantir que o tempo de cada pacote seja o mesmo, para não piorar a qualidade da voz transmitida.

Para diversas aplicações industriais, o QoS é importante: não se pode aceitar que uma mensagem de alarme, por exemplo, demore muito a chegar. EtherCAT - *Ethernet for Control Automation Technology* é uma modificação de Ethernet para mensagens com diferentes QoS.

## Sessão

Esta camada cria, gerencia e termina conexões entre 2 pontos de uma rede. É mais comum na telefonia, onde se gera um caminho fixo para uma determinada ligação. Basicamente esta camada gera a tabela que é usada pela camada de rede.

## Apresentação

Faz a mudança no formato dos dados. Exemplo: troca de quebra de linha entre sistemas Unix e Windows, mudança de codificação windows 1252 para UTF, etc.

Inclui também a criptografia, que é a base para garantir a privacidade da comunicação ponto-a-ponto.

## Aplicação

A aplicação final

### 9.1.2 Internet

A Internet define uma pilha de apenas 4 protocolos: o de enlace, que engloba também o físico; o IP – *Internet Protocol*, equivalente ao de rede do modelo OSI; o de transporte, que pode ser um entre vários, sendo o TCP e o UDP os mais comuns e o de aplicação, que envolve sessão, apresentação e aplicação num só.

A figura 9.2 mostra a sequência que é feita com uma informação passada por http pela internet.

