

EDITORA CONHECIMENTO LIVRE

AUTOMAÇÃO DE PROCESSOS COM AS LINGUAGENS DEFINIDAS PELA NORMA IEC 61131-3, ATRAVÉS DOS AMBIENTES DE DESENVOLVIMENTO INTEGRADOS RSLOGIX 5000 E INTOUCH

Luiz Alberto Oliveira Lima Roque

LUIZ ALBERTO OLIVEIRA LIMA ROQUE

Automação de processos com as linguagens definidas pela norma IEC 61131-3, através dos ambientes de desenvolvimento integrados RSLogix 5000 e Intouch

1ª ed.

Piracanjuba-GO
Editora Conhecimento Livre
Piracanjuba-GO

Copyright© 2023 por Editora Conhecimento Livre

1ª ed.

Dados Internacionais de Catalogação na Publicação (CIP)

ROQUE, LUIZ ALBERTO OLIVEIRA LIMA
R638A Automação de processos com as linguagens definidas pela norma IEC 61131-3, através dos ambientes de desenvolvimento integrados RSLogix 5000 e Intouch / LUIZ ALBERTO OLIVEIRA LIMA ROQUE. – Piracanjuba-GO

Editora Conhecimento Livre, 2023

232 f.: il
DOI: 10.37423/2023.edcl731
ISBN: 978-65-5367-303-8
Modo de acesso: World Wide Web
Incluir Bibliografia

1. automação-industrial 2. diagrama-de-blocos-de-funções 3. grafcet 4. linguagem-ladder 5. rslogix-5000 I. ROQUE, LUIZ ALBERTO OLIVEIRA LIMA II. Título

CDU: 620

https://doi.org/10.37423/2023.edcl731

O conteúdo dos artigos e sua correção ortográfica são de responsabilidade exclusiva dos seus respectivos autores.

EDITORA CONHECIMENTO LIVRE

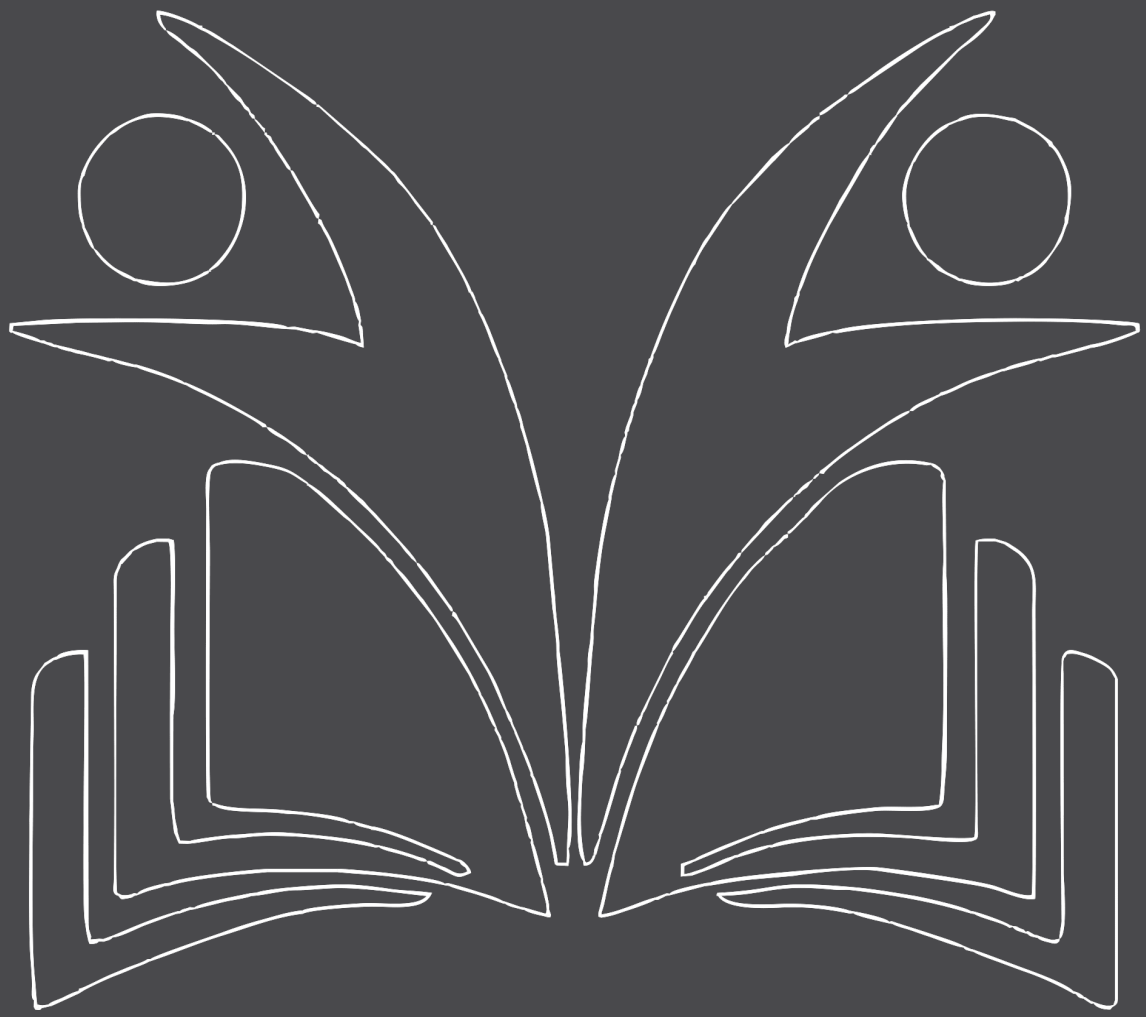
Corpo Editorial

MSc Edson Ribeiro de Britto de Almeida Junior
MSc Humberto Costa
MSc Thays Merçon
MSc Adalberto Zorzo
MSc Taiane Aparecida Ribeiro Nepomoceno
PHD Willian Douglas Guilherme
MSc Andrea Carla Agnes e Silva Pinto
MSc Walmir Fernandes Pereira
MSc Edisio Alves de Aguiar Junior
MSc Rodrigo Sanchotene Silva
MSc Wesley Pacheco Calixto
MSc Adriano Pereira da Silva
MSc Frederico Celestino Barbosa
MSc Guilherme Fernando Ribeiro
MSc. Plínio Ferreira Pires
MSc Fabricio Vieira Cavalcante
PHD Marcus Fernando da Silva Praxedes
MSc Simone Buchignani Maigret
Dr. Adilson Tadeu Basquerote
Dra. Thays Zigante Furlan
MSc Camila Concato
PHD Miguel Adriano Inácio
MSc Anelisa Mota Gregoleti
PHD Jesus Rodrigues Lemos
MSc Gabriela Cristina Borborema Bozzo
MSc Karine Moreira Gomes Sales
Dr. Saulo Cerqueira de Aguiar Soares
MSc Pedro Panhoca da Silva
MSc Helton Rangel Coutinho Junior
MSc Carlos Augusto Zilli
MSc Euvaldo de Sousa Costa Junior
Dra. Suely Lopes de Azevedo
MSc Francisco Odecio Sales
MSc Ezequiel Martins Ferreira
MSc Eliane Avelina de Azevedo Sampaio

Editora Conhecimento Livre
Piracanjuba-GO
2023

Automação de processos com as linguagens definidas pela norma IEC 61131-3, através dos ambientes de desenvolvimento integrados RSLogix 5000 e Intouch

10.37423/2023.edcl731



PREFÁCIO

A automação está presente nas mais variadas atividades da sociedade moderna, e engloba tópicos como: irrigação agrícola, colheita de flores, alimentação de animais, extração e armazenamento de ovos em granjas, envasamento de líquidos (leite, refrigerante, cerveja, sucos, água, produtos de limpeza, etc.), preenchimento e vedação de recipientes (sacos de arroz, feijão, amendoim, balas, caixas de chocolates, ervilha, pães de forma, milho, grãos, etc), fabricação de produtos (smartphones, automóveis, computadores, cimento, estruturas metálicas), além de diversos processos industriais, típicos de siderúrgicas, atividades navais, prospecção petrolífera, gestão ambiental, estação de tratamento de efluentes, pavimentação de estradas, aeronáutica e atividades relacionadas a segurança.

Um dos mais relevantes elementos utilizados na automação de processos é o Controlador Lógico Programável (CLP), que é um computador capaz de se adequar às adversidades características de ambientes industriais, como ruídos, umidade, altas temperaturas, poeira e demais intempéries, que dificilmente seriam suportadas por dispositivos de processamento domésticos ou de escritórios, como notebooks.

O CLP é composto de interfaces de entrada, saída, unidade de processamento de dados e comunicação de rede. Junto às entradas são interligados chaves ou sensores, que medirão as grandezas físicas do processo, e as converterão em sinais elétricos analógicos, que podem assumir qualquer valor.

Essas grandezas analógicas precisam ser enviadas a um conversor analógico digital, para ser representadas como números binários, possibilitando dessa forma a compreensão das mesmas pela Unidade Central de Processamento (UCP), uma vez que dispositivos digitais são projetados para tratarem de valores discretos.

O programa em execução na UCP do controlador lógico programável atualizará os valores das saídas controladas pelo CLP de acordo com as grandezas medidas em suas entradas.

As principais linguagens de programação de Controladores Lógicos Programáveis são definidas pela norma IEC 61131-3 como Ladder, Diagrama de Blocos de Funções, Texto Estruturado, Grafcet e Lista de Instruções.

Este Livro apresentará diversos processos, acompanhados de programas para automação dos mesmos, elaborados em linguagem Ladder. As linguagens de programação de CLP têm caráter

estritamente estático. Por esse motivo também serão desenvolvidos aplicativos para supervisão dos processos, devido à visão dinâmica do conjunto por eles proporcionados.

Considerando a relevância da empresa Rockwell Automation no cenário mundial de controle de processos, e a amplitude da série de CLPs Control Logix, que englobam quase todas as linguagens de programação definidas pela norma IEC 61131-3, o ambiente de desenvolvimento integrado (IDE) utilizado nesta publicação será o RSLogix 5000. Para supervisão de processos será utilizado o IDE Wonderware Intouch, comercializado pela multinacional francesa de automação, a Schneider Electric. Esta publicação pretende fornecer aos leitores elementos que possibilitem desenvolver conhecimentos básicos em automação de processos, no que tange aos aspectos estáticos e dinâmicos dos mesmos.

O autor é Professor do Instituto Federal Fluminense – Campus Macaé, atuando desde 2008 nos segmentos técnicos de Eletrônica e Automação Industrial, bem como nos bacharelados em Engenharia de Controle e Engenharia Elétrica, cursos onde orienta projetos de pesquisa e trabalhos de conclusão de curso, e também já exerceu funções de coordenação por 10 anos.

Luiz Alberto é doutorando em Ciências da Educação, Mestre em Engenharia de Computação e Bacharel em Engenharia Elétrica. Concluiu especializações em Automação, Computação, Sistemas, Mecatrônica, Ciência de Dados, Eletromecânica, Redes de Computadores Ensino de Ciências, Planejamento e Gestão de Educação a Distância, Educação de Jovens e Adultos, Engenharia Elétrica, Telecomunicações, Eletrotécnica, Tecnologia, Inteligência Artificial, Controle de Processos, Energias e Sustentabilidade, Informática, Engenharia Robótica, Perícia, Avaliações, Educação, Prática Pedagógica, Sociologia, Trabalho, Geomática, Engenharia Ambiental, Segurança, Projetos, Saúde, Física, Matemática, Libras, Docência, Taekwondo, Supervisão, Tutoria, Administração, Pedagogia, Ensino Técnico, Gestão Pública, Escolar e Cidades.

Sumário

1 INTRODUÇÃO **6**

1.1- ATIVAÇÃO **7**

2 PROGRAMAS **10**

2.1 RSLOGIX EMULATE 5000 ..... **10**

2.2 RSLINX **13**

2.3 RSLOGIX 5000 ..... **15**

3 PROGRAMANDO EM LADDER E SISTEMA SUPERVISÓRIO IN TOUCH **18**

3.1 PROBLEMAS DE AUTOMAÇÃO INDUSTRIAL EM LINGUAGEM LADDER **20**

3.3 PROBLEMAS DE AUTOMAÇÃO EM DIAGRAMA DE BLOCOS DE FUNÇÕES **126**

EXERCÍCIOS **129**

3.4 PROBLEMAS DE AUTOMAÇÃO EM LINGUAGEM GRAFCET **199**

4 CONSIDERAÇÕES FINAIS **227**

1 INTRODUÇÃO

A primeira revolução industrial foi impulsionada por James Watt, inventor escocês que aperfeiçoou o rendimento da máquina a valor de Thomas Newcomen em 1769, através da introdução da câmara de condensação externa, de acordo com Roque (2018).

A partir da descoberta de James Watt, a humanidade passou a ter condições de produzir em larga escala pela primeira vez na história, possibilitando o surgimento de diversos equipamentos e empreendimentos a vapor, como fábricas, navios, locomotivas e máquinas de tear.

O século XIX foi marcado por diversas descobertas científicas, como a Lei da Indução de Faraday, que possibilitou a geração de energia elétrica, juntamente com a descoberta do efeito semicondutor pelo mesmo cientista, possibilitando o desenvolvimento de diodos.

Roque (2018) destaca ainda outro relevante cientista do século XIX, Thomas Edison, que descobriu o efeito termiônico, caracterizado pela condução de elétrons no vácuo, levando ao descobrimento das válvulas.

Os primeiros dispositivos eletrônicos do século XX eram constituídos por válvulas, relés e diodos, dentre os quais se destacam os computadores desenvolvidos durante a segunda guerra mundial, caracterizados por gigantescas dimensões, e escasso poder de processamento.

William Bradford Shockley era um físico estadunidense, que revolucionou a indústria de computação, adicionando uma terceira junção ao diodo, dando origem a um novo dispositivo eletrônico, o transistor. Com essa nova invenção, os computadores ampliaram seu poder de processamento e reduziram suas dimensões, uma vez que transistores trabalham com tensões e correntes muito menores que válvulas, e não precisam esperar seu aquecimento para funcionarem, como ocorre nos dispositivos inventados por Thomas Edison.

A álgebra de boole foi desenvolvida pelo matemático inglês George Boole, em 1847, que considerava três operações lógicas (E, OU e NÃO) aplicadas sobre conjunto de números binários, admitem somente dois valores, 0 (zero) ou 1 (um). A utilização da álgebra de boole aliada aos transistores possibilitou revolução ímpar nos computadores, tornando-os menores, mais rápidos e leves.

A lógica binária foi fundamental à compactação das dimensões dos equipamentos computacionais, uma vez que a necessidade de fios nos barramentos dos dispositivos foi bastante diminuída, devido ao fato de se deixar de utilizar numeração decimal no hardware.

O circuito integrado (C.I.) foi inventado por Jack Kilby em 1958, conforme Roque (2018), tratando-se de pastilha semicondutora com pequenas dimensões, capaz de armazenar grande quantidade de dispositivos elétricos e eletrônicos, como diodos, transistores, resistores, capacitores e indutores.

Em 1971 foi desenvolvido o primeiro microprocessador programável, o Intel 4004, que compreendia 2300 transistores, unidade central de processamento (UCP), registradores de 4 bits, 46 instruções, Unidade Lógica e Aritmética (ULA), Unidade de Controle (UC) e memória, seguindo a arquitetura computacional de Jon Von Newman, elaborada pelo matemático húngaro em 1949, segundo Roque (2018).

Com aprimoramento das tecnologias de fabricação de semicondutores e circuitos integrados (CI), a década de 1980 caracterizou-se pelo início da manufatura de CI's em larga escala (LSI – Large Scale Integration), compreendendo milhares de transistores numa mesma pastilha, viabilizando o surgimento de microcomputadores, dentre os quais se destacam o computador pessoal da empresa norte-americana IBM (IBM– PC).

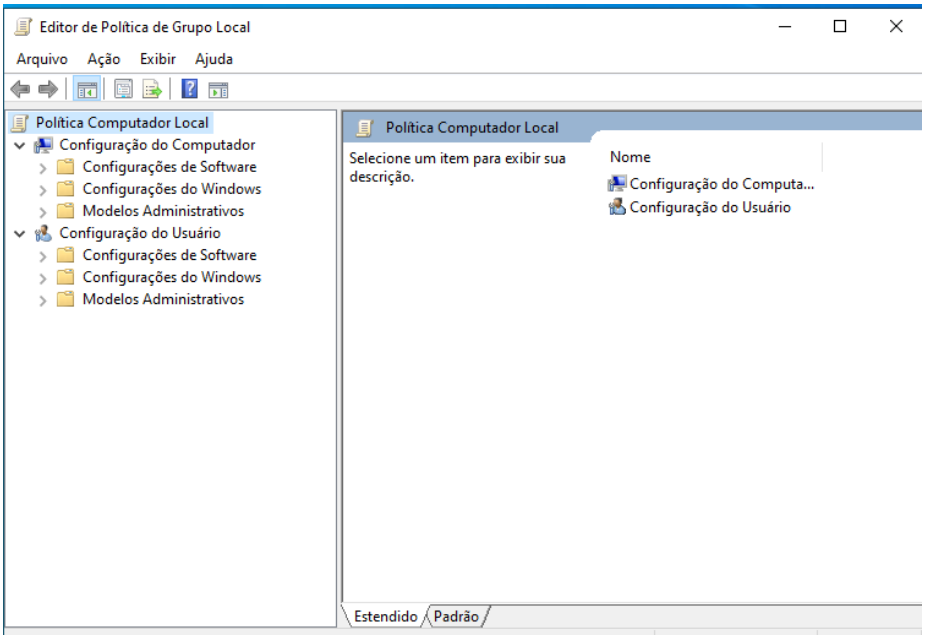
A partir de 1980 surgiram sistemas operacionais para computadores pessoais, como Windows (desenvolvido pela microsoft), Linux e o Mac OS (sistema operacional da Apple).

O RSLogix 5000 é um software da Rockwell Automation para programação de CLPs da Linha CompactLogix, que diferentemente da linha Micrologix, é capaz de programar em Grafcet (ou Sequential Function Chart - SFC) e Texto Estruturado (Structured Text - ET), além das linguagens Ladder e Lista de Instruções (Instruction List - IL). Este trabalho permite compreender o RSLogix 5000, algumas de suas funcionalidades e em seguida, implementar a automação de processos dosadores juntamente com sistemas supervisórios, que é o objetivo geral deste projeto.

1.1- ATIVAÇÃO

O programa para ativação dos softwares da família Rockwell é do tipo em lotes, podendo ser carregado automaticamente durante a inicialização do windows 10, através do aplicativo editor de política local do sistema operacional da microsoft, conhecido por gpedit.msc. Após abrir o gpedit.msc, surgirá a seguinte tela (fig. 1):

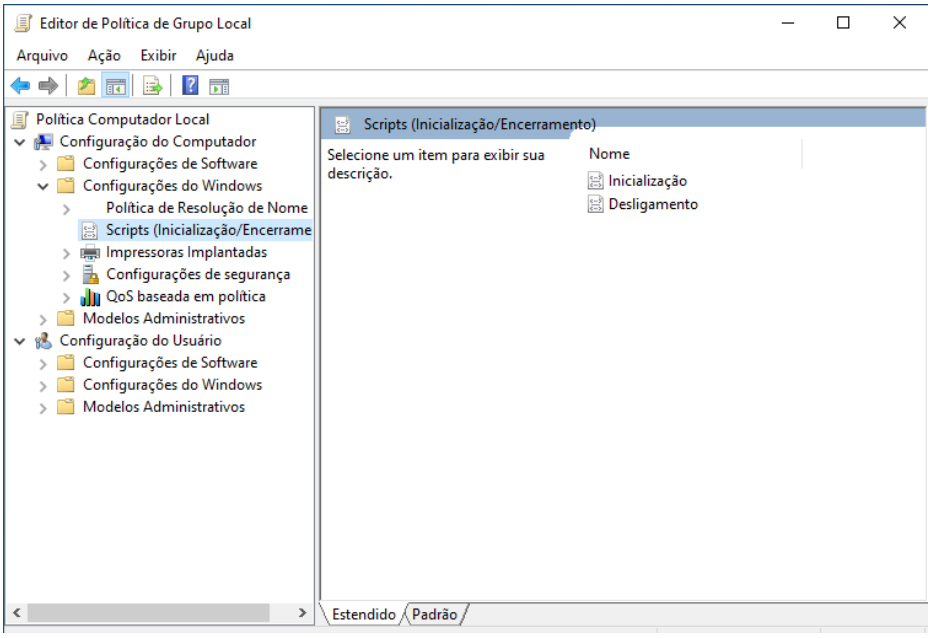
Figura 1 – Tela do editor gpedit.msc



Fonte: Própria.

O próximo passo consiste em clicar duas vezes sobre a pasta configurações do windows, que se apresenta como subconjunto das configurações do computador, para acessar os scripts de inicialização, conforme seguinte figura (fig. 2):

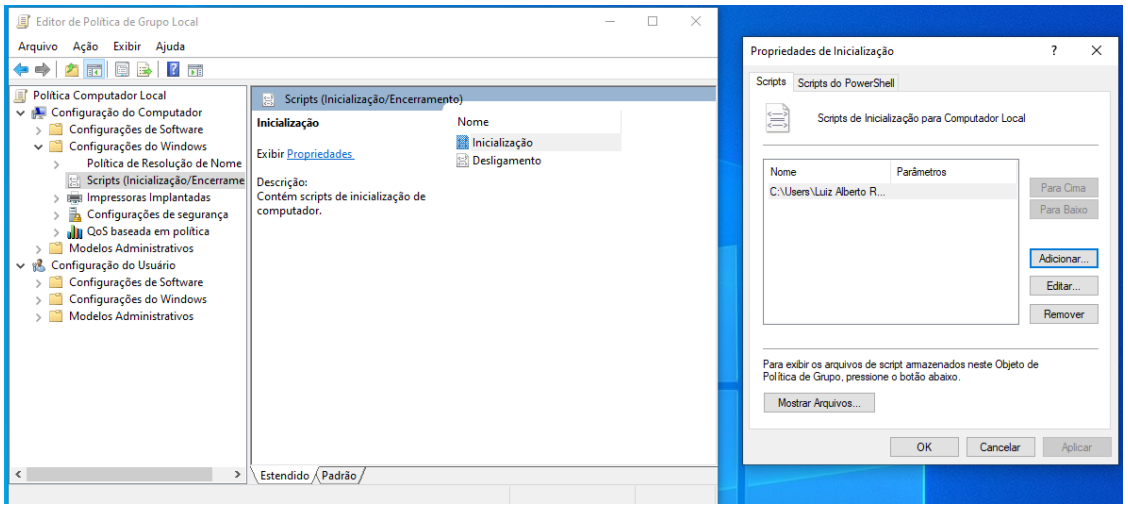
Figura 2 – Tela do editor gpedit.msc



Fonte: Própria.

Em seguida, clica-se duas vezes sobre a opção de inicialização, mostrada na parte direita da tela, referente aos possíveis scripts (que são de inicialização ou encerramento). Isso proporcionará o surgimento de outra tela, para adicionar o script desejado (fig. 3):

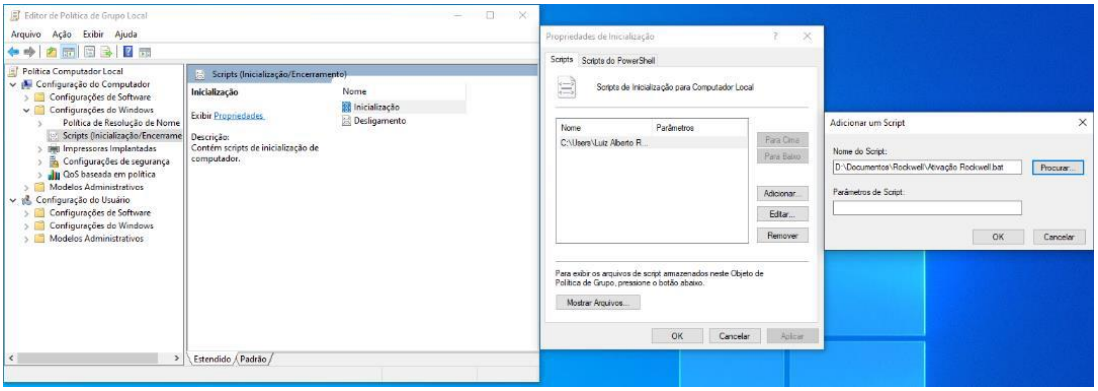
Figura 3 – Tela do editor gpedit.msc



Fonte: Própria.

Pressionando-se o botão adicionar, a tela adquire a seguinte configuração, possibilitando a localizar o script que se deseja adicionar, clicando a seguir em OK (fig. 4).

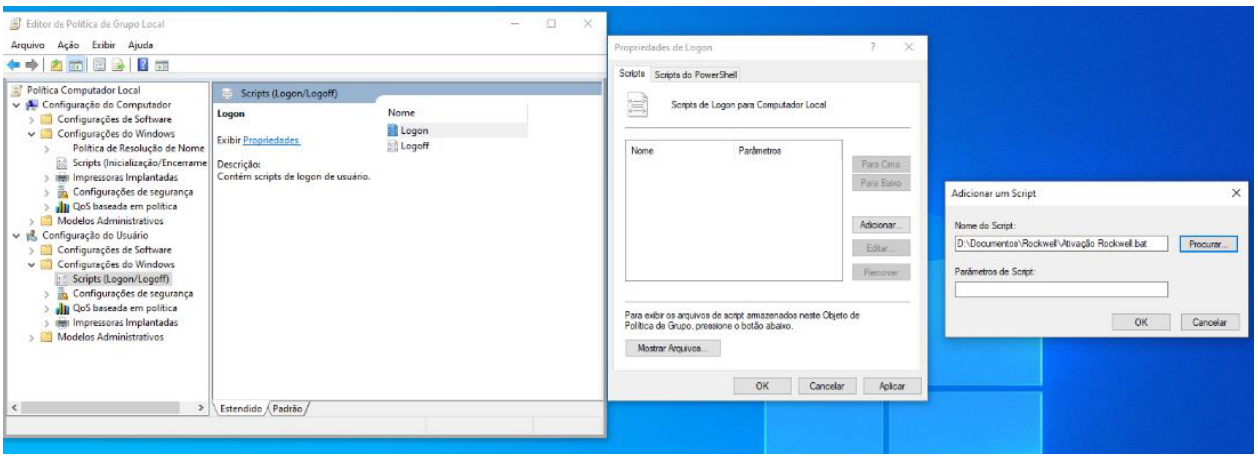
Figura 4 – Tela do editor gpedit.msc



Fonte: Própria.

Recomenda-se também adicionar o aplicativo de ativação nas configurações de Logon do usuário, conforme a tela apresentada na figura 5. Isso é realizado porque há usuário que deixam seus notebooks em operação constante e, como a ativação tem o prazo de sete dias para expirar, precisará ser renovada sempre que houver logon, ou inicialização do windows.

Figura 5 – Tela do editor gpedit.msc



Fonte: Própria.

2 PROGRAMAS

Neste trabalho foram utilizados três softwares da Rockwell Automation para programação e simulação de CLPs da linha CompactLogix: RSLogix Emulate 5000, RSLinx e RSLogix 5000; além do programa de sistemas supervisórios Wonderware Intouch.

O RSLogix Emulate 5000 permite simular um CLP para programação. Neste programa é possível ter um CLP virtual para experimento e teste das lógicas de programação.

O RSLogix 5000 é o programa onde são desenvolvidas todas as lógicas de programação e definidas as variáveis de processo, neste programa também se executa o programa carregado no CLP ou simulado no software RSLogix Emulate 5000.

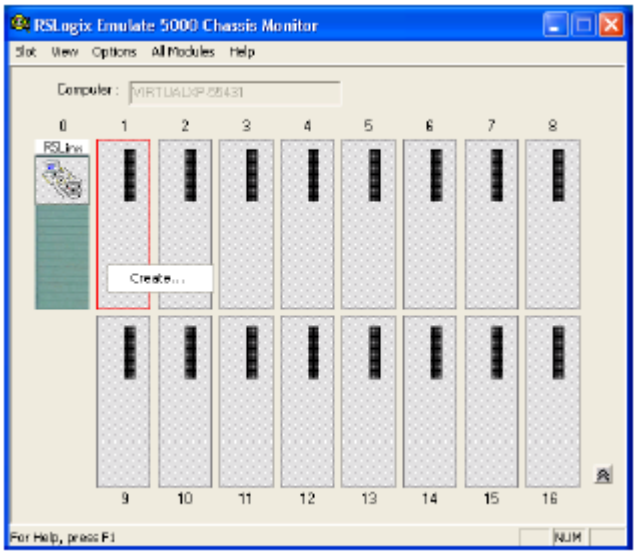
Já o software RSLinx é responsável pela comunicação de dados entre o RSLogix 5000 e o CLP físico, ou o virtual simulado no Emulate, e também com sistemas supervisórios.

Antes de iniciar a programação da lógica de controle de processos, deve-se executar os softwares na sequência a seguir, de acordo com o roteiro apresentado passo a passo.

2.1 RSLOGIX EMULATE 5000

Inicia-se esse programa, clicando com o botão direito do mouse no ‘slot 1’, conforme figura 6.

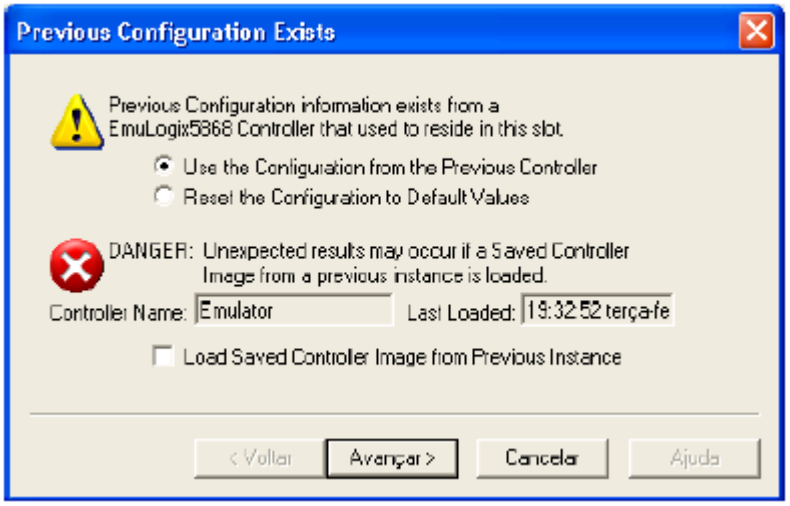
Figura 6 – Tela do RSLogix Emulate 5000



Fonte: Própria.

Após selecionar a opção de criar, escolhe-se o tipo de módulo '*Emulator RSLogix Emulate 5000 Controller'* e pressiona-se o botão 'OK', fazendo com que surja a janela abaixo, na qual se clica com o mouse sobre a opção 'Avançar', de acordo com a figura 7.

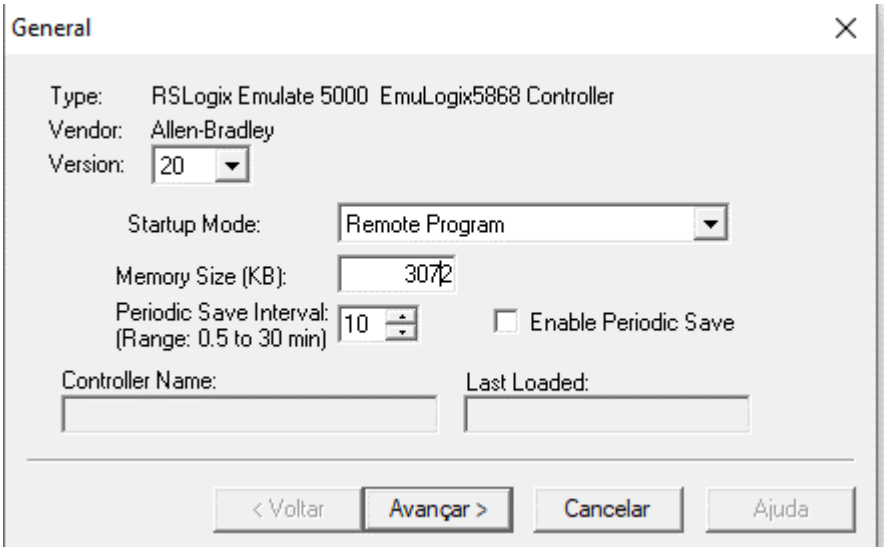
Figura 7 – Tela do RSLogix Emulate 5000



Fonte: Própria.

Em seguida deve-se verificar a versão do modulo, pois deve ser compatível com o projeto que será criado no RSLogix 5000. Neste caso a versão é a 20, conforme configurado na próxima imagem, onde se seleciona novamente a opção 'avançar', clicando em seguida sobre o botão '*Concluir*', de acordo com a figura 8.

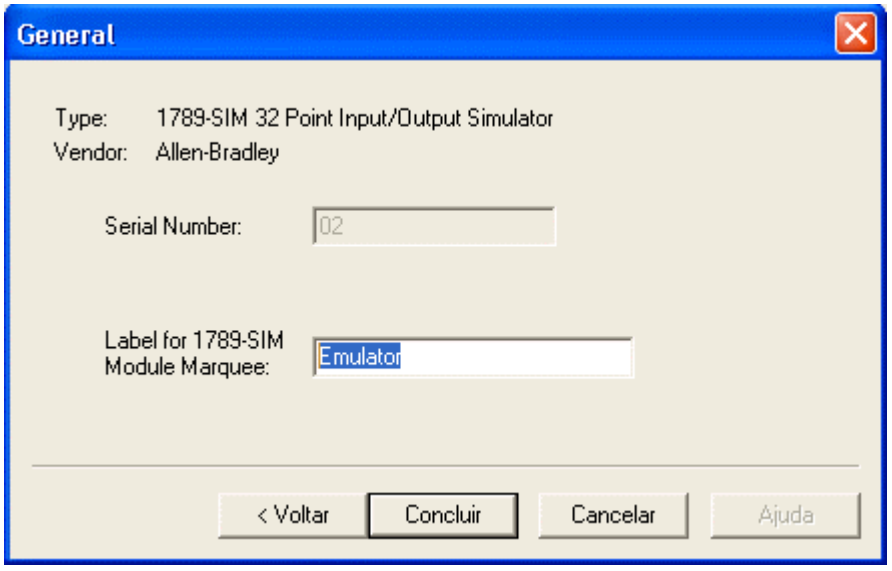
Figura 8 – Tela do RSLogix Emulate 5000



Fonte: Própria.

Agora, no '*slot 2*' do RSLogix Emulate 5000, seleciona-se o tipo de módulo de simulação de entradas e saídas '*1789 SIM 32 Point Input/Output Simulator*'. Logo após, clica-se em '*Avançar*' e na próxima janela, atribui-se um nome ao módulo. Neste caso o módulo foi nomeado '*Emulator*', conforme figura 9, na qual se posiciona o mouse sobre o botão '*Concluir*', para em seguida clicar nele.

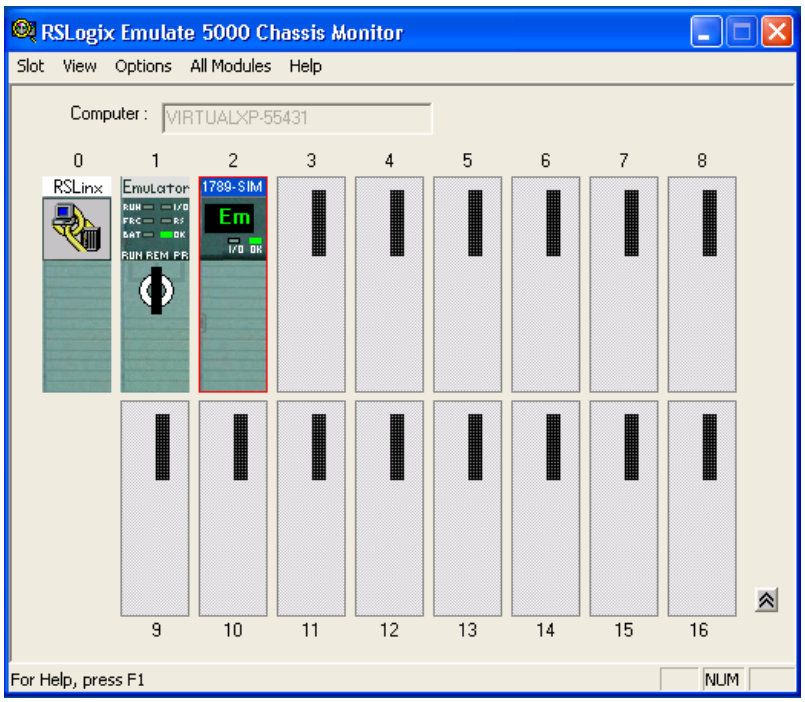
Figura 9 – Tela do RSLogix Emulate 5000



Fonte: Própria.

Ao fim desta etapa o RSLogix Emulate 5000 estará desta forma (fig.10):

Figura 10 – Tela do RSLogix Emulate 5000



Fonte: Própria.

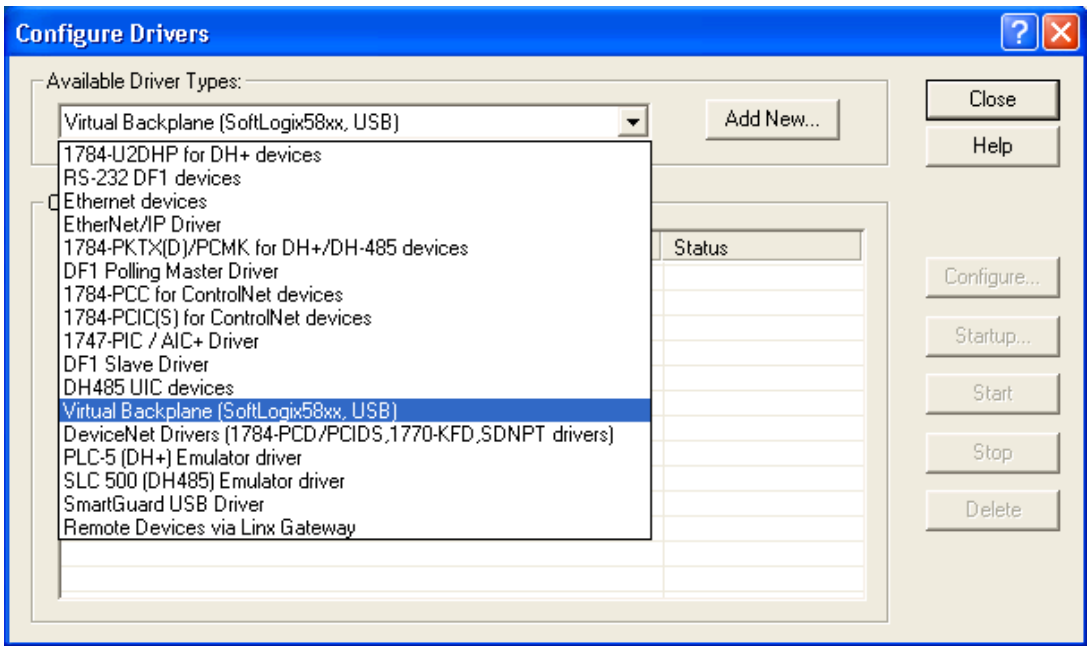
Clica-se com o botão direito do mouse sobre o ícone do RSLinx no *'Slot 0',* ou abre-se o RSLinx através de atalho localizado na Área de Trabalho, ou no Menu Iniciar, para iniciar a próxima etapa.

IMPORTANTE: Antes de desativar o RSLogixEmulate 5000, é fundamental remover os módulos adicionados aos slots 1 e 2, respectivamente associados ao emulador e ao CLP que será simulado. Caso o aplicativo seja desligado antes de remover esses módulos, na próxima vez em que for carregado e se efetuar downloads de códigos desenvolvidos no RSLogix 5000, há grande risco de ocorrer o erro *Controller is not in correct mode*, cuja solução só pode ocorrer com desinstalação e posterior instalação do software RSLogixEmulate 5000.

2.2 RSLINX

Na janela do RSLinx, deve-se pressionar as opções de menu identificadas por '*Communications*' => '*Configure Drivers...*' e, na janela que abrir, seleciona-se a opção conforme a imagem abaixo, clicando em '*Add New*' (fig. 11):

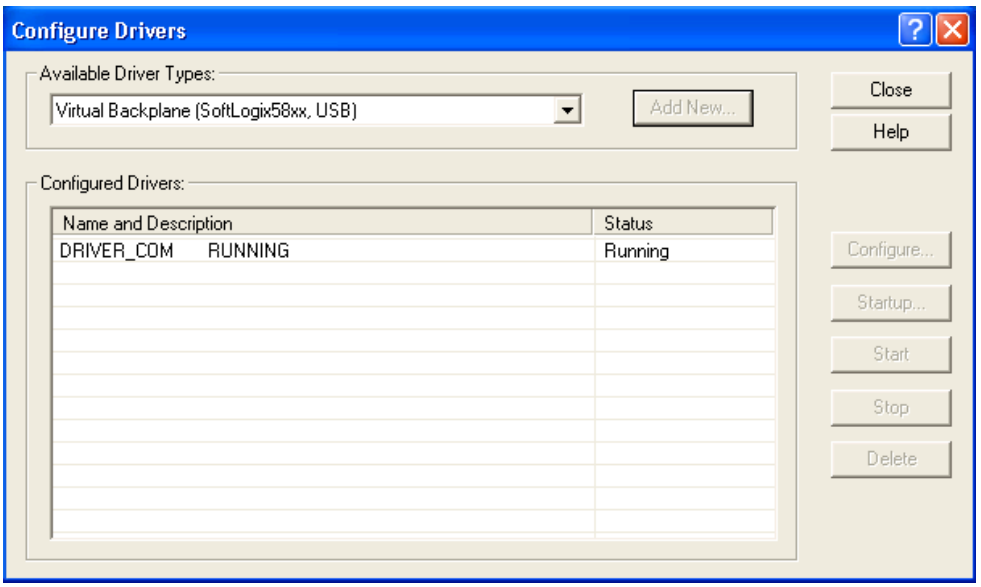
Figura 11 – Tela do RSLinx



Fonte: Própria.

Logo após surgirá uma tela solicitando atribuição de nome para este driver. Neste exemplo, designou-se ao driver o nome '*DRIVER_COM*'. Na janela de configuração que abrirá, deve-se assegurar que a opção '*Station Number*' esteja em 0, referente ao '*Slot 0*' em que RSLinx se encontra no Emulate. Após esta etapa, a configuração de drivers estará conforme figura 12.

Figura 12 – Tela do RSLinx



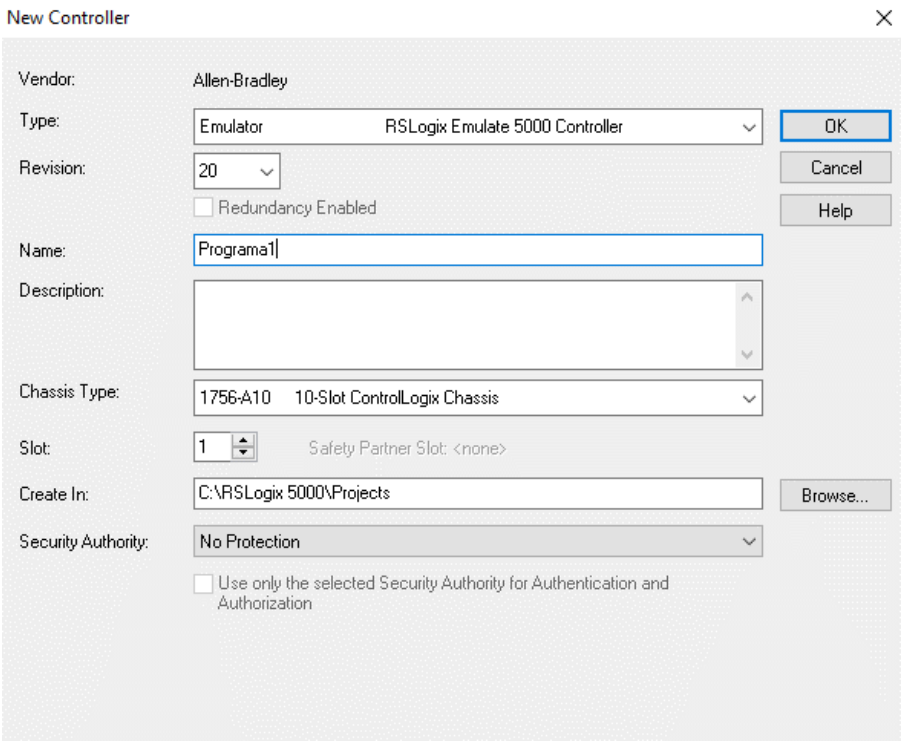
Fonte: Própria.

Seleciona-se "*Close*", para em seguida abrir o RSLogix 5000.

2.3 RSLOGIX 5000

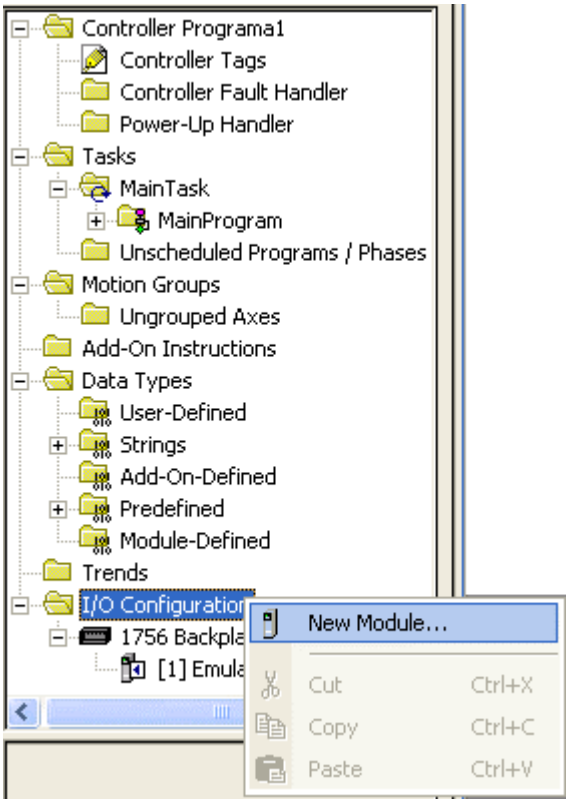
Ao abrir este software, na opção "*File*" seleciona-se "*New*". Nesta nova janela nomeia-se o programa, designando o mesmo como "*Programa1*". Seleciona-se no menu Type a opção "*Emulator RSLogix Emulate Controller*", conforme imagem a seguir. Pressiona-se a seguir o botão "*OK*" (fig. 13).

Figura 13 – Tela do RSLogix 5000



Fonte: Própria.

Figura 14 – Tela do RSLogix 5000

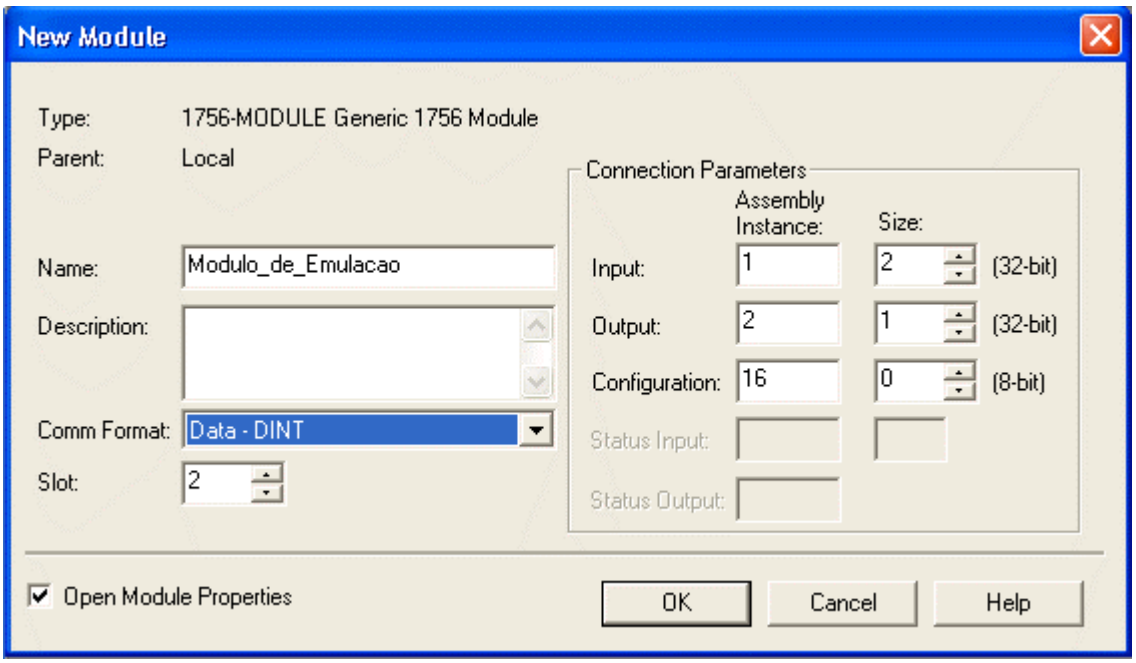


Fonte: Própria.

Clica-se com o botão direito do mouse na pasta "*I/O Configuration*", selecionando "*New Module*", de acordo com a figura 14. Na janela que abrir, em "*Other*" clica-se duplamente na opção que abrir. O próximo passo consiste em configurar o módulo de emulação das entradas e saídas.

Procede-se à configuração conforme a figura 15, clicando-se em "*OK*" nesta janela e na próxima que abrir.

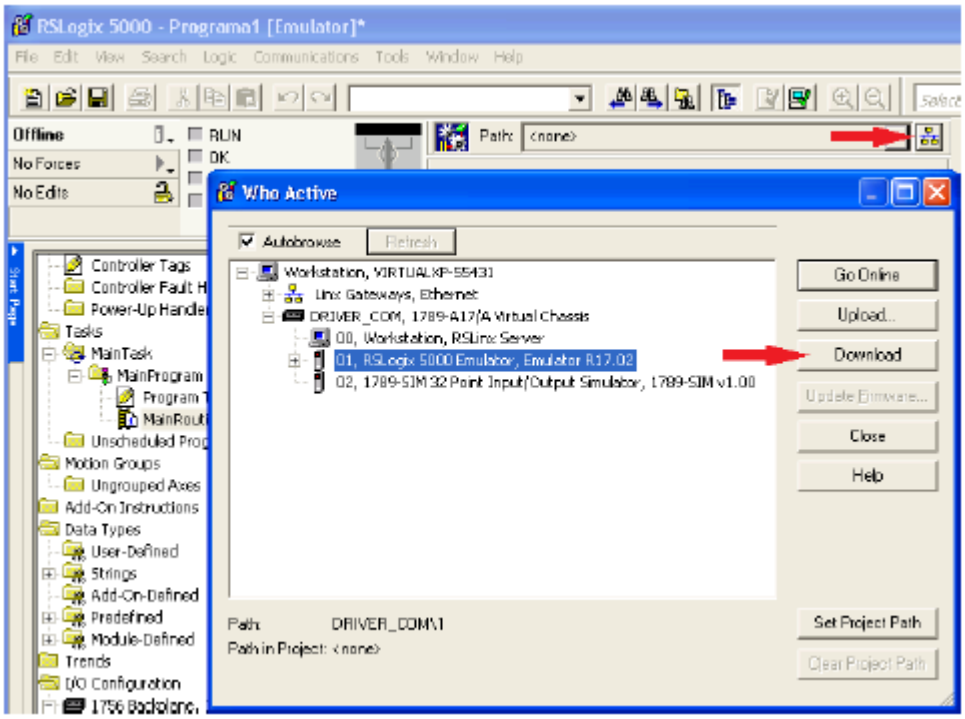
Figura 15 – Tela do RSLogix 5000



Fonte: Própria.

Clica-se em *"Who Active"*, para selecionar a opção conforme imagem e pressionar o botão "*Download*". No aviso que aparecer deve-se clicar em "*Download*" novamente (fig. 16).

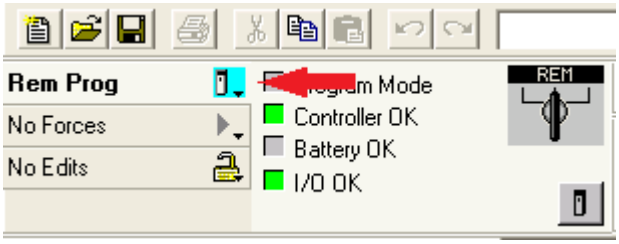
Figura 16 – Tela do RSLogix 5000



Fonte: Própria.

Clica-se sobre o ícone azul, onde se seleciona a opção "*Run Mode*" (fig. 17). Após esta etapa inicia-se de fato a programação.

Figura 17 – Tela do RSLogix 5000



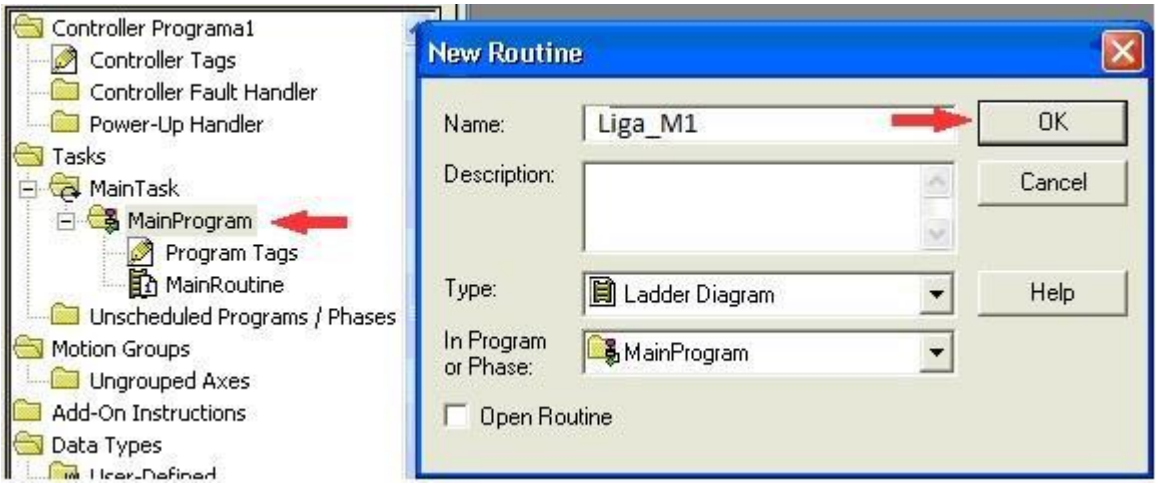
Fonte: Própria.

A rotina principal (*Main Routine*) é programada com a linguagem Ladder, porém nesta rotina não serão desenvolvidos programas para controlar processos diretamente, uma vez que a mesma é utilizada para chamar outras rotinas através da instrução JSR, que poderão ser programadas nas demais linguagens especificadas pela norma IEC 61131-3. A chamada à instrução JSR será compreendida no próximo tópico.

3 PROGRAMANDO EM LADDER E SISTEMA SUPERVISÓRIO IN TOUCH

Inicialmente, cria-se uma nova rotina. Clica-se com o botão direito do mouse na pasta "*MainProgram*", onde se seleciona a opção "*New Routine*…". Em seguida, define-se o tipo da linguagem de programação como sendo "Ladder Diagram", nomeando a nova rotina (neste caso ela foi chamada de Liga_M1) e clicando em "OK". Para abrir a nova rotina, efetua-se duplo clique no ícone que aparecerá na pasta "*MainProgram*" (fig. 18).

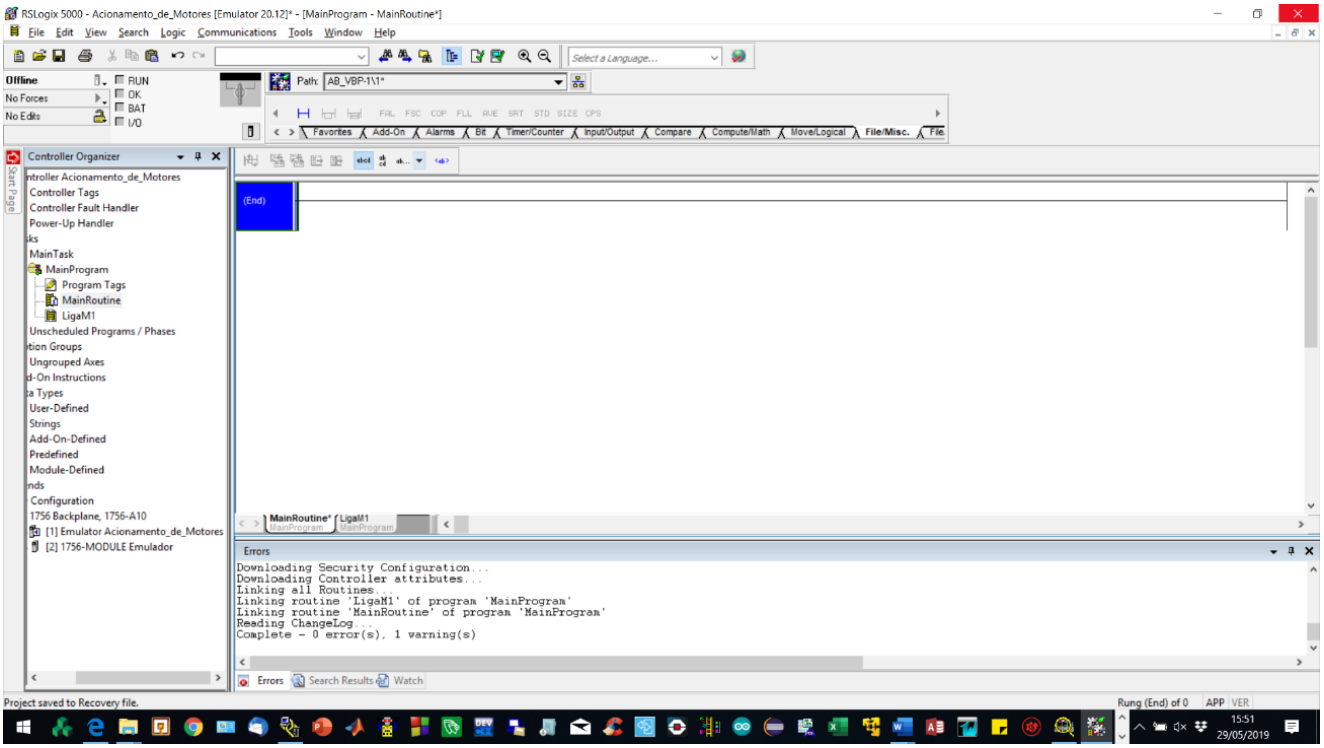
Figura 18 – Tela do RSLogix 5000



Fonte: Própria.

