FACULDADE DE TECNOLOGIA SENAI MARIANO FERRAZ
CURSO DE PÓS-GRADUAÇÃO EM AUTOMAÇÃO E CONTROLE

RENATO DE PIERRI

DESENVOLVIMENTO DE APLICAÇÃO JAVA PARA COMUNICAÇÃO COM O CLP UNITRONICS V350 USANDO O PROTOCOLO TCP/IP

SÃO PAULO
2015

RENATO DE PIERRI

DESENVOLVIMENTO DE APLICAÇÃO JAVA PARA COMUNICAÇÃO COM O CLP UNITRONICS V350 USANDO O PROTOCOLO TCP/IP

Monografia apresentada à Faculdade de Tecnologia SENAI Mariano Ferraz, como requisito para obtenção do título de Especialista em Automação e Controle.

Orientador: Prof. Douglas da Serra Ogata

São Paulo

2015

P622 Pierri, Renato de
Desenvolvimento de aplicação Java para comunicação com o CLP Unitronics V350 usando o protocolo TCP/IP / Renato de Pierri. – São Paulo, 2015.
91 f. il.

Inclui bibliografia.

Monografia (Pós-Graduação) – Faculdade Tecnologia SENAI “Mariano Ferraz”. Curso Automação e Controle.
Orientador: Prof. Douglas da Serra Ogata

1. Automação industrial. 2. CLP. 3. Java. I. Título.

CDD 629.89

RENATO DE PIERRI

DESENVOLVIMENTO DE APLICAÇÃO JAVA PARA COMUNICAÇÃO COM O CLP UNITRONICS V350 USANDO O PROTOCOLO TCP/IP

Monografia apresentada à Faculdade de Tecnologia SENAI Mariano Ferraz, como requisito para obtenção do título de Especialista em Automação e Controle.

_____

Douglas da Serra Ogata
Orientador

Banca examinadora

_____

Prof. Douglas da Serra Ogata
SENAI – Serviço Nacional de Aprendizagem Industrial

_____

Prof. Ms. Daniel Barbuto Rossato
SENAI – Serviço Nacional de Aprendizagem Industrial

_____

Prof. Ms. José Ricardo da Silva
SENAI – Serviço Nacional de Aprendizagem Industrial

São Paulo, 2 de Dezembro de 2015.

Dedico esse trabalho a todos os meus colegas de serviço da extinta SID Informática que, à época, técnicos eletrônicos assim como eu, assumiram a manutenção dos caixas eletrônicos do cliente Bradesco e tiveram que lavar cada um dos equipamentos, para depois fazer os devidos reparos e os colocar em funcionamento. Heróicos tempos da hora extra infinita, em que chupa cabra era coisa de folclore e nos sentíamos os faxineiros mais bem pagos do mundo. A vida seguiu, e na educação continuada, compartilho com meus amigos o pouco que aprendi.

RESUMO

Este trabalho é baseado no controlador lógico programável Unitronics *Vision* 350 e discorre sobre as funcionalidades de um controlador lógico programável e de sua interface humano máquina. Apresenta suas principais configurações, aborda a elaboração de circuitos para testar suas entradas e saídas analógicas e digitais, bem como propõe dois métodos para comunicação de dados empregando o protocolo TCP/IP, permitindo a escrita e leitura de dados tanto no controlador lógico programável como em um banco de dados hospedado em um servidor Linux, sem utilizar o padrão de interoperabilidade de automação industrial OPC.

Palavras-chave: Controlador lógico programável. Java. Linux. Banco de dados. TCP/IP. *Ethernet*.

ABSTRACT

This work is based on the Unitronics Vision 350 programmable logic controller and discusses the functionality of a programmable logic controller and its human machine interface. Presents its main settings, addresses the development of test circuits for its analog and digital inputs and outputs, as well as propose two methods for data communication using the TCP/IP protocol, allowing the reading and writing of data in the programmable logic controller and also in a database hosted on a Linux server, without using the interoperability standard for industrial automation OPC.

Keywords: Programmable logic controller. Java. Linux. Database. TCP/IP. Ethernet.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
AD	Analógico – Digital
ASCII	*American Standard Code for Information Interchange*
CLP	Controlador Lógico Programável
ERP	*Enterprise Resource Planning* – Planejamento dos Recursos da Empresa
IHM	Interface Humano Máquina
IP	Protocolo de Internet
OPC	Padrão de interoperabilidade de automação industrial
PLC	*Programmable Logic Controller* – Controlador Lógico Programável
SCADA	*Supervisory Control And Data Acquisition* – Controle Supervisório e Aquisição de Dados
TCP	*Transmission Control Protocol* - Protocolo de Controle de Transmissão
TI	Tecnologia da Informação
XML	*Extensible Markup Language* – Linguagem de Marcação Estendida

SUMÁRIO

1 **INTRODUÇÃO**

No decorrer do curso de pós-graduação de automação e controle, um dos temas recorrentes das aulas foi a apresentação da pirâmide de automação, segmentando as áreas de chão de fábrica, dados, tecnologia da informação e a área de planejamento de recursos da empresa (ERP) (NOF, 2009, p. 804).

Em uma das aulas nos foi apresentado o padrão de interoperabilidade para automação industrial chamado de OPC, livre, que tem por finalidade padronizar a transferência de dados entre as camadas chão de fábrica / dados, com a camada da área de tecnologia da informação (TI), assegurando que os dados das camadas inferiores sejam transmitidos de maneira homogênea para as camadas de TI.

Ao pesquisar um pouco mais sobre o padrão no site da organização OPC *Foundation* (https://opcfoundation.org), constatou-se que as especificações são fornecidas apenas aos associados à fundação, livre de quaisquer ônus, desde que o associado esteja em dia com suas obrigações.

Para se associar à fundação, tem que se utilizar um e-mail corporativo, não serve e-mails do Gmail, Yahoo, Hotmail e outros. Ao tentar a filiação, seguindo as regras do site, somente foi possível se cadastrar utilizando um e-mail de um servidor localizado nos Estados Unidos, domínio ".com".

Conforme informado no e-mail de cadastramento, OPC *Foundation* (2015), o custo da associação para se ter acesso ao documento de especificação do padrão OPC varia entre US$900,00 e US$18.000,00 de acordo com o tamanho da empresa.

Devido ao custo e à dificuldade de se conseguir acesso à especificação do padrão OPC, optou-se por não se concluir a filiação à fundação.

Observando o mercado, nota-se que além dos fornecedores tradicionais de produtos e soluções de automação industrial, outros fabricantes vêm apresentando soluções de baixo custo que dispõem de um ambiente de desenvolvimento colaborativo, multiplataforma, de *software* realmente livre, que suporta conexão à banco de dados.

Essas soluções incluem interfaces e sensores de fácil integração que podem ser aproveitados pela indústria inicialmente em situações não críticas e eventualmente se estendendo a outras áreas, ou seja, o mercado está evoluindo e é necessário e oportuno que se levantem as seguintes questões, mesmo que recorrentes:

- O que é e para que serve um CLP e a IHM?
- Quais as principais configurações e sinais de um CLP?
- Como testar as entradas de um CLP?
- Como um CLP poderia gravar em um banco de dados sem utilizar OPC?
- Como uma aplicação poderia ler dados de um CLP sem utilizar OPC?

Longe de querer levantar a bandeira a favor de uma ou outra solução ou dizer que este trabalho traz um ponto final a um assunto tão extenso, as respostas aos itens acima procuram levar o profissional da área de automação a avaliar as alternativas disponíveis para integrar as informações geradas por um CLP às demais camadas de um sistema de automação conforme descrito por Nof (2009, p. 804).

1.1 **Objetivo**

Esse estudo tem a intenção de apresentar uma visão geral do controlador lógico programável (CLP), sua finalidade, suas entradas e saídas, descrever em linhas gerais a finalidade da interface humano máquina (IHM), abordar um método que permita ao CLP persistir dados em uma base de dados e implementar em Java o protocolo de comunicação ASCII do CLP em estudo, permitindo que uma aplicação rodando em um computador leia e escreva dados no CLP pela interface *Ethernet* utilizando o protocolo TCP/IP.

1.2 **Justificativa**

De acordo com Pierro, Joia e Ribeiro (2001) as mudanças tecnológicas e socioculturais contemporâneas nos impõem a necessidade de atualização constante do conhecimento.

Ao mesmo tempo, segundo o SENAI São Paulo (http://www.sp.senai.br/Senaisp/WebForms/Cursos/CursosTipos.aspx?Tipo=117&Menu=33), o curso de Pós Graduação em Automação e Controle oferece aos alunos a possibilidade de especialização, ampliando a formação inicial obtida no curso superior e ampliando o leque de oportunidades de atuação profissional, afirmação alinhada com Pierro, Joia e Ribeiro (2001, p.13) que afirmam que a educação

continuada visa "responder às múltiplas necessidades formativas que os indivíduos têm no presente e terão no futuro", indicando que o indivíduo pode e deve prosseguir com seus estudos, suprindo suas necessidades específicas.

Em outra linha de raciocínio, diversa da abordagem inicial, existe a indústria de automação em constante evolução que já ha algum tempo vem oferecendo itens como CLPs com IHM incorporada entre outras soluções, fora o lançamento no mercado de uma série de dispositivos inteligentes, com alta capacidade de processamento, baixo consumo de energia, incluindo a possibilidade de comunicação *Ethernet* e WiFi utilizando protocolo TCP/IP, permitindo fácil integração a sistemas de bancos de dados.

Embora esses dispositivos (*Raspberry*, *Arduino*, *BeagleBone* e kits de desenvolvimento em geral) não sejam CLPs como a indústria esteja acostumada a utilizar, eles são de baixo custo, contam com um ambiente de desenvolvimento colaborativo e multiplataforma, incluindo uma família de interfaces e sensores de fácil integração que podem eventualmente serem utilizados em aplicações industriais em contraponto aos sistemas e soluções proprietárias oferecidas pelos fabricantes tradicionais de CLPs.

Em resposta a esse cenário que demanda o aprimoramento contínuo do profissional de automação, esse trabalho além de visar ampliar o leque de oportunidades de atuação profissional, se justifica pela necessidade de se discutir um ou mais métodos para que um CLP possa gravar informações em uma base de dados utilizando a interface *Ethernet* sem necessidade do emprego de interfaces OPC e utilizando soluções de código aberto, empregando Java nesse caso, mostrando que é possível desenvolver sistemas que empregam tecnologia de código aberto para uso com CLP a exemplo das utilizadas pelos dispositivos inteligentes que vem sendo lançados no mercado.

1.3 **Metodologia**

Para atender as necessidades desse trabalho, serão detalhadas as características do CLP escolhido, serão abordados os tipos de sinais de entrada e saída digitais e analógicos suportados pelo CLP, propondo alguns circuitos de teste simples bem como trabalhando sua plataforma de desenvolvimento.

Serão apresentadas duas técnicas para disponibilizar os dados gerados por um CLP a um sistema computacional via interface de rede *Ethernet* utilizando protocolo TCP/IP, sem adotar o padrão de interoperabilidade de automação industrial (OPC).

Na primeira técnica, denominada "TCP *server*", o CLP enviará os dados a um servidor Linux a intervalos regulares e os dados enviados serão persistidos numa base de dados MySQL.

Na segunda técnica, chamada "TCP *client*" uma aplicação Java rodando no computador Linux irá implementar o protocolo de comunicação ASCII do fabricante Unitronics, permitindo que os registros internos do CLP sejam acessados pela interface *Ethernet* do CLP, utilizando o protocolo TCP/IP e sem empregar o módulo OPC.

Esta aplicação permitirá que os registros internos do CLP sejam lidos e escritos a partir do computador.

Esta monografia parte do pressuposto que a instalação e configuração da infraestrutura necessária ao suporte do servidor Linux está pronta, assim como seu ambiente de desenvolvimento Java, *firewall*, máquinas virtuais, Eclipse e seus *plugins*.

2 **REFERENCIAL TEÓRICO:**

2.1 **CLP**

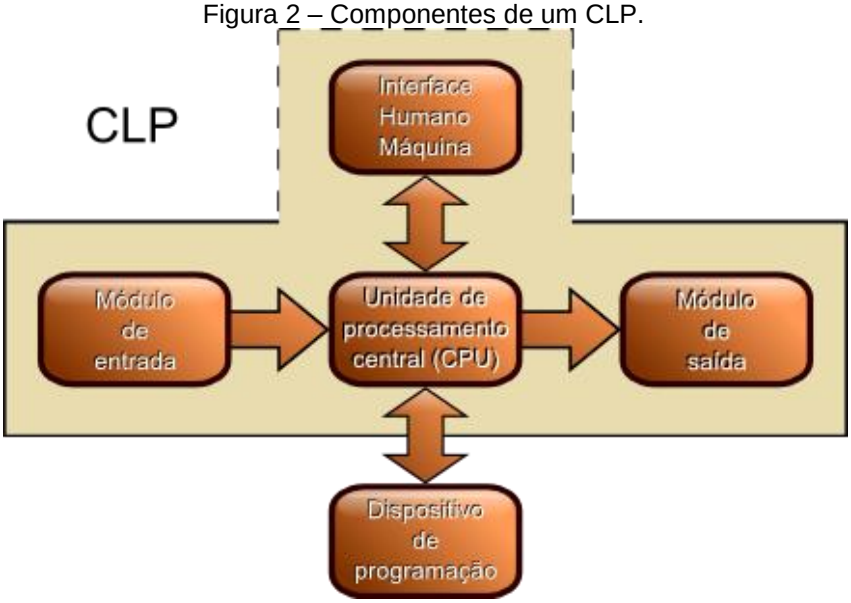
Antes do advento do CLP, para automatizar um processo ou uma linha de produção somente eram utilizados relés, contatores e outros dispositivos eletromecânicos. A solução tinha de ser projetada, componentes especificados, feita a montagem da fiação altamente complexa, sendo cara, de difícil manutenção e alteração custosa, tanto que uma solução desse tipo é referida pelo termo "*hard-wired*", segundo Siemens (2014, p. 6), termo que transmite o conceito de "permanentemente conectado", "que não permite mudanças" (MERRIAM-WEBSTER, 2014). Na figura 1 abaixo, *Internet Pinball Machine Database* (2015) mostra parte do interior da máquina de fliperama “*Bow and Arrow*” da empresa “*Bally Manufacturing Corp*” que utiliza apenas componentes eletromecânicos para automatizar o registro da pontuação do jogador bem como controlar o comportamento da bolinha no campo de jogo. A dificuldade em se mudar a programação da máquina refazendo a fiação dos relés exemplifica o conceito "*hard-wired*".

Figura 1 – Automação na indústria do entretenimento.

Fonte: INTERNET PINBALL MACHINE DATABASE, 2015.

Segundo Alves (2013, p. 178), O CLP foi desenvolvido por especificações da indústria automobilística a partir de 1968. O CLP tinha por função reduzir os custos de manutenção e instalação dos painéis de controle industriais que utilizavam relés e Siemens (2014, p. 4) complementa dizendo que PLC ou CLP em português, é o acrônimo para o termo "Controlador Lógico Programável". CLP é um computador que possui um *hardware* e sistema operacional específico para uso em aplicações industriais e comerciais, constituído por módulo de entrada, unidade de processamento e módulo de saída, podendo incorporar uma interface humano máquina (IHM) conforme figura 2. Ainda de acordo com a figura 2, há o bloco à parte chamado "Dispositivo de programação" que ao ser conectado no CLP, tem por função programar monitorar ou configurar valores armazenados no CLP, não fazendo parte do CLP propriamente dito.

Figura 2 – Componentes de um CLP.



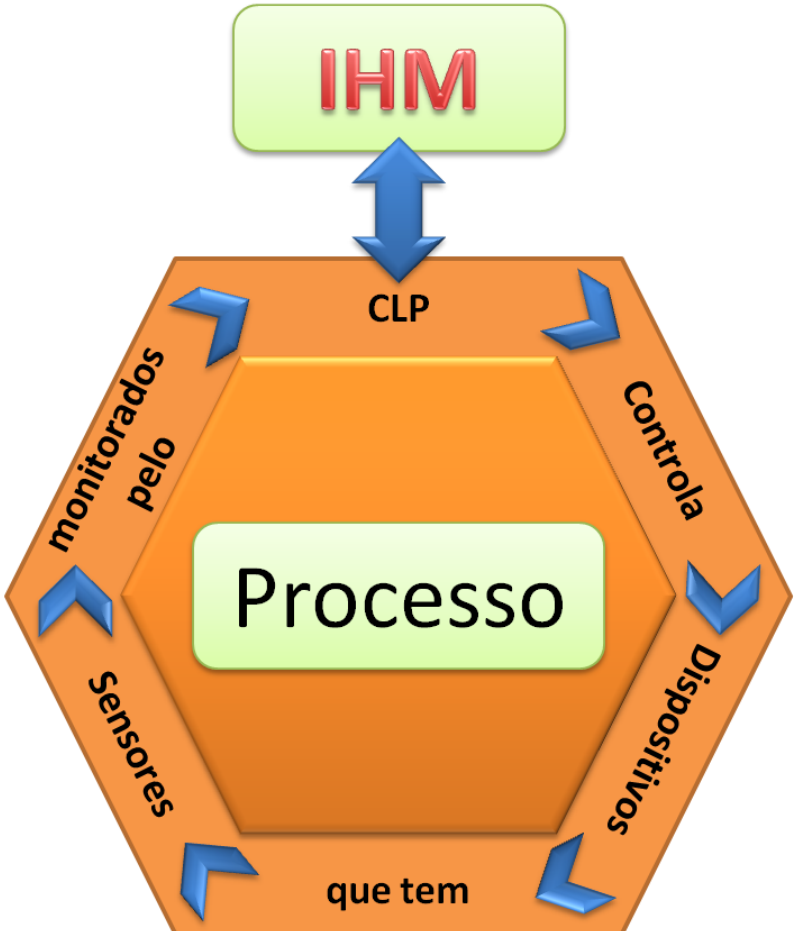
Fonte: Adaptado de Siemens (2014, p. 5).

O CLP pode ser utilizado para automatizar uma linha de montagem, processos em refinarias de petróleo, plantas de tratamento de água bem como praticamente qualquer atividade que envolva uma série de passos a serem executados em uma determinada sequência e condições específicas. Basicamente o CLP funciona coletando dados do processo pelo módulo de entrada como informações vindas de outros CLPs (comunicação de dados), leituras de temperatura, pressão, velocidade e sinais digitais entre outros, faz o processamento dessas grandezas de acordo com o programa armazenado na unidade de processamento, cujo resultado é enviado ao módulo de saída que pode, entre outras opções, se comunicar com outro dispositivo, acionar um motor, uma válvula, acender uma lâmpada ou mesmo persistir o resultado do processamento em uma base de dados.

2.2 **IHM**

Para Quezada-quezada et al. (2014) a interface humano máquina (IHM) serve para representar na forma de texto ou gráfica em um mostrador, dispositivos de uma planta como: motores, tanques, válvulas, aquecedores e reatores entre outros, além de grandezas monitoradas por sensores como temperatura, pressão, vazão, nível entre outros, exibindo informações relativas ao processo em andamento na mesma, conforme representado na figura 3.

Figura 3 – Situando a IHM.



Fonte: Elaborado pelo autor.

As informações retornadas pelos sensores ao CLP são direcionadas para a IHM que as converte em mensagens de texto ou elementos gráficos como alterando a cor de uma válvula de acordo com seu estado, dispensando ao operador, por exemplo, da necessidade de se escalar em tanques ou inspecionar presencialmente equipamentos em áreas classificadas a fim de se coletar informações relativas ao andamento do processo.

2.2.1 **Entrada de dados na IHM**

Pela IHM, um operador pode ler as variáveis do processo, parametrizar CLPs, interromper processos, trocar receitas, atender sinais de alarmes apresentados pela IHM, comandar e acompanhar visualmente na IHM o processo suportado pela planta a partir de uma área segura, sem necessidade de acesso direto à mesma.

De acordo com Weg (2015) e Unitronics (2015a), figura 4 abaixo apresenta dois exemplos de CLPs com IHM incorporada.

Figura 4 – Exemplos de CLPs com IHM incorporada.

Fonte: WEG, 2015 e UNITRONICS, 2015a.

2.3 **CLP e os sistemas de informação industrial**

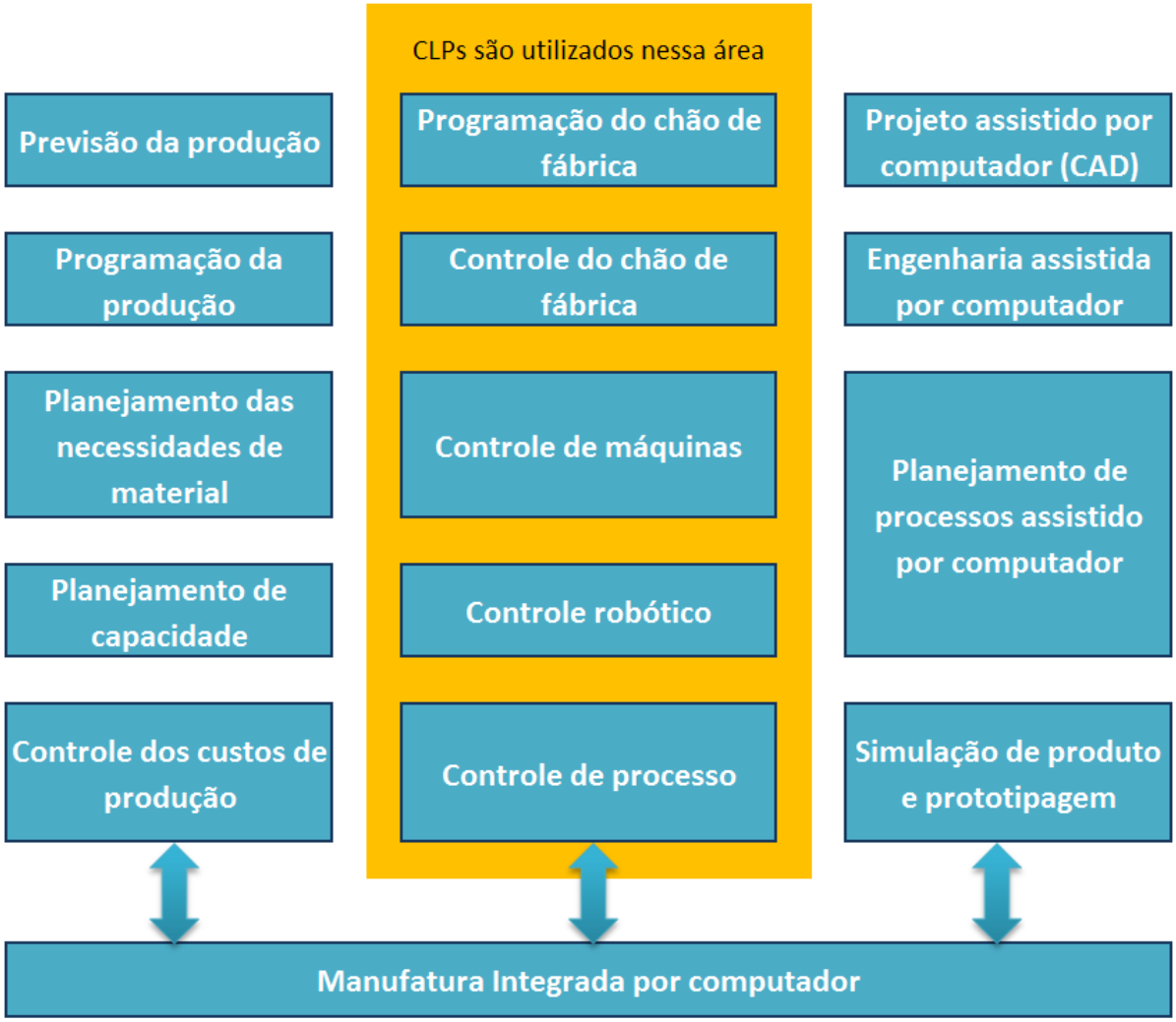
Segundo O'Brien (2001, p. 178-181) é possível identificar diversos sistemas de informação que apoiam a manufatura integrada por computador. Manufatura integrada por computador trata-se de um conceito global que vem a enfatizar o uso de sistemas computadorizados com o objetivo de simplificar e automatizar os processos de produção e as funções organizacionais, facilitando a integração dos processos de produção pelo uso de redes de telecomunicações bem como aplicando conceitos da tecnologia da informação.

Ainda de acordo com O'Brien (2001, p. 179) os computadores podem ser utilizados para controlar processos em refinarias de petróleos bem como em outros setores industriais.

São computadores dedicados, com finalidades especiais chamados controladores lógicos programáveis. Esses computadores controlam processos industriais monitorando grandezas físicas, as quais são disponibilizadas ao computador via sensores digitais ou conversores analógico – digitais. Uma vez coletadas as informações, elas são processadas, seguindo algoritmos e modelos matemáticos, de acordo com a programação do CLP.

De acordo com a figura 5, os dados coletados e processados pelo CLP podem ser integrados aos demais setores da produção pelo uso das redes de comunicação e sistema de "*Data Warehouse*", ajudando no gerenciamento do processo industrial como um todo (O'BRIEN, 2001, p.148). Exemplificando: durante a operação de um grupo de equipamentos, dados processados pelos CLPs são alimentados em um "*Data Warehouse*", reportando informações a respeito do consumo dos insumos naquele processo fabril. Por sua vez, a área de planejamento de necessidades de material analisa a taxa de consumo de insumos utilizados pelas máquinas em questão, fazendo a projeção das quantidades necessárias de acordo com a programação da produção e providenciando o correto dimensionamento do estoque, garantindo que o custo de produção fique dentro da meta estimada pela área de controle dos custos de produção.

Figura 5 – Sistemas de informação industrial.



Fonte: Adaptado de O'Brien (2001, p. 179).

3 DESENVOLVIMENTO

3.1 CLP empregado na monografia

Esta monografia é baseada no CLP *Vision* modelo V350-J-T2 com interface de rede padrão *Ethernet*, os resultados apresentados a seguir podem ser diferentes caso outros CLPs sejam utilizados. O CLP foi configurado de acordo com as especificações do fabricante (UNITRONICS, 2015b). O *software* programador Unitronics *Vision* OPLC IDE foi baixado da *internet* e configurado de acordo com instruções contidas no *help* em geral, principalmente no tópico "*Hardware Configuration*" e subtópicos.

