UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO, DE CIÊNCIAS EXATAS E EDUCAÇÃO
DEPARTAMENTO DE ENG. DE CONTROLE, AUTOMAÇÃO E COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Raul Guilherme Siofre Pinesso

Desenvolvimento de *function blocks* para implementar *datalogger* em CLP

Blumenau
2024

Raul Guilherme Siofre Pinesso

Desenvolvimento de *function blocks* para implementar *datalogger* em CLP

Trabalho de Conclusão de Curso de Graduação em Engenharia de Controle e Automação do Centro Tecnológico, de Ciências Exatas e Educação da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Engenheiro de Controle e Automação.
Orientador: Prof. Dr.Guilherme Brasil Pintarelli

Blumenau
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Pinesso, Raul Guilherme Siofre
Desenvolvimento de function blocks para implementar datalogger em CLP / Raul Guilherme Siofre Pinesso ; orientador, Guilherme Brasil Pintarelli, 2024.
60 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Blumenau, Graduação em Engenharia de Controle e Automação, Blumenau, 2024.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Engenharia de Controle e Automação. 3. CLP. 4. Blocos funcionais . 5. datalogger. I. Pintarelli, Guilherme Brasil. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

Raul Guilherme Siofre Pinesso

Desenvolvimento de *function blocks* para implementar *datalogger* em CLP

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Engenheiro de Controle e Automação" e aprovado em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação.

Blumenau, dia de mês de 2024.

Banca Examinadora:

Documento assinado digitalmente
Guilherme Brasil Pintarelli
Data: 08/07/2024 20:28:25-0300
CPF: ***.523.129-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Guilherme Brasil Pintarelli
Universidade Federal de Santa Catarina - UFSC

Documento assinado digitalmente
Leonardo Mejia Rincon
Data: 08/07/2024 20:56:59-0300
CPF: ***.678.849-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Leonardo Mejia Rincon
Universidade Federal de Santa Catarina - UFSC

Documento assinado digitalmente
Ebrahim Samer El Youssef
Data: 10/07/2024 08:59:35-0300
CPF: ***.252.109-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Ebrahim Sammer El Youssef
Universidade Federal de Santa Catarina - UFSC

Dedico este trabalho aos meus pais e aos meus irmãos que
sempre estiveram presentes, mesmo que longe.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Fabiano Pinesso e Silvia Andreia Siofre, por toda educação dada, que me tornou a pessoa que sou hoje, e por não medirem esforços durante minha graduação para que eu concluísse essa etapa da minha vida. Aos meus irmãos, Fabiano Pinesso Junior e Augusto Siofre Pinesso, pelo apoio e parceria de sempre.

Agradeço a Cris, por todo o incentivo e compreensão para que esse trabalho fosse finalizado da melhor maneira possível.

Agradeço a todo corpo de professores da UFSC Blumenau, em especial ao meu orientador Dr. Guilherme Brasil Pintarelli, que sempre foi muito solícito dando todo suporte para a realização desse trabalho.

Por fim, agradeço a todos os colegas que fizeram parte da minha graduação que me ajudaram diretamente ou indiretamente de alguma forma.

"Todas as inovações eficazes são surpreendentemente simples. Na verdade, maior elogio que uma inovação pode receber é haver quem diga: Isto é óbvio! Por que não pensei nisso antes?" (Peter Drucker)

RESUMO

Este trabalho aborda a criação de blocos funcionais para utilizar o Controlador Lógico Programável (CLP) como um datalogger, destacando sua importância na área de tratamento de efluentes. A pesquisa foi motivada pela necessidade de coleta e tratamento eficientes de dados nas estações de tratamento. O principal objetivo deste estudo é criar um bloco funcional desenvolvido usando lógica de programação em CLP para realizar a leitura e armazenamento de dados obtidos de sensores de campo. Além disso, buscou-se criar um temporizador de alta precisão e simular o escalonamento de dois tipos diferentes de sensores usando um gerador de sinal. Para alcançar os objetivos propostos, foram utilizadas linguagens de programação do CLP. O estudo incluiu coleta e análise de dados, cálculo de médias e uso do sistema de *words* e *bits* do fabricante do CLP. Os resultados foram obtidos por meio de testes realizados em um CLP comercial, simulando sensores com o uso de um gerador de sinal. A partir dos resultados, foi possível estimar que o programa pode ser utilizado com sensores de diferentes sinais analógicos. Concluiu-se que o bloco pode ser modificado conforme necessário e que a rotina de leituras foi automatizada. Uma vez configurado, o programa pode operar de forma autônoma, sem necessidade de monitoramento externo. Os resultados deste trabalho têm implicações positivas para a aplicação prática em estações de tratamento de água e efluentes. Este estudo contribui para a área de saneamento básico, simulando um ambiente de tratamento de água que poderia ser aplicado em tanques de tratamento reais.

Palavras-chave: CLP M340; Array Dinâmica; Linguagem Ladder; Diagrama de Blocos Funcionais; Texto Estruturado.

ABSTRACT

This work addresses the creation of functional block to use the Programmable Logic Controller (PLC) as a datalogger, highlighting its importance in the field of effluent treatment. The research was motivated by the need for efficient data collection and processing in treatment plants. The main objective of this study is to create a functional block developed using PLC programming logic to perform the reading and storage of data obtained from field sensors. Additionally, a highly precise timer was sought to be created, and the scheduling of two different types of sensors using a signal generator was simulated. To achieve the proposed objectives, PLC programming languages and functional blocks were utilized for task execution. The study included data collection and analysis, average calculation, and the use of the PLC manufacturer's word and bit system. The results were obtained through tests conducted on a commercial PLC, simulating sensors using a signal generator. From the results, it was estimated that the program can be used with sensors of different analog signals. It was concluded that the block can be modified as needed and that the reading routine was automated. Once configured, the program can operate autonomously without external monitoring. The results of this work have positives implications for practical application in water and effluent treatment plants. This study contributes to the field of basic sanitation by simulating a water treatment environment that could be applied in real treatment tanks.

Keywords: M340 PLC; Dinamic Array; Ladder Language; Function Block Diagram; Structured Text.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

CLP	Controlador Lógico Programável
CTU	UP counter
DIN	Deutsches Institut für Normung
ERP	Enterprise Resources Planing
ETA	Estação de Tratamento de Água
FB	Function Blocks
FBD	Function Block Diagram
IEC	International Electrotechnical Comission
IL	Instruction List
ISO	International Organization for Standardization
LD	Ladder Diagram
M340	Modicon 340
MES	Manufacturing Execution System
SCADA	Supervisory Control And Data Acquisition
SFC	Sequential Function Chart
ST	Structured Text
TOF	Timer off-delay
TON	Timer on-delay
TP	Pulse timer

SUMÁRIO

1 INTRODUÇÃO

A evolução dos computadores e dos Controladores Lógicos Programáveis (CLPs) na indústria tem impulsionado uma transformação significativa no campo da automação e controle industrial. Esses avanços tecnológicos permitem maior precisão, eficiência e flexibilidade na gestão de processos industriais. Além disso, a integração de sistemas avançados de monitoramento e controle tem possibilitado a coleta e análise de grandes volumes de dados, aprimorando a tomada de decisões e a manutenção preditiva. Como resultado, as indústrias podem alcançar níveis mais elevados de produtividade e segurança operacional. Sistemas de aquisição de dados e controle de dispositivos vêm sendo desenvolvidos para diferentes áreas de atuação, tanto industriais como científicas. O seu objetivo é apresentar ao observador os valores das variáveis ou parâmetros que estão sendo medidos. (INSTRUMATIC, 2011).

Uma ETA (Estação de Tratamento de Água) pode produzir mais de 100 mil litros de água tratada por segundo. O sistema sequencial de tratamento da água inclui: cloração, alcalinização, coagulação, floculação, decantação, filtração, desinfestação, fluoração e transporte para reservatórios de bairros. Esse processo é longo, bem estabelecido e necessita de grandes investimentos financeiros para que a população receba água de qualidade em suas casas (FLORES, 2020). A necessidade de tratamento de dados adquiridos nos tanques onde ocorrem os processos citados é fundamental para garantir a eficácia e a eficiência dessas etapas. As estações de tratamento de efluentes desempenham um papel crítico na remoção de poluentes e substâncias contaminantes de águas residuais industriais e domésticas, tornando-as seguras para o descarte no meio ambiente ou, em alguns casos, para a reutilização. No entanto, a fim de garantir a realização bem-sucedida desse objetivo, os dados resultantes do processo de tratamento devem ser coletados, monitorados e analisados de acordo com as regras definidas.

A falta de gestão e análise de dados dados em estações de tratamento de efluentes pode levar a consequências negativas. Primeiramente, a falta de informações em tempo real sobre o processo pode resultar em um controle ineficaz da estação, o que pode levar a descargas inadequadas de efluentes tratados no meio ambiente, causando danos à saúde pública e ao ecossistema. Além disso, a ausência de análise de dados pode resultar em um uso ineficiente dos recursos, como energia e produtos químicos, aumentando os custos operacionais. Portanto, a gestão e análise de dados é crucial para garantir que as estações de tratamento de efluentes operem de forma sustentável, cumprindo as regulamentações ambientais e contribuindo para a preservação do meio ambiente.

Em uma estação de tratamento de efluentes, variáveis devem ser monitoradas e parâmetros devem ser ajustados, incluindo fluxo de água, concentrações de substâncias químicas, níveis de pH, temperatura e outros. Esses dados podem ser provenientes de uma variedade de fontes, como sensores instalados em diferentes pontos do processo,

instrumentos de laboratório e sistemas de monitoramento ambiental. Uma vez que os dados são coletados de múltiplas fontes, o próximo passo é o tratamento de dados. Isso pode envolver o *data parsing*, a filtragem e processamento de sinais, correção de erros e anomalias, fusão de dados e armazenamento para que o sistema possa tomar decisões com base neles. O *data parsing* envolve interpretação e conversão dos dados brutos de sensores para um formato estruturado e utilizável. A filtragem e processamento de sinais eliminam o ruído e extraem características significativas dos dados coletados. A correção de erros e detecção de anomalias garantem a integridade dos dados transmitidos. A fusão de dados combina informações de múltiplos sensores para obter uma representação mais precisa e confiável. O armazenamento de dados envolve salvar informações coletadas em bancos de dados estruturados para recuperação e análise futura.

A capacidade do CLP de tratar dados de várias fontes traz benefícios pois permite que a estação de tratamento de efluentes seja adaptável às condições em tempo real, melhorando a eficiência operacional. Além disso, a capacidade de agregar informações de diferentes fontes facilita a detecção de tendências, a análise de desempenho e a identificação de problemas potenciais. Outro benefício dos CLPs é que, caso exista algum na planta, pode ser reaproveitado. Este reaproveitamento não só economiza tempo e recursos, mas também minimiza interrupções nos processos existentes. Além disso, os CLPs são conhecidos por serem soluções de baixo custo, oferecendo uma excelente relação custo-benefício para a automação e controle industrial. Com investimentos baixos, é possível implementar melhorias significativas na coleta e análise de dados. Assim, ao aproveitar a infraestrutura existente e incorporar a capacidade avançada de tratamento de dados dos CLPs, as estações de tratamento de efluentes podem alcançar um nível superior de desempenho operacional."

Nesse contexto, este trabalho tem como foco o estudo e aplicação do CLP para aprimorar a eficiência, as práticas em conformidade com regulamentos e a sustentabilidade nos processos de tratamento de efluentes. O projeto foi realizado em colaboração com a empresa Hidryco, Figura 1, sediada em Blumenau, Santa Catarina, a qual se especializa no desenvolvimento de sistemas de automação industrial para tratamento de água e efluentes, bem como sistemas de deslocamento de fluidos.

Figura 1 – Hidryco Blumenau.

Fonte: (HIDRYCO, 2023).

1.1 OBJETIVOS

Nas seções abaixo são descritos o objetivo geral e os objetivos específicos deste TCC.

1.1.1 Objetivo Geral

O objetivo geral deste trabalho consiste em desenvolver blocos funcionais usando CLP com a finalidade de realizar a aquisição e o tratamento de dados obtidos a partir de diferentes sensores em campo.

1.1.2 Objetivos Específicos

Para a execução do objetivo geral apontado, os seguintes objetivos específicos devem ser realizados para o desenvolvimento do projeto ocorrer de maneira esperada:

1. Ler e armazenar valores de sensores externos ao CLP;
2. Construir um bloco de função no CLP para gerir os passos de tempo da amostragem de sinais (*ticks*);
3. Automatizar rotinas de leituras;
4. Construir um bloco de função CLP para escalonar variáveis de forma *user friendly*;
5. Implementar e testar programas em CLP comercial.

2 REVISÃO BIBLIOGRÁFICA

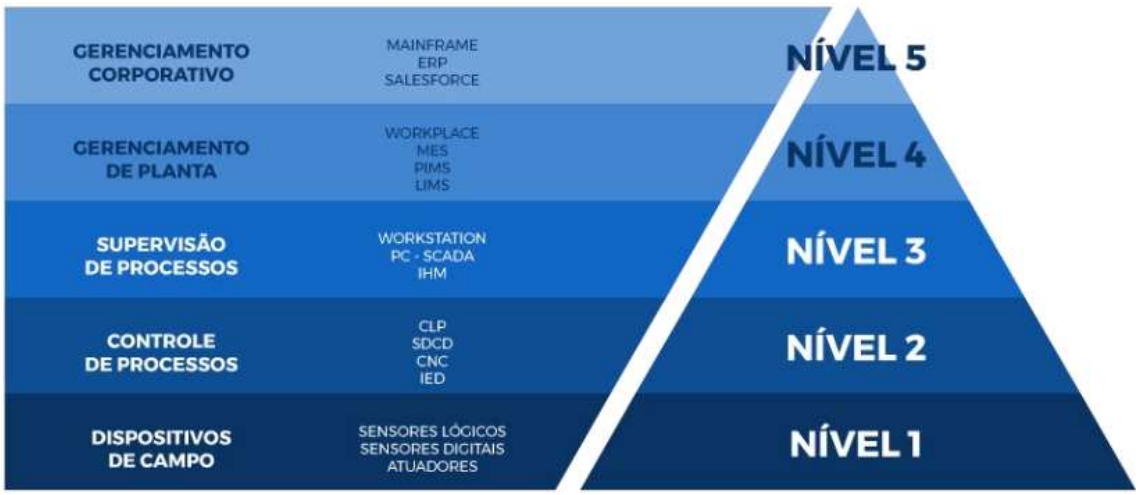
Neste capítulo, são apresentados os conteúdos relevantes que foram utilizados para o desenvolvimento deste trabalho.

2.1 AUTOMAÇÃO DE PROCESSOS

A automação refere-se a qualquer sistema apoiado em computadores e ou redes lógicas que substitua o trabalho humano em favor da segurança das pessoas, da qualidade dos produtos, da rapidez da produção ou da redução de custos, assim aperfeiçoando os complexos objetivos das industrias e dos serviços (MORAES; CASTRUCCI, 2007). Quando se fala sobre automação industrial, nem sempre se tem uma noção exata do tamanho desta área do conhecimento. Inclusive, iniciantes na automação industrial poderiam acreditar que a mesma se limita a pequenos campos isolados sem nenhuma correlação. Contudo, a automação industrial é abrangente e engloba não só a indústria, como também o comércio, os serviços e a logística. Isso por que, o objetivo de automatizar os trabalhos é melhorar a eficiência de produção e reduzir custos (GROOVER, 2010).

A pirâmide da automação, elaborada através do modelo funcional ISA95, é um método para implementar automação em indústrias, e categoriza as atividades em cinco níveis, como mostra a Figura 2, cada um com funções, responsabilidades e equipamentos específicos. A pirâmide especifica meios de comunicação entre seus cinco níveis, Ou seja, há fluxo da informação da base da pirâmide (composta de equipamentos de campo), até o topo (composta de sistemas de gerenciamento corporativo). A norma ISA95 tem como objetivo estabelecer uma interface mais eficiente entre as funções de controle e as funções corporativas.

Figura 2 – Pirâmide da Automação.



Fonte: (ALTUS, 2023).

Uma breve descrição de cada nível, segundo (MORAES; CASTRUCCI, 2007), a seguir:

- **Nível 1:** é o nível das máquinas, dispositivos e componentes (chão de fábrica).
- **Nível 2:** é o nível caracterizado pela presença de um controlador industrial, como um CLP.
- **Nível 3:** permite a supervisão do processo produtivo da planta; normalmente é constituído por bancos de dados com informações dos índices de qualidade da produção, relatórios e estatísticas de processo, índices de produtividade, algoritmos de otimização da operação produtiva. Neste nível encontram-se sistemas de supervisão SCADA.
- **Nível 4:** é o nível responsável pela programação e pelo planejamento da produção, realizando o controle e a logística dos suprimentos. Nesse nível também considera-se softwares do tipo *manufacturing execution system* (MES).
- **Nível 5:** é o nível responsável pela administração dos recursos da empresa, em que se encontram os softwares para gestão de vendas e gestão financeira; é também onde se realizam a decisão e o gerenciamento de todo o sistema. No nível 5 também encontra-se softwares do tipo *enterprise resources planing* (ERP).

2.2 CONTROLADORES LÓGICO PROGRAMÁVEIS - CLP

O CLP é um tipo de computador industrial que pode ser programado para executar funções de controle. Esses controladores reduziram o circuitos e ligações associadas aos circuitos de controle convencional a relé, além de apresentar outros benefícios, como a facilidade de programação e instalação, controle de alta velocidade, compatibilidade de rede, verificação de defeitos e conveniência de teste e alta confiabilidade (PETRUZELLA, 2014).

Nos dias atuais, um CLP é largamente utilizado em aplicações industriais, considerando sua elevada capacidade de processamento e de funcionamento em tempo real, sendo projetado para controlar múltiplas entradas e saídas e também para funcionar em ambientes hostis, pois suporta grandes variações de temperatura e tem imunidade a ruídos elétricos e resistência a vibração e a impacto (ROSÁRIO, 2012). Na Figura 3 é mostrado o rack com um CLP M340 (Modicon 340) e uma fonte de alimentação da marca *Schneider Eletric*. Esse será o dispositivo que será utilizado nesse trabalho.

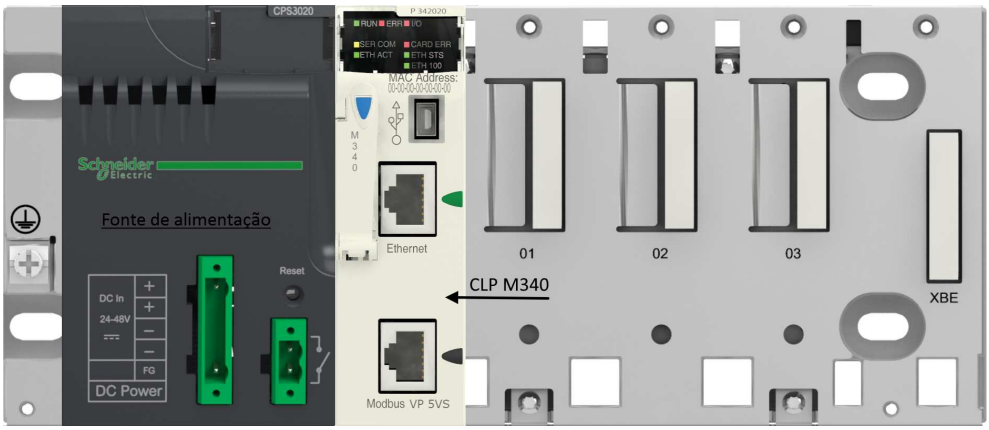
Na década de 60 e 70, o CLP era empregado com o propósito de substituir o relé lógico. Entretanto, devido à sua crescente diversidade de funcionalidades, o CLP agora é empregado em contextos variados e possui funções de rotinas avançada e comunicação. A estrutura do CLP é fundamentada em princípios semelhantes (i.e., contém CPU, RAM e memória de dados) arquitetônicos utilizados em computadores, o que confere a ele a

capacidade não somente de desempenhar as funções de um relé, mas também de abranger outras aplicações, como temporização, contagem, realização de cálculos, comparação e processamento de sinais analógicos.