Aparecerá a seguinte figura, onde se vê a rotina LigaM1, na qual devem ser inseridos os elementos do programa Ladder (fig. 19).

Figura 19 – Início da programação em linguagem Ladder

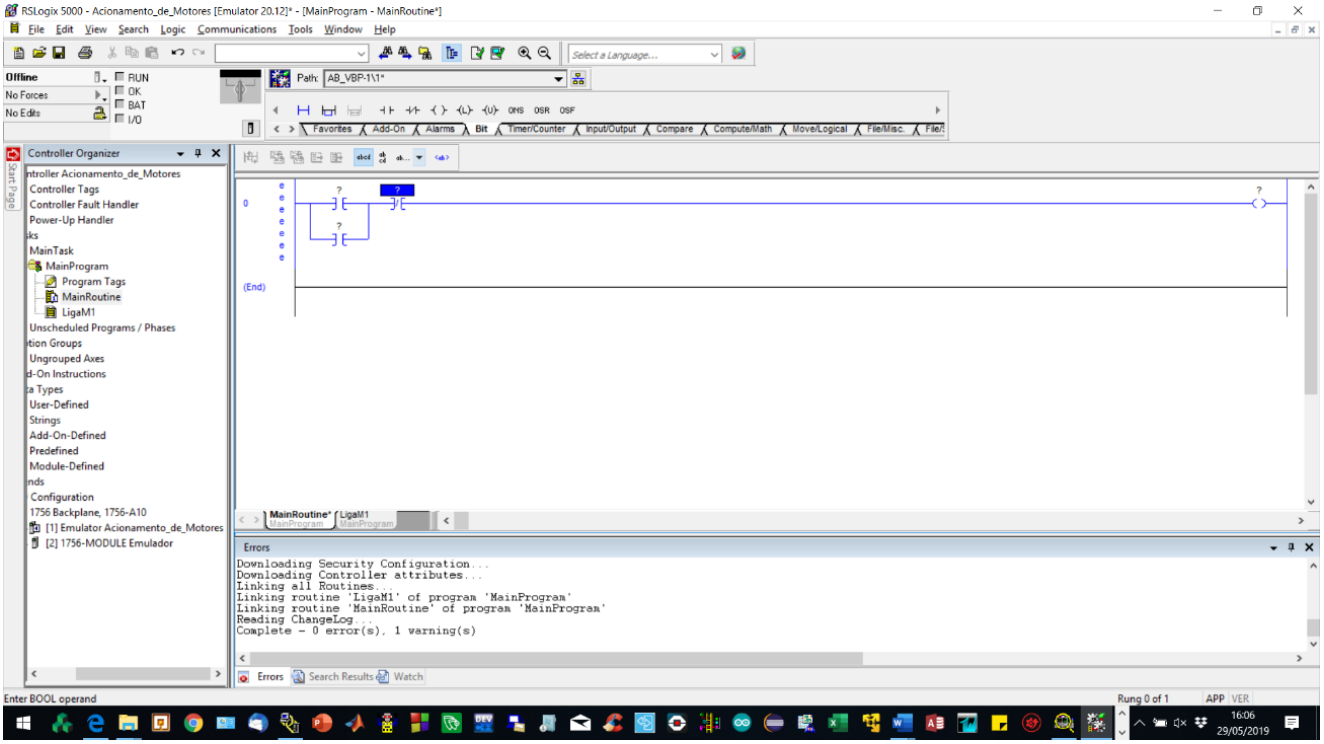


Fonte: Própria.

3.1 PROBLEMAS DE AUTOMAÇÃO INDUSTRIAL EM LINGUAGEM LADDER

1. Considera-se agora a elaboração de um programa para acionar e desligar um motor, cujo arranjo de contatos e saída fica de acordo com a figura 20.

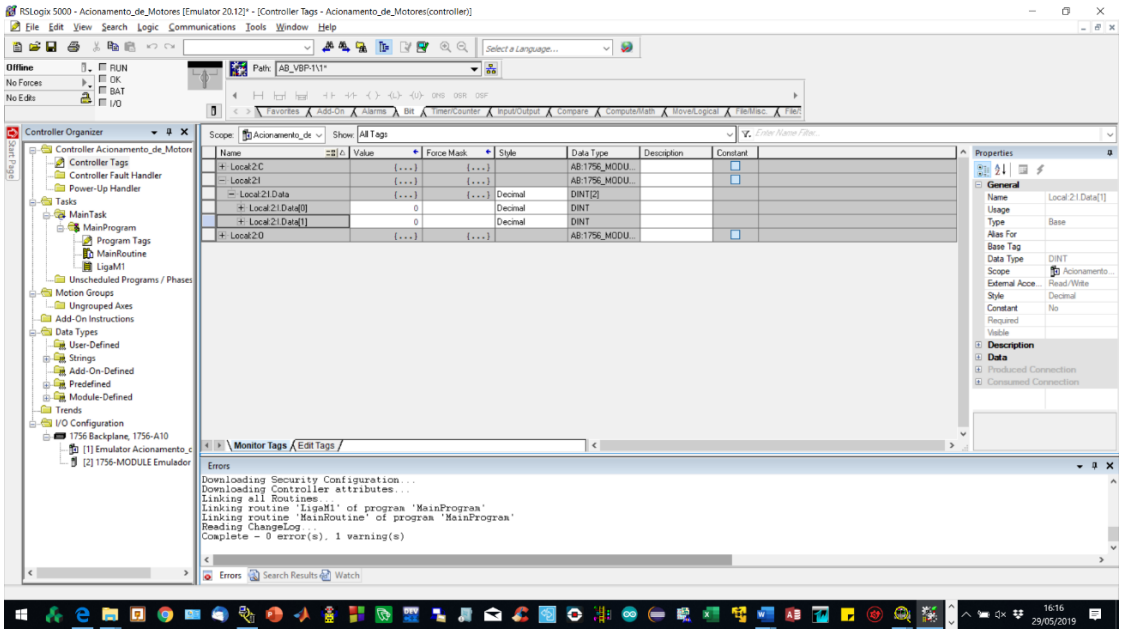
Figura 20 – Início da programação em linguagem Ladder



Fonte: Própria.

Para proceder ao endereçamento dos elementos integrantes do programa Ladder, é preciso clicar sobre *Controller Tags*, para obter a tabela de endereços de entrada e saída (fig. 21).

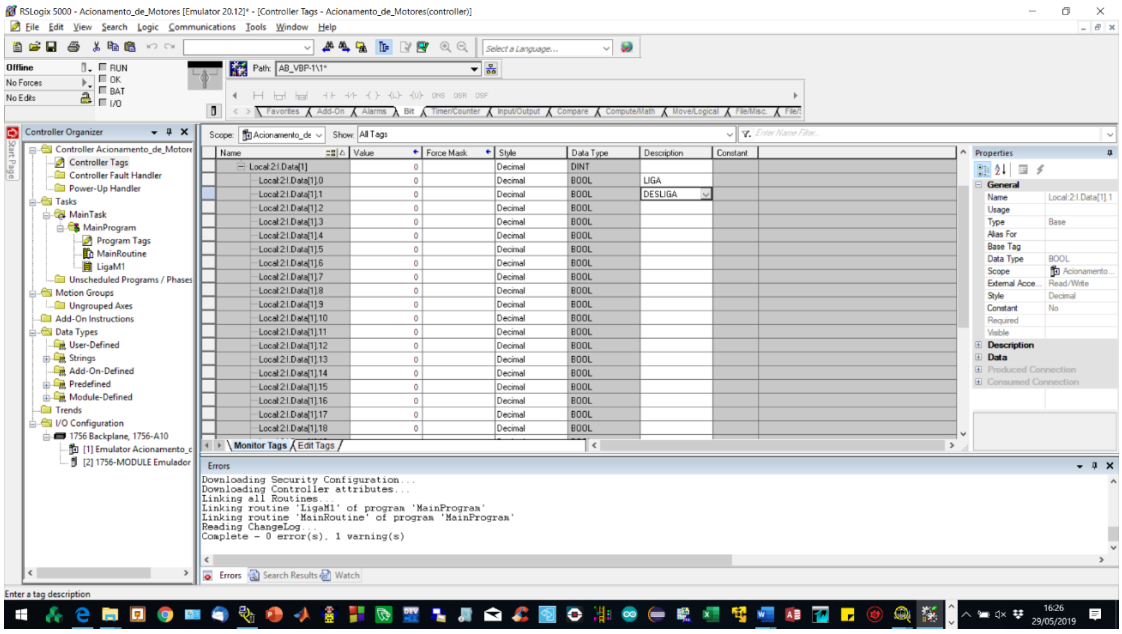
Figura 21 – Tabela de endereços



Fonte: Própria.

A partir deste ponto, basta expandir os endereços disponíveis para o segundo byte de entrada, conforme exposto na figura 22.

Figura 22 – Tabela de endereços



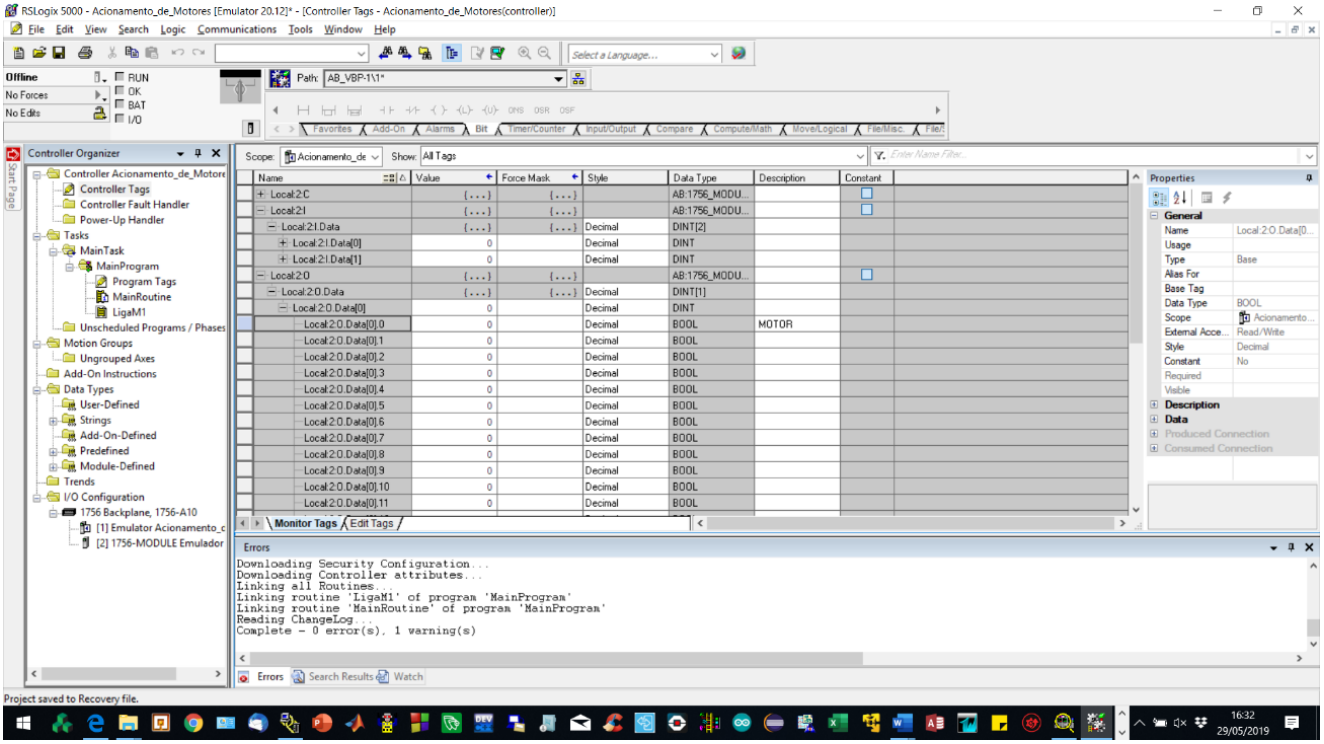
Fonte: Própria.

O próximo passo consiste em copiar o endereço do contato normalmente aberto, que representará o botão liga, para colar o mesmo na interrogação do programa Ladder.

Pode-se também digitar a descrição de cada contato na coluna description. Por exemplo, aos endereços Local:2:I.Data[1].0 e Local:2:I.Data[1].1 atribuiu-se a descrição LIGA e DESLIGA.

Em seguida, expandem-se os endereços de saída, para obter a figura 23, onde se insere a descrição do dispositivo controlado, que pode ser associado a Local:2:O.Data[0].0

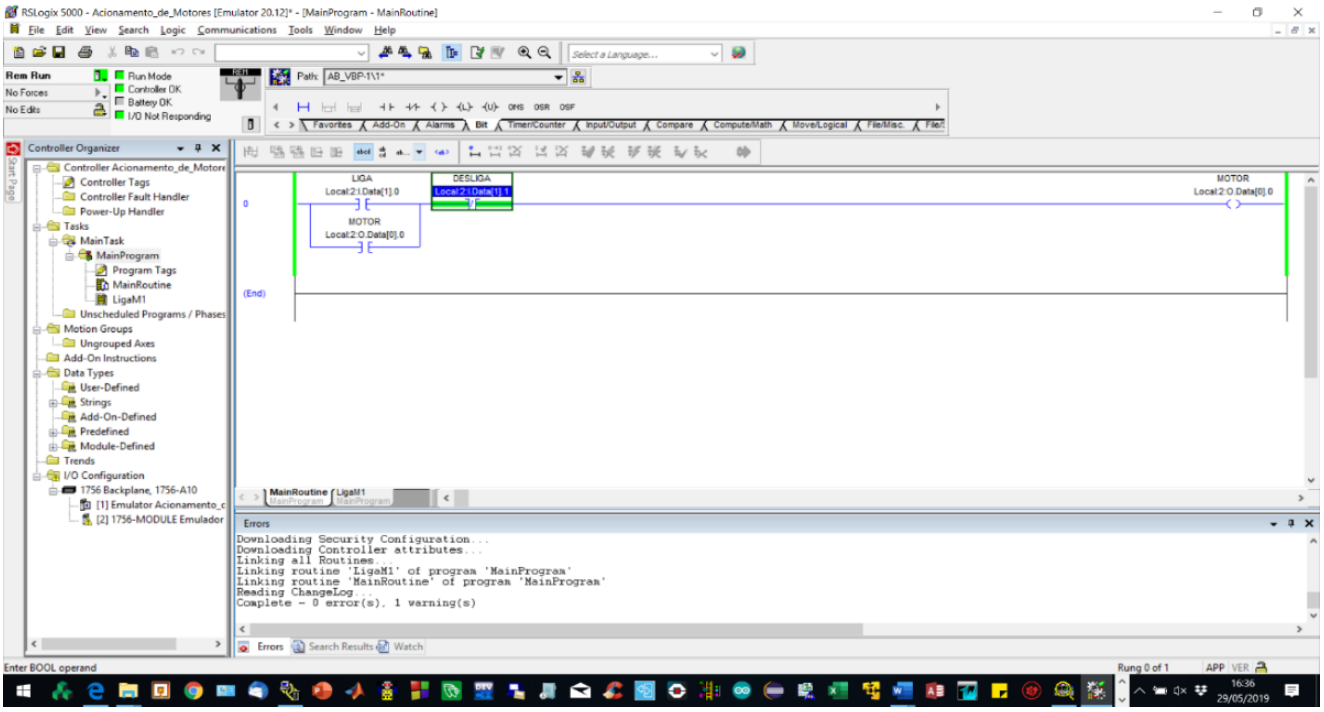
Figura 23 – Tabela de endereços



Fonte: Própria.

Com os endereços inseridos no programa ladder, o código adquire as seguintes características, com o download já efetuado, após a devida verificação de erros (fig. 24):

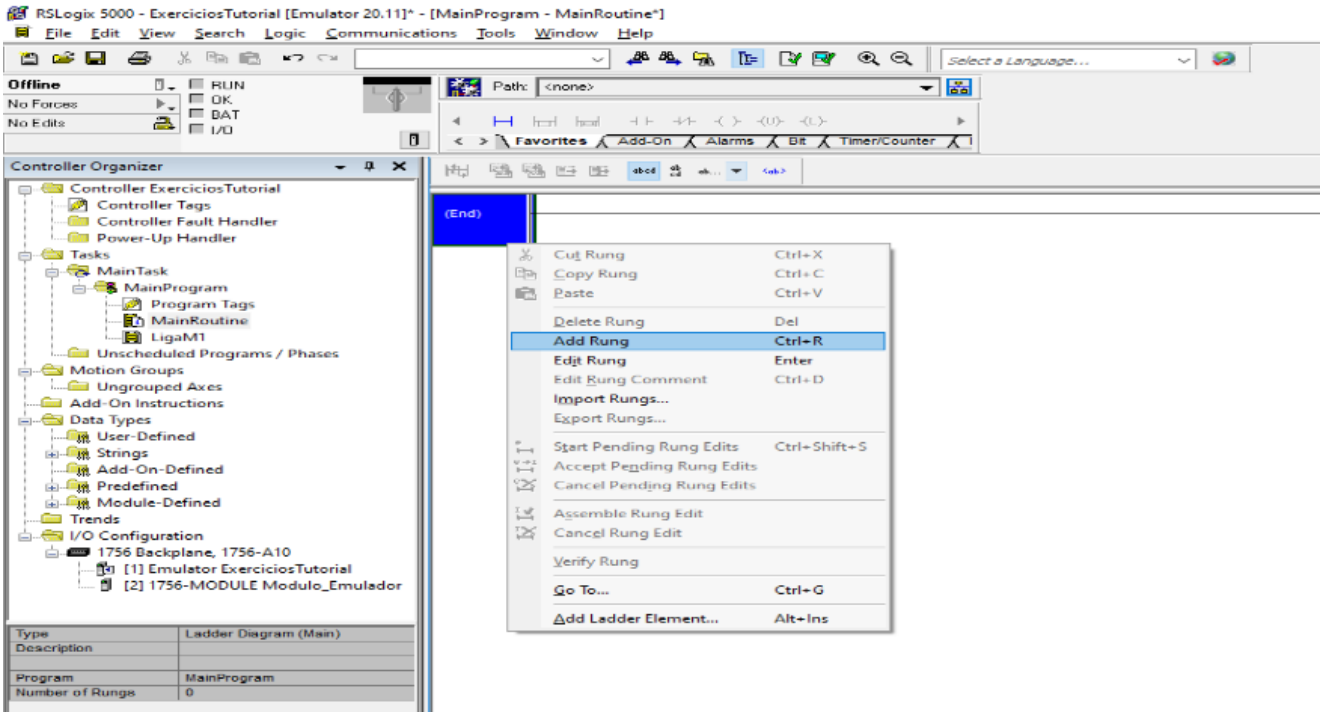
Figura 24 – Primeiro programa em linguagem Ladder



Fonte: Própria.

Entretanto, a rotina que é executada pelo ambiente de desenvolvimento integrado é chamada de MainRoutine. Por esse motivo, é preciso inserir nela uma instrução em linguagem ladder, que realize um salto para a rotina LigaM1, desenvolvida neste exemplo. Tal instrução se chama JSR, desempenhando imprescindível função, para direcionar o fluxo de execução a programas desenvolvidos em outras linguagens, bastando para isto especificar o nome do código como argumento.

Figura 25 – Primeiro programa em linguagem Ladder

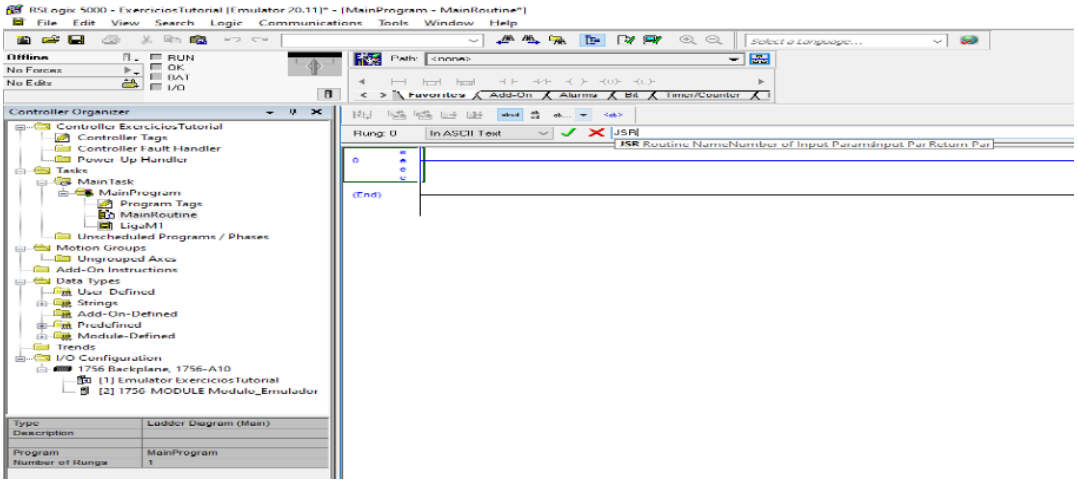


Fonte: Própria.

Para inserir JSR, clica-se duas vez sobre a rotina principal (MainRoutine), pressiona-se o botão direito do mouse no interior do quadrado que antecede a linha, e marca-se a opção Add Rung (degrau) no quadro que surgir, conforme figura 25.

Clica-se duas vezes no interior do quadrado que antecede a linha ou degrau adicionado, fazendo surgir a seguinte tela (fig. 26), onde se digita o nome da instrução JSR:

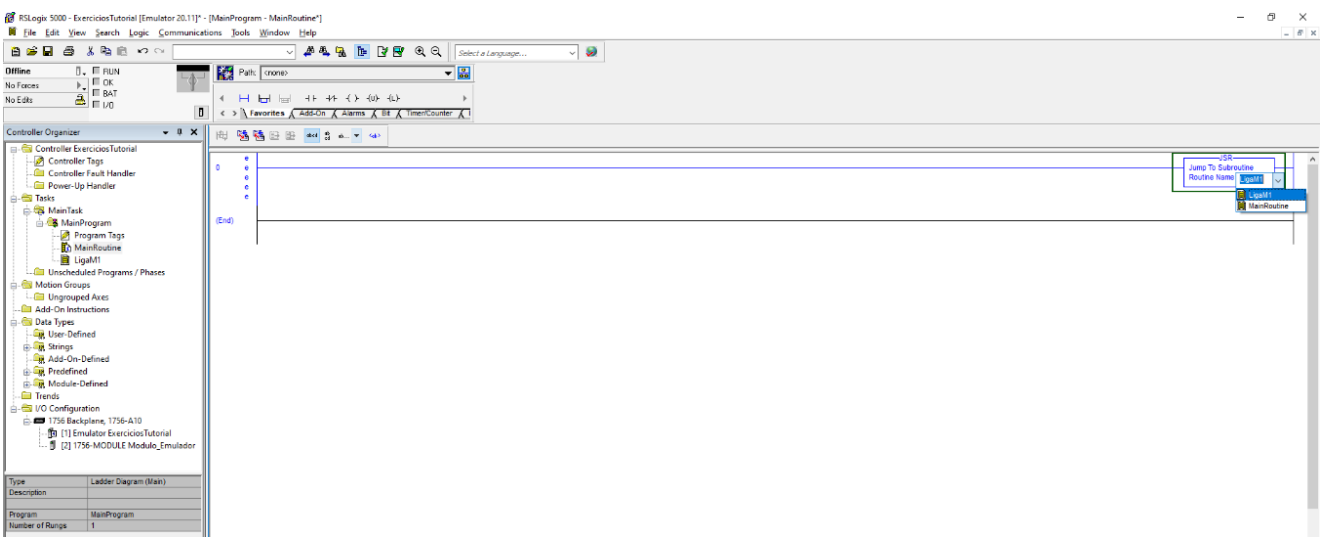
Figura 26 – Primeiro programa em linguagem Ladder



Fonte: Própria.

Pressiona-se o botão ENTER, o que proporcionará a inserção da instrução JSR na primeira linha ou degrau da rotina principal, possibilitando selecionar o nome da rotina a ser carregada, que é LigaM1 (fig. 27).

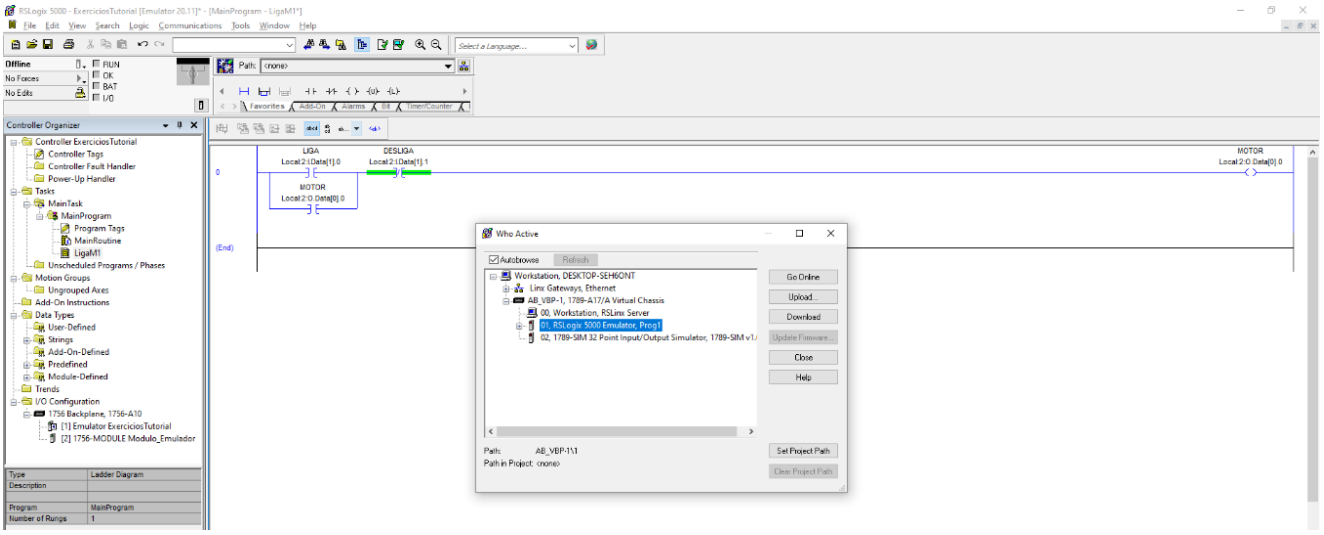
Figura 27 – Primeiro programa em linguagem Ladder



Fonte: Própria.

Uma vez que foi escolhida a rotina LigaM1 a ser carregada pela rotina principal, clica-se duas vezes sobre LigaM1, pressionando a seguir o botão WhoActive (sobre o qual o mouse está parado), fazendo surgir a figura 28, para efetuar o download do programa.

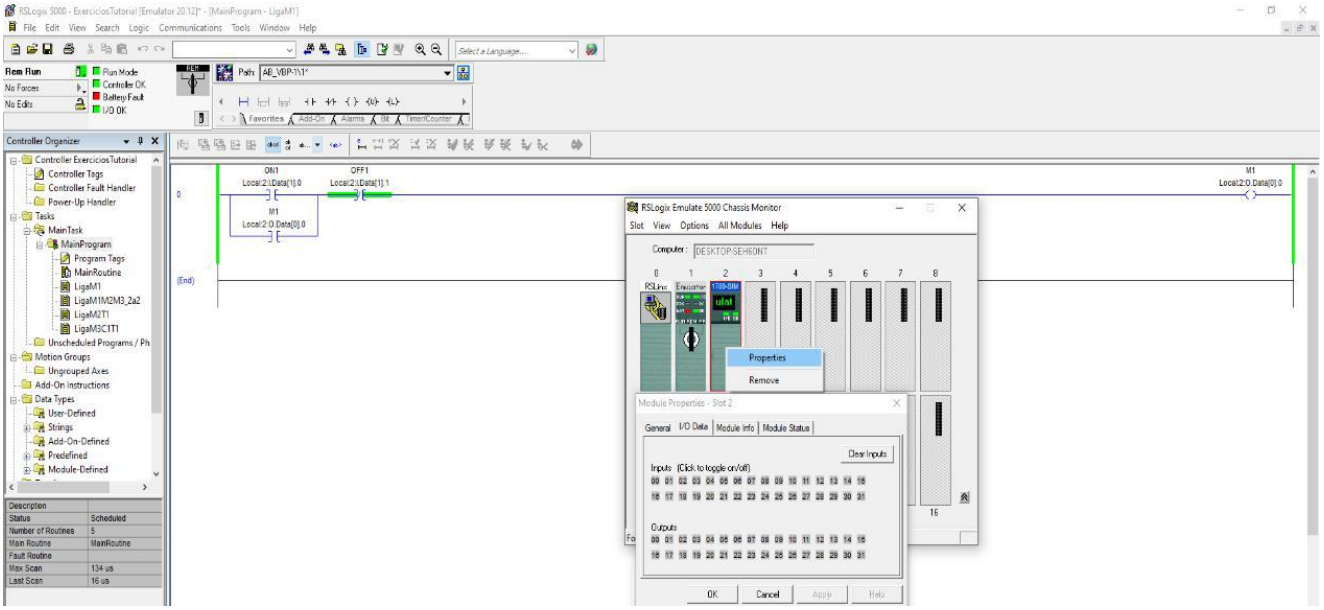
Figura 28 – Primeiro programa em linguagem Ladder



Fonte: Própria.

Após efetuar o download do código sonre o nó 01, é preciso verificar o funcionamento do programa ladder com o auxílio do RSLogix 5000 Emulator. Para tanto, clica-se com o botão direito sobre o segundo slot ou barramento, com o intuito de Acessar as propriedades dos dados de entrada e saída do CLP emulado no segundo módulo, conforme figura 29.

Figura 29 – Comunicação do RSLogix Emulate 5000 com o programa Ladder



Fonte: Própria.

Para simular o acionamento do botão liga, cujo endereço está associado ao contato virtual ON1 (Local:2:I.Data[1].0), clica-se sobre a entrada 00 nas propriedades do módulo 2, a qual se comportará

como um botão de trava, ou seja, manterá ON1 fechado, comutado de seu estado de repouso NA (normalmente aberto).

Isso proporcionará condições à circulação de corrente virtual até a saída M1, a tivando a mesma, além de comutar contatos associados ao seu endereço, como o selo ligado em paralelo a ON1, o qual passará a ser fechado, enquanto a saída estiver ativa, de acordo com a figura 30.

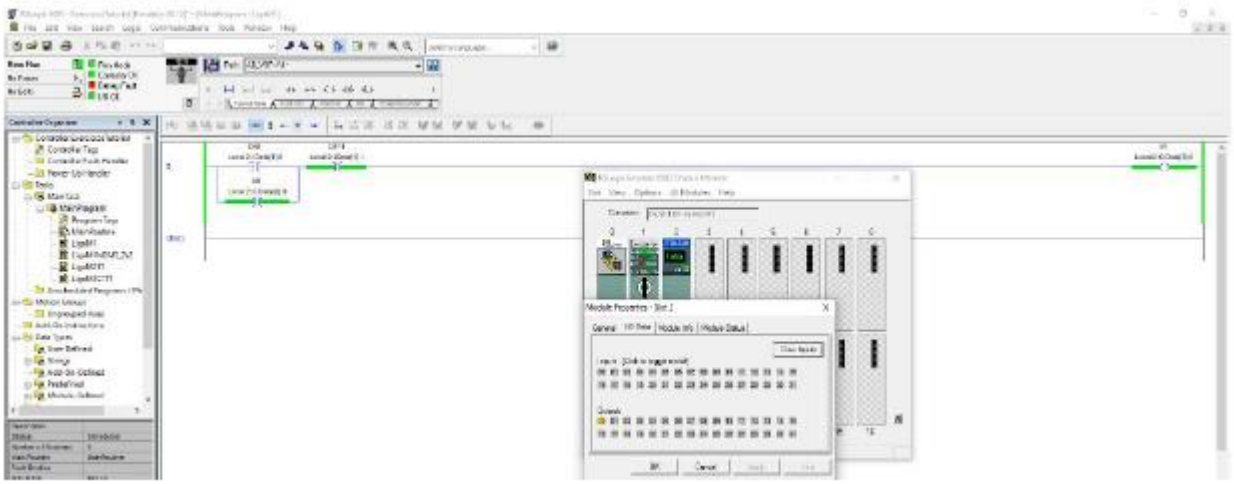
Figura 30 – Comunicação do RSLogix Emulate 5000 com o programa Ladder



Fonte: Própria.

Um exemplo de botão tipo trava é o interruptor de luz, que comuta seu estado após acionamento, devendo ser novamente manipulado, para retornar à posição de repouso. Como M1 está ativada e seu contato selo comutou para fechado, não há necessidade de manter o botão ON1 acionado, pois haverá um caminho em paralelo ao mesmo, para manter a circulação de corrente virtual, e o consequente acionamento da saída.

Figura 31 – Comunicação do RSLogix Emulate 5000 com o programa Ladder

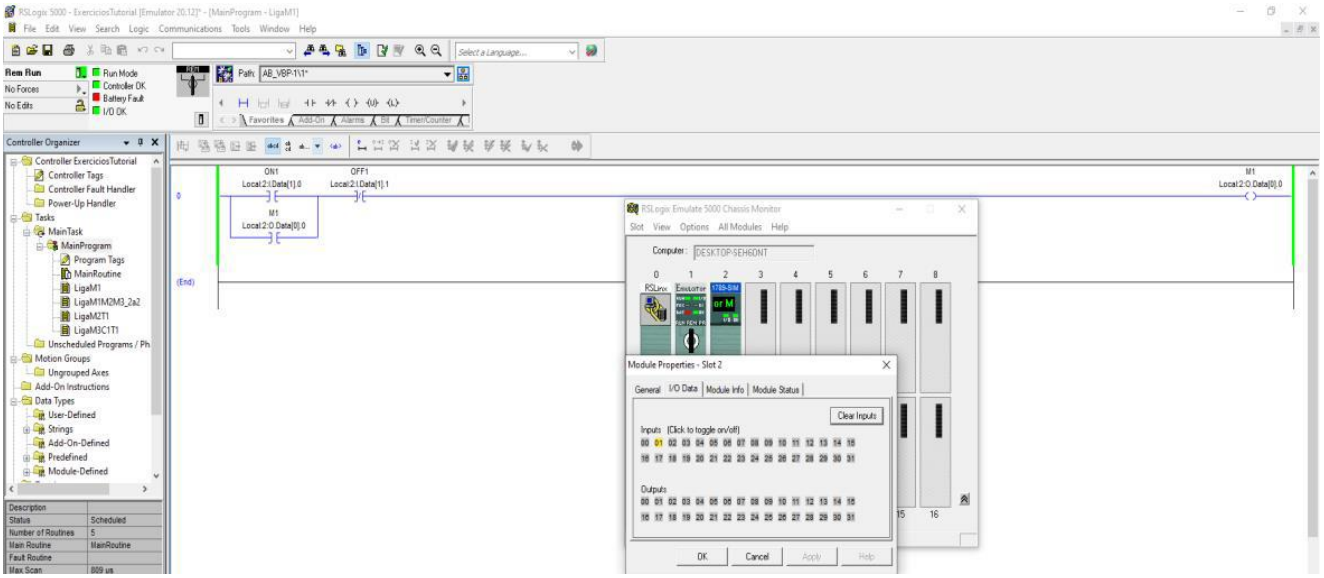


Fonte: Própria.

Dessa forma, pode-se clicar novamente sobre o botão do tipo trava conectado à entrada 00 das propriedades do módulo inserido no segundo barramento, para o contato virtual ON1 retornar ao estado de repouso NA, conforme figura 31.

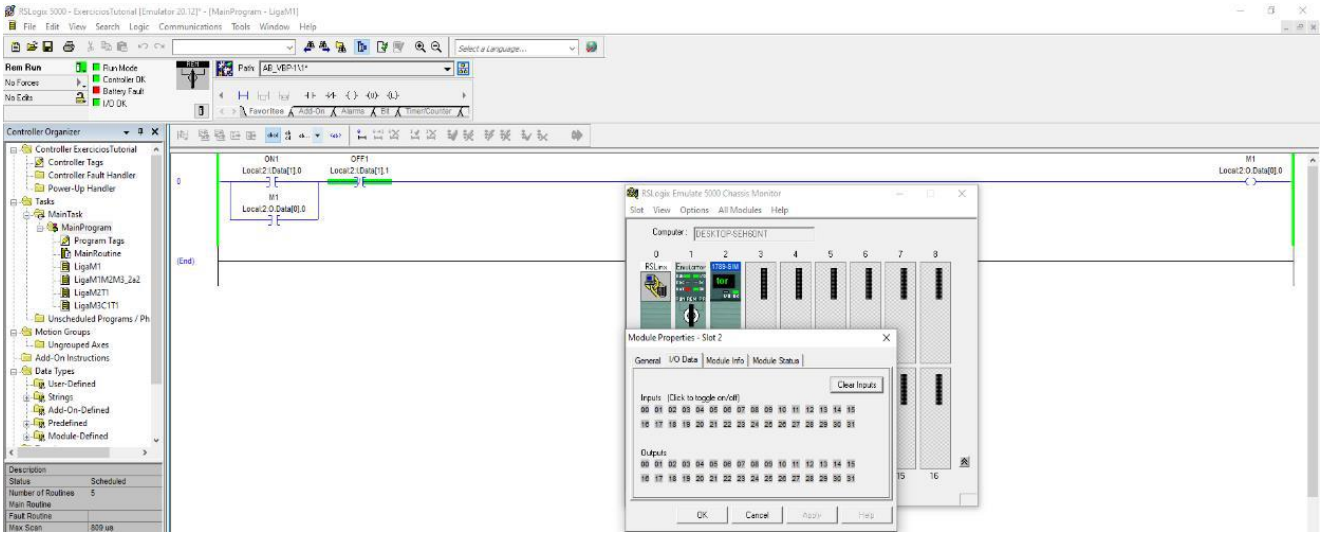
Para desativar a saída, precisa-se comutar o contato virtual OFF1, cujo estdo de repouso é normalmente fechado (NF). Essa mudança de estado é alcançada cliclando-se sobre a entrada 01 nas propriedades do módulo conectado ao barramento 2, de acordo com a figura 32. Essa ação fará o contato OFF1 abrir, interrompendo a circulação de corrente virtual na linha 0 do programa ladder, o que desativará a saída M1.

Figura 32 – Comunicação do RSLogix Emulate 5000 com o programa Ladder



Fonte: Própria.

Figura 33 – Comunicação do RSLogix Emulate 5000 com o programa Ladder



Fonte: Própria.

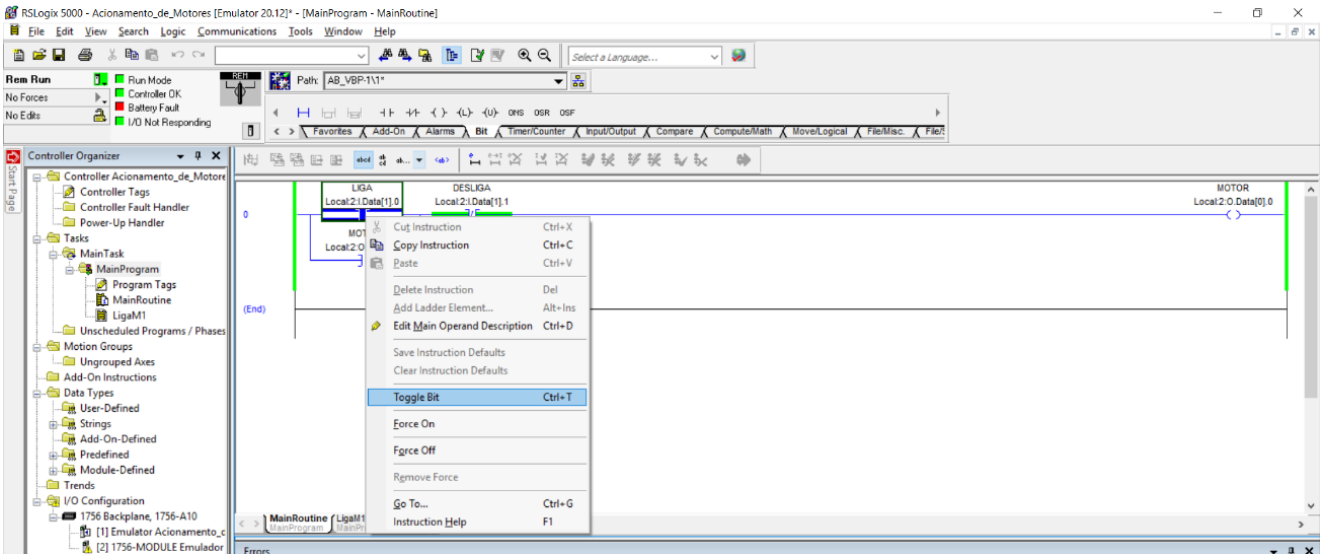
Uma vez que a saída foi desligada, o contato virtual OFF1 pode retornar ao seu estado de repouso NF, o que é conseguido clicando-se novamente sobre a entrada 01, nas propriedades do módulo conectado ao segundo barramento, conforme figura 33.

Entretanto, pode-se controlar o código ladder diretamente, sem manipular as propriedades de entrada do módulo através do RS Logix Emulate 5000. Para tanto, basta clicar com o botão direito do mouse sobre um dos contatos no programa, e selecionar a opção Toggle Bit. Atuando sobre os

contatos virtuais do programa desta forma direta, os mesmos se comportarão como um botão do tipo impulso, atuando de maneira análoga à campainha residencial.

Isso significa que após clicar em Toggle Bit diretamente sobre o contato ON1 no programa ladder, o mesmo se comportará como se tivesse recebido um pulso único de subida, ou seja, comutará rapidamente de seu estado de repouso NA, fechando e acionando a saída M1 devido à circulação de corrente virtual, para em seguida retornar a ser aberto, conforme figura 34.

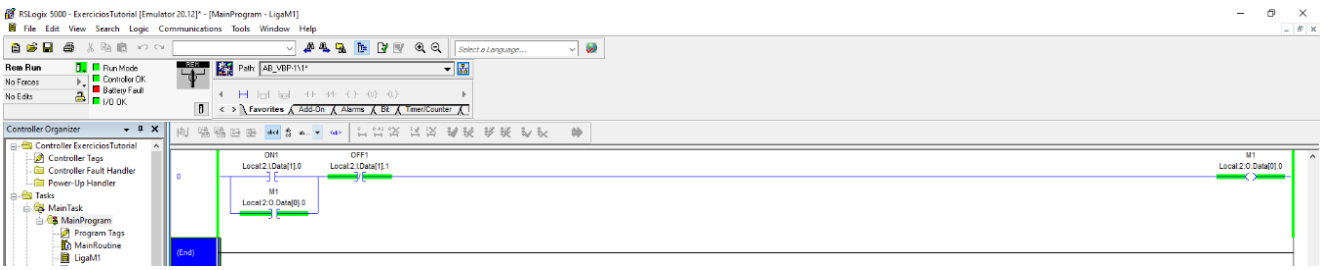
Figura 34 – Alterando estado dos contatos do programa ladder



Fonte: Própria.

Isso proporciona condições ao acionamento da saída M1 (fig. 35).

Figura 35 – Acionamento da saída M1

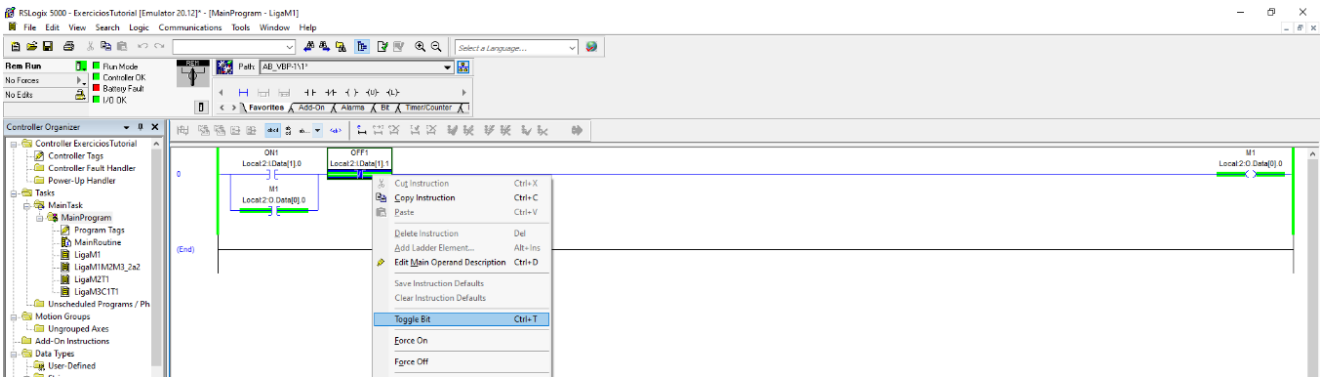


Fonte: Própria.

É importante destacar que a comutação de ON1 é imperceptível, pois ocorre apenas durante um ciclo de varredura do CLP. Entretanto a mudança de estado ocorre, uma vez que a saída M1 é ativada, depois de selecionar a opção Toggle Bit, no menu disponibilizado após clicar com o botão direito sobre a entrada ON1.

Para desligar o motor atuando diretamente sobre o contato virtual OFF1, clica-se novamente com o botão direito sobre o mesmo, para em seguida selecionar a opção Toggle Bit, conforme figura 36.

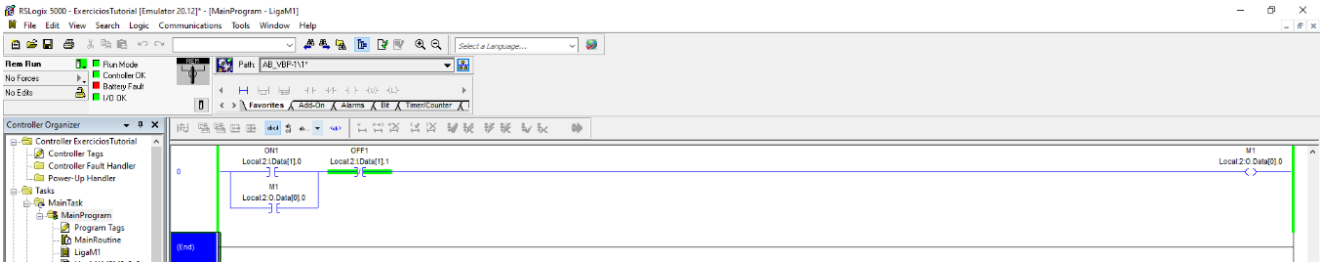
Figura 36 – Desativando a saída M1



Fonte: Própria.

Com isso, o contato OFF1 comutará rapidamente para aberto, desativando M1, para logo após retomar a posição de repouso NF (fig. 37).

Figura 37 – Desativando a saída M1



Fonte: Própria.

2) Pede-se desenvolver sistema supervisório Intouch para o problema 1.

Para comunicar o RSLogix 5000 com o Intouch, é preciso que o programa ladder esteja em execução, juntamente com o RSLogix5000 Emulate. Em seguida, deve-se criar um tópico no RSlinx, e aplicar o mesmo sobre o driver do CLP emulado, que está em execução no segundo nó do driver AB_VP-1. O endereçamento da *tagname* no sistema supervisório é o mesmo dos elementos constituintes do programa Ladder RSLogix 5000.

Seguem as telas de configuração do RSLinx, conforme figuras 38 e 39.

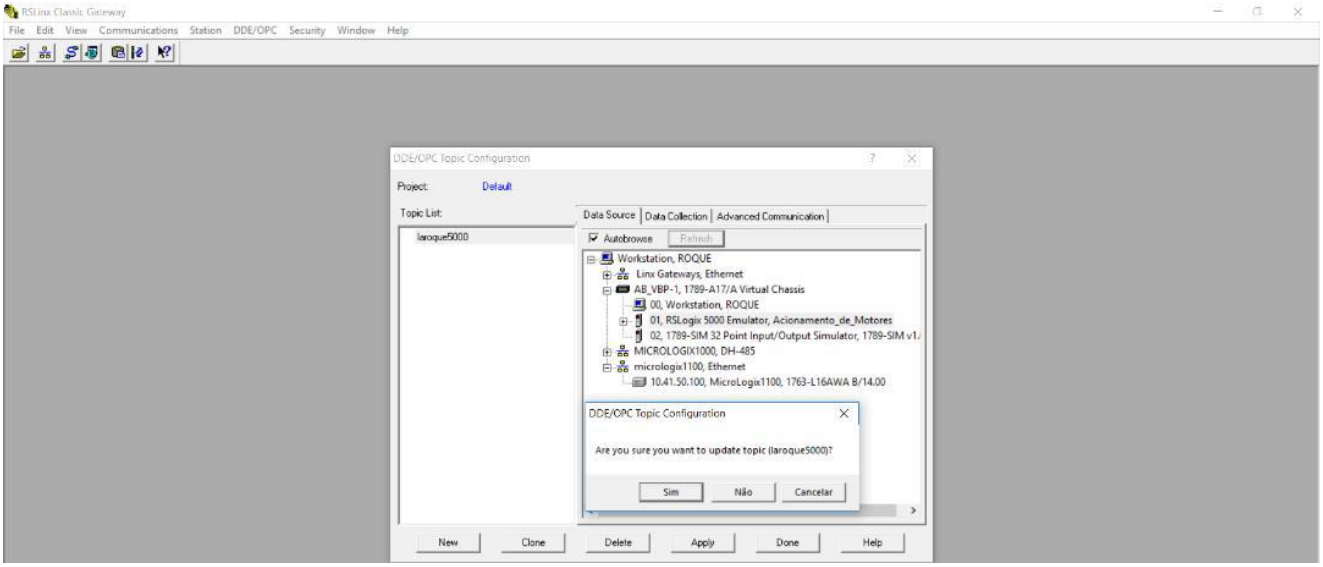
Figura 38 – Tela de configuração do RSLinx



Fonte: Própria.

Clica-se em New Topic, nomeando-o laroque5000, ou qualquer outra designação, a critério do usuário. Aplica-se o mesmo ao driver AB_VP-1.

Figura 39 – Tela de configuração do RSLinx



Fonte: Própria.

A seguir, são apresentadas as telas de configuração do sistema supervisório Intouch. Após iniciar uma aplicação no supervisório Intouch, é preciso criar uma janela, clicando-se em File e New Window (fig. 40).

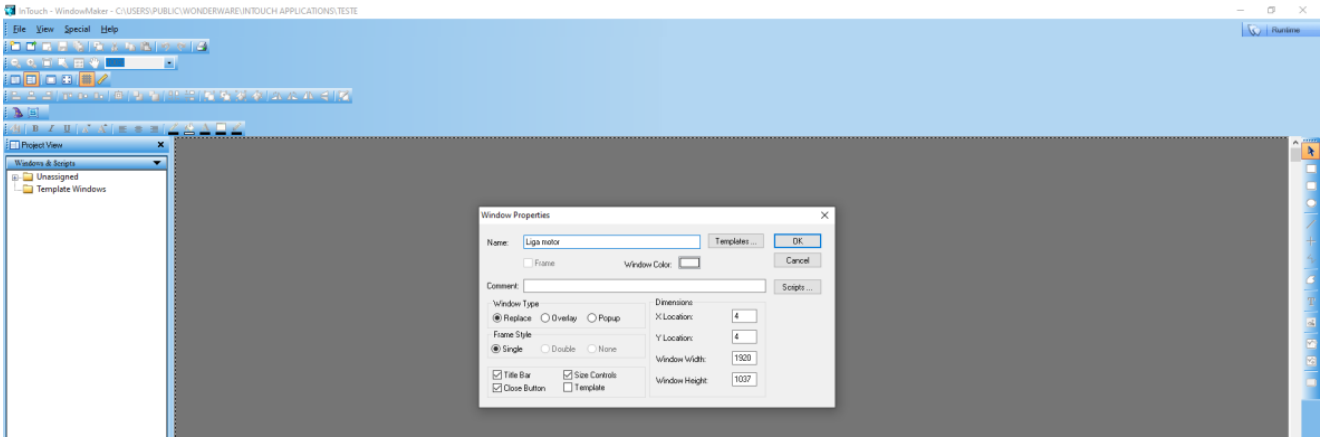
Figura 40 – Tela de configuração do InTouch



Fonte: Própria.

O próximo passo consiste em atribuir nome à janela que surgirá, e pressionar OK (fig.41).

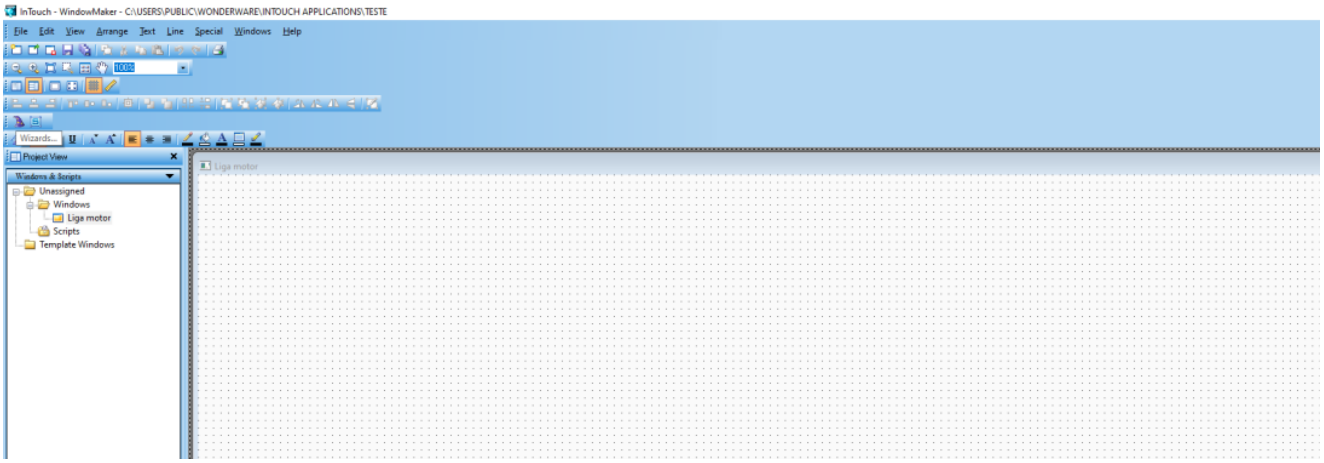
Figura 41 – Atribuindo nome à janela criada



Fonte: Própria.

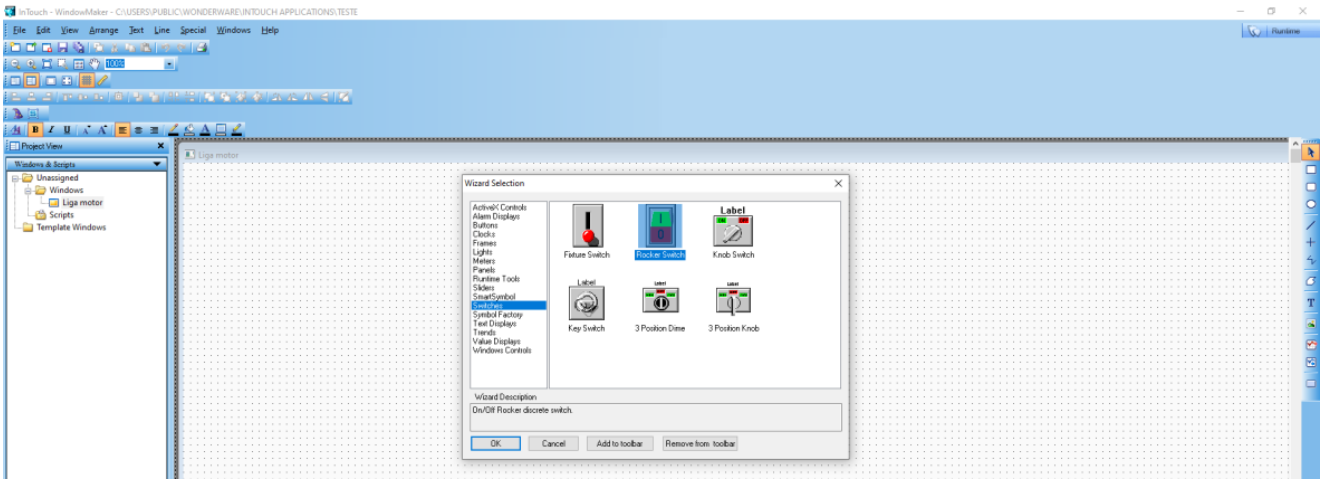
Posiciona-se em seguida o mouse sobre o chapéu de mago (*Wizards*), para acessar a biblioteca de objetos do sistema supervisório.