3.2 CLP – módulo de entrada

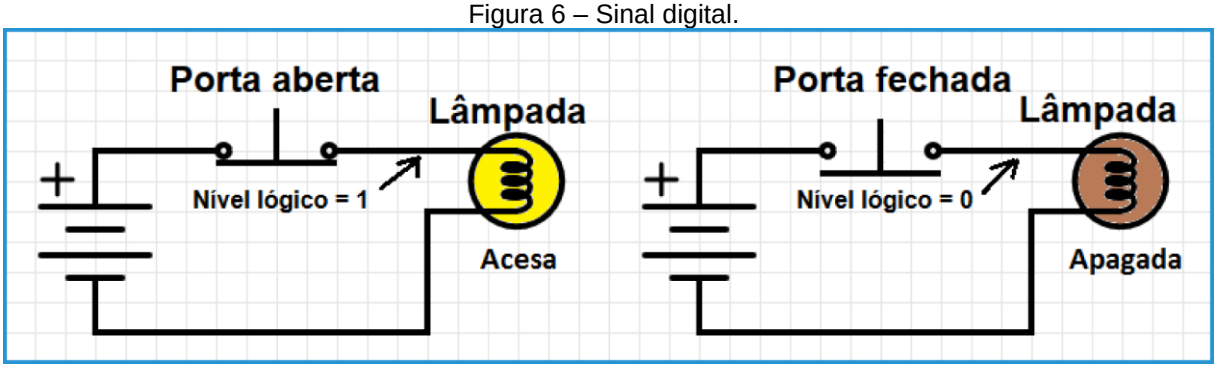
O módulo de entrada do CLP é responsável por fazer a leitura dos sinais provenientes dos sensores, que podem ser classificados em duas classes principais, sinais digitais e sinais analógicos.

3.2.1 Sinal digital

Conforme descrito por Bryan e Bryan (1997, p. 36), para entender o conceito de sinal digital, pode-se considerar um interruptor de luz de cortesia instalado na porta de um automóvel que opere da seguinte forma: quando o interruptor está acionado, ele gera um sinal indicando que a porta do automóvel está aberta, acendendo a luz de cortesia. Ao fechar a porta do veículo, esse mesmo interruptor é desligado, apagando a luz de cortesia e indicando que a porta está fechada.

Conforme diagrama na figura 6, o sinal gerado por esse tipo de interruptor (sensor), que é capaz de apresentar apenas dois estados válidos, indicando que a porta do automóvel ou está aberta (nível lógico = 1) ou está fechada (nível lógico = 0), é chamado de sinal digital.

Figura 6 – Sinal digital.



Fonte: Elaborado pelo autor.

A mesma denominação vale para as entradas do CLP que são capazes de reconhecer esse tipo de sinal. Quando a especificação de um determinado CLP informa que existem 12 entradas digitais, isso significa que o CLP consegue ler e processar essas 12 entradas indicando se o sinal está ou não está presente em cada uma das entradas do CLP.

3.2.2 **Sinal digital no CLP**

Existe uma configuração que influencia na maneira que o CLP converte o sinal elétrico em sinal digital para uso no processamento interno do CLP.

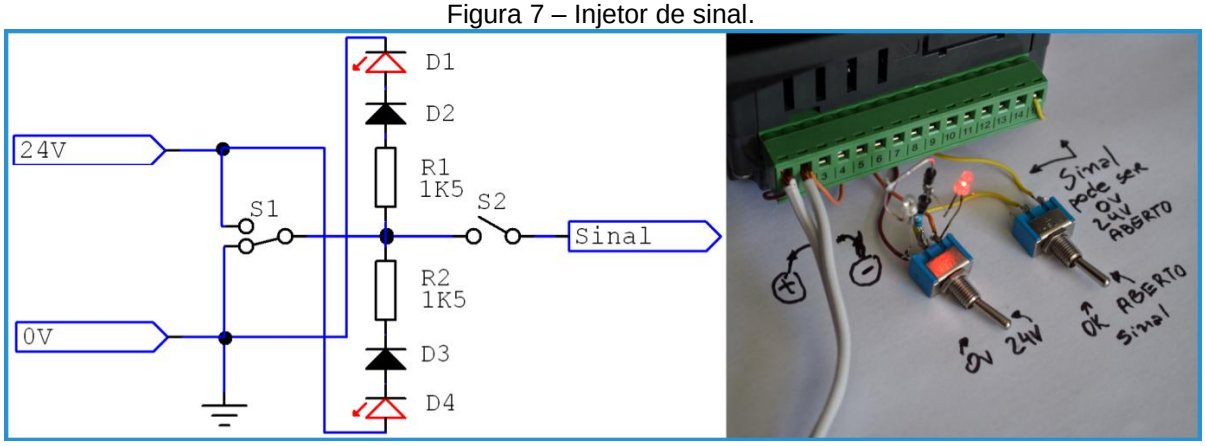
Trata-se da configuração do tipo de ligação, PNP ou NPN, que é configurado por *jumpers* conforme indicado no quadro 4. Essa configuração serve para estabelecer a correspondência entre o nível do sinal elétrico presente na entrada de sinal do CLP com o nível lógico a ser convertido pelo CLP e utilizado na lógica *Ladder*. A entrada do CLP recebe um sinal elétrico a ser interpretado como um sinal digital que pode ter 3 condições: 24V, 0V ou “circuito aberto”, que pode ser um fio desconectado ou um contato aberto de uma chave, por exemplo. Embora “circuito aberto” não seja um nível lógico propriamente dito, o projetista e o técnico de manutenção precisam saber como o CLP interpretaria o sinal de uma determinada entrada caso a mesma esteja com seu circuito aberto.

3.2.3 **PNP e NPN - Testando o funcionamento**

Para identificar como as entradas digitais do CLP se comportam em relação à configuração PNP e NPN, montou-se o seguinte ensaio:

Conforme a figura 7, foi elaborado um circuito capaz de injetar um sinal elétrico em uma entrada do CLP, simulando os três estados possíveis: 24V, 0V e cabo aberto.

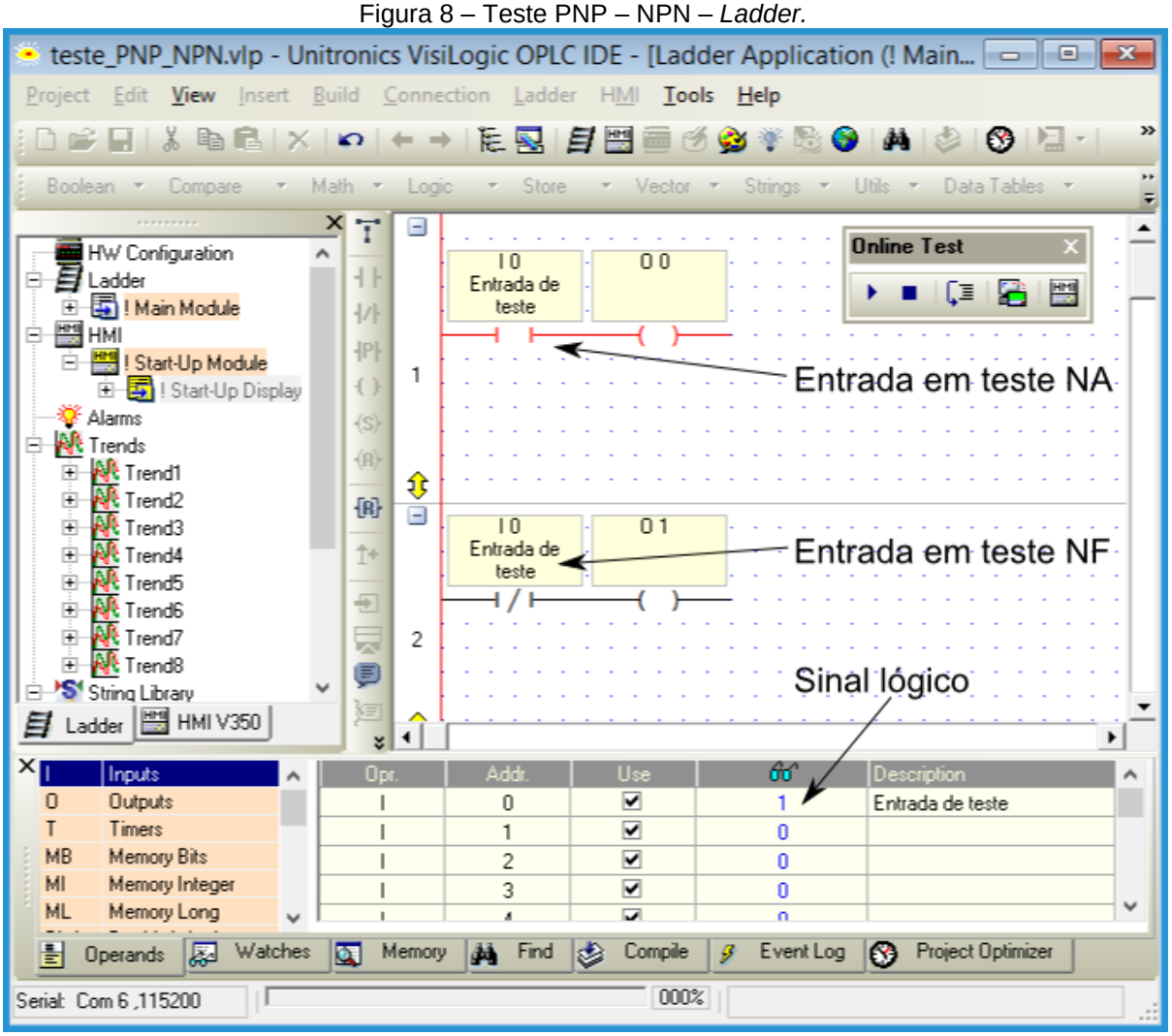
Figura 7 – Injetor de sinal.



Fonte: Elaborado pelo autor.

Conforme a figura 8, montou-se um programa *Ladder* para ler o sinal da entrada em teste, utilizando o *software* de configuração do CLP:

Figura 8 – Teste PNP – NPN – *Ladder.*



Fonte: Elaborado pelo autor.

Uma vez montado o ambiente de teste, foi realizado o ensaio, injetando o sinal e medindo o consumo de corrente na entrada digital do CLP, configurado como PNP e NPN, de acordo com o quadro 4, e observando o resultado no *software* de configuração do fabricante.

A seguir são apresentados os resultados do ensaio, detalhando como o sinal elétrico é interpretado quando a entrada digital do CLP está configurada em modo PNP ou modo NPN, inclusive considerando o caso de circuito aberto.

3.2.4 **Comportamento da ligação PNP**

Conforme o quadro 1, na ligação PNP é considerado nível lógico '1' quando é aplicado 24V na entrada. No nível lógico '1', os contatos NA/NF são comutados e a entrada do CLP consome 8mA de corrente. Zero volts na entrada em análise ou circuito aberto significa nível lógico '0'. Nessa condição, os contatos NA/NF ficam em estado de repouso e a entrada tem um consumo de corrente desprezível.

Quadro 1 – Entrada do CLP configurada no modo PNP.

Tensão de entrada	**Nível lógico**	***Ladder* NA -| |-**	***Ladder* NF -|/|-**	**Consumo (mA)**
0 Volts	0	Aberto	Fechado	≅ 0
24 Volts	1	Fechado	Aberto	8
Desconectado	0	Aberto	Fechado	0

Fonte: Elaborado pelo autor.

3.2.5 **Comportamento da ligação NPN**

Conforme o quadro 2, na ligação NPN é considerado nível lógico '1' quando é aplicado 0V na entrada. Nessa condição, os contatos NA/NF são comutados e a entrada do CLP consome 8mA de corrente. 24V na entrada em análise ou circuito aberto significa nível lógico '0'. Nessa condição, os contatos NA/NF ficam em estado de repouso e a entrada tem um consumo de corrente desprezível.

Quadro 2 – Entrada do CLP configurada no modo NPN

Tensão de entrada	**Nível lógico**	***Ladder* NA -| |-**	***Ladder* NF -|/|-**	**Corrente (mA)**
0 Volts	1	Fechado	Aberto	8
24 Volts	0	Aberto	Fechado	≅ 0
Desconectado	0	Aberto	Fechado	0

Fonte: Elaborado pelo autor.

3.2.6 **Entradas analógicas**

O CLP não trabalha apenas com sinais digitais. Em algumas condições e de acordo com o processo controlado pelo CLP é necessário medir grandezas que seu comportamento apresenta uma variação contínua dentro de uma escala de valores.

Alguns exemplos de sinais analógicos a citar são: temperatura, velocidade, peso, pressão, nível e vazão entre outros.

Essas grandezas analógicas precisam ser representadas digitalmente para poderem ser processadas pelo CLP.

O CLP objeto de estudo possui duas entradas analógicas que serão configuradas para receber sinais que variam entre 0 e 10V (AN0) e de 0 a 20mA (AN1). O quadro 3 apresenta as especificações dessas entradas.

Quadro 3 – Especificação das entradas analógicas do CLP V350-J-T2.

	Corrente		**Tensão**
Tipo de sinal	0 a 20mA	4 a 20mA	0 a 10V
Resolução	10 bits	10 bits menos o range de 0 a 4mA	10 bits
Método de conversão	Aproximação sucessiva		
Unidades de conversão	1023	820	1023
Valor de cada unidade de conversão	incrementos de 20mA/1023	incrementos de 20mA/1023	incrementos de 10V/1023
Fora de escala	Sim, a saída do conversor apresentará o valor 1024		
Precisão	0,9%		
Impedância de Entrada	243 Ω	243 Ω	>150K Ω
Valor máximo de entrada	25mA @ 6V	25mA @ 6V	15V

Fonte: Adaptado de Unitronics (2015c).

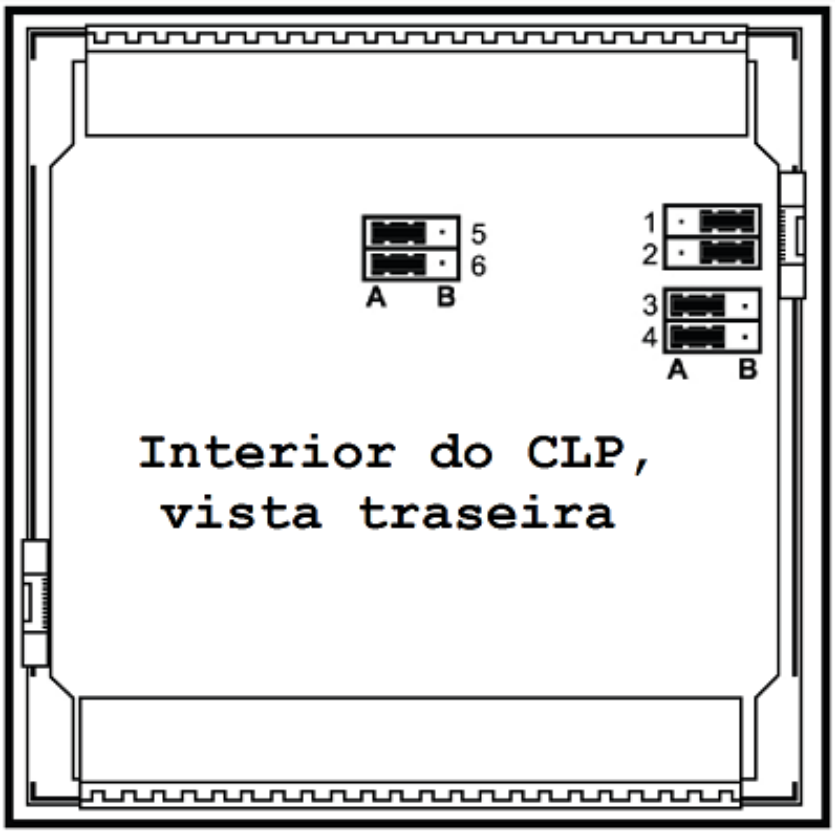
De acordo com Unitronics (2015c), o sinal analógico convertido para digital é representado por um número inteiro que pode assumir qualquer valor entre 0 e 1023 (10bits) que varia de acordo com o nível do sinal de entrada.

O valor 1024 serve para representar que o sinal de entrada ultrapassou o máximo permitido na entrada analógica (condição de *overflow*), conforme indicado no quadro 3.

Respeitando as especificações contidas no quadro 3, esse sinal digital que pode assumir qualquer valor entre 0 e 1024 serve para representar tanto o sinal analógico de “0 a 10V” como os sinais de “0 a 20mA” e de “4 a 20mA”, também analógicos.

O modo como o *hardware* em estudo é configurado para receber o sinal analógico, tensão ou corrente, é configurado por meio de *jumpers*, conforme especificado por Unitronics (2015b) no guia de instalação e adaptado no quadro 4 e na figura 9.

Figura 9 – *Jumpers* do CLP V350-J-T2.



Fonte: UNITRONICS, 2015b.

Quadro 4 – Configuração de *hardware* – CLP V350-J-T2.

Configuração das entradas digitais 0 - 11		
Ajuste	JP1 (todas entradas)	
NPN	A	
PNP	B	
Configuração das entradas 10 e 11 para serem analógicas ou digitais		
Ajuste	JP5 (entrada 10)	JP6 (entrada 11)
Digital	A	A
Analógica	B	B
Configuração das entradas analógicas AN0 e AN1		
Ajuste	JP3 (AN0)	JP4(AN1)
Tensão	A	A
Corrente	B	B

Fonte: UNITRONICS, 2015b.

Uma vez que o *hardware* é configurado conforme o quadro 4, para tratar adequadamente o sinal elétrico injetado nas entradas analógicas, AN0 e AN1 no caso, é necessário indicar no *software* programador VisiLogic em qual endereço de memória será armazenado o resultado da conversão analógico – digital (AD).

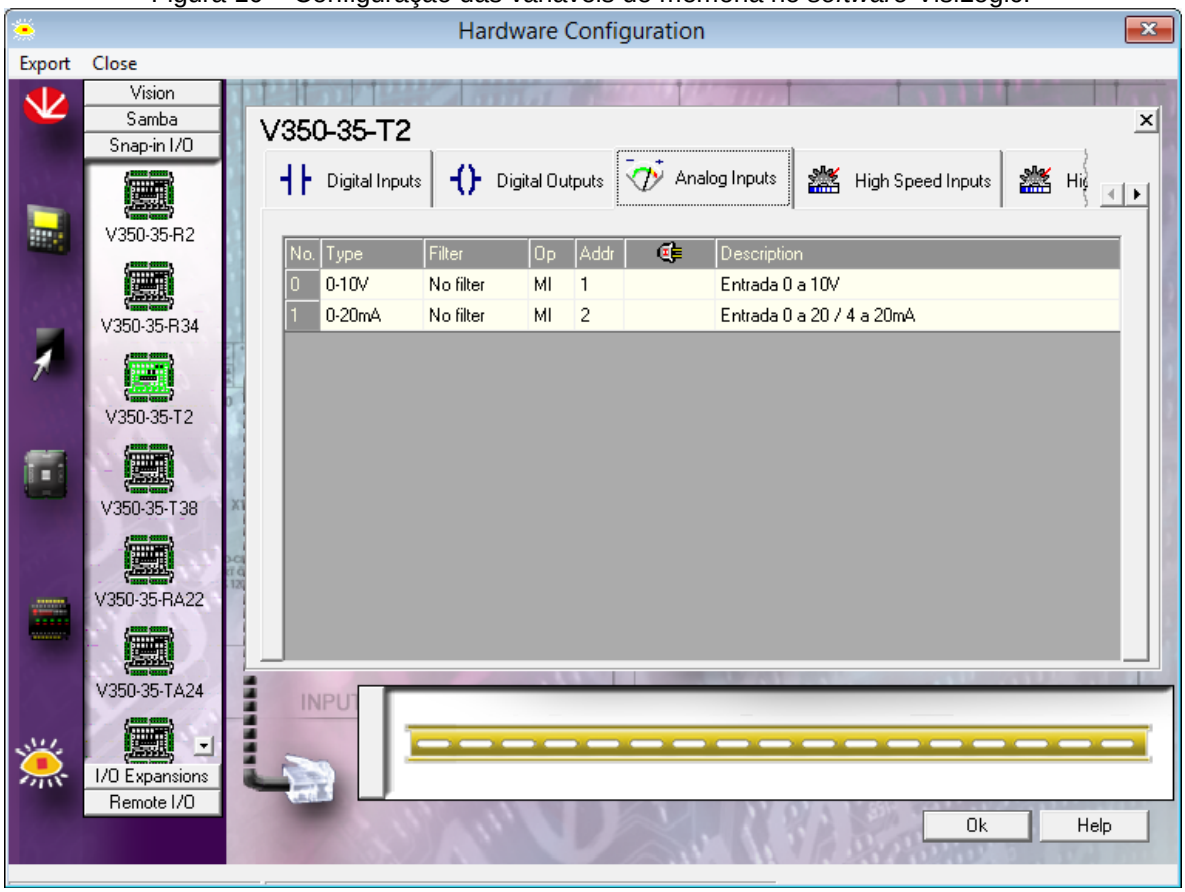
Para configurar o registro de memória onde será armazenado o resultado da conversão AD, deve-se abrir o *software* VisiLogic, ir no menu "*View*", escolher a opção para visualizar as configurações de *hardware*.

Na tela de configuração de *hardware*, deve-se assegurar que o modelo do CLP e opcionais estão corretamente selecionados.

Após identificar o CLP e módulos instalados, basta navegar até a aba "*Analog Inputs*" e configurar seus parâmetros, entre eles, determinar o endereço de memória que irá armazenar o resultado da conversão AD.

A figura 10 mostra a tela de configuração de *hardware* do *software* VisiLogic indicando que a memória MI1 está configurada para receber os valores da conversão AD da entrada AN0 e que a memória MI2 está configurada para receber os valores da conversão AD da entrada AN1.

Figura 10 – Configuração das variáveis de memória no *software* VisiLogic.



Fonte: Elaborado pelo autor.

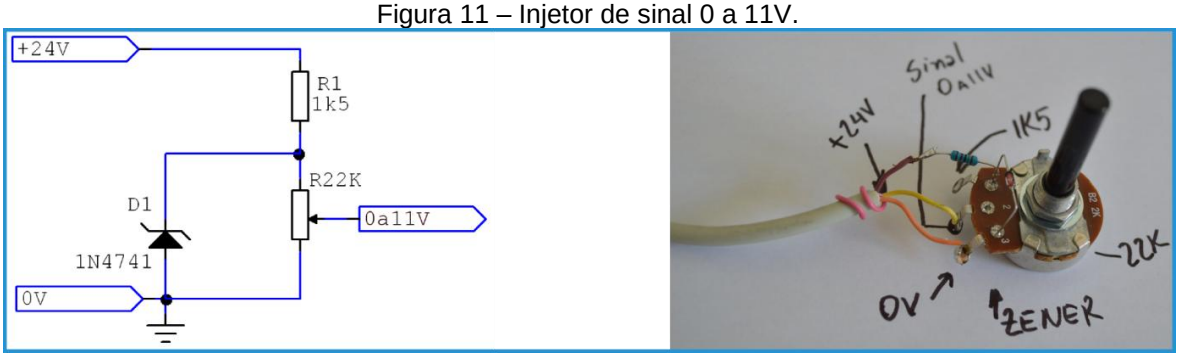
3.2.7 **Testando a entrada analógica 0-10V**

A figura 11 mostra um exemplo de circuito para testar a entrada analógica 0 a 10V do CLP. Note que ela não é isolada. Trata-se de uma fonte de diodo zener que reduz a tensão de alimentação do CLP de 24V para 11V (CIPELLI; SANDRINI, 1980).

Como a entrada 0 a 10V do CLP é de alta impedância, >350kohms, ela consome pouca corrente não havendo necessidade de se montar um circuito mais elaborado.

Pelo potenciômetro é possível regular o sinal de saída entre 0 e 11V, garantindo que toda a escala de tensão do conversor AD seja percorrida, inclusive simulando a condição de *overflow*, não sendo objetivo do teste analisar a precisão da conversão de analógico para digital.

Figura 11 – Injetor de sinal 0 a 11V.



Fonte: Elaborado pelo autor

.

Após a configuração do *hardware* e *software* Visilogic conforme especificado anteriormente, para testar o funcionamento da entrada AN0 (0 a 10V) foi elaborado o programa *Ladder* e carregado no CLP a seguir:

3.2.8 **Programa *Ladder* para testar a entrada 0 a 10V**

A entrada analógica AN0 tem seu valor convertido para digital. O valor obtido é armazenado na variável MI1, conforme definido na configuração de *hardware* (figura 10). A variável MI1 contém um valor inteiro e seu valor pode variar de 0 a 1023 de acordo com a tensão de entrada, indicando o valor 1024 em caso de *overflow*.

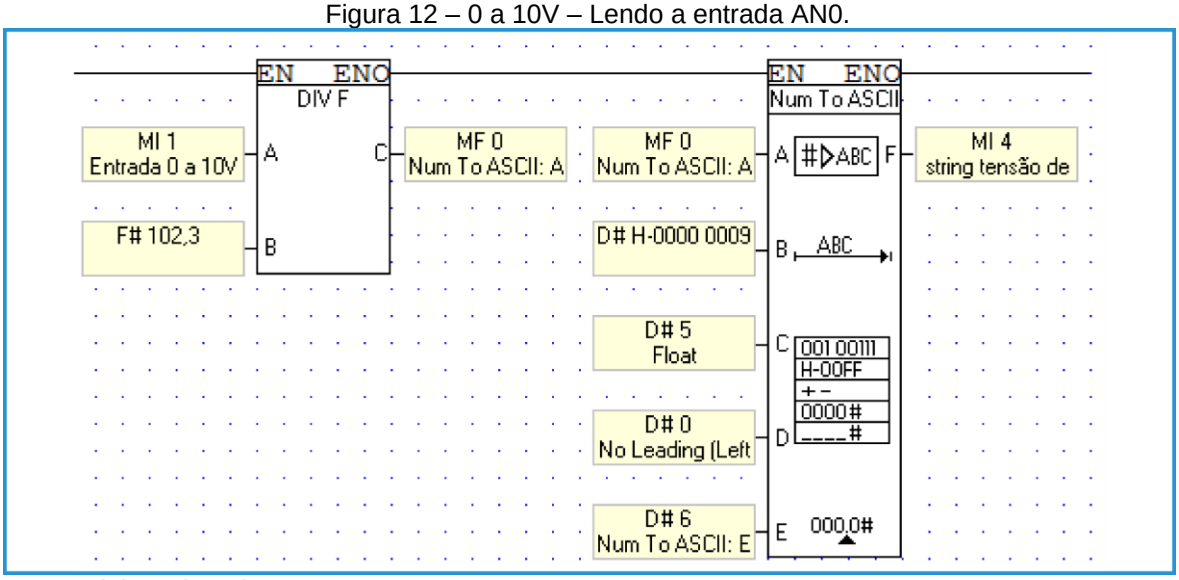
O primeiro bloco da figura 12 serve para dividir o valor de MI1 por 102,3, gerando um número *float* que corresponde ao valor de 0 a 10V da entrada. O resultado da divisão é armazenado na variável MF0.