Os CLPs caracterizam-se, segundo (MORAES; CASTRUCCI, 2007), por:

- robustez adequada aos ambientes industriais;
- programação por meio de computadores pessoais (PCs);
- linguagens amigáveis para o projetista de automação de eventos discretos;
- permitir tanto o controle lógico quanto o controle dinâmico (P+I+D);
- incluir modelos capazes de conexões em grandes redes de dados.

Figura 3 – Controlador Lógico Programável M340 e Fonte 24/48V *Schneider Eletric*.



Fonte: (Autor, 2023).

2.2.1 Modicon 340

O CLP M340, da fabricante Schneider Eletric, foi o dispositivo escolhido para o desenvolvimento do presente trabalho, possui um processador eficiente que permite a execução rápida de tarefas de controle, ideal para aplicações que exigem alta velocidade e precisão e possui uma estrutura de memória projetada para suportar uma ampla gama de aplicações industriais. O M340 suporta a expansão de memória através de cartões SD, permitindo aumentar a capacidade de armazenamento de programas e dados. Para os dois tipos de processador, a Tabela 1 mostra o tamanho máximo e o tamanho padrão dos dados.

Tipos de Objetos	**Endereço**	**Tamanho máximo para o processador BMX P34 1000**	**Tamanho padrão para o processador BMX P34 1000**	**Tamanho máximo para o processador BMX P34 20x0x**	**Tamanho padrão para o processador BMX P34 20x0x**
Bit interno	%Mi	16250	256	32634	512
Bits Entrada/Saída	%Ir.m.c %Qr.m.c	(1)	(1)	(1)	(1)
Sistema Bits	%Si	128	128	128	128
Words internas	%MWi	32464	512	32464	1024
Words constantes	%KWi	32760	128	32760	256
Sistema *words*	%SWi	168	168	168	168

Tabela 1 – Tamanho máximo e padrão dos dados localizado de acordo com o tipo de processador do CLP M340.

Tipos de Objetos	**Endereço**	**Tamanho máximo processadores BMX P34**	**Tamanho padrão processadores BMX P34**
Bits saída e bits interno	%M (0x)	65530	1504
Bits entrada e bits interno	%I (1x)	65530	1504
Words entrada e *Words* internas	%IW	65530	512
Words saída e *Words* internas	%MW (4x)	65530	512

Tabela 2 – Tamanho máximo e padrão dos dados localizado no caso de configuração de estado de RAM.

A Tabela 2, mostra o tamanho padrão e máximo dos dados localizados na RAM, e os tipos de objetos com seus respectivos endereços de memória. Por exemplo, em uma *array* dinâmica usa-se endereços de memória *word* %MW, a maior variação que pode ser definida nessa *array* é de tamanho [0...65530], ou seja, 65531 posições da *array* podem ser alocadas. Nesse trabalho será utilizado *arrays* em memória com o tamanho de 50, ou seja, menos que 1% da memória total.

2.3 PROGRAMAÇÃO DE CLP

A IEC *(International Electrotechnical Comission)*, é uma organização internacional de padronização que compreende todos os comitês eletrotécnicos nacionais. O objetivo da IEC é promover a cooperação internacional em todas as questões relativas à padronização nos campos elétricos e eletrônico.(IEC61131-3, 2013). A IEC 61131 é uma norma internacional que estabelece um padrão para controladores lógicos programáveis (CLPs) e seus dispositivos periféricos associados. Esta norma abrange diversos aspectos importantes dos CLPs, incluindo terminologia, requisitos de equipamento, testes e métodos de programação. Entra suas partes mais notáveis está a parte 3, que define cinco linguagens de programação padrão. A IEC 61131 visa garantir interoperabilidade entre dados e aplicações.

A IEC 61131-3 é a terceira parte da norma IEC 61131 que especifica a sintaxe e a semântica de duas linguagens textuais, Lista de Instruções (IL) e Texto Estruturado (ST), e duas linguagens gráficas, Diagrama Ladder (LD) e Diagrama de Blocos Funcionais (FBD) e o SFC (*Gráfico Sequencial de Funções*, que é colocado na norma como um método de programação e não propriamente uma linguagem, mas por simplicidade muitas vezes é apresentado como linguagem). O programa pode ser composto por módulos únicos ou múltiplos implementados em qualquer uma dessas linguagens, onde cada linguagem possui seu próprio editor dedicado. (RAMANATHAN, 2014).

Antes do surgimento da norma IEC 61131-3, diversas linguagens de programação eram utilizadas para programar CLPs, cada uma seguindo suas próprias normas, como NEMA, GRAFCET e DIN 40719, entre outras. Essas normas variavam de acordo com diferentes países, resultando em uma falta de padronização na programação dos CLPs. Com isso existiam diferenças relativamente grandes entre CLPs de vários fabricantes. Isto era especialmente verdadeiro em relação à escolha da linguagem de programação e ao design da linguagem implementada nos CLPs. A padronização é um item de *benchmark* entre diferentes modelos de CLP. Por causa disso, é comercialmente interessante para alguns fabricantes aderirem ao padrão. Isso tornou mais fácil a transição de um fabricante de CLP para outro, bem como, em certa medida, ajudou os clientes a entender melhor o que estão adquirindo (HANSSEN, 2015).

2.3.1 Software de programação de CLP

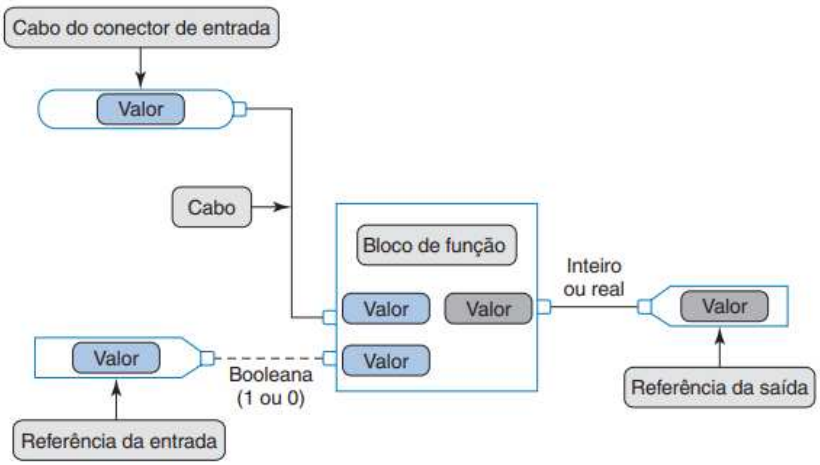
É necessário estabelecer um modo para o software de um computador pessoal (CP) comunicar-se com o controlador lógico programado (CLP); conexão conhecida como configuração de comunicações. O método usado para configurar as comunicações varia para cada modelo de controlador (PETRUZELLA, 2014). Os softwares de programação variam de fabricante para fabricante, e são ferramentas para o desenvolvimento e manutenção de sistemas automatizados, pois permite que programas e projetos sejam criados, simulados, depurados e implementados. O software usado para o desenvolvimento deste trabalho

foi o EcoStruxure^{TM} Control Expert, que é a plataforma de desenvolvimento do CLP M340. O Control Expert fornece, segundo (SCHNEIDER, 2023), as seguintes linguagens de programação para criar o programa do usuário: Diagrama de blocos funcionais (FBD), diagrama ladder (LD), lista de instruções (IL), texto estruturado (ST), Gráfico de função sequencial (SFC). Todas essas linguagens de programação podem ser usadas juntas no mesmo projeto e estão em conformidade com a IEC 61131-3.

2.3.2 Linguagem Diagrama de Blocos (*Function Block Diagram, FBD)*

A linguagem de programação em diagrama de blocos, derivada da eletrônica digital, representa uma abordagem gráfica de programação que facilita ver o fluxo do sinal. É utilizado símbolos gráficos que representam operações lógicas, entradas e saídas, a fim de criar uma representação visual da lógica de controle. Essa abordagem gráfica leva vantagem na facilidade da compreensão e criação de sistemas de automação complexos. Na Figura 4 pode ser vista a estrutura de um programa de blocos de função. Um diagrama de blocos de função consiste em quatro elementos básicos: bloco de função, referência, conectores dos fios e fios (PETRUZELLA, 2014).

Figura 4 – Exemplo de estrutura com diagrama de blocos de função.



Fonte: (PETRUZELLA, 2014).

2.3.3 Linguagem Ladder

O Diagrama Ladder é a linguagem mais utilizada para programação de CLP. É uma linguagem gráfica onde os elementos básicos são baseados em uma analogia com diagramas de relés físicos, portanto, não representa uma grande mudança de paradigma para os técnicos que não estão habituados às novas técnicas informáticas e linguagens de programação (BENDER *et al.*, 2008). A Tabela 3 mostra os componentes que constituem a linguagem *ladder* e o quanto eles se assemelham com os elementos de contatores elétricos.

Instrução	**Representação**
Contato normalmente aberto - NA	-| |-
Contato normalmente fechado - NF	-|/|-
Bobina	-( )-
Bobina inversa (acionada, desenergiza)	-(I)-
Bobina set	-(S)-
Bobina reset	-(R)-

Tabela 3 – Instruções para diagrama ladder.

Segundo (MORAES; CASTRUCCI, 2007), graficamente, as regras que constituem os elementos básicos-bobinas, contatos e linhas - são:

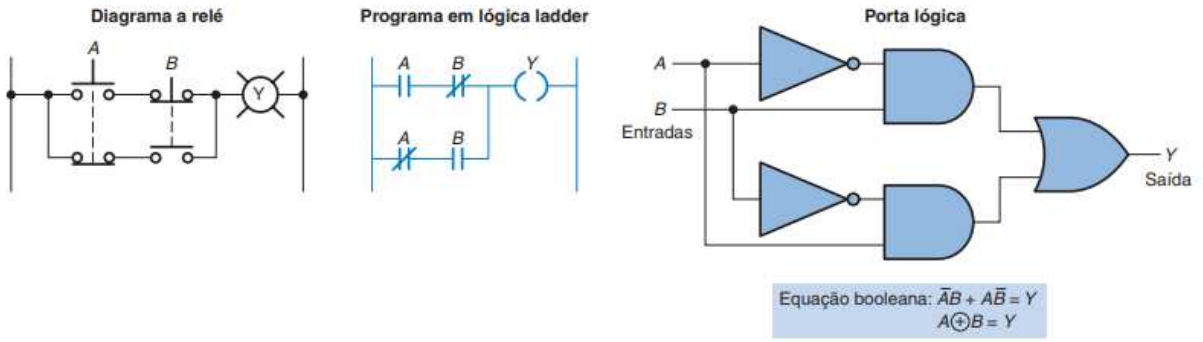
- bobinas sempre ficam totalmente à direita das linhas horizontais;
- linhas verticais são denominadas linhas-mãe;
- das linhas verticais partem linhas horizontais que podem ligar-se a mais linhas verticais, e assim por diante;
- as sequências de causa e efeito orientam-se da esquerda para a direita e de cima pra baixo;
- a habilitação das linhas horizontais, da qual decorre o acionamento das bobinas, depende da afirmação dos contatos à sua esquerda.

Na Figura 5 mostra-se a comparação e relação entre o diagrama lógico ladder a relé, o programa em lógica ladder e o circuito equivalente com porta lógica. No exemplo a saída Y é ligada quando o botão de comando A ou B for pressionado, mas não simultaneamente. Este circuito pode ser configurado utilizando apenas os contatos normalmente abertos do botão de comando como entrada para o programa. Através da Figura 5 pode-se notar os contatos representando entrada e saída do programa com algumas condições, ficando responsável por acionar ou controlar algum processo ou dispositivo. As três lógicas tem a mesma finalidade e explicando na forma booleana, quando A for 1 e B 0 e A for 0 e B 1, Y será 0.

2.3.4 Linguagem Texto Estruturado (*Structured Text*, ST)

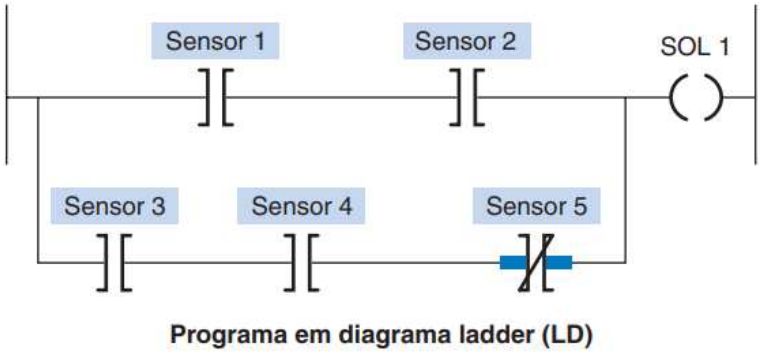
O texto estruturado é uma linguagem de alto nível semelhante à linguagem Pascal da norma ISO 7185. Com essa linguagem, é possível chamar blocos funcionais, funções e atribuições e executar condicionalmente instruções e tarefas de repetição. Sua flexibilidade é de fácil entendimento, proporcionando a programadores e desenvolvedores de softwares a interpretação do programa do processo industrial (a linguagem permite que os profissionais entendam como o programa funciona e como ele gerencia e automatiza as operações dentro de um ambiente industrial). Desenvolvida para controle industrial, a linguagem de programação ST trabalha com “expressões”, constituídas de operadores e operandos que retornam um valor quando executados (DA SILVA, 2021).

Figura 5 – Lógica ladder em comparação com diagrama a relé e porta lógica.



Fonte: (PETRUZELLA, 2014).

Figura 6 – Programa para CLP em ladder e texto estruturado equivalente.




```
IF Sensor_1 AND Sensor_2 THEN
        SOL_1 := 1;
ELSEIF Sensor_3 AND Sensor_4 AND NOT Sensor_5 THEN
        SOL_1 := 1;
END_IF;
```

Programa em texto estruturado

Fonte: (PETRUZELLA, 2014).

A Figura 6 mostra como o texto estruturado e a programação com diagrama ladder podem ser utilizados para produzir a mesma saída lógica, aplicação que tem o objetivo de energizar um solenoide (SOL) 1 sempre que existir uma das duas seguintes condições do circuito:

- As chaves sensor 1 e sensor 2 estiverem fechadas.
- As chaves sensor 3 e sensor 4 estiverem fechadas e a chave sensor 5 estiver aberta.

Essa linguagem foi desenvolvida para permitir que programadores escrevam algorit-

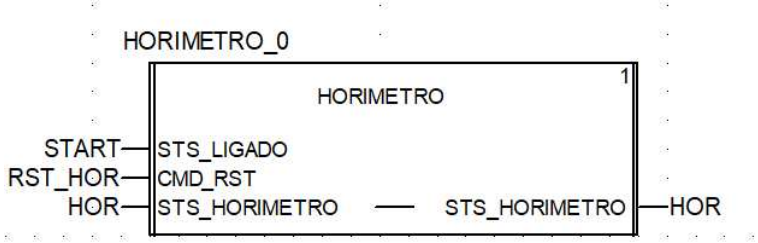
mos de controle complexos de maneira clara e estruturada e também na programação de blocos funcionais e funções, com o objetivo de ocultar os detalhes de sua implementação interna aos usuários do programa.

2.3.5 Blocos Funcionais (FB)

O conceito de bloco funcional é uma das partes possíveis na arquitetura de software disposta na IEC 61131-3. Ele permite a programação hierárquica e estruturada do projeto do *software*. Para efeitos de linguagem de programação um bloco funcional é uma unidade de organização de programa que podem armazenar informações, produzir um ou mais valores, e é possível utilizar em múltiplas instâncias/partes do programa. Os blocos de função podem ser utilizados para a criação de elementos de *software* reutilizáveis. Esses blocos possuem um conjunto de dados, os quais podem ser alterados por um algoritmo interno. Somente o conjunto de dados é mantido na memória para determinada instância do bloco de função. Os estados internos dos dados são mantidos entre uma execução e outra. Os blocos também podem ser utilizados para a criação de outros blocos funcionais, aumentando a capacidade de reutilização do *software*. São elementos dos blocos funcionais: variáveis de entrada, variáveis de saída e variáveis internas. A reutilização dos blocos, geralmente empregados em diversas aplicações, facilita o trabalho dos programadores, pois eles não precisarão reprogramar a lógica do processo novamente; eles só precisam buscar a pasta com o bloco funcional.(DA SILVA, 2021).

Na Figura 7 é mostrado um bloco funcional de um horímetro que pode ser reutilizado dentro do software. É ligado junto ao bloco a variável global START, que é a mesma usada para acionar o programa final, fazendo com que o horímetro funcione enquanto o programa estiver em modo RUN. A variável CMD_RST é usada quando houver a necessidade de resetar o horímetro, através da variável global RST_HOR, e a variável STS_HORIMETRO indica o tempo que está em contagem. O bloco foi desenvolvido em linguagem ladder pelo autor, e pode ser visto o desenvolvimento na seção Apêndice deste documento.

Figura 7 – Bloco funcional de um horímetro.



Fonte: (Autor, 2023).

2.3.6 Ciclo de Varredura (*Scan*) do CLP

Um CLP é um exemplo de um sistema em tempo real, uma vez que os resultados de saída devem ser produzidos em resposta a condições de entrada dentro de um tempo limitado, caso contrário, ocorrerá uma operação não intencional. O comportamento em tempo real de um CLP geralmente está associado ao seu tempo de varredura, que é o tempo que leva para ler todas as entradas, executar o programa lógico e escrever todas as saídas de volta (ALVES; MORRIS, 2018)

Ao realizar a execução de um programa, o CLP precisa ter a capacidade de monitorar qualquer alteração que ocorra em um dispositivo externo sob seu controle. Em cada ciclo de operação, o processador do CLP examina todas as entradas, adquire os valores correspondentes e ativa ou desativa as saídas de acordo com as instruções programadas pelo usuário. Esse procedimento é conhecido como ciclo de varredura do programa e é fundamental para o funcionamento em tempo real do controlador. Durante um clico, mostrado na Figura 8, o CLP primeiro realiza a atualização dos valores, em seguida processa as instruções do programa, varrendo cada degrau da programação, e por último faz a varredura da saída.

Figura 8 – Ciclo de processamento (scan).

Fonte: (MORAES; CASTRUCCI, 2007).

Pelo fato de uma entrada poder mudar a qualquer momento, repete-se esse ciclo constantemente enquanto o CLP estiver no modo de funcionamento (RUN). O tempo necessário para completar um ciclo de varredura é chamado de *tempo de ciclo de varredura* e indica a rapidez de reação do controlador às mudanças nas entradas; ele pode variar de 1 a 20 milissegundos (PETRUZELLA, 2014).