Figura 42 – Acessando o chapéu de mago (*Wizards*)



Fonte: Própria.

Figura 43 – Escolha do botão Rocker Switch

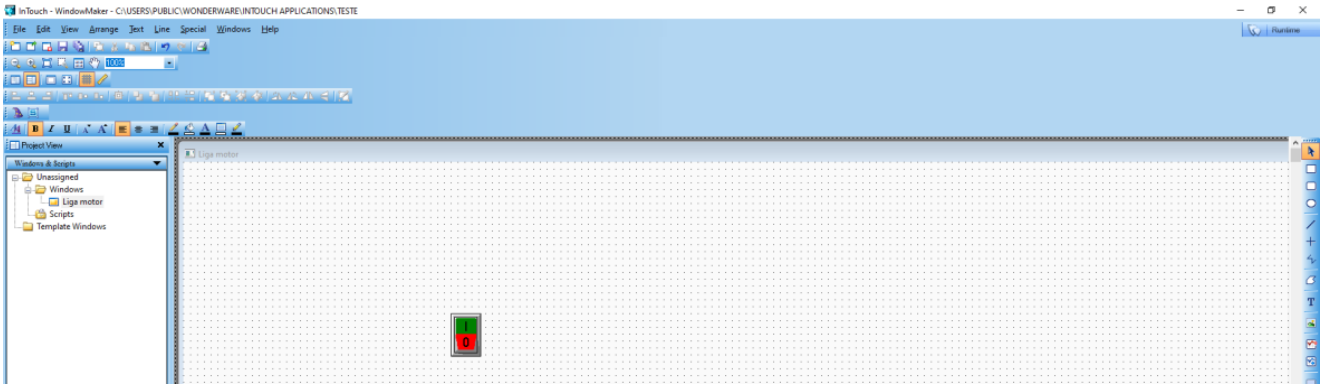


Fonte: Própria.

Clica-se em o mouse sobre Wizards, originando a figura 43, na qual se escolhe o botão Rocker Switch, que será utilizado na tela do supervisório.

A próxima etapa consiste em pressionar OK e clicar dentro da janela, finalizando a inclusão do botão escolhido (fig. 44).

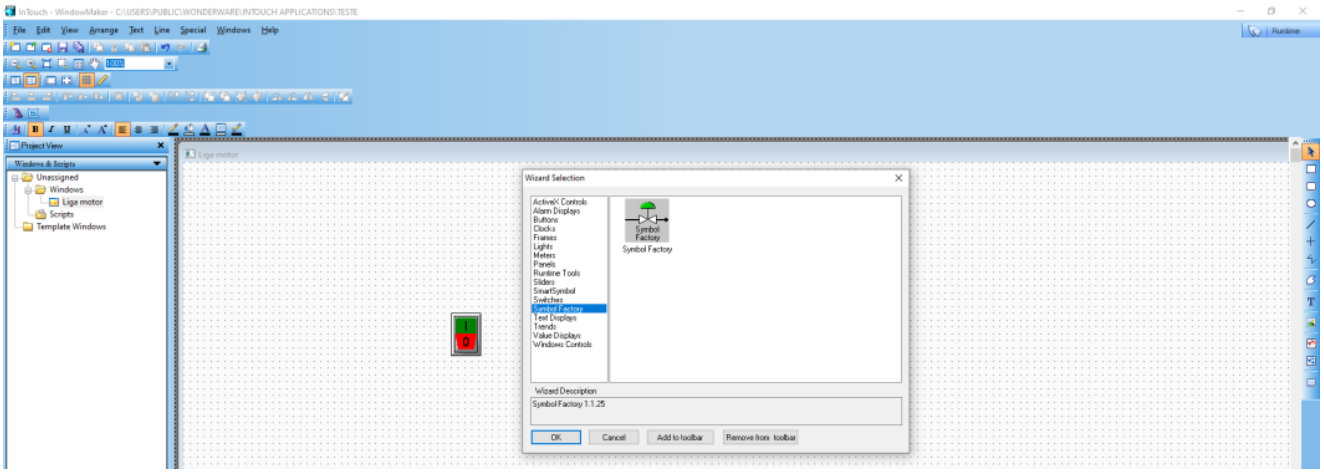
Figura 44 – Inclusão do botão na tela do supervisório InTouch



Fonte: Própria.

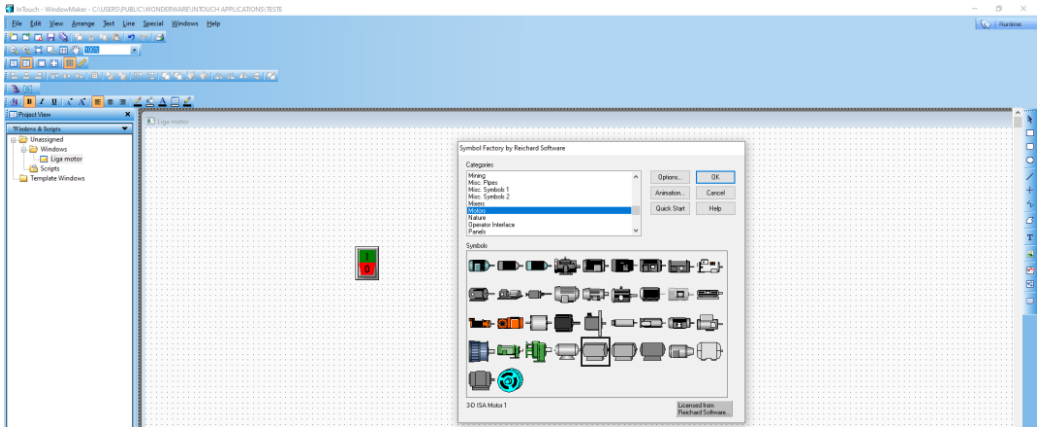
Para selecionar o motor, basta clicar novamente sobre *Wizards*, selecionar a opção *Symbol Factory*, e pressionar OK, originando a figura 45.

Figura 45 – Inclusão do botão na tela do supervisório InTouch



Fonte: Própria.

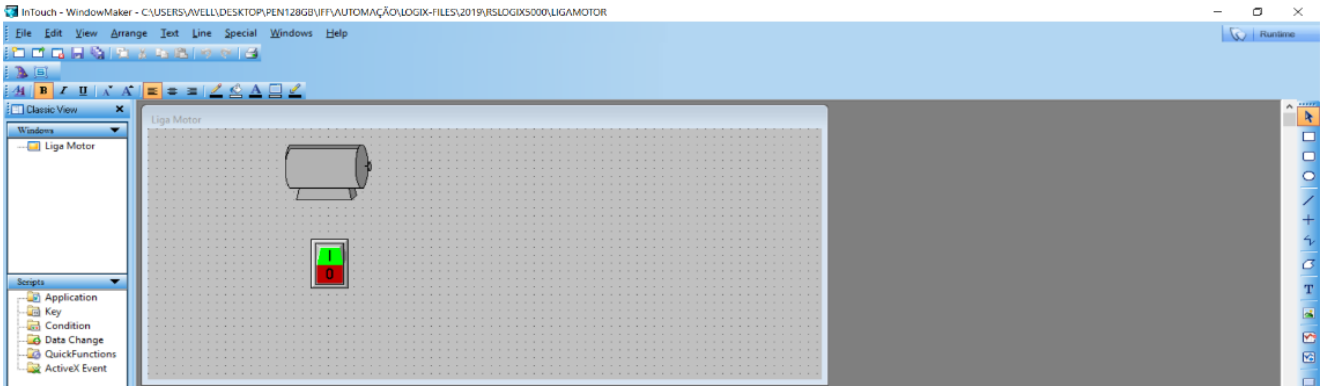
Figura 46 – Seleção do motor 3-D ISA



Fonte: Própria.

Após pressionar o botão OK e clicar sobre a tela do supervisório, surgirá a figura 46, na qual se escolhe a categoria Motors, clicando-se a seguir sobre o motor 3-D ISA. Após pressionar OK, o motor está inserido sobre a tela (fig.47).

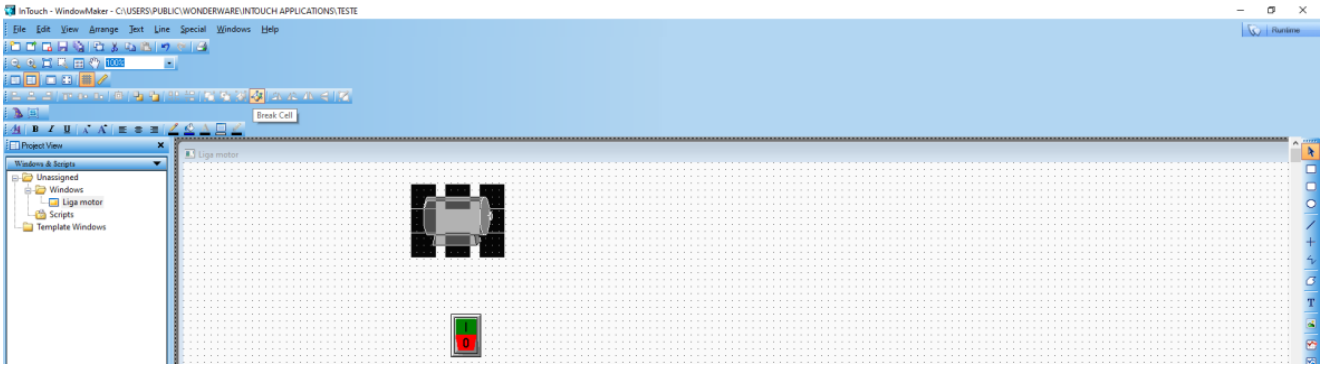
Figura 47 – Motor e botão inseridos na tela do sistema supervisório.



Fonte: Própria.

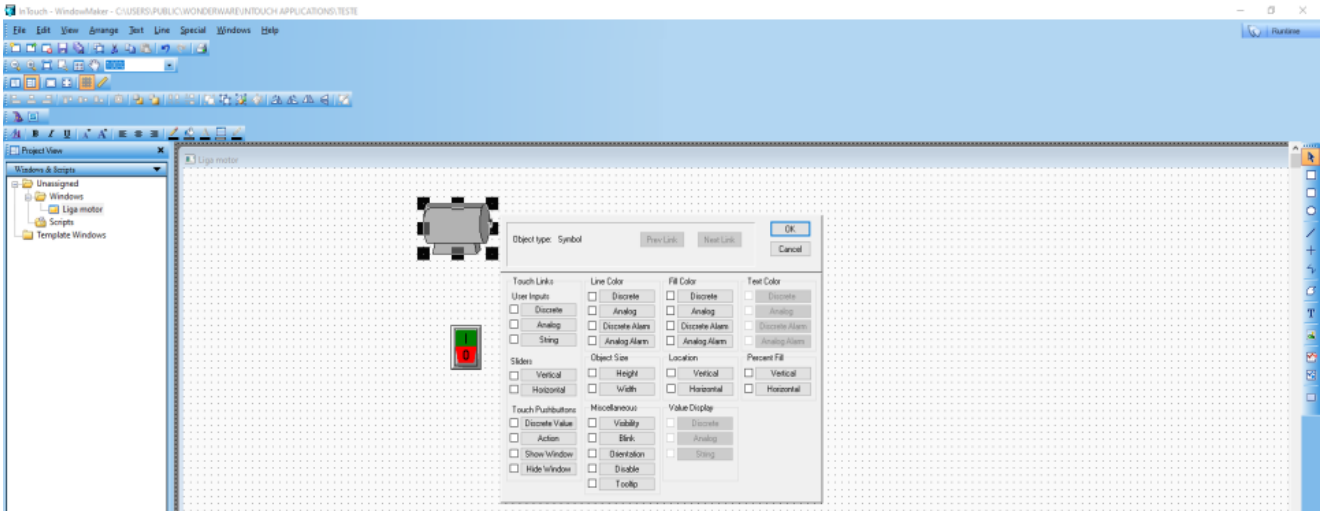
Agora é necessário acessar as propriedades dos objetos inseridos. Para isso, é preciso quebra as células dos mesmos. Inicia-se esse processo com o motor. É preciso clicar uma vez sobre o mesmo, e selecionar o botão Break Cell, conforme figura 48.

Figura 48 – Quebrando a célula do motor



Fonte: Própria.

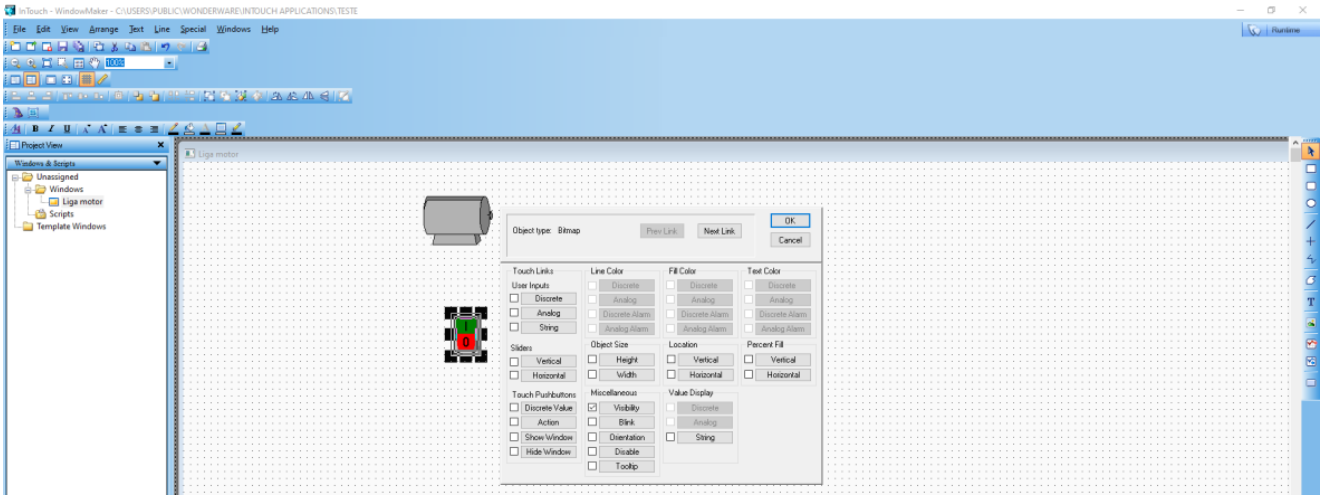
Figura 49 – Acesso às propriedades do motor



Fonte: Própria.

Feito isso, os próprios marcadores de seleção do motor são reduzidos, indicando que suas propriedades já podem ser acessadas, o que pode ser feito clicando-se duas vezes sobre ele, conforme figura 49. Repetindo os mesmos procedimentos realizados sobre o botão, será possível acessar suas propriedades (fig. 50), analogamente ao motor.

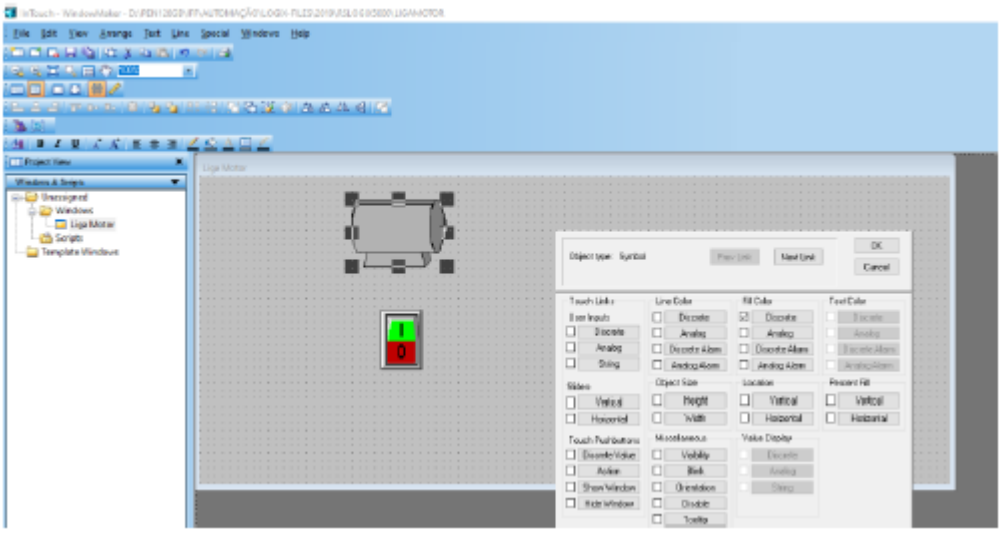
Figura 50 – Acesso às propriedades do botão



Fonte: Própria.

O próximo passo do sistema supervisório consiste em configurar os dois objetos, botão e motor, para se comunicarem com o programa ladder. Para tanto, clica-se duas vezes sobre o motor, para acessar suas propriedades, conforme já mostrado na figura 49. A seguir marca-se a caixa de seleção da propriedade Discrete, pertencente à classe das propriedades do motor designada por Fill Color, conforme figura 51.

Figura 51 – Acesso às propriedades Fill Color Discrete, do motor

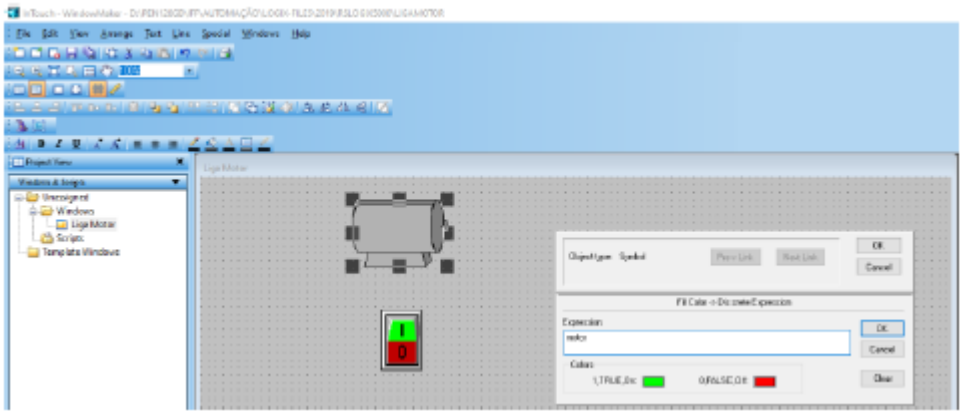


Fonte: Própria.

O próximo passo consiste em pressionar o botão da propriedade *Discrete*, pertencente à classe *Fill Color*, dando origem à figura 52, onde se atribui o nome de motor à expressão que controlará as cores

de preenchimento do objeto homônimo, definidas como verde para ligado e vermelho, caso esteja desligado.

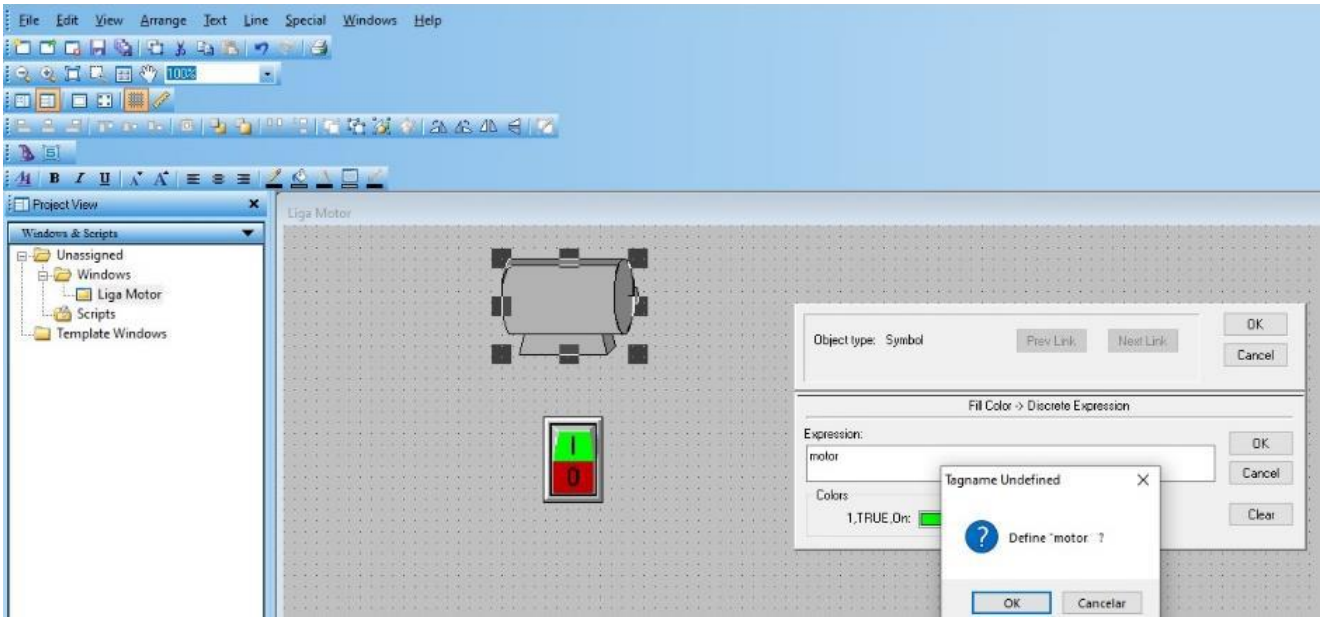
Figura 52 – Acesso às propriedades *Fill Color Discrete*, do motor



Fonte: Própria.

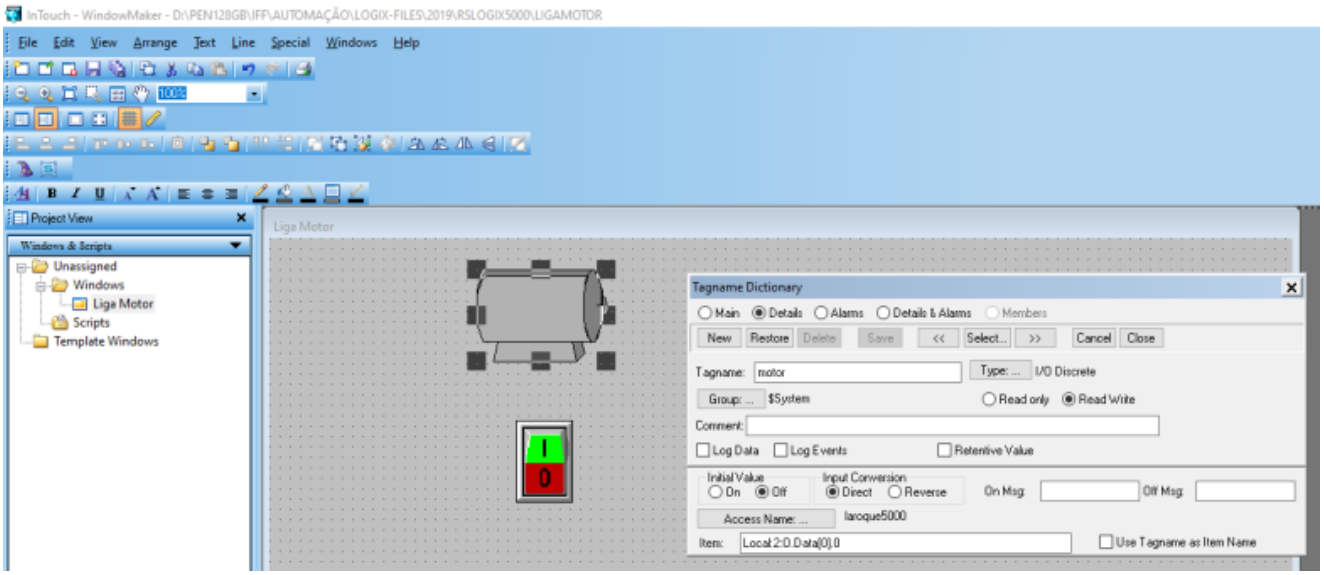
Ao pressionar o botão OK, surgirá tela para confirmar a criação da expressão motor.

Figura 53 – Definições da tagname motor



Fonte: Própria.

Figura 54 – Definições da tagname motor



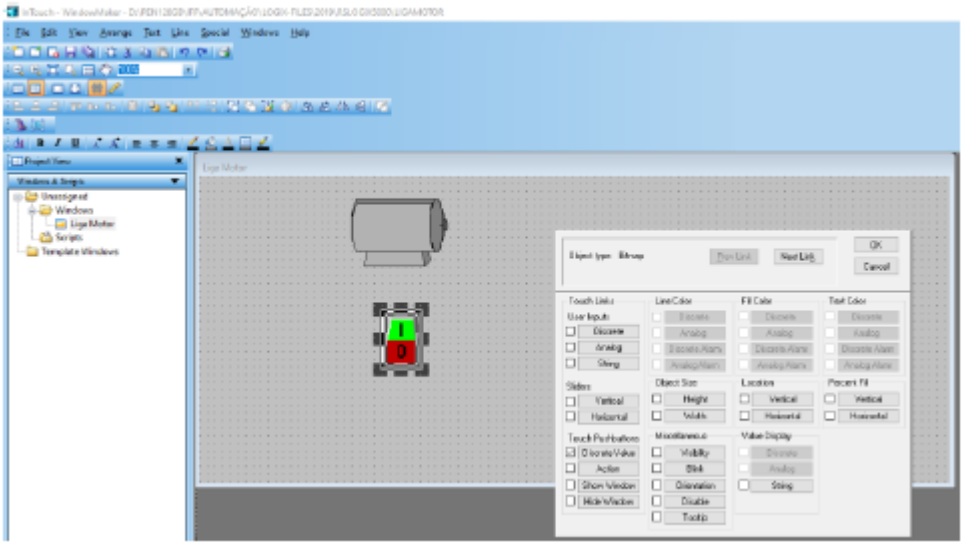
Fonte: Própria.

Pressiona-se o botão OK na figura 53, para confirmar a criação da tagname (expressão ou variável) motor, originado a figura 54, na qual se registram suas definições.

O tipo da *tagname* motor deve ser I/O Discrete, porque o sistema supervisório se comunicará com o programa ladder elaborado no RSLogix 5000. O conteúdo do campo *Item* deve conter o endereço do elemento do programa ladder com o qual o sistema supervisório se comunicará, que é Local:2:O.Data[0].0 para, em seguida, pressionar o botão Save, e gravar as definições para a expressão criada, que controlará as cores de preenchimento.

Clicando-se duas vezes sobre o objeto botão, consegue-se visualizar suas classes e respectivas propriedades, conforme figura 55.

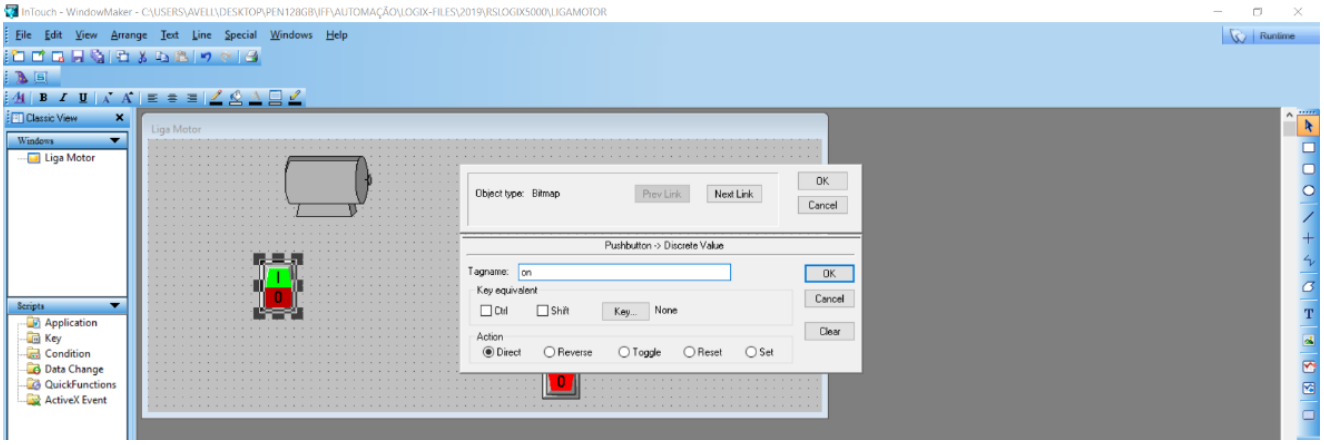
Figura 55 – Acesso à propriedade Touch Pushbuttons I/O Discrete, do botão Liga.



Fonte: Própria.

Marca-se a propriedade Discrete Value, pertencente à classe Touch Pushbuttons, do objeto botão, originando a figura 56, na qual se atribuí à tagname o nome on, junto com a ação Direct.

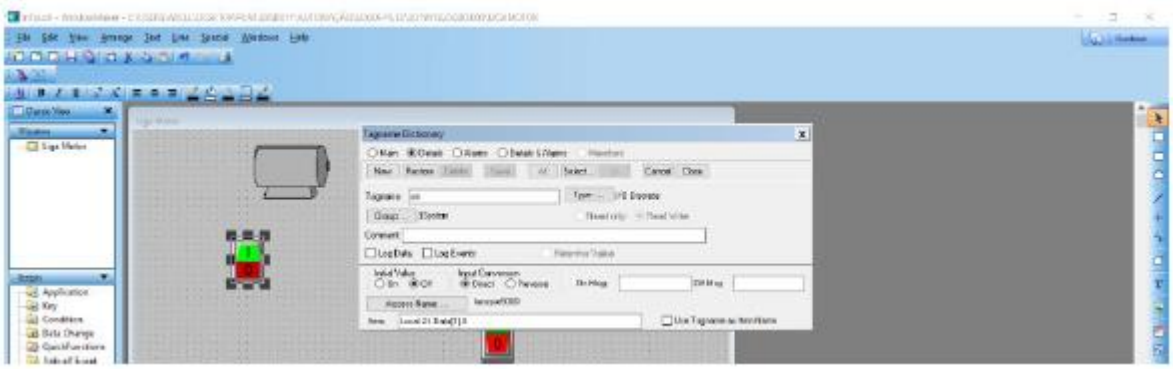
Figura 56 – Definição da tagname on.



Fonte: Própria.

Dessa forma, o comportamento do botão será análogo a uma chave do tipo impulso, que comuta o estado de repouso do contato somente enquanto estiver pressionado. Ao se soltar o botão, o contato do programa ladder retornará ao estado de repouso. Após pressionar o botão OK (fig. 56) e confirmar a criação da tagname (variável) on, surgirá a figura 57, onde se definem os atributos tipo e Item como I/O Discrete e Local:2:I.Data[1].0 (endereço do contato liga do programa ladder, com o qual o supervisório se comunica), respectivamente.

Figura 57 – Definição da tagname on

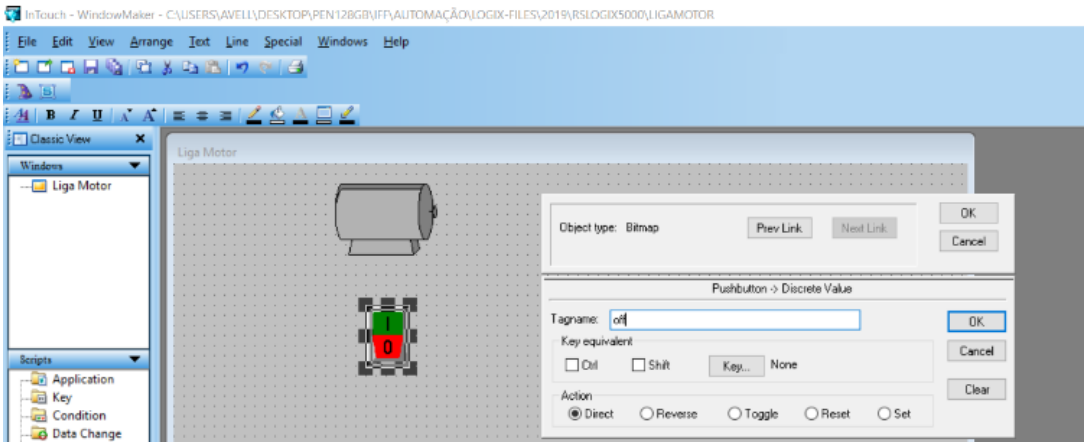


Fonte: Própria.

É chegado o momento de criar um botão desliga no sistema supervisório, para se comunicar com o contato OFF1 do programa ladder, cujo estado de repouso é normalmente fechado, associado em série com o botão liga 1, que está em paralelo com o contato selo da saída do motor. Dessa forma, quando o botão desliga do InTouch for pressionado, o contato OFF1 no programa ladder comutará para aberto, impedindo circulação de corrente virtual, o que desativará a bobina de saída que controla o motor, desligando a máquina.

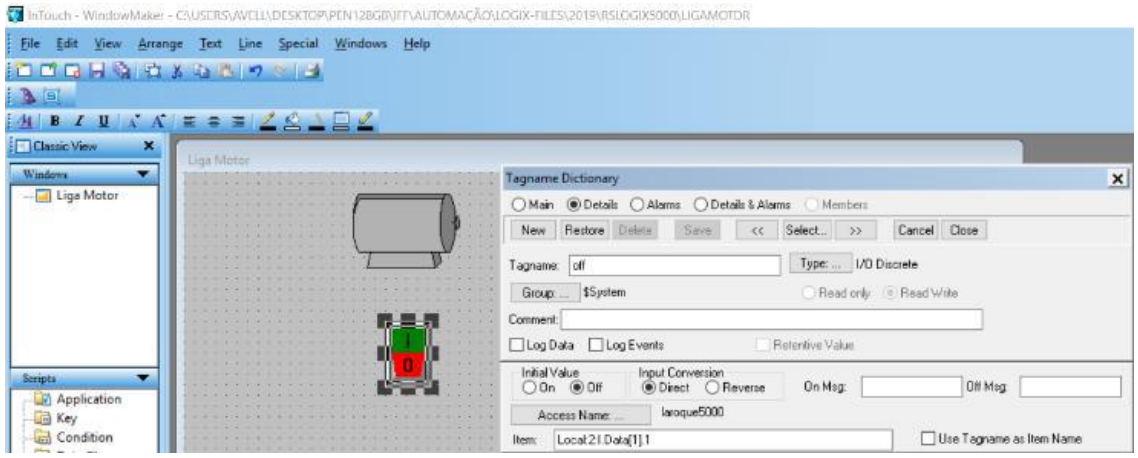
As configurações do botão desliga são as mesmas que as efetuadas anteriormente para o botão liga, exceto para a tagname, designada por off, e o endereço do Item, o qual corresponde ao endereço do contato OFF1 no programa ladder, ou seja, Local:2:I.Data[1].0, conforme figuras 58 e 59.

Figura 58 - Configurações do botão desliga



Fonte: Própria.

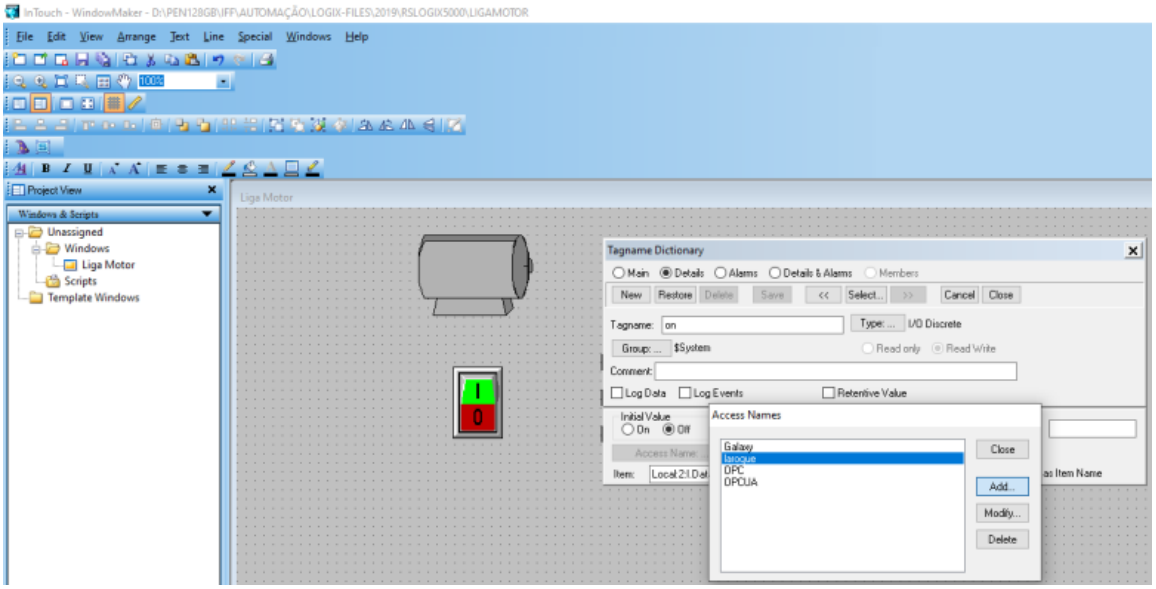
Figura 59 - Configurações do botão desliga



Fonte: Própria.

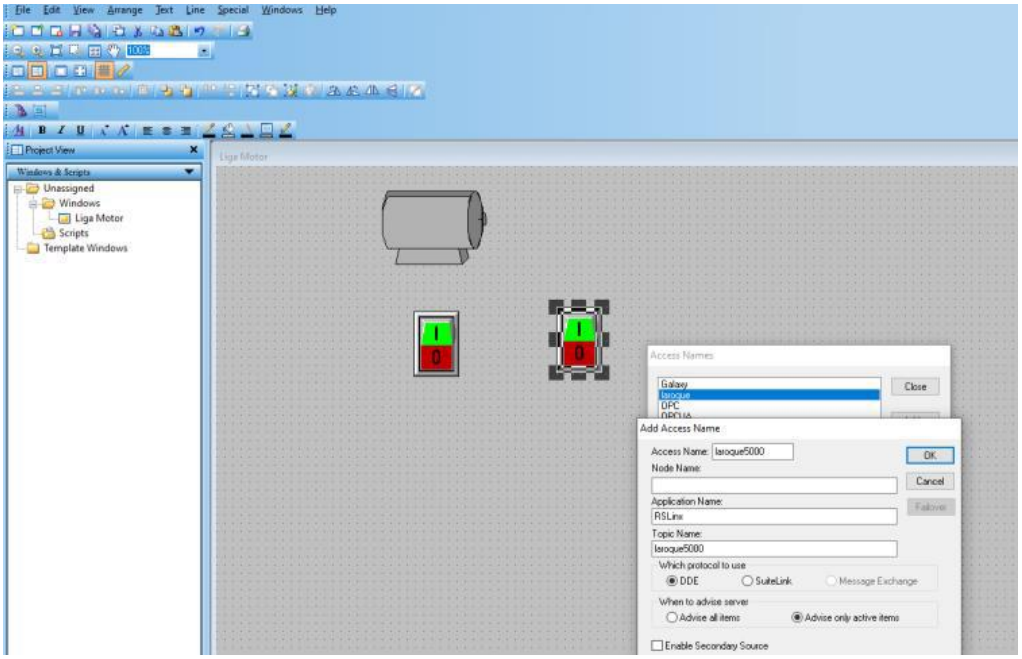
Nesse momento é preciso explicar a função do botão *Access Name*, que trata da criação de um nome de acesso, para se comunicar com o tópico criado no programa RSLinx, cujo nome é laroque5000. Para tal fim, clica-se sobre o botão *Access Name*, obtendo-se a figura 60.

Figura 60 - Configurações do nome de acesso



Fonte: Própria.

Figura 61 - Configurações do nome de acesso



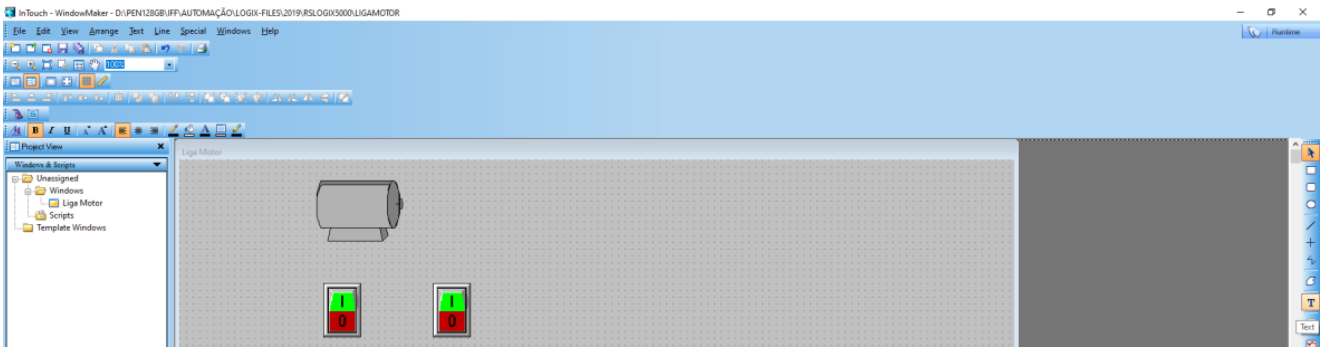
Fonte: Própria.

Clica-se sobre o botão Add para adicionar novo nome de acesso, originando a figura 61, na qual se preenchem os campos *Access name* e *Topic name* com laroque5000, que é o mesmo nome do tópico criado no RSLinx, para estabelecer comunicação entre este programa e o sistema supervisório.

O protocolo configurado na figura 61 é o DDE (*Dinamic Data Exchange*), e o campo Node Name deve ser deixado em branco. Após pressionar o botão OK, o nome de acesso criado será associado a todos os objetos inseridos na janela do sistema supervisório.

Resta agora inserir abaixo dos botões dois objetos de texto, um de cada vez, para escrever neles o nome Liga e Desliga, possibilitando ao usuário do InTouch identificar qual deles é o responsável por ativar ou desativar o motor, conforme exposto nas figuras 62 e 63.

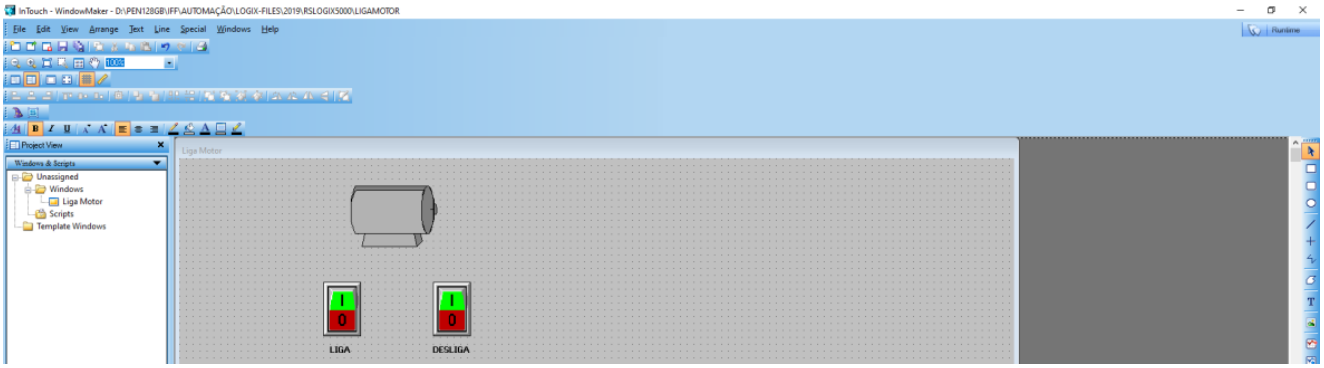
Figura 62 - Configurações do objeto texto



Fonte: Própria.

Clicando-se sobre o objeto texto da figura 62, e pressionando novamente o mouse abaixo do botão esquerdo, digita-se a seguir seu nome, que é LIGA. Procedendo analogamente com o botão direito, pressiona-se o objeto texto mais uma vez, clicando-se abaixo do botão direito, para digitar DESLIGA (fig. 63).

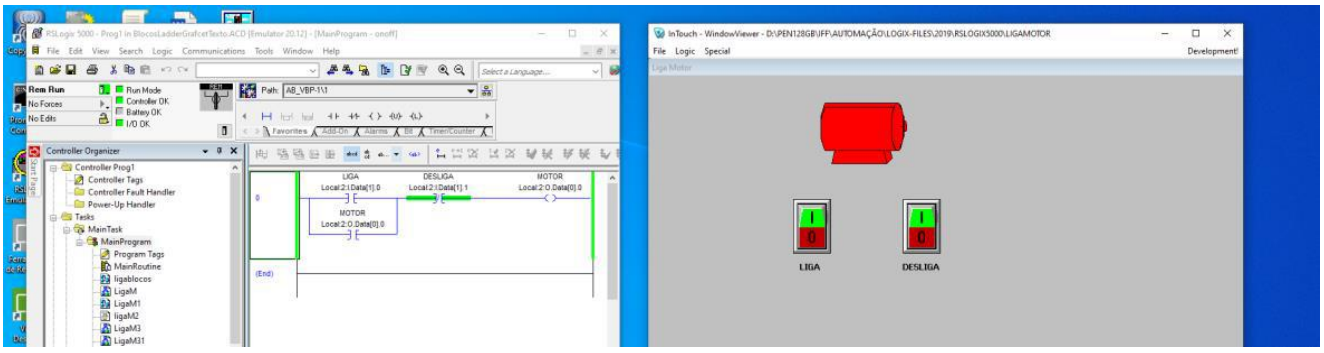
Figura 63 - Configurações do objeto texto



Fonte: Própria.

Após colocar em execução o programa ladder ligamotor, conforme instruções do começo do livro, chegou o momento de colocar o InTouch em modo de execução. Para isso, clica-se com o o mouse sobre o botão situado no canto superior direito do supervisório, identificado por Runtime. Na janela de entrada que surgir, basta clicar sobre o botão Demo Mode (fig. 64)

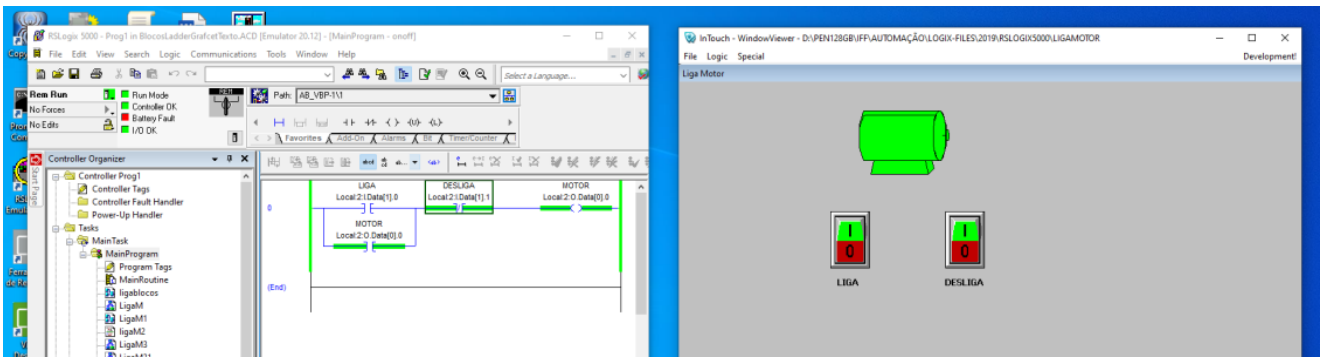
Figura 64– Operando o programa ladder a partir do sistema supervisório



Fonte: Própria.

Ao clicar sobre o botão liga da janela do InTouch, o contato homônimo comutará do estado de repouso normalmente aberto (NA), tornando-se fechado, habilitando circulação de corrente virtual até a bobina de saída que aciona o motor, cujo endereço é Local:2:O.Data[0].0 (figura 65).

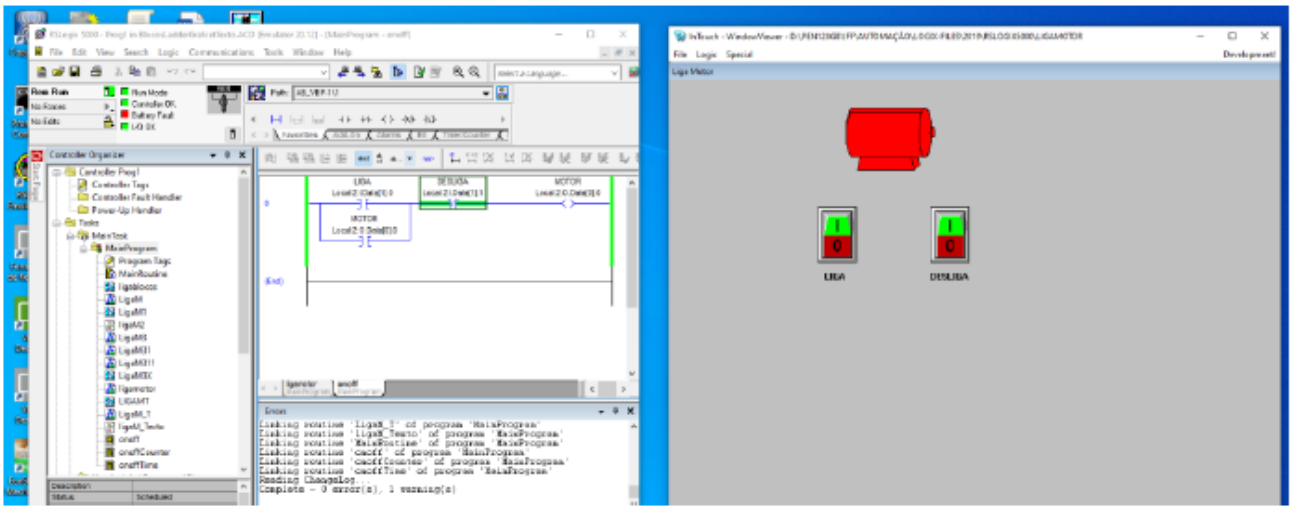
Figura 65– Acionando o motor do programa ladder a partir do sistema supervisório



Fonte: Própria.

Ao clicar sobre o botão desliga do sistema supervisório, o contato normalmente fechado homônimo do programa ladder comutará para aberto, interompendo a circulação de corrente virtual e desativando o motor, moficando sua cor de preenchimento de verde, relacionada a atividade, para vermelho, associada a inatividade, conforme figura 66.

Figura 66 – Desativando o motor do programa ladder a partir do sistema supervisório

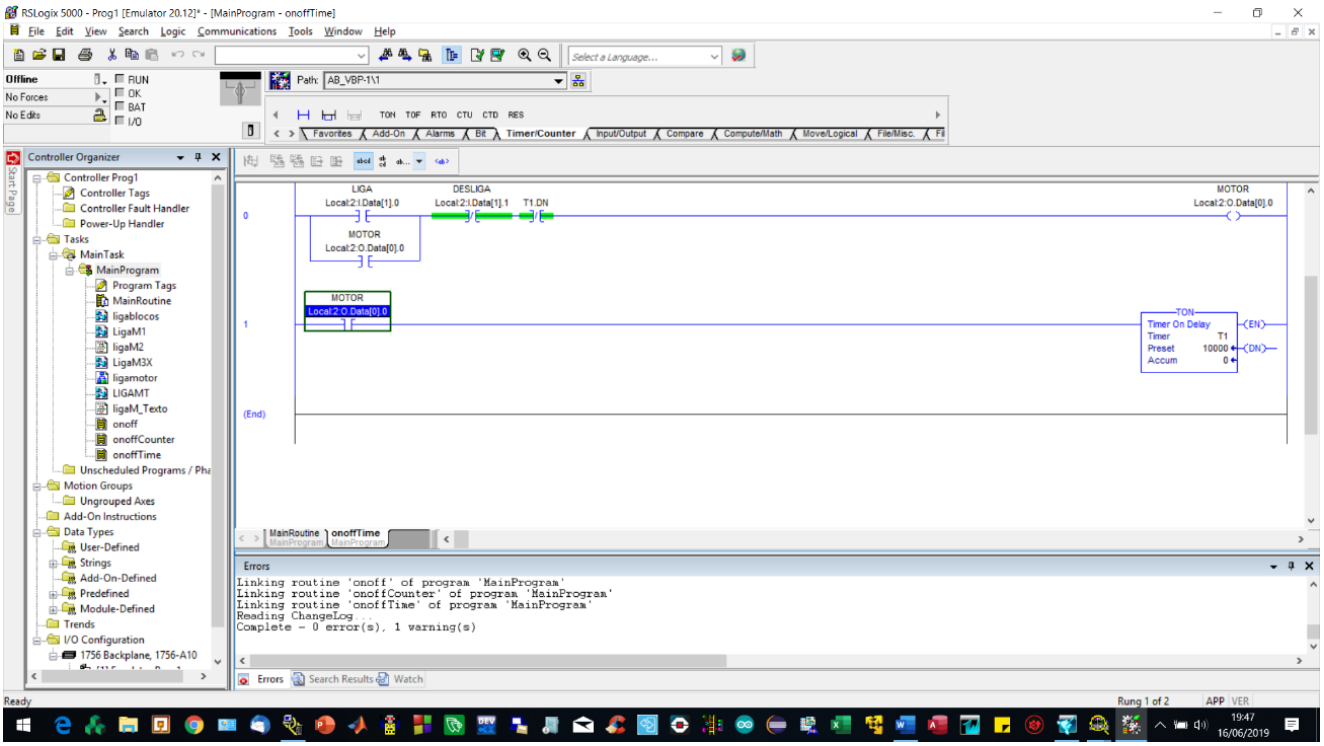


Fonte: Própria.

3) pede-se elaborar programa em linguagem Ladder, que permite ligar e desligar manualmente um motor, sendo que o mesmo deve ser automaticamente desativado, 10 segundos após entrar em operação.

A figura 67 apresenta tela com a solução do problema.

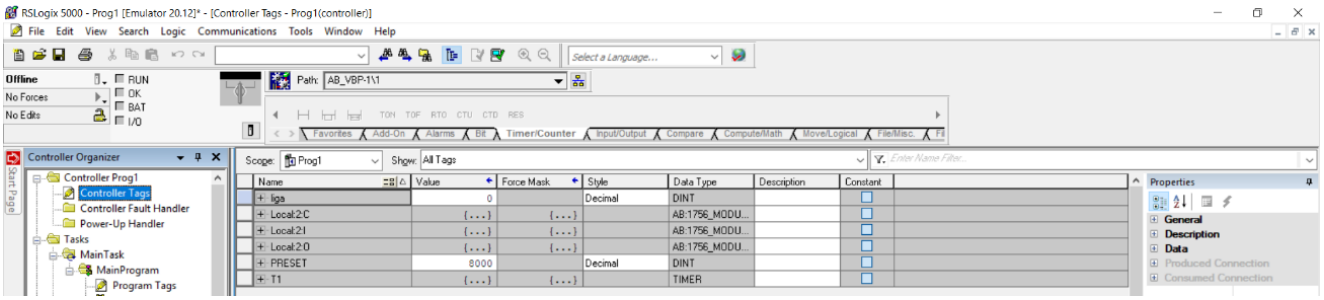
Figura 67 – Solução do terceiro problema



Fonte: Própria.

É preciso dissertar sobre a criação de *tags* ou variáveis. Há dois tipos de *tags*, que podem ser definidas no controlador, acessíveis clicando-se sobre *Controller Tags*, conforme figura 68.

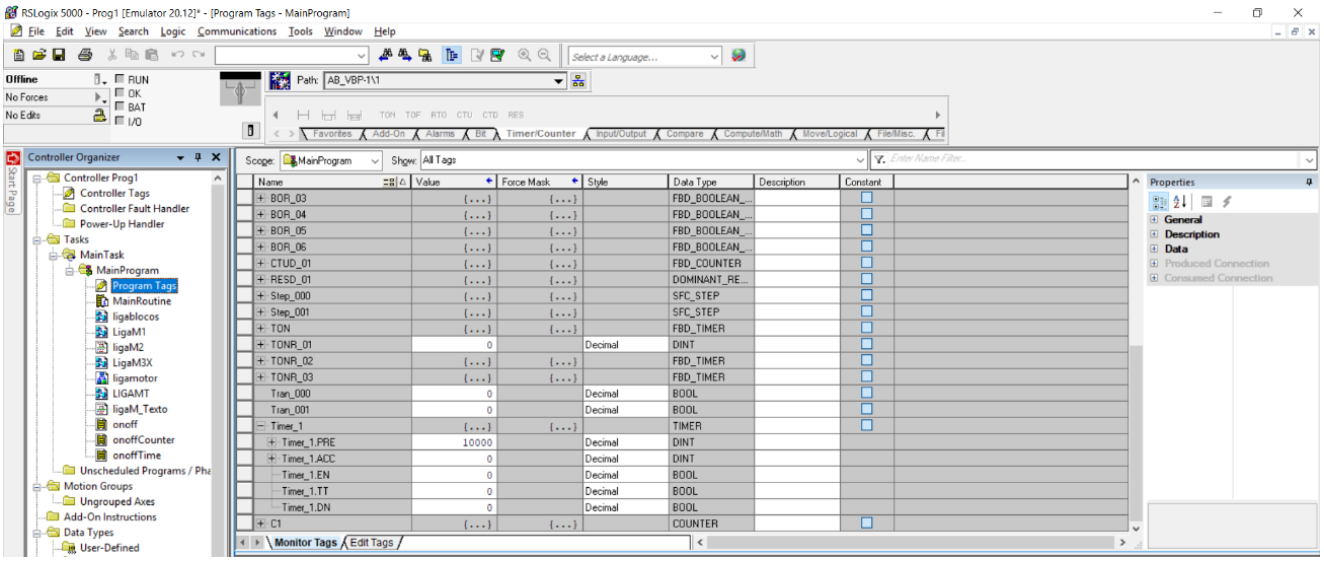
Figura 68 – Tags (variáveis) do controlador



Fonte: Própria.