Devido às características do CLP, o valor de MF0, que indica a tensão de entrada, precisa ser convertido para *string* a fim de ser apresentado na IHM.

A conversão de número *float* para *string* é realizada pelo segundo bloco da figura 12.

O resultado da conversão de MF0 de *float* para *string* é armazenado em uma *string* com 9 *bytes* de comprimento. Essa *string* começa na variável MI4, ocupando 5 posições de memória (MI4 até MI8).

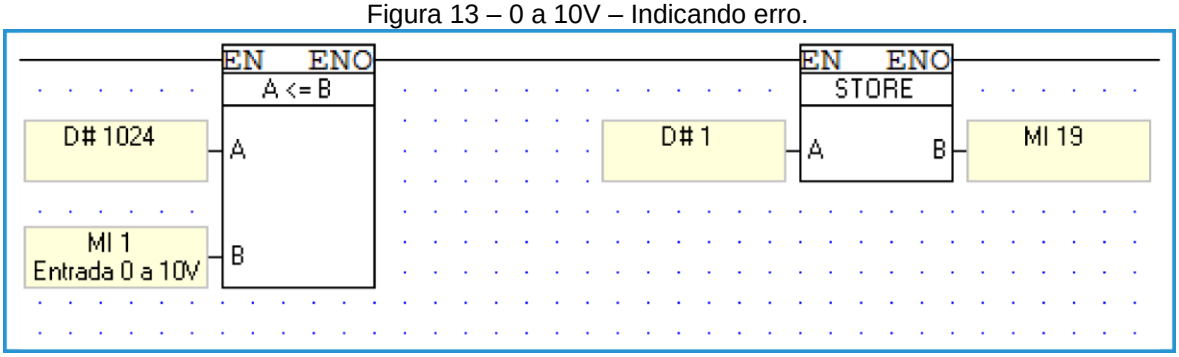
Figura 12 – 0 a 10V – Lendo a entrada AN0.



Fonte: Elaborado pelo autor.

Os blocos comparadores das figuras 13 e 14 trabalham em conjunto. Quando um está ativo, o outro está inativo e vice versa. Caso MI1 seja igual ou maior que 1024, o bloco comparador da figura 13 indica a condição de *overflow* ativando o bloco "*Store direct*" da figura 13, armazenando o valor '1' na variável MI19. Ao mesmo tempo MB11 é desligada, tornando visível a moldura vermelha.

Figura 13 – 0 a 10V – Indicando erro.

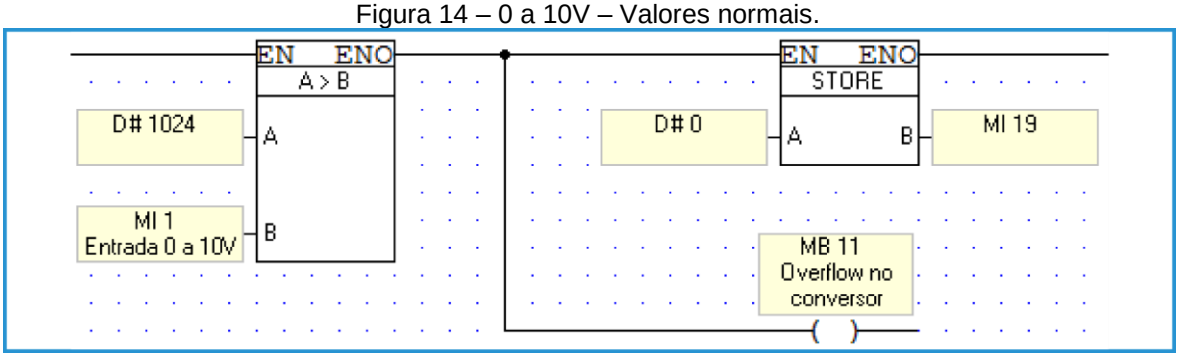


Fonte: Elaborado pelo autor.

Caso o bloco comparador da figura 14 esteja ativo, significa que o sistema está operando dentro da escala. Nessa condição o bloco "*Store direct*" armazena o valor '0' em MI19 desativando a mensagem de erro na IHM.

Ao mesmo tempo MB11 é ativado, inibindo a moldura vermelha em torno da *string* que exibe o valor da tensão aplicada no conversor AD.

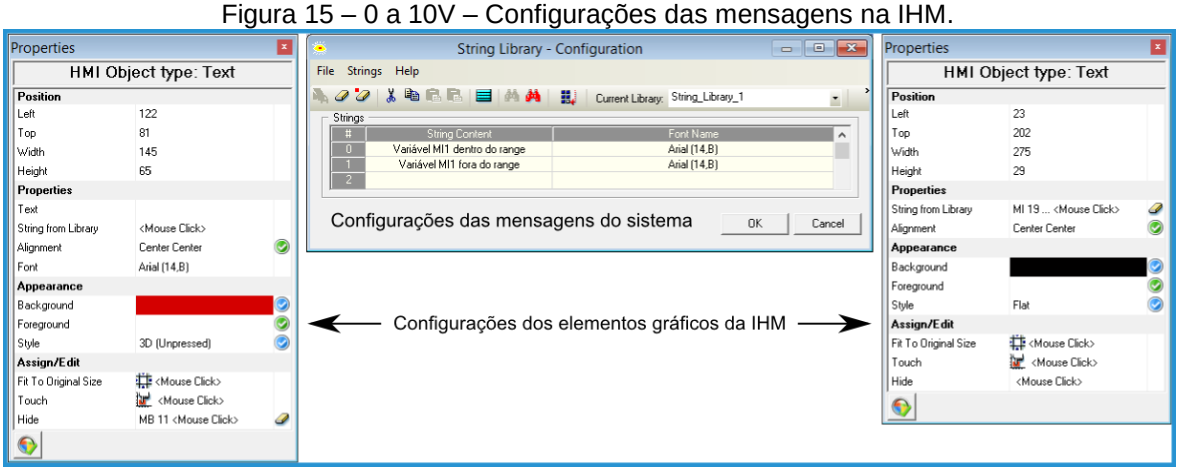
Figura 14 – 0 a 10V – Valores normais.



Fonte: Elaborado pelo autor.

Na IHM e de acordo com a figura 16, foram adicionados os seguintes elementos ativos: barra de LEDs associada ao valor da variável MI1, mostrador indicando o valor da *string* MI4, mensagens de aviso dependentes da variável MI19, e moldura vermelha cuja exibição depende do valor de MB11.

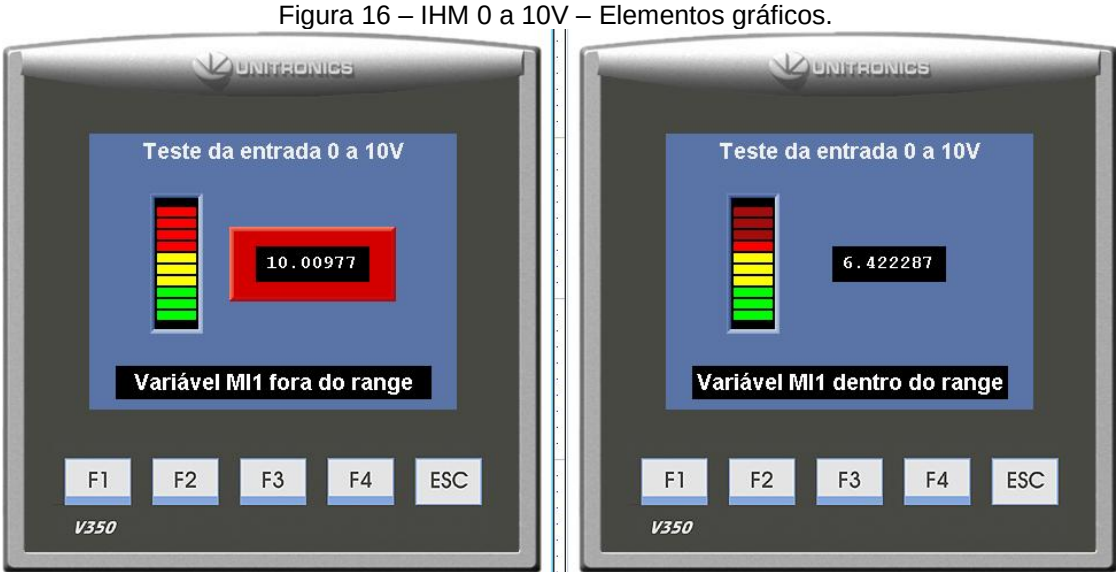
Figura 15 – 0 a 10V – Configurações das mensagens na IHM.



Fonte: Elaborado pelo autor.

Na figura 15 são exibidas a configurações dos elementos gráficos da IHM e as mensagens de texto a serem exibidas de acordo com o valor da variável MI19.

Figura 16 – IHM 0 a 10V – Elementos gráficos.



Fonte: Elaborado pelo autor.

A figura 17 mostra a configuração da barra de LEDs, que utiliza os valores de MI1 diretamente, sem qualquer conversão. Também é exibida a configuração do mostrador numérico, atrelado à *string* MI4 que tem 9 *bytes* de comprimento.

Figura 17 – 0 a 10V – Barra de LED e mostrador numérico.

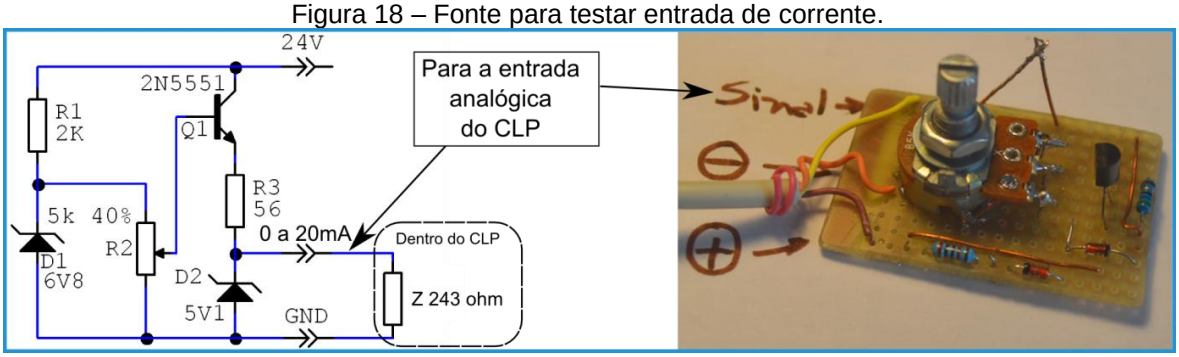
Fonte: Elaborado pelo autor.

3.2.9 **Testando a entrada analógica 0-20mA / 4-20mA**

Para testar a entrada de corrente do CLP, de acordo com a figura 18, foi construída uma fonte de tensão convencional sem compensação de temperatura (CIPELLI; SANDRINI, 1980), de 0 a 6,3V com circuito de proteção R3 – zener D2.

Essa proteção zener serve para garantir que o sinal injetado na entrada de corrente não extrapole os valores especificados no quadro 3, o que poderia vir a danificar a entrada de corrente do CLP. A fonte construída consegue injetar de 0 a 21mA, assegurando que toda a escala de corrente do conversor AD do CLP seja percorrida, não sendo objetivo do teste analisar a precisão do conversor AD.

Figura 18 – Fonte para testar entrada de corrente.

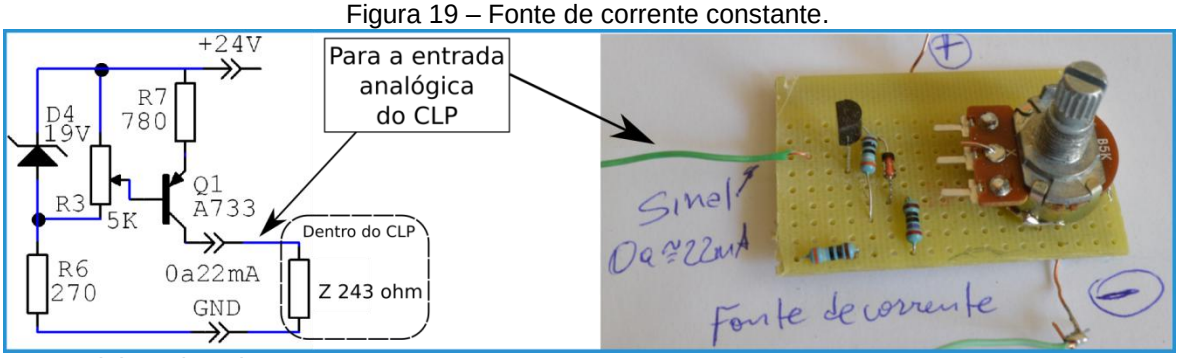


Fonte: Elaborado pelo autor.

Por outro lado, Bryan e Bryan (1997, p. 268) resaltam que é melhor utilizar loop de corrente para transmissão de sinal, dado que a resistência da fiação não tem efeito sobre um *loop* de corrente.

Atendendo a Bryan e Bryan (1997), é apresentada na figura 19 uma fonte de corrente constante para testar a entrada do CLP.

Figura 19 – Fonte de corrente constante.



Fonte: Elaborado pelo autor.

A fonte é baseada na descrição de Cipelli e Sandrini (1980, p. 259) cujo projeto foi convertido para utilizar transistor PNP e adicionou-se um potenciômetro R3 para ajuste da corrente de saída.

O zener D4 garante que a tensão sobre a malha “R7 – Vbe de Q1” seja constante garantindo a estabilidade da corrente na malha “R7 – Q1 – Carga”, em torno de 22mA.

A corrente de saída máxima é cerca de 22mA@5,1V. Caso haja necessidade de uma tensão de saída maior na fonte de corrente, pode-se reduzir o valor de D4 para 15V por exemplo, procedendo com redimensionamento de R6, R7 e Q1 o que elevaria a tensão de saída máxima para cerca de 8V.

Após a configuração do *hardware* e do *software* VisiLogic conforme especificado no quadro 4 e figuras 9 e 10, foi elaborado o programa a seguir para acompanhar o comportamento da entrada de corrente (AN1).

3.2.10 **Programa para testar a entrada de corrente AN1**

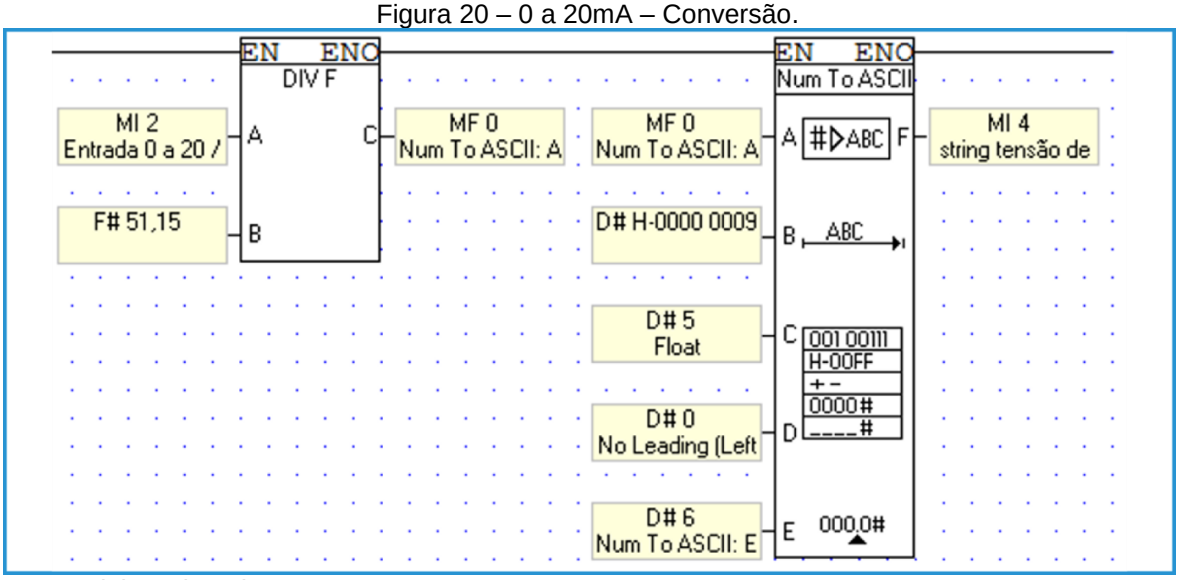
Trata-se do mesmo algoritmo empregado no programa anterior, sendo que será abordado apenas as diferenças entre os mesmos.

A entrada analógica AN1 tem seu valor convertido para digital. O valor obtido é armazenado na variável MI2, conforme definido na configuração de *hardware* (figura 10). A variável MI2 contém um valor inteiro e seu valor pode variar de 0 a 1023 de acordo com a corrente de entrada, indicando o valor 1024 em caso de *overflow*.

O primeiro bloco da figura 20 serve para dividir o valor de MI2 por 51,15, gerando um número *float* que corresponde ao valor de 0 a 20mA injetado na entrada analógica. Ainda na figura 20, o resultado da divisão é armazenado na variável *float* MF0.

O bloco "NumToASCII” pega o valor de MF0 e o converte para *string*. A *string* obtida pela conversão de MF0 é armazenada em MI4 que é uma *string* de nove posições ocupando os endereços de memória MI4 a MI8.

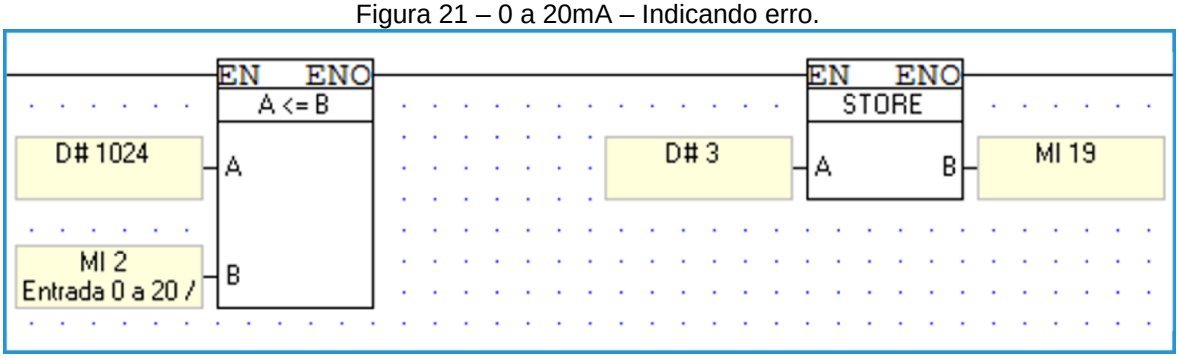
Figura 20 – 0 a 20mA – Conversão.



Fonte: Elaborado pelo autor.

Os blocos comparadores das figuras 21 e 22 trabalham em conjunto. Quando um está ativo, o outro está inativo e vice versa, da mesma forma que foi abordado anteriormente. A diferença agora é que a variável MI2 está sendo monitorada, que corresponde à entrada de corrente do CLP.

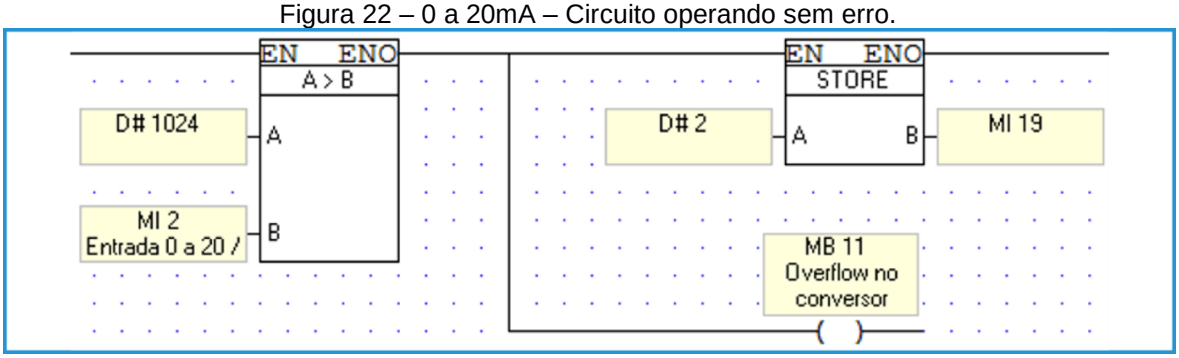
Figura 21 – 0 a 20mA – Indicando erro.



Fonte: Elaborado pelo autor.

Como já informado anteriormente, MB11 presente na figura 22 serve para controlar a exibição da moldura vermelha na IHM e MI19 é utilizado pelas caixas de texto para selecionarem qual mensagem será apresentada na IHM.

Figura 22 – 0 a 20mA – Circuito operando sem erro.



Fonte: Elaborado pelo autor.

3.3 **Testando a saída do CLP**

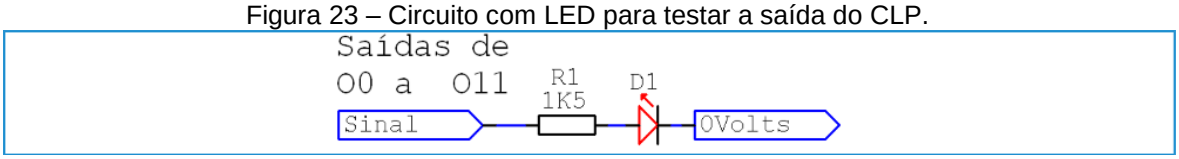
O CLP em estudo possui 12 saídas PNP a transistores empregando a tecnologia P-MOSFET, sendo que as saídas “O0”, “O1”, “O2”, “O3”, “O4”, “O5” e “O6” são de alta velocidade conseguindo operar até a frequência de 500Hz, com largura de pulso de até 2 milissegundos. Esse tipo de saída pode ser empregada para comandar controladores PWM (controladores por modulação de largura de pulso), permitindo o acionamento de inversores de frequência para controle de motores AC, fazer o acionamento proporcional de motores de corrente contínua e controlar cargas resistivas proporcionalmente, entre outras aplicações. De acordo com o quadro 5, seguem as especificações dos sinais de saída do CLP escolhido e figura 23 apresenta um diagrama de ligação de um LED na saída do CLP que pode ser utilizado para testar o comportamento da saída do mesmo em condições para acionamentos a baixa frequência.

Quadro 5 – Especificação das saídas do CLP em estudo.

Saídas do CLP	Especificação
Número de saídas	12 (PNP)
Tipo de saída	P-MOSFET (dreno aberto)
Isolação	Nenhuma
Corrente máxima por saída (carga resistiva)	0,5A por saída
Corrente máxima por saídas ligadas simultaneamente (carga resistiva)	3A
Frequência máxima (carga resistiva)	50Hz
Frequência máxima (carga indutiva)	0,5Hz
Máxima frequência do PWM	0,5kHz
Proteção de curto circuito	Sim
Indicação de curto circuito	Via *software*
Queda de tensão máxima qudo ligado	0,5Vdc no máximo
Tensão de operação da fonte	20,4 a 28,8Vdc
Tensão nominal da fonte	24Vdc
Saídas de 0 a 6 podem ser utilizadas como saídas para PWM	

Fonte: Adaptado de Unitronics (2015c).

Figura 23 – Circuito com LED para testar a saída do CLP.



Fonte: Elaborado pelo autor.

3.4 Outras funcionalidades do CLP em estudo

O CLP possui interfaces serial e *Ethernet* para carga de programas e monitoração remota, bem como suporte a SD *card*, sendo que informações adicionais podem ser obtidas consultando a documentação do produto.

3.5 Integrando o CLP ao sistema computacional

Neste tópico serão apresentadas duas possibilidades de se integrar um CLP a um sistema de manufatura, seguindo o conceito relatado por O'Brien (2001).

Na primeira abordagem é apresentada a aplicação “TCP *server*”, um sistema no qual o CLP irá persistir na base de dados do servidor uma palavra contendo o estado de seus registros, conforme diagrama da figura 24. Os dados persistidos na

base de dados ficam disponíveis para uso pelos sistemas de controle de manufatura e de gerenciamento de banco de dados (SGBD).

Figura 24 – Aplicação TCP *server*.

Fonte: Elaborado pelo autor.

Na segunda abordagem e de acordo com a figura 25, apresenta-se a aplicação “TCP *client*”. Trata-se de um sistema no qual a aplicação no servidor irá enviar comandos para a porta de serviço 20256 do CLP e o mesmo irá responder ao comando recebido, havendo a possibilidade de se construir um conjunto de comandos e os colocar para rodar em sequência. Nesta abordagem será implementado o protocolo de comunicação da Unitronics no formato ASCII em uma aplicação Java, conforme especificado no manual do fabricante do produto (UNITRONICS, 2004).

Figura 25 – Aplicação TCP *client.*

Fonte: Elaborado pelo autor.

3.6 **Aplicação TCP *server***

Conforme já abordado na figura 24, a aplicação TCP *server* proposta deve fornecer meios para que o CLP persista dados em uma base de dados hospedada em um servidor.

3.6.1 **Escopo do sistema e aplicação TCP *server* proposta**

- A aplicação no CLP irá montar e enviar ciclicamente uma palavra de estado ao servidor pela porta *Ethernet* e não fará qualquer verificação se a gravação ocorreu com sucesso, sem utilizar conectores OPC no servidor.
- O CLP será configurado utilizando as ferramentas gratuitas do fabricante.
- A aplicação do lado do servidor deve persistir no banco de dados a palavra que chegar pela porta 20001.

Não está previsto redundância, retransmissão de dados e nem autenticação do CLP.