O CLP usado nesse projeto (M340 Schneider Eletric) usa, segundo *data sheet* da Scheneider Eletric, a varredura vertical pelo método de coluna, no qual o processador

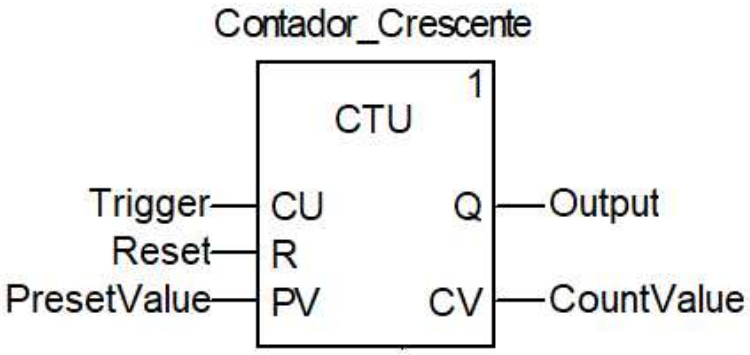
examina as instruções de entrada e de saída a partir da entrada do comando em cima, à esquerda, no diagrama ladder, verticalmente, coluna por coluna e página por página. As páginas são executadas em sequência. Outro padrão básico de varredura que alguns CLPs usam é o de maneira horizontal, conhecido como método degrau, no qual o processador examina as instruções de entradas e de saídas a partir do primeiro comando, na parte superior esquerda, horizontalmente, degrau por degrau.

2.3.7 Contadores

Contadores são usados para contar eventos, pulsos, ciclos, ou qualquer outra coisa que possa ser quantificada em um processo industrial. Eles são usados para controlar sequências de operações, como a contagem de peças em uma linha de produção ou o monitoramento do tempo decorrido em um processo.

Os contadores do CLP normalmente são retentivos, isto é, qualquer que seja o número contido na contagem no momento que o processo é desligado, ele será restaurado para o contador quando este for novamente energizado. Eles podem ser projetados para contar de modo crescente ou decrescente até o valor pré-ajustado. O contador crescente é incrementado de 1 cada vez que o degrau que contém o contador é energizado; o contador decrescente é decrementado de 1 cada vez que o degrau que contém o contador é energizado. Essas transições no degrau podem resultar de algum evento ocorrido no programa; por exemplo, peças que passam por um sensor ou pelo acionamento de uma chave-limite. O valor pré-ajustado de um controlador programável pode ser estabelecido pelo operador ou pode ser carregado na posição de memória como um resultado de uma decisão do programa (PETRUZELLA, 2014).

Figura 9 – Bloco de função do tipo contador incremental (CTU).



Fonte: Adaptado da (IEC61131-3, 2013).

Na Figura 9 pode-se ver um contador do tipo CTU (*Counter UP*), onde a cada pulso (*trigger*) incrementa 1 ao *CountValue*, onde a saída (*Output*) só será verdadeira quando o valor pré setado for igual ao valor contado (PV = CV). Existem também contadores CTD

(*Counter Down*), que decrementa, a cada pulso, 1 ao *CountValue*, e o CTUD (*Counter Up and Down*), que tem as duas funções citadas no mesmo contador.

2.3.8 Sistema de Words e Bits - CLP M340 Schneider Eletric

O Modicon M340, Modicon M580, Premium, Atrium, Quantum e Momentum PLCs utilizam bits do sistema %Si que indicam o estado do CLP, ou podem ser usados para controlar como ele opera. Esses bits podem ser testados no programa do usuário para detectar qualquer desenvolvimento funcional que exija um procedimento de processamento definido. Alguns desses bits devem ser redefinidos para seu estado inicial ou normal pelo programa. No entanto, os bits do sistema que são redefinidos para seu estado inicial ou normal pelo sistema não devem ser redefinidos pelo programa ou pelo terminal (ELECTRIC, 2020).

Figura 10 – Exemplos de sistemas de bits e suas funções.

Bit `Symbol`				
%S11 `WDG`	Function	Watchdog overflow		
	Initial State	0		
	Platforms	**M340:** Yes **M580:** Yes **M580 Safety:** Yes	**Quantum:** Yes **Momentum:** Yes	**Premium:** Yes **Atrium:** Yes
	Normally at 0, this is set to 1 by the system as soon as the task execution time becomes greater than the maximum execution time (i.e. the watchdog) declared in the task properties. **NOTE:** On M580 Safety, this bit takes into account an overrun on SAFE task.			
%S12 `PLCRUNNING`	Function	PLC in RUN		
	Initial State	0		
	Platforms	**M340:** Yes **M580:** Yes **M580 Safety:** Yes	**Quantum:** Yes **Momentum:** Yes	**Premium:** Yes **Atrium:** Yes
	This bit is set to 1 by the system when the PLC is in RUN. It is set to 0 by the system as soon as the PLC is no longer in RUN (STOP, INIT, etc.).			
%S13 `1RSTSCANRUN`	Function	First cycle after switching to RUN		
	Initial State	–		
	Platforms	**M340:** Yes **M580:** Yes **M580 Safety:** Yes	**Quantum:** Yes **Momentum:** Yes	**Premium:** Yes **Atrium:** Yes
	Switching the PLC from STOP mode to RUN mode (including after a cold start with automatic start in run) is indicated by setting system bit %S13 to 1. This bit is reset to 0 at the end of the first cycle of the MAST task in RUN mode.			

Fonte: (ELECTRIC, 2020).

Na Figura 10 é mostrado algumas funções que podem ser ativadas na programação do CLP. Por exemplo o bit %S11 (*Watchdog overflow*) é acionado pelo sistema assim que o tempo de execução da tarefa se torna maior que o tempo máximo de execução (ou seja, o watchdog) declarado na tarefa propriedades. O Quadro 1, ver ANEXO, mostra a primeira página do documento fornecido pela *Schneider Eletric* que contém o sistema de bits usados para o desenvolvimento deste projeto (%S6 e %S13). Para o bit %S6 (TB1SEC), o documento descreve que atua como um temporizador interno que regula a mudança de status deste bit e é assíncrono em relação ao ciclo do CLP. O bit %S13 (1RSTSCANRUN), serve como primeiro bit de digitalização, normalmente definido como 0, é definido como 1 pelo sistema durante o primeiro ciclo da tarefa mestre após o CLP ser colocado em RUN. Todos esses bits discutidos serão utilizados no trabalho.

2.3.9 Manipulação de Dados

As instruções de manipulação de dados permitem que dados numéricos armazenados na memória do controlador sejam operados dentro do programa de controle. O seu uso amplia a capacidade de um controlador desde um simples controle liga/desliga baseado em lógica binária, até uma tomada de decisão quantitativa envolvendo comparações de dados, aritmética e conversões – que, por sua vez, podem ser aplicados aos controles de posicionamento e analógicos. Existem dois tipos básicos de classes de instruções para realizar uma manipulação de dados: instruções que operam palavra de dados e as que operam arquivo, bloco ou dados que envolvem palavras múltiplas. Cada instrução de manipulação de dados requer mais duas palavras da memória de dados para operação, que, na forma singular, podem ser referidas como registro ou como palavra, de acordo com o fabricante. Os termos tabela ou arquivo são utilizados geralmente quando um grupo consecutivo de memória de palavras de dados relacionados são referenciados (PETRUZELLA, 2014).

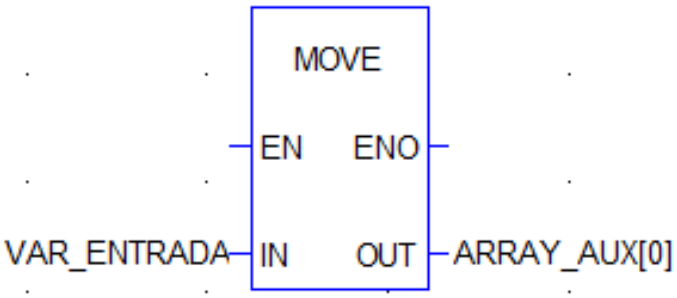
As instruções de manipulação de dados permitem o movimento, manipulação ou armazenagem de dados, podendo ser em um grupo único ou em grupos múltiplos de uma área de memória de dados de um CLP para outro. O uso dessas instruções no CLP em aplicações que requerem uma geração ou manipulação de uma grande quantidade de dados reduzem muito a complexidade e a quantidade requerida de programação. A manipulação de dados pode ser feita em duas grandes categorias: transferência de dados e comparação de dados.

Para a transferência de dados, pode-se usar blocos de funções como o bloco MOVE, a fim de atribuir o valor de entrada à saída. Os tipos de dados dos valores de entrada e saída devem ser idênticos, mesmo o bloco aceitando todos os tipos de dados.

Na Figura 11 tem-se o bloco Move onde, ao incluir uma variável ou apenas um valor na entrada (IN), a saída (OUT) será atribuída a esse valor, podendo ser uma variável normal, uma *array* ou o que o programador estiver necessitando alocar a essa entrada. No exemplo usado na Figura 11 pode-se notar a transferência de uma variável, para uma

posição da *array* declarada. Nesse caso o valor atribuído a VAR_ENTRADA será movido para a posição 0 da *array* no momento que EN for verdadeiro.

Figura 11 – Exemplo de instrução para transferir uma variável para uma posição de *array*.



Fonte: (Autor, 2023).

2.4 GERADOR DE SINAL ANALÓGICO

O gerador de sinal de tensão e corrente de ±12V/0-24mA é o instrumento de teste elétrico comumente utilizado durante a instalação, ajuste, revisão e manutenção de equipamentos elétricos no local. O gerador possui controles que permitem ao usuário ajustar a amplitude (a "altura" do sinal), a frequência (quantas vezes o sinal se repete por segundo) e a forma da onda. Por exemplo, ao medir a temperatura de um tanque de água, um gerador de sinal pode simular a saída de um sensor de temperatura variando sua tensão de 0 a 10V, correspondente a uma faixa de temperatura predeterminada. O gerador de sinal tem suas limitações quando o assunto é ruído e distorção, porque em alguns casos, sinais analógicos podem introduzir ruído ou distorção, afetando a precisão dos testes, e também sobre a calibração, onde precisam ser calibrados regularmente para garantir que os sinais gerados sejam precisos.

2.5 REDES INDUSTRIAIS

Uma rede é uma combinação de hardware e software que envia dados de um local para outro. O hardware consiste no equipamento físico que transporta sinais de um ponto da rede para outro. O software consiste em conjuntos de instruções que fazem possível os serviços que esperamos de uma rede (FOROUZAN, 2009).

Redes Industriais referem-se a sistemas de comunicação projetados para a automação e controle de processos industriais. Essas redes são usadas em ambientes de fabricação e automação para interconectar dispositivos e máquinas, permitindo que eles compartilhem informações e coordenem suas operações. As redes industriais são projetadas para serem confiáveis, robustas e capazes de suportar ambientes industriais desafiadores, onde há altos níveis de ruído elétrico, temperaturas extremas e outras condições adversas.

Essas redes desempenham um papel fundamental na indústria moderna, facilitando a automação de processos, melhorando a eficiência, reduzindo os custos de produção e permitindo a coleta de dados para análise e tomada de decisões. Elas podem ser usadas em uma ampla variedade de aplicações industriais, incluindo manufatura, controle de processos químicos, automação de fábricas, monitoramento de equipamentos, entre outros. Redes industriais geralmente empregam protocolos de comunicação específicos, como o Modbus, Profibus, Ethernet, CAN, Profinet, e muitos outros, dependendo das necessidades e requisitos da aplicação específica.

2.5.1 Protocolo Modbus

O Modbus é um protocolo de comunicação que permite uma fácil comunicação dentro de todos os tipos de arquitetura de rede (MODBUS, 2012). Ele foi desenvolvido pela empresa Modicon em 1979 e tem sido uma escolha popular para a comunicação entre dispositivos em sistemas de controle industrial, devido à sua simplicidade e confiabilidade.

O Modbus está posicionado, segundo (MODBUS, 2012), na camada 7 do modelo OSI, onde fornece comunicação cliente/servidor entre dispositivos conectados em diferentes tipos de barramentos ou redes. Atualmente, ele é implementado utilizando:

- TCP/IP sobre Ethernet;
- Transmissão assíncrona serial por meio de diversos tipos de mídia (cabo: EIA/TIA-232-E, EIA-422, EIA/TIA-485-A; fibra, rádio, etc.);
- Modbus PLUS, uma rede de passagem de token de alta velocidade.

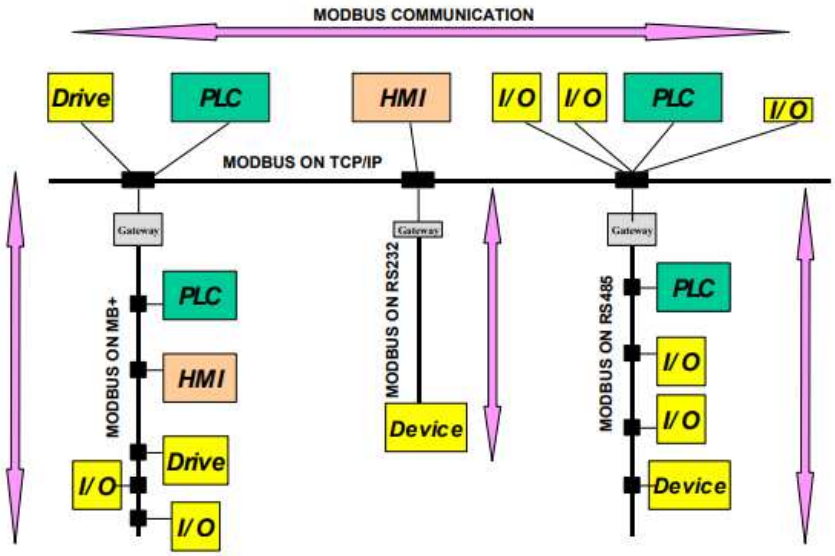
O protocolo Modbus suporta, principalmente, dois tipos de mensagens: mensagens de leitura (*read*) e mensagens de escrita (*write*). As mensagens de leitura são usadas para obter informações de um dispositivo, enquanto as mensagens de escrita são usadas para enviar comandos ou dados para um dispositivo. O Modbus pode ser usado tanto em comunicações serial (como RS-232 ou RS-485) quanto em comunicações Ethernet (como o Modbus TCP/IP). Os dispositivos Modbus são endereçados por números que variam de 1 a 247 no Modbus RTU. No Modbus TCP/IP, o endereçamento é feito usando endereços IP e portas.

As mensagens Modbus são formatadas em pacotes de bytes que incluem informações como o código da função (leitura ou escrita), endereço do dispositivo, registro de início, número de registros e dados. A estrutura do pacote varia entre as variantes do protocolo. O protocolo Modbus é utilizado para a comunicação entre dispositivos como sensores, controladores e atuadores. Entre suas funções principais, a leitura de registros de entrada permite coletar dados de sensores e entradas analógicas, fornecendo informações sobre o ambiente ou processo monitorado. A leitura de registros de bobina é utilizada para verificar o estado de saídas digitais, como relés e interruptores, ajudando monitorar o status de dispositivos binários. Além das funções de leitura, o Modbus também oferece a leitura de

registros de holding, que são registros que podem ser lidos e escritos, armazenando dados de configuração e estados operacionais dos dispositivos. Isso permite acessar informações detalhadas sobre os dispositivos e fazer ajustes conforme necessário. Por exemplo, os registros de holding podem armazenar valores de setpoints, parâmetros de controle ou outras informações importantes para a operação do sistema.

O protocolo também suporta funções de escrita, incluindo a escrita em registros de bobina, que permite controlar dispositivos digitais ao ligar ou desligar saídas, e a escrita em registros de holding, usada para atualizar configurações e estados de operação dos dispositivos. Na comunicação Modbus, normalmente, um dispositivo age como o mestre (master) que inicia as solicitações e outros dispositivos são escravos (slaves) que respondem às solicitações do mestre.

Figura 12 – Exemplo da Arquitetura de Rede MODBUS.



Fonte: (MODBUS, 2012).

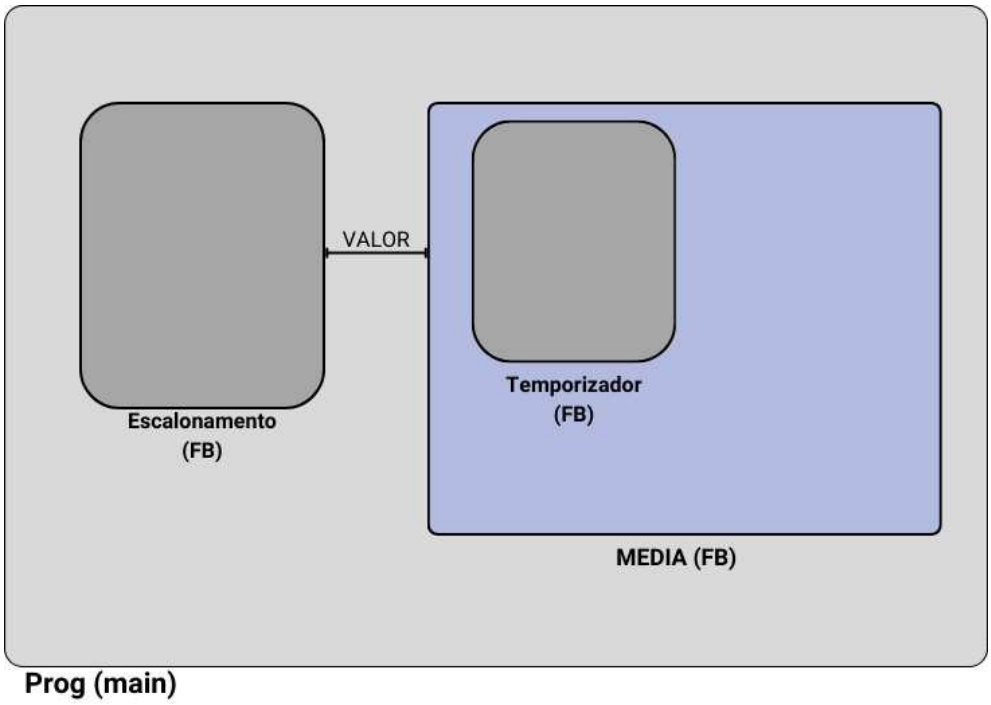
A Figura 12 mostra como o protocolo MODBUS permite uma comunicação fácil dentro de todos os tipos de arquiteturas de rede. Todos os tipos de dispositivos (CLP, IHM, Painel de Controle, Driver, Controle de Movimento, Dispositivo de E/S) podem utilizar o protocolo MODBUS para iniciar uma operação remota. A mesma comunicação pode ser feita tanto em uma linha serial quanto em redes Ethernet TCP/IP. Os gateways permitem a comunicação entre vários tipos de barramentos ou redes utilizando o protocolo MODBUS.

3 METODOLOGIA

Neste capítulo, é abordado o desenvolvimento do trabalho e os passos realizados para a criação dos blocos de função. Foi utilizado o CLP M340, programado no software EcoStruxure^{TM} Control Expert (antigo Unity Pro), que é um software de programação desenvolvido pela Schneider Electric para a criação e gerenciamento de aplicações em CLPs. E também um gerador de sinal com saídas de 0-10V e 4-20mA para o desenvolvimento do projeto. A escolha pelo M340 foi pelo fato do CLP estar disponível junto a empresa onde o trabalho foi desenvolvido.

Serão criados três blocos de função. O primeiro bloco é o do temporizador, uma vez que foi desenvolvido usando o bloco funcional do tipo contador incremental, adicionando um bit de temporizador na sua entrada CU, cujo a base de tempo é predefinida de acordo com o sistema de *bits* e *words* do CLP M340 fornecido pela Schneider Eletric. O bloco do temporizador será usado dentro do bloco de função da Média, onde a cada leitura realizada pelo sensor de acordo com o tempo desejado, enviará o valor atual do sensor para a *array* dinâmica, onde a mesma dará início aos cálculos. O último bloco criado será o bloco de escalonamento, que é uma equação desenvolvida e programada em Texto Estruturado, para ajustar os valores recebidos do gerador de sinal, escalonando de acordo com o tipo de sinal de saída do gerador. A Figura 13 mostra a arquitetura planejada para o projeto e a colocação dos blocos funcionais e suas utilidades.