Variáveis criadas no escopo do controlador têm características globais, e podem ser utilizadas na comunicação com sistemas supervisórios. Entretanto, variáveis locais alcançam somente o programa elaborado, e não conseguem transferir seus estados a programas de supervisão. Tais variáveis podem ser criadas clicando-se sobre *Program Tags* (fig. 69).

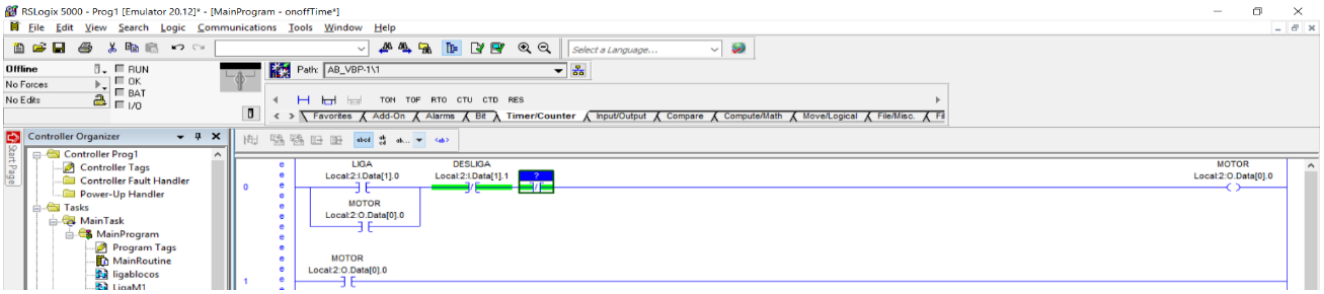
Figura 69 – Criação de *tags* (variáveis) do controlador



Fonte: Própria.

O esboço do programa pedido é representado na figura 70.

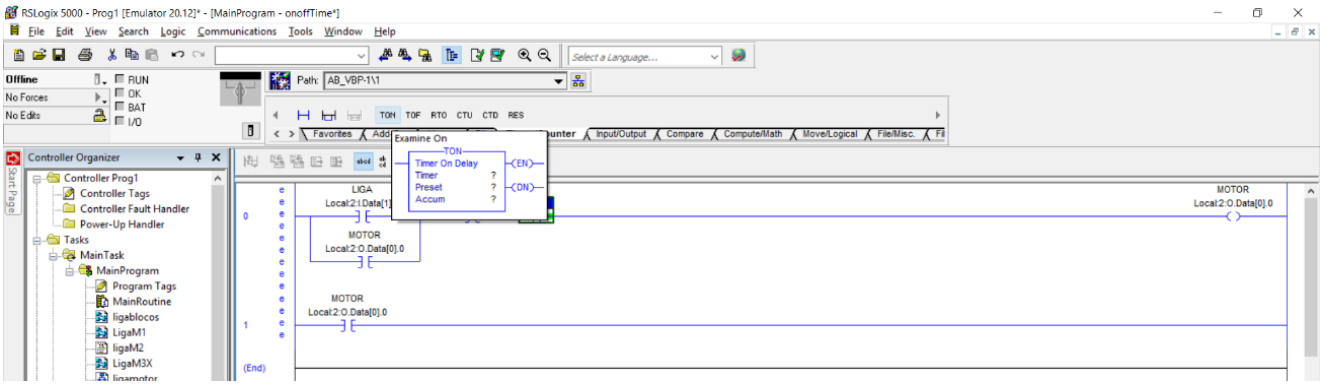
Figura 70 – Criação do terceiro programa



Fonte: Própria.

Torna-se necessário definir o temporizador, o que é feito clicando-se sobre a instrução homônima, de acordo com a figura 71.

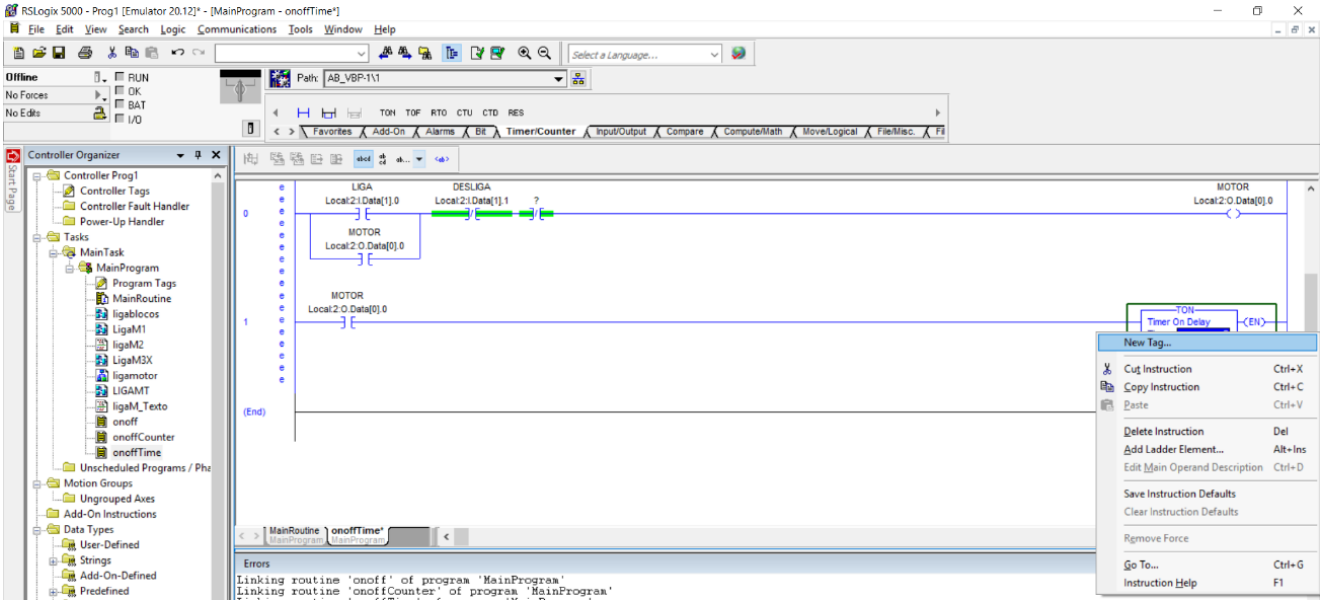
Figura 71 – Desenvolvimento do terceiro programa



Fonte: Própria.

A próxima etapa consiste em criar a *tagname* ou variável do temporizador, e isto é alcançado clicando-se com o botão direito do mouse dentro do campo Timer, situado no interior da instrução temporizadora TON, se onde marca a opção *New Tag*, pressionando-se enter em seguida (fig. 72).

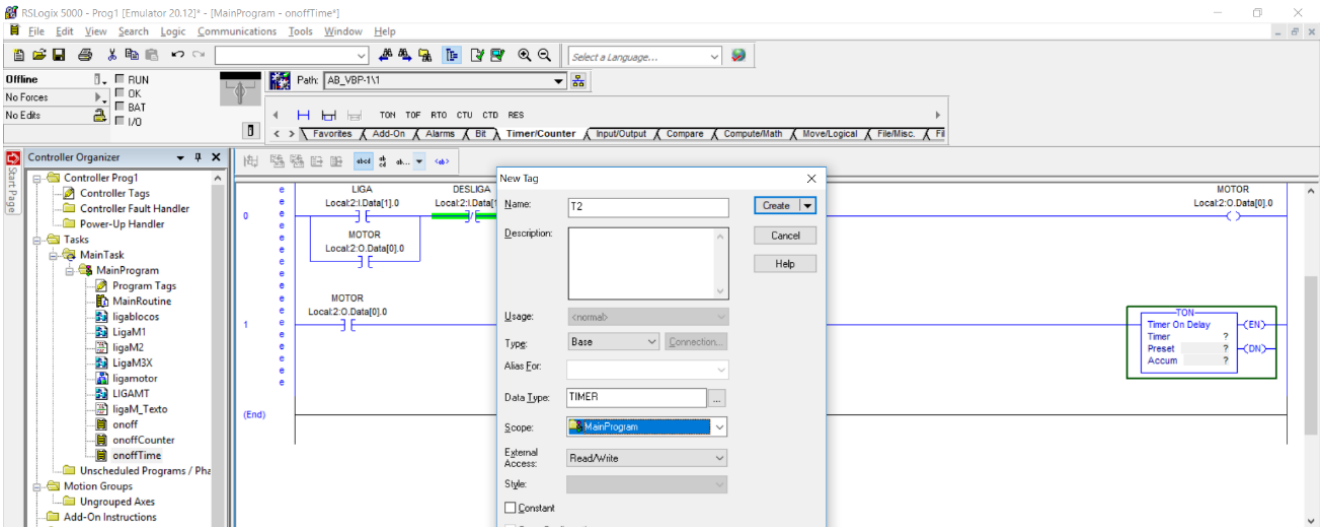
Figura 72 – Criando *tag* para o temporizador



Fonte: Própria.

No quadro que surgir digita-se T2 no campo *Name*, para informar o nome da variável criada, conforme figura 73. O tipo de dados (*Data_Type*) será automaticamente definido como *Timer*, uma vez que a variável foi elaborada no escopo de uma instrução temporizadora. É preciso estabelecer o escopo da variável, que pode ser local, escolhido por padrão através da opção *Main Program*.

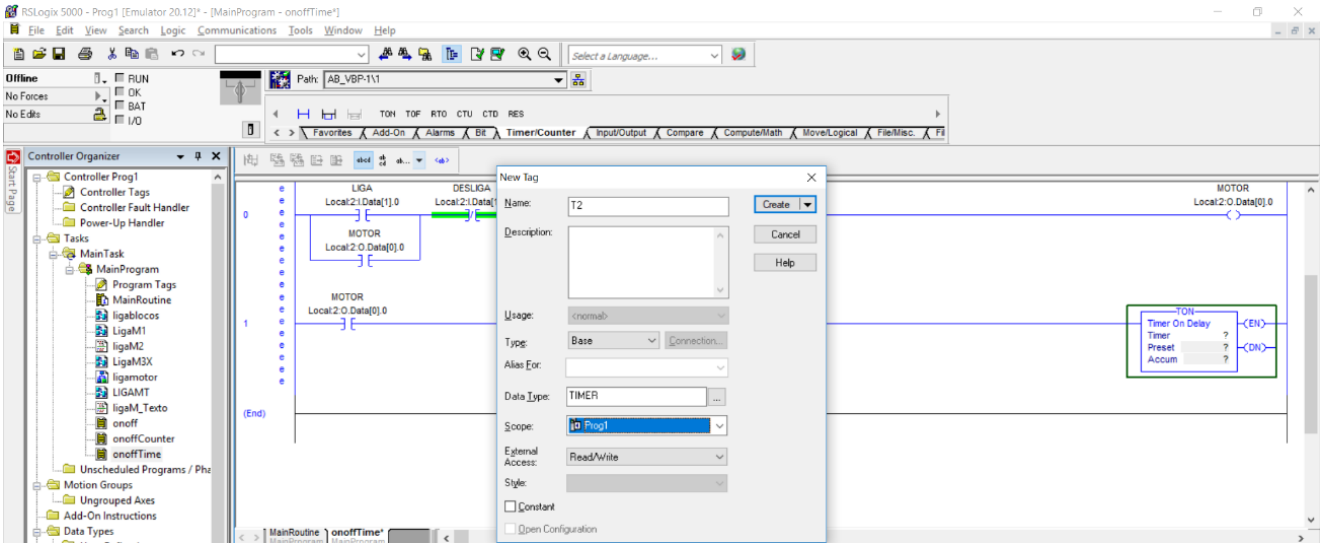
Figura 73 – Preenchimento dos campos da tag criada para o temporizador



Fonte: Própria.

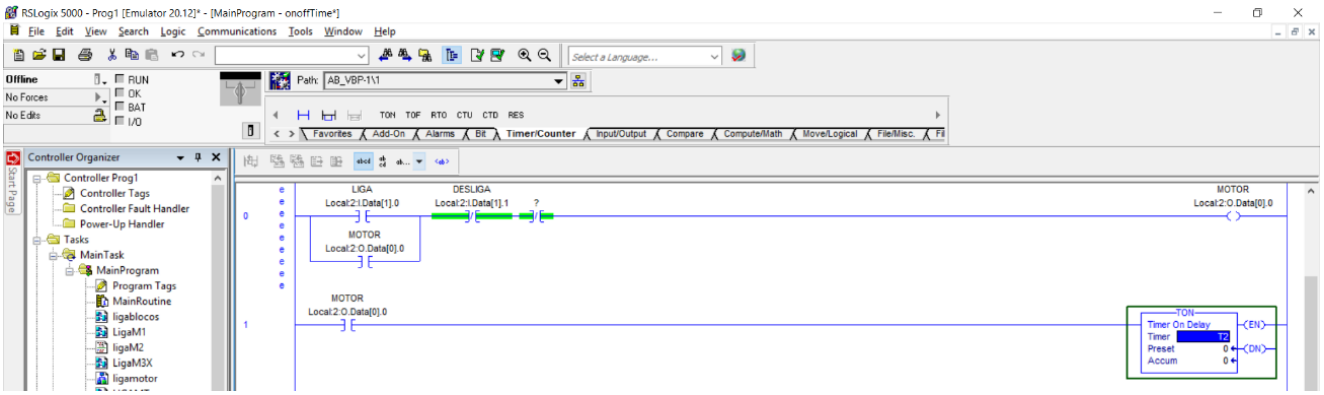
Como se deseja realizar comunicação com o sistema supervisório, é preciso modificar o escopo para o nome do programa, conferindo à tagname escopo global, após pressionar Create (fig.74), o que origina a figura 75.

Figura 74 – Mudança do escopo da tagname T2 para global



Fonte: Própria.

Figura 75 – Tagname T2 já criada



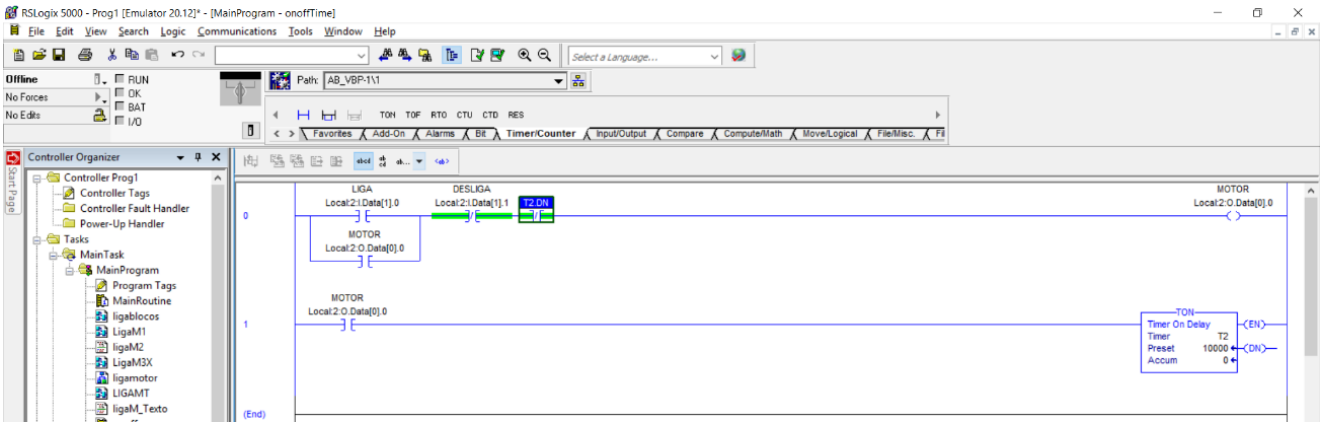
Fonte: Própria.

Clicando-se sobre *Controller Tags* pode-se observar a recém criada variável temporizadora T2, com seu escopo global.

Surge o momento de endereçar corretamente o contato normalmente fechado associado em série com as instruções da primeira linha do programa Ladder, cuja função é desativar automaticamente o motor, 10 segundos após acionado. Dessa forma, o endereço será T2.DN, uma vez que a saída DN do temporizador T2 será ativada após o tempo decorrido igualar o conteúdo estabelecido no campo PRESET, o qual receberá o valor 10000, uma vez que o mesmo é especificado em milissegundos.

Dessa forma, a solução do terceiro problema pode ser vista na figura 76.

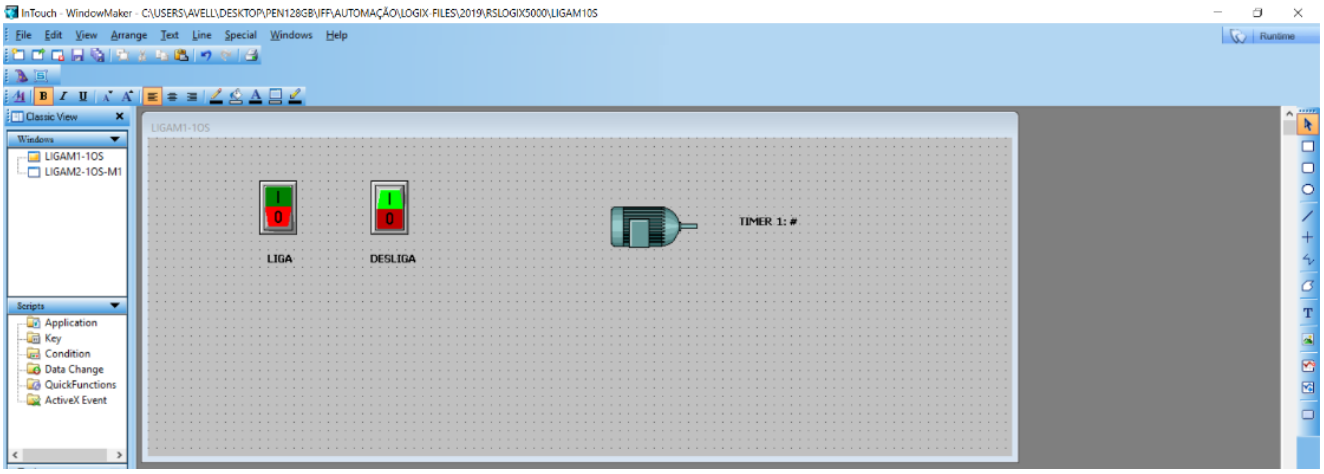
Figura 76 – Solução do terceiro problema



Fonte: Própria.

4) solicita-se o desenvolvimento da tela de sistema supervisório (figura 77) para o programa anterior, acompanhada das devidas configurações de objetos.

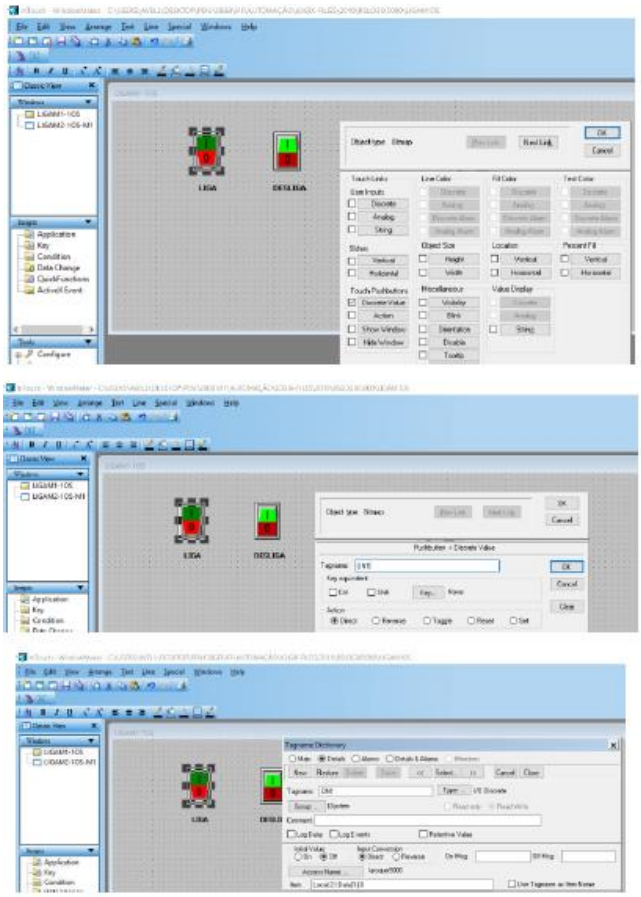
Figura 77 – Sistema supervisório do terceiro problema



Fonte: Própria.

A figura 78 traz três ilustrações, que apresentam as configurações do botão Liga, cujas instruções se encontram na página 33.

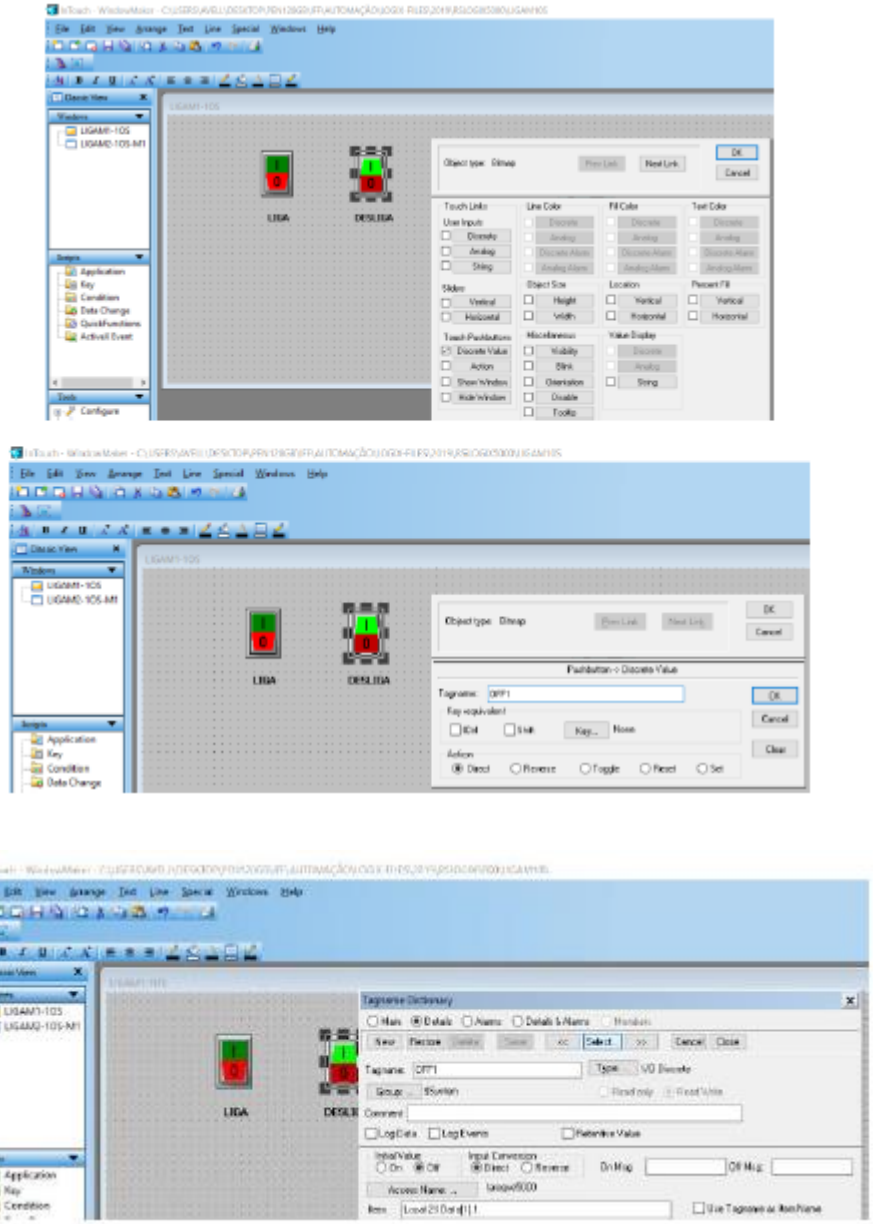
Figura 78 – Configurações do botão Liga



Fonte: Própria.

A figura 79 traz três ilustrações, que apresentam as configurações do botão Liga, cujas instruções se encontram na página 35.

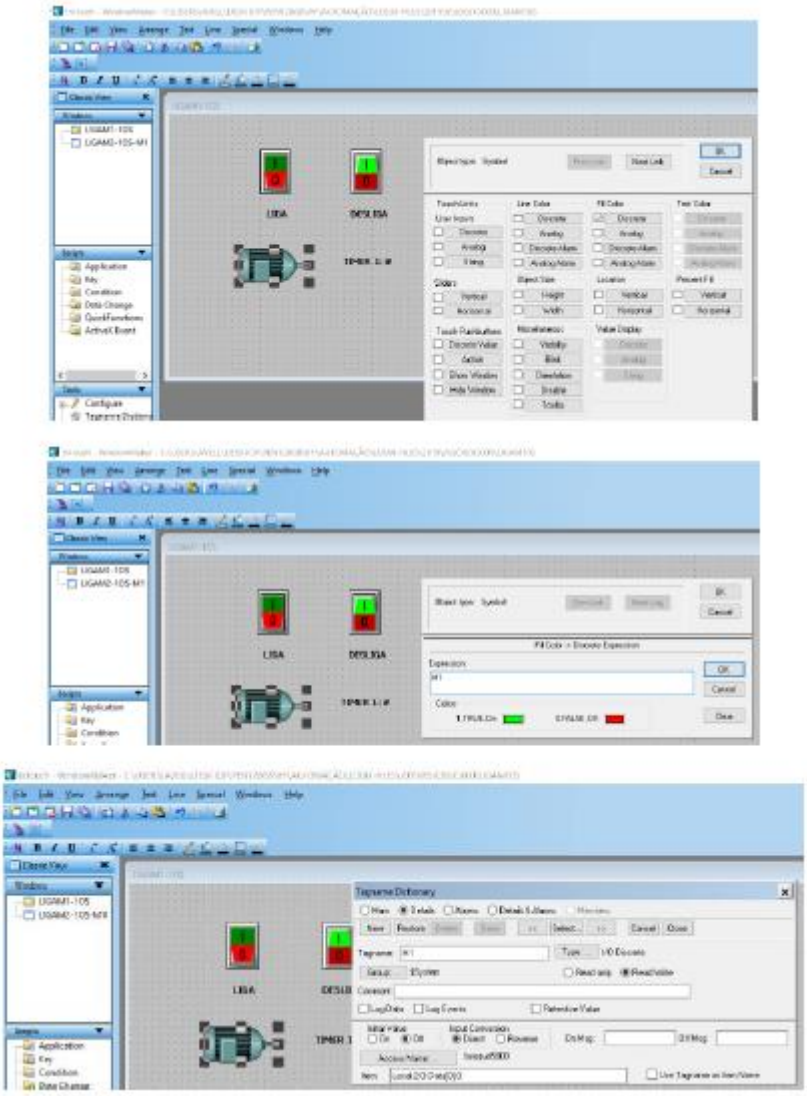
Figura 79 – Configurações do botão Desliga



Fonte: Própria.

A figura 80 traz três ilustrações, que apresentam as configurações do botão Liga, cujas instruções se encontram na página 33.

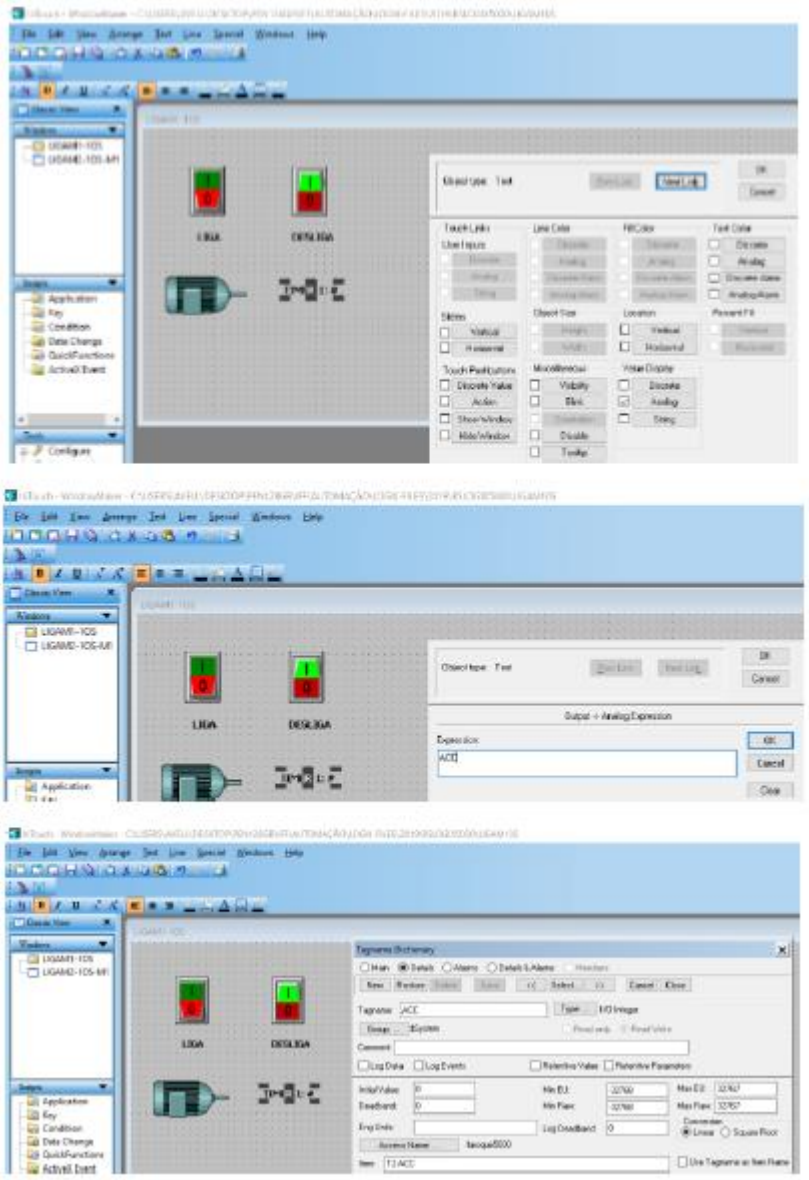
Figura 80 – Configurações do motor



Fonte: Própria.

Seguem as telas de configuração do caractere #, integrante do objeto texto TIMER 1: #, que exibirá o valor do campo ACC, o acumulador da instrução temporizadora T2, conforme figura 81. A propriedade onde será criada a *tagname* ACC é a *Analog*, integrante da classe *Value Display*. O tipo de ACC será definido como I/O Integer e seu Item receberá o endereço do registro ACC, pertencete ao temporizador T2.ACC, no programa ladder.

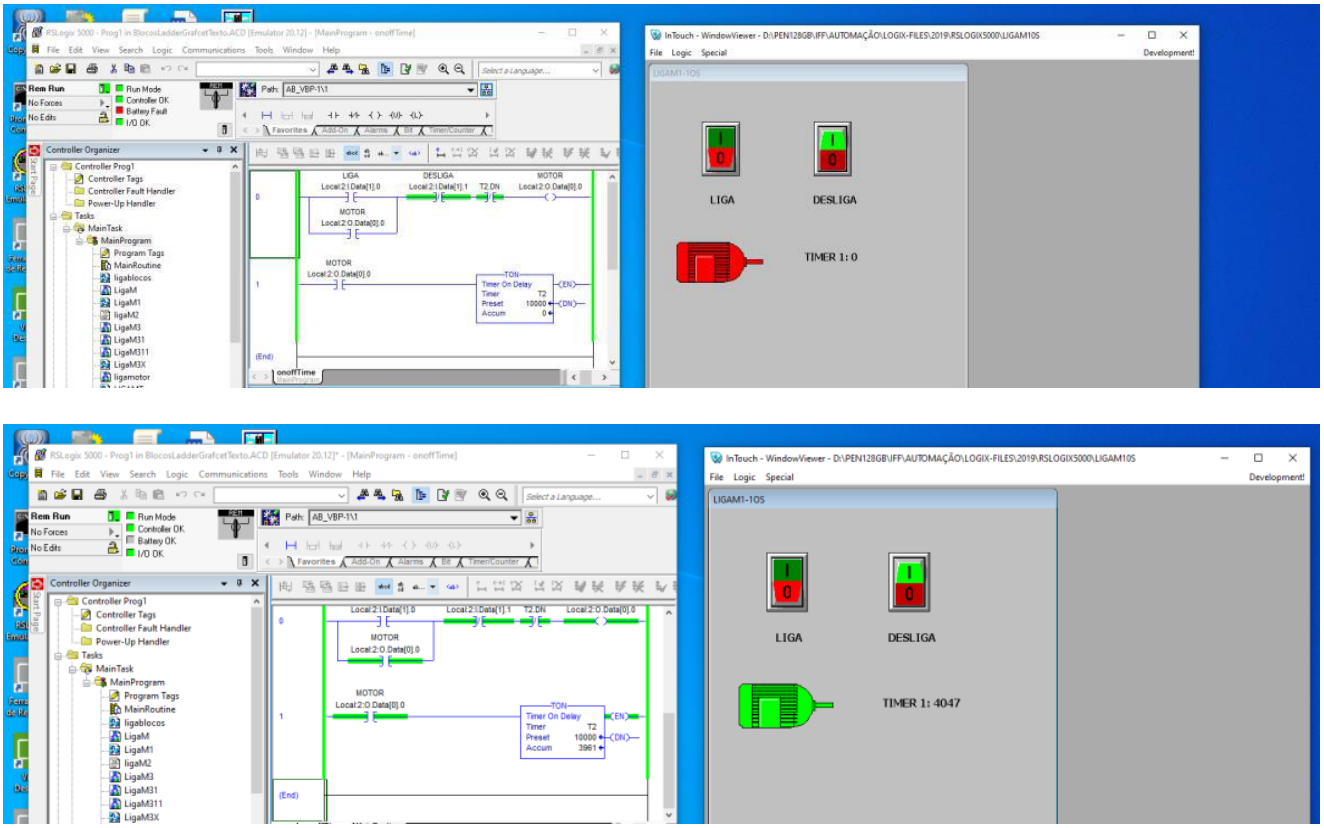
Figura 81 – Configurações do objeto texto TIMER 1: #



Fonte: Própria.

A terceira ilustração (fig. 81) mostra o nome de acesso laroque5000, cujas instruções para criação se encontram na página 35. Seguem telas do supervisório em funcionamento, controlando o programa Ladder, elaborado no ambiente de desenvolvimento integrado do RSLogix 5000 (fig. 82). A primeira ilustração representa o sistema desligado (fig. 82), enquanto a segunda traz o estado do conjunto após o acionamento do botão Liga.

Figura 82 – Comunicação entre supervisório e programa Ladder.



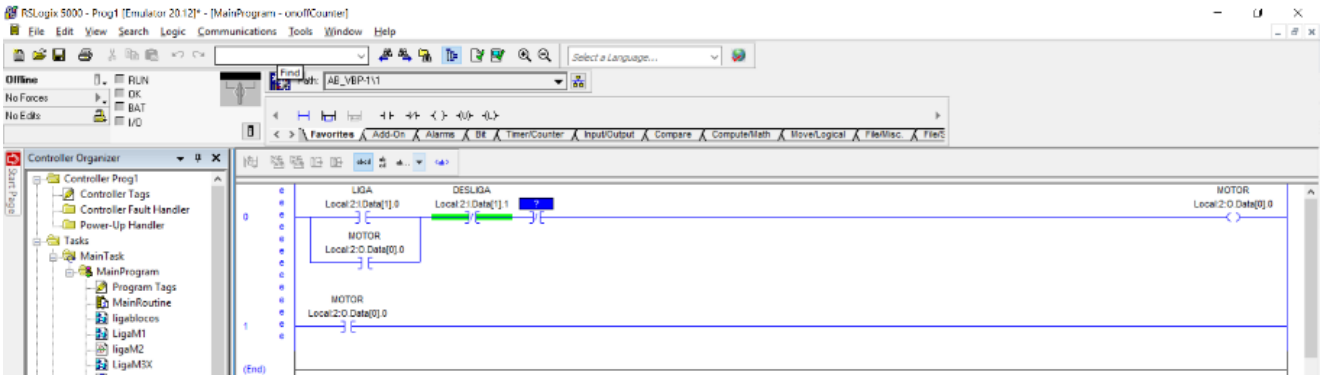
Fonte: Própria.

Quando o registro Accum alcançar o valor estabelecido em PRESET, de 10000 milisegundos, ou 10 s, a saída DN do temporizador será ativada, comutando do estado de repouso normalmente fechado (NF) o contato T2.DN, evento que impedirá a continuidade da circulação de corrente virtual, e desativará automaticamente a saída do motor, retornando sua cor de preenchimento para vermelho, indicativo de inatividade, enquanto a cor verde do motor representa seu funcionamento.

5) Pede-se elaborar programa em linguagem Ladder, que permite ligar e desligar manualmente um motor, sendo que o mesmo deve ser automaticamente desativado, após entrar em operação por duas vezes.

Segue um esboço inicial para o código pedido, onde o contato normalmente fechado, identificado por um sinal de interrogação a ele associado, receberá o endereço da saída DN do contador que será inserido, para desligar automaticamente o motor, após o mesmo funcionar por duas vezes, ou seja, na terceira tentativa de acionamento (fig. 83).

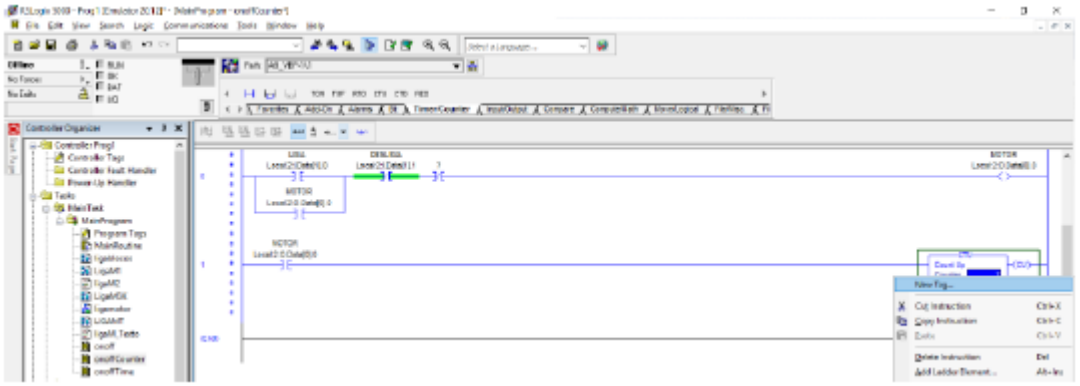
Figura 83 – Início da programação do problema 5



Fonte: Própria.

Procede-se à inserção da instrução de contagem, na qual se clica com o botão direito do mouse, para criar nova tag, conforme figura 84.

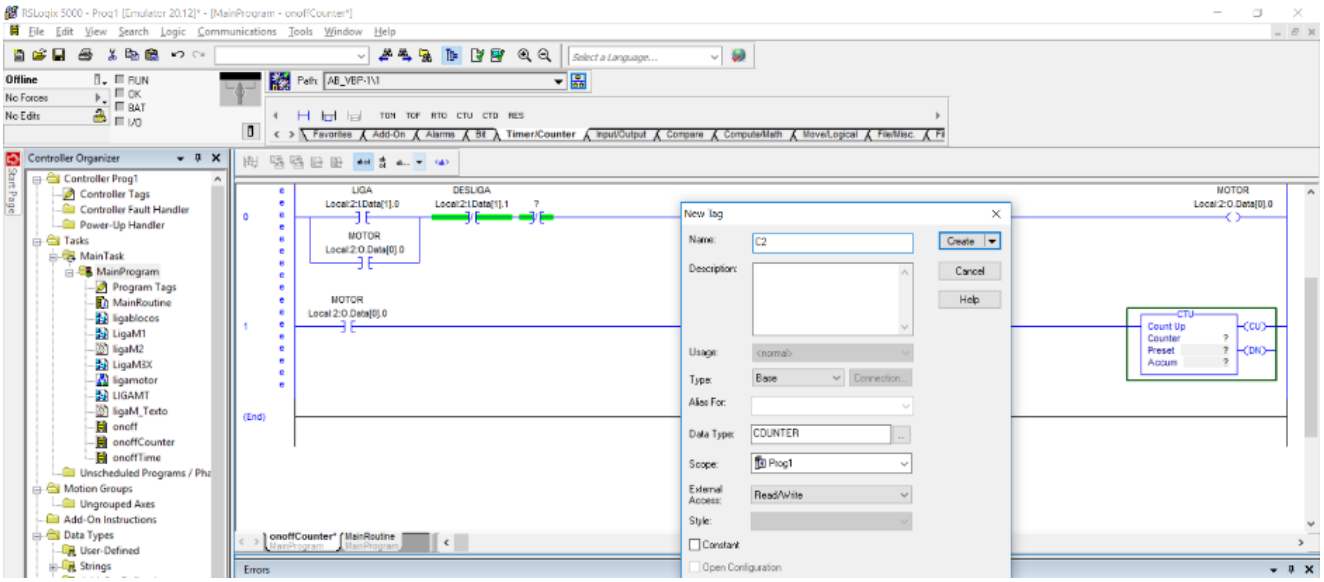
Figura 84 – Inserção da instrução de contagem crescente (CTU)



Fonte: Própria.

Digita-se na tag o nome C2, definindo escopo global (evidenciado pelo nome do arquivo do projeto, que é Prog1) para a variável criada, com o intuito de viabilizar a comunicação com o sistema supervisório (fig. 85)

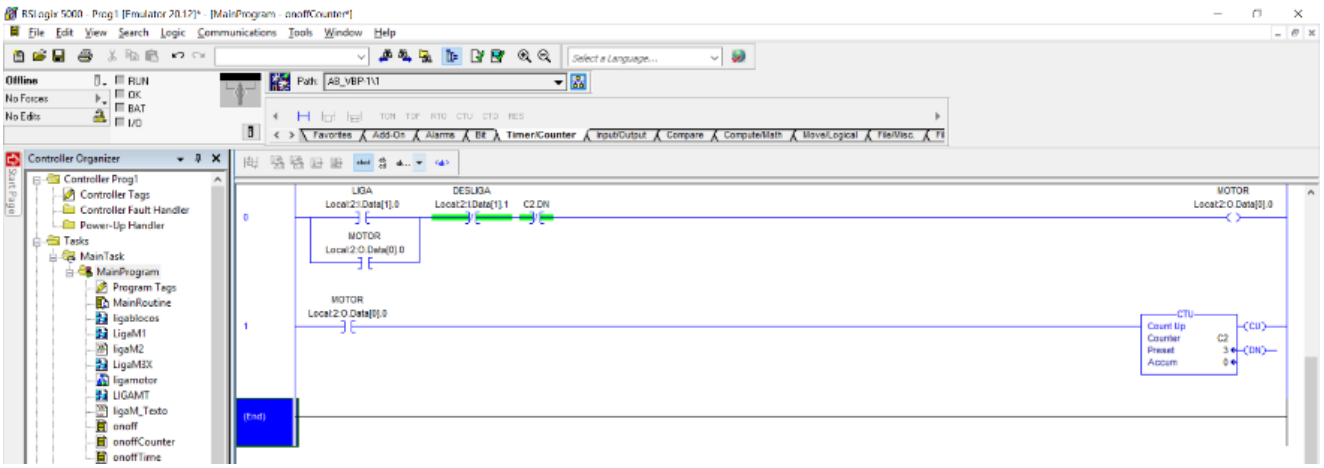
Figura 83 – Configurações da instrução CTU



Fonte: Própria.

Digita-se o número 3 no campo PRESET, preenchendo-se o endereço do último contato normalmente fechado (NF) da primeira linha com C2.DN (fig. 86).

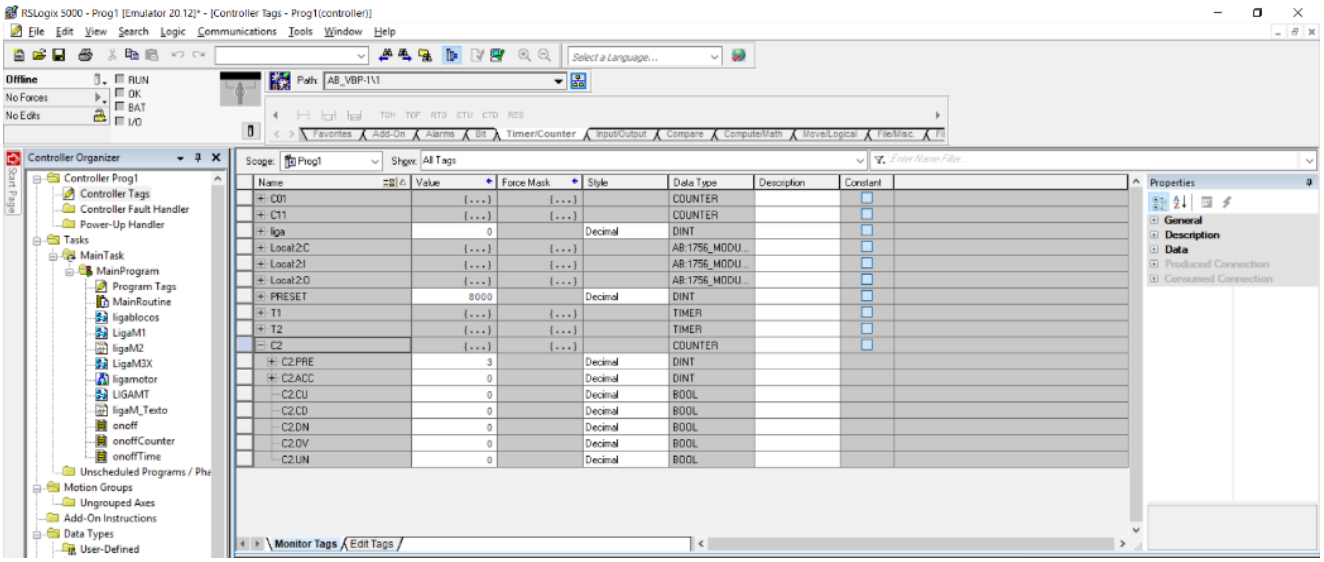
Figura 86 – Configurações da instrução CTU



Fonte: Própria.

Verifica-se a criação da variável global C2, clicando-se em Controller Tags, de acordo com a figura 87.

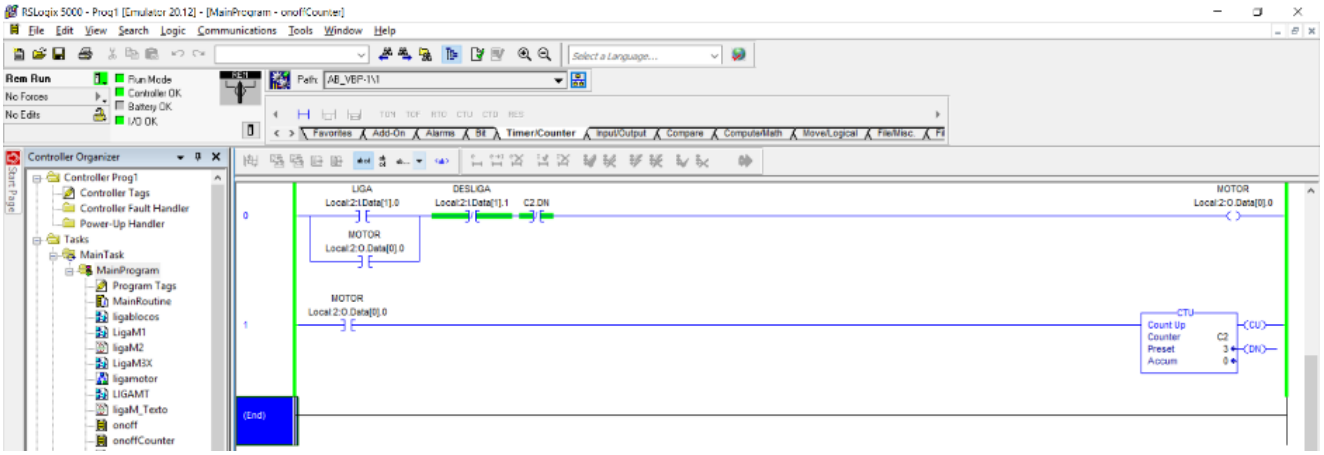
Figura 87 – Verificando a criação da *tagname* C2



Fonte: Própria.

Efetua-se o download, procedendo-se à execução do programa (fig. 88)

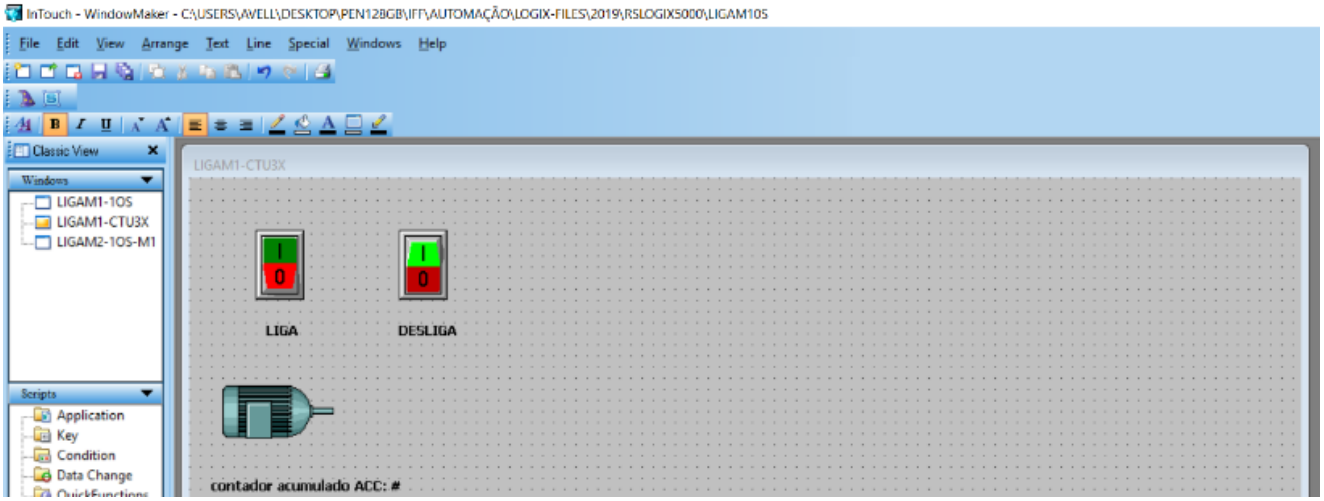
Figura 88 – Programa Ladder em execução



Fonte: Própria.

6) Solicita-se o desenvolvimento da tela de sistema supervisório (fig. 89) para o programa anterior, acompanhada das devidas configurações de objetos.

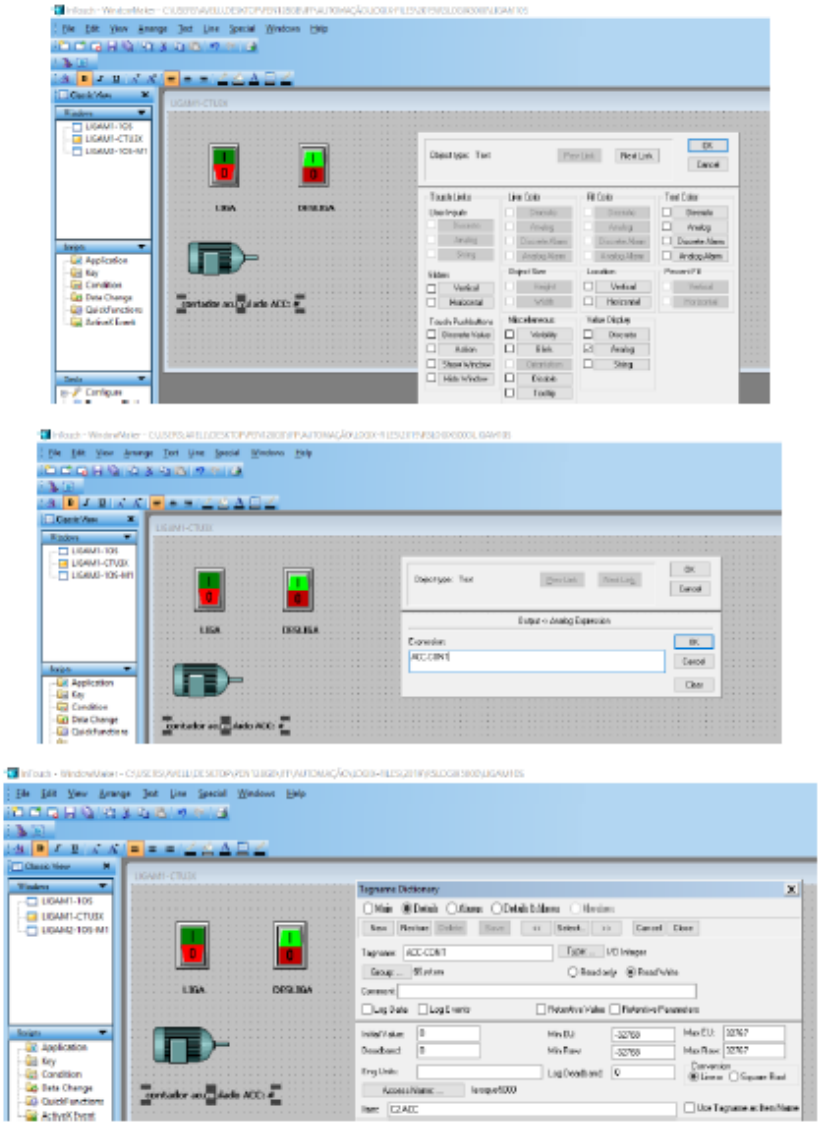
Figura 89 – Sistema supervisório do programa ladder do item 5



Fonte: Própria.

A única configuração diferente do último supervisório desenvolvido (vide problema número 4, na página 42) é a da exibição do acumulador de contagens, cujas ilustrações são exibidas na figura 90.