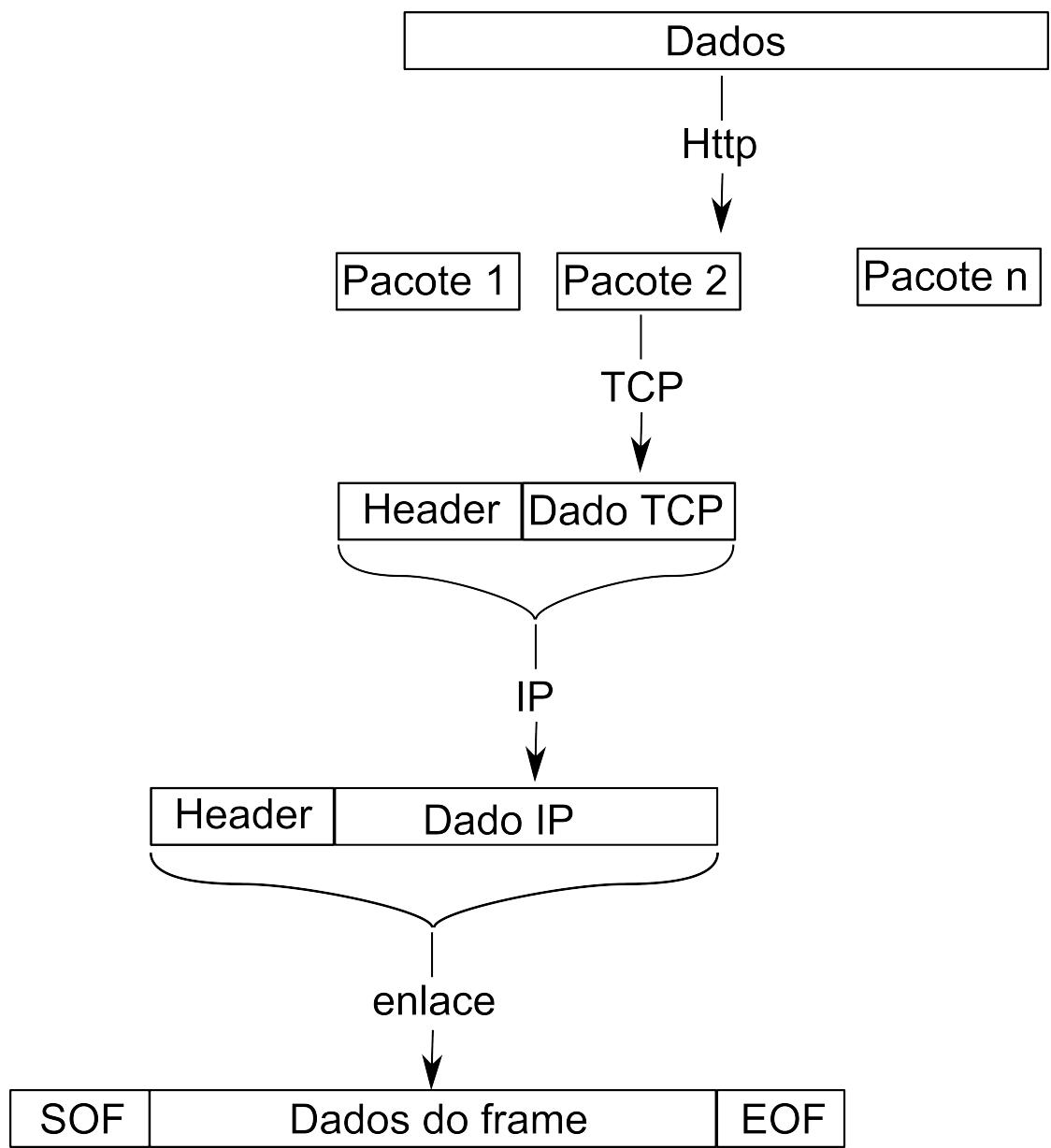


Figura 9.2: Encapsulamento da informação na pilha TCP/IP

## 9.2 Protocolos Industriais

Existem vários protocolos de comunicação industrial para diferentes aplicações. Dentre estes, podemos destacar 3 tipos de aplicação mais comuns:

1. Comunicação entre CLP e sensores e atuadores simples (nível 1 e nível 0).
2. Comunicação entre elementos do nível 1 (CLPs, CNCs, inversores, servomotores) e destes com supervisórios (nível 2).
3. Comunicação entre supervisórios e sistemas de gerência (nível 2 e nível 3).

Redes de comunicação do tipo 1 normalmente conectam um elevado número de dispositivos e requerem pequenos atrasos na comunicação, embora transmitam uma pequena quantidade de dados. Isto implica em um hardware simplificado, para permitir seu uso numa maior quantidade de dispositivos, e um protocolo determinístico, que permita controlar finamente o momento de aquisição de dados. Isto normalmente é conseguido usando uma camada física e de enlace própria da rede, tais como em redes Modbus RTU, HART, CAN, e AS-i.

Comunicações do tipo 2 tem as mesmas características de requerer pequenos atrasos, mas acoplada a uma quantidade de dados bem maior, como por exemplo num robô industrial, em que se transmite a posição de cada um de seus eixos. Isto pode ser obtido com uma camada física e de enlace própria, como na Profibus DP, o que é custoso, tanto em termos de desenvolvimento quanto implementação, ou pela aplicação de uma camada ethernet como base da rede, como nas redes Modbus TCP, Profinet e EtherCAT.

Já comunicações do tipo 3 são normalmente entre computadores e aí o uso delas sobre internet já é bem mais comum, como no caso de OPC-UA.

A tabela 9.1 lista algumas características destes protocolos de redes industriais, para facilitar a comparação entre eles.

### 9.2.1 Modbus

O protocolo Modbus foi criado em 1979 e a princípio independe das camadas físicas e de enlace, definindo um pacote de transmissão da camada de transporte. Isto permite que se estabeleça uma rede Modbus com comunicação serial, sobre internet, usando links de rádio ou até mesmo através de mensagens SMS.

Na prática, existem duas versões mais comuns de Modbus: Modbus RTU, que utiliza uma conexão serial como RS485, RS432 ou RS422, e Modbus TCP, que se conecta pelo protocolo TCP/IP.

A comunicação Modbus é feita através de pacotes, que se diferenciam entre a versão RTU e TCP, como pode ser visto nas tabelas 9.2 e 9.3.

Tabela 9.1: Resumos de protocolos de redes industriais

Protocolo	ano	velocidade	número de pontos	características
Modbus RTU	1979	~1 Mbps	254	Mestre escravo. Roda sobre RS432 ou RS485
Modbus TCP	-	definido pela rede	ilimitado	Mestre escravo. Roda sobre TCP-IP
HART	1986		15	Definido como um sinal digital sobre um sinal 4-20 mA.
CAN	1987		indefinido	Provedor-. Acesso por CSMA-CD.
Profibus DP	1989	12 Mbps	126	
AS-i	1994		62	Comunicação e alimentação sobre apenas 2 fios.
EtherCAT	2003			
OPC-UA	2006			

Tabela 9.2: Pacote Modbus RTU.