Figura 13 – Arquitetura de software.



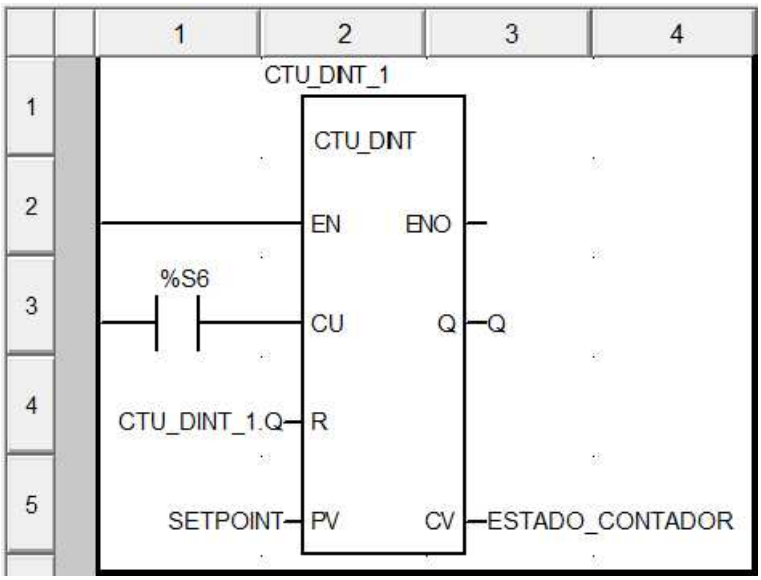
Fonte: (Autor, 2023).

3.1 ESTRUTURA DIVISORA DE CLOCK A PARTIR DE UM CONTADOR

O primeiro passo na programação do projeto foi transformar o bloco de função de contagem crescente em um temporizador. Optou-se por desenvolver usando o bloco CTU ao invés de um temporizador diretamente fornecido pelo software (TON, TOF, TP), pois a fabricante disponibiliza um sistema de *bits* e *words*, onde usou-se o bit %S6, que é tratado nos manuais da Schneider Electric como um temporizador interno que regula a mudança de status deste bit e é assíncrono e em relação ao ciclo do CLP, ou seja, não corre junto a cada scan do equipamento, evitando o delay visto em 2.3.6. A sua base de tempo é de 1 segundo e foi alocado na entrada CU, que é a entrada de disparo do contador.

As entradas esperada do bloco temporizador são a de SETPOINT, onde a variável global SET_POINT é a responsável por definir o tempo que a saída Q será ativada; e uma variável para resetar o bloco. Já para as saídas do bloco, espera-se uma variável que mostre o valor da contagem atual do bloco, e uma variável booleana Q, uma vez que CV contenha o valor predefinido em PV, ative a saída do bloco para dar sequência na programação.

Figura 14 – Contador Crescente com o bit de acionamento a cada segundo.



Fonte: (Autor, 2023).

O valor de PV (*preset value*) é pré definido antes de começar a contagem. Foi criado a variável SETPOINT para receber qualquer valor inteiro. A saída **Q** aciona-se somente quando o CV (Count Value) estiver com o mesmo valor do PV (foi criado a variável ESTADO_CONTADOR para representar o CV). Como o bit %S6 é acionado a cada 1 segundo, o CV acrescenta 1 a cada vez que CU é acionado, ou seja, a cada 1 segundo. Por exemplo, se for solicitado uma leitura a cada 1 minuto, o SETPOINT é pré definido 60 que, após CU ser ativado 60 vezes (nesse caso, uma vez por segundo), o ESTADO_CONTADOR iguala ao valor de SETPOINT, acionando a saída **Q** e resetando o bloco para uma nova contagem.

3.2 ALOCAÇÃO EM *ARRAY*S DINÂMICAS

Para armazenar os dados lidos de fontes externas para dentro do bloco de leitura, foi utilizado *array*s dinâmicas uma vez que, fossem um conjunto de elementos do mesmo tipo de dado (por exemplo, do tipo real ou do tipo inteira), e com a flexibilidade de seu tamanho ser ajustado durante a execução do programa, em oposição a uma *array* estática, cujo tamanho é fixo durante a compilação.

3.2.1 Tamanho da *array*

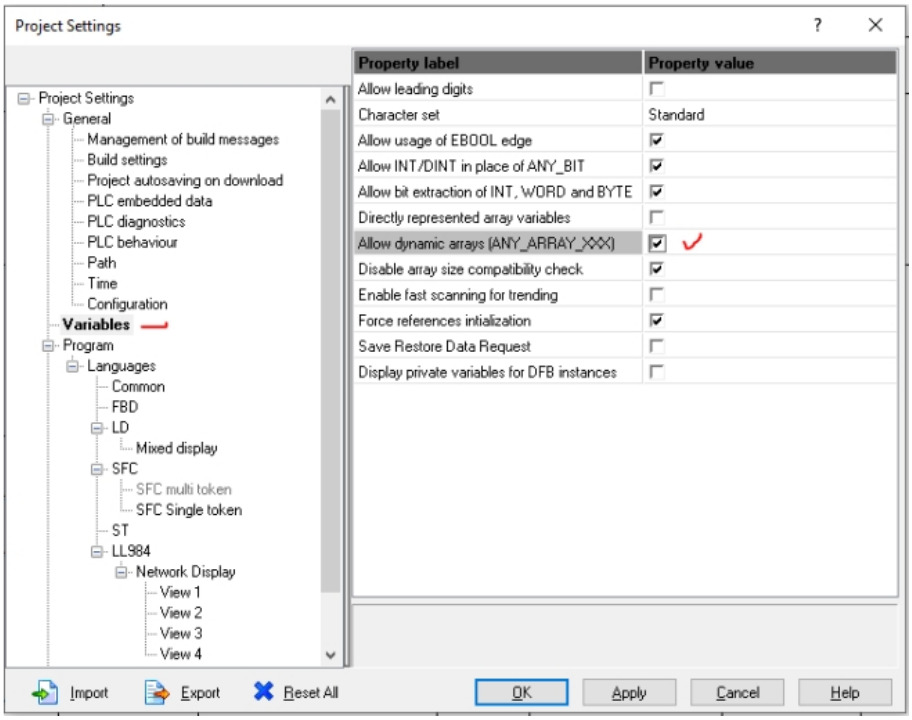
O primeiro passo para trabalhar com *array* dinâmica na programação do M340, no *Control Expert*, é habilitar o seu uso. Para isso, no software, como visto na Figura 15, a habilitação se encontra em Tools, Project Settings, Variable e sinalizar o property label escrito: *Allow dynamic arrays* (ANY_ARRAY_XXX).

Após habilitado, a *array* local é criada dentro do bloco. Por conta de valores em ponto flutuante que um sensor pode fornecer, foi escolhida uma *array* do tipo real, com a seguinte sintaxe: ANY_ARRAY_REAL. Ao criar a *array* dinâmica, é escolhido de qual tipo ela vai ser, e também quantas alocações ela vai permitir definindo o seu intervalo. Como visto na Tabela 2, o tamanho máximo do endereço da *array* é de 65530, onde pode ser definida com um intervalo de [0...65530], gerando 65531 posições para serem preenchidas. Porém, ao alocar uma *array* como variável global na entrada da *array* dinâmica dentro do bloco, e definindo o seu intervalo de posições, implicará em quantos espaços serão usados na *array* principal. Uma vez que a variável global contenha um intervalo de [0...99], ou seja, 100 posições, a *array* dinâmica usará somente as 100 primeiros posições de todas as disponíveis.

3.3 BLOCO FUNCIONAL MÉDIA

Para tratar e analisar os dados, foi criado, na seção Derived FB Types do *Control Expert*, um bloco, programado em *Ladder*, que recebe o dado lido, aloca-o em uma posição da *array*, e assim que obter o número de dados solicitados, faz a média dos valores,

Figura 15 – Localização no software para habilitar *arrays* Dinâmicas.



Fonte: (Autor, 2023).

alocando essa média em outra *array*, enquanto reseta a primeira *array* para receber novos valores. O intuito é usar o bloco pronto na parte geral do software, para ligar suas variáveis internas com as variáveis globais. A seguir serão discutidos as linhas e instruções Ladder individuais. Observa-se que a coluna da esquerda contém os números da sequência da programação que compõe o código final.

Para a entrada do bloco Média, espera-se que contenha uma variável booleana para dar o início do programa, alocada na variável interna EN; é necessário uma variável inteira para definir o tempo para cada leitura; uma variável do tipo inteira que será predefinida com a quantidade de alocações permitidas na *array* que será usada para realizar os cálculos; uma variável do tipo real para receber o valor do sensor e dar continuidade no código de programação do bloco e uma variável do tipo inteira que será predefinida com a quantidade de leituras armazenadas na *array* dinâmica.

Para a saída do bloco Média, espera-se que contenha uma *array* para armazenar os valores que serão realizado a média; uma variável do tipo real que mostrará a última média realizada e uma *array* dinâmica para alocação das médias.

A Figura 16 mostra todas as variáveis declaradas e usadas na programação do bloco Média e o tipo de cada uma. É incluso também o bloco do temporizador, a fim de obter um tempo independente da variação do tempo de varredura do CLP, foi usado a estrutura, visto em 3.1.

Figura 16 – Variáveis declaradas para o bloco Média.

Name	n...	Type	Value	Comment
MEDIA		<DFB>		
<inputs>				
SET_POINT	1	DINT		Tempo em segundos
VAR_LENGTH	3	INT		Quantidade de posições que a Array vai ter para ir alocando os valores
VAR_ENTRADA	5	REAL		Recebe o valor do sensor
DIV_MEDIA_INT	7	INT		
<outputs>				
ARRAY_AUX	1	ANY_ARRAY_REAL		Armazena os valores que serão feitos a média. Resetada após ciclo.
MEDIA_ARR	3	REAL		
MEDIA_TABELA	5	ARRAY[0..100] OF REAL		Tabela da Média
<inputs/outputs>				
<public>				
<private>				
TEMPORIZADOR_0		TEMPORIZADOR		
RESET		EBOOL		
SAIDA		EBOOL		
STATUS_CTU		DINT		
MEM_AUX		EBOOL		
ARRAY_LENGTH		REAL		
VALOR_ARRAY		REAL		
ACIONA3		EBOOL		
DIV_MEDIA		REAL		

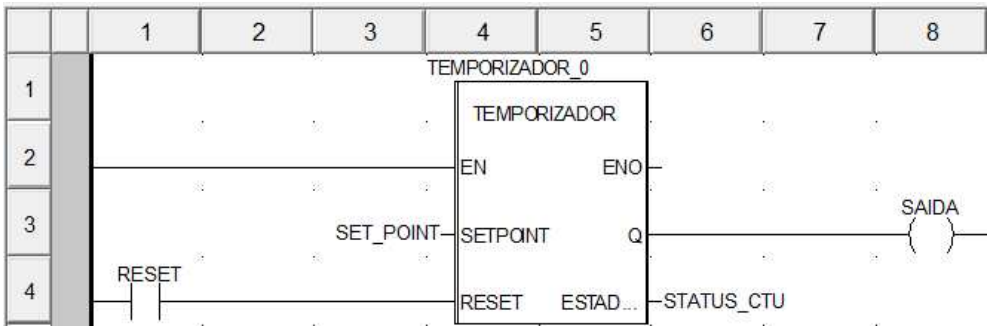
Fonte: (Autor, 2023).

As variáveis declaradas, Figura 16, para a entrada (*input*) do bloco Média são: a variável SET_POINT, do tipo double int, e recebe o valor do tempo, em segundos, de cada leitura; a variável VAR_LENGTH, do tipo inteira, que define quantas posições da *array* MEDIA_TABELA será preenchida com os valores lidos; a variável VAR_ENTRADA, do tipo real, e vai receber, através de uma variável global, o valor do sensor; a variável DIV_MEDIA_INT, do tipo inteira, é a parte que será definida a quantidade de leituras armazenadas na *array* dinâmica, o que não precisa ser o mesmo tamanho da *array*. Por exemplo, a *array* dinâmica pode ter 500 posições mas apenas 120 posições serão usadas para por os dados, que será definido pela variável DIV_MEDIA_INT.

Na saída (*output*) do bloco Média, tem-se a variável ARRAY_AUX, uma *array* dinâmica, que armazena os valores para realização do cálculo da média. A variável MEDIA_ARR, do tipo real, que será o valor da média após o cálculo, e a *array* dinâmica MEDIA_TABELA, que contém as médias na qual a programação para o cálculo é realizada.

Na parte *private*, onde são declarada as variáveis do bloco Média, contém todas as demais variáveis que contribuem na lógica de programação, mas que não precisam estar na entrada ou na saída do bloco e dependendo de uma variável global.

Figura 17 – Bloco temporizador dentro da lógica do bloco Média.

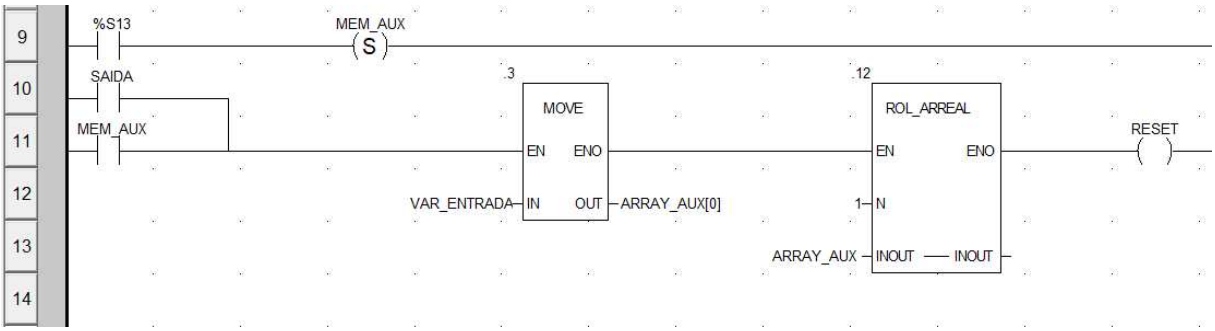


Fonte: (Autor, 2023).

O bloco temporizador, visto a sua programação na Figura 14, foi implementado no começo do programa que calcula a média, Figura 17, pois a cada acionamento da sua saída, o restante do programa será lido e realizará suas funções. A variável SET_POINT recebe o valor de uma variável global, indicando o tempo necessário para fazer uma leitura, e assim que a variável booleana STATUS_CTU chegar no valor programado de tempo, o bloco aciona a variável SAIDA que vai dar sequência na lógica, realizando a leitura do sensor e reiniciando a contagem.

A fim de realizar uma leitura do sensor assim que o CLP for alterado para o modo RUN, o bit %S13 é acionado assim que o programa estiver rodando e vai setar a bobina MEM_AUX para acionar a linha do bloco MOVE. A função desse bit é definida da seguinte forma: Primeiro bit de varredura. Normalmente definido como 0, é definido como 1 pelo sistema durante o 1º ciclo da tarefa mestre após o CLP ser colocado em RUN. Na lógica de programação, o bit aciona uma bobina, como visto na Figura 18, que faz o programa ler a variável atual do sensor independente do temporizador, evitando esperar o ciclo inteiro de tempo para obter o primeiro valor, que dependendo do valor pré configurado de tempo, pode ser de horas de espera por cada leitura.

Figura 18 – Lógica acionando o bit %S13 e movendo o valor do sensor para *array*.



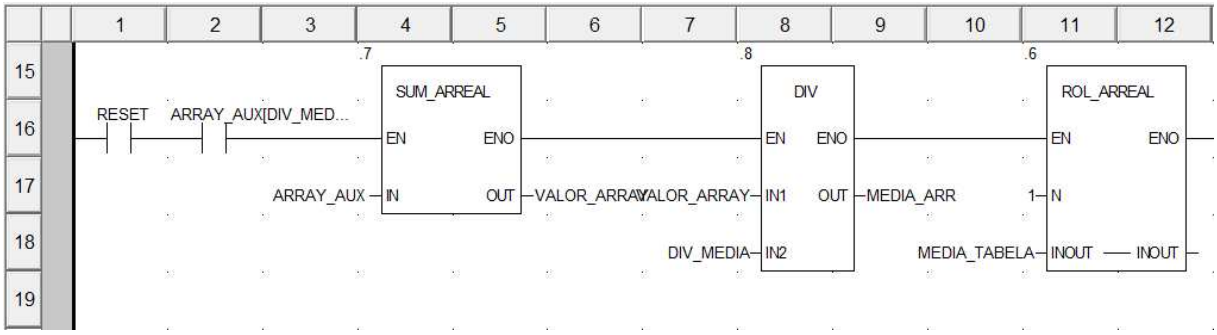
Fonte: (Autor, 2023).

Ao acionar a variável booleana SAIDA, o valor da variável VAR_ENTRADA, que recebe de uma variável global externa denominada VALOR (que é a saída do gerador de sinal escalonada do tipo real), é movido para a *array* ARRAY_AUX na posição 0. Essa *array* vai ser preenchida com a quantidade de leituras pré configurada a fim de obter a média dos valores, assim a *array* *array*_AUX auxilia no processo, armazenando os valores e realizando as operações matemáticas para se obter o valor que será realocada em outra *array*, que conterá apenas os valores das médias.

O bloco ROL_ARREAL, que é um bloco que desloca os valores da *array* na direção ascendente dos índices, ver ANEXO, é responsável por rotacionar os valores na *array* dinâmica. A variável **N** dentro do bloco, significa quantas posições o bloco vai rotacionar dentro da *array*. Nesse caso, está programada para rotacionar apenas 1 posição, sendo assim, todo último valor lido, será alocado na posição [1] da *array*, deixando a posição [0] sem nenhum valor.

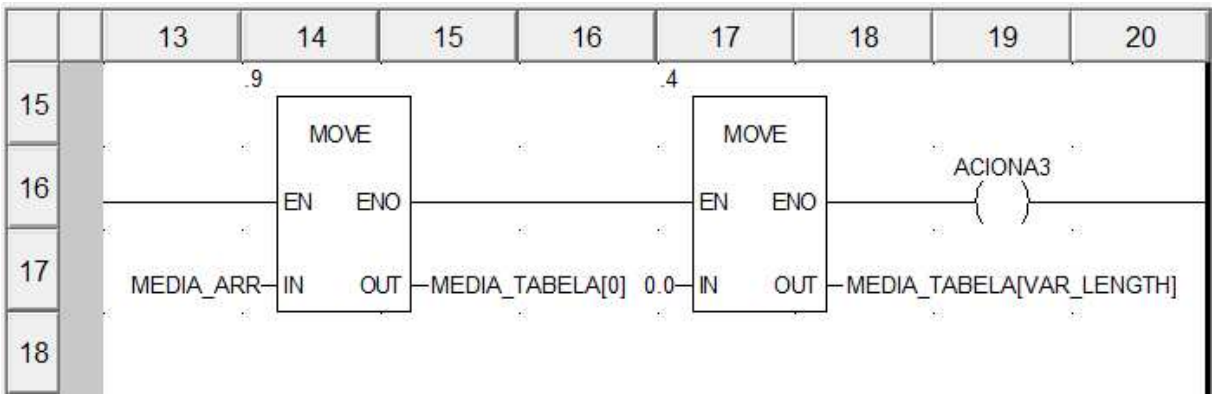
Em seguida, como mostram as Figuras 19 e 20, o bloco SUM_ARREAL é usado para somar todos os valores ocupados da *array* dinâmica, essa soma, denominada VALOR_ARRAY, é dividida pelo número de posições ocupadas na *array*, mas dessa vez do tipo real(visto na Figura 21), onde foi ajustada para o começo da programação do bloco, e alocada em outra *array*, MEDIA_TABELA, onde tem-se a tabela das médias, seguindo o mesmo caminho da Figura 18, cuja variável VAR_LENGTH determinará quantas posições da *array* MEDIA_TABELA serão usadas.