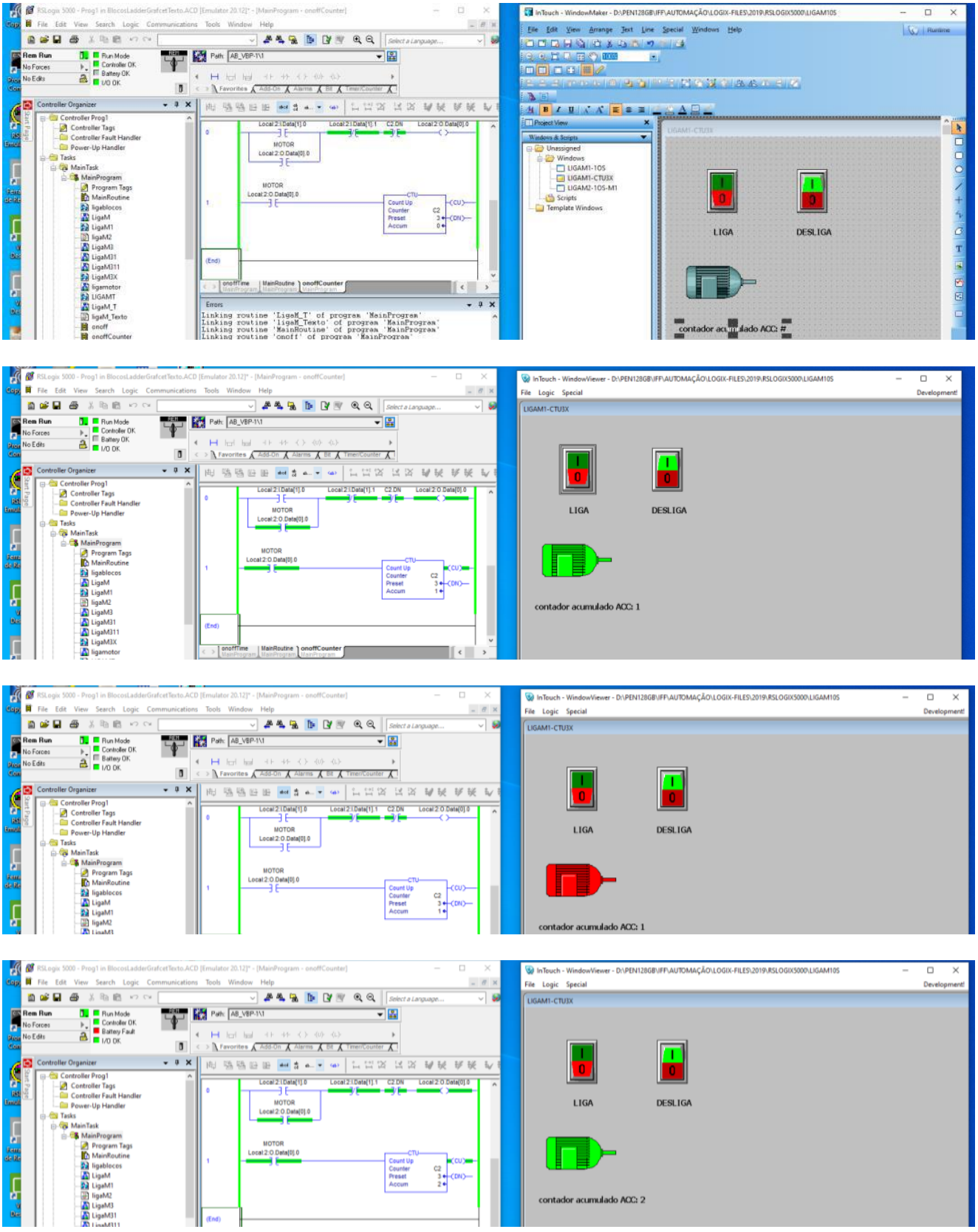
Figura 91 – Telas de configuração do acumulador de contagens

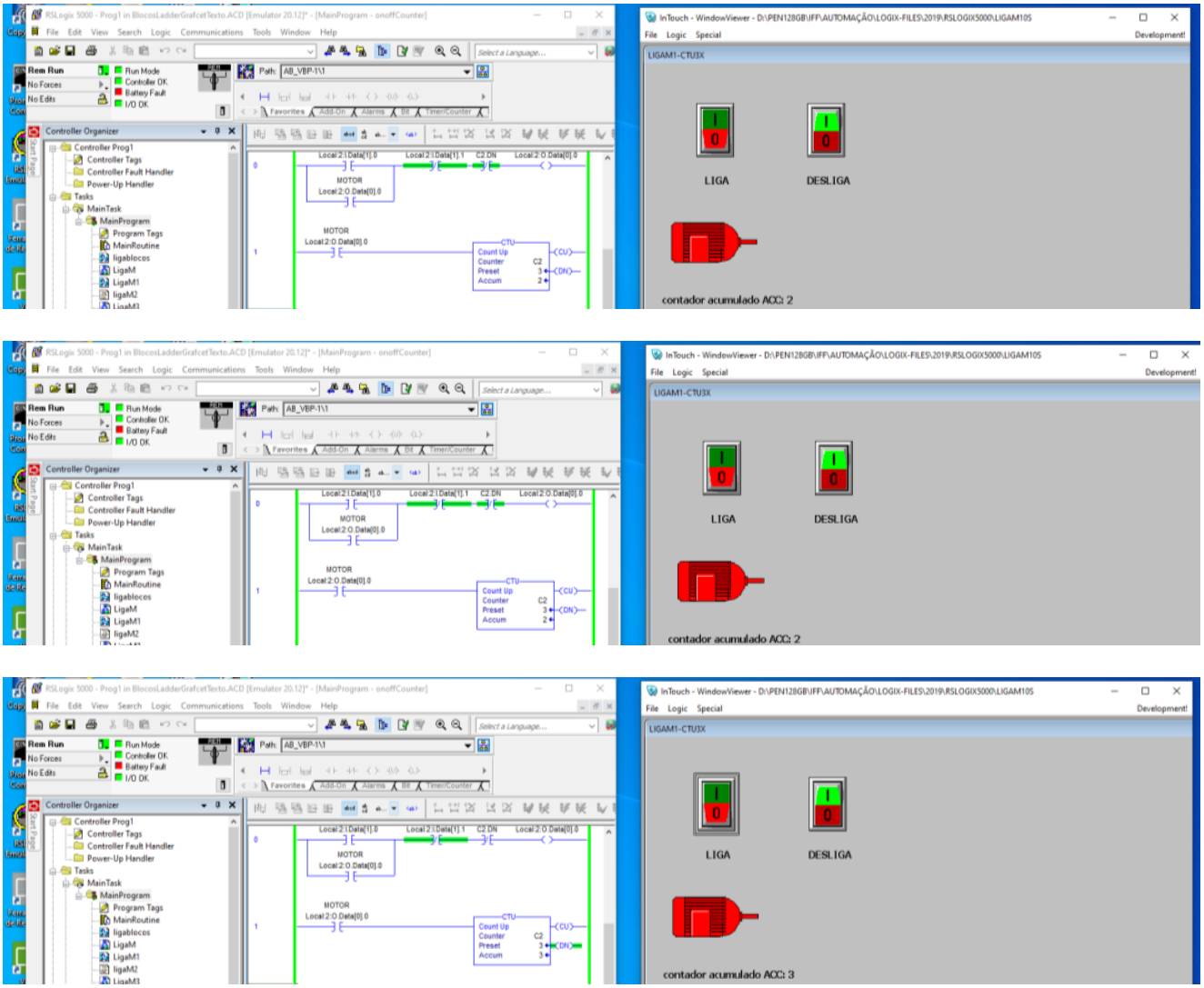


Fonte: Própria.

Seguem as 7 telas do supervisório em operação com o programa (fig. 92).

Figura 92 – Telas do supervisório do programa





Fonte: Própria.

A primeira tela da figura 92 apresenta o programa ladder e o sistema supervisório em execução. A segunda tela mostra o estado deles, após o acionamento do botão Liga no Intouch. Percebe-se a comutação do estado de repouso normalmente aberto (NA) para fechado, no botão liga no programa ladder, o primeiro a esquerda da linha 0.

Essa comutação possibilitará circulação de corrente virtual até a saída, alterando o estado de repouso os contatos que tenham seu endereço (Local:2:O.Data[0].0), como o contato Motor da linha 1. Dessa forma, esse contato comutará para fechado, realizando transição de borda de subida (Mudança de nível logico baixo para alto) junto à entrada do contador, fazendo com que o acumulador da instrução CTU seja incrementado de uma unidade.

Para contabilizar novo acionamento, é preciso antes desativar o motor, pressionando-se o botão desliga no Intouch, alterando o estado de repouso normalmente fechado (NF) para aberto do contato

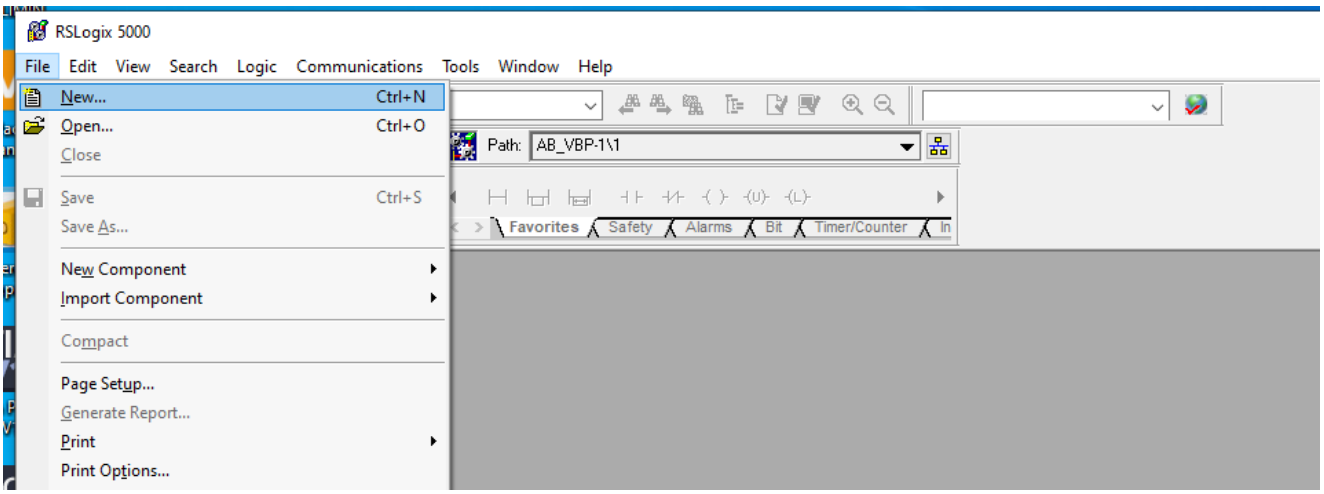
virtual Local:2:I.Data[1].1, impedindo a continuidade de circulação de corrente virtual, desativando o motor, conforme visto na terceira ilustração da figura 92.

Após sucessivos acionamentos e desativações do motor, chegará o momento em que o acumulador do contador igualará o valor estipulado em seu campo Preset, o que causará o acionando da saída DN de CTU, modificando o estado de repouso NF do contato da linha 0 que comuta para aberto, impedindo novos acionamentos do motor, conforme o enunciado do problema 5.

7) Pede-se elaborar programa em linguagem Ladder, que permite ligar e desligar manualmente um motor, sendo que o mesmo deve ser automaticamente desativado, após entrar em operação por mais de duas vezes, ou depois de decorridos 10 segundos de funcionamento contínuo.

Após carregar os softwares RSLinx e RSLogix 5000 Emulate, deve-se abrir o ambiente de desenvolvimento integrado RSLogix 5000, e criar novo projeto, clicando na opção New, que surgirá após selecionar o item File (Fig. 93).

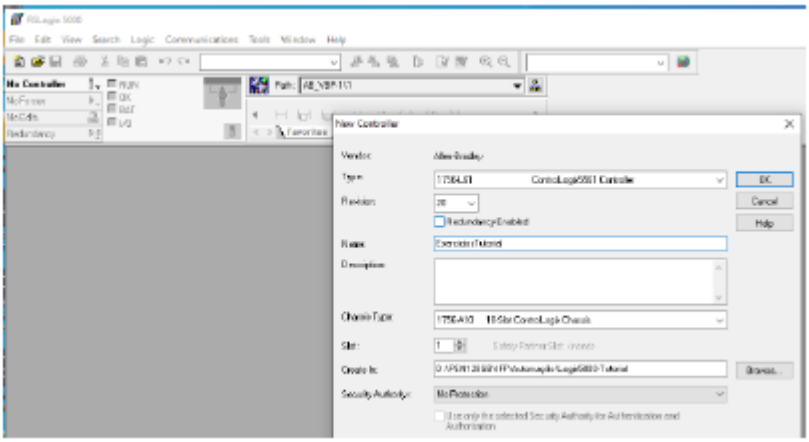
Figura 93 – Programação Ladder



Fonte: Própria.

Com isso, surgirá tela (fig. 94), a qual será preenchida com dados como tipo do controlador (1756-L61), nome do arquivo criado (ExerciciosTutorial) e tipo do Chassis (1756-A10), que é o modelo de CLP escolhido na programação.

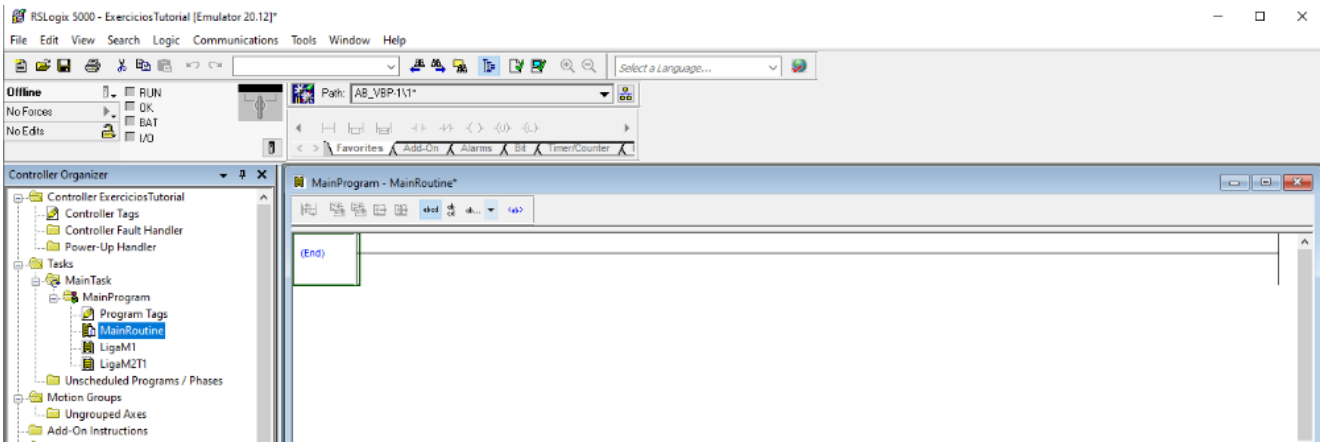
Figura 94 – Configurações da Tela inicial do RSLogix 5000



Fonte: Própria.

Surgirá uma tela, na qual se deve clicar duas vezes sobre a rotina principal (MainRoutine), originando a figura 95.

Figura 95 – Rotina principal

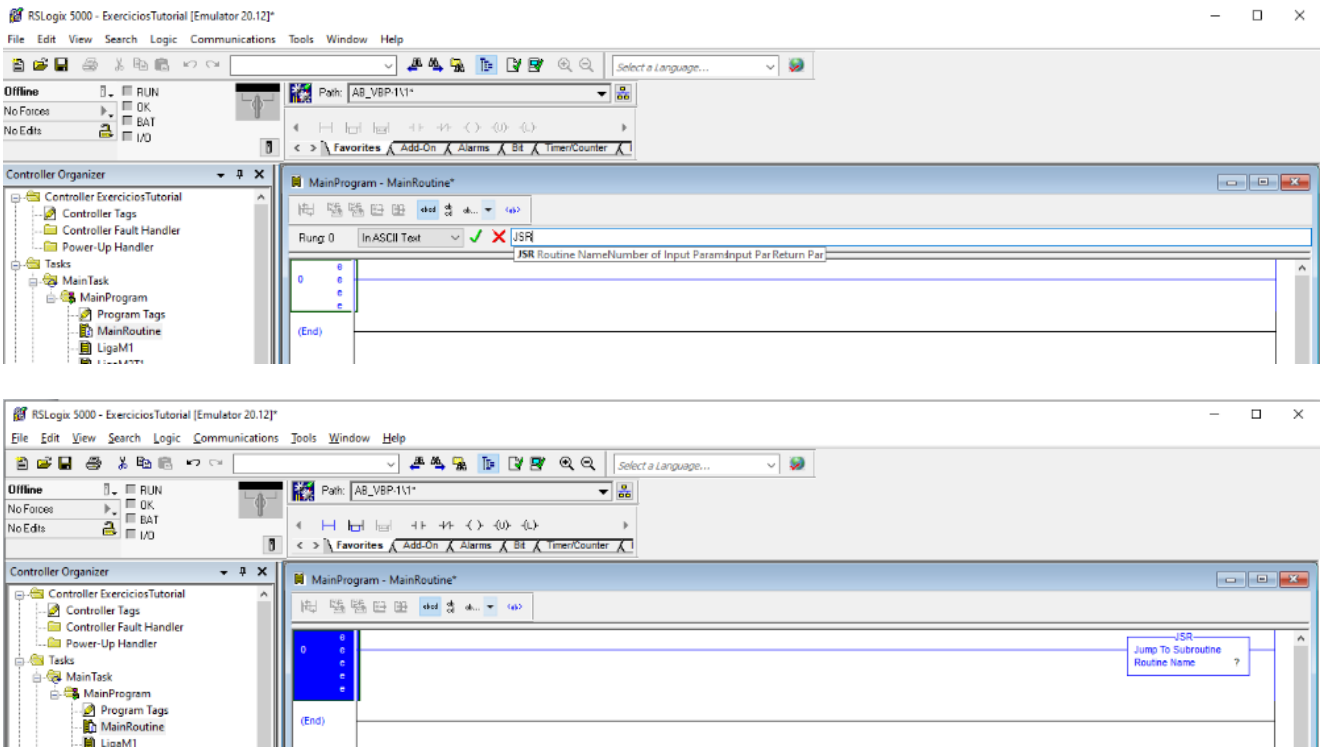


Fonte: Própria.

Efetua-se a seguir clique duplo sobre o retângulo inicial (onde se percebe a palavra End) do programa ladder a ser elaborado na rotina principal, alterando a tela de acordo com a primeira ilustração da figura 96, proporcionando condições para que seja digitada a instrução JSR, que direcionará o fluxo de operação do ambiente de desenvolvimento integrado (IDE) da rotina principal do RSLogix 5000 para o programa desejado, que pode ser elaborado em qualquer linguagem especificada pela norma IEC 61131-1.

Após digitar a instrução JSR conforme figura 96, a tela dará origem à segunda ilustração, onde é possível selecionar o nome da rotina para a qual o RSLogix 5000 direcionará seu fluxo de interpretação de códigos.

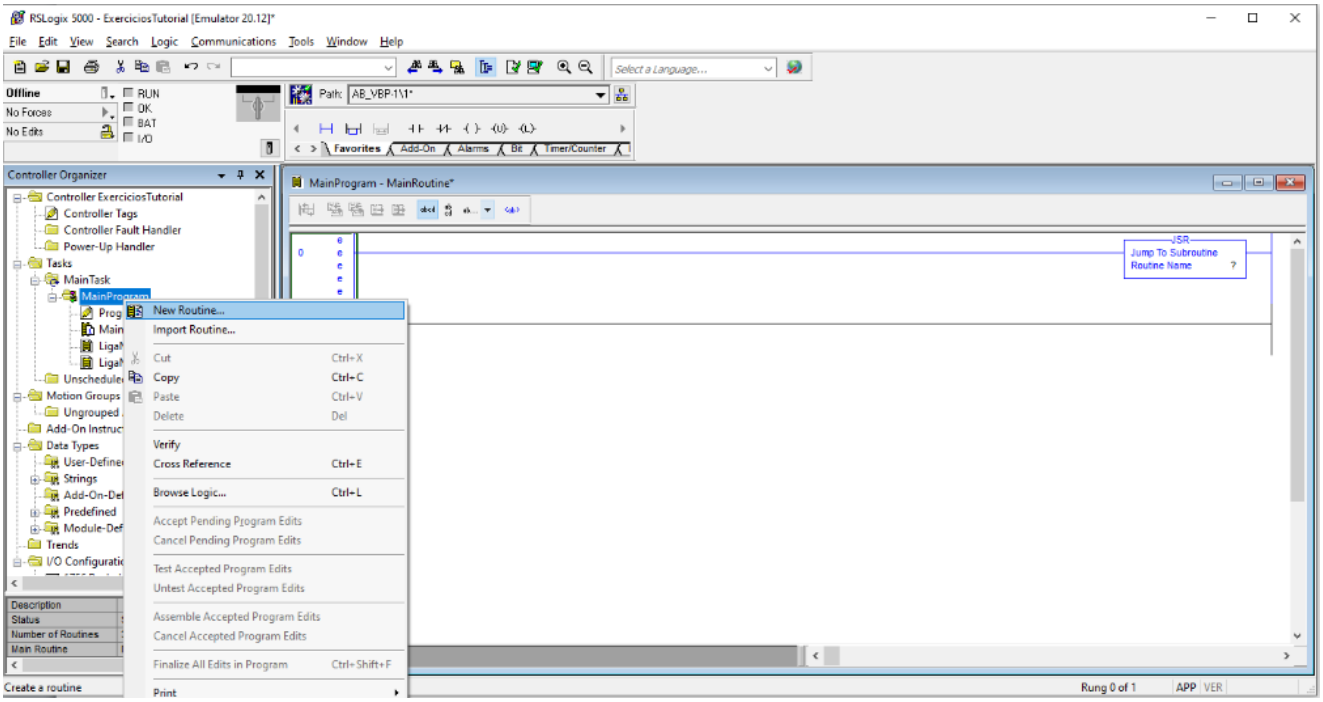
Figura 96 – Configurando instrução JSR na rotina principal



Fonte: Própria.

Para criar o código que realizará os pedidos do enunciado deste exercício, clica-se com o botão direito sobre a aba *Main Program* e, no quadro de opções que surgir, marca-se *New Routine* (fig. 97)

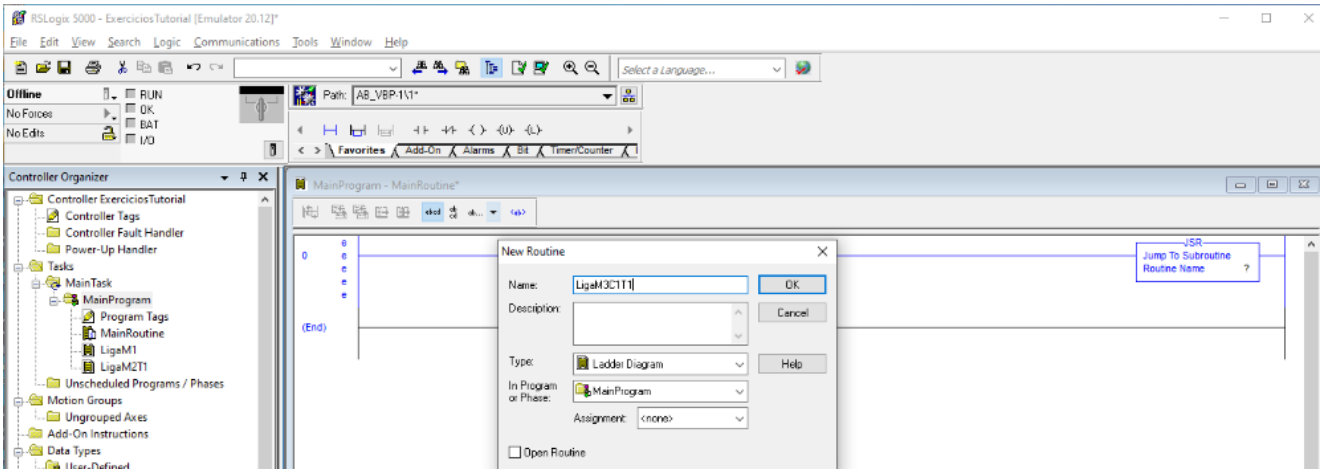
Figura 97 – Definição do nome do programa



Fonte: Própria.

Digita-se o nome do programa criado, que será LigaM3T1C1 (fig.98).

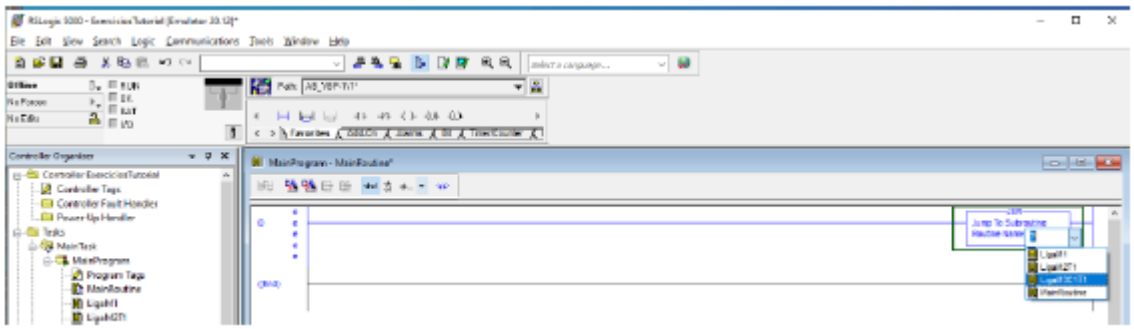
Figura 98 – Definição do nome do programa



Fonte: Própria.

A seguir, retorna-se a *Main Routine*, especificando na instrução JSR (*jump tp subroutine)* o nome da subrotina para qual o fluxo de instruções da rotina principal será redirecionado (fig. 99).

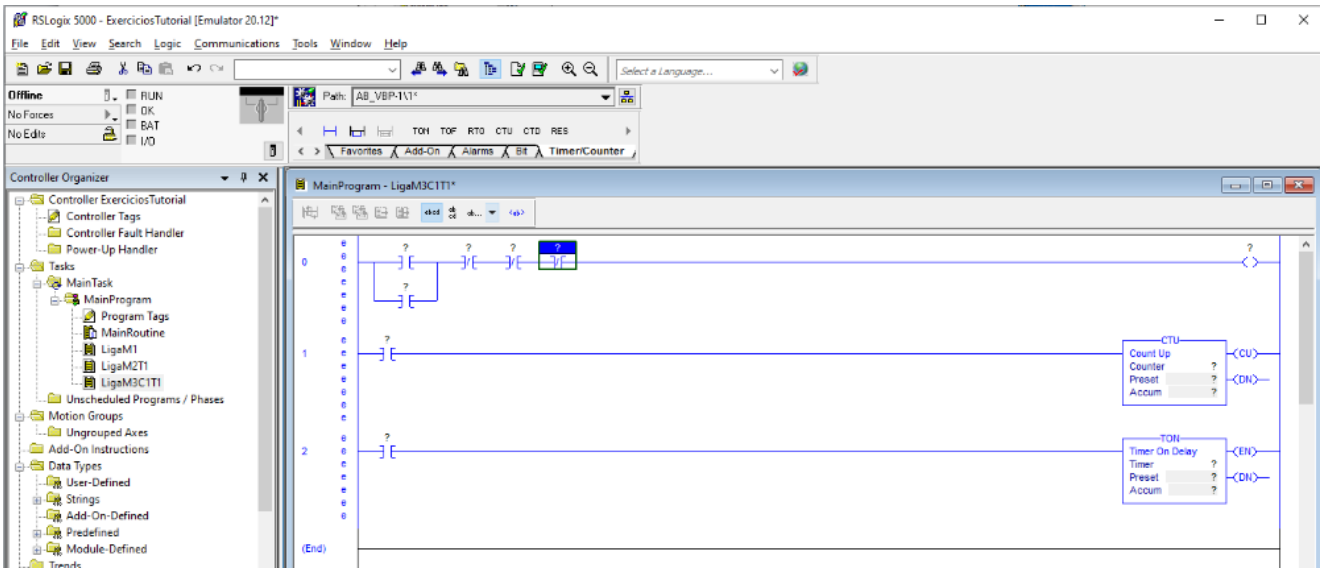
Figura 99 – Selecionado o programa criado para abertura na rotina principal JSR



Fonte: Própria.

A seguir clica-se duas vezes sobre a rotima LigaM3C1T1, para desenvolver o programa ladder que se pede no exercício, cujo esboço de solução está na figura 100.

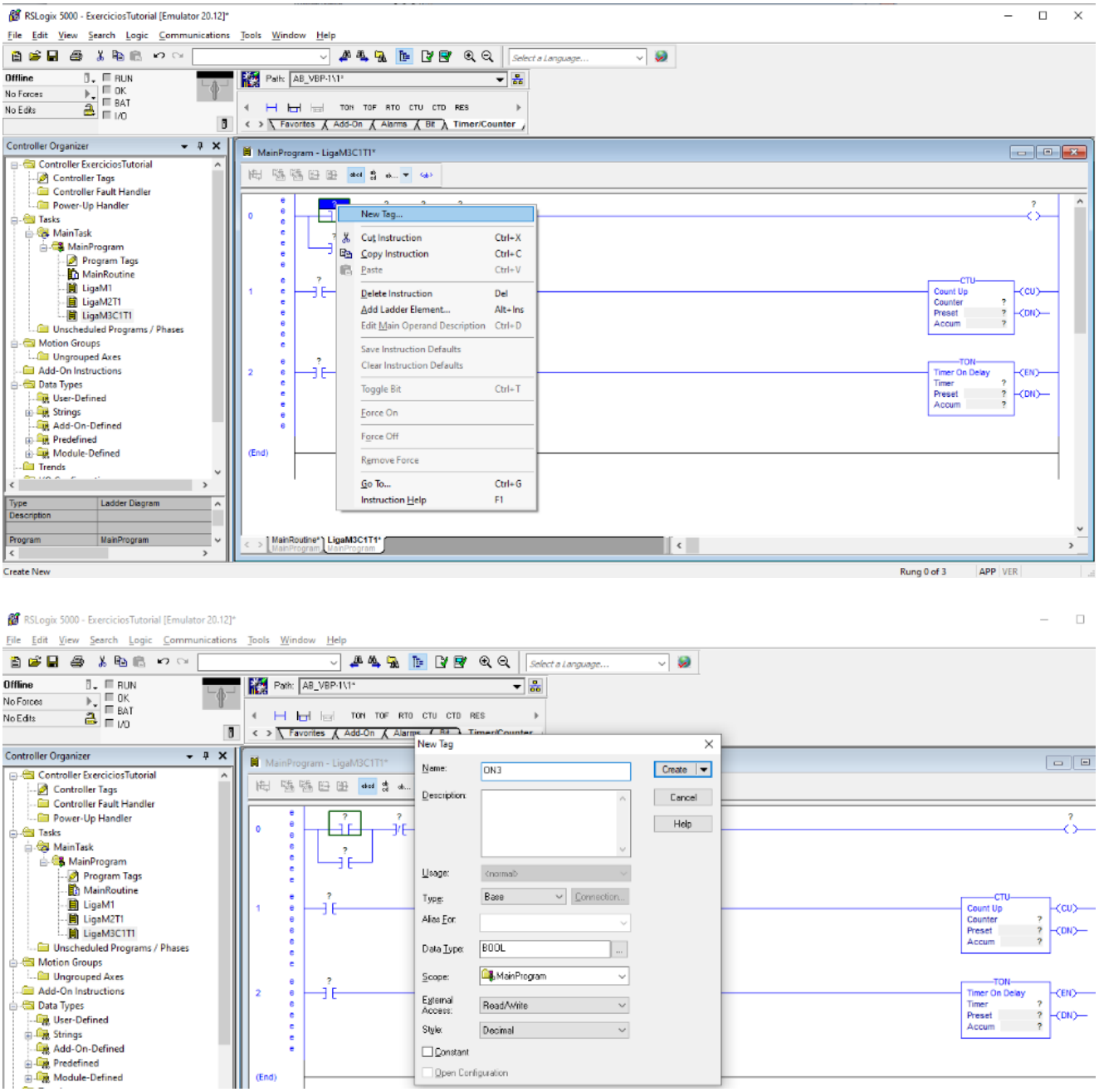
Figura 100 – Solução inicial para o programa solicitado.



Fonte: Própria.

Em seguida procede-se à criação das *tags* (rótulos) de cada elemento inserido no código ladder, que são uma espécie de variável, que possibilita ao interpretador identificar cada instrução do programa. O contato normalmente aberto (NA) da linha 0 representa o botão liga do motor. Para criar seu respectivo rótulo (*tag*), clica-se com o botão direito sobre o mesmo, selecionando a opção *New Tag*, conforme primeira ilustração da figura 97. Isso fará com que surja outro quadro, intitulado *New Tag*, no qual se especifica o nome do rótulo (ON3), o escopo da variável (se é global ou se restringe ao programa em questão) e o tipo de dados, enquanto os demais parâmetros já vêm configurados, de acordo com a segunda ilustração (fig. 101).

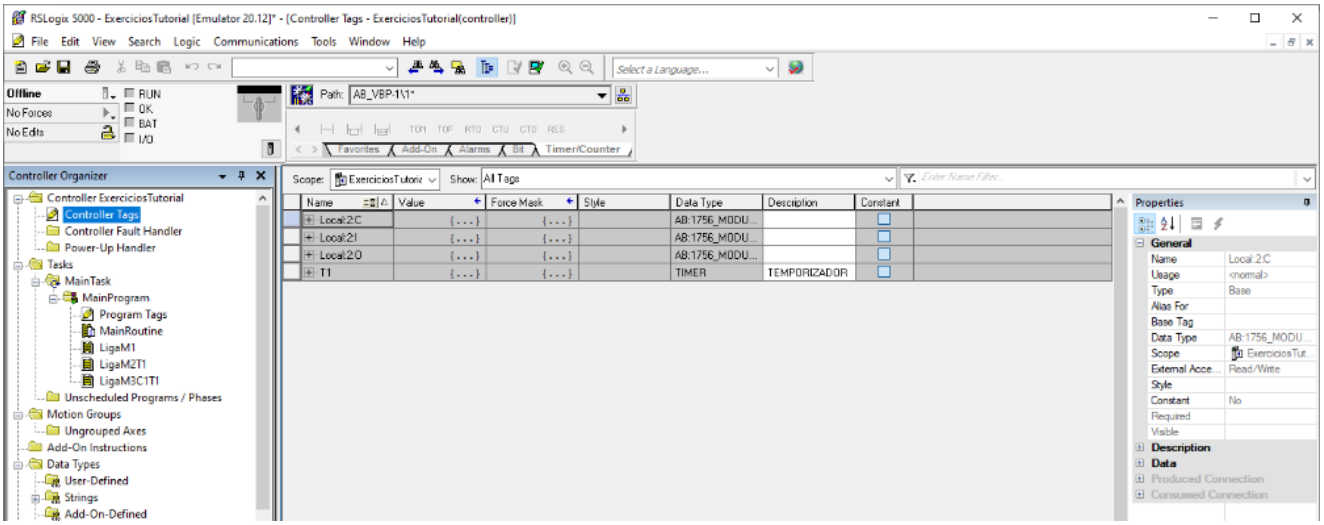
Figura 101 – Configuração de contatos do programa



Fonte: Própria.

Após digitar o nome da *tag*, deve-se pressionar o botão *Create,* para confirmar a criação do rótulo. Entretanto, devido a possibilidade de Acessar o conteúdo de cada bit dos rótulos, recomenda-se criar *tags* clicando-se duas vezes sobre o ícone *Controller Tags*, localizado no topo do quadro *Controller Organizer*, conforme figura 102.

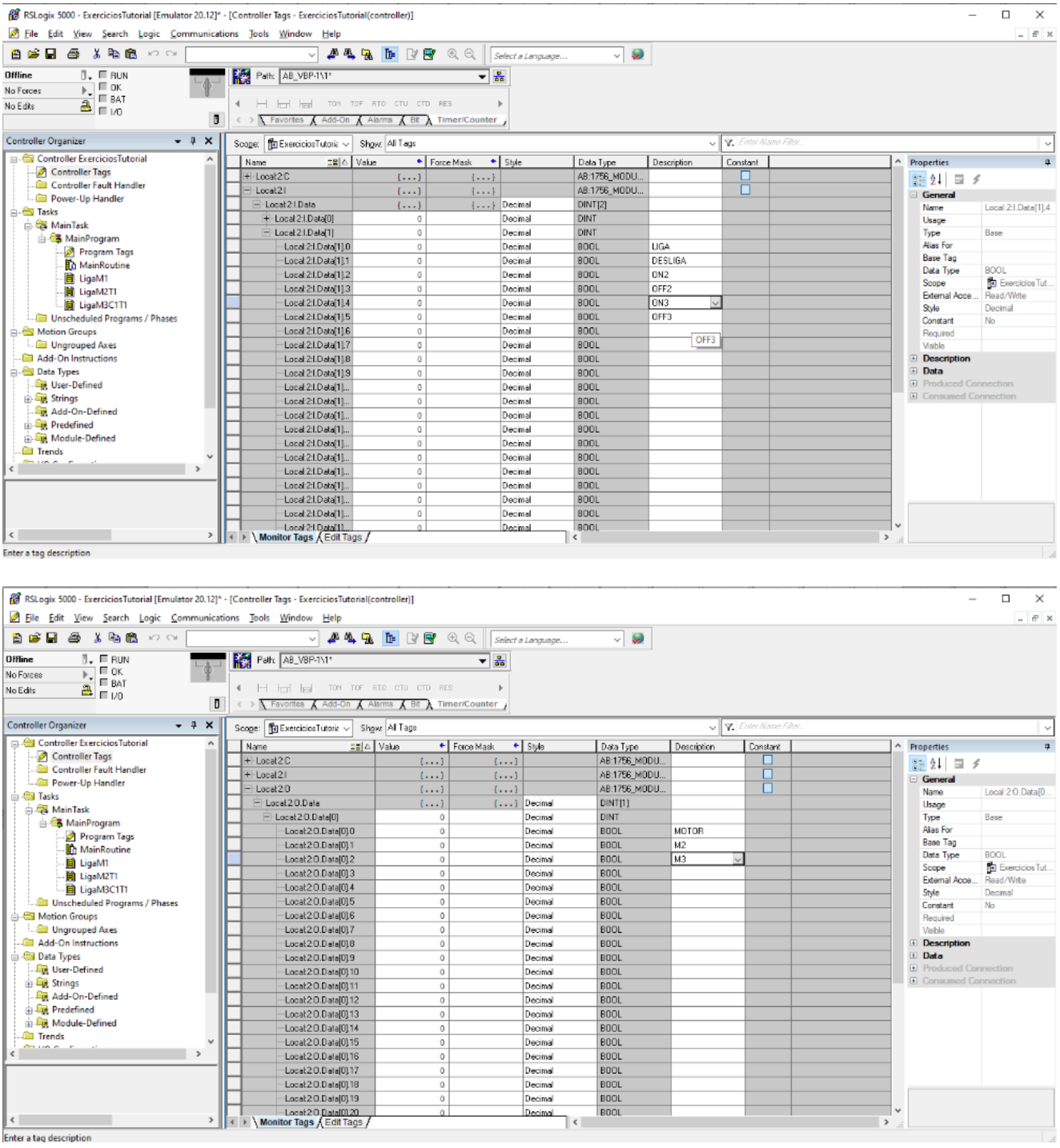
Figura 102 – Verificação das *tags* do controlador



Fonte: Própria.

Accessam-se as palavras de entrada clicando-se sobre o caractere +, ao lado da palavra Local 2:I, obtendo as seguintes configurações, onde se atribui aos endereços Local:2:I.Data[1].4 e Local:2:I.Data[1].5 as descrições ON3 e OFF3, relacionadas aos contatos responsáveis por ligar e desligar o motor, respectivamente, de acordo com a primeira ilustração da figura 99. Acessam-se a seguir os conjuntos de bits da palavra de saída do controlador, atribuindo-se ao endereço Local:2:O.Data[0].2 a descrição M3, que associa a terceira saída ao controle do motor desta atividade, conforme segunda ilustração (fig. 103).

Figura 103 – Verificação das tags do controlador

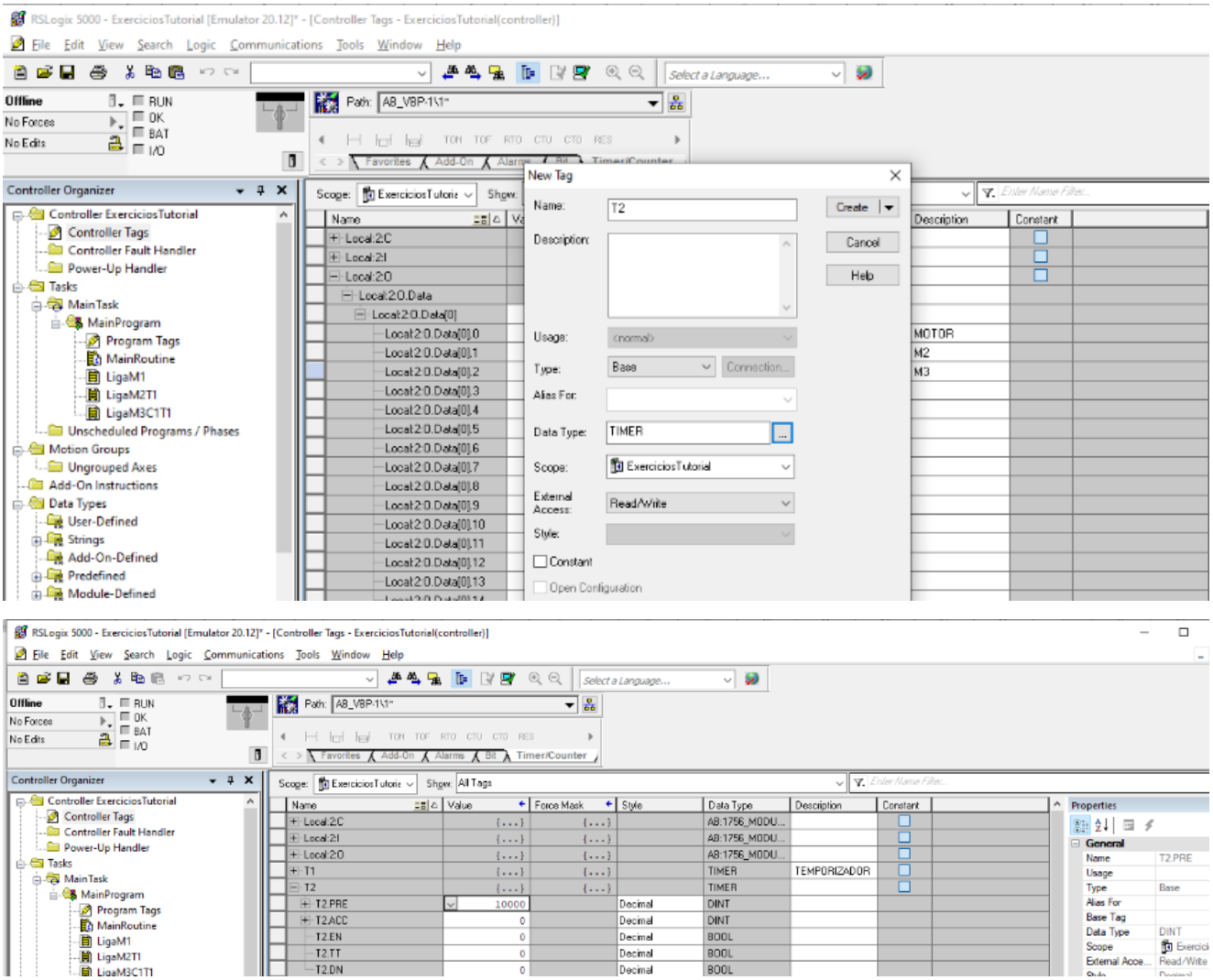


Fonte: Própria.

Precisa-se agora efetuar a criação da *tag* associada à temporização. Para tanto, clica-se com o botão direito sobre *Controller Tags*, atribuindo a este rótulo as seguintes configurações, onde T2 é o nome da variável, TIMER é seu tipo de dados, e ExerciciosTutorial define seu escopo como global, válido em todos os códigos elaborados no corrente projeto. Após fornecer os dados da nova *tag*, pressiona-se o botão *Create*, de acordo com a primeira ilustração (fig. 104).

A partir daí a tag T2 já foi criada, necessitando que seja configurado seu tempo para acionar a saída DN, que deve ser digitado no parâmetro PRE com o valor 10000, equivalendo a 10 segundos, pois a unidade de medição é especificada em milissegundos, conforme segunda ilustração (fig. 104).

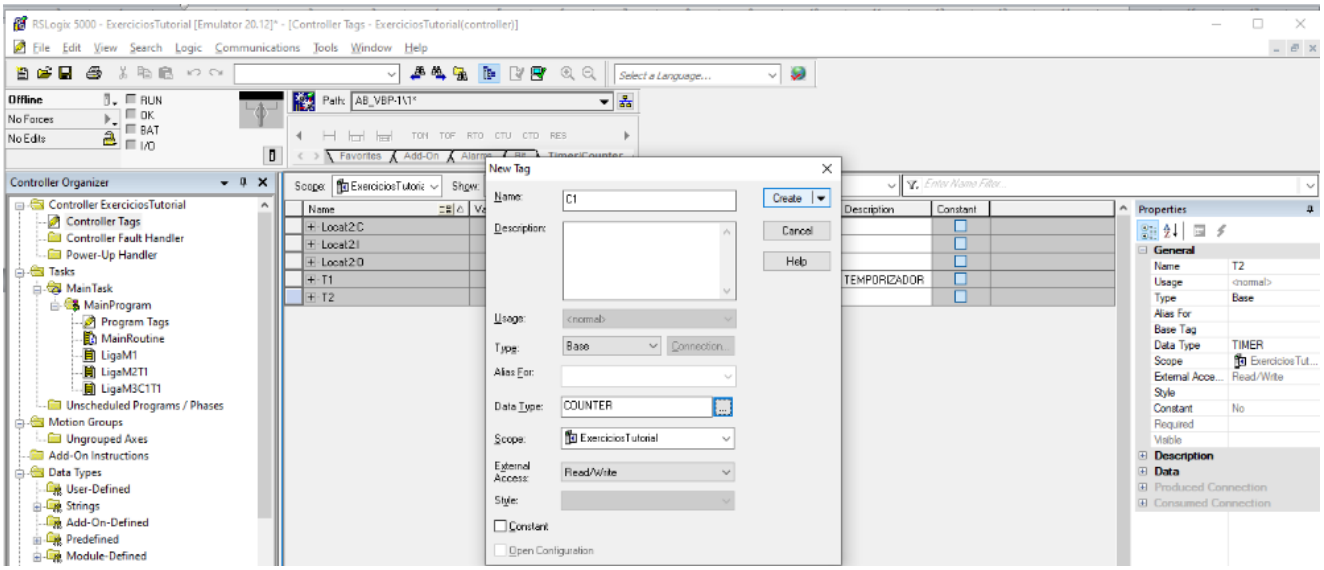
Figura 104 – Criação da tag de temporização



Fonte: Própria.

Chega o momento de criar o contador C1. Para isso, devendo para isso clicar novamente com o botão direito sobre *Controller Tags*, e escolher a opção *New Tag*, fazendo surgir o seguinte quadro, cujos parâmetros devem ser preenchidos (fig. 105).

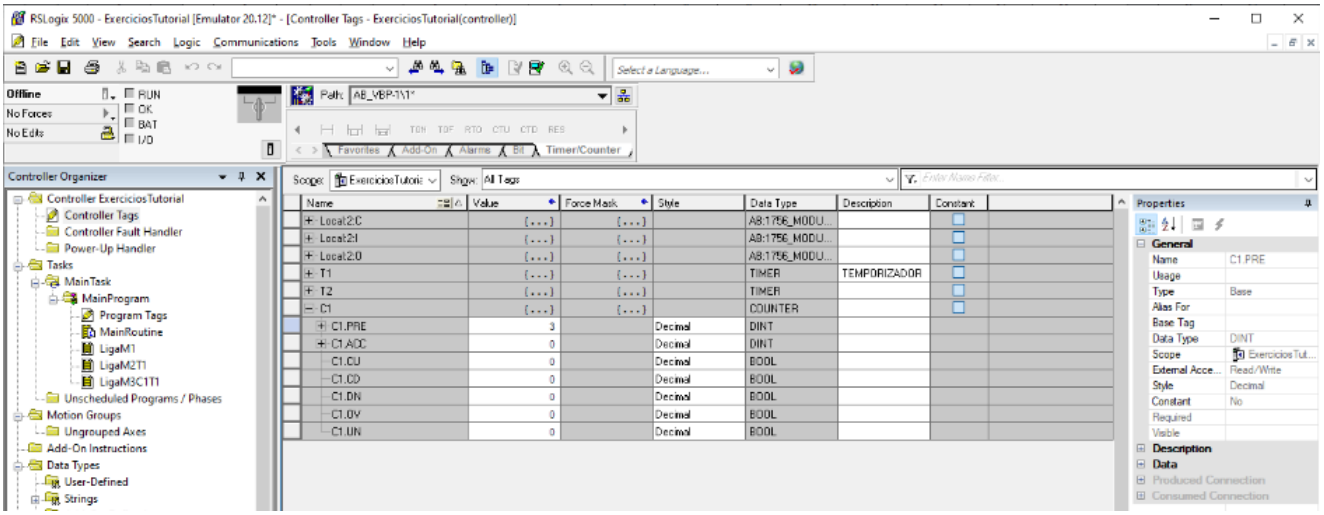
Figura 105 – Criação da *tag* C1



Fonte: Própria.

Após pressionar o botão create, Accessam-se as propriedades da *tag* de contagem criada, para digitar no campo PRE (*preset*) o limite superior de acionamentos. Como os acionamentos devem ser restringidos após duas operações, *preset* receberá o valor 3 (fig. 106)

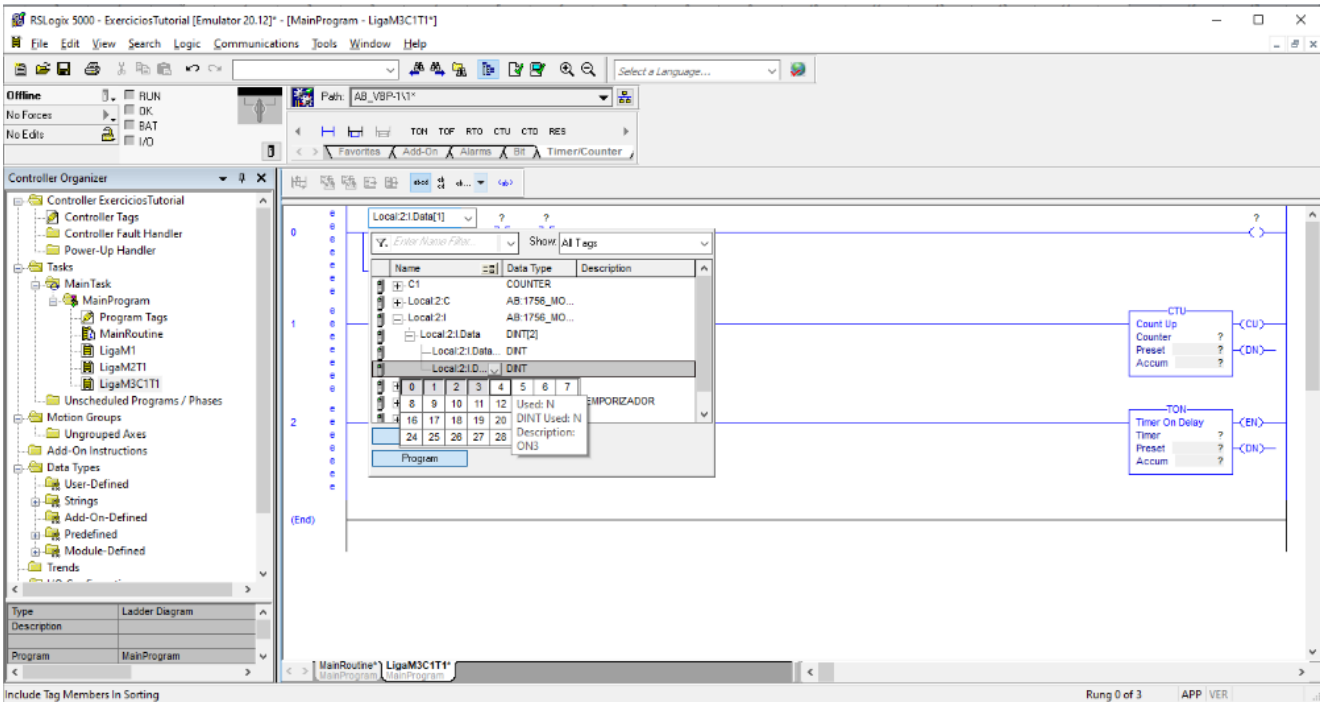
Figura 106 – Configuração do campo *PRESET* da *tag* C1



Fonte: Própria.

A próxima etapa consiste em retornar ao programa ladder, e associar cada uma das tags criadas ao respectivo elemento integrante do programa ladder. Isso é feito clicando-se sobre o botão de interrogação de cada elemento, fazendo surgir um quadro de procura dos endereços das tags, para associar as mesmas às respectivas instruções, de acordo com a figura 107.

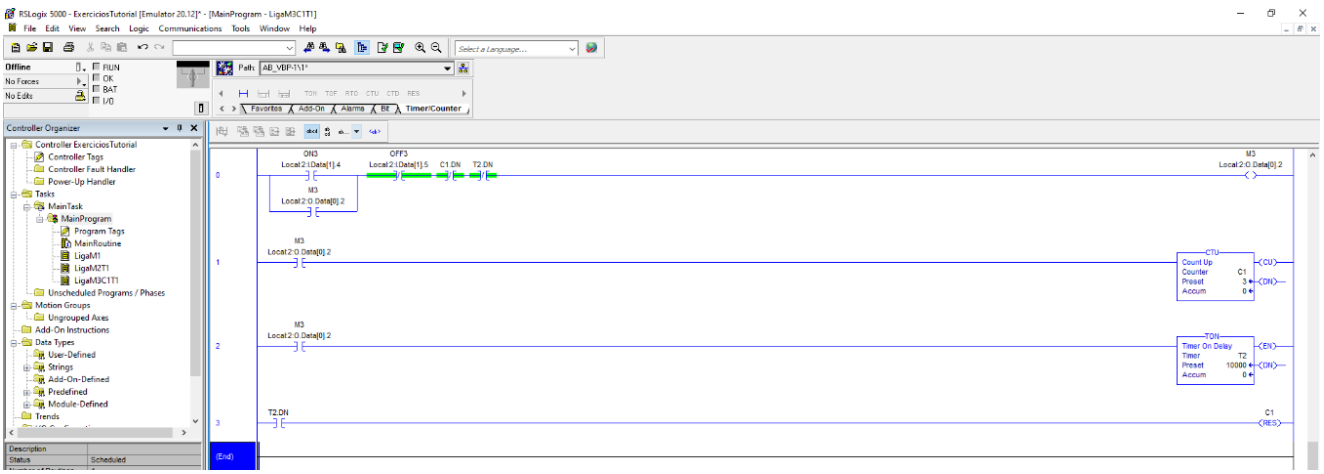
Figura 107 – Associando as tags criadas aos elementos do programa



Fonte: Própria.

O formato final do diagrama ladder adquire a configuração da figura 108, após atribuir o endereçamento das tags às respectivas instruções.

Figura 108 – Programa ladder finalizado



Fonte: Própria.

A última linha do programa é a quarta (identificada pelo número 3, uma vez que a linha inicial recebe o número zero como identificador), cuja função é realizar reset automático da instrução CTU, possibilitando continuidade do processo. Entretanto, a versão do programa apresentada na figura 108 tem uma falha. Quando ocorrer o terceiro acionamento, o contador será automaticamente resetado.

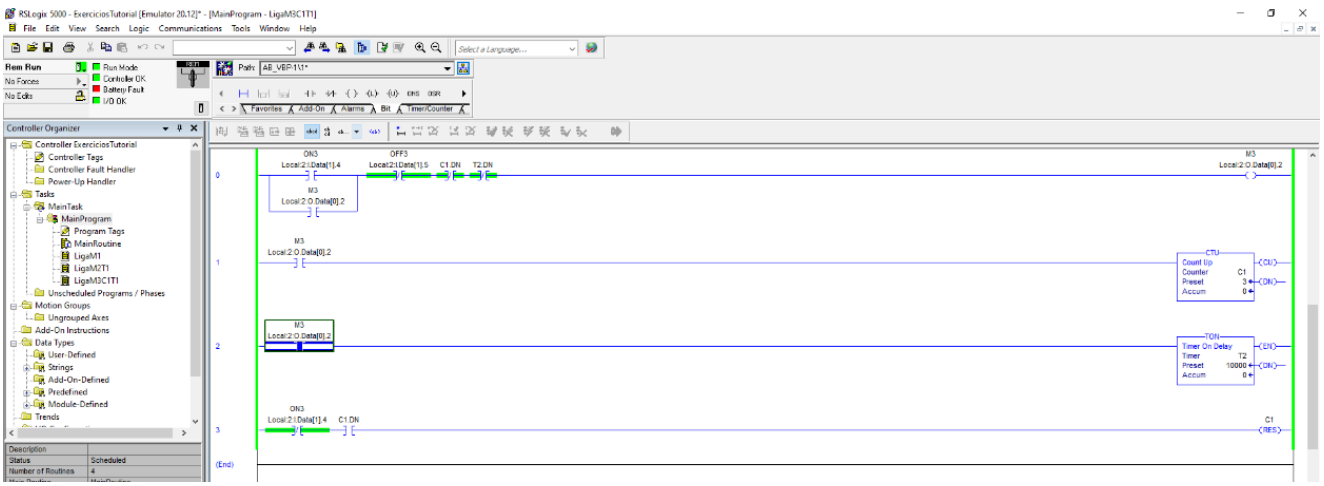
Porém, a varredura (scan) do CLP é relizada muito rapidamente, de forma que o programa ladder ainda percebe o botão ON3 acionado, mesmo após o reset do contador, fazendo com que o último acionamento seja também contabilizado como o primeiro da nova sequência.

Para evitar o supracitado erro, modifica-se o programa conforme figura 109, onde se acrescenta um contato virtual normalmente fechado (NF), associado em série com o contato normalmente aberto (NA) endereçado por T2.DN, na última linha do programa.

Assim, a saída referente à instrução RESET do contador será acionada somente após o botão ON3 ser detectado como solto pelo programa, pois nesta condição seu contato virtual posicionado na quarta linha retornará ao estado de repouso (NF), enquanto o contato virtual T2.DN comutará para fechado, devido ao fato do registo ACC alcançar o valor estabelecido no campo PRE do contador (que é igual a três).

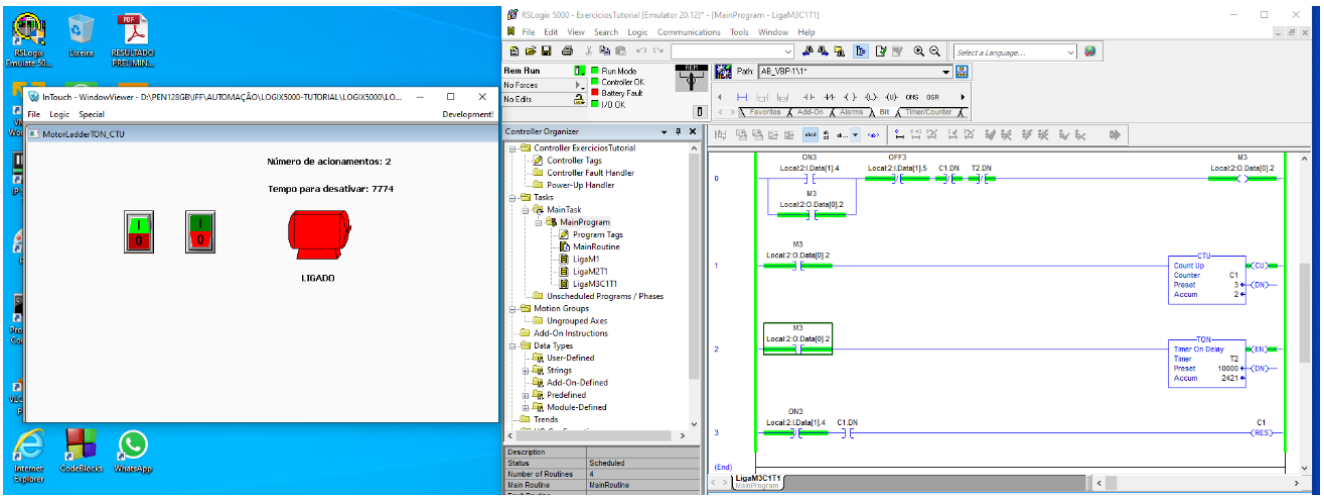
A figura 110 apresenta a tela do sistema supervisório, para controlar o programa ladder elaborado.