<b>Campo:</b>	início	endereço	função	dados	CRC	fim
<b># bytes:</b>	3,5	1	1	$n$	1	3,5

No modbus RTU, os campos de início e fim na verdade indicam um tempo mínimo em que o canal deve ficar em silêncio. O endereço é o endereço de cada dispositivo, onde o mestre é o zero e podem ter até 254 escravos, dependendo das restrições da camada inferior. A função é o comando que está sendo dado pelo mestre ou respondido pelo escravo e a quantidade de dados depende justamente desta função. O CRC é *Cyclical Redundancy Check*, um código de detecção de erros.

No Modbus TCP, manda-se o pacote por TCP/IP, pela porta 502. O ID de transação define que pedaço da comunicação está sendo efetuado (0 para pedido de leitura, 1 para a resposta, etc.) e o ID de protocolo indica o protocolo que está sendo usado (valor é 2). O endereço é em geral ignorado, pois se usa diretamente o IP, e não se tem nem campos de início, fim ou CRC, que já são implementados na pilha TCP/IP.

O Modbus define 4 tipos de dados: **entradas discretas** ou **discrete inputs**,

Tabela 9.3: Pacote Modbus TCP.

Campo:	ID de transação	ID de protocolo	tamanho pacote	endereço	função	dados
# bytes:	2	2	2	1	1	$n$

que são binários apenas de leitura; **bobinas** ou **coils**, binários de escrita e leitura; **registradores de entrada** ou **input registers**, que são *words* de 16 bits de leitura e **holding registers**, de 16 bits e de leitura e escrita. Com base nestes tipos, as funções disponíveis mais comuns são:

- 1** – ler bobinas.
- 2** – ler entradas discretas.
- 3** – ler registradores holding.
- 4** – ler registradores de entrada.
- 5** – escrever numa única bobina.
- 6** – escrever num único registrador holding.
- 15** – escrever em múltiplas bobinas.
- 16** – escrever em múltiplos registradores holding.

### 9.2.2 HART

HART é a sigla para *Highway Addressable Remote Transducer*. Tem como principal característica ser um sinal digital enviado sobre um sinal analógico de 4 a 20 mA. Tem seu uso como um protocolo ponto-a-ponto quando usado com esta característica, onde um tipo de sinal fica sendo transmitido de forma analógica enquanto outros, tais como outras medições, parâmetros do sensor ou sinais de calibração, são transmitidos de forma digital.

Um outro uso é efetivamente como rede, fazendo uma ligação em paralelo (*multi-drop*) de vários dispositivos, alimentados através da corrente do loop, que neste caso fica fixa em 4 mA. Dependendo da versão do protocolo, é possível conectar até 15 ou 64 dispositivos.

Como o HART manda o sinal digital sobre o de 4 a 20 mA, ele não pode definir 0 e 1 como uma grande diferença de tensão. Além disso, precisa de alguma forma enviar o dado serial sem o uso de um sinal de clock. Isto é conseguido usando a modulação FSK – *Frequency Shift Keying*, onde o bit 0 é transmitido como um

sinal alternado de baixa amplitude ( $0,5\text{ mA}$ ) com frequência  $2200\text{ Hz}$  e o 1 como um sinal da mesma amplitude na frequência  $1200\text{ Hz}$ .

É um protocolo mestre-escravo, tendo uma taxa de transmissão pequena, capaz de 2 mensagens por segundo. Pode também ser configurado pelo mestre para usar um modo burst, onde o escravo fica mandando dados continuamente, a uma taxa de 3 mensagens por segundo.

Assim como no Modbus, usa um endereço para definir o escravo com que deseja se comunicar e um campo de comando, equivalente ao de função do Modbus. Tem alguns comandos HART que são aceitos por todo dispositivo HART, outro conjunto de *best practices* que é implementado pela maioria dos dispositivos e pode ainda ter comandos específicos para determinado dispositivo.

### **9.2.3 CAN**

Originalmente significava *Car Area Network*, pois foi criado pela Bosch para fazer uma rede de sensores dentro de um carro. Atualmente tem seu uso em redes de sensores industriais, servindo de base para outros protocolos, tais como o Device-Net.

#### **9.2.4 Profibus DP**

#### **9.2.5 AS-i**

#### **9.2.6 EtherCAT**

#### **9.2.7 OPC-UA**