Figura 19 – Linha de programação calculando a média.



Fonte: (Autor, 2023).

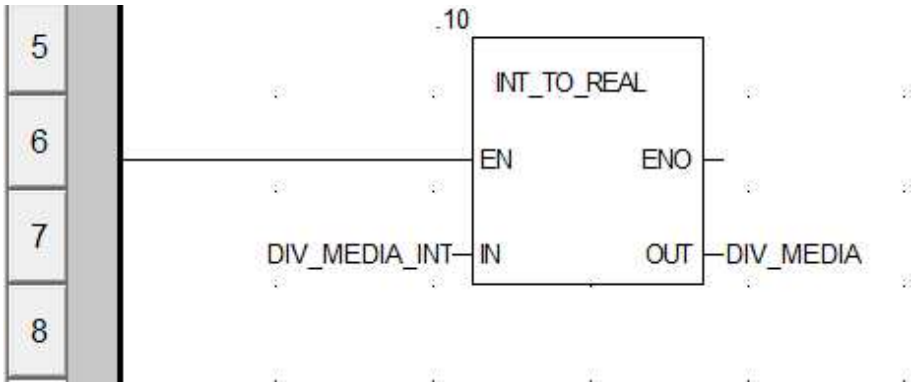
Figura 20 – Continuidade da linha da programação da média.



Fonte: (Autor, 2023).

A variável DIV_MEDIA_INT serve para definir o número de posições usadas dentro de uma *array* dinâmica, porém para realizar o cálculo da média, precisa-se usar números reais, uma vez que esse número de posições será o divisor na parte do cálculo. Com isso, usa o bloco INT_TO_REAL para a variável do tipo real DIV_MEDIA receber o valor da variável do tipo inteira e dar sequência nos cálculos.

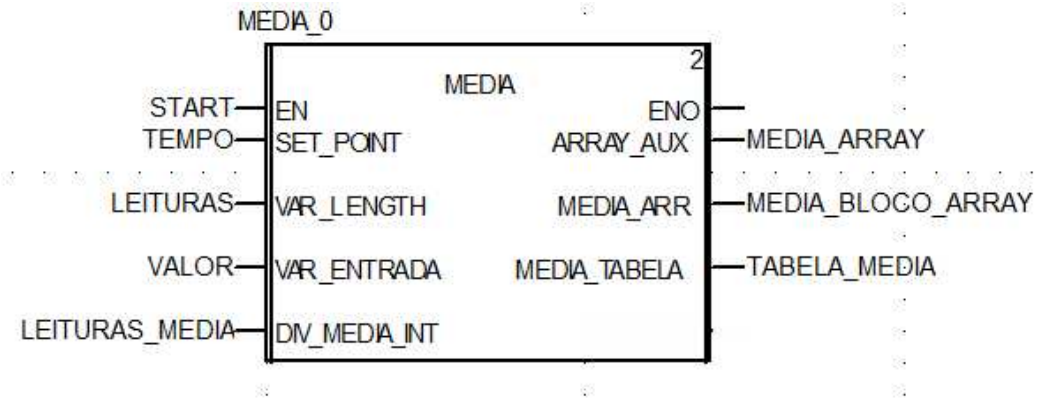
Figura 21 – Variável que define o número de posições a ser usada.



Fonte: (Autor, 2023).

A Figura 22 contém o bloco Média pronto para o uso, junto com suas variáveis globais que serão editadas de acordo com as necessidades desejadas, como o start do programa, o tempo de cada leitura, a quantidade de leituras a serem consideradas dentro de uma *array*, o valor recebido pelo gerador de sinal, entre outras. Essas variáveis globais serão discutidas no Capítulo 4.

Figura 22 – O bloco Média pronto e suas variáveis globais.



Fonte: (Autor, 2023).

3.4 BLOCO ESCALONAMENTO

Foi desenvolvido uma equação para escalonar os valores recebidos do gerador que, posteriormente, foi adaptada para a linguagem de programação texto estruturado no CLP.

A fórmula desenvolvida, primeiro no papel, é:

$$y = (x - x\_min)\frac{(y\_máx - y\_min)}{(x\_máx - x\_min)} + (y\_min); \quad (1)$$

onde,

- y = Saída Escalonada;
- x = Entrada Analógica;
- x_min = Sinal mínimo de entrada possível;
- x_máx = Sinal máximo de entrada possível;
- y_min = Sinal mínimo desejado na saída;
- y_máx = Sinal máximo desejado na saída.

O valor que aparecerá no programa para realizar os cálculos da média será a variável **y**, que precisa ter os limites definidos. Como por exemplo a medição da temperatura de um tanque com água. Primeiro define um valor mínimo e um valor máximo para a temperatura, Assim, quando o gerador de sinais (que pode variar de 0 a 10V) estiver em 0V, a temperatura do tanque será igual ao valor mínimo que foi definido. Por outro lado, quando o gerador estiver em 10V, a temperatura do tanque será igual ao valor máximo setado.

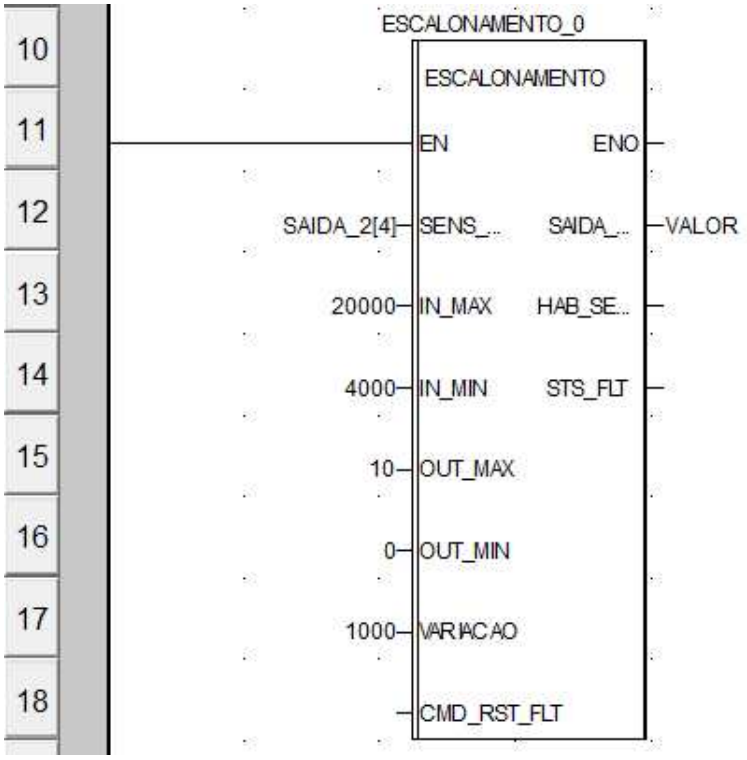
Ajustando a equação (1), na linguagem de programação ST e definindo todas as variáveis, a programação, desenvolvida em texto estruturado, é da seguinte forma:

```
// programacao do bloco de escalonamento
IF (SAIDA_REAL < 0.0) THEN
SAIDA_REAL := 0.0;
END_IF;

IF (HAB_SENSOR) THEN
SAIDA_REAL := ((INT_TO_REAL(SENSOR - IN_MIN)*INT_TO_REAL(OUT_MAX-OUT_MIN)/
    INT_TO_REAL(IN_MAX-IN_MIN))+INT_TO_REAL(OUT_MIN));

        IF (SAIDA_REAL < 0.0) THEN
        SAIDA_REAL := 0.0;
        END_IF;

    ELSIF (NOT HAB_SENSOR) THEN
SAIDA_REAL := 0.0;
END_IF
```

Sendo as variáveis:

- SAIDA_REAL = y;
- SENSOR = x;
- IN_MIN = y_min;
- IN_MAX = y_máx;
- OUT_MIN = x_min;
- OUT_MAX = x_máx;

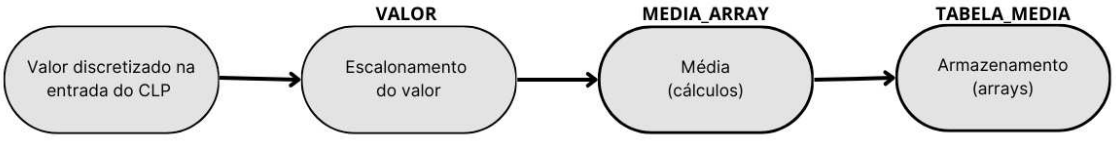
Figura 23 – Bloco de escalonamento.



Fonte: (Autor, 2023).

A Figura 23 mostra o bloco de escalonamento pronto e suas configurações. Nesse caso, ele está configurado para receber sinais de 4-20mA do gerador de sinal analógico. A variável global VALOR, na saída do bloco, é a mesma variável que está na entrada do bloco da Figura 22, onde após o escalonamento, vai fornecer o valor da leitura para o programa realizar as tarefas programadas. Também mostra que, quando o gerador estiver com a corrente de 20mA na saída, o seu valor dentro do programa será da unidade 10, e quando estiver em 4mA, o valor apresentado será 0. A Figura 24 exibe o diagrama da informação destacando as variáveis em evidência na programação do projeto e os principais passos na realização do projeto.

Figura 24 – Diagrama da Informação.



Fonte: (Autor, 2023).

4 RESULTADOS E DISCUSSÃO

Nesta seção, é apresentado todos os resultados obtidos a partir do desenvolvimento da lógica de programação e implementação.

4.1 SIMULAÇÃO COM GERADOR DE SINAL

Com a programação do CLP finalizada, os testes foram iniciados com o gerador de sinal analógico configurado, primeiramente, para gerar sinais de 4-20mA.

Figura 25 – Gerador de sinal configurado para 4-20mA.

Fonte: (Autor, 2023).

4.1.1 Predefinições das variáveis

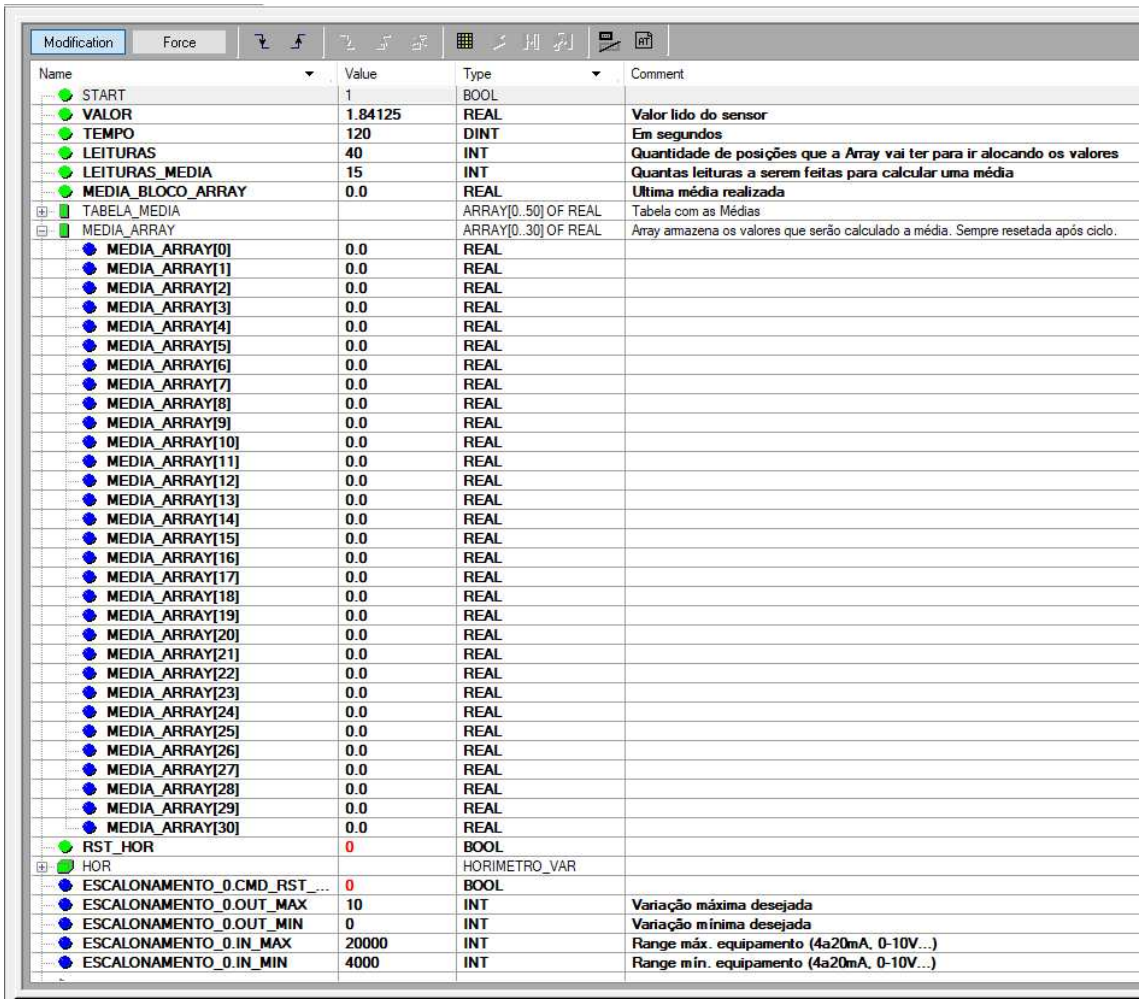
As variáveis globais vistas na Figura 26, necessárias ao desenvolvimento do trabalho, são definidas abaixo:

- START é a variável booleana que, quando setada de 0 para 1, deixará o CLP no modo RUN, dando início a varredura do sistema.
- TEMPO é a variável inteira que foi configurada para realizar a leitura do sensor a cada 120 segundos (2 minutos).

- VALOR é a variável do tipo real que recebe o valor do bloco de escalonamento, onde esse recebe o valor do gerador de sinal analógico e realiza a conversão, visto na Figura 23.
- LEITURAS é a variável do tipo inteira que define quantas posições da *array* dinâmica TABELA_MEDIA serão alocadas com os valores calculados na MEDIA_ARRAY. Para os testes, foi definido 40 alocações
- LEITURAS_MEDIA é a variável do tipo inteira que define quantas posições serão alocadas na *array* MEDIA_ARRAY para realizar a média de todas elas e alocar o resultado na *array* LEITURAS_MEDIA. Para os testes, definiu-se que a cada 15 leituras seria feito a média. Com o tempo predefinido em 2 minutos por leitura, ao final de 30 minutos, obtém-se 15 leituras e o cálculo é realizado.
- MEDIA_BLOCO_ARRAY apresenta a última média realizada.
- A *array* TABELA_MEDIA é do tipo real, e foi predefinida com 51 posições [0...50], que armazena as médias já calculadas.
- A *array* MEDIA_ARRAY é do tipo real, e aloca os valores do sensor a cada 120 segundos. Definiu-se o seu tamanho com 31 posições [0...30] e foi ajustada para alocar 15 leituras e realizar o cálculo da média das mesmas.
- HOR é o bloco do horímetro, que servirá para acompanhar o tempo que o processo está em modo RUN, podendo ser resetado quando a variável booleana RST_HOR estiver em 1. Sua lógica de programação encontra-se na Figura 34.
- Por fim pode-se observar as variáveis do bloco de escalonamento, onde tem-se a saída máxima e mínima, que serão configuradas de acordo com a necessidade, e a entrada do bloco que, nesse caso, está recebendo sinais de 4-20mA. Nessa configuração, quando o gerador estiver em 20mA, a variável VALOR estará em 10, e quando o gerador estiver em 4mA, a variável VALOR estará em 0.

As linhas são as variáveis do bloco Média, cada uma com o seu tipo definido pela coluna Type, a coluna Name representa os nomes dessas variáveis, a coluna Value é o valor de cada variável, a partir dela pode-se notar informações sobre o programa, como identificar se o programa está rodando, qual o tempo que foi programado para ler, quantas leituras a *array* vai precisar para realizar o cálculo da média, quantas médias a *array* vai alocar. Também consegue-se perceber, pela coluna Value, o valor de cada posição da *array*, e por último os valores de escalonamento, podendo identificar se o gerador de sinal está enviando sinais de 4-20mA ou 0-10V.

Figura 26 – Configuração para iniciar os testes.