Figura 109 – Aperfeiçoamento do progrma ladder



Fonte: Própria.

Figura 110 – Tela do sistema supervisório para controle do programa ladder

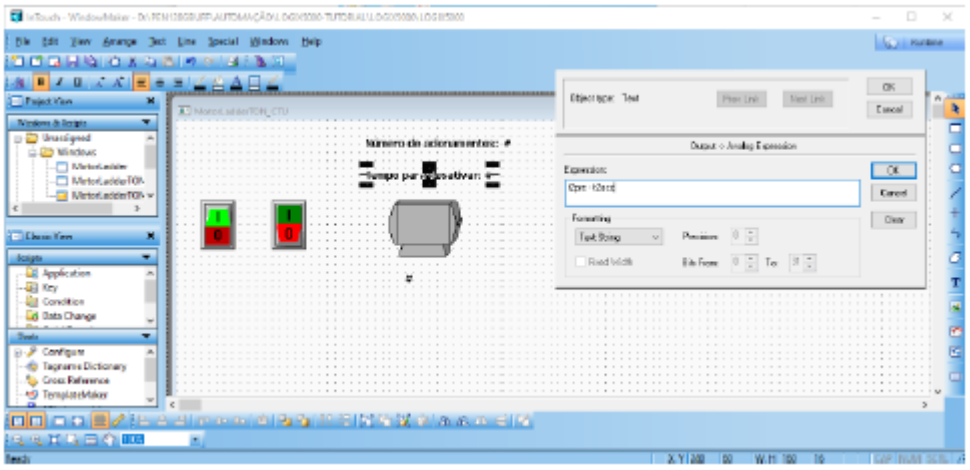


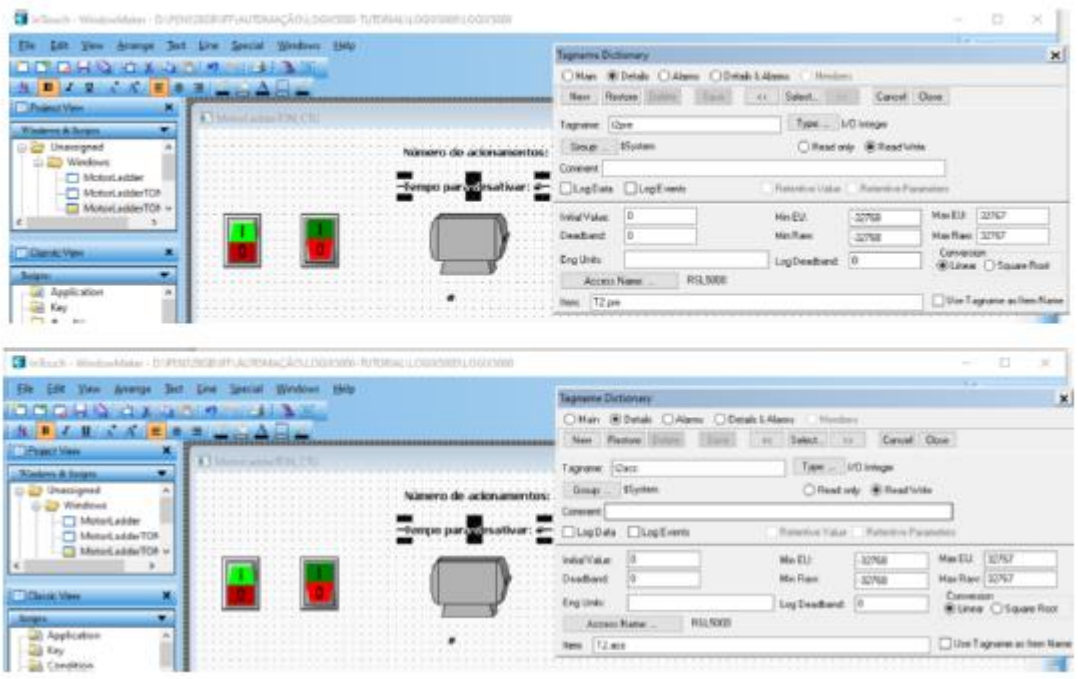
Fonte: Própria.

Convém destacar a primeira ilustração (fig. 111), com as configurações para exibir o tempo de operação restante ao motor, após seu acionamento, obtido pela subtração entre os campos preset e accum do temporizador.

As definições das expressões t2pre e t2acc são apresentadas nas duas ilustrações seguintes da figura 111.

Figura 111 – Configurações dos objetos texto para exibir tempo e acionamentos

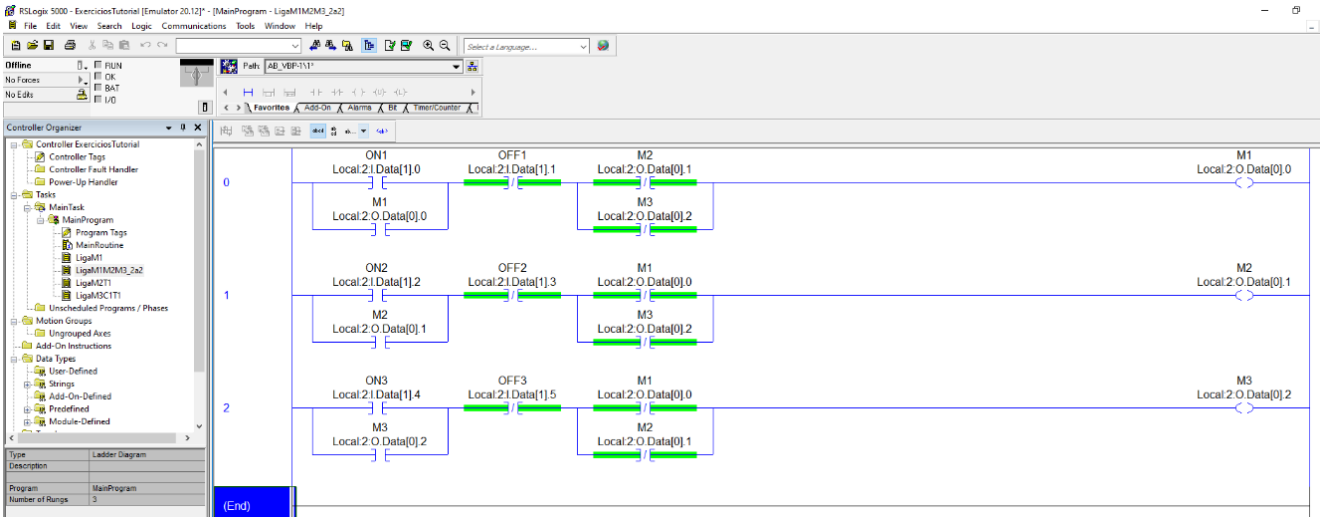




Fonte: Própria.

8) Determinado processo industrial requer o acionamento independente de três motores. Pede-se elaborar código ladder (fig. 112), acompanhado do respectivo sistema supervisório, que permita ligar ou desligar cada uma dessas máquinas separadamente, considerando a restrição de que quando duas delas estiverem em operação, a terceira deve ser automaticamente impedida de entrar em funcionamento.

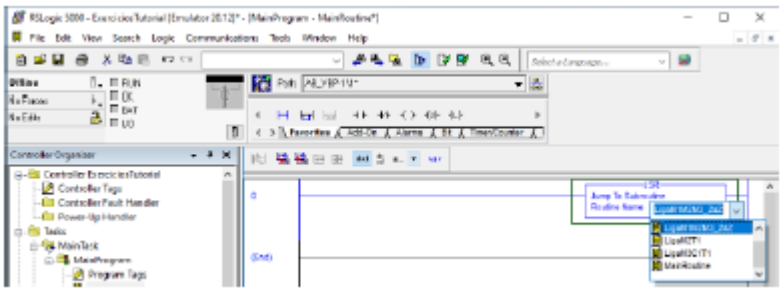
Figura 112 - código ladder para a automação do processo solicitado



Fonte: Própria.

Antes de executar o programa ladder, é preciso acessar a rotina principal, e mudar o argumento da instrução JSR para o nome do programa que se acabou de elaborar, conforme figura 113.

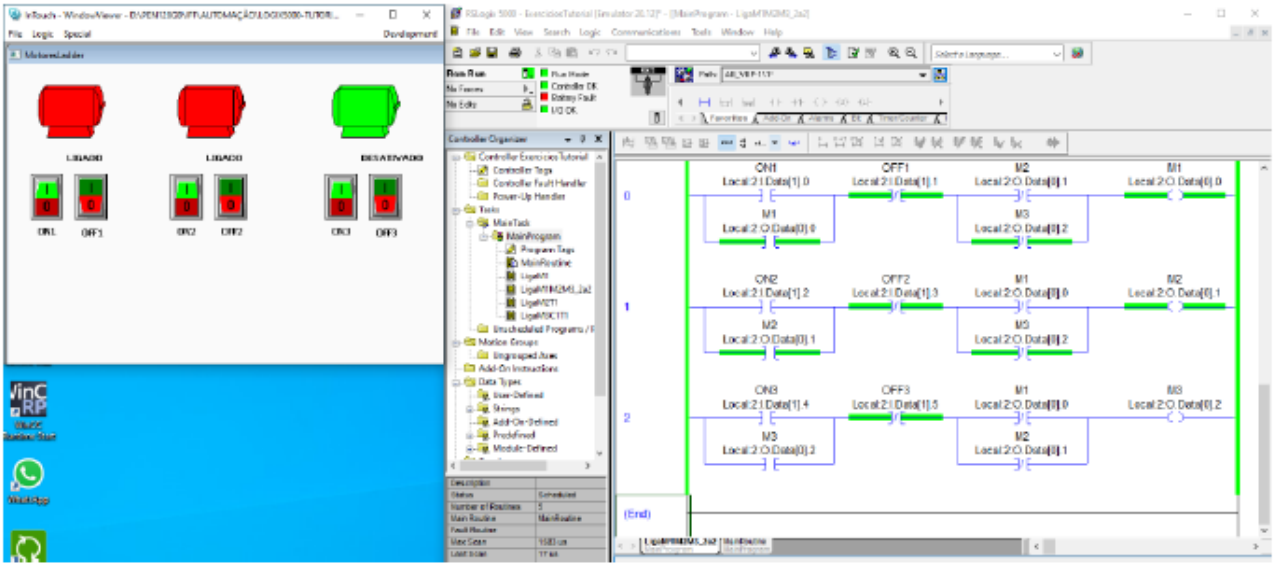
Figura 113 – Configurando a instrução JSR do programa principal



Fonte: Própria.

Seguem a tela do sistema supervisório e o código ladder para a automação do processo solicitado, conforme figura 114. Percebe-se que os motores M1 e M2 estão em operação, com cor de preenchimento vermelha, impedindo eventuais tentativas de acionamento do terceiro motor pela linha 2, mesmo que o botão ON3 seja pressionado. Isso ocorre porque devido ao funcionamento de M1 e M2, os contatos que têm o endereço destes motores (que estão associados em paralelo na terceira linha do código) comutam de sua posição de repouso (normalmente fechado – NF), tornando-se ambos abertos, impedindo circulação de corrente virtual oriunda do acionamento do botão ON3.

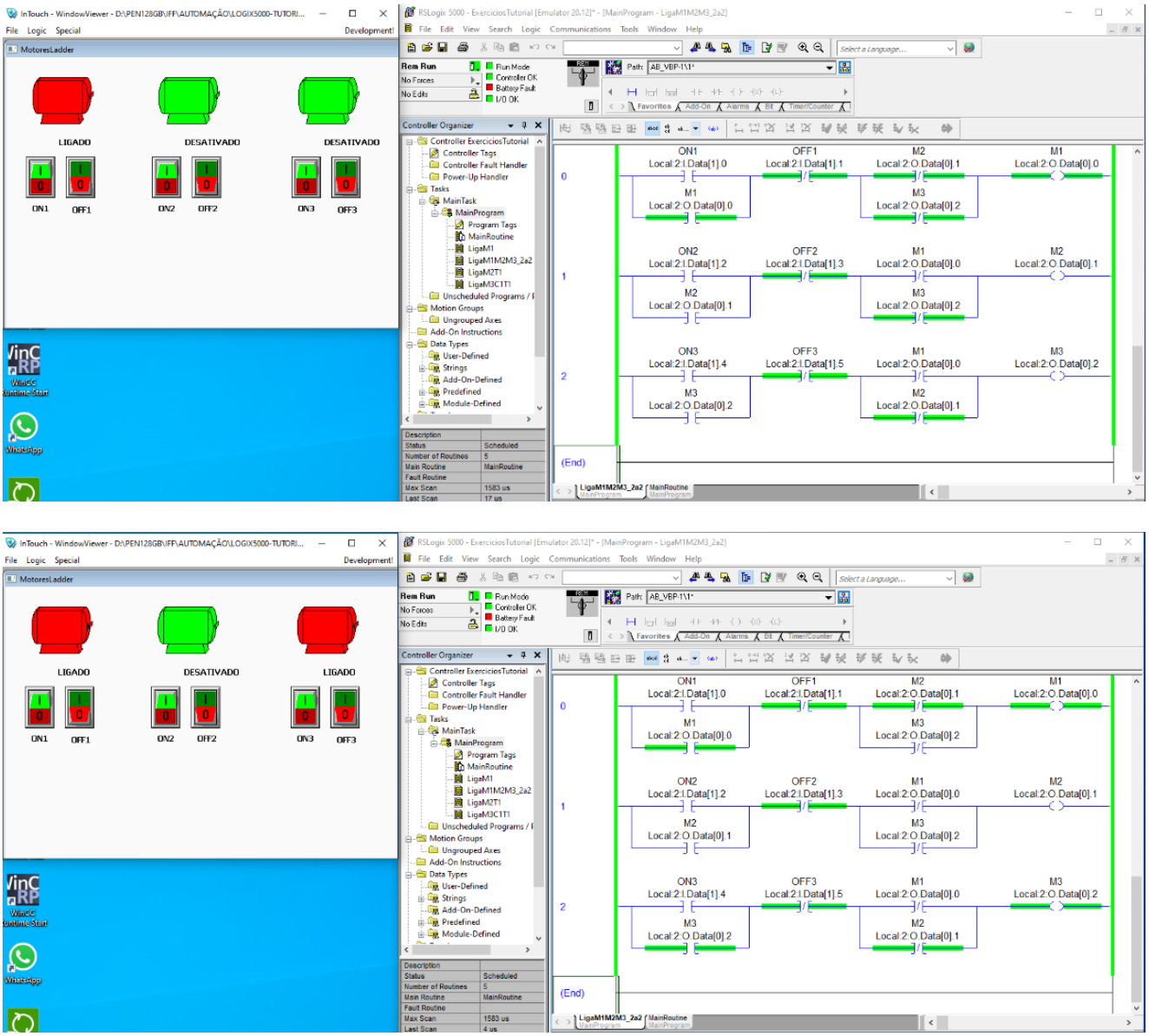
Figura 114 – Tela do sistema supervisório e o código ladder para a automação do processo



Fonte: Própria.

Entretanto, quando ocorrer o desligamento de pelo menos um dos motores (M1 ou M2), a linha 2 proporcionará condições à circulação de corrente virtual até a saída M3, caso ON3 seja acionado, de acordo com a primeira ilustração da figura 111. Quando o botão ON3 for pressionado, o respectivo contato NA da linha 2 comutará da posição de repouso, fechando momentaneamente, o que possibilitará circulação de corrente virtual até a saída M3, acionando este motor, conforme segunda ilustração da figura 115.

Figura 115 – Operação do programa ladder através do sistema supervisório

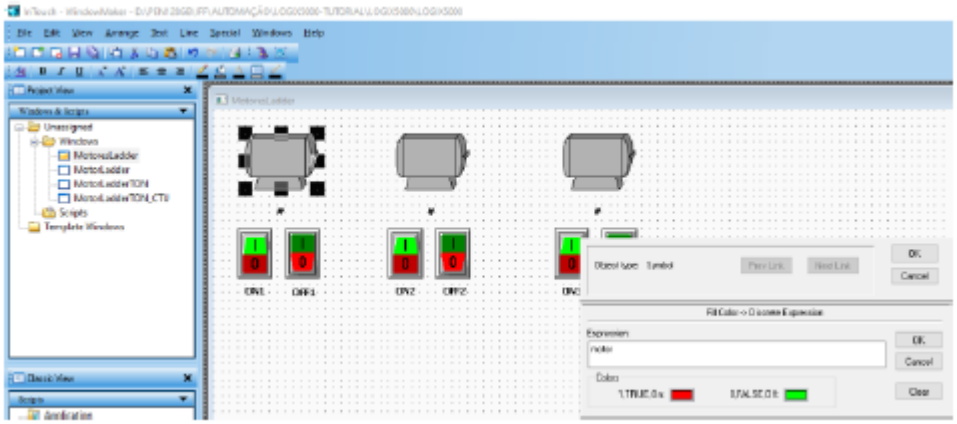


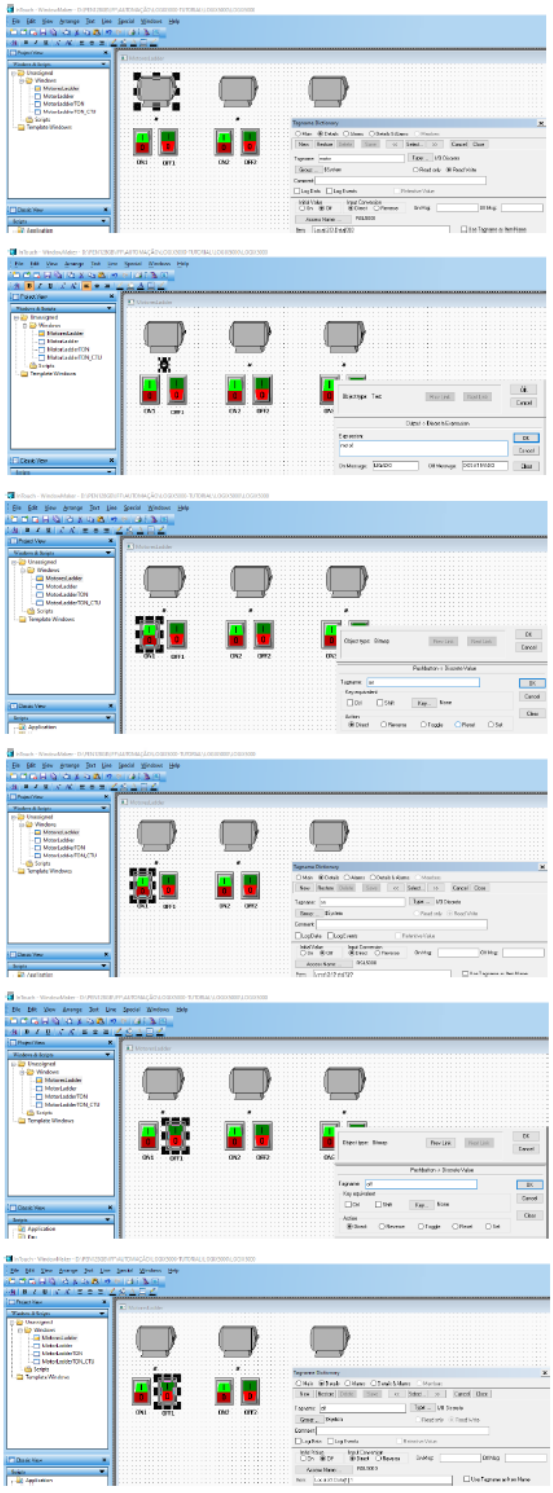
Fonte: Própria.

Com M1 e M3 em operação, seus contatos normalmente fechados (NF) que estão associados em paralelo na linha 1 comutam para o estado aberto, impedindo o acionamento de M2, mesmo que o botão ON2 seja pressionado.

A figura 116 apresenta as ilustrações das configurações de componentes do supervisório relacionados a M1.

Figura 116 – Configurações de componentes do supervisório relacionados a M1.



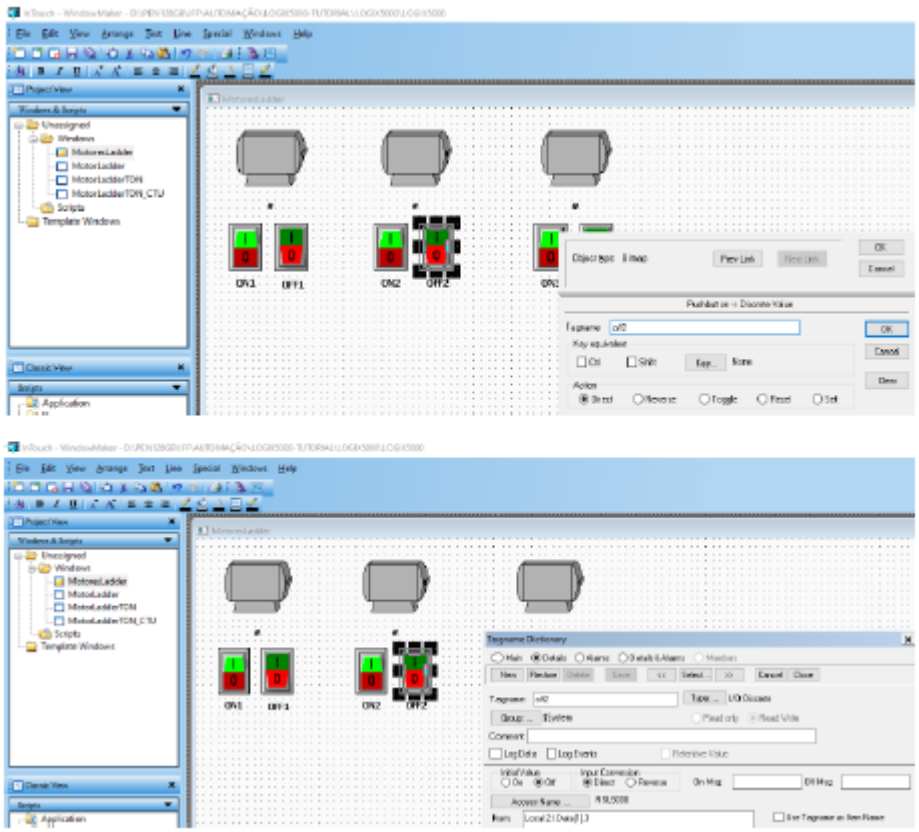


Fonte: Própria.

A figura 117 apresenta as ilustrações das configurações dos componentes do supervisório relacionados a M2.

Figura 117 – Configurações de componentes do supervisório relacionados a M2.

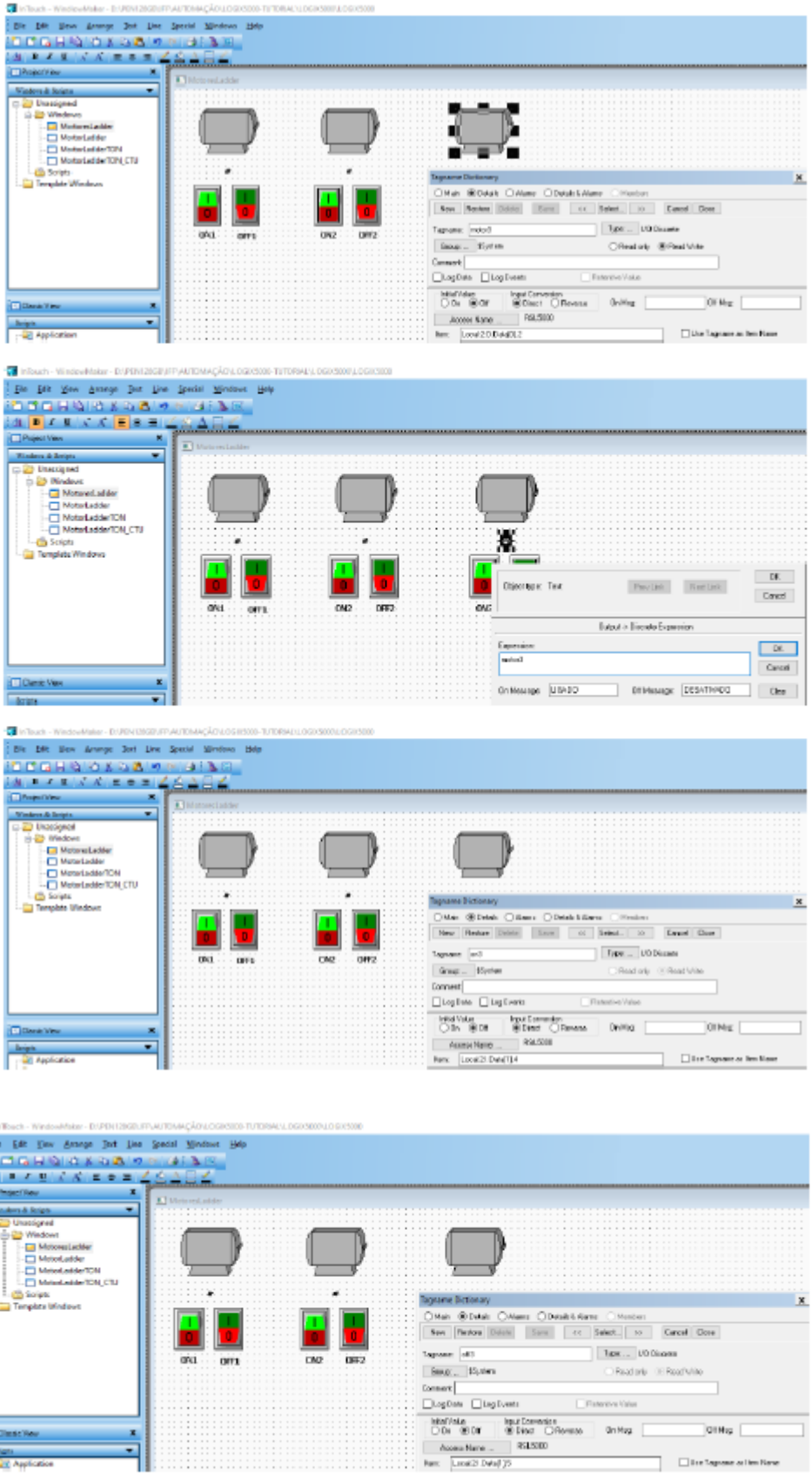




Fonte: Própria.

A figura 118 apresenta as ilustrações das configurações dos componentes do supervisório relacionados a M3.

Figura 118 – Configurações de componentes do supervisório relacionados a M3.

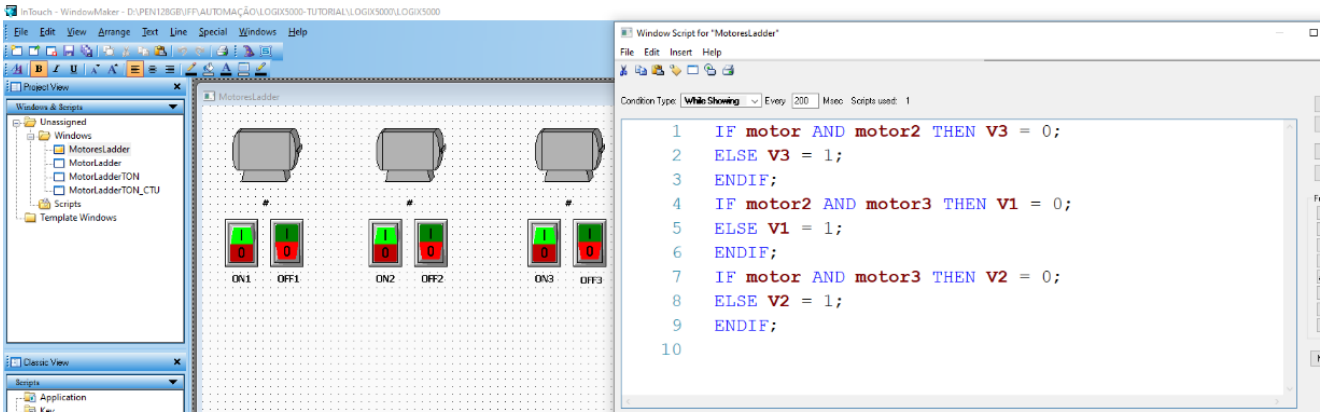


Fonte: Própria.

9) Solicita-se desenvolver script para o sistema supervisório anterior (fig. 119), que apague os botões para ligar e desligar um dos motores, sempre que as demais máquinas estiverem em operação. Ressalta-se a necessidade do código restaurar a visibilidade dos botões, toda vez que um dos dois motores em funcionamento simultâneo for desativado.

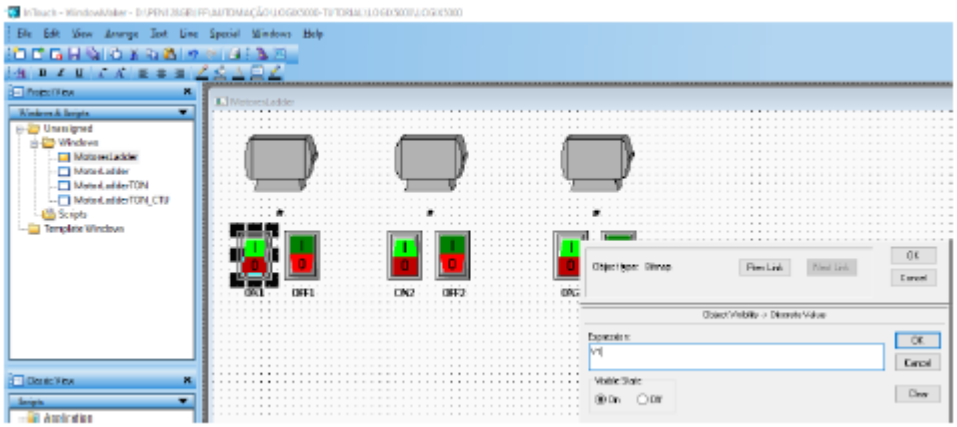
Segue o script elaborado, acompanhado das configurações adicionais para os objetos, conforme figuras 119, 120, 121, 122 e 123.

Figura 119 – Script do sistema supervisório relacionado ao problema 8



Fonte: Própria.

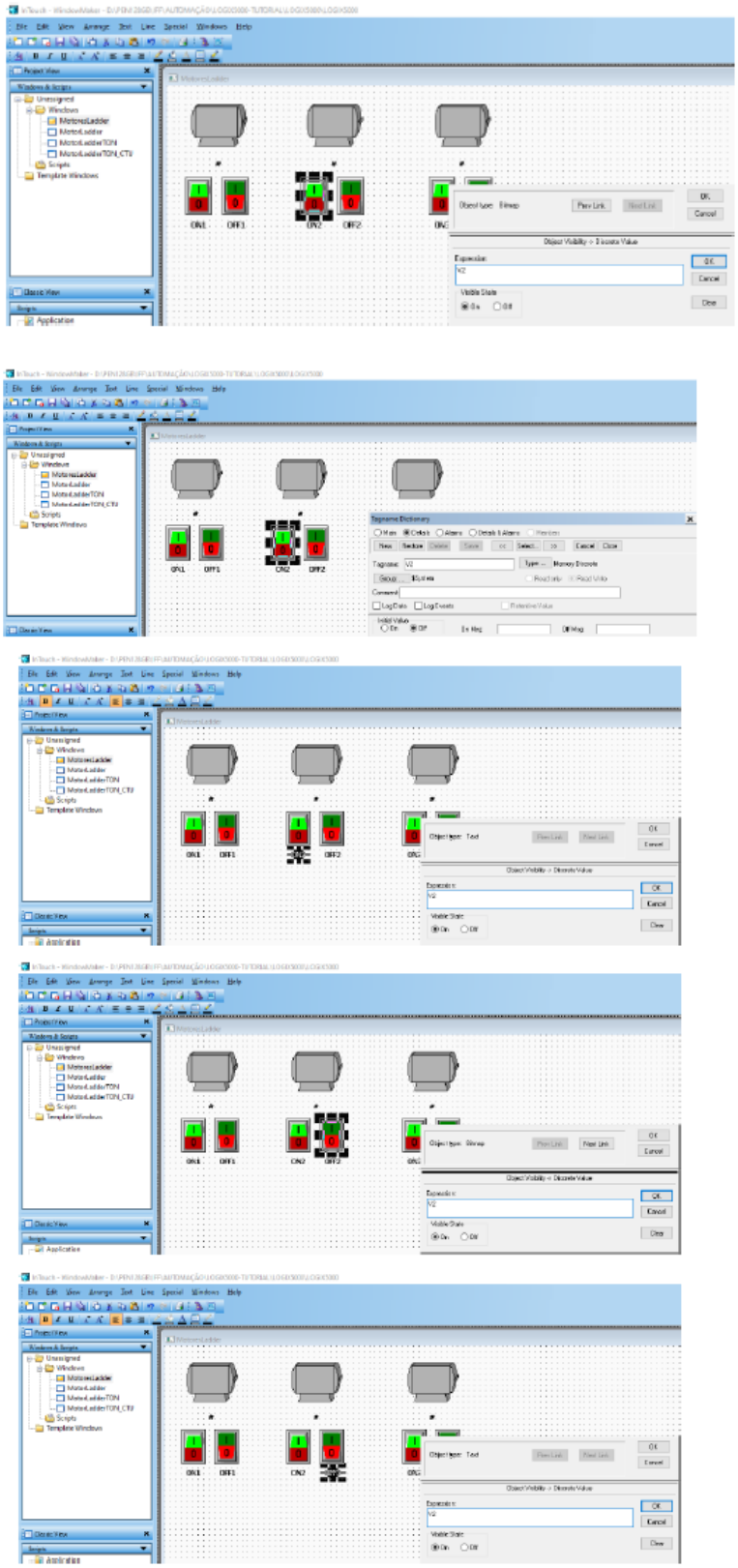
Figura 120 – Configurações de visibilidade para botões e rótulos do primeiro motor.





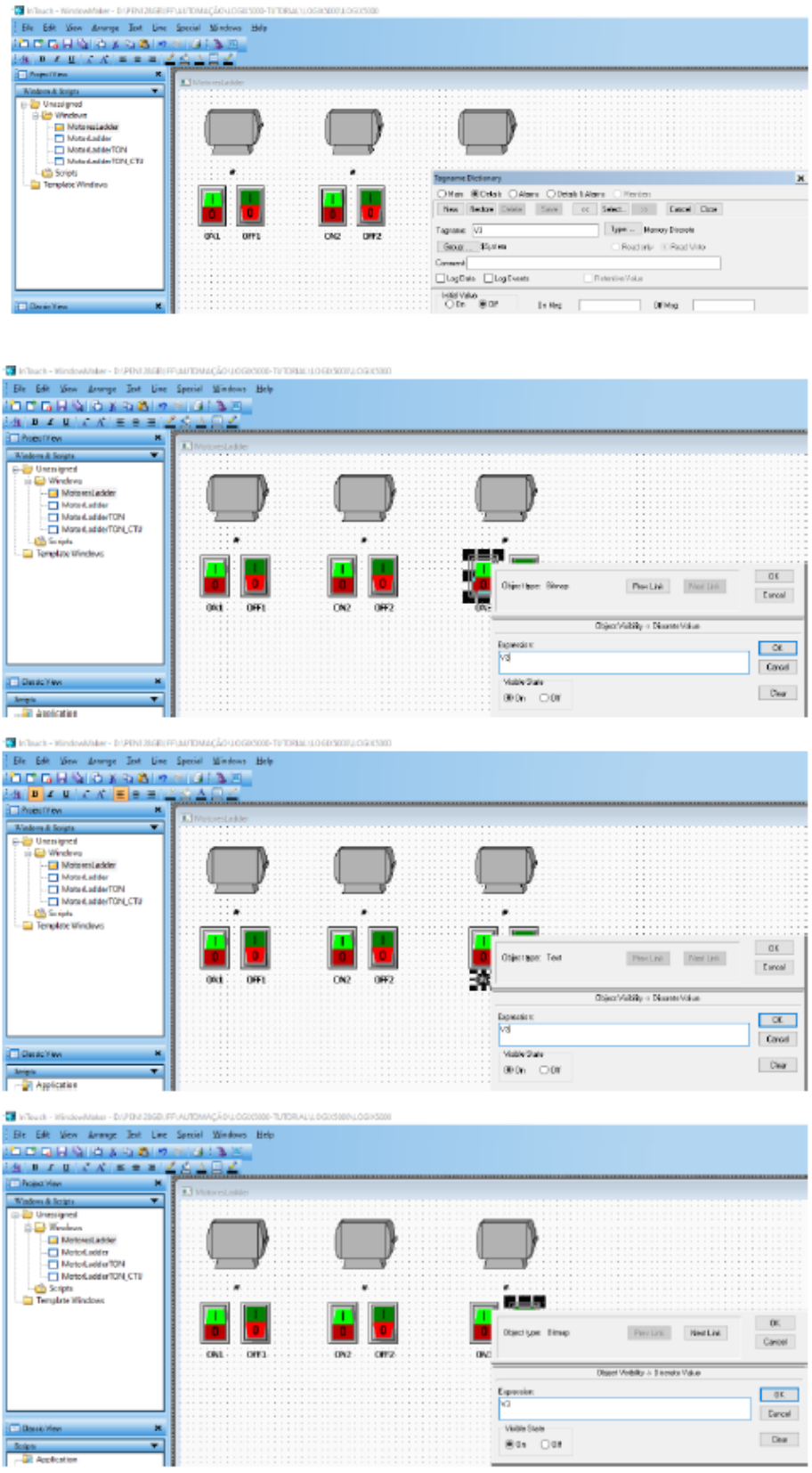
Fonte: Própria.

Figura 121 – Configurações de visibilidade para botões e rótulos do segundo motor.



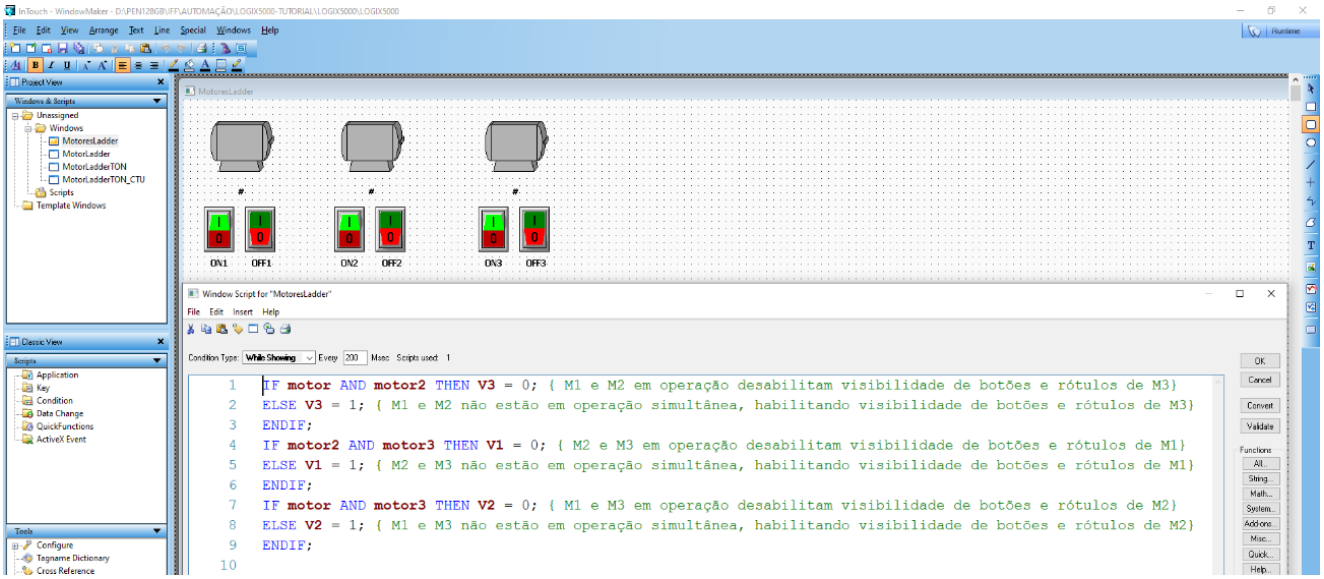
Fonte: Própria.

Figura 122 – Configurações de visibilidade para botões e rótulos do terceiro motor.



Fonte: Própria.

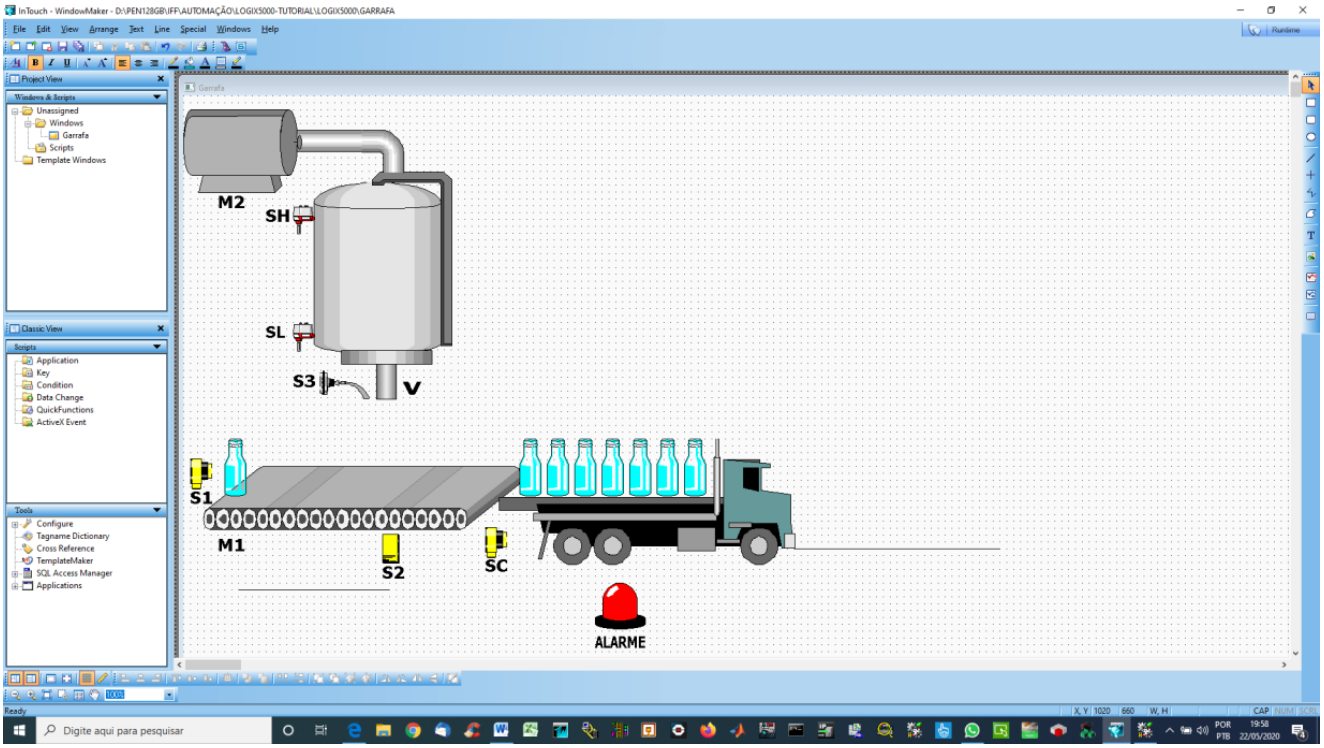
Figura 123 – Explicação do script do sistema supervisório relacionado ao problema 8



Fonte: Própria.

10) Seja um processo industrial para envasamento de garrafas, representado por seu sistema supervisório na figura 124, onde se observam os seguintes componentes utilizados: S1 (sensor de presença de garrafas sobre a esteira); M1 (motor da esteira que transporta as garrafas); V (válvula que possibilitará fluxo de líquido para a garrafa); S2 (sensor que detecta quando a garrafa estiver alinhada exatamente abaixo da válvula V); S3 (sensor que detecta quando nível de líquido da garrafa estiver alto); SC (sensor de presença do caminhão); SL (sensor de nível baixo do tanque); SH (sensor de nível alto do tanque); ALARME (luz que avisa quando o caminhão estiver cheio de garrafas) e M2 (motor que injeta líquido no interior do tanque).

Figura 124 – Sistema supervisório para envasamento de garrafas



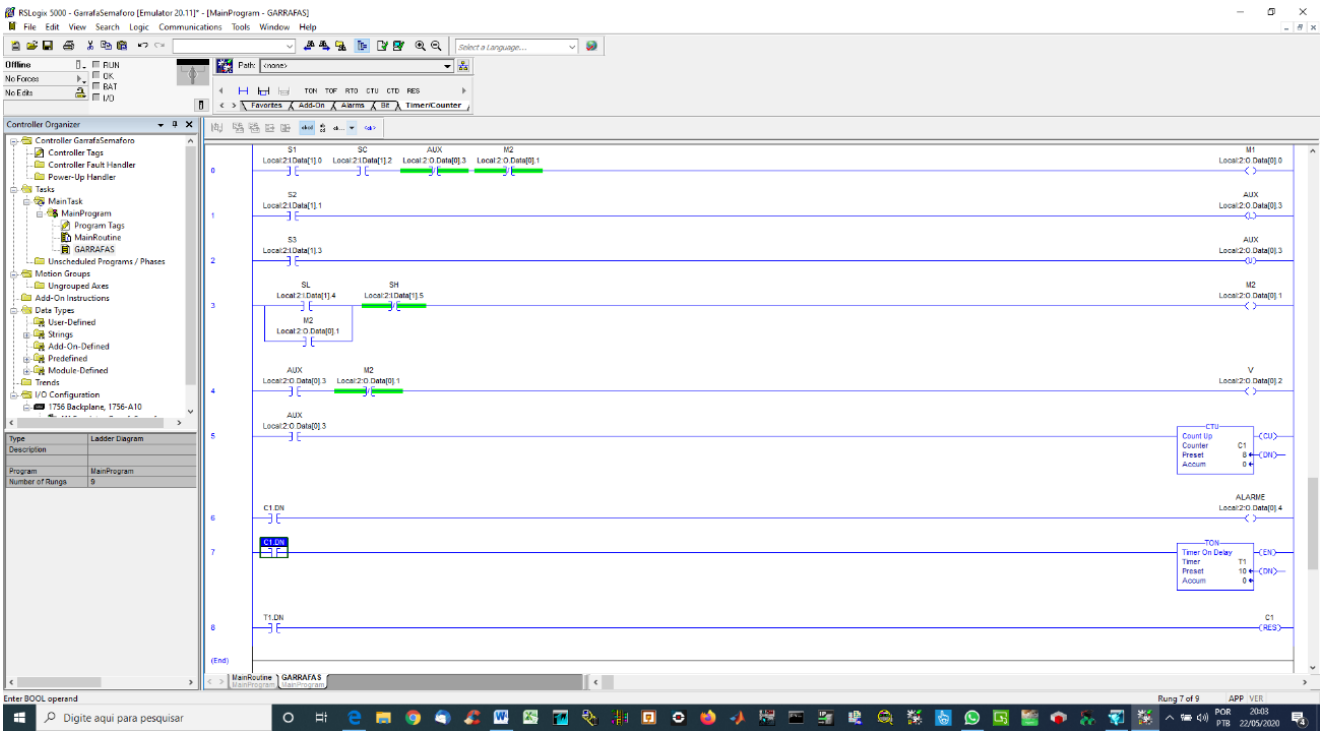
Fonte: Própria.

A dinâmica do processo consiste em movimentar a esteira através do acionamento de M1, sempre que S1 detectar garrafas presentes sobre a mesma, e enquanto houver caminhão, cuja presença é percebida pelo sensor SC. Assim que S2 detectar que a garrafa está exatamente abaixo da válvula V, a esteira deve ser desligada desativando-se M1, enquanto a válvula V é aberta, para preenchimento da garrafa com líquido. Quando S3 detectar que a garrafa está cheia, a válvula V será fechada, enquanto o motor M1 se movimentará novamente, deslocando o recipiente até o interior do caminhão.

Cada garrafa nova será inserida na extremidade do caminhão quando S1 detectar a presença da mesma, o que é simulado clicando-se sobre o contato virtual associado ao sensor S1, fazendo com que o processo se repita novamente. Quando o caminhão estiver com sete garrafas, o alarme será acionado por 10 segundos (com luz piscando), e o caminhão iniciará movimentação à direita por 300 pixels, quando então os recipientes serão descarregados, e o veículo retornará ao ponto de partido executando marcha a ré. A presença do caminhão é simulada clicando-se sobre o contato virtual endereçado pelo sensor SC. Assim que retornar à base, o caminhão desaparecerá. Tão logo o sistema detecte nível baixo de líquido no tanque, a esteira M1 parará, a válvula V1 fechará, e o motor M2 entrará em operação, permanecendo assim até que SH detecte tanque cheio, quando então M2 será desativado, e o processo da esteira retornará a sua operação normal. Pede-se elaborar programa

ladder que realize a automação deste processo. Segue o programa ladder que realiza a automação do processo descrito (fig. 125).

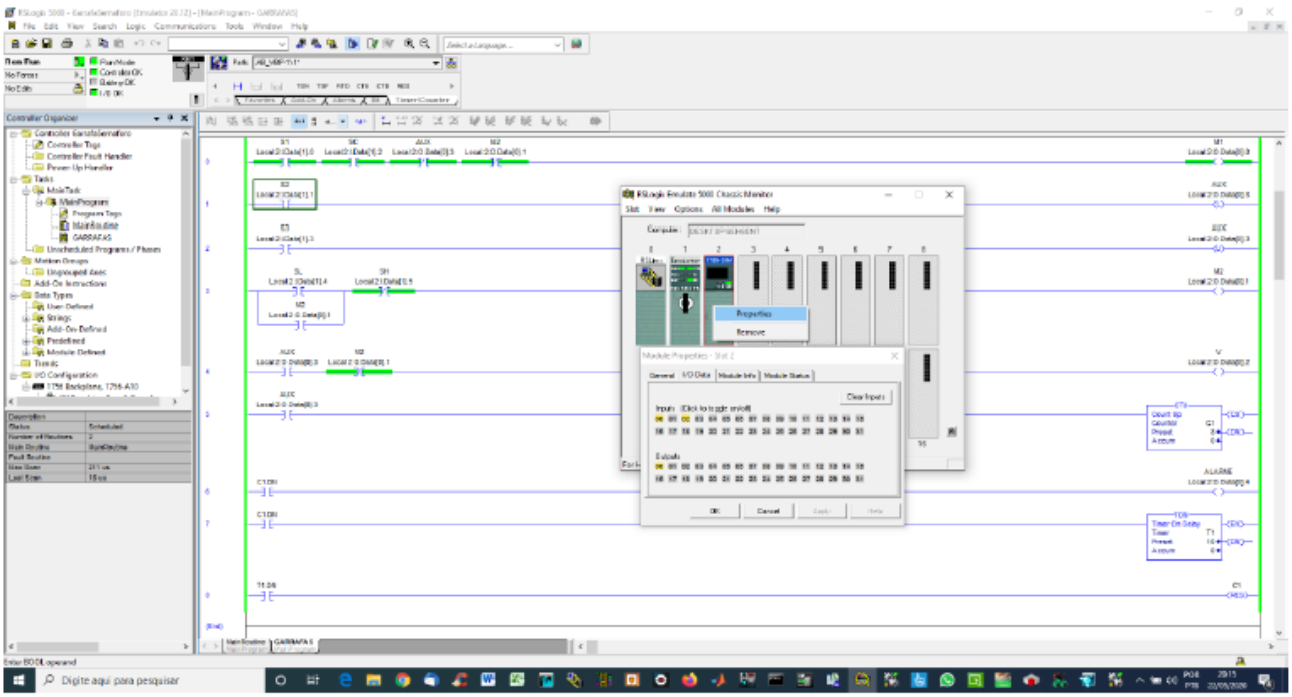
Figura 125 – Programa ladder para automação do envasamento de garrafas



Fonte: Própria.

Para simular a presença de garrafas e de caminhão e acionar M1 (Local:2:O.Data[0].0), clica-se sobre as entradas dos respectivos sensores (S1 → Local:2:I.Data[1].0 e S2 → Local:2:I.Data[1].2) através das propriedades do segundo módulo (Slot 2), acessadas pelo RSLogix Emulate 5000 (fig. 126).

Figura 126 – Acionamento do programa ladder através do RSLogix emulate 5000



Fonte: Própria.

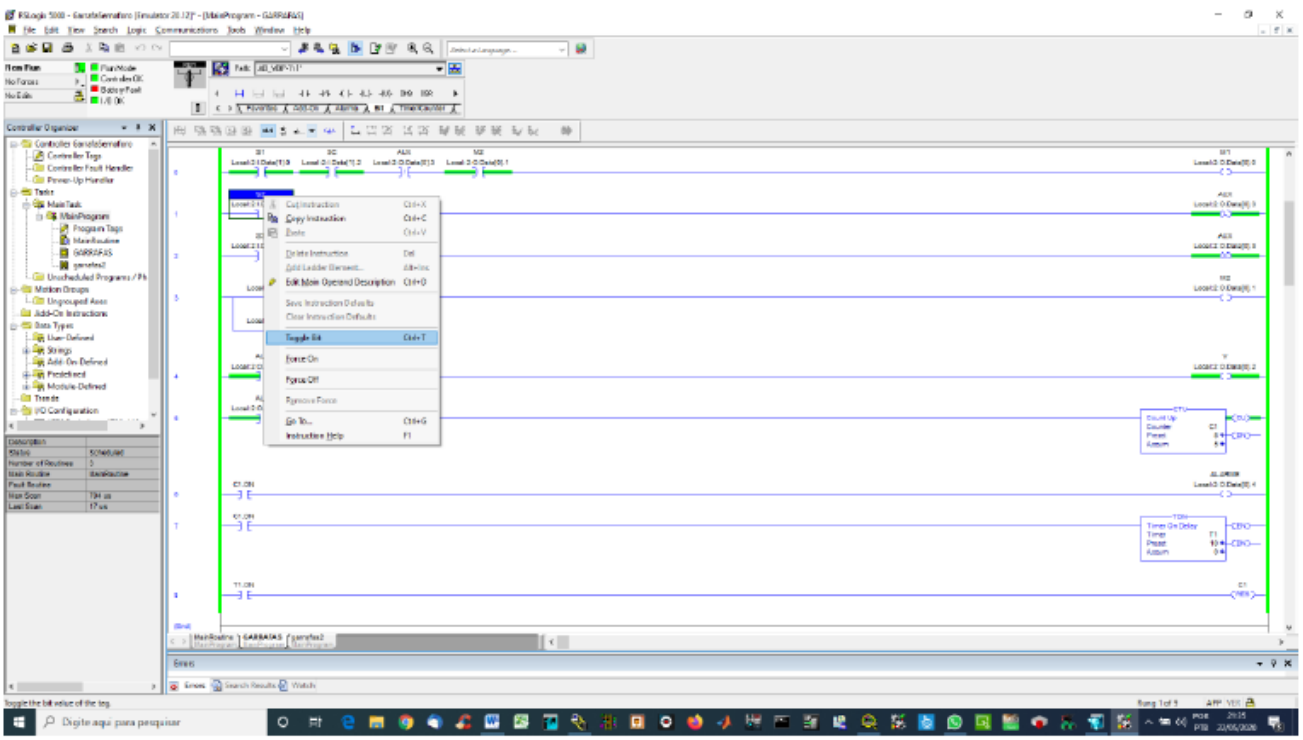
As luzes amarelas verificadas nas entradas 00 e 02, obtidas após clicar em seus respectivos botões, causam a comutação dos contatos S1 e SC no programa ladder, fechando os mesmos, e proporcionando condições de circulação de corrente virtual até a saída M1, provocando o acionamento da esteira, com iluminação da saída 00 constatada nas propriedades do módulo 2 do RSLogix Emulate 5000.

Quando S2 detectar o alinhamento da garrafa abaixo da válvula, situação que é simulada clicando-se no programa ladder com o botão direito sobre o contato do sensor S2, cujo endereço é Local:2:I.Data[1].2, e marcando a opção Toggle bit no menu que surgirá, para disparar um pulso único de subida sobre a saída AUX (Local:2:O.Data[0].3), acionando a mesma. Há um contato virtual com o endereço da saída AUX na primeira linha do programa, cujo estado de repouso é normalmente fechado. Dessa forma, o acionamento da saída AUX por S2 na linha 1 também interromperá a circulação de corrente virtual na linha 0, devido a comutação de seu contato para aberto, acarretando a desativação da saída que controla o motor M1 (Local:2:O.Data[0].0).

Simultaneamente às ações descritas no parágrafo anterior, o acionamento da saída AUX provoca a comutação do contato virtual equivalente, situado no início da linha 4, o que ativará a saída que controla a válvula V

(Local:2:O.Data[0].2), abrindo a mesma para iniciar o preenchimento de líquido na garrafa, conforme visto na figura 127.