- Os dados serão persistidos em um banco de dados MySQL em um servidor Ubuntu com Java 1.8, não sendo a portabilidade entre plataformas o foco deste trabalho.
- Mostrar um pacote de dados sendo enviado do CLP ao servidor e a resposta do comando.
- O objetivo é montar um sistema mínimo que permita que um CLP grave dados em um servidor, ficando em aberto o refinamento do sistema.

3.6.2 **Lado CLP – enviando dados ciclicamente**

Para o CLP enviar dados ciclicamente, concebeu-se que o mesmo deve mandar uma palavra de estado a cada 6 segundos conforme configuração do módulo *"Drum"*. A palavra de estado deve conter as seguintes informações: Data, Hora, 12 bits de saída e 12 bits de entrada, utilizando como separador o caractere '|' e o caractere 0x0D (CR) para indicar o fim de linha. Abaixo segue o formato da palavra de estado que será enviada do CLP para a aplicação TCP *server* no servidor:

DD/MM/YY|HH:MM:SS|O0|O1|O2|O3|O4|O5|O6|O7|O8|O9|O10|O11|<CR>

A palavra de estado deve ser encapsulada em um frame TCP para poder ser enviada pela porta *Ethernet* ao servidor, cuja explicação será apresentada junto com o diagrama *Ladder*.

3.6.3 **Lado CLP – *Ladder* e a aplicação TCP *server***

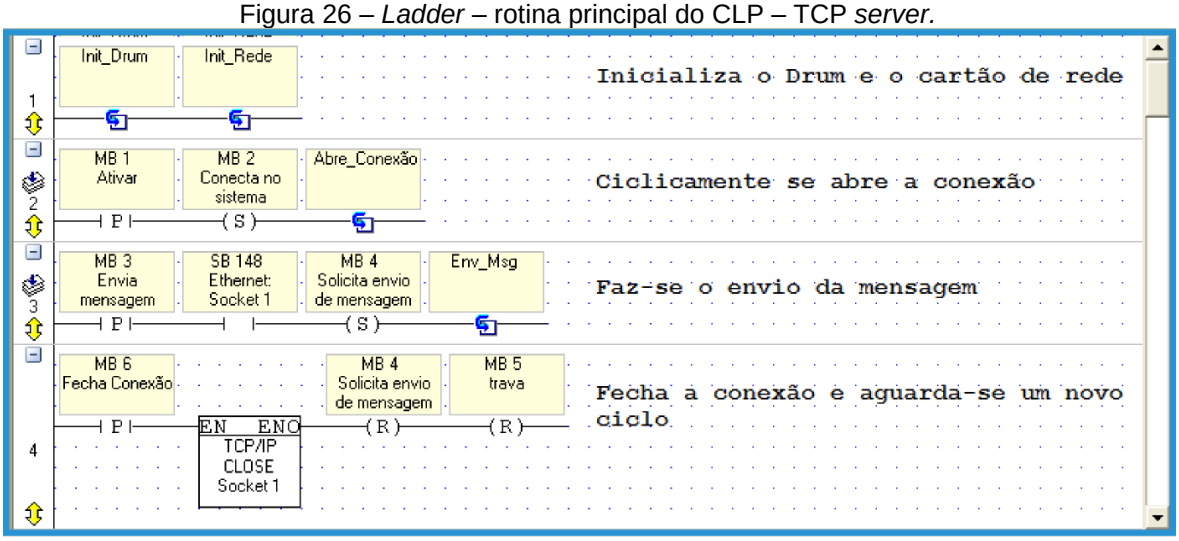
Conforme apresentado em linhas gerais na figura 26, a aplicação do CLP inicializa o *"Drum"* e o cartão de rede chamando as respectivas rotinas.

Após a inicialização de ambos, inicia-se o ciclo de envio da palavra de estado do CLP ao servidor TCP.

Durante o processo de envio de mensagens, a aplicação abre uma conexão TCP (se conecta ao servidor), e somente envia a palavra de estado após assegurar que a conexão tenha sido aberta com sucesso (SB148).

Uma vez enviada a palavra de estado, o bloco "TCP/IP CLOSE" faz o fechamento da conexão TCP, ajustando o sistema para um novo ciclo.

Figura 26 – *Ladder* – rotina principal do CLP – TCP *server.*



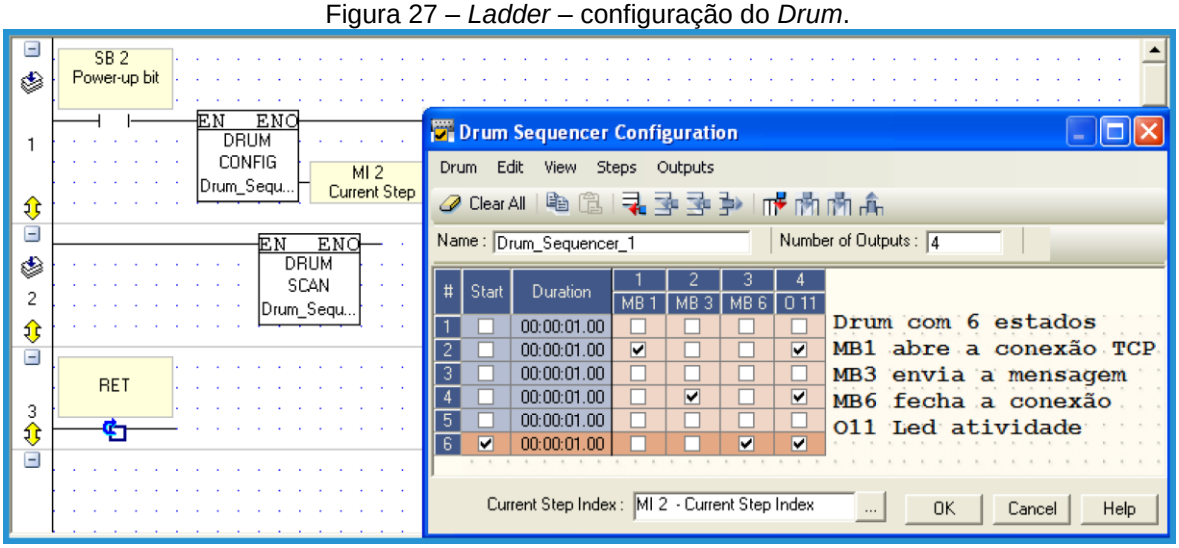
Fonte: Elaborado pelo autor.

Na figura 27, é exibida a rotina e o detalhamento da configuração do bloco *"Drum"*.

À medida que o bloco *"Drum"* gira, o sistema abre a conexão TCP pelo acionamento de (MB1), envia a palavra de status ao acionar (MB3) e o acionamento de (MB6) desencadeia o fechamento da conexão TCP.

O sinal (O11) serve para acionar um led que fica piscando, indicando que o *"Drum"* está ativo.

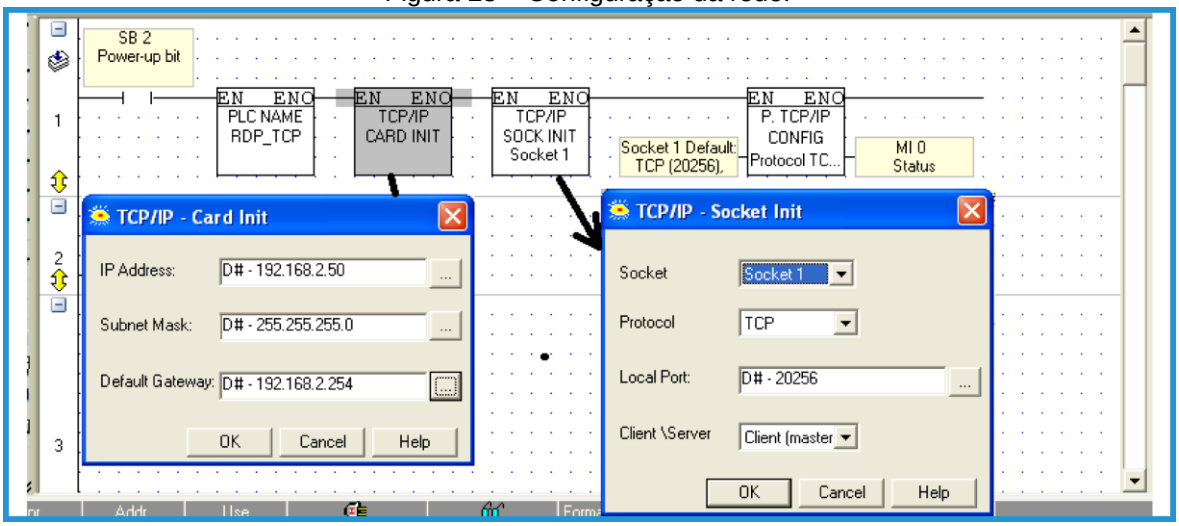
Figura 27 – *Ladder* – configuração do *Drum*.



Fonte: Elaborado pelo autor.

A figura 28 apresenta a configuração dos blocos de inicialização de rede, sendo que o bloco "TCP/IP CONFIG" é autoexplicativo.

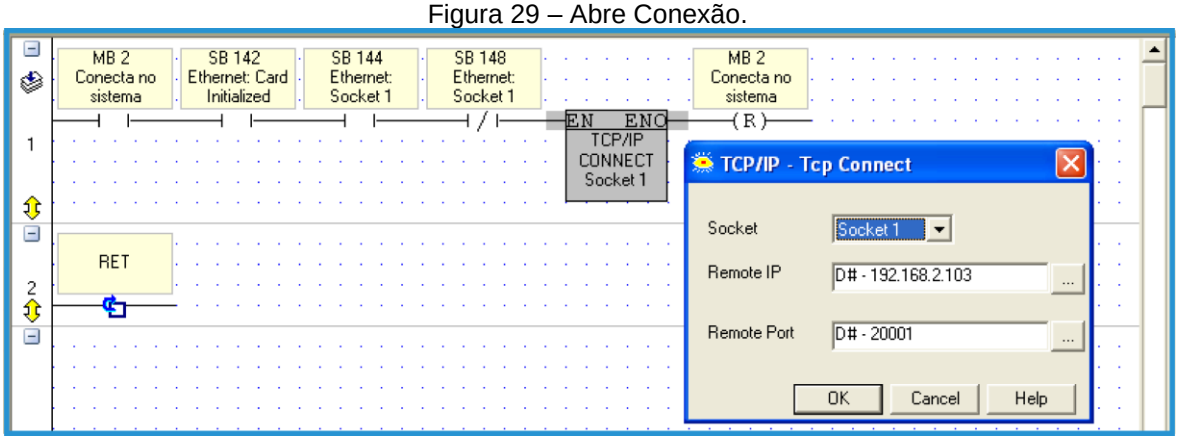
Figura 28 – Configuração da rede.



Fonte: Elaborado pelo autor.

Segundo a figura 29, o bloco "TCP/IP CONNECT" é configurado para direcionar os pacotes de dados para a porta 20001 do servidor destino, cujo endereço de IP é 192.168.2.103.

Figura 29 – Abre Conexão.



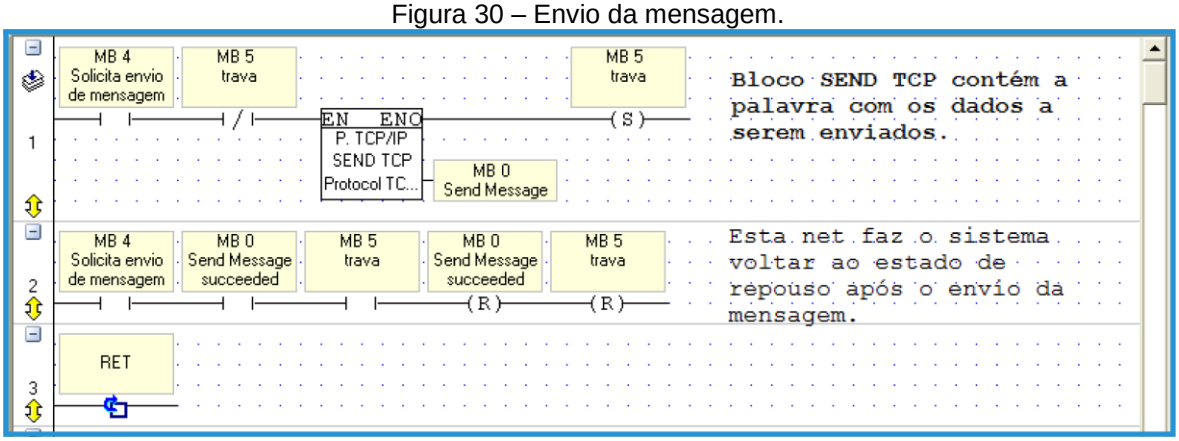
Fonte: Elaborado pelo autor.

Na figura 30, é apresentada a rotina para envio da mensagem de status. O circuito dessa rotina garante o envio da mensagem e a volta do sistema ao estado de repouso, ficando a cargo de (MB6) sinalizar o bloco "TCP/IP CLOSE" para que a conexão seja fechada.

Uma vez em repouso, o sistema ficará aguardando o início de um novo ciclo que ocorrerá quando (MB1) for acionado pelo "Drum".

O bloco “SEND TCP” deve ser configurado para mandar a mensagem no formato especificado no item 3.6.2.

Figura 30 – Envio da mensagem.



Fonte: Elaborado pelo autor.

Detalhes específicos de como configurar o bloco "SEND TCP" podem ser obtidos consultando o manual do fabricante.

3.6.4 **Lado do servidor - recebendo dados do CLP**

O projeto Maven – Java deste tópico está disponível para download no website Sourceforge.net, link “https://sourceforge.net/projects/plc-unitronics/”.

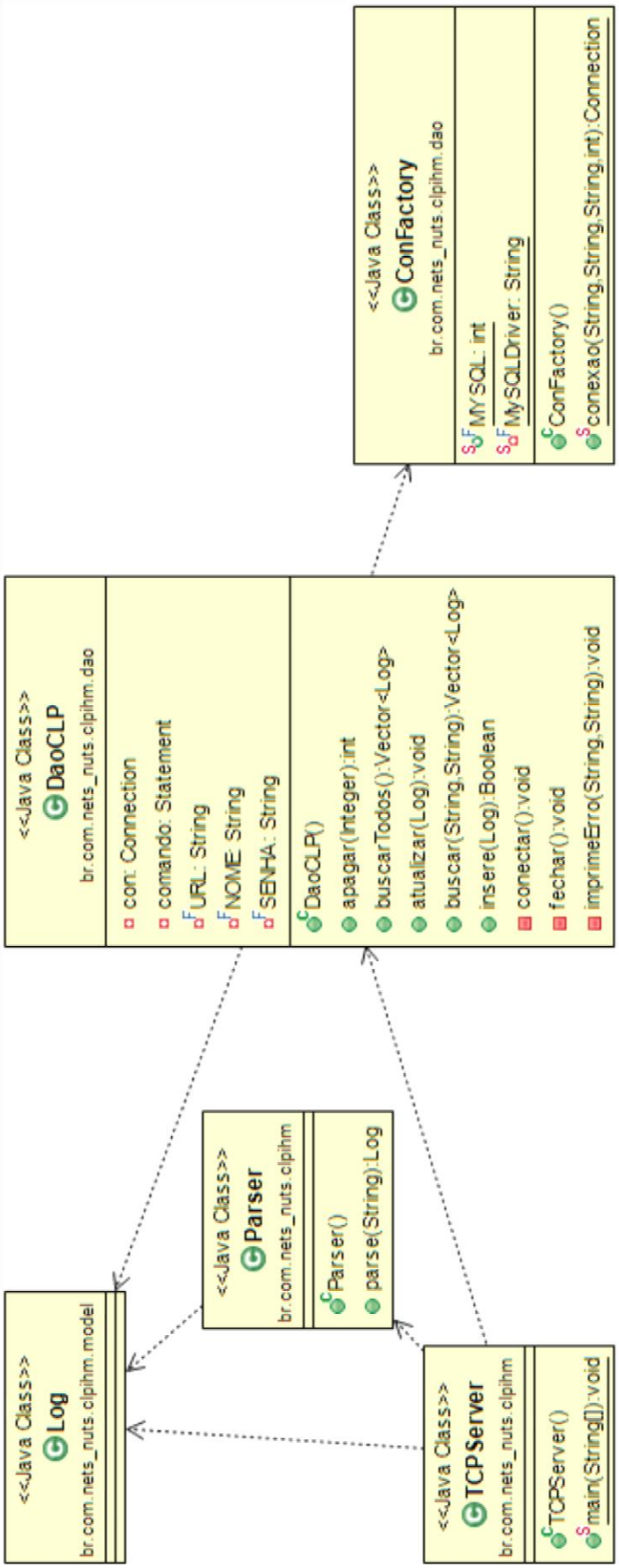
O Javadoc da aplicação TCP *server* pode ser consultado em:

“http://nets-nuts.com.br/documents/clpihm_Server_Javadoc/”

Conforme apresentado na figura 31, no lado do servidor, para receber os pacotes enviados pelo CLP, foi montada uma aplicação Java com as classes descritas a seguir.

- TCPServer: Classe responsável por escutar a porta 20001. A classe TCP Server aciona a classe Parser para analisar o pacote de dados recebido e a classe DaoCLP para fazer a inserção de dados no banco de dados.
- Parser: Classe responsável pela adequação dos dados recebidos do CLP aos requisitos do banco de dados.
- Log: Os métodos dessa classe são ferramentas para manipular a linha de log do banco de dados, tanto para persistir como para recuperar dados.
- DaoCLP: Classe responsável por fazer acesso aos objetos de dados. Nessa classe são montados os comandos SQL e instanciada a conexão ao banco de dados.
- ConFactory: Classe responsável por prover a conexão com o banco de dados.

Figura 31 – clpihm_Server – Diagrama de classes.



Fonte: Elaborado pelo autor. (alguns métodos e variáveis foram suprimidas para clareza do diagrama)

3.7 **Aplicação TCP *client***

Conforme já abordado na figura 25, a aplicação TCP *client* proposta deve iniciar a comunicação acessando o CLP, enviando um comando. O CLP por sua vez responde ao comando, que é recebido pela aplicação, que irá fazer o processamento do mesmo de acordo com a regra da aplicação.

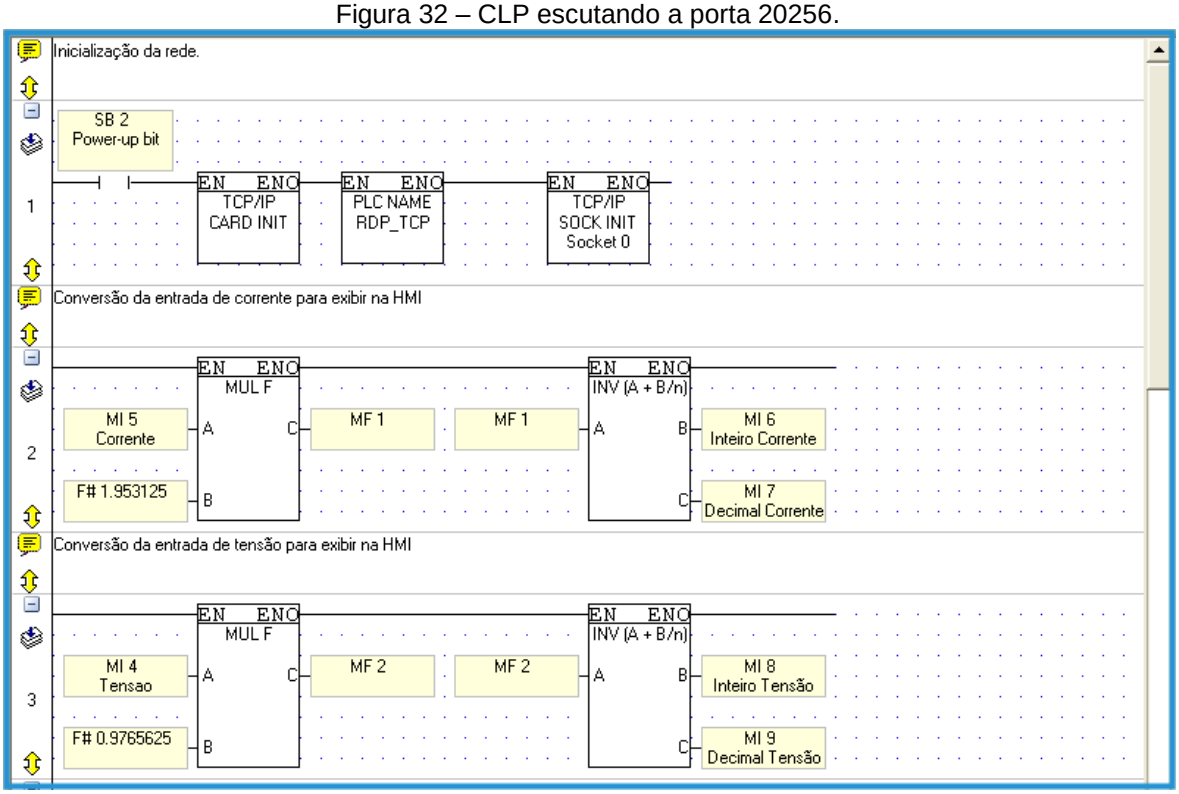
3.7.1 **Escopo do sistema e aplicação TCP *client* proposta**

- CLP irá responder aos comandos enviados de acordo com a especificação do protocolo ASCII do equipamento, Unitronics (2004).
- CLP será configurado para habilitar o funcionamento da interface de manutenção pela porta *Ethernet*
- O computador utiliza sistema operacional Linux Ubuntu, com Java 1.8, Eclipse com os *plugins* Maven e WindowBuilder, não sendo a portabilidade entre plataformas o foco desse trabalho.
- No lado do computador, pela aplicação Java desenvolvida deve ser possível construir os comandos de consulta e escrita, o cálculo do *checksum* e o envio dos comandos ao CLP em estudo, respeitando as especificações do fabricante.
- Os comandos e as respostas dos comandos enviados devem ser armazenados em uma caixa de texto (histórico de comandos), sendo possível a manipulação dos mesmos.
- Uma segunda aplicação incorporada à tela principal deve permitir o sequenciamento do histórico de comandos enviados a um intervalo de tempo configurável.
- Não há armazenamento em banco de dados.
- Não pode utilizar OPC.
- O CLP será configurado utilizando apenas as ferramentas gratuitas do fabricante. Não é foco aferir a precisão do conversor AD.
- A interface da aplicação deve fazer uma validação básica dos comandos.

3.7.2 **Lado CLP – configurações gerais**

No lado do CLP é necessário configurar a interface *Ethernet* para aceitar o protocolo de serviço. Essa configuração é feita inicializando-se o cartão *Ethernet* com seu endereço de IP, definindo o nome do CLP e configurando o soquete ‘0’ como cliente e abrindo a porta 20256, conforme ilustrado na figura 32.

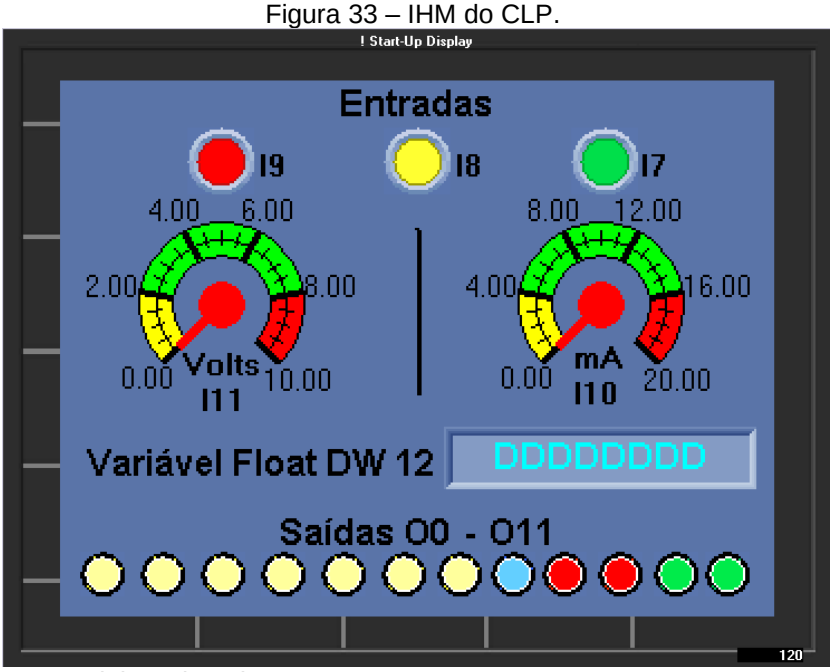
Figura 32 – CLP escutando a porta 20256.



Fonte: Elaborado pelo autor.

As *nets* 2 e 3 servem para formatar os valores lidos nas entradas analógicas do CLP de acordo com o mostrador na IHM exibido na figura 33, lembrando que a IHM está configurada para exibir as entradas digitais I7, I8 e I9, as entradas analógicas de tensão e corrente, possui acesso à variável *float* DW12 e exibe o status das 12 saídas (O0 à O11).

Figura 33 – IHM do CLP.



Fonte: Elaborado pelo autor.

3.7.3 **Lado do servidor TCP *client***

O projeto Maven – Java deste tópico está disponível para download no site sourceforge.net, link “https://sourceforge.net/projects/plc-unitronics/”

O Javadoc da aplicação TCP *client* pode ser consultado em:

“http://nets-nuts.com.br/documents/clpihm_Client_Javadoc/”

A aplicação Java desenvolvida no servidor, é composta por 6 classes apresentadas na figura 35 que são descritas a seguir.

- TelaComando – Esta classe serve para renderizar a interface do usuário (GUI) conforme figura 36. Ela possui uma série de validações e campos configuráveis para para assistir na montagem da palavra de comando a ser enviada ao CLP. É responsável por armazenar o histórico dos comandos enviados e das respostas recebidas do CLP e consegue exibir os testes automáticos de conexão como o de ping e de *timeout*, realizados pela classe TCPClient.
- TCPClient – Cria o soquete de conexão de acordo com as configurações de IP e porta e envia os comandos para o CLP. Esta classe também possui o método "testaPing(String host)" para realização do teste de conectividade.

- CalcCheckSum – Faz o cálculo do *checksum* do comando do CLP. Para o cálculo do *checksum* são considerados os caracteres do comando propriamente dito, desconsiderando o caractere de início de comando ‘/’, que é feito pelo cálculo do módulo base 256 da soma dos caracteres ASCII da palavra de comando conforme exemplos de escrita e leitura de bits apresentados na figura 34.