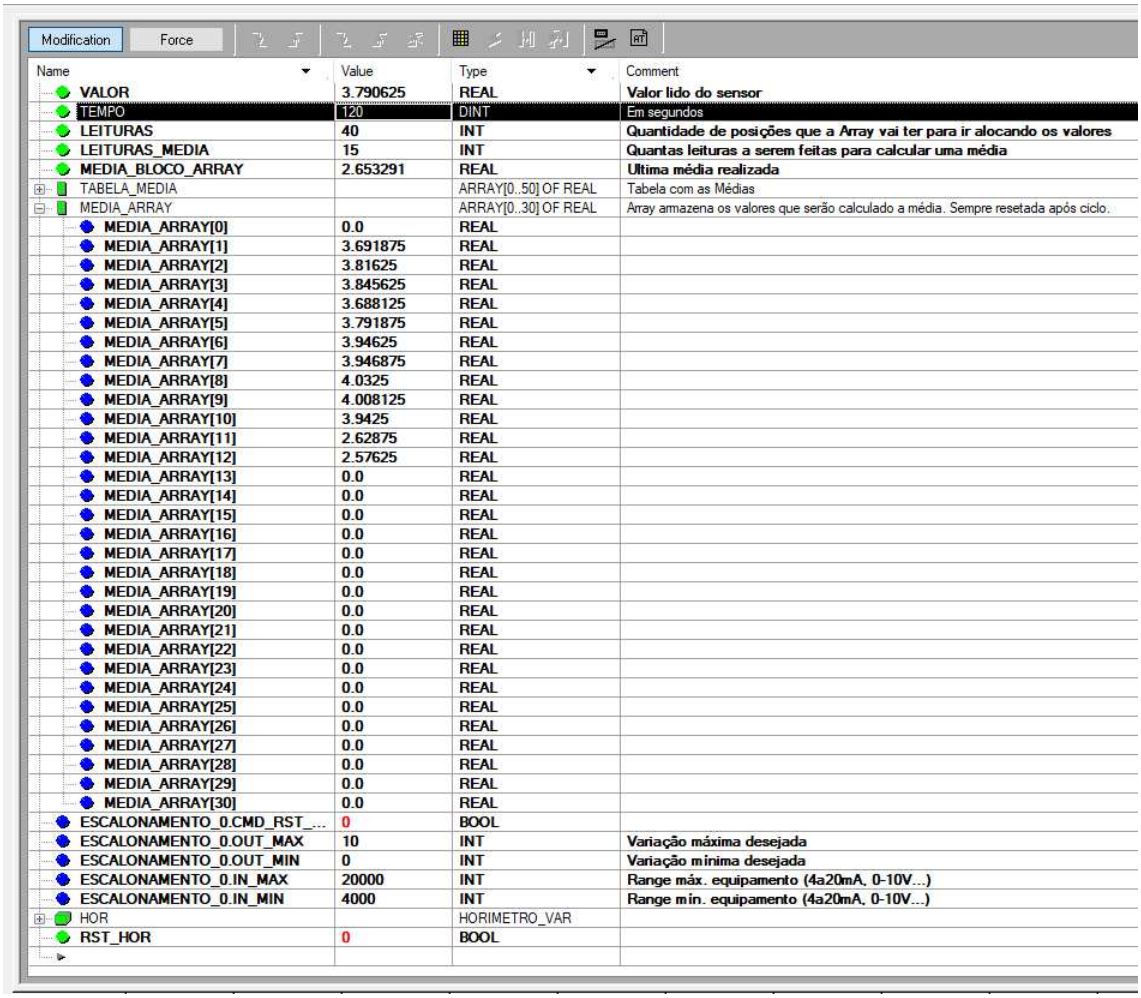
Modification | Force

Name	Value	Type	Comment
START	1	BOOL	
VALOR	1.84125	REAL	Valor lido do sensor
TEMPO	120	DINT	Em segundos
LEITURAS	40	INT	Quantidade de posições que a Array vai ter para ir alocando os valores
LEITURAS_MEDIA	15	INT	Quantas leituras a serem feitas para calcular uma média
MEDIA_BLOCO_ARRAY	0.0	REAL	Ultima média realizada
TABELA_MEDIA		ARRAY[0..50] OF REAL	Tabela com as Médias
MEDIA_ARRAY		ARRAY[0..30] OF REAL	Array armazena os valores que serão calculado a média. Sempre resetada após ciclo.
MEDIA_ARRAY[0]	0.0	REAL	
MEDIA_ARRAY[1]	0.0	REAL	
MEDIA_ARRAY[2]	0.0	REAL	
MEDIA_ARRAY[3]	0.0	REAL	
MEDIA_ARRAY[4]	0.0	REAL	
MEDIA_ARRAY[5]	0.0	REAL	
MEDIA_ARRAY[6]	0.0	REAL	
MEDIA_ARRAY[7]	0.0	REAL	
MEDIA_ARRAY[8]	0.0	REAL	
MEDIA_ARRAY[9]	0.0	REAL	
MEDIA_ARRAY[10]	0.0	REAL	
MEDIA_ARRAY[11]	0.0	REAL	
MEDIA_ARRAY[12]	0.0	REAL	
MEDIA_ARRAY[13]	0.0	REAL	
MEDIA_ARRAY[14]	0.0	REAL	
MEDIA_ARRAY[15]	0.0	REAL	
MEDIA_ARRAY[16]	0.0	REAL	
MEDIA_ARRAY[17]	0.0	REAL	
MEDIA_ARRAY[18]	0.0	REAL	
MEDIA_ARRAY[19]	0.0	REAL	
MEDIA_ARRAY[20]	0.0	REAL	
MEDIA_ARRAY[21]	0.0	REAL	
MEDIA_ARRAY[22]	0.0	REAL	
MEDIA_ARRAY[23]	0.0	REAL	
MEDIA_ARRAY[24]	0.0	REAL	
MEDIA_ARRAY[25]	0.0	REAL	
MEDIA_ARRAY[26]	0.0	REAL	
MEDIA_ARRAY[27]	0.0	REAL	
MEDIA_ARRAY[28]	0.0	REAL	
MEDIA_ARRAY[29]	0.0	REAL	
MEDIA_ARRAY[30]	0.0	REAL	
RST_HOR	0	BOOL	
HOR		HORIMETRO_VAR	
ESCALONAMENTO_0.CMD_RST_...	0	BOOL	
ESCALONAMENTO_0.OUT_MAX	10	INT	Variação máxima desejada
ESCALONAMENTO_0.OUT_MIN	0	INT	Variação mínima desejada
ESCALONAMENTO_0.IN_MAX	20000	INT	Range máx. equipamento (4a20mA, 0-10V...)
ESCALONAMENTO_0.IN_MIN	4000	INT	Range mín. equipamento (4a20mA, 0-10V...)

Fonte: (Autor, 2023).

Na Figura 27 pode-se observar as leituras sendo realizadas e alocadas nas posições predefinidas na *array*. Sendo assim, quando a posição MEDIA_ARRAY[15] for preenchida, o programa fará a média dos resultados e alocará na posição 1 da *array* TABELA_MEDIA.

Figura 27 – O programa em funcionamento e os valores sendo lidos.



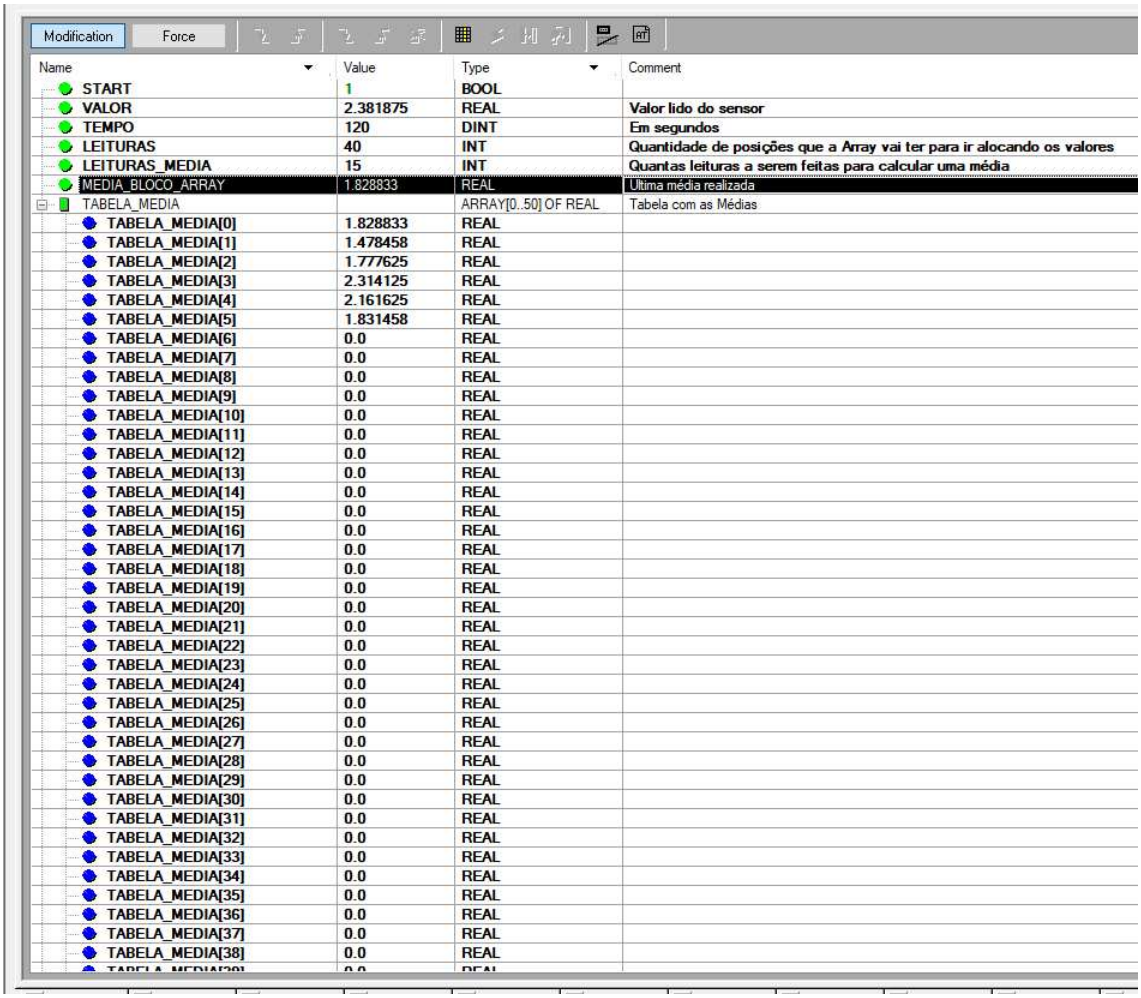
Modification | Force

Name	Value	Type	Comment
VALOR	3.790625	REAL	Valor lido do sensor
TEMPO	120	DINT	Em segundos
LEITURAS	40	INT	Quantidade de posições que a Array vai ter para ir alocando os valores
LEITURAS_MEDIA	15	INT	Quantas leituras a serem feitas para calcular uma média
MEDIA_BLOCO_ARRAY	2.653291	REAL	Ultima média realizada
TABELA_MEDIA		ARRAY[0..50] OF REAL	Tabela com as Médias
MEDIA_ARRAY		ARRAY[0..30] OF REAL	Array armazena os valores que serão calculado a média. Sempre resetada após ciclo.
MEDIA_ARRAY[0]	0.0	REAL	
MEDIA_ARRAY[1]	3.691875	REAL	
MEDIA_ARRAY[2]	3.81625	REAL	
MEDIA_ARRAY[3]	3.845625	REAL	
MEDIA_ARRAY[4]	3.688125	REAL	
MEDIA_ARRAY[5]	3.791875	REAL	
MEDIA_ARRAY[6]	3.94625	REAL	
MEDIA_ARRAY[7]	3.946875	REAL	
MEDIA_ARRAY[8]	4.0325	REAL	
MEDIA_ARRAY[9]	4.008125	REAL	
MEDIA_ARRAY[10]	3.9425	REAL	
MEDIA_ARRAY[11]	2.62875	REAL	
MEDIA_ARRAY[12]	2.57625	REAL	
MEDIA_ARRAY[13]	0.0	REAL	
MEDIA_ARRAY[14]	0.0	REAL	
MEDIA_ARRAY[15]	0.0	REAL	
MEDIA_ARRAY[16]	0.0	REAL	
MEDIA_ARRAY[17]	0.0	REAL	
MEDIA_ARRAY[18]	0.0	REAL	
MEDIA_ARRAY[19]	0.0	REAL	
MEDIA_ARRAY[20]	0.0	REAL	
MEDIA_ARRAY[21]	0.0	REAL	
MEDIA_ARRAY[22]	0.0	REAL	
MEDIA_ARRAY[23]	0.0	REAL	
MEDIA_ARRAY[24]	0.0	REAL	
MEDIA_ARRAY[25]	0.0	REAL	
MEDIA_ARRAY[26]	0.0	REAL	
MEDIA_ARRAY[27]	0.0	REAL	
MEDIA_ARRAY[28]	0.0	REAL	
MEDIA_ARRAY[29]	0.0	REAL	
MEDIA_ARRAY[30]	0.0	REAL	
ESCALONAMENTO_0.CMD_RST_...	0	BOOL	
ESCALONAMENTO_0.OUT_MAX	10	INT	Variação máxima desejada
ESCALONAMENTO_0.OUT_MIN	0	INT	Variação mínima desejada
ESCALONAMENTO_0.IN_MAX	20000	INT	Range máx. equipamento (4a20mA, 0-10V...)
ESCALONAMENTO_0.IN_MIN	4000	INT	Range mín. equipamento (4a20mA, 0-10V...)
HOR		HORIMETRO_VAR	
RST_HOR	0	BOOL	

Fonte: (Autor, 2023).

Na Figura 28 a *array* TABELA_MEDIA está sendo preenchida com a média de 15 valores lidos, cada valor lido é enviado a cada 120 segundos. Quando as 40 posições estiverem alocadas com as médias, a primeira média realizada será excluída para a seguinte ocupar a posição e fazer rotacionar os valores das médias, sempre alocando o valor mais recente na primeira posição da *array*.

Figura 28 – Tabela com as médias sendo preenchida.



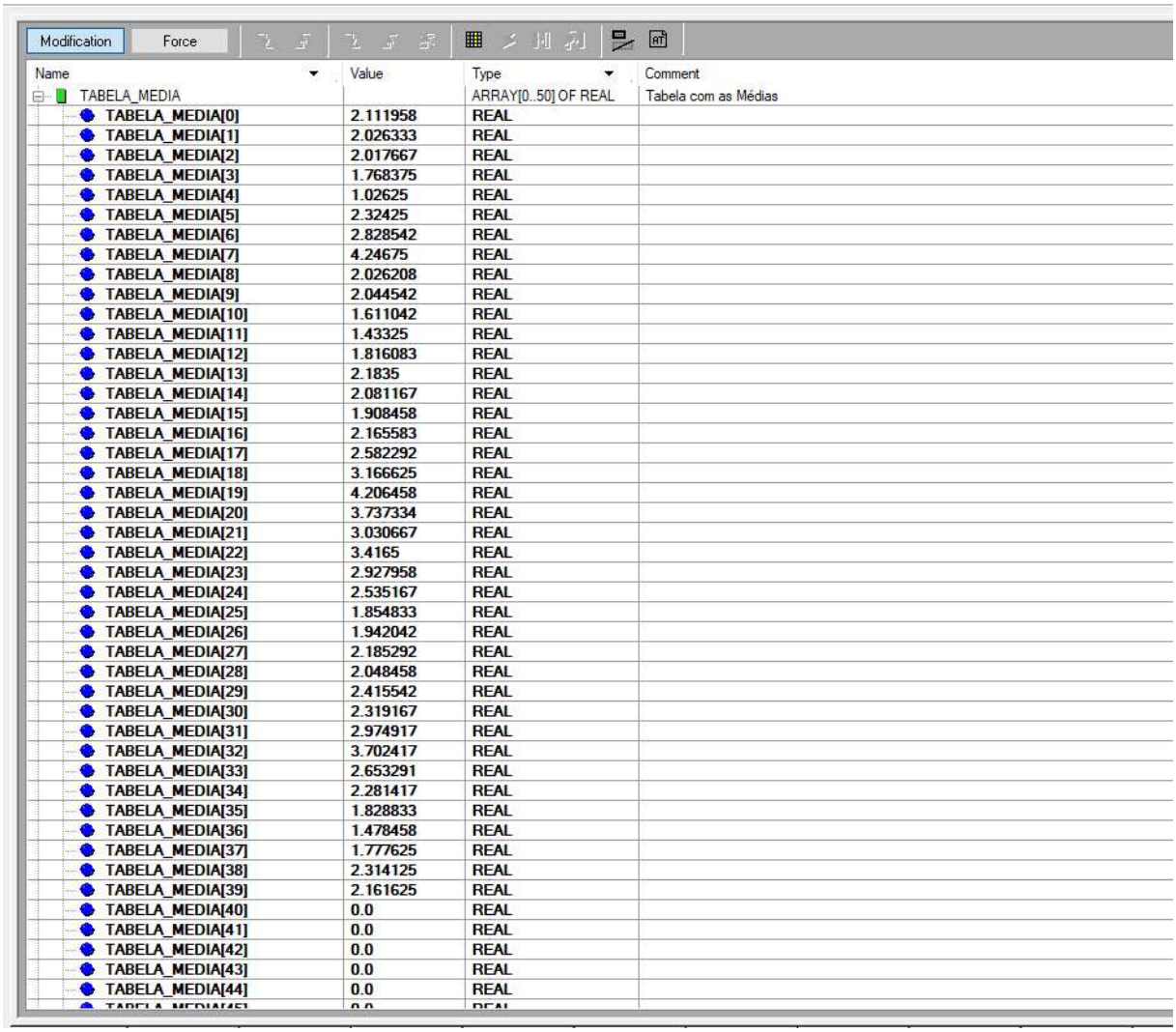
Modification | Force

Name	Value	Type	Comment
START	1	BOOL	
VALOR	2.381875	REAL	Valor lido do sensor
TEMPO	120	DINT	Em segundos
LEITURAS	40	INT	Quantidade de posições que a Array vai ter para ir alocando os valores
LEITURAS_MEDIA	15	INT	Quantas leituras a serem feitas para calcular uma média
MEDIA_BLOCO_ARRAY	1.828833	REAL	Última média realizada
TABELA_MEDIA		ARRAY[0..50] OF REAL	Tabela com as Médias
TABELA_MEDIA[0]	1.828833	REAL	
TABELA_MEDIA[1]	1.478458	REAL	
TABELA_MEDIA[2]	1.777625	REAL	
TABELA_MEDIA[3]	2.314125	REAL	
TABELA_MEDIA[4]	2.161625	REAL	
TABELA_MEDIA[5]	1.831458	REAL	
TABELA_MEDIA[6]	0.0	REAL	
TABELA_MEDIA[7]	0.0	REAL	
TABELA_MEDIA[8]	0.0	REAL	
TABELA_MEDIA[9]	0.0	REAL	
TABELA_MEDIA[10]	0.0	REAL	
TABELA_MEDIA[11]	0.0	REAL	
TABELA_MEDIA[12]	0.0	REAL	
TABELA_MEDIA[13]	0.0	REAL	
TABELA_MEDIA[14]	0.0	REAL	
TABELA_MEDIA[15]	0.0	REAL	
TABELA_MEDIA[16]	0.0	REAL	
TABELA_MEDIA[17]	0.0	REAL	
TABELA_MEDIA[18]	0.0	REAL	
TABELA_MEDIA[19]	0.0	REAL	
TABELA_MEDIA[20]	0.0	REAL	
TABELA_MEDIA[21]	0.0	REAL	
TABELA_MEDIA[22]	0.0	REAL	
TABELA_MEDIA[23]	0.0	REAL	
TABELA_MEDIA[24]	0.0	REAL	
TABELA_MEDIA[25]	0.0	REAL	
TABELA_MEDIA[26]	0.0	REAL	
TABELA_MEDIA[27]	0.0	REAL	
TABELA_MEDIA[28]	0.0	REAL	
TABELA_MEDIA[29]	0.0	REAL	
TABELA_MEDIA[30]	0.0	REAL	
TABELA_MEDIA[31]	0.0	REAL	
TABELA_MEDIA[32]	0.0	REAL	
TABELA_MEDIA[33]	0.0	REAL	
TABELA_MEDIA[34]	0.0	REAL	
TABELA_MEDIA[35]	0.0	REAL	
TABELA_MEDIA[36]	0.0	REAL	
TABELA_MEDIA[37]	0.0	REAL	
TABELA_MEDIA[38]	0.0	REAL	

Fonte: (Autor, 2023).

A Figura 29 mostra as 40 posições da *array* TABELA_MEDIA preenchida com as médias. Na Figura 28, o primeiro valor era 1.831458 e o segundo 2.161625, e, ao alocar um novo valor na primeira posição da *array*, o último valor some, rotacionado todos para a posição seguinte.

Figura 29 – A tabela com os valores transferidos para a posição seguinte a cada nova média.