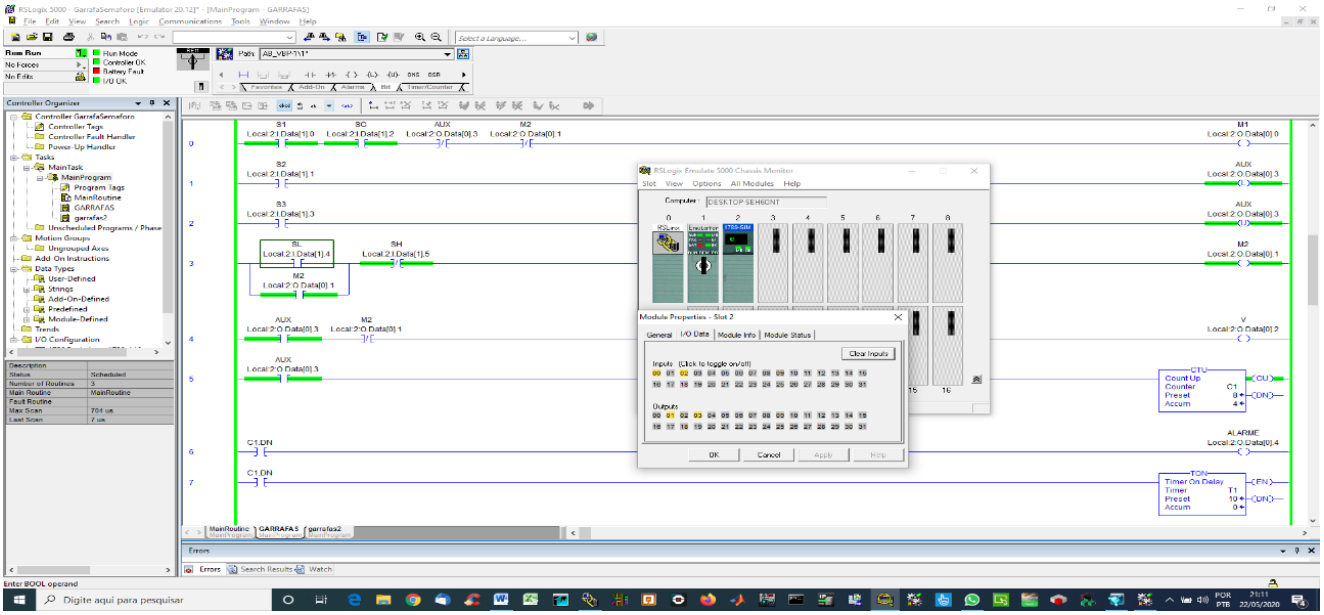
Figura 127 – Controle do programa ladder através de seus próprios contatos virtuais



Fonte: Própria.

Enquanto a válvula estiver aberta, pode ocorrer a situação de nível baixo no tanque, evento detectado pelo sensor SL, na linha 3 do programa, o que proporcionará acionamento do motor M2 devido à comutação de SL para fechado, conforme figura 128.

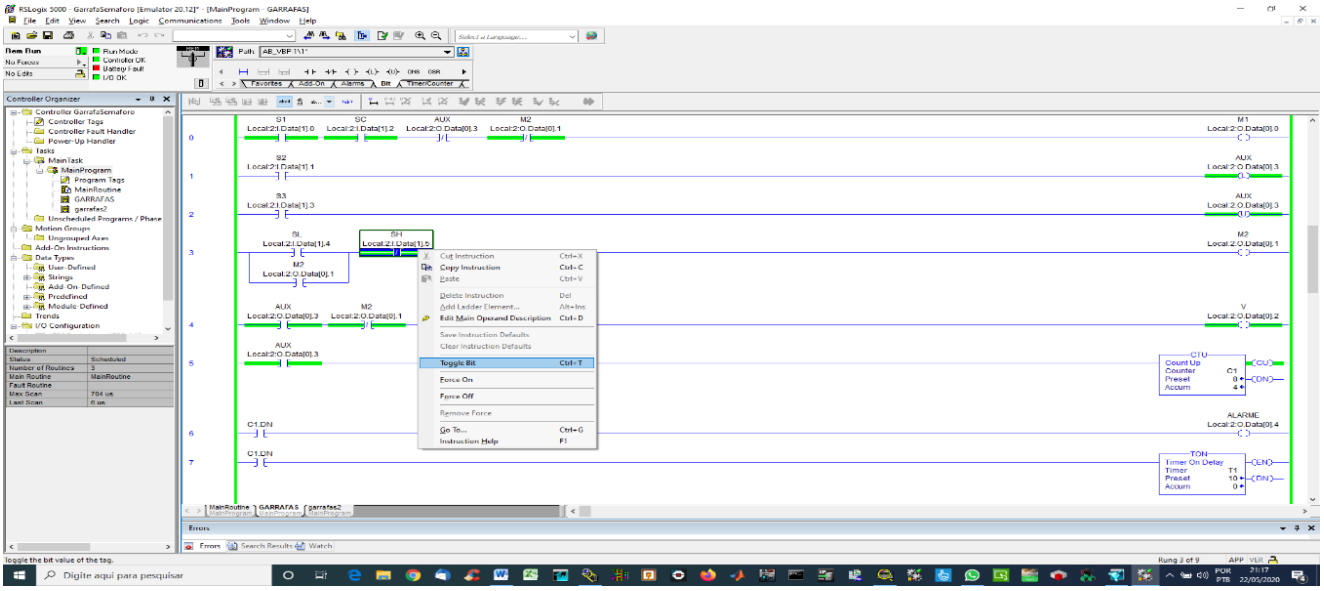
Figura 128 – Acionamento do motor M2 por detecção de baixo nível no tanque



Fonte: Própria.

O acionamento da saída M2 comutará para aberto contato normalmente fechado (NF) com o mesmo que tem o mesmo endereço de M2 na linha 4, interrompendo a circulação virtual de corrente virtual até a válvula V, o que resultará no fechamento da mesma. Simula-se a detecção do nível máximo do tanque clicando-se com o botão direito do mouse sobre SH, na linha 3, e marcando-se a opção *Toggle Bit* mo menu que surgir, inserindo no endereço do contato um pulso único de subida, que comutará momentaneamente seu estado para aberto, ação suficiente para interromper o fluxo de corrente virtual até a saída M2, desativando a bomba, e fazendo com que todos os contatos que tenham seu endereço de saída retornem aos respectivos estados de repouso NF. Isso proporcionará condições à reabertura da válvula, dando continuidade ao preenchimento de líquido na garrafa, conforme figura 129.

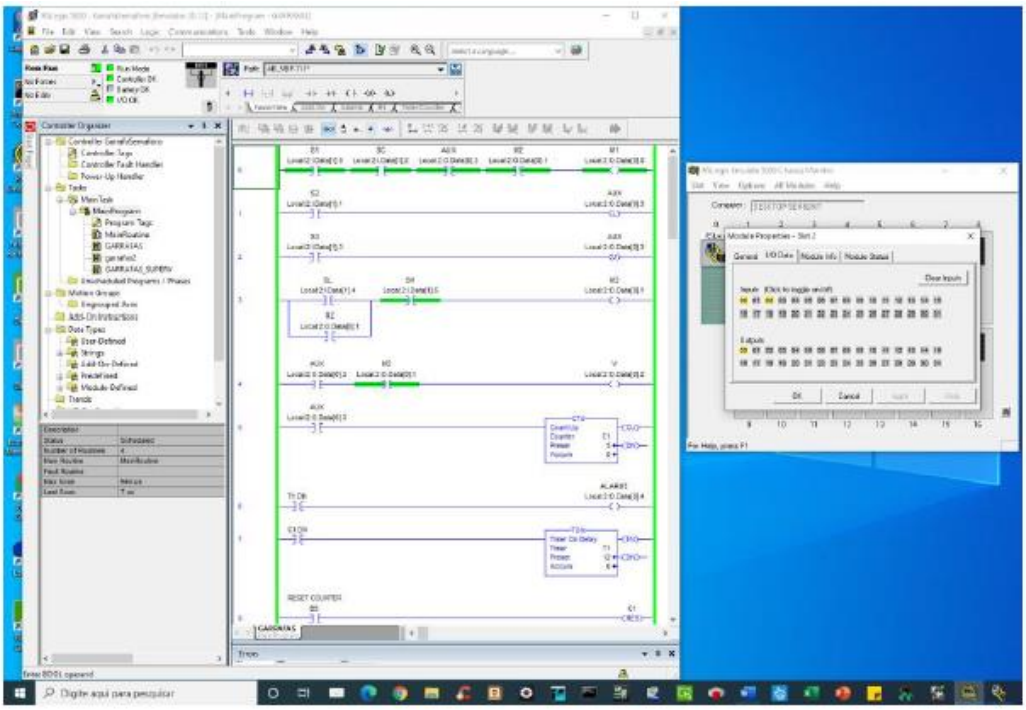
Figura 129 – Desativação de M2 pela detecção de nível alto do tanque



Fonte: Própria.

O final do preenchimento da garrafa ocorre com a detecção do seu nível máximo através do sensor S3, situação simulada clicando-se sobre o contato virtual que contém o endereço do mesmo, junto a linha 2, evento que desativará a saída AUX através da instrução *unlatch* a ela associada. Assim, o contato NF com o mesmo endereço de AUX que está na linha 0 voltará ao seu estado de repouso, fazendo com que a operação da esteira seja retomada, devido à reativação de M1, o que é percebido na figura 130.

Figura 130 – Retomando a operação de M1, motor da esteira transportadora



Fonte: Própria.

Percebe-se uma instrução de contagem na linha 5, que contabiliza uma garrafa cheia sempre que a saída AUX for acionada, o que ocorre quando S2 detecta o alinhamento da garrafa sobre a válvula.

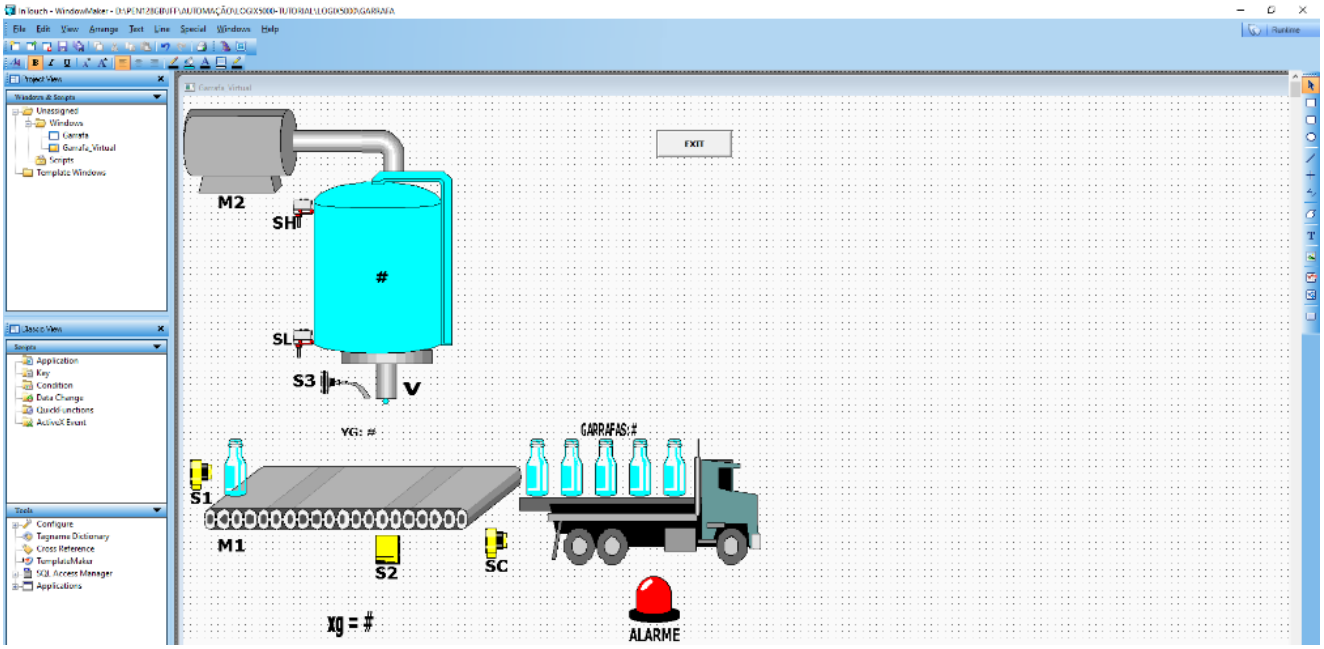
Após 8 garrafas preenchidas, a saída DN do contador C1 será ativada, uma vez que o registro de recipientes acumulados, contabilizado no campo ACC, igual o conteúdo especificado em PRESET. Nesse instante o alarme será acionado. Simultaneamente à ativação da saída DN de C1 ocorrerá temporização de 10 segundos, devido a comutação para o estado fechado do contato normalmente aberto (NA) com este endereço, situado no começo da linha 7.

Isso proporcionará transição de borda de subida junto à entrada do temporizador T1, iniciando registro de tempo decorrido, que cessará quando o conteúdo de ACC igualar o número 10, estabelecido no campo PRE de T1, ativando a saída DN do temporizador. Dessa forma, o contato endereçado por T1.DN situado no começo da linha 8 comutará para fechado, ativando instrução reset associada ao contador C1, zerando o campo ACC do mesmo, e desativando sua saída DN. Assim, o contato virtual endereçado por C1.DN e situado junto à entrada de T1 voltará ao estado NA, desativando o reset do contador, o que possibilitará a repetição do ciclo descrito no enunciado.

Analogamente, a desativação da saída DN de C1 fará com que o contato C1.DN da linha 6 retorne ao estado NA, interompendo circulação de corrente virtual até a saída do alarme, desativando o mesmo.

11) Solicita-se o desenvolvimento de sistema supervisório e script, que realize a animação do processo descrito no problema 10, conforme figura 131.

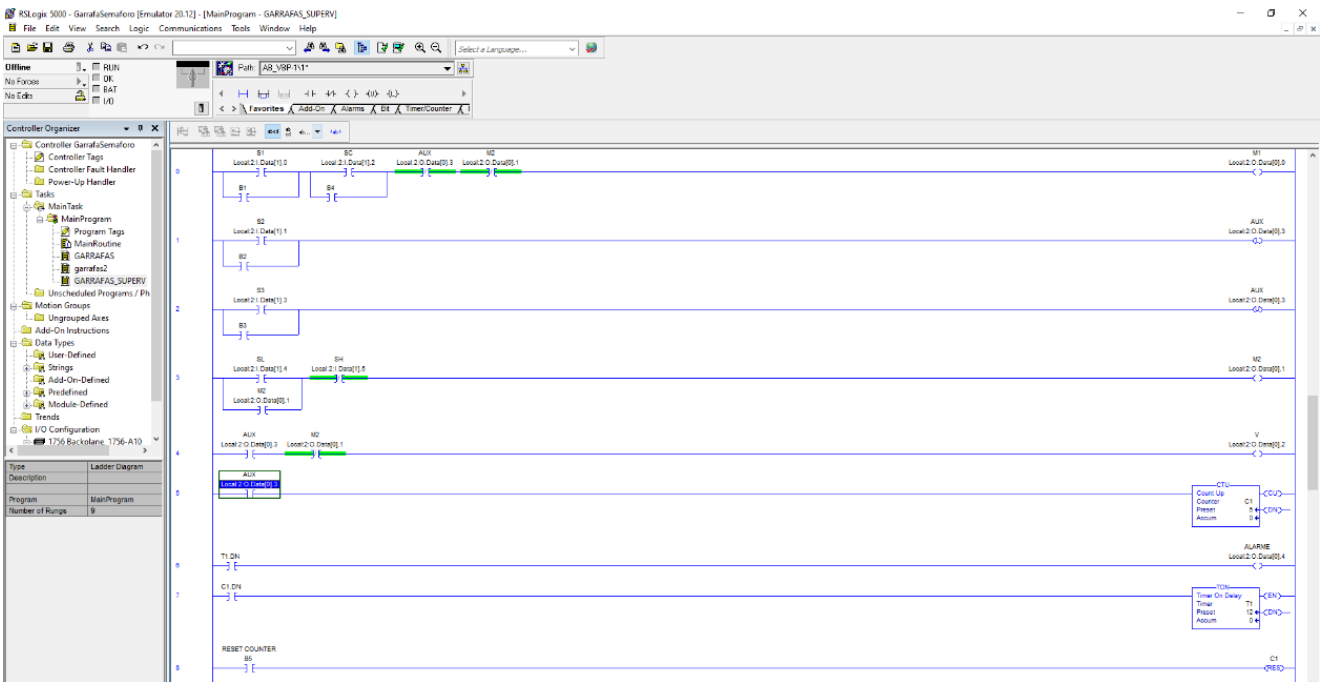
Figura 131 – Supervisório do envasamento de garrafas



Fonte: Própria.

Nesta tela observam-se 5 garrafas como o limite a ser transportado pelo caminhão. Dessa forma, depreende-se que o campo Preset da instrução de contagem deve ser reduzido para o número 5, no programa ladder GARRAFAS_SUPERV, de acordo com a figura 132.

Figura 132 – Programa ladder para automação do envasamento de garrafas



Fonte: Própria.

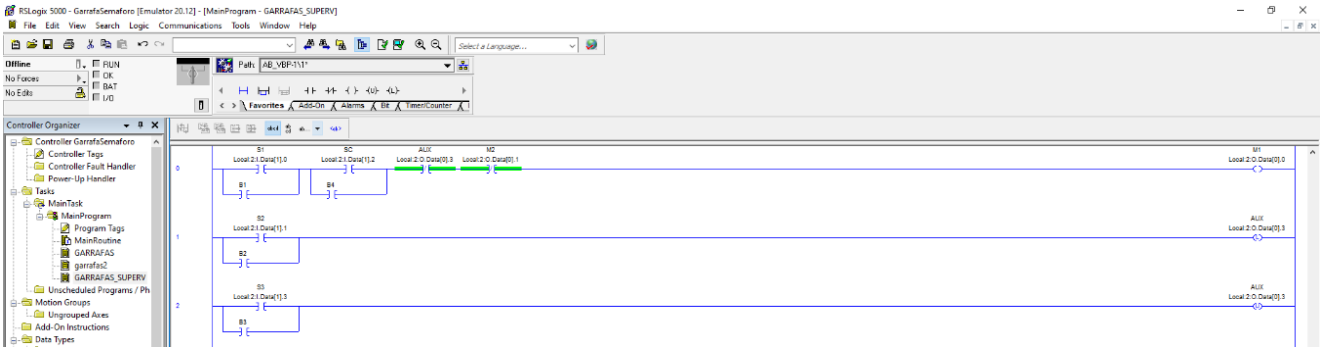
A diminuição do número de recipientes verificada na tela de supervisão (fig. 131) ocorreu devido ao número máximo de *tagnames* disponibilizados pelo sistema supervisório *In Touch*, no modo de demonstração, que limita a quantidade de variáveis criadas.

A tela de supervisão apresenta ainda um botão de saída (*Exit*), para desativar sua exibição, sempre que o usuário clicar sobre ele.

Outras modificações fora realizadas no programa ladder, para viabilizar a conexão com o sistema supervisório *In Touch*, através do protocolo DDE (*Dynamic Data Exchange*) nativo do sistema operacional Windows, que viabiliza o intercâmbio dinâmico de dados entre aplicações, durante a execução das mesmas.

Duas das alterações percebidas estão na primeira linha do programa ladder, onde se percebem dois contatos virtuais normalmente abertos (NA), identificados por B1 e B4, associados em paralelo com os contatos endereçados pelos sensores de presença de garrafa sobre a esteira (S1) e do caminhão (SC), respectivamente, de acordo com a figura 133.

Figura 133 – Modificações no programa ladder para o envasamento de garrafas



Fonte: Própria.

B1 e B4 são contatos virtuais cujo tipo de dados é booleano, viabilizando a comutação de estados simulada através da comunicação com o Sistema Supervisório. Dessa forma, um botão criado no sistema supervisório que tenha o endereço de B1 ou B4, comutará de fato seus respectivos contatos no programa ladder, viabilizando a simulação de presença de garrafa sobre a esteira (B1 em paralelo com S1) e do caminhão (B4 em paralelo com SC).

Caso botões criados na tela do supervisório para viabilizar comunicação com S1 e SC apresentassem os mesmos endereços das respectivas entradas (S1 → Local:2:I.Data[1].0 e SC → Local:2:I.Data[1].2), seu controle através do *In Touch* seria limitado, uma vez que contatos com endereçamento físico de CLPs conseguem se comportar apenas como botões do tipo impulso. Dessa forma, para manter o estado da comutação dos mesmos, é preciso fazer isto através do RSLogix 5000 emulate ou diretamente pelo sistema supervisório, por meio de contatos virtuais associados em paralelo aos contatos que recebem endereçamento físico, se o estado de repouso dos mesmos for NA. Caso os contatos que recebam endereçamento físico tenham estado de repouso normalmente fechado (NF), os contatos que receberão endereçamento virtual, para controle do sistema supervisório, precisarão estar associados em série, para obter o mesmo efeito de interrupção da corrente virtual, quando forem acionados.

Portanto, considerando que S1 e SC possibilitam circulação de corrente virtual e acionamento do motor M1, sempre que houver garrafa e caminhão presentes, os contatos virtuais B1 e B4 estarão associados em paralelo com eles, para proporcionarem condições de acionamento da saída, quando forem comutados pelos botões do sistema supervisório.

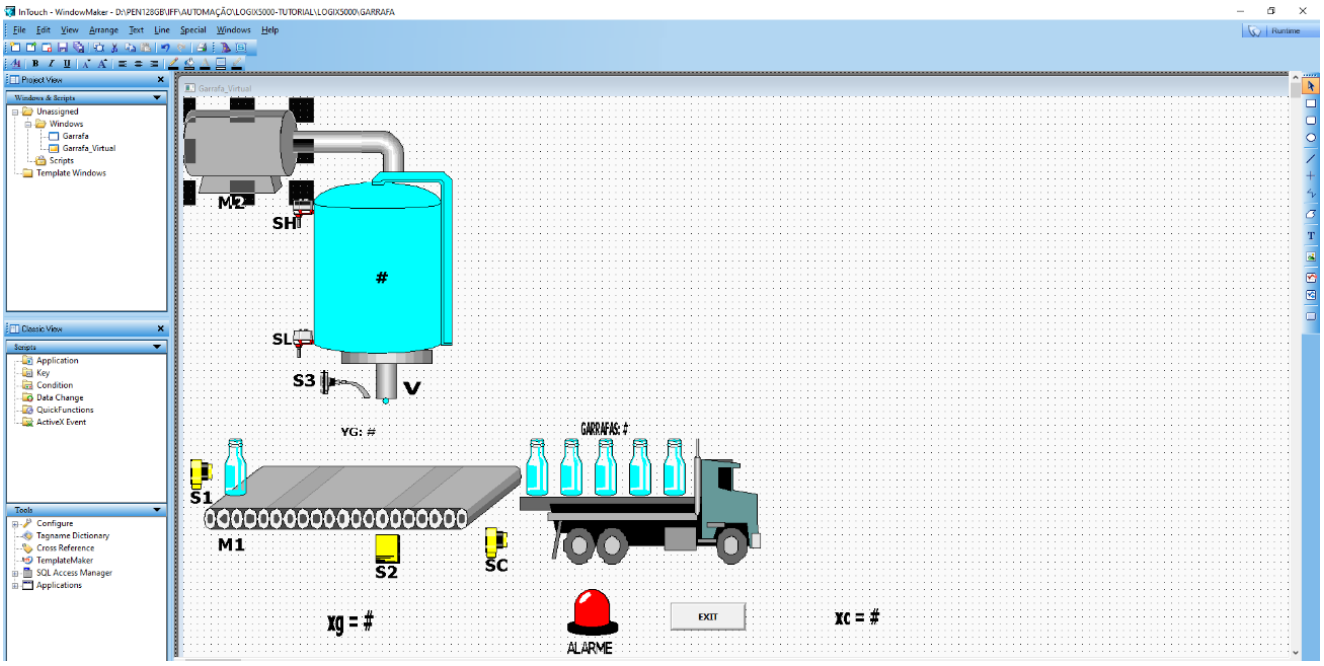
A última linha do programa ladder apresenta um contato virtual cujo estado de repouso é NA. Ele foi designado com o endereço B5, sendo declarado como do tipo *Boolean*, para acionar uma instrução de

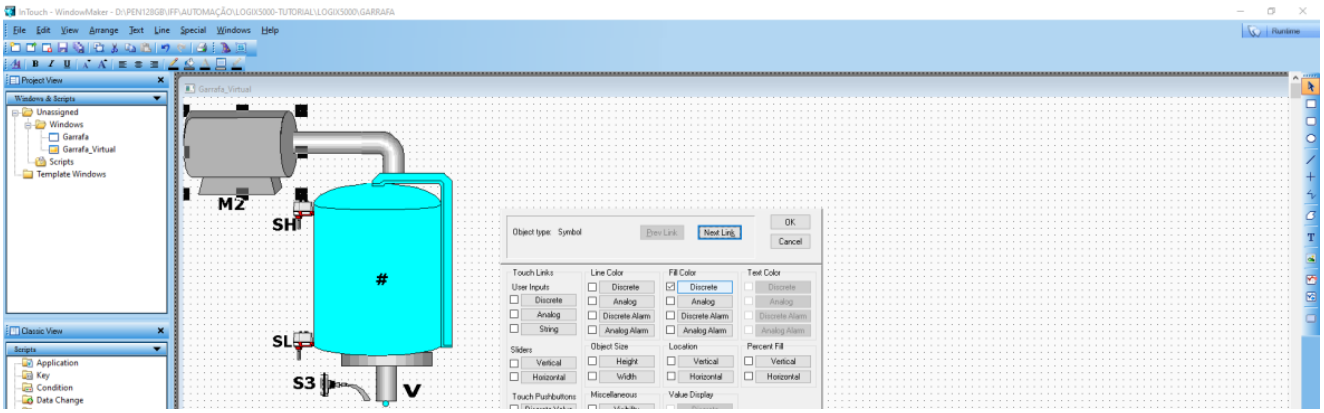
saída RES (*reset)*, aplicada sobre o contador C1, para zerar o registro ACC do mesmo, e possibilitar nova contagem de garrafas, quando houver necessidade de repetir o processo de envasamento e transporte dos recipientes.

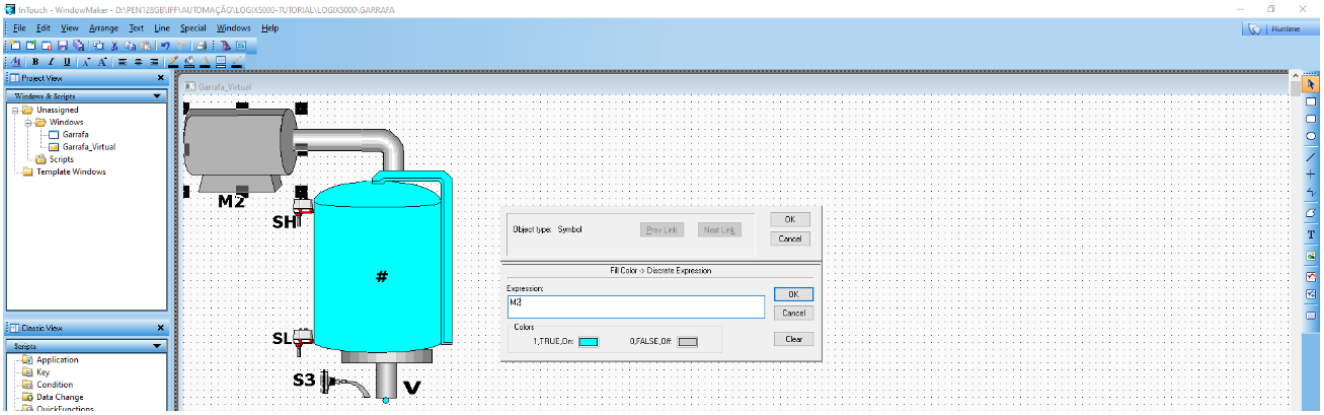
As ilustrações da figura 134 apresentam a configuração do motor que controla o fluxo de líquido para o tanque, acionado sempre que o sensor SL detectar nível baixo de água. As propriedades do motor M2 podem ser acessadas clicando-se uma vez sobre o mesmo e pressionando a opção *Break Cell*, conforme primeira captura de tela.

O próximo passo consiste em efetuar um clique duplo sobre o motor M2, o que fará surgir o quadro de propriedades do objeto (conforme segunda ilustração da figura 134), dentre as quais seleciona-se a opção Discrete, pertencente ao conjunto *Fill Color* (cor de preenchimento). Essas ações farão surgir a terceira ilustração da figura 134), onde se define o nome da etiqueta ou rótulo que definirá a variável responsável pela cor de preenchimento do motor, juntamente com a tonalidade a ser exibida pelo objeto, quando a incógnita for verdadeira (cor azul clara) ou falsa (cor cinza).

Figura 134 – Configurações do motor para preenchimento do tanque





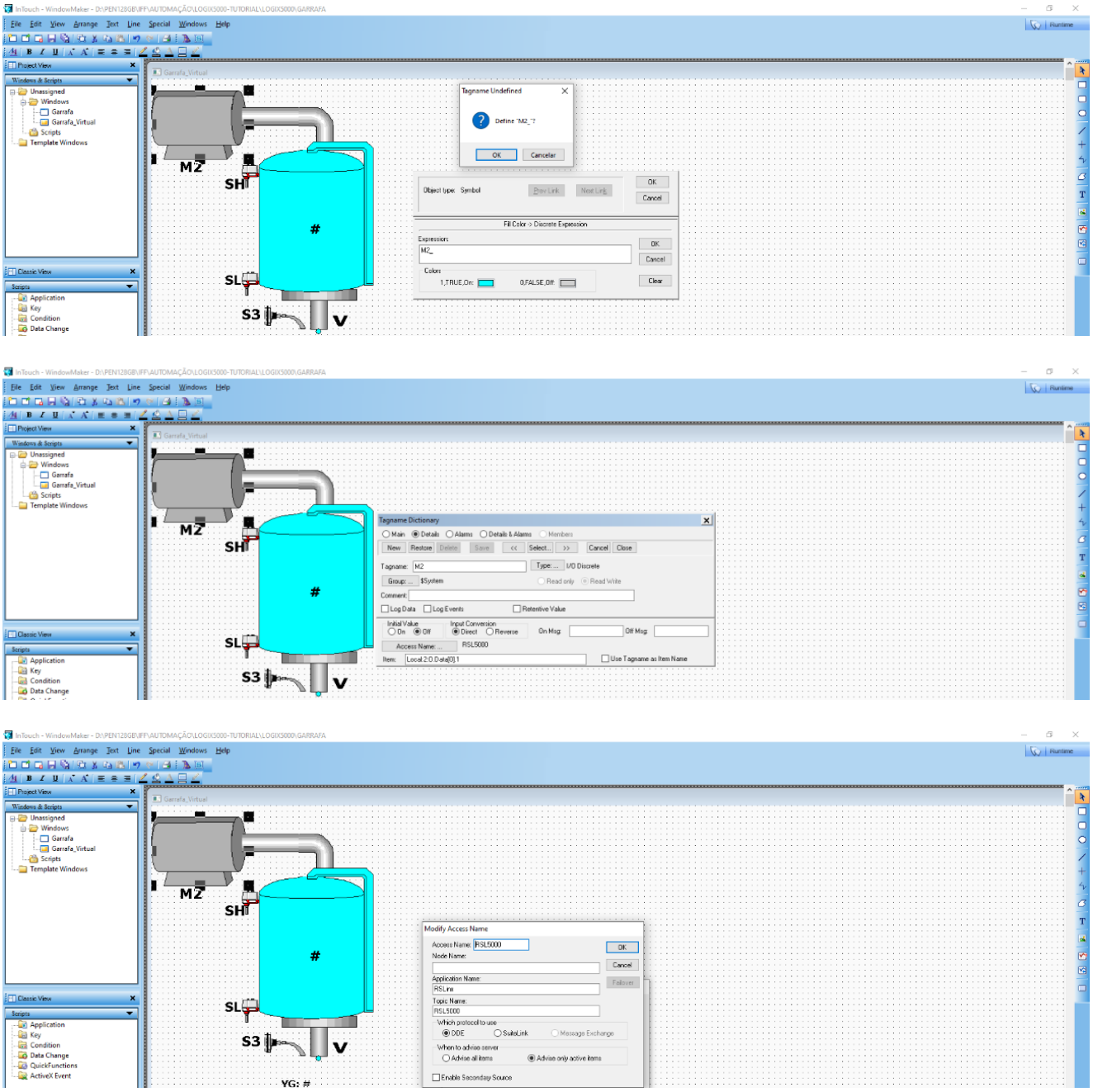


Fonte: Própria.

Designou-se a variável definida por M2, proporcionando condições ao surgimento da seguinte pergunta, após pressionar o botão OK, que confirma sua criação, de acordo com a primeira ilustração da figura 135.

As ações descritas no parágrafo anterior proporcionam a origem da segunda ilustração da figura 135, na qual surgirá o dicionário de variáveis, onde se define M2 como sendo do tipo I/O Discrete, indicando se tratar de etiqueta que se comunicará com um componente do programa ladder, cujo endereço é especificado no registro Item, tratando-se da segunda saída. As configurações do nome de Acesso podem ser verificadas clicando-se sobre seu botão, conforme terceira ilustração da figura 135.

Figura 135 – Criação da tagname M2 e nome de acesso



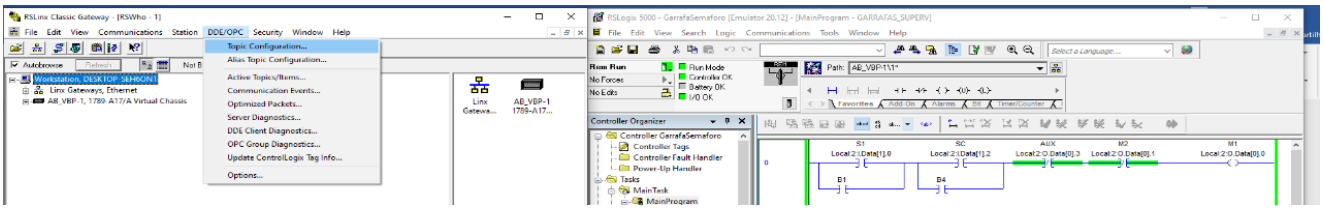
Fonte: Própria.

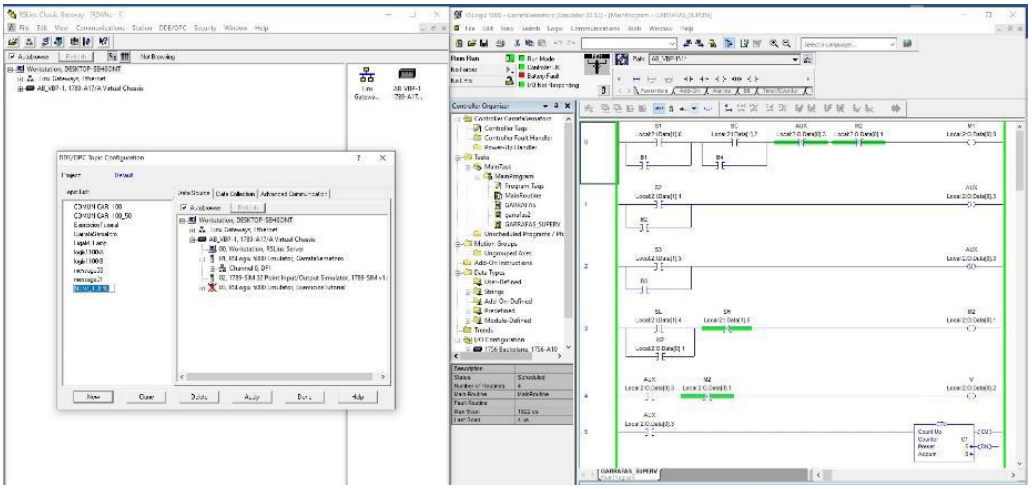
Antes de iniciar o sistema supervisório, é preciso criar previamente o nome do tópico (*Topic Name*) no programa RSLinx, e esta ação requer alguns passos. A primeira etapa consiste em deixar o programa ladder em execução, ir ao aplicativo RSLinx, e clicar nele sobre a opção DDE/OPC e, em seguida, selecionar *Topic Configuration*, de acordo com a primeira ilustração (fig. 136).

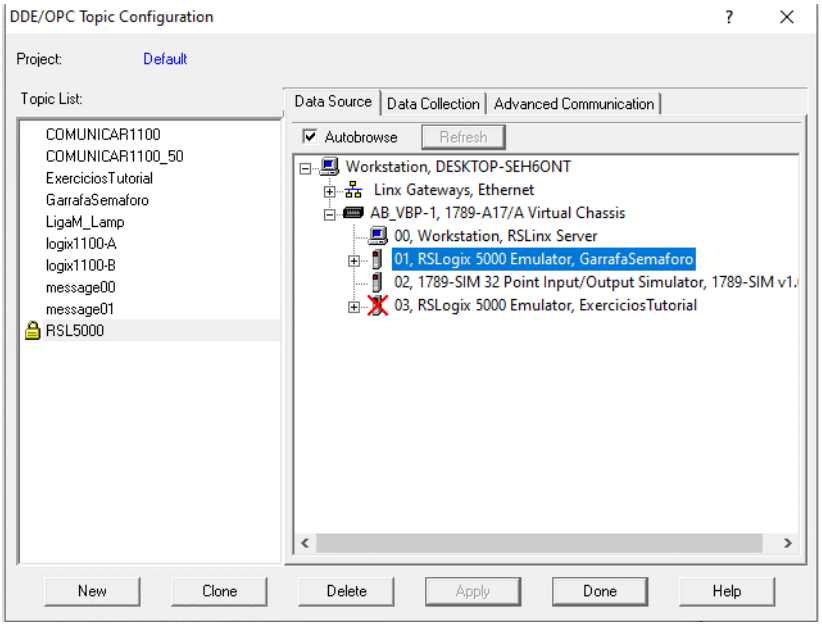
As ações anteriores viabilizarão o surgimento da segunda ilustração (fig. 136), onde se define o novo tópico clicando-se no botão *New*. A seguir, modifica-se o nome do tópico para RSL5000, aplicando-se

o mesmo ao módulo de emulação (slot 01, RSLogix 5000 Emulator), originando a 3ª ilustração (fig. 136).

Figura 136 – Configuração do tópico de comunicação no RSLinx







Fonte: Própria.

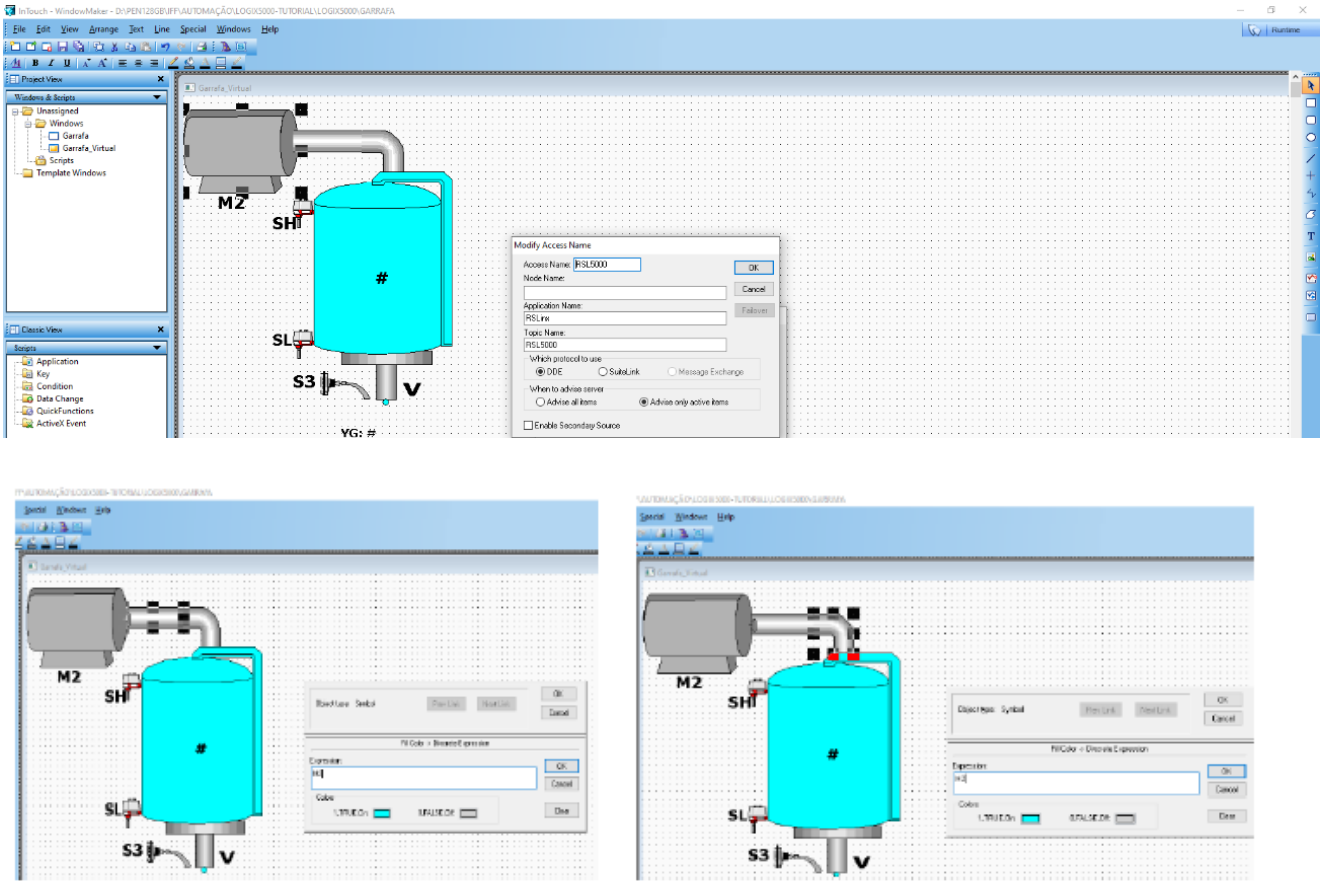
Observa-se que para o cadeado da figura anterior aparecer após a aplicação do tópico recém criado sobre o módulo ou slot 01, é preciso que o programa ladder esteja em execução, antes da criação do tópico no RSLinx. Feito isso, pressiona-se o botão *Done, minimizando* o aplicativo RSLinx, para

concentrar o foco nas telas de configurações do sistema supervisório, conforme primeira ilustração da figura 137.

Nessa primeira tela da figura 137, o Campo *Node Name* não deve ser preenchido, uma vez que os aplicativos desenvolvidos estão em modo de simulação, não havendo nó específico para execução da aplicação. Embora possa ser configurado o nome de acesso (*Access Name*) com identificador distinto do tópico (*Topic Name*), recomenda-se que ambos recebam a mesma nomenclatura atribuída ao tópico criado no RSLinx.

As telas 2 e 3 (fig.137) tratarão das cores de preenchimento dos tubos que comunicam o motor ao tanque. Para tanto, após selecionar os objetos e pressionar botão *Break Cell* em cada um deles, basta clicar duas vezes sobre os tubos reto e em curva, para Acessar suas respectivas propriedade, e atribuir o controle da cor de preenchimento dos mesmo ao identificador M2, cuja variável se comunica com a saída homônima do programa ladder.

Figura 137 – Configurações do sistema supervisório

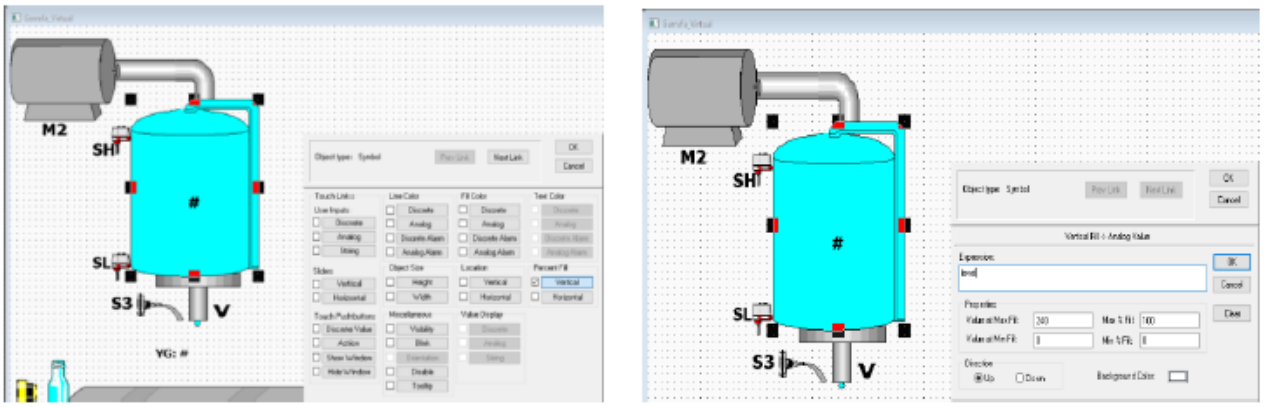


Fonte: Própria.

Isso significa que toda vez que a saída M2 estiver ativa no programa ladder, as cores de preenchimento dos tubos e do motor receberão as cores azul claro, indicando fluxo de líquido ativo para o interior do tanque. A cor cinza permanecerá enquanto M2 estiver desligado, devido ao estado falso associado durante a atribuição de cores aos possíveis valores discretos da variável.

O preenchimento vertical do tanque é controlado pela variável (tagname) level, cujas configurações podem ser verificadas acessando as propriedades do objeto, clicando-se a seguir sobre o botão *Vertical*, do conjunto *Percent Fill*, de acordo com a figura 138. Esse conjunto de ações possibilitará definir a tagname level no campo Expression, e informar as propriedades da mesma, cujos valores serão 240 e 0, associados aos preenchimentos máximo e mínio, respectivamente. O porcentual máximo de preenchimento será 100%, enquanto seu valor mínimo terá o quantitativo de 0%. A cor de fundo (Background Color) é branca, e a direção de preenchimento é ascendente.

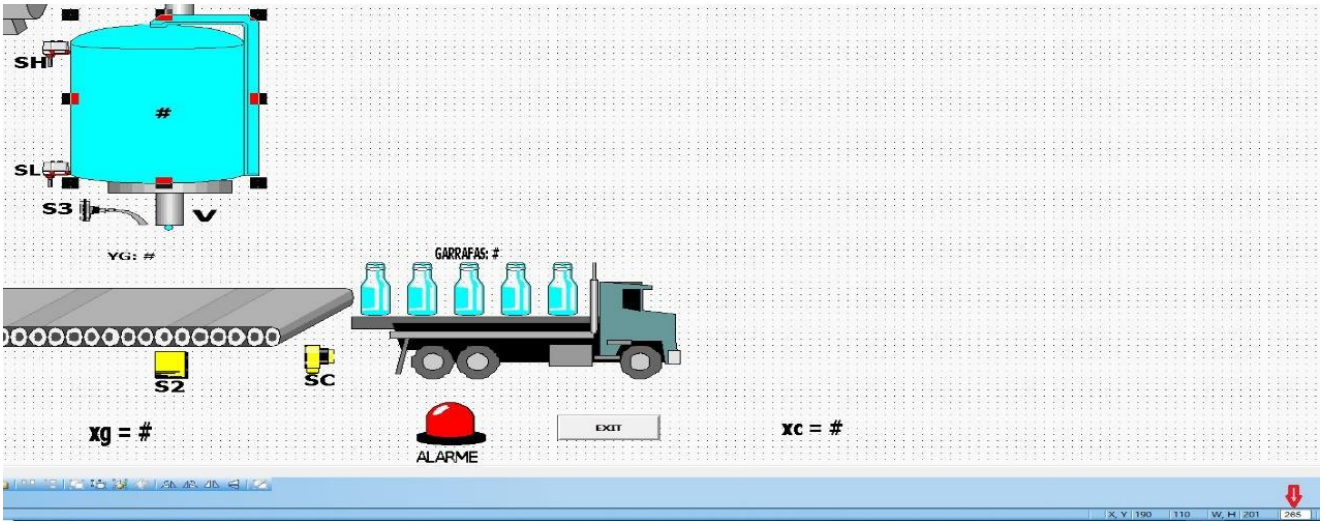
Figura 138 – Criação da tagname level.



Fonte: Própria.

O valor 240 correspondentes ao máximo preenchimento do tanque é obtido clicando-se sobre o mesmo, e medindo seu comprimento em pixels, informação que pode ser obtida atráves das coordenadas fornecidas no canto inferior direito da tela, conforme figura 139. Embora o valor lido seja 265, desconta-se 10% sobre o mesmo, para melhor visualização do fluxo líquido no interior do tanque.

Figura 139 – Máximo preenchimento do tanque

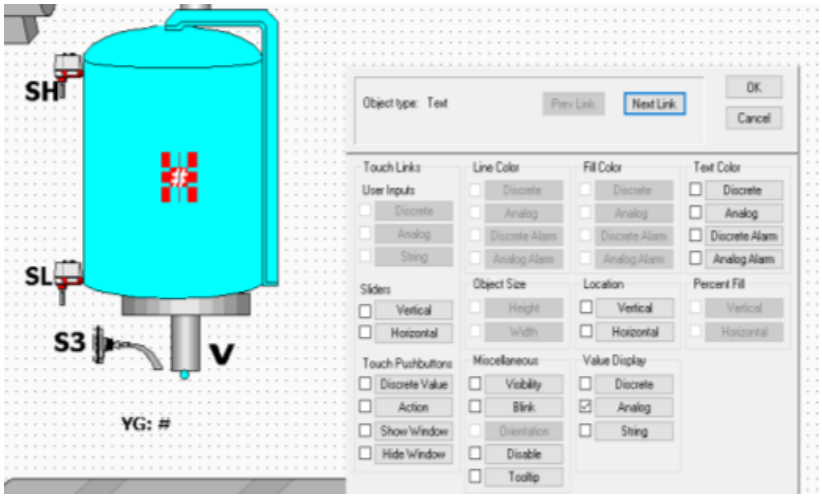


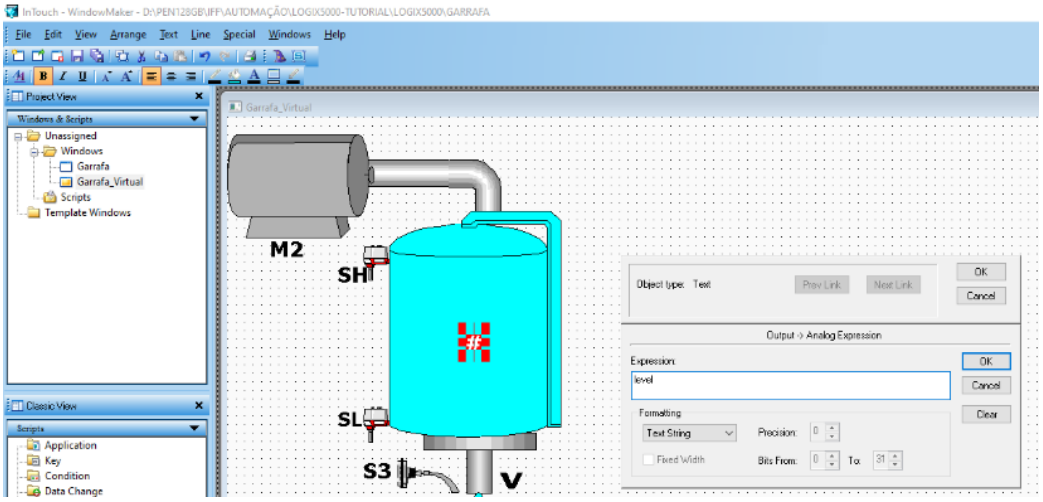
Fonte: Própria.

A Cerquilha ou *hashtag* verificada no interior do tanque foi ali posicionada para exibir os valores numéricos da variável level. Para tanto, clica-se sobre o botão de texto, situado na lateral direita do sistema supervisório, para selecionar o objeto onde se deseja digitar o caractere cerquinha. Após efetuar o procedimento, clica-se duas vezes sobre o caractere Cerquilha, para Acessar suas propriedades, de acordo com a primeira ilustração da figura 140.

Após clicar sobre o botão *Analog*, pertencente ao conjunto *Value Display*, obtem-se a segunda tela (fig. 140), onde se informa o nome da variável que será exibida pela *hastag*, que no caso do tanque se trata da *tagname* level.

Figura 140 – Configuração da cerquilha do objeto texto, para exibir o nível do tanque





Fonte: Própria.

Para Acessar as propriedades do sensor de nível alto (SH) do tanque, deve-se clicar duas vezes sobre o mesmo, conforme primeira ilustração da figura 141, onde se percebem as propriedades *Discrete Vale* e *Discrete* marcadas, que pertencem aos conjuntos *Touch Pushbuttons* e *Fill Color*, respectivamente.

As propriedades *Discrete Value* e *Discrete* do sensor SH são apresentadas nas ilustrações 2 e 3 (fig. 141), sendo obtidas após pressionar seus botões. A segunda ilustração indica que o sensor mudará sua cor para verde, enquanto M2 estiver em operação, retornando à tonalidade amarelo, quando o motor for desativado devido à detecção de nível alto no tanque.

A novidade introduzida é a tagname SH, a qual se atribui a ação Direct, conferindo à mesma um comportamento análogo a um botão do tipo impulso, tal qual uma campainha doméstica, cuja comutação de estado se mantém apenas enquanto o mesmo estiver pressionado. A quarta ilustração da figura 141 traz as configurações de SH, no dicionário de tagnames.

Figura 141 – Configuração do sensor de nível alto - SH

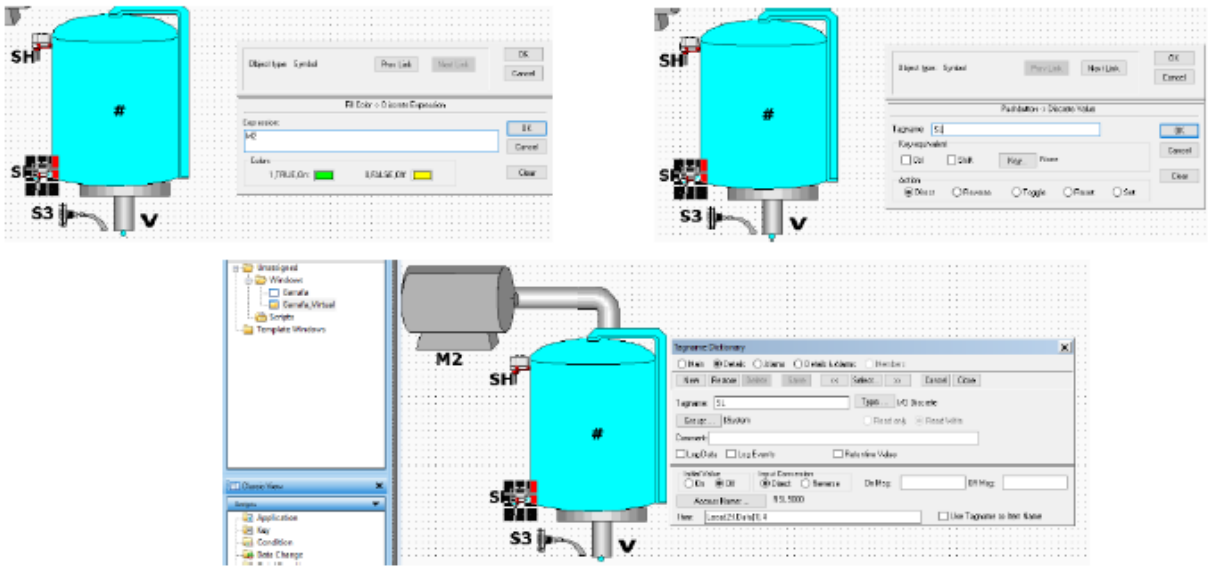


Fonte: Própria.

Atribui-se a SH o tipo I/O Discrete, uma vez que a mesma se comunicará com elemento do programa ladder cujo endereço é especificado no campo Item, no caso trata-se de contato virtual normalmente fechado (NF), que desativará M2 quando o nível máximo for detectado.

A figura 142 apresenta três tela, com as configurações do sensor de nível baixo SL, que são análogas a SH, excetuando-se o endereço do elemento do programa ladder acessado por SL, que é o contato normalmente aberto (NA).