  Maiores detalhes sobre o processo para cálculo do *checksum* podem ser obtidos consultando a documentação do fabricante ou o código da classe CalcCheckSum, no apêndice A, ou verificando o código no Gitlab.

Figura 34 – Cálculo do *checksum*.

Cálculo do check-sum

/		STX	Não entra no cálculo do check sum
0		48	Endereço do CLP
2		50	
S		83	Código do comando
A		65	
		0	
0		48	Endereço inicial
0		48	
0		48	
7		55	
0		48	Tamanho em bytes do argumento (hexa - 8 bits)
5		53	
0		48	Argumentos a enviar
1		49	
0		48	
1		49	
0		48	
1	Check	788	Soma dos valores
4	sum	20	Cáculo do módulo
		14	Conversão para ASCII

/02SA0007050101014

Gravando no CLP

/		STX	Não entra no cálculo do check sum
0		48	Endereço do CLP
2		50	
R		82	Código do comando
A		65	
		0	
0		48	Endereço inicial
0		48	
0		48	
7		55	
0		48	Tamanho em bytes do argumento (hexa - 8 bits)
5		53	
		0	Argumentos a enviar: não há
		0	
		0	
		0	
		0	
2	Check	545	Soma dos valores
1	sum	33	Cáculo do módulo
		21	Conversão para ASCII

/02RA00070521

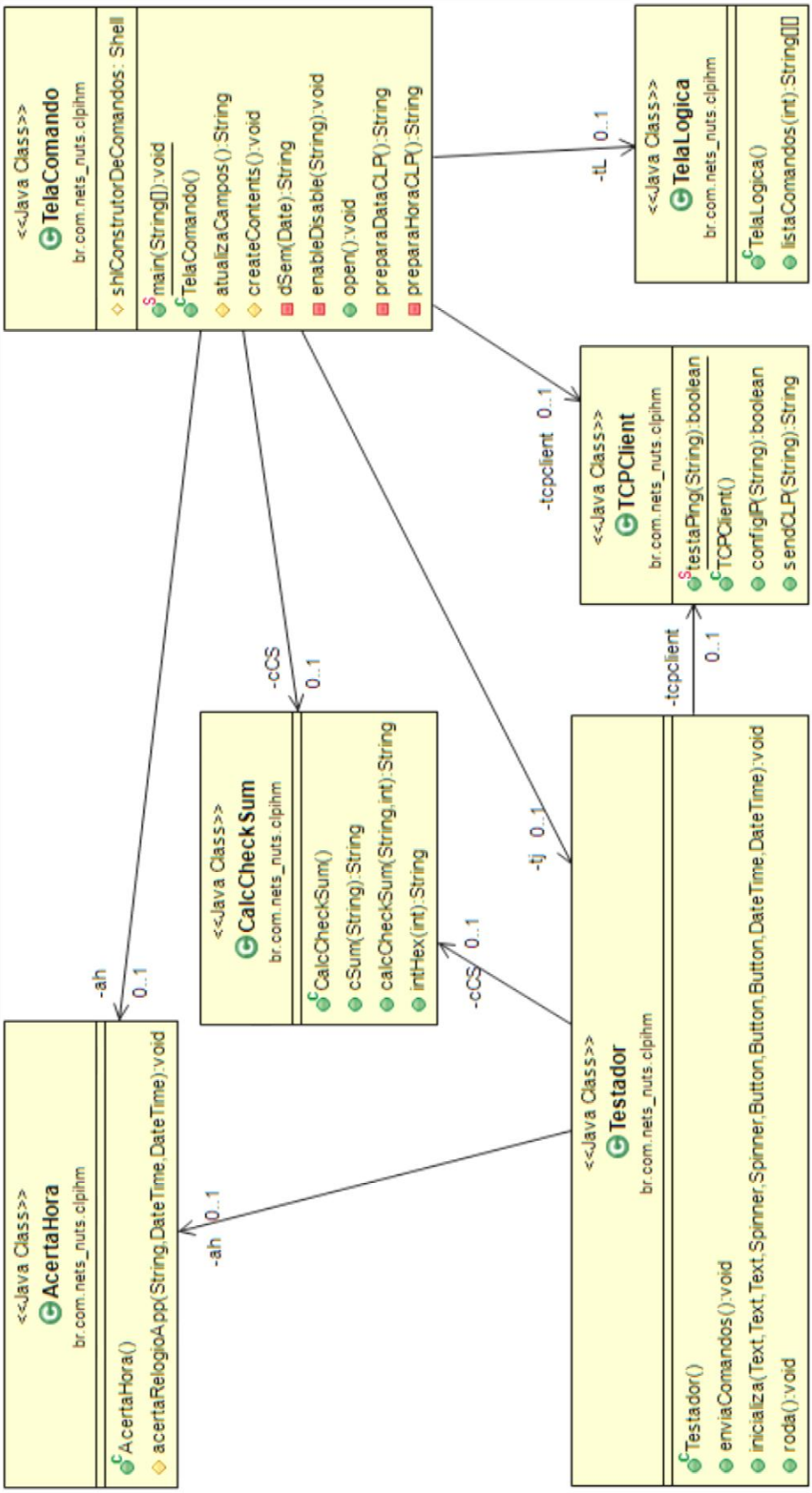
Lendo dados do CLP

Fonte: Elaborado pelo autor.

- AcertaHora – Classe para manipular os objetos de data e hora da aplicação. Responsável tanto para exibir consultas de data e hora como para auxiliar na montagem dos comandos de escrita do relógio.
- TelaLogica – Faz a leitura do arquivo "comandos.csv" que contém a descrição dos comandos disponíveis, bem como parâmetros utilizados na aplicação. Esse arquivo permite a adição de novos parâmetros.

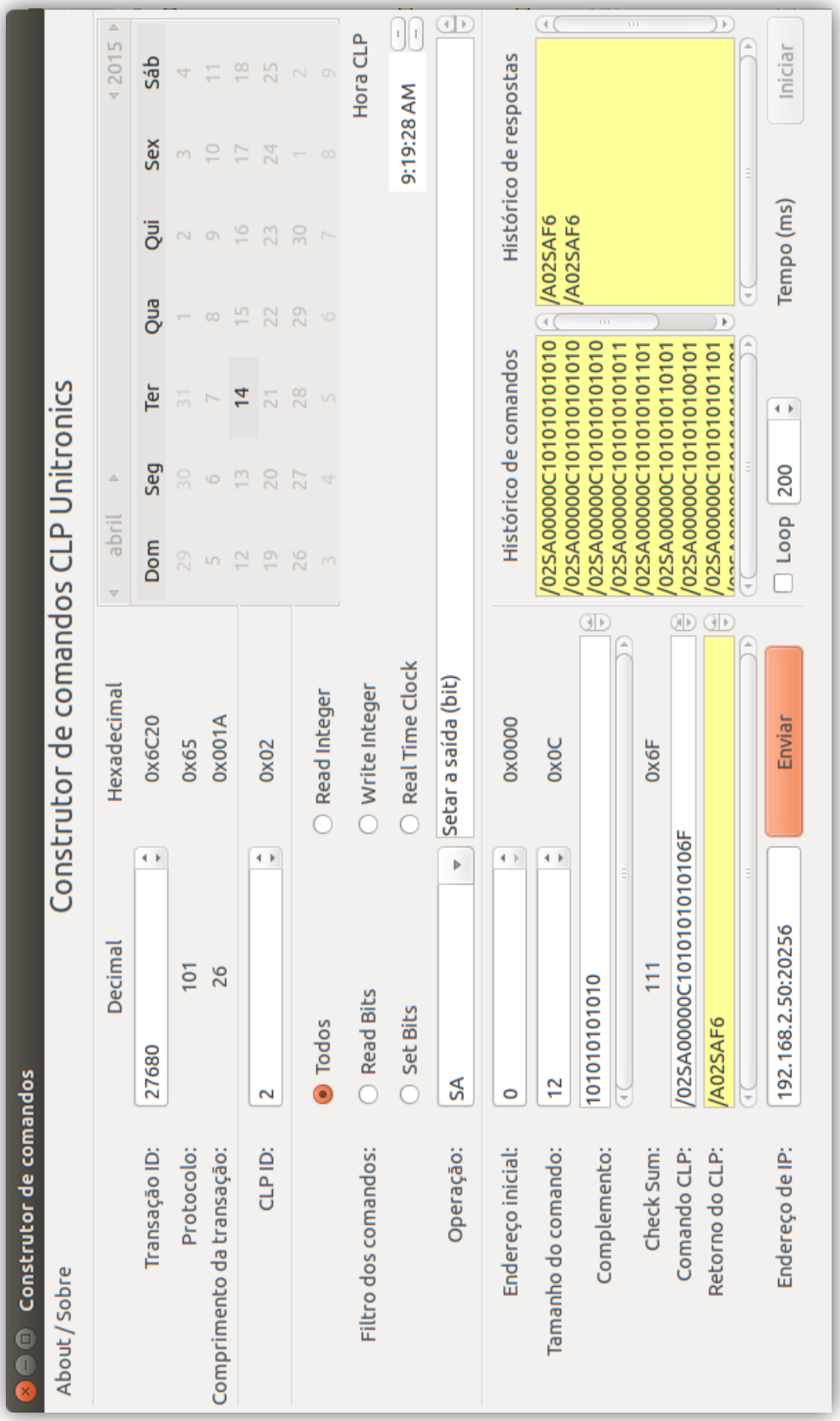
- Testador – Esta classe mostra um exempo do instanciamento das classes AcertaHora, CalcCheckSum e TCPClient, permitindo que a classe Testador consiga enviar comandos ao CLP sem necessariamente utilizar o módulo OPC. Esta classe Testador pode representar uma aplicação exemplo concebida para executar uma rotina cíclica de testes no CLP, utilizando como base o histórico de comandos previamente enviados ao CLP ao se utilizar a classe TelaComando. Como exemplo de aplicação, pode-se utilizar a classe Testador para testar maquinários, auxiliar na análise e diagnóstico de defeitos intermitentes, pode-se empregar na realização de testes de fadiga ou mesmo integrar o exemplo apresentado a qualquer outra solução que demande que um programa, Java no caso, envie comandos cíclicos a um CLP.

Figura 35 – clpihm_Client – Diagrama de classes.



Fonte: Elaborado pelo autor. (alguns métodos e variáveis foram suprimidas para clareza do diagrama)

Figura 36 – TelaComando.



Fonte: Elaborado pelo autor

4 CONCLUSÃO

Nesse estudo, na parte introdutória, é dada uma visão geral do CLP, define-se sua finalidade bem como a finalidade da IHM. Apresenta-se o CLP escolhido e sua plataforma de desenvolvimento, descreve-se algumas configurações como a configuração PNP - NPN das entradas digitais e a configuração das entradas analógicas, incluindo a configuração da plataforma de desenvolvimento.

Na apresentação das entradas e saídas do CLP, são descritos circuitos que podem ser utilizados como ferramentas de testes e diagnóstico das entradas analógicas "0 a 10V" e de "0-4 a 20mA". Embora os circuitos atendam o objetivo da monografia, por serem circuitos simples, eles carecem de mecanismos de proteção contra curto e de estabilização de temperatura, ficando para estudo posterior o projeto de circuitos de teste que incluam proteção contra curto, com tolerância à variação de temperatura e porque não, automatizados, programáveis e integrados à sistemas de diagnósticos.

Após falar sobre o funcionamento das entradas e saídas do CLP e métodos de testes, aborda-se duas técnicas para disponibilizar os dados gerados por um CLP a um sistema computacional, sendo que as ferramentas de testes apresentadas serviram de apoio ao estudo.

Ao pesquisar meios para interligar o CLP a um sistema computacional, via de regra os fabricantes ou direcionam à soluções proprietárias ou direcionam a sites que oferecem soluções de conectividade utilizando o módulo OPC *server* (Windows).

Neste ponto, sem a pretensão de responder afirmativa e definitivamente à pergunta "tem para Linux?", negada recorrentemente na área de automação, foram documentadas duas formas de conexão CLP <-> computador utilizando Java em ambiente Linux - Ubuntu, sem utilizar OPC server ou qualquer componente Windows, exceto as ferramentas necessárias para a configuração do CLP.

Na primeira configuração, chamada "TCP *server*" o CLP envia os dados e eles são persistidos numa base de dados conforme demonstrado e na segunda configuração chamada "TCP *client*" o computador é quem solicita que o CLP revele os dados de seus registros.

Por limitação do produto, em ambos casos apresentados, a comunicação foi realizada via interface *Ethernet*, não havendo suporte à criptografia dos dados (mesmo utilizando OPC - Windows). Embora não tenha sido objeto de estudo, há

indícios que a segurança dos dados de rede trafegados de e para o CLP estudado e provavelmente de outros fabricantes deve ser definida em outras camadas de segurança, devendo-se levar em consideração variáveis como roteamento de sinal, políticas de acesso físico e outras, sem descartar a possibilidade de se fazer um processo de análise de risco, ficando o tema "segurança de dados" em aberto para estudo futuro.

A plataforma de desenvolvimento utilizada foi o Eclipse com Java 1.8, gerenciador de dependências Maven e *plugin* Window Builder. Foram utilizadas bibliotecas SWT para montagem da interface do usuário, que são dependentes de plataforma. Caso haja necessidade de portar a aplicação para outro sistema operacional pode ser necessário configurar o arquivo "pom.xml" do projeto para utilizar as bibliotecas SWT adequadas e resolver demais dependências.

Não foram feitos testes de integração com o "SCADA BR" sendo que o objetivo deste estudo é apenas acessar dados no CLP e prover meios para o CLP persistir dados em um servidor. Dessa forma fica em aberto um estudo da viabilidade da integração do programa apresentado com aplicações como o "SCADA BR" (http://www.scadabr.com.br/) ou mesmo a integração da solução apresentada com ferramentas BI como a Pentaho (http://www.pentaho.com/).

Outro ponto para desenvolvimento futuro, considerando a variedade não só de CLPs, mas também de dispositivos microprocessados multifunções de baixo custo e com alto poder de processamento, comparado aos CLPs de 5 anos atrás, seria a modelagem de um framework, suportando IPV6, capaz de acolher bibliotecas como as desenvolvidas, específicas a cada tipo de hardware, permitindo o mapeamento dos comandos a tags, para desenvolvimento de sistemas supervisórios de código aberto, habilitando o monitoramento dos mais variados tipos de dispositivos, independentemente de fabricante.

REFERÊNCIAS

ALVES, José Luiz Loureiro. **Instrumentação, controle e automação de processos.** 2. ed. Rio de Janeiro: Ltc, 2013. 201 p.

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

CIPELLI, Antonio Marco Vicari; SANDRINI, Waldir João. **Teoria e desenvolvimento de projetos de circuitos eletrônicos.** 4. ed. São Paulo: Érica, 1980.

INTERNET PINBALL MACHINE DATABASE. **Bow and Arrow.** Disponível em: <http://www.ipdb.org/showpic.pl?id=362&picno=41377&zoom=2>. Acesso em: 3 jan. 2015.

MERRIAM-WEBSTER. **Hardwired.** Disponível em: <http://www.merriam-webster.com/dictionary/hardwired>. Acesso em: 22 dez. 2014.

NOF, Shimon Y. (Ed.). **Springer Handbook of Automation.** West Lafayette In 47907: Springer, 2009. 1888 p.

O'BRIEN, James A. **Sistemas de Informação:** e as Decisões Gerenciais na era da Internet. 9. ed. São Paulo: Saraiva, 2001. 436 p.

OPC FOUNDATION. **OPC Foundation Membership Application** - Email Verification Required. [mensagem pessoal] Mensagem recebida por: <renato.pierri@anhanguera.com> em 15 maio 2015.

PIERRO, Maria Clara di; JOIA, Orlando; RIBEIRO, Vera Masagão. Visões da educação de jovens e adultos no Brasil. **Cad. Cedes,** Campinas, v. 21, n. 55, p.58-77, nov. 2001. FapUNIFESP (SciELO). DOI: 10.1590/s0101-32622001000300005. Disponível em: <http://dx.doi.org/10.1590/S0101-32622001000300005>. Acesso em: 18 dez. 2014.

QUEZADA-QUEZADA, José Carlos et al. Diseño e implementación de un sistema de control y monitoreo basado en HMI-PLC para un pozo de agua potable. **Ingeniería: Investigación y Tecnología,** México, v. 15, n. 1, p.41-50, jan. 2014. Disponível em: <http://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S1405-77432014000100005&lang=pt>. Acesso em: 3 jan. 2015.

SIEMENS. **Step 2000:** Basics of PLCs. Disponível em: < http://www.nfiautomation.org/FREE_Download/Technical%20Documents/PLC/Step200BasicsofPLCsSiemens.pdf>. Acesso em: 22 dez. 2014.

UNITRONICS. **Communication with the Vision PLC.** 2004. Disponível em: <http://www.unitronics.com/Downloads/Support/Technical Library/Vision Software Miscellaneous/Miscellaneous/Communication with Unitronics PLCs.pdf>. Acesso em: 5 jan. 2015.

UNITRONICS. **V1040.** Disponível em: <http://www.unitronics.com/plc-hmi/plc-vision-enhanced/v1040->. Acesso em: 4 jan. 2015a.

UNITRONICS. **Vision OPLC V350-35-T2:** Installation Guide. Disponível em: <http://www.unitronics.com/Downloads/Support/Documents/Unitronics_Library.zip>. Acesso em: 4 jan. 2015b.

UNITRONICS. **Vision OPLC V350-35-T2/V350-J-T2:** Technical Specifications. Disponível em: <http://www.unitronics.com/Downloads/Support/Documents/Unitronics_Library.zip>. Acesso em: 4 jan. 2015c.

WEG. **PLC300.** Disponível em: <http://www.weg.net/br/Produtos-e-Servicos/Drives/CLPs-e-Controle-de-Processos/PLC300>. Acesso em: 3 jan. 2015.

APENDICE A - clpihm_Client - Classe CalcCheckSum.java

```
package br.com.nets_nuts.clpihm;

/**
 * <pre>
 * Esta classe faz o cálculo do check sum do comando do CLP,
 * e adiciona os caracteres de controle necessários.
 * Para cálculo do check sum somente são contados os caracteres
 * do comando propriamente dito, sem o caractere "/".
 * O cálculo do checksum e feito pelo cálculo do módulo da soma
 * dos caracteres ASCII da palavra de comando.
 *
 * Descrição do cabeçalho
 *
 * |0x07|0xLow|0xHigh|0x65|0x0|0xLow|0xHigh|0x2F|
 *    |     |     |      |         |     |    |
 *    |     |     |      |         |     |    +---- Começo do comando (caractere '/')
 *    |     |     |      |         |     +--------- Tamanho do comando - byte alto
 *    |     |     |      |         +--------------- Tamanho do comando - byte baixo
 *    |     |     |      +------------------------- Modo ASCII 0x65, modo binário 0x66
 *    |     |     +-------------------------------- Nr transação - byte alto
 *    |     +-------------------------------------- Nr transação - byte baixo
 *    +-------------------------------------------- Não documentado. Bug a ser resolvido
 * </pre>
 *
 * @author Renato de Pierri - renato.pierri@gmail.com
 * */
public class CalcCheckSum {

	private int rawCmdSize, valor, modulo, soma;
	private String comando_CLP, checksum, tamanhoCmd, tamLowByte,
tamHiByte,
			trID, trLowByte, trHiByte;
	private String fim = new String(new byte[] { 0x0D });

	/** Construtor da classe */
	public CalcCheckSum() {
		// TODO Auto-generated constructor stub
	}

	/*****************************************************************/

	/**
	 * <p>
	 * Este método calcula o checksum e constroi a palavra de comando.
	 * calcCheckSum recebe o comando e o nr da transação para cálculo do
	 * check sum e montagem do cabeçalho.
	 * </p>
	 *
	 * @param rawCmd
	 *            String contendo o parâmetro a ser enviado para o CLP
	 * @param transaID
	 *            Inteiro contendo o número da transação
	 * @return String com o comando completo a ser enviado ao CLP
	 * */
	public String calcCheckSum(String rawCmd, int transaID) {

		soma = 0;
		rawCmdSize = rawCmd.length();
		for (int i = 0; i < rawCmdSize; i++) {
			valor = rawCmd.charAt(i);
			soma = soma + valor;
		}
		modulo = soma % 256;
```

```
            checksum = cSum(rawCmd);

            /**
             * Fim do cálculo do checksum.
             */

            /**
             * Definindo o tamanho da palavra de comando, em hexadecimal (2
bytes)
             * */
            rawCmdSize = rawCmdSize + 4;
            tamanhoCmd = (Integer.toHexString(rawCmdSize)).toUpperCase();
            while (tamanhoCmd.length() != 4) {
                  tamanhoCmd = "0" + tamanhoCmd;

            }
            tamLowByte = "0x" + tamanhoCmd.substring(2, 4);
            tamHiByte = "0x" + tamanhoCmd.substring(0, 2);
            /**
             * Fim da definição do tamanho da palavra de comando.
             * */

            /**
             * Convertendo o número da transação de decimal para
hexadecimal (2
             * bytes).
             * */

            trID = intHex(transaID);
            trLowByte = "0x" + trID.substring(2, 4);
            trHiByte = "0x" + trID.substring(0, 2);
            /**
             * Fim da conversão do número de transação de decimal para
hexadecimal.
             * */

            /**
             * Montando o cabeçalho do comando do CLP
             * */
            String cabecalho = new String(new byte[] { 0x07,
                        Integer.decode(trLowByte).byteValue(),
                        Integer.decode(trHiByte).byteValue(), 0x65, 0x00,
                        Integer.decode(tamLowByte).byteValue(),
                        Integer.decode(tamHiByte).byteValue() });

            /**
             * Concatenando os elementos do comando do CLP:
             * <p>
             * 1- cabeçalho, 2- o início do comando, 3- o comando "/", 4- o
check
             * sum, 5- a finalização <CR>.
             * </p>
             **/

            comando_CLP = cabecalho + "/" + rawCmd + checksum + fim;
            return comando_CLP;
      }

      /*************************************************************/

      /**
```

```
	 * Converte o numero da transação de decimal para hexadecimal e
preenche a
	 * direita com zeros.
	 *
	 * @param transaID
	 *            Inteiro com o identificador da transação .
	 * @return String contendo o valor em hexa.
	 * */
	public String intHex(int transaID) {

		String trID = (Integer.toHexString(transaID)).toUpperCase();
		while (trID.length() != 4) {
			trID = "0" + trID;
		}
		return trID;
	}

	/*****************************************************************/

	/**
	 * Calcula o checksum. cSum recebe o comando e faz o cálculo do
checksum.
	 *
	 * @param rawCmd
	 *            String contendo o parâmetro a ser calculado.
	 * @return - String contendo o valor do checksum (hexa).
	 * */
	public String cSum(String rawCmd) {
		int soma = 0;
		rawCmdSize = rawCmd.length();
		for (int i = 0; i < rawCmdSize; i++) {
			valor = rawCmd.charAt(i);
			soma = soma + valor;
		}
		modulo = soma % 256;
		checksum = (Integer.toHexString(modulo)).toUpperCase();
		if (checksum.length() == 1) {
			checksum = "0" + checksum;
		}
		return checksum;
	}

	/*****************************************************************/
}
```

APENDICE B - clpihm_Client - Classe Testador.java

```
/**
 *
 */
package br.com.nets_nuts.clpihm;

import org.apache.commons.lang3.StringUtils;
import org.eclipse.swt.widgets.*;

/**
 * @author renato
 *
 */
public class Testador extends Thread {
	private AcertaHora ah = new AcertaHora();
	private TCPClient tcpclient = new TCPClient();
	private CalcCheckSum cCS = new CalcCheckSum();
	private String comando, result;
	private Text txtHistCmd, txtHistResult, endIP;
	private Spinner transacaoId, tempoMs;
	private Button btnRodarComandos, btnIniciar, btnEnviar;
	private Boolean loop;
	private int sleepThread;
	private DateTime calendario, dateTimeCLP;

	/*****************************************************************/

	/**
	 * Construtor padrão da classe.
	 */
	public Testador() {
		// TODO Auto-generated constructor stub
	}

	/*****************************************************************/

	/**
	 * Inicializa parâmetros do testador.
	 *
	 * @param tHC
	 *            Objeto Texto que contém a lista de comandos a serem
enviados
	 *            ao CLP
	 * @param tHR
	 *            Objeto Texto que receberá o retorno dos comandos
enviados ao
	 *            CLP
	 * @param IP
	 *            Objeto Text contendo o IP e porta do CLP
	 * @param trId
	 *            Objeto Spinner contendo o número da transação a ser
enviada ao
	 *            CLP
	 * @param tMs
	 *            Objeto Spinner contendo a temporização do envio de
comandos ao
	 *            CLP
	 * @param bR
	 *            Objeto Button estilo checkbox que controla o loop e
habilita o
```

```
	 *            envio de comandos ao CLP
	 * @param bI
	 *            Objeto Button que inicia o loop de envio de comandos ao
CLP
	 * @param bE
	 *            Objeto Button do construtor de comandos que envia
comandos ao
	 *            CLP
	 * @param cal
	 *            Objeto DateTime estilo calendário
	 * @param dCLP
	 *            Objeto DateTime estilo relógio
	 * */
	public void inicializa(Text tHC, Text tHR, Text IP, Spinner trId,
			Spinner tMs, Button bR, Button bI, Button bE, DateTime
cal,
			DateTime dCLP) {
		txtHistCmd = tHC;
		txtHistResult = tHR;
		endIP = IP;
		transacaoId = trId;
		tempoMs = tMs;
		btnRodarComandos = bR;
		btnIniciar = bI;
		btnEnviar = bE;
		loop = btnRodarComandos.getSelection();
		sleepThread = tempoMs.getSelection();
		calendario = cal;
		dateTimeCLP = dCLP;
	}

	/*************************************************************/

	/**
	 * Inicia a thread que envia os comandos ao CLP
	 * */
	public void roda() {
		new Thread(new Runnable() {
			public void run() {
				while (loop) {
					try {
						Thread.sleep(1000);
					} catch (Exception e) {
					}
					Display.getDefault().asyncExec(new Runnable()
{
						public void run() {
							enviaComandos();
						}
					});
				}
			}
		}).start();
	}

	/*************************************************************/

	/**
	 * Envia comandos ao CLP. Este método extrai cada item do histórico
de
	 * comandos, calcula o checksum, monta o comando e envia ao CLP.
```

```
	 * */
	public void enviaComandos() {
		loop = btnRodarComandos.getSelection();
		sleepThread = tempoMs.getSelection();
		txtHistResult.setText("");
		tcpclient.configIP(endIP.getText());
		int inicio, fim;
		String cmdPuro;
		inicio = 0;
		fim = 1;
		/**
		 * Passa todo o texto para maiúsculo, extrai os comandos e
calcula o
		 * número de comandos
		 */

	txtHistCmd.setText(StringUtils.upperCase(txtHistCmd.getText()));
		String v = txtHistCmd.getText();
		String values[] = StringUtils.split(v, "\n");
		int tamArray = values.length;
		int i = 0;
		if (!btnRodarComandos.getSelection()) {
			i = tamArray;
		}
		/** A thread só existe por conta desse while */
		while (i < tamArray) {
			if (!btnRodarComandos.getSelection()) {
				i = tamArray - 1;
			}
			if (i == 0) {
				fim = fim + values[i].length();
			} else {
				fim = fim + values[i].length() + 1;
			}
			/**
			 * Aqui eu seleciono o item, só para ficar bonito na tela
do
			 * usuário. Não faço nada de útil com isso
			 */
			txtHistCmd.setSelection(inicio, fim);
			cmdPuro = StringUtils.remove(values[i], "/");
			Display.getCurrent().getActiveShell().redraw();
			Display.getCurrent().getActiveShell().update();
			Display.getCurrent().getActiveShell().layout();
			/** Montando o comando */
			if (cmdPuro.length() > 3) {
				comando = cCS.calcCheckSum(cmdPuro,

	Integer.parseInt(transacaoId.getText()));
				/** Enviando para o TCP client e pegando o retorno
*/
				try {
					result = tcpclient.sendCLP(comando);

				} catch (Exception e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}
				/**
				 * Se for comando de relógio, faz-se a atualização
do calendário
```

```
                        * e relógio.
                        */
                       if (StringUtils.contains(result, "RC")) {

      ah.acertaRelogioApp(StringUtils.substring(result, 6,
                                          result.length() - 2), calendario,
dateTimeCLP);

      Display.getCurrent().getActiveShell().redraw();

      Display.getCurrent().getActiveShell().update();

      Display.getCurrent().getActiveShell().layout();
                       }

                       if (StringUtils.equals(txtHistResult.getText(),
"")) {
                             txtHistResult.setText(result);
                       } else {
                             txtHistResult.setText(txtHistResult.getText()
+ "\n"
                                          + result);
                       }
                       result = "";
                 }
                 inicio = fim;
                 try {
                       Thread.sleep(Long.parseLong(tempoMs.getText()), 0);
                 } catch (NumberFormatException e) {
                       // TODO Auto-generated catch block
                       e.printStackTrace();
                 } catch (InterruptedException e) {
                       // TODO Auto-generated catch block
                       e.printStackTrace();
                 }
                 i++;
           }
           System.out.println("linhas no widget" + tamArray);
      }
}
```

ANEXO A - Especificação do protocolo de comunicação ASCII

UNITRONICS

Headquarters
Unitronics (1989) (R"G) Ltd.
Unitronics Building, Airport City
P.O.B. 300, Ben Gurion Airport, Israel 70100
Tel: + 972 (3) 977 8888 Fax: + 972 (3) 977 8877

Communication with the Vision™ PLC

Document Version: 20/9/04

Contents

Introduction

This document is intended to enable developers to create applications that communicate data between Unitronics' PLCs and other applications.