Modification | Force

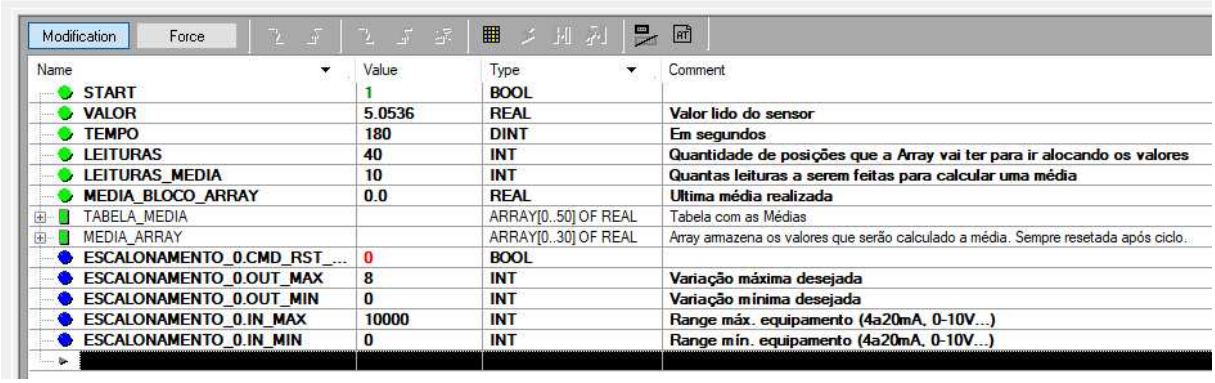
Name	Value	Type	Comment
TABELA_MEDIA		ARRAY[0..50] OF REAL	Tabela com as Médias
TABELA_MEDIA[0]	2.111958	REAL	
TABELA_MEDIA[1]	2.026333	REAL	
TABELA_MEDIA[2]	2.017667	REAL	
TABELA_MEDIA[3]	1.768375	REAL	
TABELA_MEDIA[4]	1.02625	REAL	
TABELA_MEDIA[5]	2.32425	REAL	
TABELA_MEDIA[6]	2.828542	REAL	
TABELA_MEDIA[7]	4.24675	REAL	
TABELA_MEDIA[8]	2.026208	REAL	
TABELA_MEDIA[9]	2.044542	REAL	
TABELA_MEDIA[10]	1.611042	REAL	
TABELA_MEDIA[11]	1.43325	REAL	
TABELA_MEDIA[12]	1.816083	REAL	
TABELA_MEDIA[13]	2.1835	REAL	
TABELA_MEDIA[14]	2.081167	REAL	
TABELA_MEDIA[15]	1.908458	REAL	
TABELA_MEDIA[16]	2.165583	REAL	
TABELA_MEDIA[17]	2.582292	REAL	
TABELA_MEDIA[18]	3.166625	REAL	
TABELA_MEDIA[19]	4.206458	REAL	
TABELA_MEDIA[20]	3.737334	REAL	
TABELA_MEDIA[21]	3.030667	REAL	
TABELA_MEDIA[22]	3.4165	REAL	
TABELA_MEDIA[23]	2.927958	REAL	
TABELA_MEDIA[24]	2.535167	REAL	
TABELA_MEDIA[25]	1.854833	REAL	
TABELA_MEDIA[26]	1.942042	REAL	
TABELA_MEDIA[27]	2.185292	REAL	
TABELA_MEDIA[28]	2.048458	REAL	
TABELA_MEDIA[29]	2.415542	REAL	
TABELA_MEDIA[30]	2.319167	REAL	
TABELA_MEDIA[31]	2.974917	REAL	
TABELA_MEDIA[32]	3.702417	REAL	
TABELA_MEDIA[33]	2.653291	REAL	
TABELA_MEDIA[34]	2.281417	REAL	
TABELA_MEDIA[35]	1.828833	REAL	
TABELA_MEDIA[36]	1.478458	REAL	
TABELA_MEDIA[37]	1.777625	REAL	
TABELA_MEDIA[38]	2.314125	REAL	
TABELA_MEDIA[39]	2.161625	REAL	
TABELA_MEDIA[40]	0.0	REAL	
TABELA_MEDIA[41]	0.0	REAL	
TABELA_MEDIA[42]	0.0	REAL	
TABELA_MEDIA[43]	0.0	REAL	
TABELA_MEDIA[44]	0.0	REAL	

Fonte: (Autor, 2023).

4.1.2 Teste com Gerador de sinal com saídas de 0 a 10V

Para dar início aos testes com o gerador de sinal funcionando agora com sua saída fornecendo tensões de 0 a 10V, foi alterado os valores de máximo e mínimo do sistema, visto na Figura 30, a fim de simular com valores diferentes em relação ao teste anterior, e, quando o gerador estiver gerando 10V, o sistema vai receber o valor 8, e quando o gerador estiver gerando 0V de sinal, o sistema vai receber o valor 0, realizando o escalonamento com valores entre 0-10V (0 a 8).

Figura 30 – Valores predefinidos para iniciar os testes.

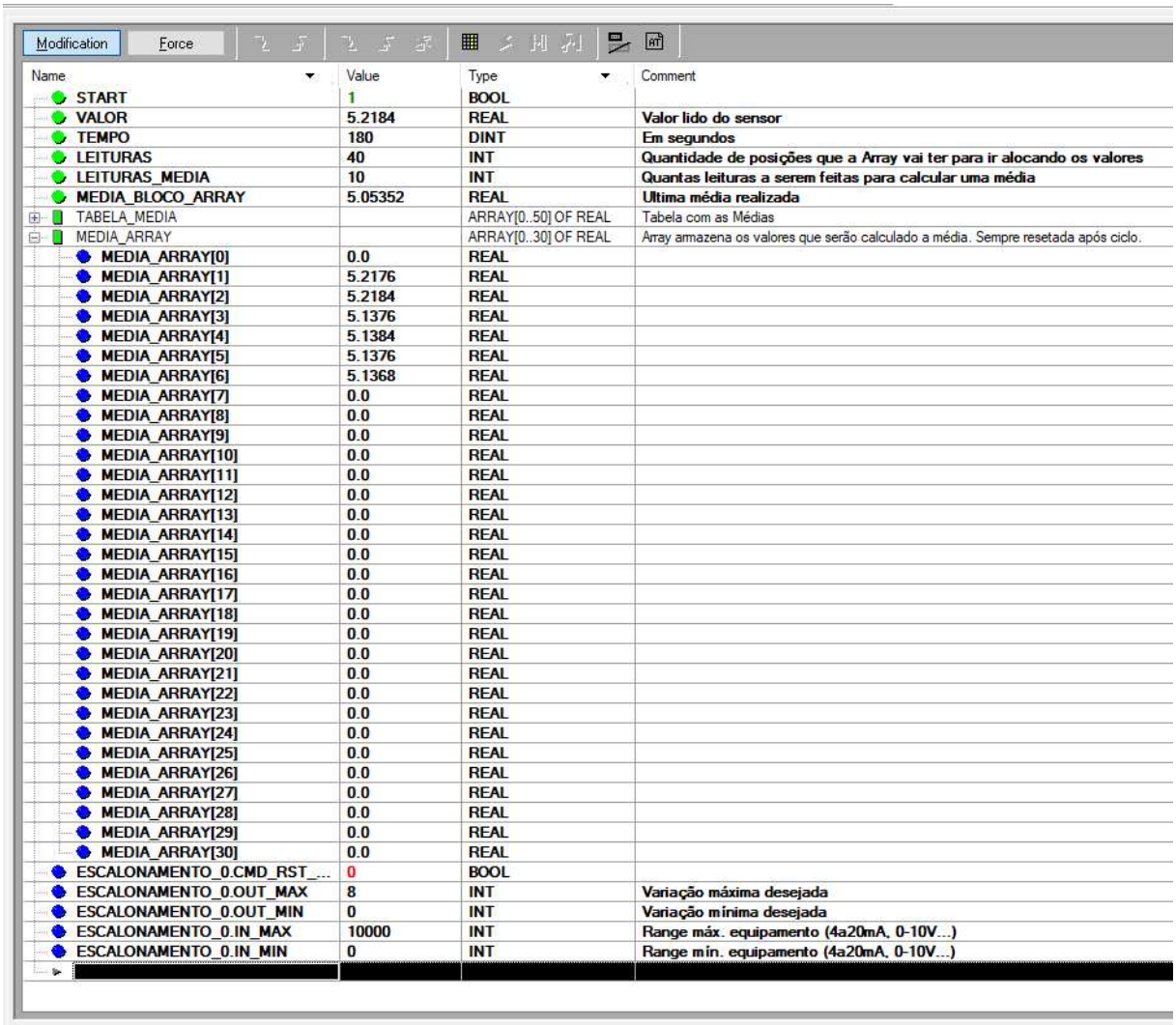


Name	Value	Type	Comment
START	1	BOOL	
VALOR	5.0536	REAL	Valor lido do sensor
TEMPO	180	DINT	Em segundos
LEITURAS	40	INT	Quantidade de posições que a Array vai ter para ir alocando os valores
LEITURAS_MEDIA	10	INT	Quantas leituras a serem feitas para calcular uma média
MEDIA_BLOCO_ARRAY	0.0	REAL	Ultima média realizada
TABELA_MEDIA		ARRAY[0..50] OF REAL	Tabela com as Médias
MEDIA_ARRAY		ARRAY[0..30] OF REAL	Array armazena os valores que serão calculado a média. Sempre resetada após ciclo.
ESCALONAMENTO_0.CMD_RST_...	0	BOOL	
ESCALONAMENTO_0.OUT_MAX	8	INT	Variação máxima desejada
ESCALONAMENTO_0.OUT_MIN	0	INT	Variação mínima desejada
ESCALONAMENTO_0.IN_MAX	10000	INT	Range máx. equipamento (4a20mA, 0-10V...)
ESCALONAMENTO_0.IN_MIN	0	INT	Range mín. equipamento (4a20mA, 0-10V...)

Fonte: (Autor, 2023).

A fim de testar com um tempo de leitura diferente do teste com o gerador de 4-20mA, o tempo foi alterado para 180 segundos (3 minutos), e agora a média será realizada a cada 10 leituras, mostrando que pode ser editável dependendo da necessidade de quem estiver trabalhando com esse processo. O intuito foi alterar o tempo de leitura e a quantidade de leituras alocadas para realizar o cálculo de média.

Figura 31 – Array que realiza o cálculo da média sendo preenchida a cada 180 segundos.



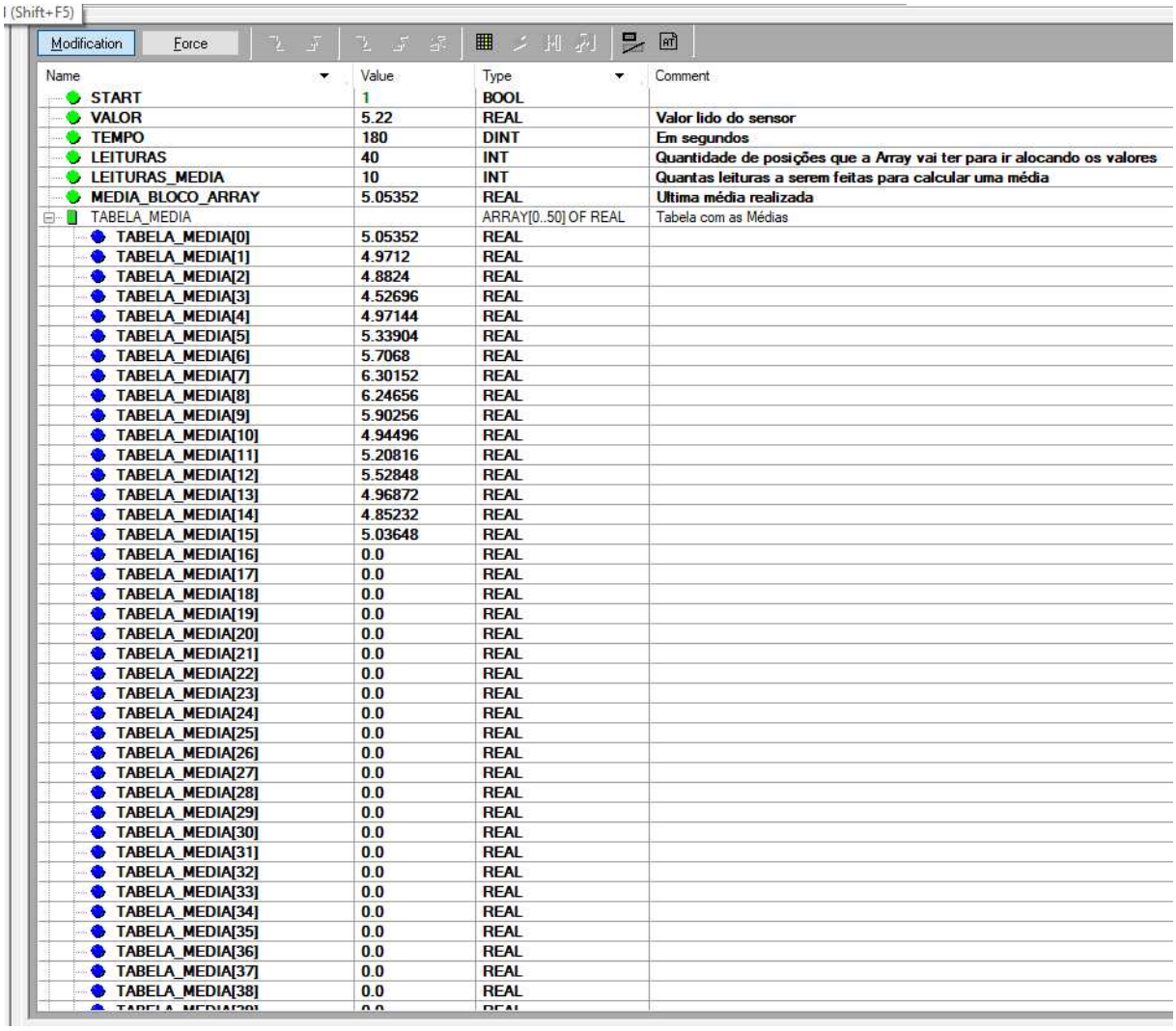
Modification | Force

Name	Value	Type	Comment
START	1	BOOL	
VALOR	5.2184	REAL	Valor lido do sensor
TEMPO	180	DINT	Em segundos
LEITURAS	40	INT	Quantidade de posições que a Array vai ter para ir alocando os valores
LEITURAS_MEDIA	10	INT	Quantas leituras a serem feitas para calcular uma média
MEDIA_BLOCO_ARRAY	5.05352	REAL	Ultima média realizada
TABELA_MEDIA		ARRAY[0..50] OF REAL	Tabela com as Médias
MEDIA_ARRAY		ARRAY[0..30] OF REAL	Array armazena os valores que serão calculado a média. Sempre resetada após ciclo.
MEDIA_ARRAY[0]	0.0	REAL	
MEDIA_ARRAY[1]	5.2176	REAL	
MEDIA_ARRAY[2]	5.2184	REAL	
MEDIA_ARRAY[3]	5.1376	REAL	
MEDIA_ARRAY[4]	5.1384	REAL	
MEDIA_ARRAY[5]	5.1376	REAL	
MEDIA_ARRAY[6]	5.1368	REAL	
MEDIA_ARRAY[7]	0.0	REAL	
MEDIA_ARRAY[8]	0.0	REAL	
MEDIA_ARRAY[9]	0.0	REAL	
MEDIA_ARRAY[10]	0.0	REAL	
MEDIA_ARRAY[11]	0.0	REAL	
MEDIA_ARRAY[12]	0.0	REAL	
MEDIA_ARRAY[13]	0.0	REAL	
MEDIA_ARRAY[14]	0.0	REAL	
MEDIA_ARRAY[15]	0.0	REAL	
MEDIA_ARRAY[16]	0.0	REAL	
MEDIA_ARRAY[17]	0.0	REAL	
MEDIA_ARRAY[18]	0.0	REAL	
MEDIA_ARRAY[19]	0.0	REAL	
MEDIA_ARRAY[20]	0.0	REAL	
MEDIA_ARRAY[21]	0.0	REAL	
MEDIA_ARRAY[22]	0.0	REAL	
MEDIA_ARRAY[23]	0.0	REAL	
MEDIA_ARRAY[24]	0.0	REAL	
MEDIA_ARRAY[25]	0.0	REAL	
MEDIA_ARRAY[26]	0.0	REAL	
MEDIA_ARRAY[27]	0.0	REAL	
MEDIA_ARRAY[28]	0.0	REAL	
MEDIA_ARRAY[29]	0.0	REAL	
MEDIA_ARRAY[30]	0.0	REAL	
ESCALONAMENTO_0.CMD_RST_...	0	BOOL	
ESCALONAMENTO_0.OUT_MAX	8	INT	Variação máxima desejada
ESCALONAMENTO_0.OUT_MIN	0	INT	Variação mínima desejada
ESCALONAMENTO_0.IN_MAX	10000	INT	Range máx. equipamento (4a20mA, 0-10V...)
ESCALONAMENTO_0.IN_MIN	0	INT	Range mín. equipamento (4a20mA, 0-10V...)

Fonte: (Autor, 2023).

A Figura 31 apresenta a tabela com as posições da *array* sendo preenchidas com os valores em que foram lidos a cada ciclo de tempo.

Figura 32 – Tabela com as médias calculadas.



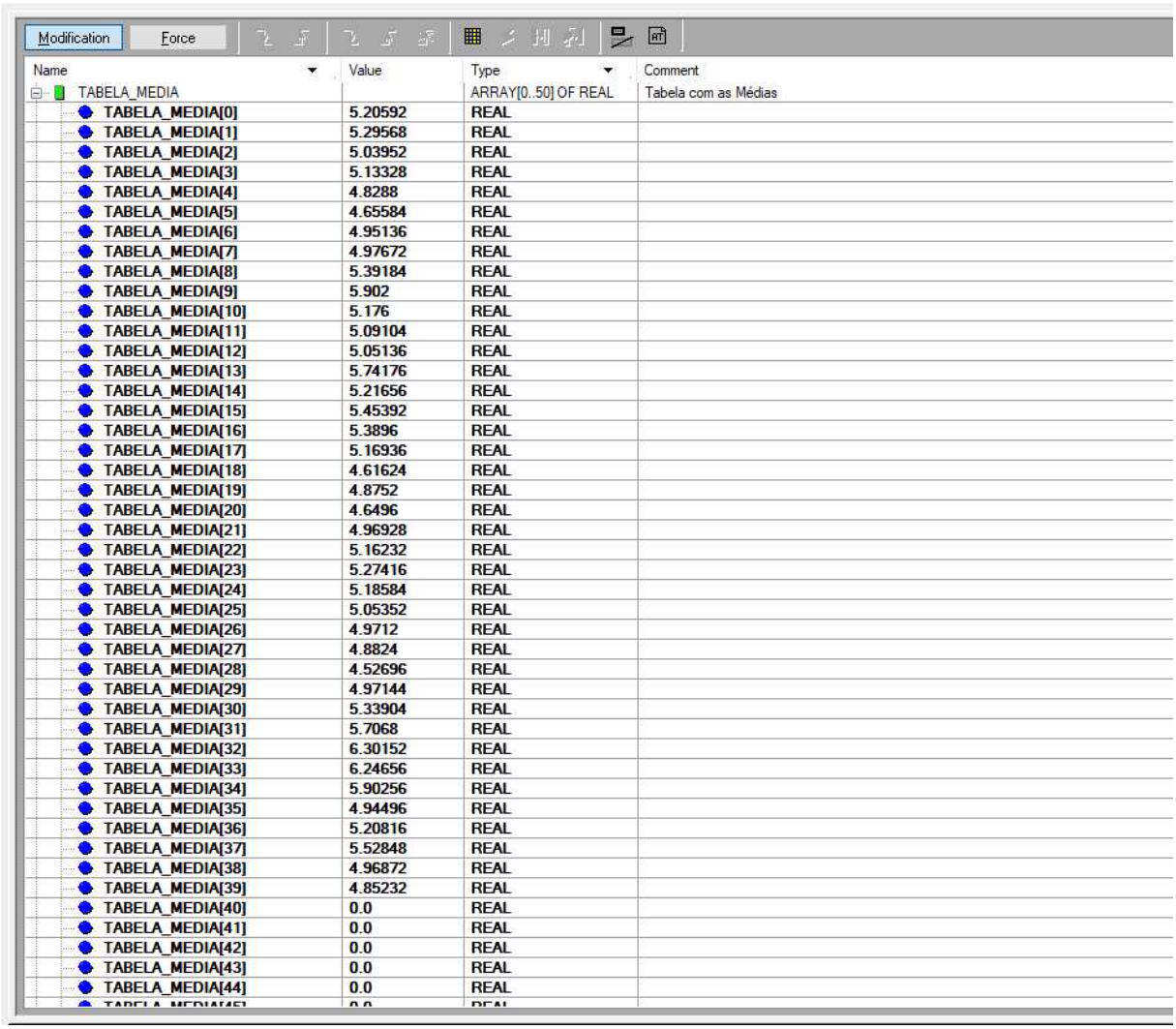
Name	Value	Type	Comment
START	1	BOOL	
VALOR	5.22	REAL	Valor lido do sensor
TEMPO	180	DINT	Em segundos
LEITURAS	40	INT	Quantidade de posições que a Array vai ter para ir alocando os valores
LEITURAS_MEDIA	10	INT	Quantas leituras a serem feitas para calcular uma média
MEDIA_BLOCO_ARRAY	5.05352	REAL	Ultima média realizada
TABELA_MEDIA		ARRAY[0..50] OF REAL	Tabela com as Médias
TABELA_MEDIA[0]	5.05352	REAL	
TABELA_MEDIA[1]	4.9712	REAL	
TABELA_MEDIA[2]	4.8824	REAL	
TABELA_MEDIA[3]	4.52696	REAL	
TABELA_MEDIA[4]	4.97144	REAL	
TABELA_MEDIA[5]	5.33904	REAL	
TABELA_MEDIA[6]	5.7068	REAL	
TABELA_MEDIA[7]	6.30152	REAL	
TABELA_MEDIA[8]	6.24656	REAL	
TABELA_MEDIA[9]	5.90256	REAL	
TABELA_MEDIA[10]	4.94496	REAL	
TABELA_MEDIA[11]	5.20816	REAL	
TABELA_MEDIA[12]	5.52848	REAL	
TABELA_MEDIA[13]	4.96872	REAL	
TABELA_MEDIA[14]	4.85232	REAL	
TABELA_MEDIA[15]	5.03648	REAL	
TABELA_MEDIA[16]	0.0	REAL	
TABELA_MEDIA[17]	0.0	REAL	
TABELA_MEDIA[18]	0.0	REAL	
TABELA_MEDIA[19]	0.0	REAL	
TABELA_MEDIA[20]	0.0	REAL	
TABELA_MEDIA[21]	0.0	REAL	
TABELA_MEDIA[22]	0.0	REAL	
TABELA_MEDIA[23]	0.0	REAL	
TABELA_MEDIA[24]	0.0	REAL	
TABELA_MEDIA[25]	0.0	REAL	
TABELA_MEDIA[26]	0.0	REAL	
TABELA_MEDIA[27]	0.0	REAL	
TABELA_MEDIA[28]	0.0	REAL	
TABELA_MEDIA[29]	0.0	REAL	
TABELA_MEDIA[30]	0.0	REAL	
TABELA_MEDIA[31]	0.0	REAL	
TABELA_MEDIA[32]	0.0	REAL	
TABELA_MEDIA[33]	0.0	REAL	
TABELA_MEDIA[34]	0.0	REAL	
TABELA_MEDIA[35]	0.0	REAL	
TABELA_MEDIA[36]	0.0	REAL	
TABELA_MEDIA[37]	0.0	REAL	
TABELA_MEDIA[38]	0.0	REAL	

Fonte: (Autor, 2023).

Como mostrado na Figura 32, quando 10 leituras são feitas, a *array* MEDIA_ARRAY faz o cálculo da média e manda o valor para a *array* TABELA_MEDIA, onde será alocado na primeira posição cada vez que vier um valor novo.

Essa parte da programação é mostrada na Figura 19, onde o resultado do cálculo da média (MEDIA_ARR) é movido para a variável MEDIA_TABELA, e quando o bloco chamado MEDIA (Figura 22) estiver pronto, a variável global TABELA_MEDIA, que é a *array* dinâmica em questão, vai receber os valores da variável MEDIA_TABELA.

Figura 33 – Tabela com todas as posições predefinidas preenchidas.



Modification | Force

Name	Value	Type	Comment
TABELA_MEDIA		ARRAY[0..50] OF REAL	Tabela com as Médias
TABELA_MEDIA[0]	5.20592	REAL	
TABELA_MEDIA[1]	5.29568	REAL	
TABELA_MEDIA[2]	5.03952	REAL	
TABELA_MEDIA[3]	5.13328	REAL	
TABELA_MEDIA[4]	4.8288	REAL	
TABELA_MEDIA[5]	4.65584	REAL	
TABELA_MEDIA[6]	4.95136	REAL	
TABELA_MEDIA[7]	4.97672	REAL	
TABELA_MEDIA[8]	5.39184	REAL	
TABELA_MEDIA[9]	5.902	REAL	
TABELA_MEDIA[10]	5.176	REAL	
TABELA_MEDIA[11]	5.09104	REAL	
TABELA_MEDIA[12]	5.05136	REAL	
TABELA_MEDIA[13]	5.74176	REAL	
TABELA_MEDIA[14]	5.21656	REAL	
TABELA_MEDIA[15]	5.45392	REAL	
TABELA_MEDIA[16]	5.3896	REAL	
TABELA_MEDIA[17]	5.16936	REAL	
TABELA_MEDIA[18]	4.61624	REAL	
TABELA_MEDIA[19]	4.8752	REAL	
TABELA_MEDIA[20]	4.6496	REAL	
TABELA_MEDIA[21]	4.96928	REAL	
TABELA_MEDIA[22]	5.16232	REAL	
TABELA_MEDIA[23]	5.27416	REAL	
TABELA_MEDIA[24]	5.18584	REAL	
TABELA_MEDIA[25]	5.05352	REAL	
TABELA_MEDIA[26]	4.9712	REAL	
TABELA_MEDIA[27]	4.8824	REAL	
TABELA_MEDIA[28]	4.52696	REAL	
TABELA_MEDIA[29]	4.97144	REAL	
TABELA_MEDIA[30]	5.33904	REAL	
TABELA_MEDIA[31]	5.7068	REAL	
TABELA_MEDIA[32]	6.30152	REAL	
TABELA_MEDIA[33]	6.24656	REAL	
TABELA_MEDIA[34]	5.90256	REAL	
TABELA_MEDIA[35]	4.94496	REAL	
TABELA_MEDIA[36]	5.20816	REAL	
TABELA_MEDIA[37]	5.52848	REAL	
TABELA_MEDIA[38]	4.96872	REAL	
TABELA_MEDIA[39]	4.85232	REAL	
TABELA_MEDIA[40]	0.0	REAL	
TABELA_MEDIA[41]	0.0	REAL	
TABELA_MEDIA[42]	0.0	REAL	
TABELA_MEDIA[43]	0.0	REAL	
TABELA_MEDIA[44]	0.0	REAL	

Fonte: (Autor, 2023).

A Figura 33 apresenta a tabela rotacionando os valores e assim que todas as posições predefinidas são preenchidas, o valor na última posição some dando lugar para as seguintes, sempre adicionando um novo valor na primeira posição. Na Figura 32, a primeira média calculada foi 5.03648 e a segunda média foi 4.85232, e na Figura 33 é mostrado o primeiro valor sendo excluído para dar lugar a leituras mais novas e por assim em diante.

5 CONCLUSÃO

A motivação para a realização deste trabalho teve como base desenvolver uma programação no CLP para criar um bloco funcional que coletasse dados a partir de predefinições configuráveis pelo usuário, de acordo com suas necessidades e o meio de uso da aplicação. Os resultados obtidos confirmaram que os objetivos foram plenamente alcançados, demonstrando a eficácia do bloco funcional em proporcionar a flexibilidade e funcionalidade esperadas para a coleta de dados. A aplicação na bancada de testes mostrou que o programa consegue ser aplicável em campo, seja para medir temperatura, pressão entre outros tipos de medição que possa ser ligado um sensor ao CLP. A principal contribuição deste projeto foi fazer a base de programação e a validação dos testes na qual fui proposto a realizar com a empresa Hidryco. O trabalho segue aberto para dar continuidade aos propósitos da empresa e seguir caminhos mais longos, utilizando outras ferramentas, como supervisórios e IHM, fazendo outros tipos de cálculos e tabelas para satisfazer o propósito do projeto.

REFERÊNCIAS

ALTUS. **Pirâmide Automação**. Altus. 2023. Disponível em: https://www.altus.com.br/post/502/o-que-e-automacao-industrial-3f.

ALVES, Thiago; MORRIS, Thomas. OpenPLC: An IEC 61,131–3 compliant open source industrial controller for cyber security research. **Computers & Security**, Elsevier, v. 78, p. 364–379, 2018.

BENDER, Darlam Fabio; COMBEMALE, Benoıt; CRÉGUT, Xavier; FARINES, Jean Marie; BERTHOMIEU, Bernard; VERNADAT, François. Ladder metamodeling and PLC program validation through time Petri nets. *In*: SPRINGER. MODEL Driven Architecture–Foundations and Applications: 4th European Conference, ECMDA-FA 2008, Berlin, Germany, June 9-13, 2008. Proceedings 4. [*S.l.*: *s.n.*], 2008. P. 121–136.

DA SILVA, Edilson Alfredo. **Introdução às linguagens de programação para CLP**. [*S.l.*]: Editora Blucher, 2021.

ELECTRIC, Schneider. **EcoStruxure™ Control Expert - System Bits and Words, Reference Manual**. Schneider Electric. 2020. Disponível em: https://www.se.com/ca/en/download/document/EIO0000002135/.

FLORES, Denisele. **Saneamento básico no Brasil**. Escola Educação. 2020. Disponível em: https://escolaeducacao.com.br/saneamento-basico/.

FOROUZAN, Behrouz A. **Comunicação de dados e redes de computadores**. [*S.l.*]: AMGH Editora, 2009.

GROOVER, Mikell P. **Fundamentals of modern manufacturing: materials, processes, and systems**. [*S.l.*]: John Wiley & Sons, 2010.

HANSSEN, Dag H. **Programmable logic controllers: a practical approach to IEC 61131-3 using CODESYS**. [*S.l.*]: John Wiley & Sons, 2015.

HIDRYCO. **Hidryco**. Hidryco Smart Water. 2023. Disponível em: https://www.hidryco.com.br/.

IEC61131-3, International Standard. **IEC61131**. IEC. 2013. Disponível em: https://webstore.iec.ch/publication/4552.

INSTRUMATIC. **Sistemas de Supervisão e Aquisição de Dados**. Instrumatic Medição e Controle. 2011. Disponível em: https://www.instrumatic.com.br/artigo/sistemas-de-supervisao-e-aquisicao-de-dados/.

MODBUS. MODBUS Application Protocol Specification V1. 1b3. **Modbus_Application_Protocol_V1_1b3. pdf**, 2012.

MORAES, Cıcero Couto; CASTRUCCI, Plınio. **Engenharia de automação industrial**. [*S.l.*]: LTC, 2007.

PETRUZELLA, Frank D. **Controladores lógicos programáveis**. [*S.l.*]: AMGH Editora, 2014.

RAMANATHAN, Ramakrishnan. The IEC 61131-3 programming languages features for industrial control systems. *In*: IEEE. 2014 world automation congress (wac). [*S.l.*: *s.n.*], 2014. P. 598–603.

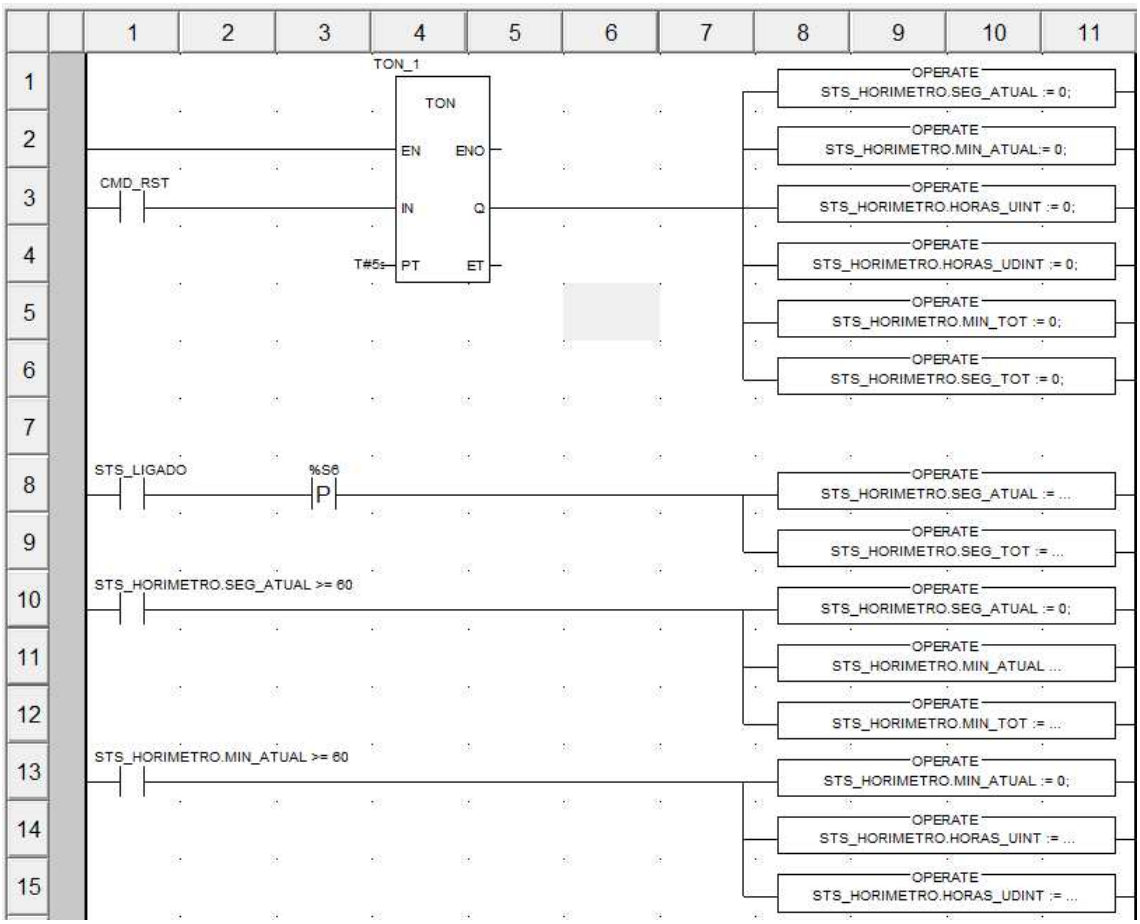
ROSÁRIO, João Maurıcio. **Automação industrial**. [*S.l.*]: Editora Baraúna, 2012.

SCHNEIDER. **EcoStruxure™ Control Expert - Operating Modes**. [*S.l.*], 2023. Disponível em: https://www.se.com/us/en/download/document/33003101K01000/. Acesso em: 1 dez. 2023.

APÊNDICE A – Lógica de programação do bloco funcional

Esse bloco foi desenvolvido com o intuito de mostrar o tempo que o programa está em modo *RUN*, além de mostrar algumas funções da programação Ladder e o uso do bloco temporizador TON.

Figura 34 – Desenvolvimento do bloco funcional Horímetro.



Anexos

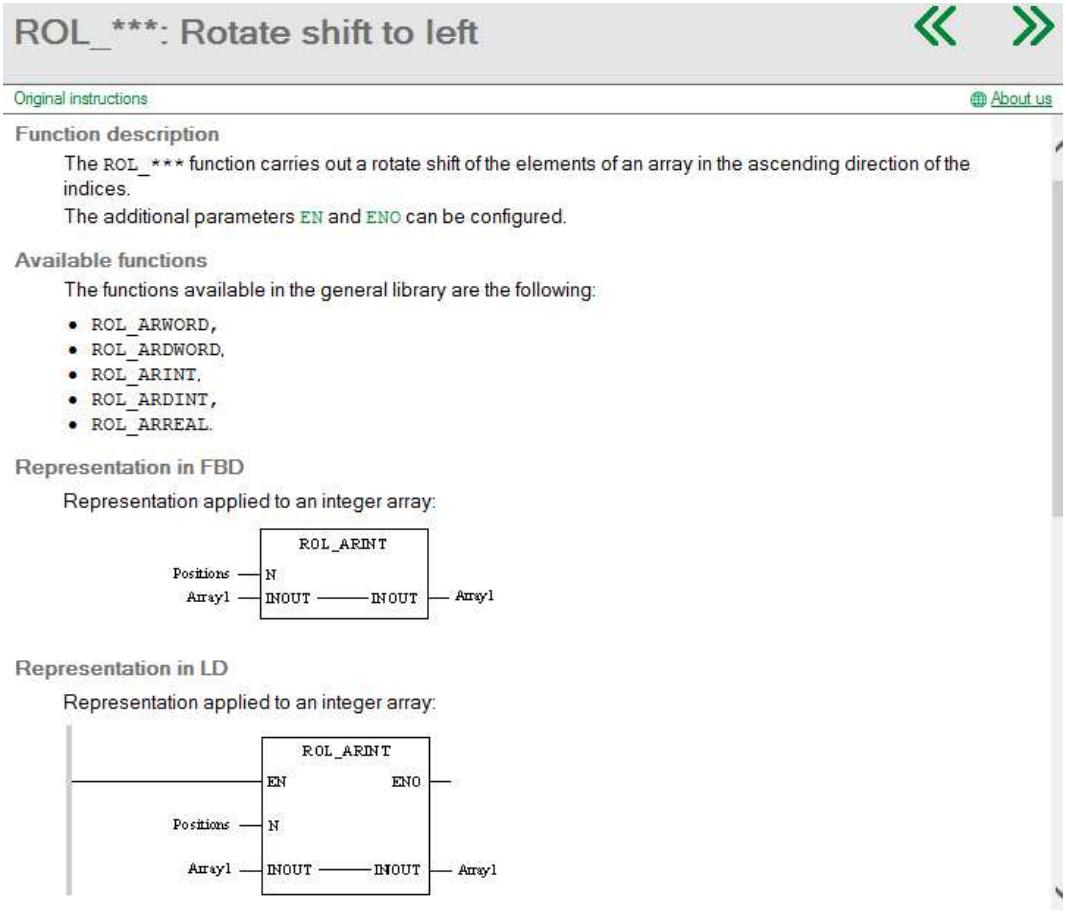
ANEXO A – System Bits & Words M340

Quadro 1 – System Bits & Words - M340.

SysM340	System Bits & Words	
%S0 coldstart	Cold Start	Normalmente em 0, este bit é definido como 1 por: • restauração de energia com perda de dados (falha da bateria), • o progarma do usuário, • o terminal, • uma troca de cartucho, • carregamento de programa.
%S1 warmstart	Warm Start	Normalmente em 0, este bit é definido como 1 por: • restauração de energia com salvamento de dados, • o programa do usuário, • o terminal, • uma troca de cartucho, • É redefinido para 0 pelo sistema no final do primeiro ciclo completo e antes da atualização das saídas.
%S4 TB10MS	tempo base 10 ms	Um temporizador interno regula a mudança de status deste bit. Ele é assíncrono em relação ao ciclo do CLP.
%S5 TB100MS	tempo base 100 ms	O mesmo que %S4.
%S6 TB1SEC	tempo base 1s	O mesmo que %S4.
%S7 TB1MIN	tempo base 1 min	O mesmo que %S4.
%S12 PLC RUNNING	CLP em execução	O mesmo que %S4.
%S13 1RSTSCANRUN	1º ciclo após mudar para RUN	Primeiro bit de digitalização. Normalmente definido para 0, este é definido como 1 pelo sistema durante o 1º ciclo da tarefa principal após o CLP entrar no modo RUN.

ANEXO B – Bloco rotate

Figura 35 – Descrição do bloco ROL_ARREAL.



Fonte: Software Control Expert.