The Vision™ series offers two formats of data exchange between the PLC and the calling application:

ASCII format,

Binary format.

M90/91 controllers support ASCII format only. Binary format is not supported.

While usage of the ASCII format enables the user to exchange homogeneous data types with the PLC, the binary format enables him to either read or write heterogeneous data types from / to it, within a single request. Hence, if the application requires that different type values be included in a single request (such as MB and DW), a binary format must be used.

This document describes the format of the data requests that the calling application can send to the PLC and the PLC message response format.

Communicating with the Vision™ PLC

An application can communicate with the PLC either by serial communication or by TCP. To communicate via TCP (Ethernet), you must add a header to each command as described in Appendix 3: PCOM via Ethernet.

The format of the data exchange is independent of the communication protocol; in either case ASCII and binary formats may be implemented.

Notes

1. Either ASCII or Binary message format may be used; there is no connection between the two types.
2. PLCs of types M90, M91 and Vision 120 do not support the TCP communication protocol.
3. Vision 230, 260, and 280 controllers must have an installed Ethernet port to implement TCP.

ASCII Message Format

ASCII message format enables values to be both read from and written to the PLC.

In this format:

Frames sent to the PLC should start with a slash character '/' (ASCII 47) and should be terminated by <CR> (ASCII 13)

Frames received from the PLC start with '/A' (slash and the letter A) and are terminated by <CR>.

Note In this document, the combined '/A', is referred to as STX1.

The format of a command sent to the PLC is: <STX><UnitID><CommandCode><Command parameters><CheckSum><ETX>

STX="/"

The UnitID should be two characters long (pad with zeroes if needed).

Note Note that UnitID ='00' refers to a directly connected PLC.

The checksum (CRC) should be two characters long (pad with zeroes if needed). See Appendix 1 regarding checksum calculation.

Command parameters are command-dependent.

ETX = CR (Carriage Return, ASCII value: 13).

The values to and from the PLC are represented in hexadecimal, (aka 'Hex as ASCII'), and thus enable an easy debug process. For example, the PLC will respond with 06D9 when queried for a Memory Integer register containing a decimal value of 1753.

Note In the examples below:
CRC represents the check sum value.
Spaces have been added just for clarity. A real space is indicated by <SPACE>.

RE, RA, RB, GS, RT, RM - Read bits

To PLC:

<STX><CC><ADDRESS><LENGTH><CRC><ETX>

From PLC:

<STX1><CC><VALUES><CRC><ETX>

CC (Command Code) values:

"RE" - Read inputs

"RA" - Read outputs

"RB" - Read memory bits

"GS" - Read system bits

"RT" - Read timer scan bits

"RM" - Read counter scan bits

Example (read 5 memory bits from address 32 to 36):

PC: / 01 RB 0020 05 CRC <CR>

PLC: / A 01 RB 10010 CRC <CR>

01 means PLC with UnitID=01

0020 means start Read operation at address 32 (0020 hex)

05 means read 5 values, beginning from the start address.

10010 means bits 32 (20 hex) to 36 (24 hex)

are 1, 0,0,1,0 respectively.

SA, SB, SS - Set bits

To PLC:

<STX><CC><ADDRESS><LENGTH><CRC><ETX>

From PLC:

<STX1><CC><VALUES><CRC><ETX>

CC (Command Code) values:

"SA" - Set outputs

"SB" - Set memory bits

"SS" - Set system bits

Example (set 5 memory bits from address 32 to 36):

PC: / 01 SB 0020 05 10010 CRC <CR>

PLC: / A 01 SB CRC <CR>

01 means PLC with UnitID=01

0020 means start at address 32 (0020 hex).

05 means length of 5 memory bits.

10010 means set bits 32 (20 hex) to 36 (24 hex).

are set to 1,0,0,1,0 respectively.

RW, RNL/D/H/J/F, GF, GT, GP,GX, GY - Read integers (16 and 32 bits)

To PLC:

<STX><CC><ADDRESS><LENGTH><CRC><ETX>

From PLC:

<STX1><CC><VALUES><CRC><ETX>

Note When Command Codes are longer than 2 characters, only the first two characters will be included in the response from the PLC.

CC (Command Code) values:

"RW" - Read MIs

"RNL" - Read MLs (*)

"RND" - Read MDWs (*)

"GF" - Read SIs

"RNH" - Read SLs (*)

"RNJ" - Read SDWs (*)

"RNF" – Read MFs (memory floats) (See note about floats) (*)

"GT" - Read Timer's current value

"GP" - Read Timer's preset value

"GX" - Read Counter's current value

"GY" - Read Counter's preset value

* The Command Code from PLC will be "RN"

Example (read 2 memory integers from address 32 to 33):

PC: / 01 RW 0020 02 CRC <CR>

PLC: / A 01 RW 01020304 CRC <CR>

01 means PLC with UnitID=01

0020 means start address of 32 (0020 hex)

02 means length of 2

01020304 means:

Value in addresses 32 (20 hex) is 0102 (hex).

Value in address 33 (21 hex) is 0304

Example (read 2 memory longs from address 32 to 33):

PC: / 01 RNL 0020 02 CRC <CR>

PLC: / A 01 RN 1122334405060708 CRC <CR>

01 means PLC with UnitID=01

0020 means starting address is 32 (0020 hex)

02 means a read vector length of 2

1122334405060708 means:

Value in address 32 (20 hex) is 11223344 (hex).

Value in address 33 (21 hex) is 05060708 (hex)

Example (read a float value from MF (15)

PC: / 01 RNF 000F 01 CRC <CR>

PLC: /A 01 RN E6AE4640 CRC <CR>

01 means PLC with UnitID=01

000F means starting address is 15 (000F hex)

RNF is the command code.

01 means a read vector length of 1

E6AE4640 means:

The return value

The return value represents the float value of 12345.67

Note The bits are represented (according to IEEE 754 standard) 4640E6AE hence, in order to derive the actual value of the words, they should be transposed. (E6AE 4640 → 4640 E6AE).

SW, SNL/D/H/J, SF - Write integers (16 and 32 bits)

To PLC:

<STX><CC><ADDRESS><LENGTH><CRC><ETX>

From PLC:

<STX1><CC><VALUES><CRC><ETX>

Note When Command Codes are longer than 2 characters, only the first two characters will be included in the response from the PLC..

CC (Command Code) values:

"SW" - Write MIs

"SNL" - Write MLs (*)

"SND" - Write MDWs (*)

"SNF" – write MFs (*)

"SF" - Write SIs

"SNH" - Write SLs (*)

"SNJ" - Write SDWs (*)

* Command Code from PLC will be "SN"

Example (set values of two memory integers) :

PC: / 01 SW 0020 02 01020304 CRC <CR> // 16 bits

PLC: / A 01 SW <CR>

01 means PLC with UnitID=01

0020 means first register address is 32 (20hex).

02 means the write vector length is 2.

01020304 means that:

value 258 (0102 hex) will be written into address 32 (20 hex).

value 772 (0304 hex) will be written into addresses 33 (21 hex)

Example (set values of two memory longs):

PC: / 01 SNL 0020 02 1122334405060708 CRC <CR> // 32 bits

PLC: / A 01 SN <CR>

01 means PLC with UnitID=01

0020 means the start address is 32 (20hex).

02 means write into two registers.

1122334405060708 means:

value of 287454020 (11223344 hex) will be written to address 32 (20 hex)

value of 84281096 (5060708 hex) will be written to address 33 (21 hex)

Note: Data for each register should be indicated explicitly.

RC - Read RTC

To PLC:

<STX><CC<CRC><ETX>

From PLC:

<STX1><CC><VALUE><CRC><ETX>

CC (Command Code) values:

"RC" – Read Real Time Clock

Example Read Real Time Clock

PC: / 01 RC CRC <CR>

PLC: / A 01 RC 00 30 15 01 29 04 01 CRC <CR>

The time is: 15:30:00, Sunday (01), 29/04/2001

SC - Set RTC

To PLC:

<STX><CC<CRC><ETX>

From PLC:

<STX1><CC><VALUE><CRC><ETX>

CC (Command Code) values:

" SC " – Set Real Time Clock

Example: Set Real Time Clock

PC: / 01 SC 00 30 15 01 29 04 01 CRC <CR>

PLC: / A 01 SC <CR>

Set time to 15:30:00, Sunday (01), 29/04/2001

Appendix 2: Checksum calculation

A. Generating a checksum for an ASCII request

The checksum is generated by accumulating the ASCII values of the entire message and calculating the 256 modulo value

For example, the message "/01RE002005" will be appended with a CRC value, in which the checksum is calculated **without** the STX (/).

48+49+82+69+48+48+50+48+48+53=543

543 mod 256=31

31 in hexadecimal representation is 1F.

Hence the CRC is "1F" and the entire message will be:

"/01RE0020051F" + <CR>

B. Generating a checksum for a binary request

Calculate the accumulated sum of the bytes in the vector.

Get the value of the sum modulo 65536 (10000 hex)

The CRC is the 2 complement of the result.

If lSum is a variable holding the accumulated sum, then

CRC = (Not (lSum Mod &H10000)) + 1 (VB syntax).

CRC = (~(lSum % 0x10000)) + 1 (C/C++ syntax).

Note that the CRC is a two-byte value (while in ASCII format it is a one-byte value).

Appendix 3: PCOM via Ethernet

In order to communicate via TCP (Ethernet), you must add a 6-byte long header to each command. The header is always in binary format, whether or not the command itself is given in ASCII or Binary Format.

Ethernet Header format

Bytes 0& 1: Transaction identifier.
This number is assigned by the user. The number is used by the PCOM master to identify which slave is answering a command request.

Byte 2: Select Protocol.
Identifies the protocol used for the command request. Enter:
101 (DEC) for ASCCI
102 (DEC) for Binary

Byte 3: Reserved. Leave blank.

Bytes 4&5: Length of transaction.
Enter the number of bytes required for the command request.

Notes

1. PLCs of types M90, M91 and Vision 120 do not support the TCP communication protocol.
2. Vision 230, 260, and 280 controllers must have an installed Ethernet port
3. The header must be added after message has been configured according to the message format. The Ethernet header is like an envelope, and does not affect the message. In addition, the checksums contained in the message and the length declared within the message, do not include the Ethernet header.

The Ethernet header is also irrelevant to the message contained within the PLC response. To check the response, note whether bytes 0-3 in the Ethernet header are identical to the header in the data request. After this has been checked, strip the header away to process the serial message.

Example

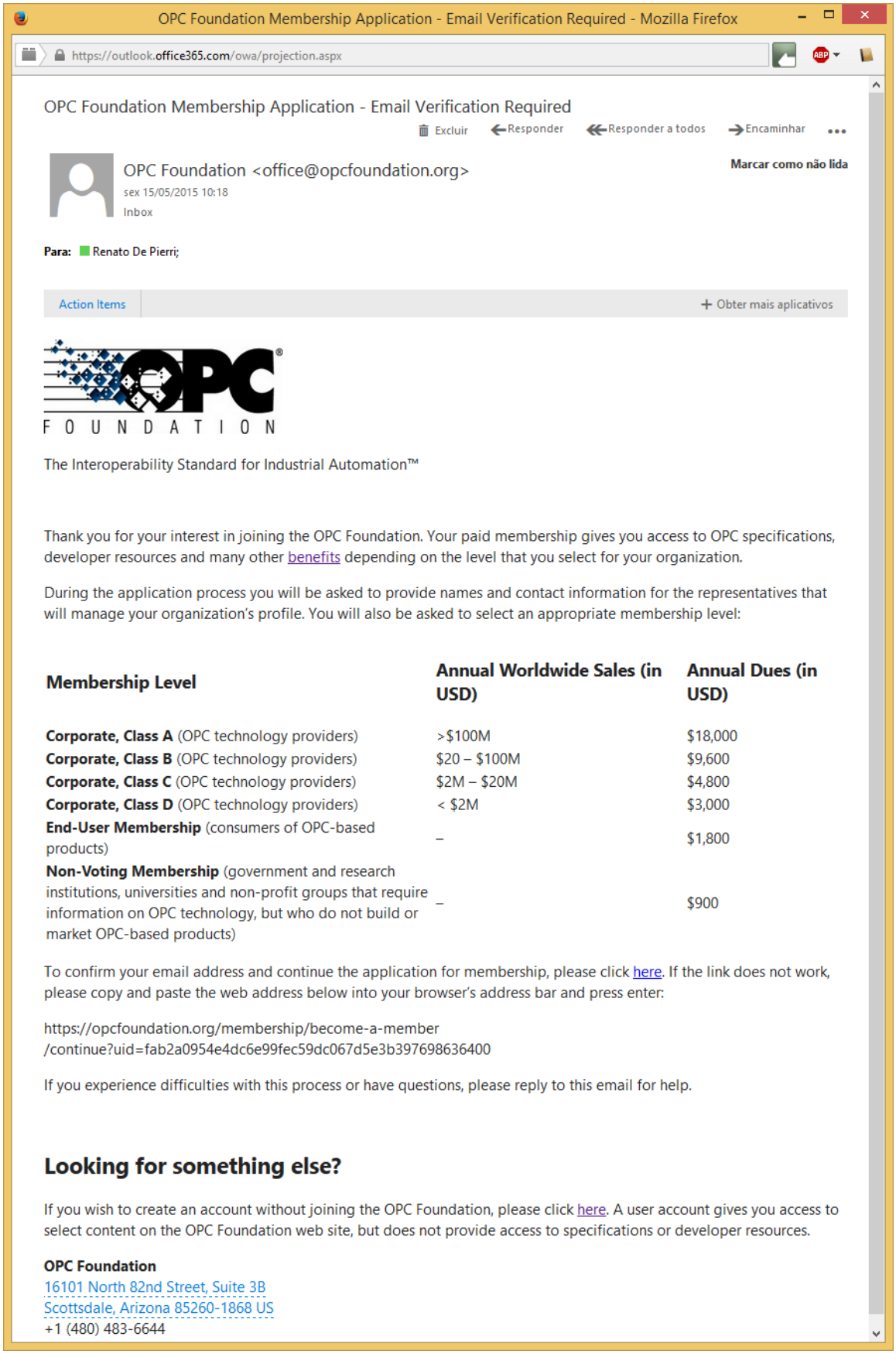
Message "/01RE0020051F" + <CR>

Byte	Decimal	ASCII
0	47	/
1	48	0
2	49	1
3	82	R
4	69	E
5	48	0
6	48	0
7	50	2
8	48	0
9	48	0
10	53	5
11	49	1
12	70	F
13	13	<CR.

If the size of this ASCII message is 14 bytes, the Transaction identifier is 1234 (DEC), the Ethernet header should be as follows:

Byte	Decimal	Comment
0	210	Low byte of 1234
1	4	High byte of 1234
2	101	Protocol: ASCII
3	0	Reserved
4	14	Low byte of Length of Transaction
5	0	High byte of Length of Transaction

ANEXO B - e-mail de filiação à fundação OPC



OPC Foundation Membership Application - Email Verification Required - Mozilla Firefox

https://outlook.office365.com/owa/projection.aspx

OPC Foundation Membership Application - Email Verification Required

Excluir | Responder | Responder a todos | Encaminhar | ...

OPC Foundation <office@opcfoundation.org>

sex 15/05/2015 10:18

Inbox

Marcar como não lida

Para: Renato De Pierri;

Action Items | + Obter mais aplicativos

OPC FOUNDATION

The Interoperability Standard for Industrial Automation™

Thank you for your interest in joining the OPC Foundation. Your paid membership gives you access to OPC specifications, developer resources and many other benefits depending on the level that you select for your organization.

During the application process you will be asked to provide names and contact information for the representatives that will manage your organization's profile. You will also be asked to select an appropriate membership level:

Membership Level	Annual Worldwide Sales (in USD)	Annual Dues (in USD)
Corporate, Class A (OPC technology providers)	>$100M	$18,000
Corporate, Class B (OPC technology providers)	$20 – $100M	$9,600
Corporate, Class C (OPC technology providers)	$2M – $20M	$4,800
Corporate, Class D (OPC technology providers)	< $2M	$3,000
End-User Membership (consumers of OPC-based products)	–	$1,800
Non-Voting Membership (government and research institutions, universities and non-profit groups that require information on OPC technology, but who do not build or market OPC-based products)	–	$900

To confirm your email address and continue the application for membership, please click here. If the link does not work, please copy and paste the web address below into your browser's address bar and press enter:

https://opcfoundation.org/membership/become-a-member
/continue?uid=fab2a0954e4dc6e99fec59dc067d5e3b397698636400

If you experience difficulties with this process or have questions, please reply to this email for help.

Looking for something else?

If you wish to create an account without joining the OPC Foundation, please click here. A user account gives you access to select content on the OPC Foundation web site, but does not provide access to specifications or developer resources.

OPC Foundation
16101 North 82nd Street, Suite 3B
Scottsdale, Arizona 85260-1868 US
+1 (480) 483-6644

ANEXO C - Montando a giga do CLP

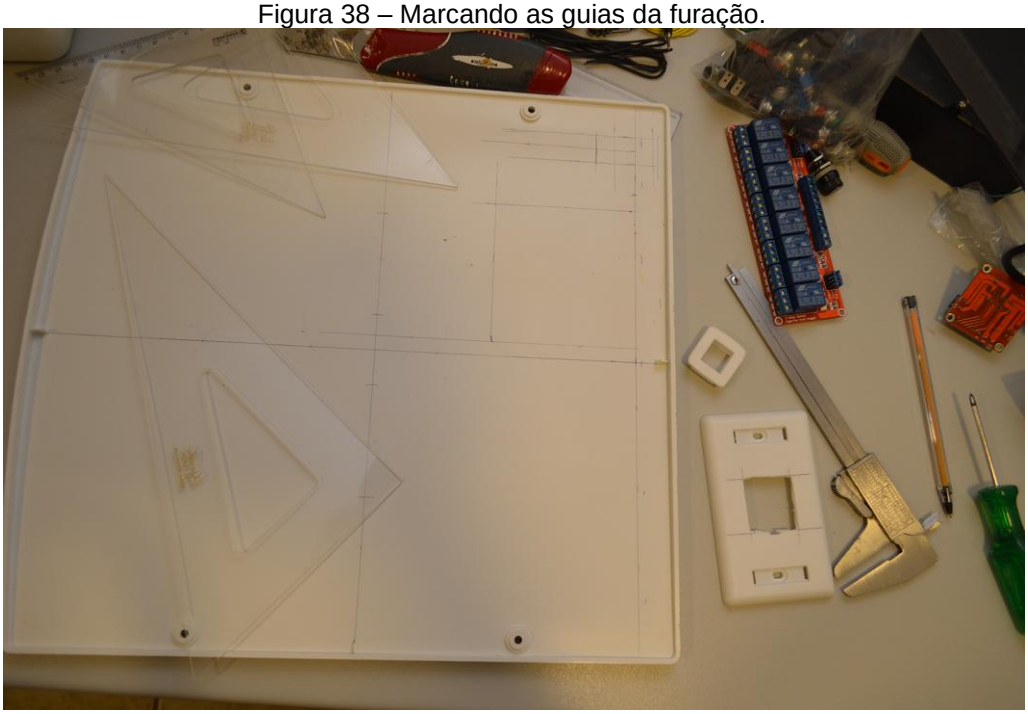
Para facilitar o estudo do CLP, foi construída uma giga de testes. As figuras 37 a 60 registram a construção da mesma. Foi escolhida a caixa de passagem de sobrepor marca Tigre modelo CPT30.

Figura 37 – Escolhendo a caixa.

Fonte: Elaborado pelo autor.

A figura 38 mostra duas linhas perpendiculares desenhadas na tampa, que é a marcação base para as demais furações e o espelho da tomada RJ45 já cortado.

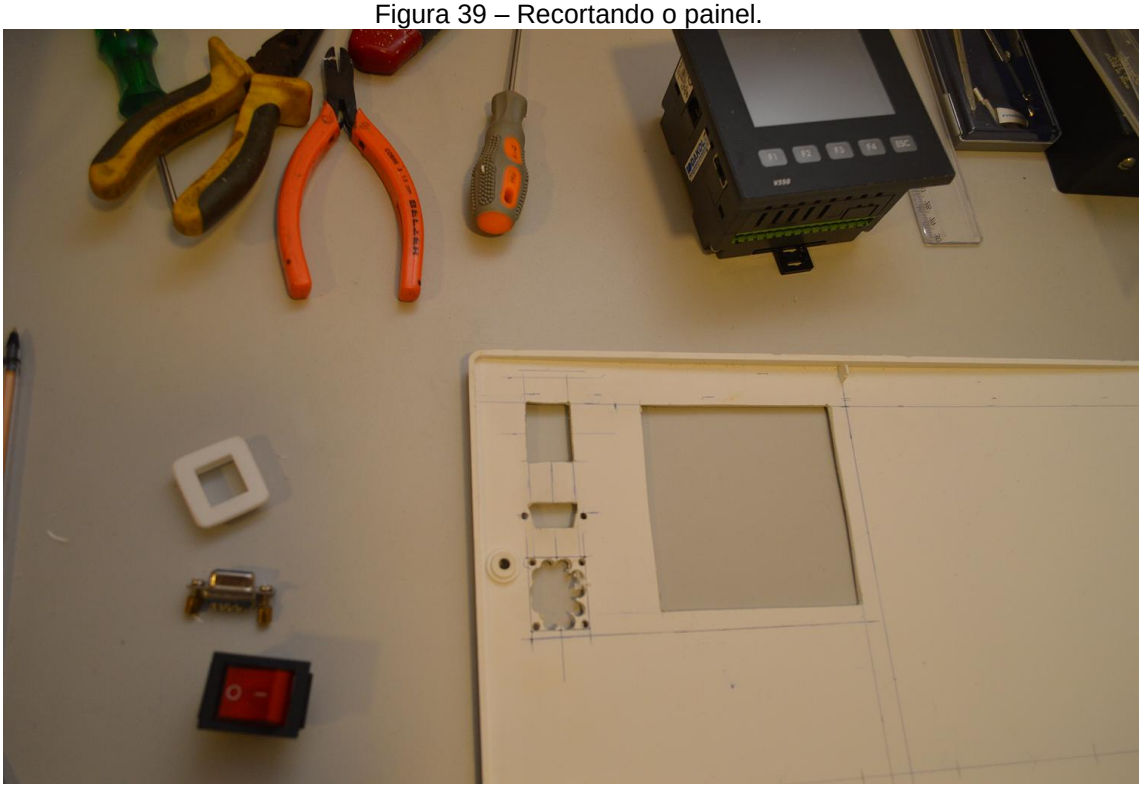
Figura 38 – Marcando as guias da furação.



Fonte: Elaborado pelo autor.

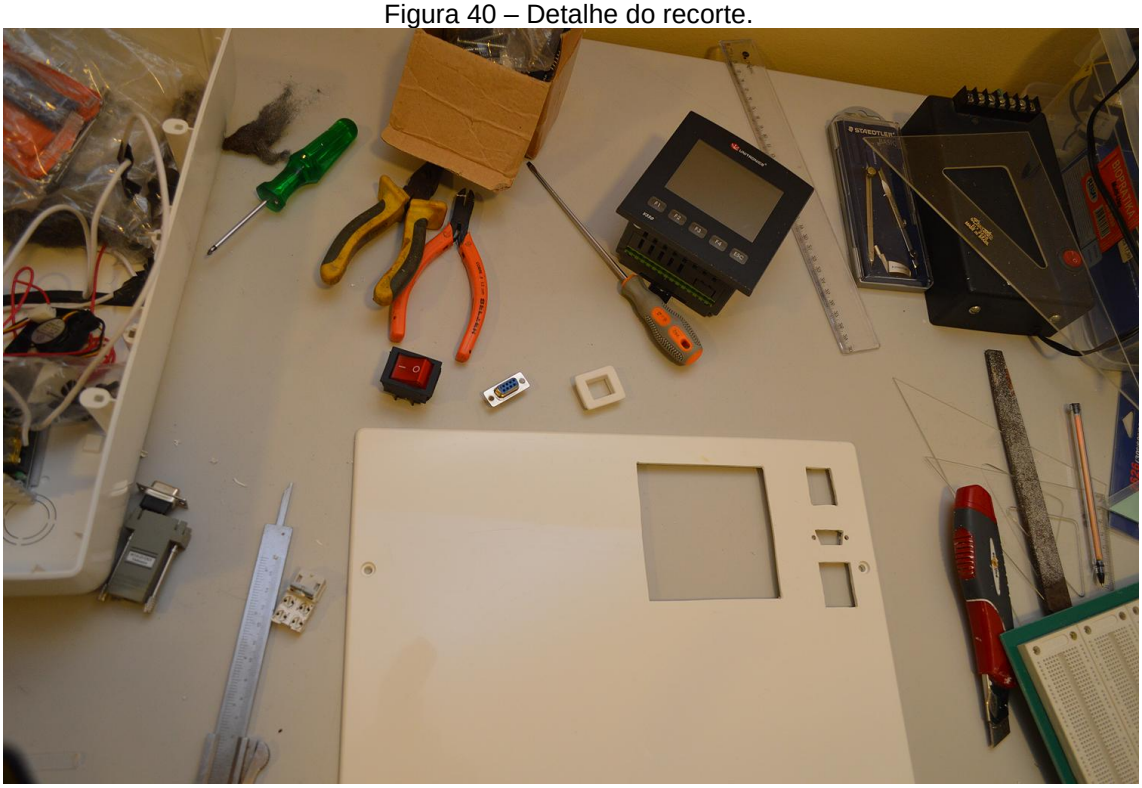
As figuras 39 e 40 mostram a furação para o conector de rede *Ethernet*, o conector DB9, interruptor geral e CLP.

Figura 39 – Recortando o painel.



Fonte: Elaborado pelo autor.

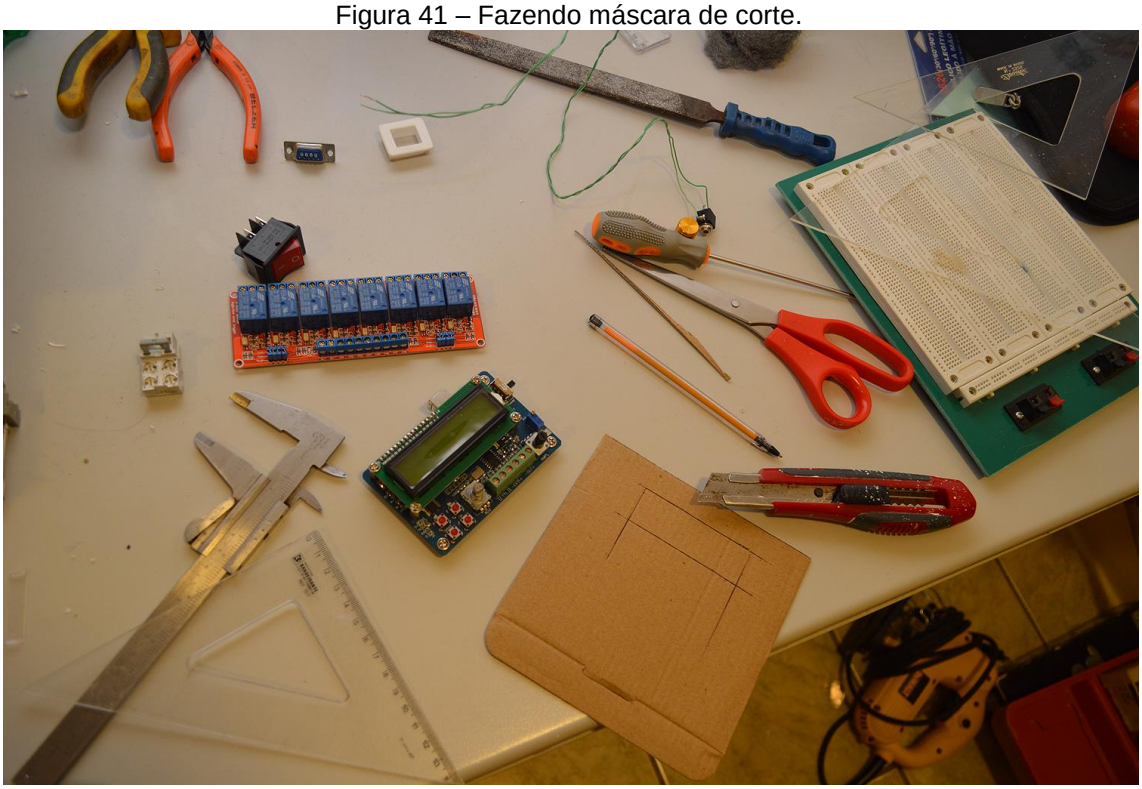
Figura 40 – Detalhe do recorte.



Fonte: Elaborado pelo autor.

Figuras 41, 42 e 43 mostram a confecção da máscara de corte do gerador de funções e demais componentes.

Figura 41 – Fazendo máscara de corte.

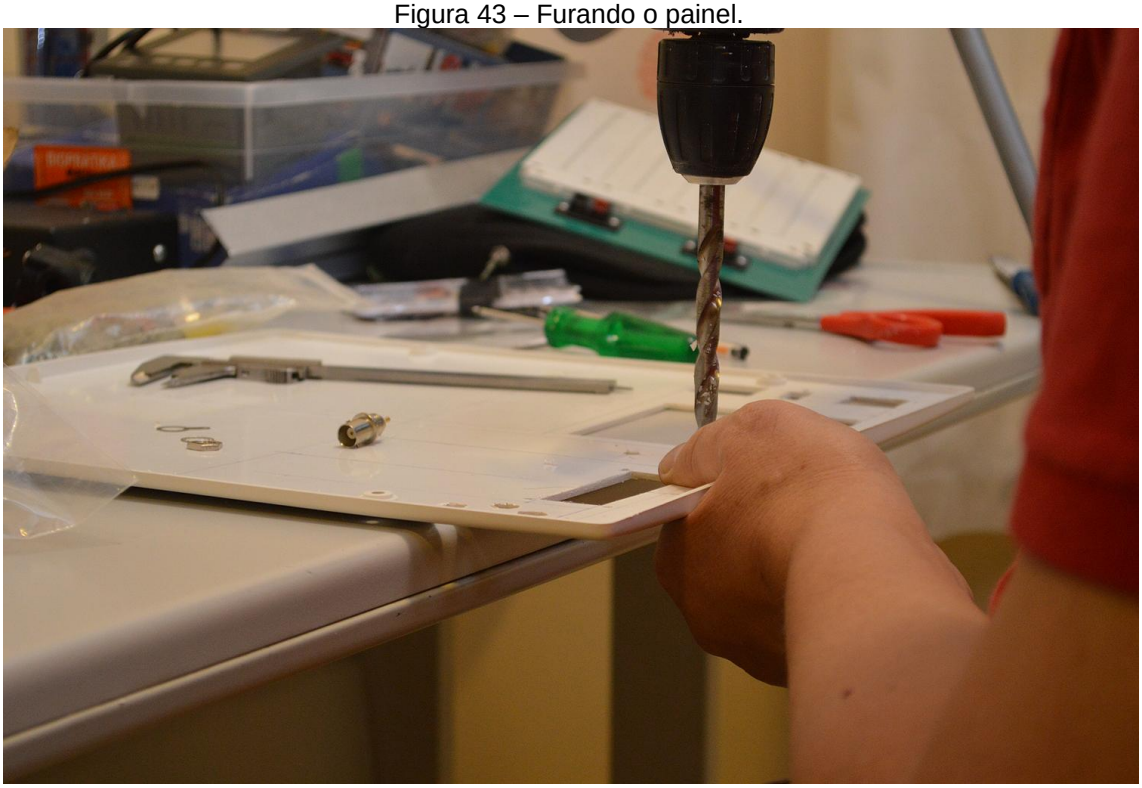


Fonte: Elaborado pelo autor.

Figura 42 – Preparando a furação.

Fonte: Elaborado pelo autor.

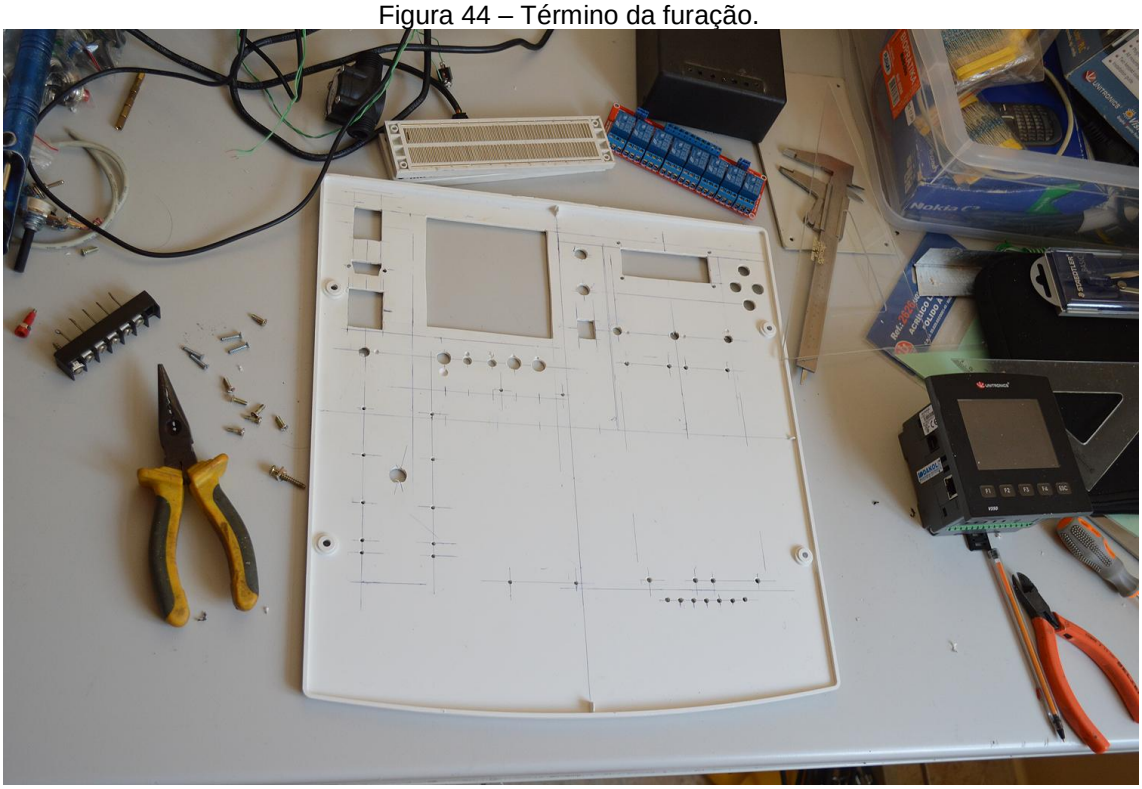
Figura 43 – Furando o painel.



Fonte: Elaborado pelo autor.

A figura 44 mostra painel com a furação pronta.

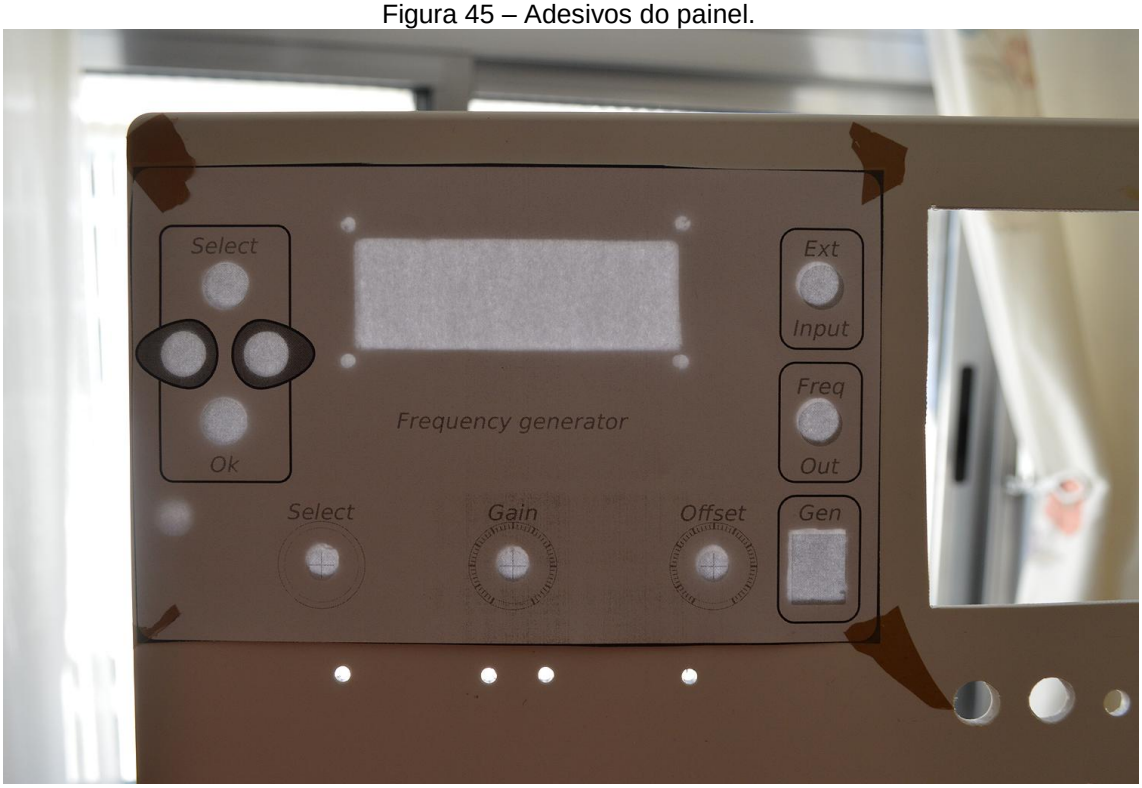
Figura 44 – Término da furação.



Fonte: Elaborado pelo autor.

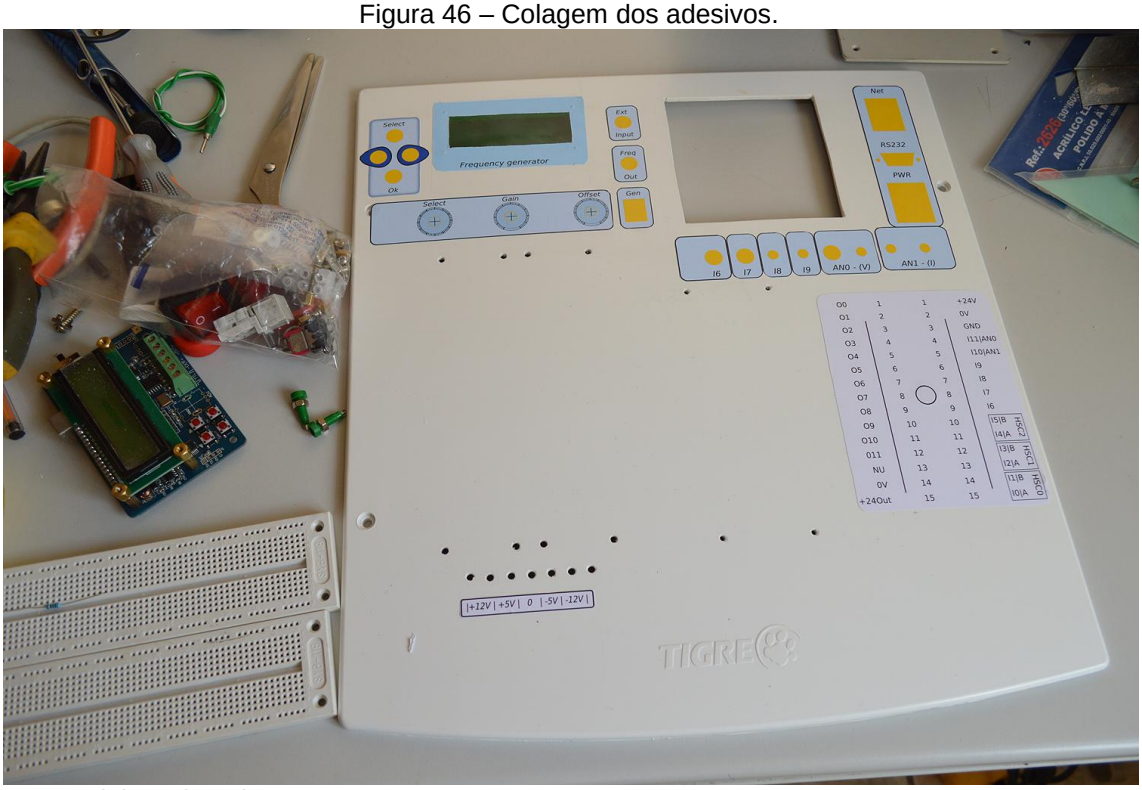
As figuras 45 e 46 mostram a confecção dos adesivos do painel. Outra opção seria confeccionar o adesivo do painel e o utilizar como marcação para a furação.

Figura 45 – Adesivos do painel.



Fonte: Elaborado pelo autor.

Figura 46 – Colagem dos adesivos.

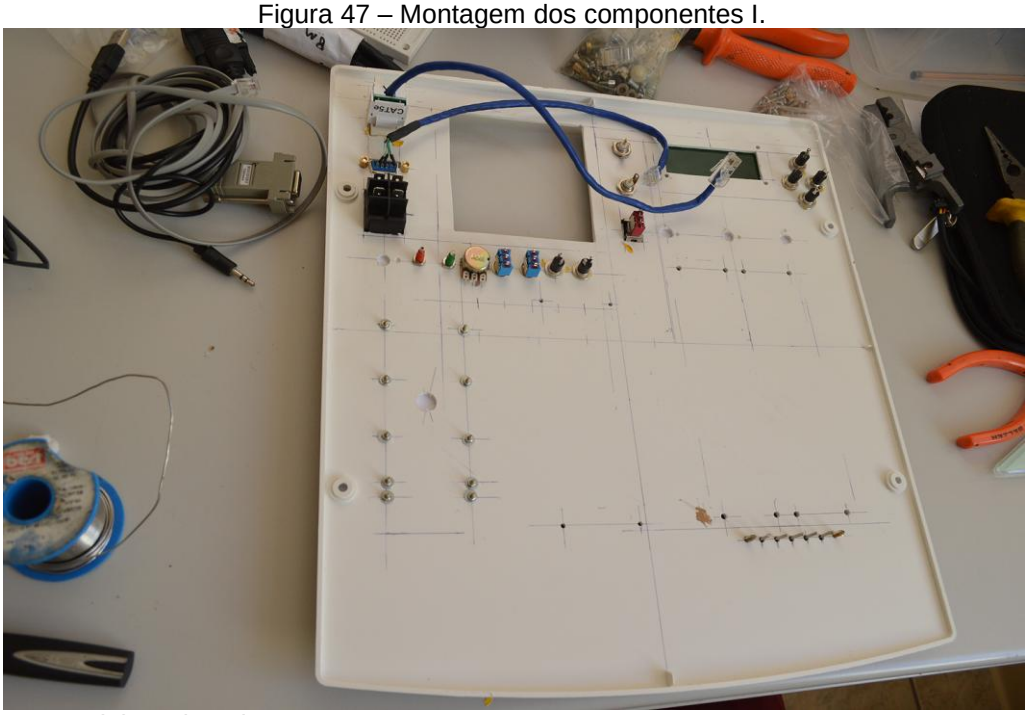


Fonte: Elaborado pelo autor.

Note que é utilizado um gel verde de refletor de palco para fazer o visor do LCD.

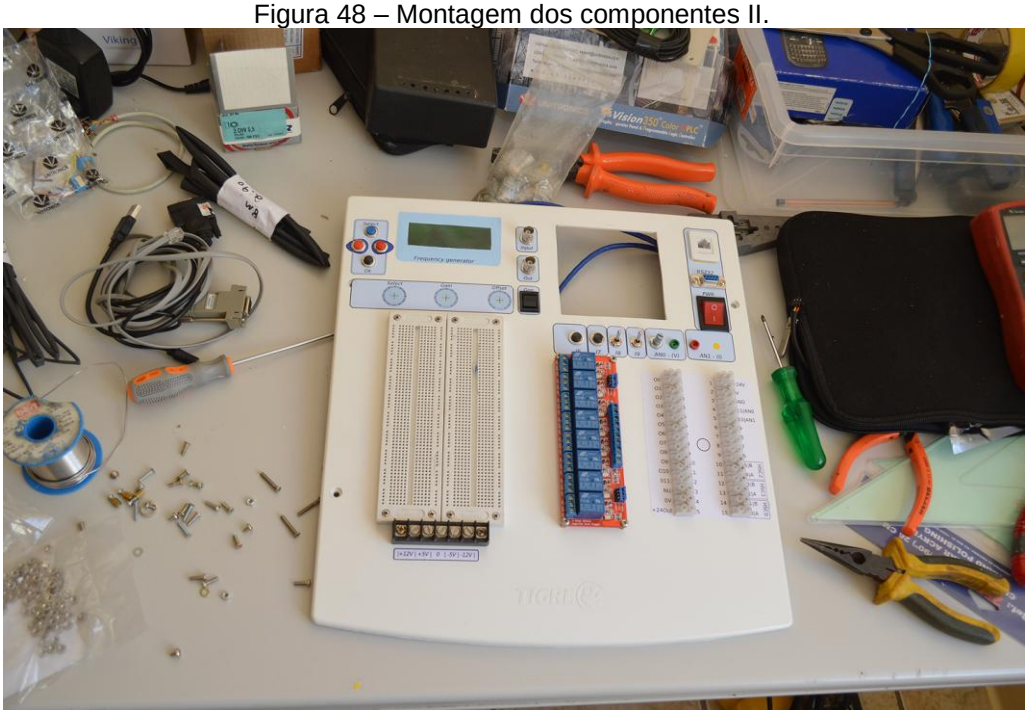
As figuras 47 e 48 mostram o início da montagem dos componentes, sempre começando dos menores para os maiores. Esta etapa é complexa porque não há espaço para erros. É necessário colocar todos parafusos, espaçadores, empregar espaguete termo-retrátil e todos recursos disponíveis a fim de se obter um efeito razoável.

Figura 47 – Montagem dos componentes I.



Fonte: Elaborado pelo autor.

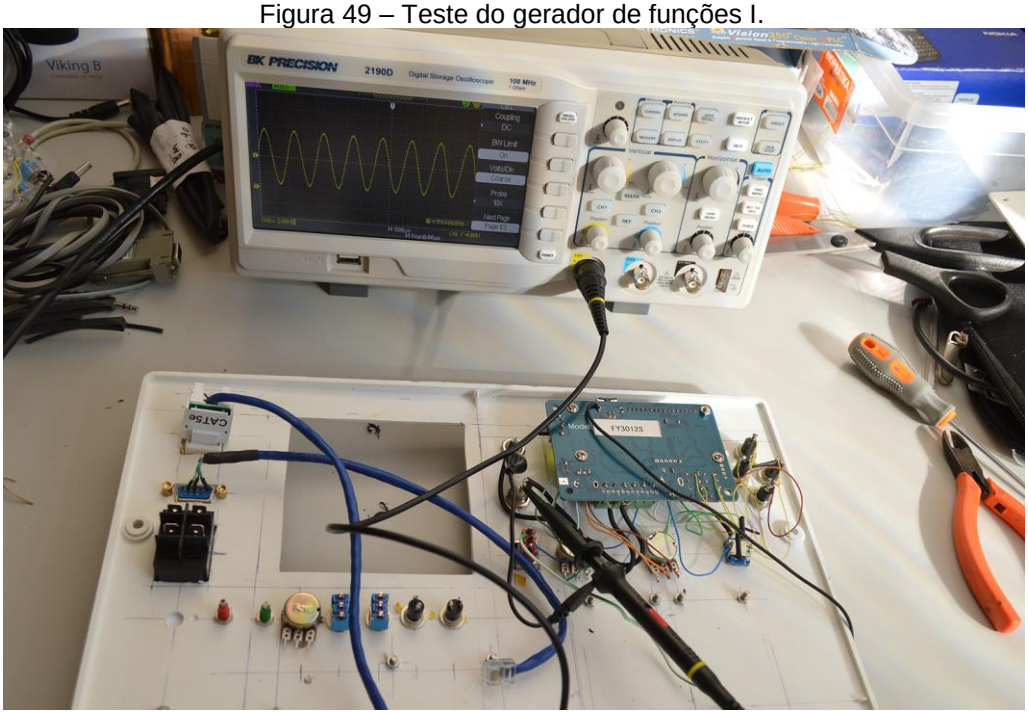
Figura 48 – Montagem dos componentes II.



Fonte: Elaborado pelo autor.

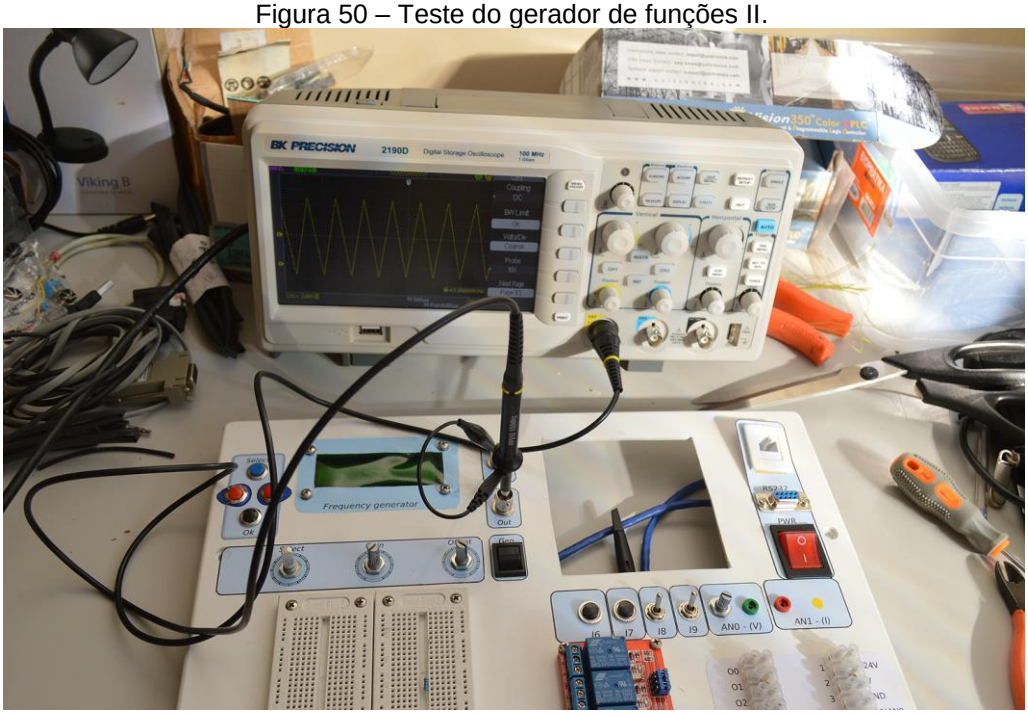
As figuras 49 e 50 mostram a instalação do gerador de funções. Por questão de espaço, alguns componentes foram movidos da PCB (placa de circuito impresso) do gerador para o painel da giga.Os testes abaixo são para assegurar que todos os controles estão funcionando ok.

Figura 49 – Teste do gerador de funções I.



Fonte: Elaborado pelo autor.

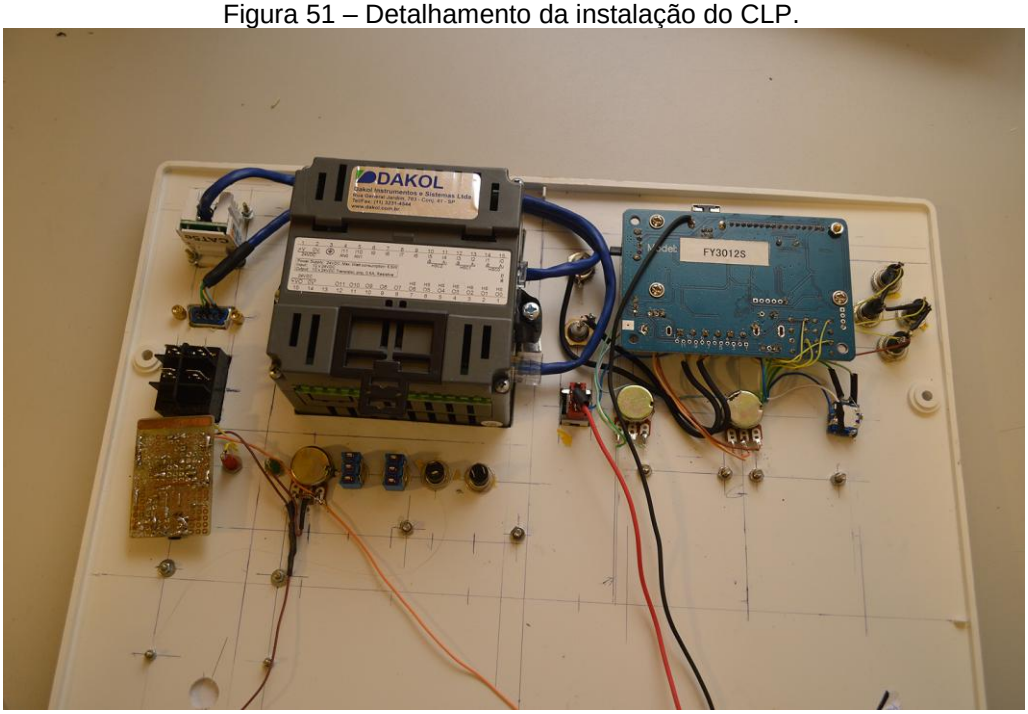
Figura 50 – Teste do gerador de funções II.



Fonte: Elaborado pelo autor.

A figura 51 mostra o gerador de funções, o CLP, a fonte de corrente já instalados.

Figura 51 – Detalhamento da instalação do CLP.



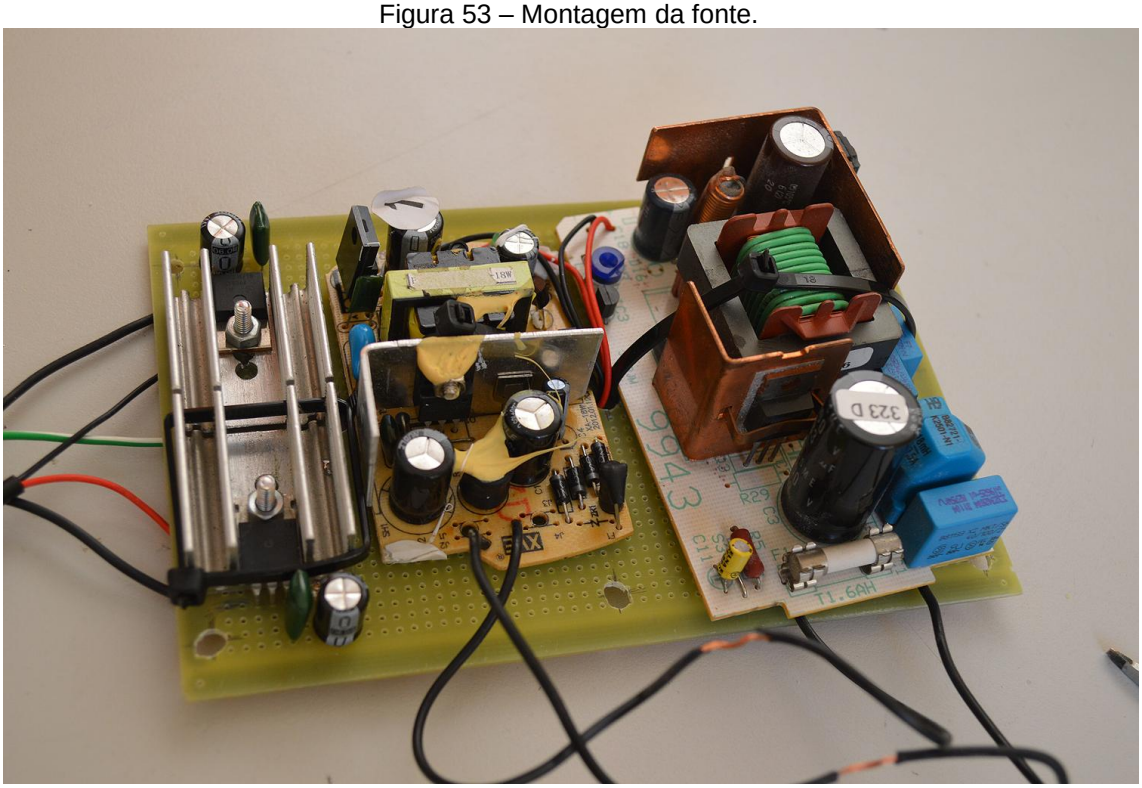
Fonte: Elaborado pelo autor.

As figuras 52 e 53 mostram a montagem de uma fonte "+12V 2,4A", "+5V 1A", 0, "-5V 1A", "-12V 2A" a partir de fontes de roteadores antigos que seriam descartadas. As fontes são desmontadas, coloca-se em uma caixa e adiciona-se os reguladores LM7805 e LM7905 para se obter as tensões +5 e -5V. Essas tensões ficam disponíveis para alimentar o *bread board* e o gerador de funções e para alimentação de circuitos eletrônicos de baixo consumo de corrente.

Figura 52 – Desmontagem da fonte chaveada.

Fonte: Elaborado pelo autor.

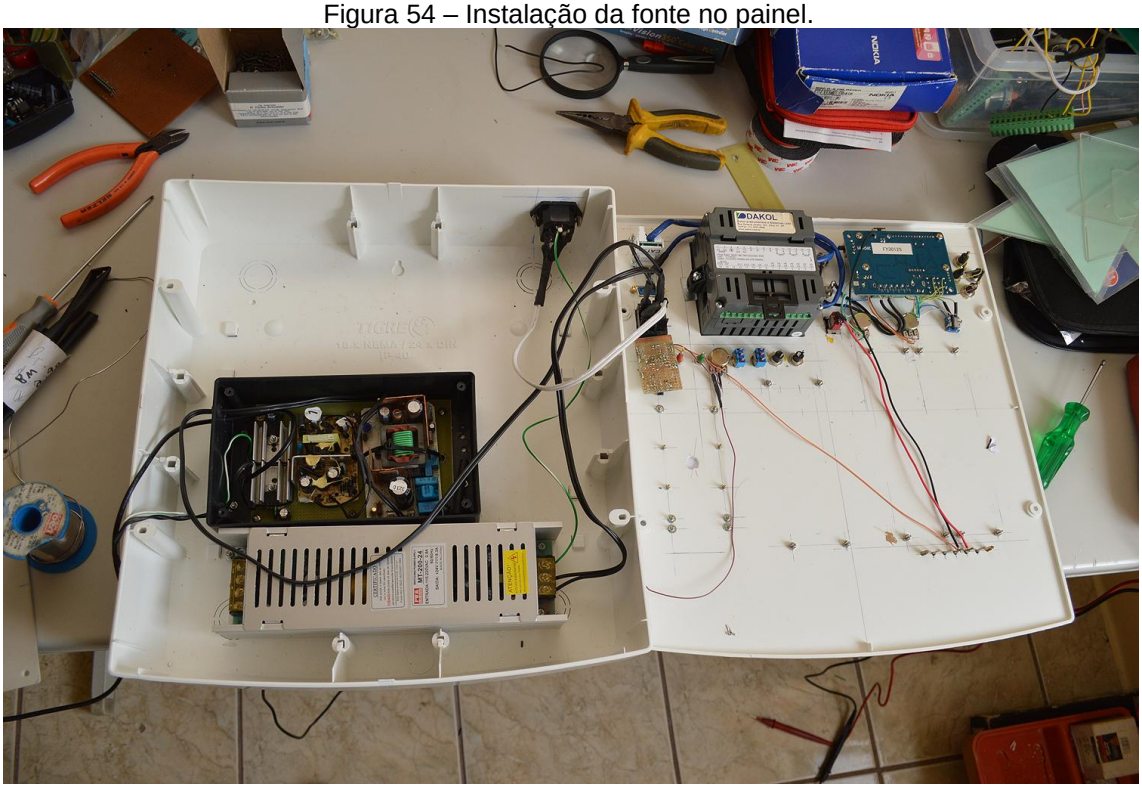
Figura 53 – Montagem da fonte.



Fonte: Elaborado pelo autor.

A figura 54 mostra a instalação das fontes de alimentação do *bread board* e do CLP. Para o CLP utilizou-se uma fonte chaveada de 24V 8,3A exclusiva, com a saida disponível na régua de bornes do CLP para acionamento de cargas maiores.

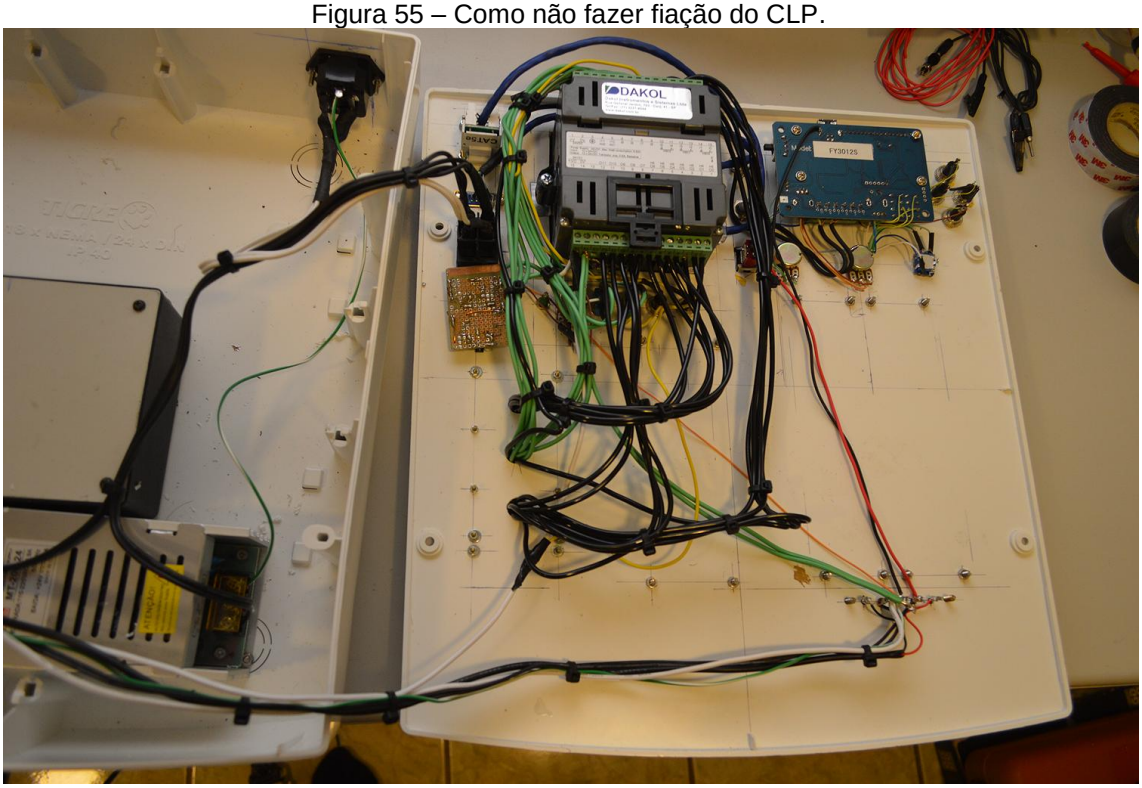
Figura 54 – Instalação da fonte no painel.



Fonte: Elaborado pelo autor.

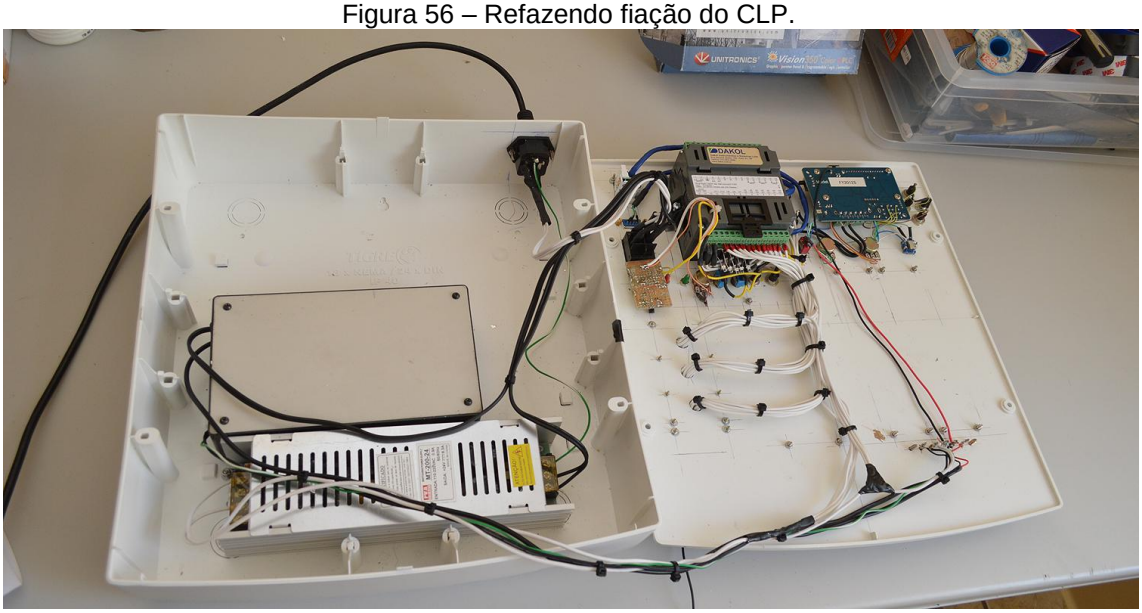
A figura 55 mostra a conexão utilizando fiação 1,5mm^2. Cabos duros, difícil manuseio, chegou a quebrar conectores e potenciômetros e a caixa fechava com dificuldade. Joguei tudo fora, comprei um rolo de cabo 0,75mm^2, bornes e refiz a fiação direito conforme figura 56.

Figura 55 – Como não fazer fiação do CLP.



Fonte: Elaborado pelo autor.

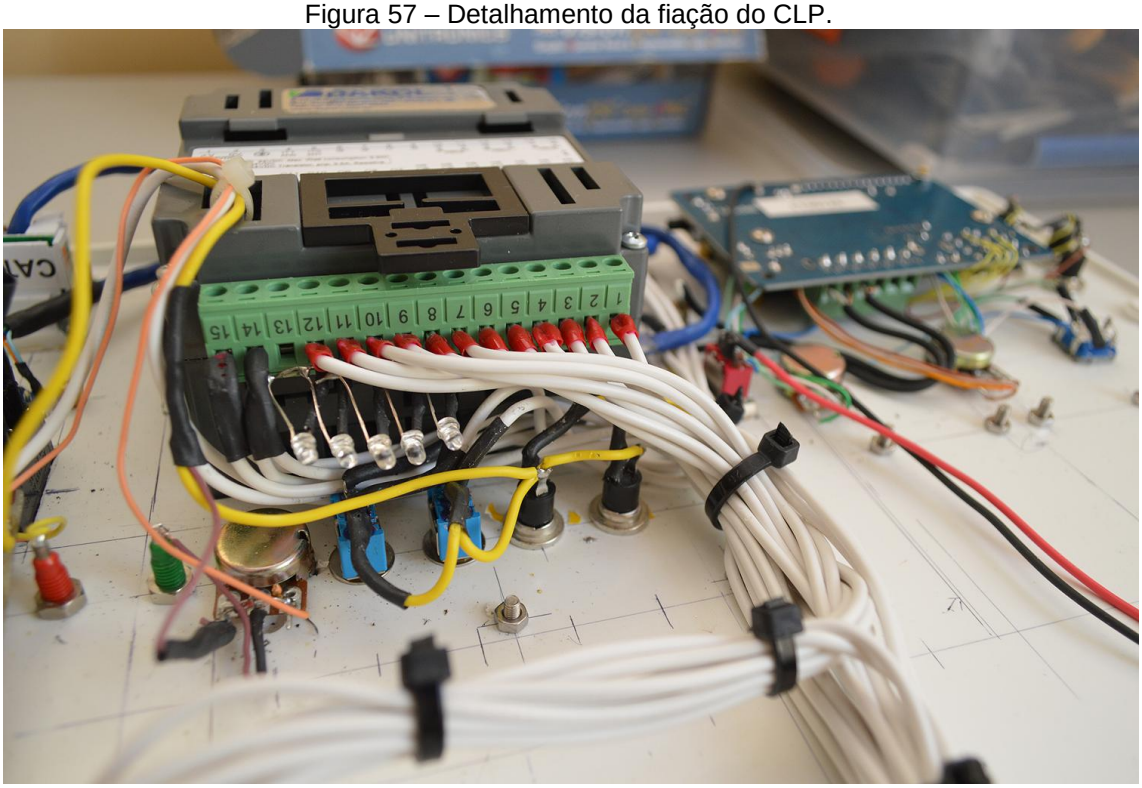
Figura 56 – Refazendo fiação do CLP.



Fonte: Elaborado pelo autor.

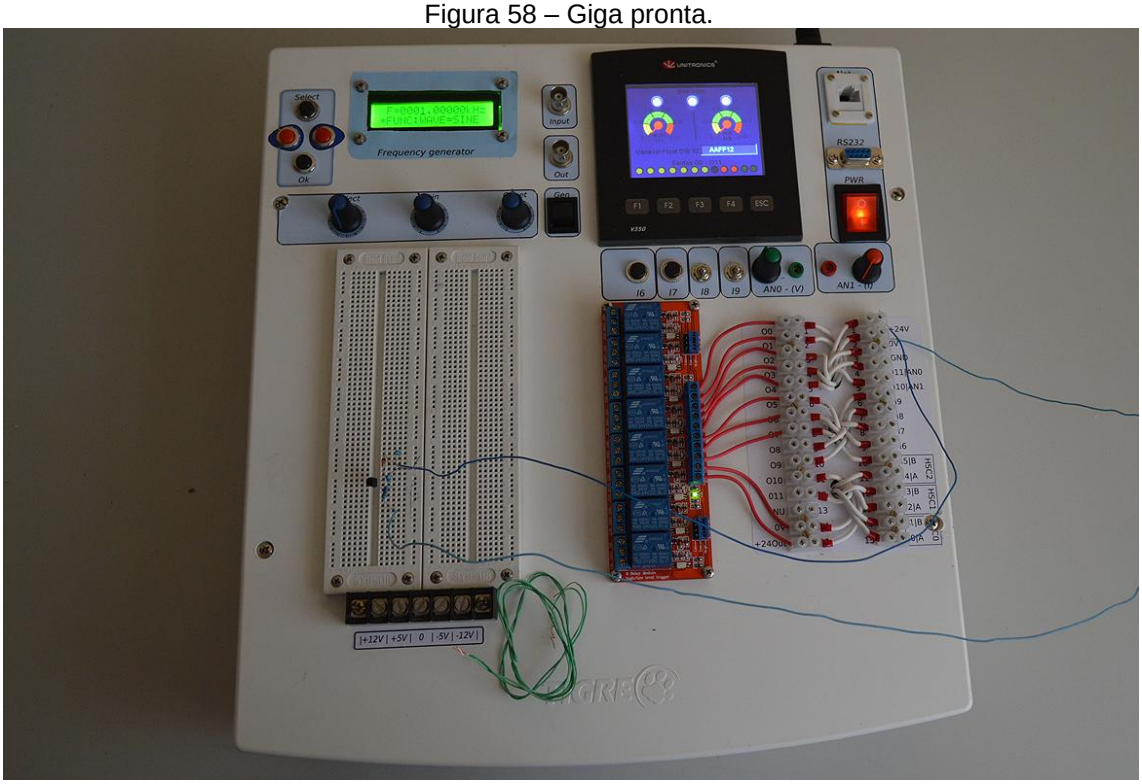
A figura 57 apresenta detalhe da fiação e montagem e as figuras 58, 59 e 60 mostram a giga pronta.

Figura 57 – Detalhamento da fiação do CLP.



Fonte: Elaborado pelo autor.

Figura 58 – Giga pronta.

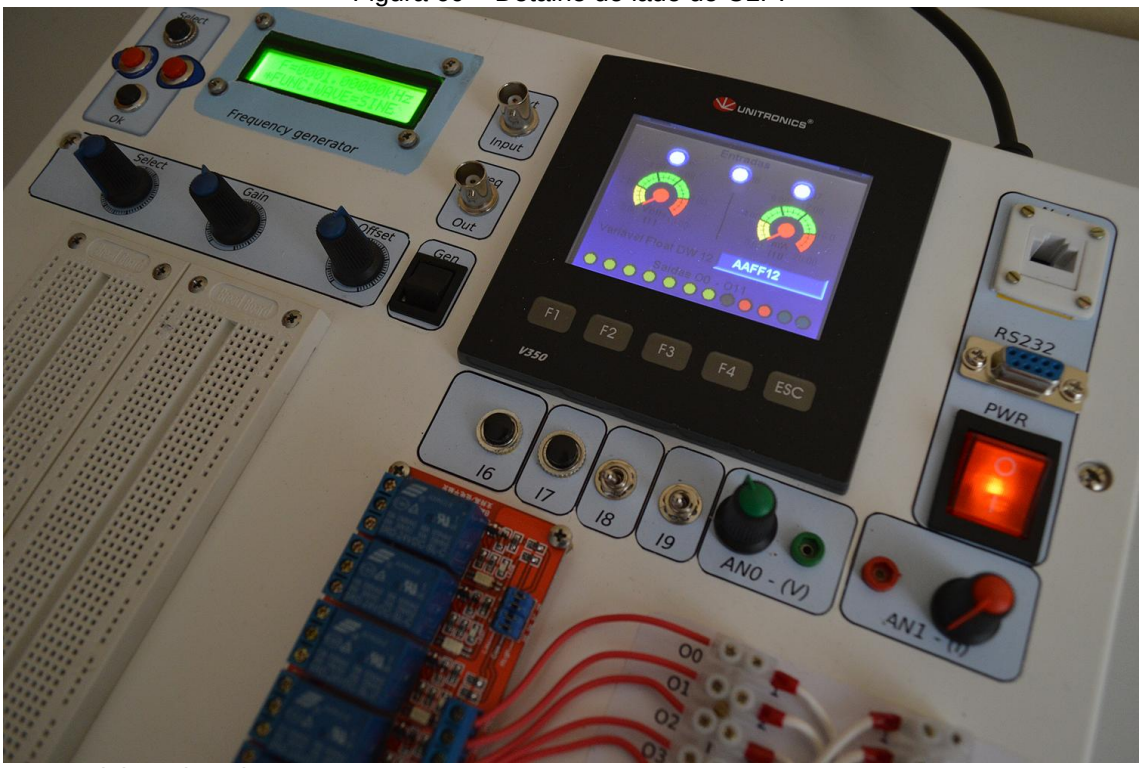


Fonte: Elaborado pelo autor.

Figura 59 – Detalhe do gerador de funções.

Fonte: Elaborado pelo autor.

Figura 60 – Detalhe do lado do CLP.



Fonte: Elaborado pelo autor.