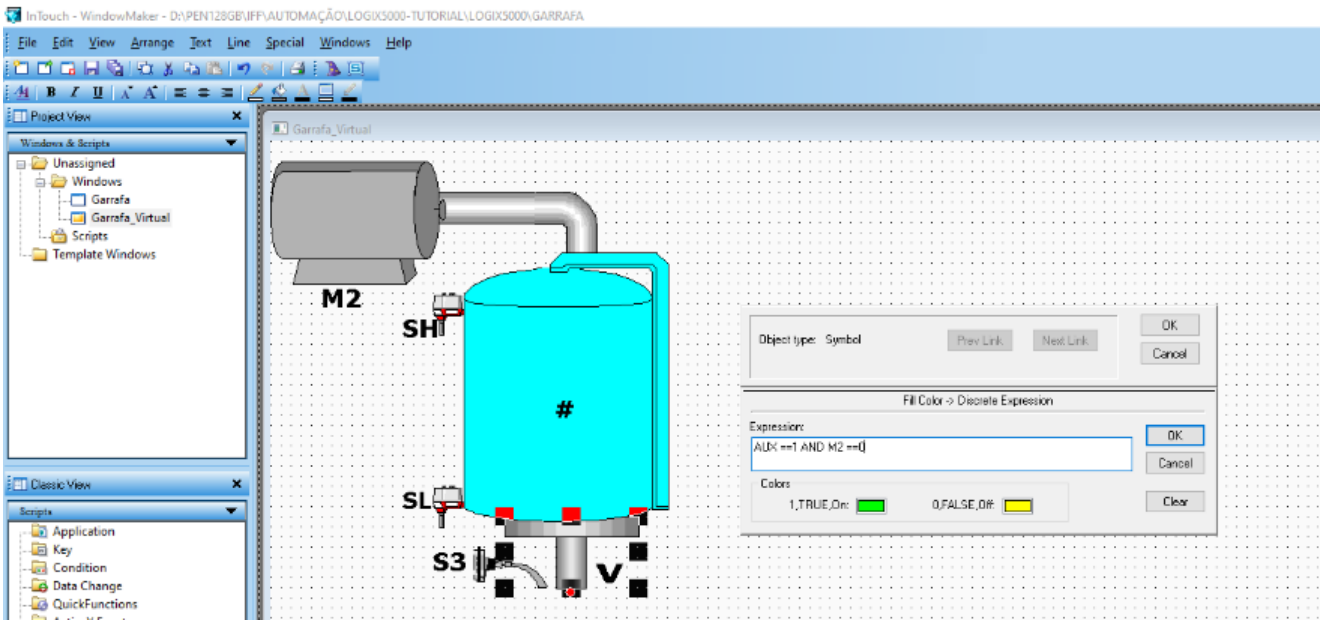
Figura 142 – Configuração do sensor de nível alto - SH



Fonte: Própria.

A figura 143 apresenta as configurações da válvula V, cuja expressão de preenchimento torna sua cor verde para indicar a abertura da mesma, somente quando a garrafa estiver abaixo dela (condição caracterizada por AUX == 1) e M2 se encontrar inoperante (M2 == 0). Caso pelo menos uma das condições não for verificada, a expressão retornará valor lógico falso, conferindo cor amarela à válvula, correspondendo ao fechamento da mesma.

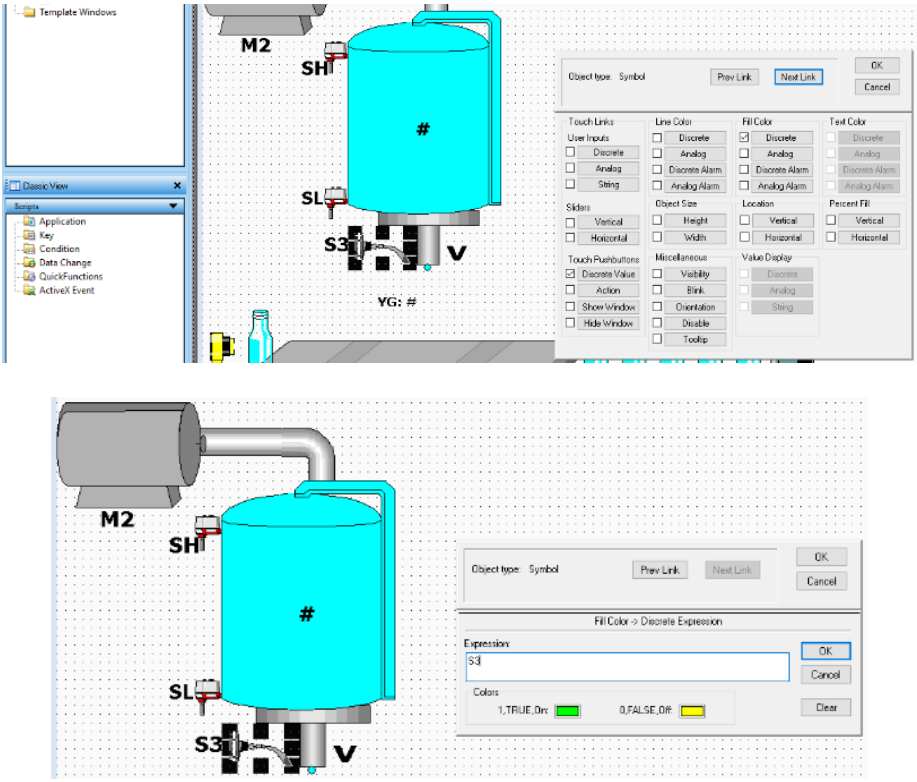
Figura 143 – Configuração da válvula V



Fonte: Própria.

O sensor S3 apresenta duas propriedades, evidenciadas na primeira ilustração da figura 144. Uma de suas propriedades define sua cor de preenchimento, que é controlada pela variável S3, cujas definições são vistas na segunda tela da figura 144, onde a propriedade Item é homônima, uma vez que a mesma se comunica com o contato NA situado na linha 2 do programa ladder, para desligar a saída AUX, sempre que S3 detectar que a garrafa está cheia. Dessa forma, a válvula V da linha 4 do programa ladder será automaticamente fechada, uma vez que lá há um contato NA com o endereço de AUX, que retornará ao estado de repouso aberto após a desativação deste saída, impedindo a circulação de corrente virtual que mantinha o fluxo de líquido entre o tanque e o recipiente posicionado sobre a esteira.

Figura 144 – Configuração do sensor S3



Fonte: Própria.

A propriedade *Discrete Value* encontra-se no conjunto *Touch Pushbuttons*, que define a tagname S3 (figura) para se comunicar com o contato homônimo situado na terceira linha do programa Ladder, com o intuito de desativar a saída AUX, e simular detecção de nível alto da garrafa após o usuário clicar com o mouse sobre o sensor S3.

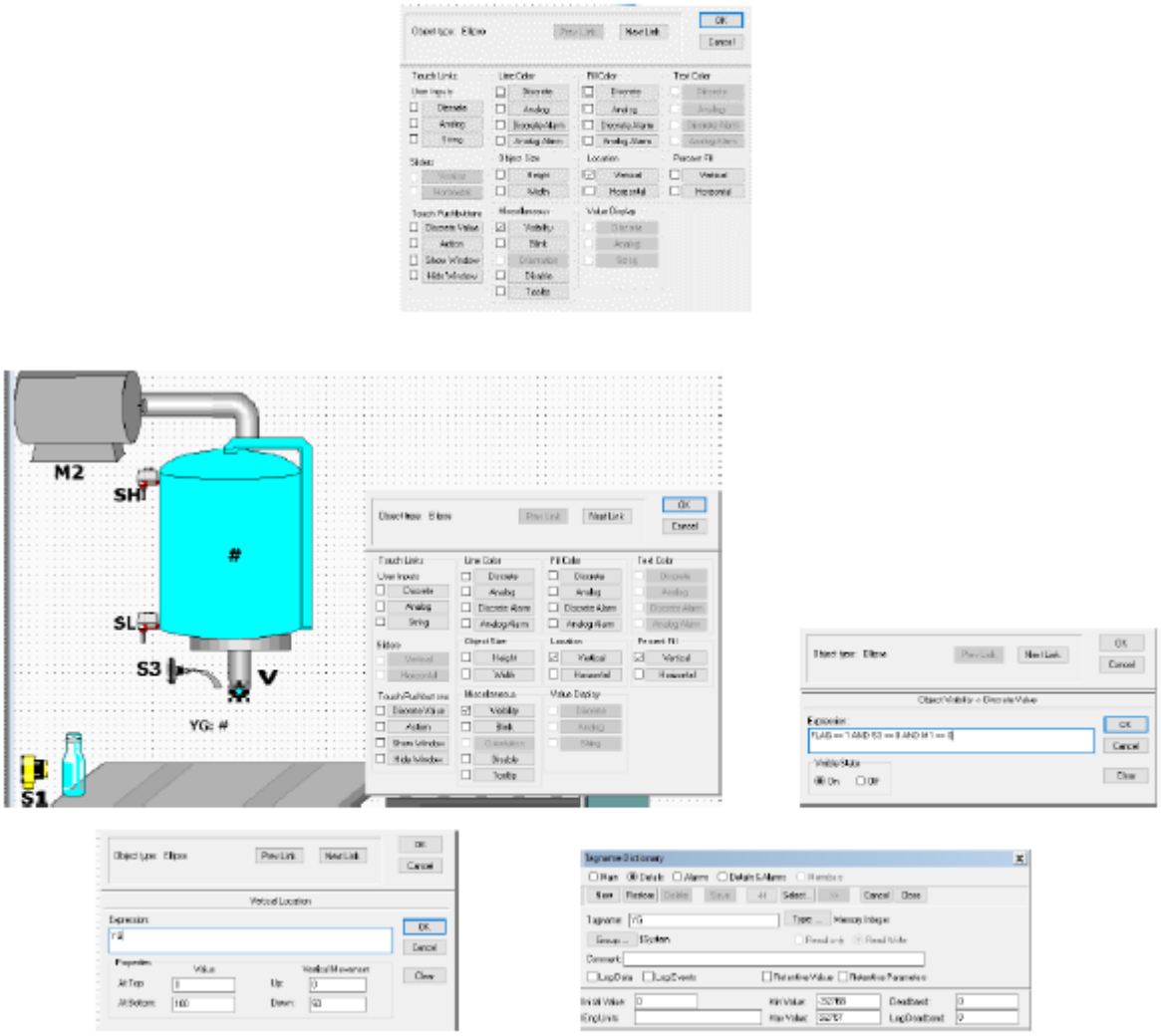
A gota que flui após a abertura da válvula apresenta duas propriedades, vistas na primeira ilustração da figura 145, cujos nomes são *Visibility* (integrante do conjunto *Miscelaneous)* e Vertical (componente do agrupamento Location).

A condição de visibilidade é acessada clicando-se sobre a propriedade homônima na segunda tela (fig. 145), cuja expressão é percebida na terceira ilustração (fig. 145), indicando que a gota será visível somente enquanto a tagname FLAG for igual a 1 (condição que ocorre quando a garrafa estiver alinhada abaixo da válvula), S3 == 0 (indicando que o nível máximo não foi atingido) e M1 == 0 (mostrando que a esteira se encontra parada).

As ilustrações 4 e 5 da figura 145 apresentam a propriedade Vertical, juntamente com sua definição no dicionário de tagnames ou identificadores de objetos. YG controla o deslocamento vertical da gota, cujo valor no topo ou inicío do movimento é 0, enquanto ao final de sua movimentação, próximo ao topo da garrafa, atribuí-se a YG o número 100.

Com relação ao deslocamento vertical de YG, tagname definida como o tipo inteiro (*Memory Integer*), a movimentação superior (Up) é nula, enquanto o deslocamento inferior (*Down*) recebe o valor 50, equivalente ao número de pixels situados entre a válvula e o topo da garrafa, cuja medição é simples, conforme mencionado anteriormente neste mesmo projeto.

Figura 145 – Configuração da gota que saí da válvula

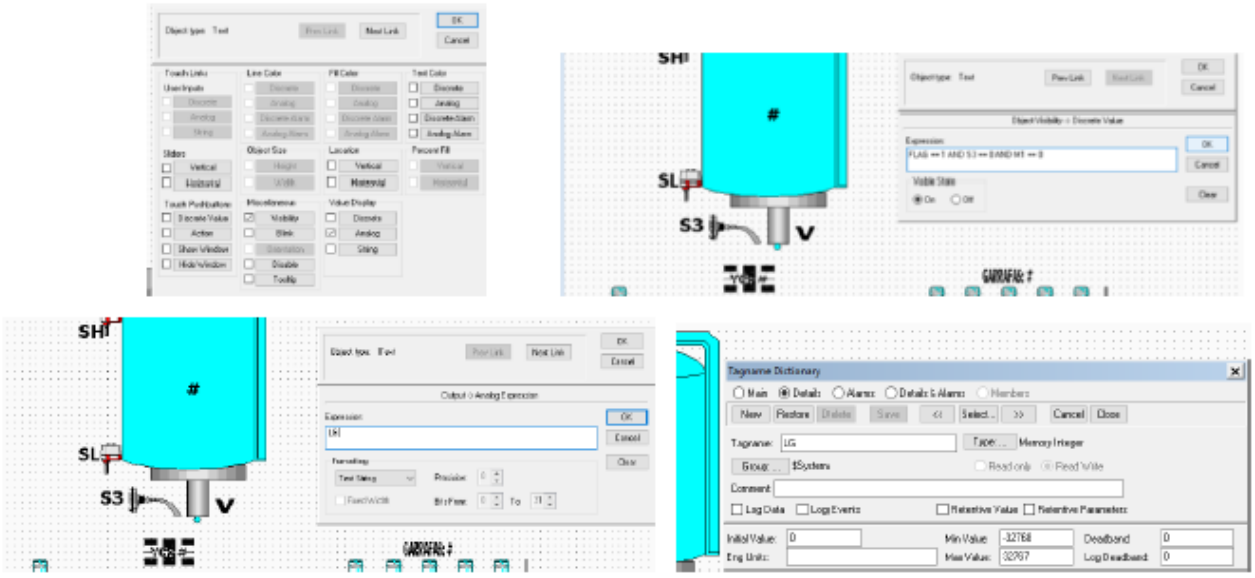


Fonte: Própria.

A primeira ilustração da figura 146 apresenta as propriedades do objeto de texto (YG: #, cuja função consiste em mostrar o valor do preenchimento da garrafa), que são *Visibility* e Analog. A definição da propriedade de visibilidade é apresentada na segunda ilustração (fig. 146), onde se observa que suas condições são as mesmas da gota, cuja explanação foi realizada anteriormente.

As telas 3 e 4 (fig. 146) definem a *tagname* que estabelece o valor do nível de preenchimento da garrafa como LG, a qual está associada à propriedade Analog, conforme mostra a figura, onde se percebe que seu tipo é inteiro (*Memory Integer*).

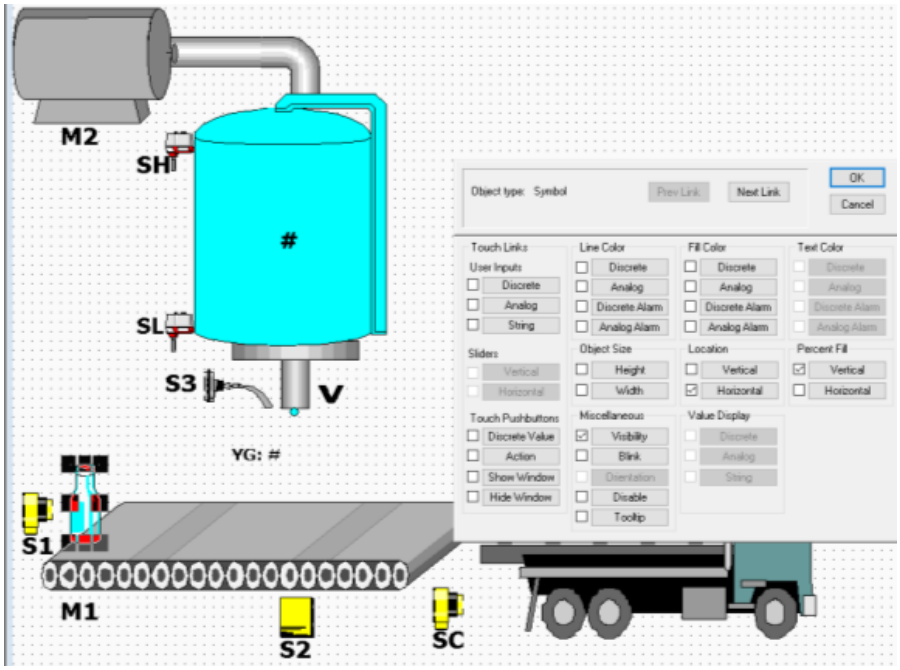
Figura 146 – Configuração de LG



Fonte: Própria.

A garrafa sobre a esteira pode ser vista na extremidade esquerda da mesma, sendo suas propriedades apresentadas na figura 147.

Figura 147 – Garrafa sobre a esteira



Fonte: Própria.

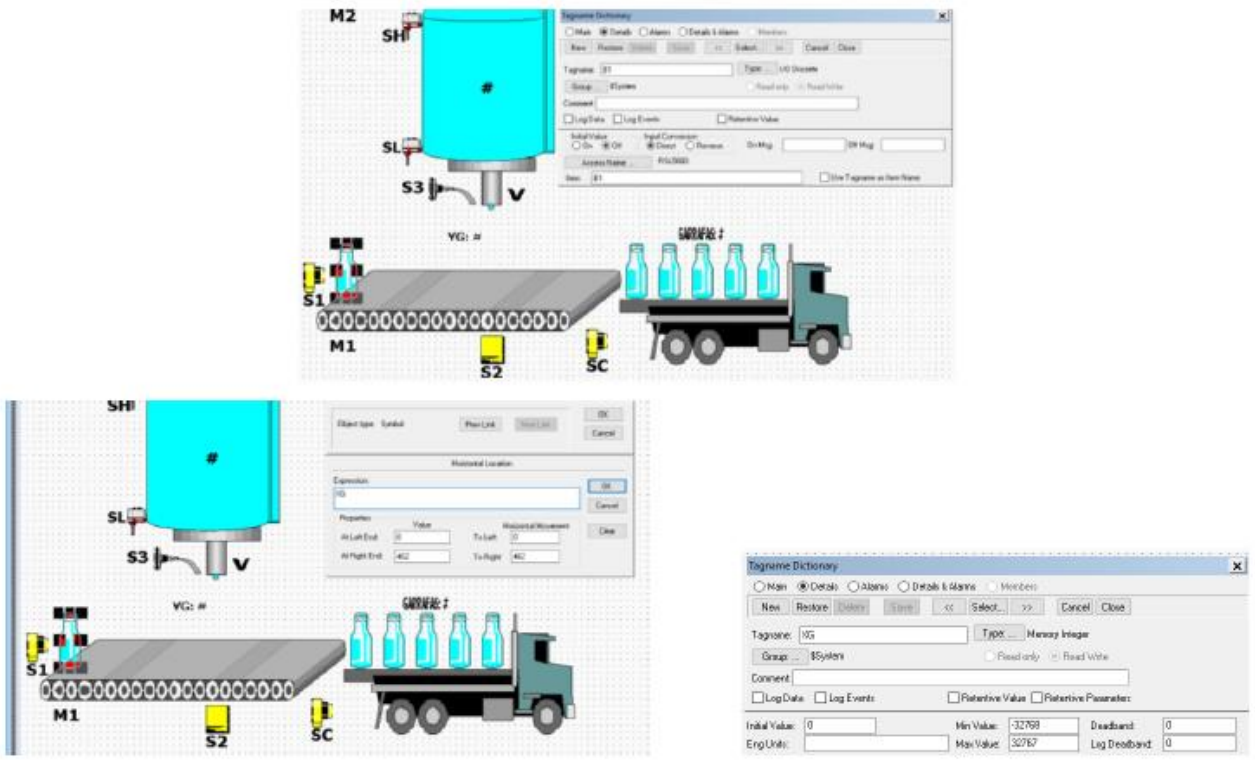
A garrafa apresenta três propriedades, cujos nomes são *Visibility* (integrante do conjunto *Miscelaneous*), *Horizontal* (componente do agrupamento *Location*) e *Vertical* (agregada à classe

Percent Fill). A condição de visibilidade é Accessada clicando-se sobre a propriedade anônima, cuja tagname B1 e respectivas definições podem ser vistas na figura 148.

Vê-se na primeira ilustração (fig. 148) o identificador B1, responsável pela visibilidade da garrafa, é do tipo I/O Discrete, indicando que o mesmo se comunica com elemento homônimo situado à linha 0 do programa ladder, cuja função é simular a detecção de garrafa sobre a esteira, condição alcançada definindo o estado de repouso do contato virtual como normalmente aberto (NA). Dessa forma, sempre que S1 detectar garrafa, o que é conseguido clicando-se sobre o sensor homônimo do sistema supervisório, seu contato virtual no programa ladder comutará para fechado, alcançando a condição verdadeira, tornando o recipiente visível da tela referente à dinâmica do processo.

Percebe-se nas ilustrações 2 e 3 (fig. 148) que a propriedade Horizontal da garrafa situada sobre a esteira está associada à tagname XG, conforme figura, na qual se percebe o tipo Memory Integer associado à variável, e números idênticos atribuídos aos valores sobre as extremidades esquerda e direita, bem como às movimentações horizontais esquerda e direita. O número 462 é a quantidade de pixels ocupada pela largura da esteira. Atribuindo-se esse valor ao deslocamento direito, e zero ao deslocamente esquerdo, configura-se a garrafa para se movimentar estritamente sobre a esteira.

Figura 148 – Propriedades da garrafa

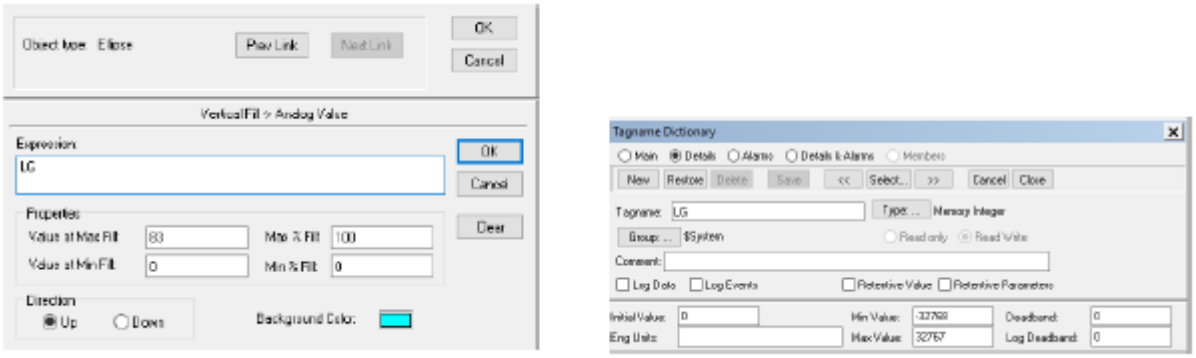


Fonte: Própria.

Ao se atribuir o mesmo número de pixels da esteira à variável XG, responsável por seu deslocamento horizontal, garante-se que cada incremento de XG corresponda ao avanço de um pixel sobre a tela (fig. 148).

A figura 149 apresenta a terceira propriedade da garrafa, cujas definições foram anteriormente mostradas. Essa propriedade é controlada pela expressão LG, *tagname* responsável pelo controle do nível de preenchimento da garrafa, sendo por este motivo atribuído a ela o tipo *Memory Integer*. O valor de preenchimento máximo da garrafa corresponde ao número de pixels da mesma, que é igual a 83.

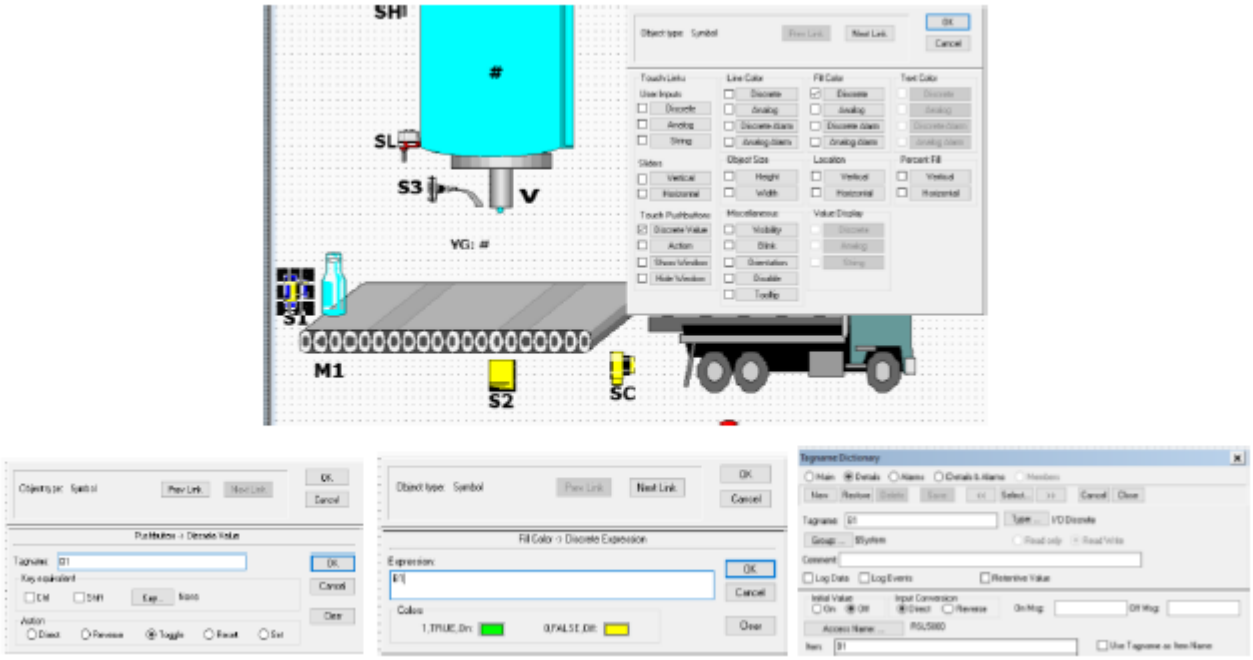
Figura 149 – Propriedades da garrafa



Fonte: Própria.

As propriedades do sensor de presença da garrafa são vistas nas ilustrações da figura 150, devidamente assinaladas e identificadas por *Discrete Value* (pertencente à classe *Touch Pushbuttons*) e *Discrete* (agregada pela classe *Fill Color*).

Figura 150 – Propriedades do sensor de presença da garrafa

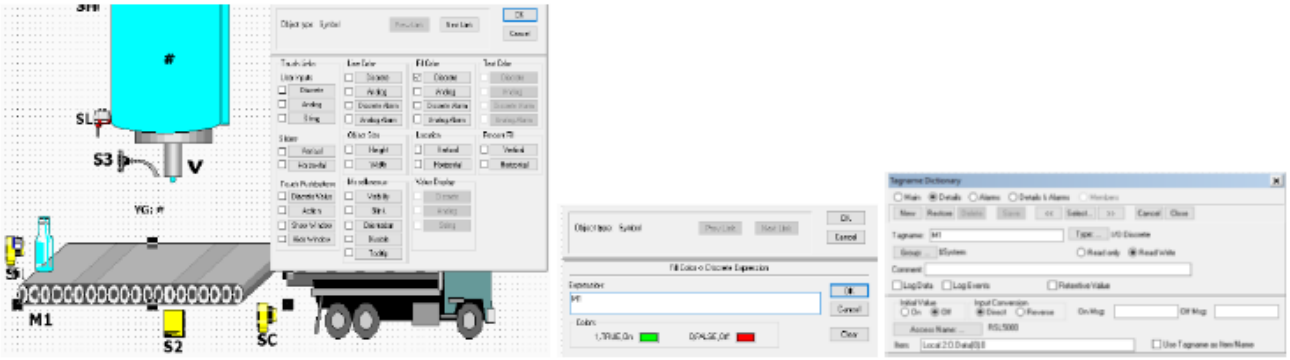


Fonte: Própria.

Clicando-se sobre os botões das propriedades de S1, acessam-se suas respectivas *tagnames*, apresentadas na figura 145. Trata-se da variável B1, responsável por simular a presença da garrafa quando se clicar sobre o sensor, condição em que a cor do mesmo mudará de amarelo para verde, devido à comutação do contato homônimo situado à linha 0 do programa Ladder, e definido com estado de repouso NA.

A esteira M1 tem uma única propriedade (*Discrete*, pertencente à classe *Fill Color*), controlada pela *tagname* M1, que governará suas cores de preenchimento, de acordo com a figura 151. Percebe-se M1 definido como tipo *I/O Discrete* e Item Local:2:O.Data[0].0, endereço idêntico ao da saída da esteira, na linha) do programa Ladder. Dessa forma, assim que a esteira entrar em operação, sua saída será ativada, comutando o estado dos objetos que compartilhem seu endereço. Isso resultará na mudança de tonalidade da esteira, que comutará da cor vermelha, referente ao repouso, para a cor verde, associada à movimentação da mesma.

Figura 151– Propriedades da esteira



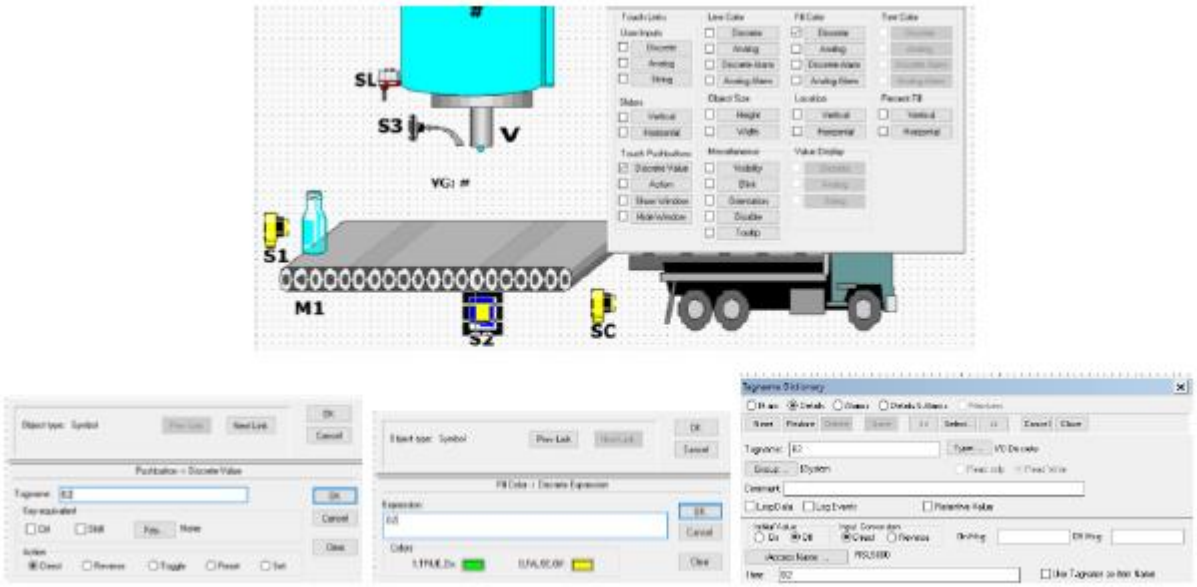
Fonte: Própria.

O sensor S2 é responsável por detectar o alinhamento da garrafa sob a válvula, sendo as propriedades de seu objeto vistas nas telas da figura 152. As propriedades do sensor S2 (*Discrete Value* - classe *Touch Pushbutton* e *Discrete* – grupo *Fill Color*) são controladas pela mesma *tagname*, B2, cujas definições são apresentadas nas ilustrações 2, 3 e 4 (fig. 152).

Percebe-se B2 como sendo tipo I/O Discrete, com Item especificado pelo endereço B2, visto na linha 1 do programa Ladder. Isso significa que quando se clicar sobre B2 no sistema supervisório, a propriedade *Discrete Value* da classe *Touch Pushbutton* (configurada com ação *Direct*) comutará momentaneamente o contato virtual de seu estado de repouso normalmente aberto, de maneira análoga a um botão do tipo impulso, proporcionando condições de fluxo de corrente virtual até a saída AUX, acionada por instrução autorretentiva, que mantém a mesma ativada, mesmo após B2 voltar ao estado NA.

O acionamento de AUX comutará seu contato homônimo do estado de repouso normalmente fechado (NF), que está na linha 0 do programa ladder, abrindo o mesmo e desativando M1, para que ocorra o preenchimento da garrafa. Enquanto o sensor S2 detectar garrafa alinhada sob a válvula, sua cor comutará de amarelo para verde, devido as suas definições da cor de preenchimento, conforme ilustrações 3 e 4 da figura 152.

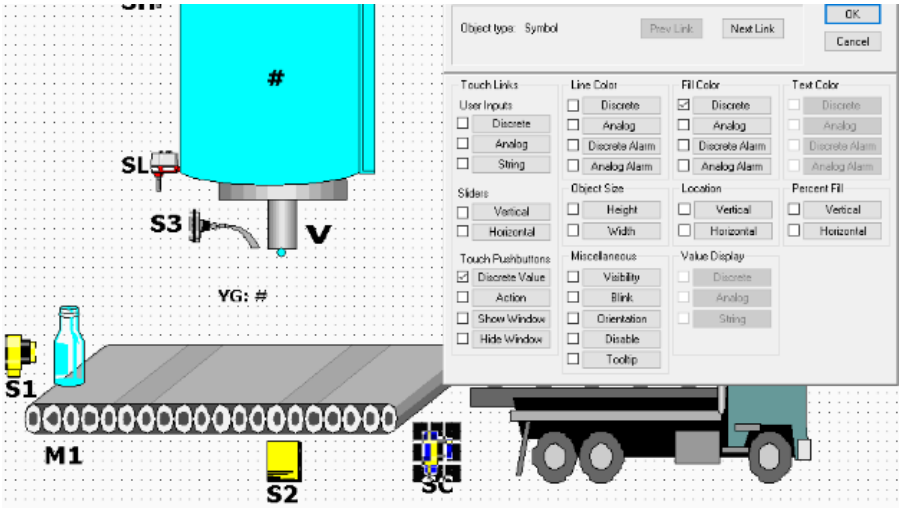
Figura 152 – Configurações do sensor S2



Fonte: Própria.

A figura 153 traz as propriedades utilizadas no sensor de caminhão SC, que são *Discrete Value* e *Discrete*, localizadas nas classes *Touch Pushbuttons* e *Fill Color*, respectivamente.

Figura 153 – Propriedades do sensor SC

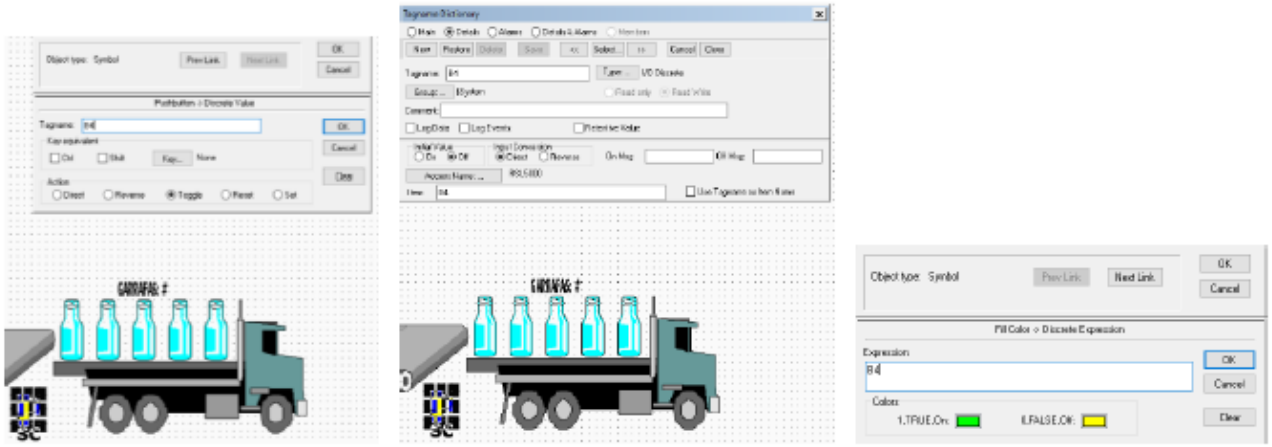


Fonte: Própria.

Clicando-se sobre o botão da propriedade Discrete Value, obtem-se a figura 154, na qual surge sua *tagname* associada, identificada pela variável B4, cujas definições são tipo I/O Discrete e Item B4, cujo endereço é o mesmo do contato homônimo B4, que aparece na linha 0 do programa ladder, para simular a presença do caminhão. Quando SC perceber o caminhão, o que será simulado clicando-se sobre o mesmo, a tag B4 comutará de repouso, modificando o estado do Item B4 e seu respectivo contato virutal normalmente aberto no programa ladder, fechando o mesmo, e proporcionando

condições de circular corrente virtual até a saída de M1, desde que o sensor de garrafas S1 também detecte presença de recipientes, comutando o contato B1 para fechado.

Figura 154 – Definições da tagname B4, do sensor SC



Fonte: Própria.

Percebe-se também que a propriedade *Fill Color Discrete* também é controlada pela *tagname* B4, conforme terceira ilustração da figura 154, na qual se percebe a configuração de cor amarela para o sensor SC, enquanto o caminhão não for detectado. Após clicar em SC, a variável B4 comuta do estado de repouso NA, passando a ser verdadeira ou acionada, o que modifica a cor do sensor para verde, seguindo a configuração previamente determinada para o objeto.

Figura 155 – Propriedades do caminhão

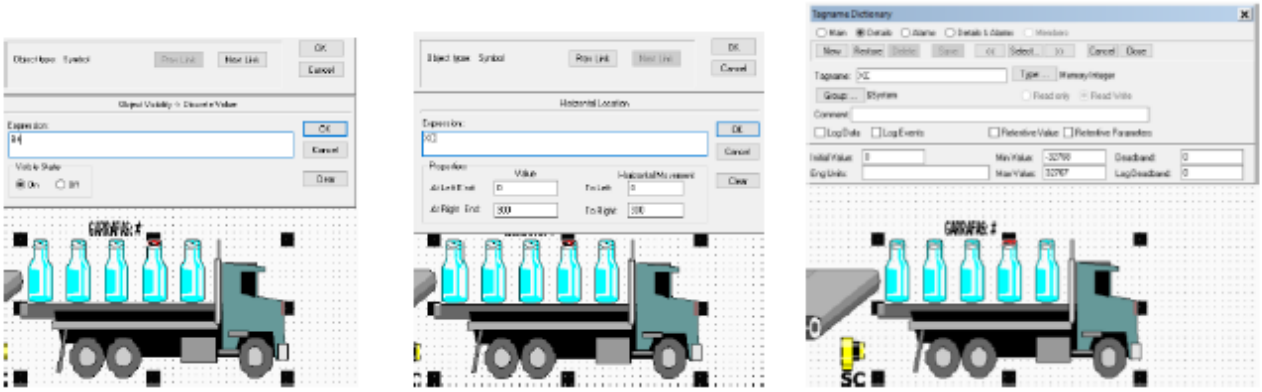


Fonte: Própria.

Clicando-se duas vezes sobre o caminhão, determinam-se as propriedades a ele associadas, conforme figura 155, que são *Miscellaneous - Visibility* e *Location - Horizontal*.

A figura 156 traz as *tagnames* relacionadas às supracitadas propriedades, que são B4, interligada à propriedade de visibilidade, e XC, associada ao deslocamento horizontal. De acordo com as configurações apresentadas, percebe-se que quando se clicar em SC, a variável B4 comutará para o estado *On*, tornando o caminhão visível.

Figura 156 – Criação da tagnames B4 e XC

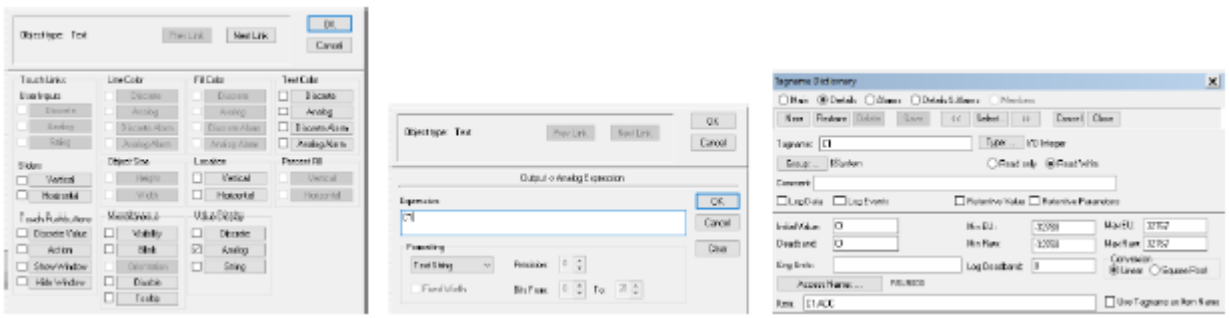


Fonte: Própria.

A *tagname* XC é do tipo *Memory Integer*, conforme terceira ilustração (fig. 156) e define o posicionamento horizontal do caminhão, cujo movimento será nulo à esquerda, e compreenderá 300 pixels a partir da extremidade direita do veículo. Os valores de XC receberam definição idêntica aos números especificados para a movimentação horizontal.

Para Acessar as propriedades do texto *Garrafas:#*, realiza-se clique duplo sobre a cerquilha, obtendo as telas da figura 157, onde se percebe a utilização da propriedade *Value DisplayAnalog* no objeto. Pressionando-se o botão *Analog*, Accessa-se a variável C1, *tagname* que determina o valor analógico a ser exibido pela cerquilha.

Figura 157 – Configuração do valor mostrado no objeto de texto Garrafas:#



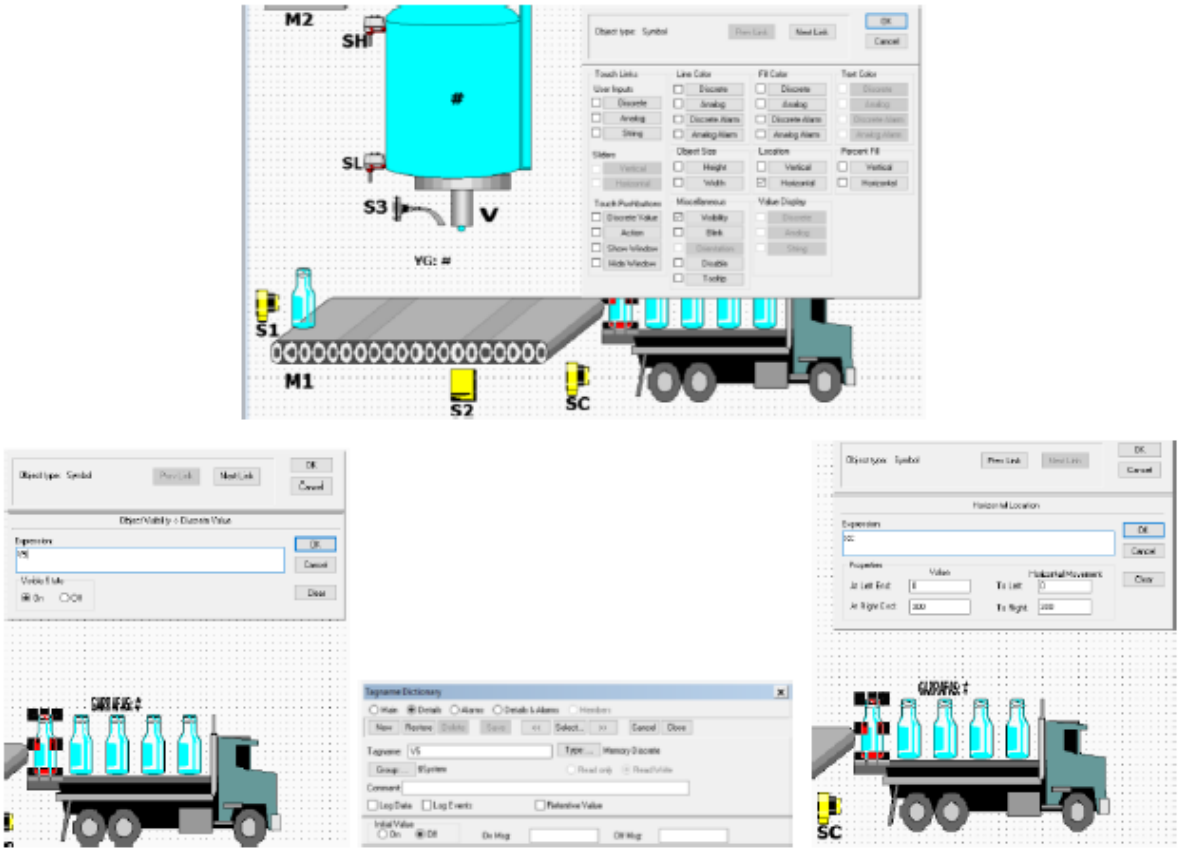
Fonte: Própria.

Clicando-se duas vezes sobre C1 obtém-se as definições do identificador no dicionário de *tagname*, que são tipo: *I/O Integer* e Item: *C1.ACC*.

Como C1 é do tipo de entrada e saída inteiro, ele comportará valores analógicos manipulados pela variável homônima do programa ladder, que é do tipo contador. As especificações de Item constituem endereço do elemento do programa ladder, que consiste no registro ACC do contador C1, responsável por acumular o quantitativo de garrafas preenchidas com líquido, informação que será exibida no caractere cerquilha, após o texto Garrafas:# (fig. 157).

Cada uma das 5 garrafas sobre o caminhão terá as seguinte propriedades associadas a elas, conforme figura 158. Clicando-se sobre os botões associados à propriedades *Miscellaneous Visibility* e *Location Horizontal* Accessam-se as *tagnames* relacionados a elas, conforme ilustrações 2, 3 e 4 (fig. 158).

Figura 158 – Configuração das propriedades sobre as garrafas no caminhão



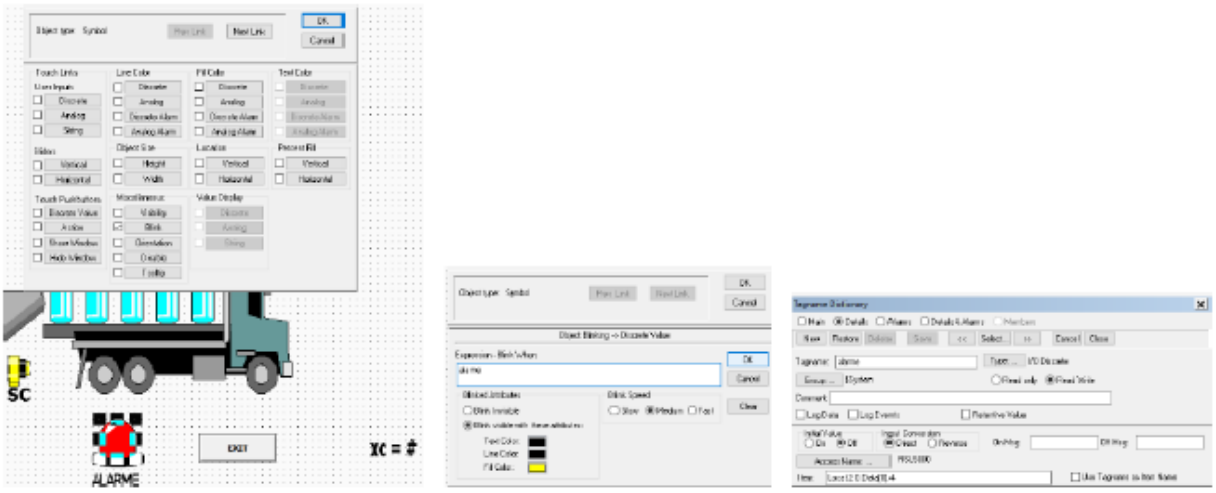
Fonte: Própria.

Considerando que o deslocamento horizontal das garrafas cheias será de acordo com a movimentação do caminhão, que as transportará, é compreensível que se atribua a cada um dos recipientes dispostos sobre o veículo a mesma *tagname* associada ao deslocamento dele.

A visibilidade de cada uma das cinco garrafas, dispostas da esquerda para a direita, é controlada pelas *tagnames* V5, V4, V3, V2 e V1, respectivamente definidas como sendo do tipo Memory Discrete. A primeira garrafa a ser preenchida é depositada junto à extremidade direita da carroceria do caminhão, enquanto o último recipiente a entrar no veículo ficará disposto sobre seu lado esquerdo.

A figura 159 apresenta a propriedade *Miscellaneous Blink*, controlada pela *tagname* alarme, definida com tipo *I/O Discrete* e Item Local:2:O.Data[0].4, que é o endereço da saída do alarme no programa ladder, situada na linha 4, ativada doze segundos após contabilizar 5 garrafas preenchidas com líquido. A ideia consiste em acionar o alarme após todos os recipientes estiverem cheios, para que o motorista se prepare para dirigir o caminhão lotado até o seu destino, situado a 300 pixels de distância, na tela supervisora.

Figura 159 – Configuração de propriedade e *tagname* do alarme

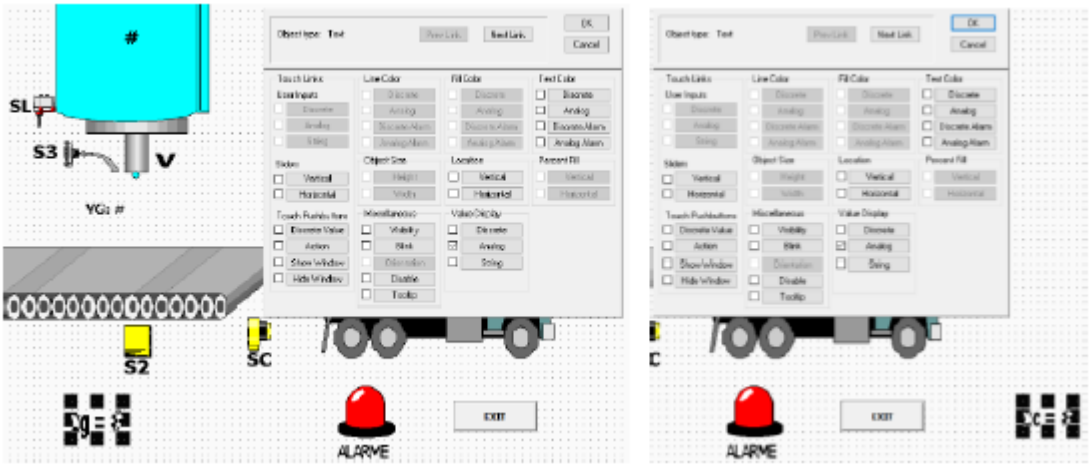


Fonte: Própria.

Quando o alarme for acionado no código Ladder, as definições das propriedades *Miscellaneous Blink* na tela do sistema supervisório alternarão entre a cor de fundo vermelha, e a cor de preenchimento amarela, para enfatizar lotação máxima de carga ao motorista de caminhão.

Conforme figura 160, os textos XG = # e XC = # tem apenas uma propriedade, *Value Display Analog*, controlada pelas *tagnames* XG e XC, para exibir respectivamente os valores do deslocamento da garrafa sobre a esteira, e do caminhão acompanhado de seus 5 recipientes cheios. As variáveis XG e XC já tiveram suas configurações anteriormente exibidas, como sendo do tipo *Memory Integer*.

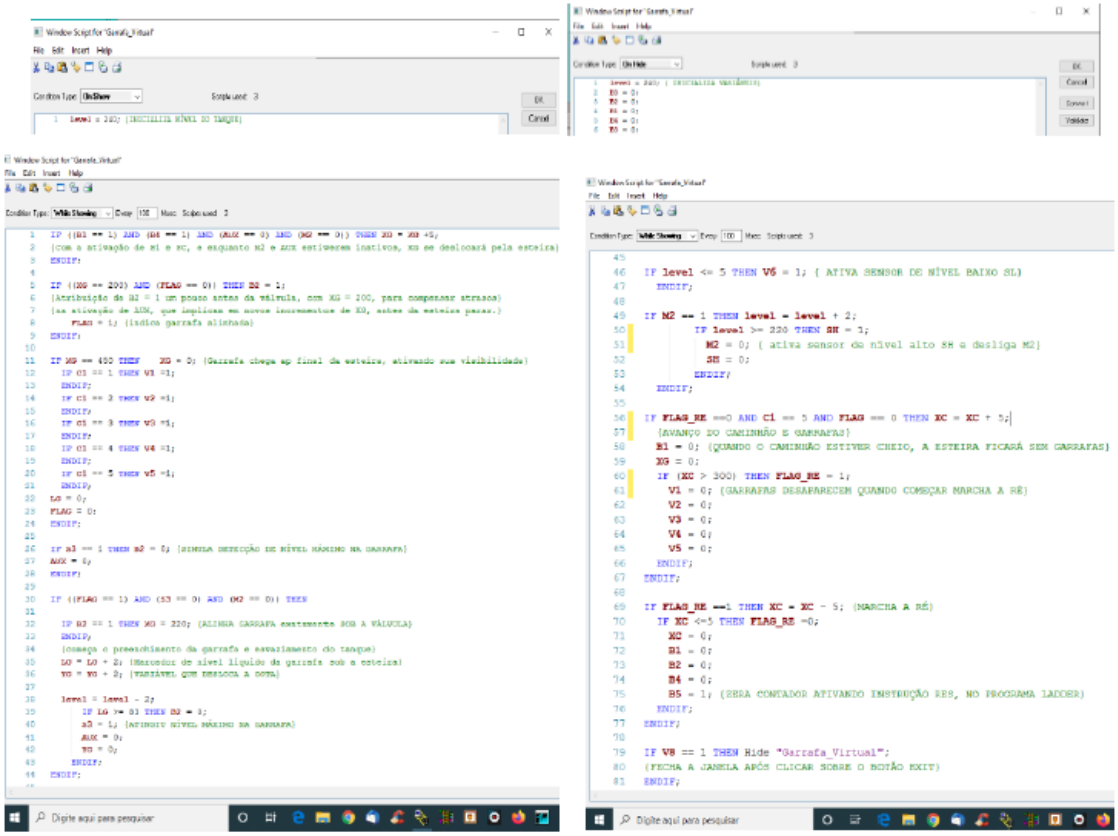
Figura 160 – Propriedades XG e XC, mostradas nos respectivos objetos de texto



Fonte: Própria.

Encerrando este projeto, as telas da figura 161 mostram os scripts que proprocionam a visão dinâmica do processo, através da tela de supervisão, associados às respectivas condições de execução, configuradas para enquanto a tela estiver em exibição ().

Figura 161 – Telas do Script com comentários explicando seu funcionamento



Fonte: Própria.

O código a seguir trará a programação textual do script supervisório executada a cada 100 milissegundos, para a condição enquanto a tela estiver em exibição.

```
IF ((B1 == 1) AND (B4 == 1) AND (AUX == 0) AND (M2 == 0)) THEN XG = XG
+5;
{Com a ativação de S1 e SC, e enquanto M2 e AUX estiverem inativos, XG se
deslocará pela esteira}
ENDIF;

IF ((XG == 200) AND (FLAG == 0)) THEN B2 = 1;
{Atribuição de B2 = 1 um pouco antes da válvula, com XG = 200, para
compensar atrasos}
{na ativação de AUX, que implicam em novos incrementos de XG, antes da
esteira parar.}
   FLAG = 1; {indica garrafa alinhada}
ENDIF;

IF XG == 450 THEN    XG = 0; {Garrafa chega ao final da esteira, ativando sua
visibilidade}
 IF C1 == 1 THEN V1 =1;
 ENDIF;
 IF C1 == 2 THEN V2 =1;
 ENDIF;
 IF C1 == 3 THEN V3 =1;
 ENDIF;
 IF C1 == 4 THEN V4 =1;
 ENDIF;
 IF C1 == 5 THEN V5 =1;
 ENDIF;
LG = 0;
FLAG = 0;
ENDIF;

IF S3 == 1 THEN B2 = 0; {SIMULA DETECÇÃO DE NÍVEL MÁXIMO NA
GARRAFA}
```

```
AUX = 0;
ENDIF;

IF ((FLAG == 1) AND (S3 == 0) AND (M2 == 0)) THEN

 IF B2 == 1 THEN XG = 220; {ALINHA GARRAFA exatamente SOB A
VÁLVULA}
 ENDIF;
 {começa o preenchimento da garrafa e esvaziamento do tanque}
 LG = LG + 2; {Marcador de nível líquido da garrafa sob a esteira}
 YG = YG + 2; {VARIÁVEL QUE DESLOCA A GOTA}

 level = level - 2;
    IF LG >= 83 THEN B2 = 0;
        S3 = 1; {ATINGIU NÍVEL MÁXIMO NA GARRAFA}
        AUX = 0;
        YG = 0;
       ENDIF;
ENDIF;

IF level <= 5 THEN V6 = 1; { ATIVA SENSOR DE NÍVEL BAIXO SL}
 ENDIF;

IF M2 == 1 THEN level = level + 2;
          IF level >= 220 THEN SH = 1;
               M2 = 0; { ativa sensor de nível alto SH e desliga M2}
               SH = 0;
              ENDIF;
 ENDIF;

IF FLAG_RE ==0 AND C1 == 5 AND FLAG == 0 THEN XC = XC + 5;
 {AVANÇO DO CAMINHÃO E GARRAFAS}
 B1 = 0; {QUANDO O CAMINHÃO ESTIVER CHEIO, A ESTEIRA FICARÁ
SEM GARRAFAS}
```

```
  XG = 0;
  IF (XC > 300) THEN FLAG_RE = 1;
        V1 = 0; {GARRAFAS DESAPARECEM QUANDO COMEÇAR MARCHA
A RÉ}
        V2 = 0;
        V3 = 0;
    V4 = 0;
    V5 = 0;
  ENDIF;
ENDIF;

IF FLAG_RE ==1 THEN XC = XC - 5; {MARCHA A RÉ}
  IF XC <=5 THEN FLAG_RE =0;
    XC = 0;
    B1 = 0;
        B2 = 0;
    B4 = 0;
        B5 = 1; {ZERA CONTADOR ATIVANDO INSTRUÇÃO RES, NO
PROGRAMA LADDER}
  ENDIF;
ENDIF;

IF V8 == 1 THEN Hide "Garrafa_Virtual";
{FECHA A JANELA APÓS CLICAR SOBRE O BOTÃO EXIT}
ENDIF;
```

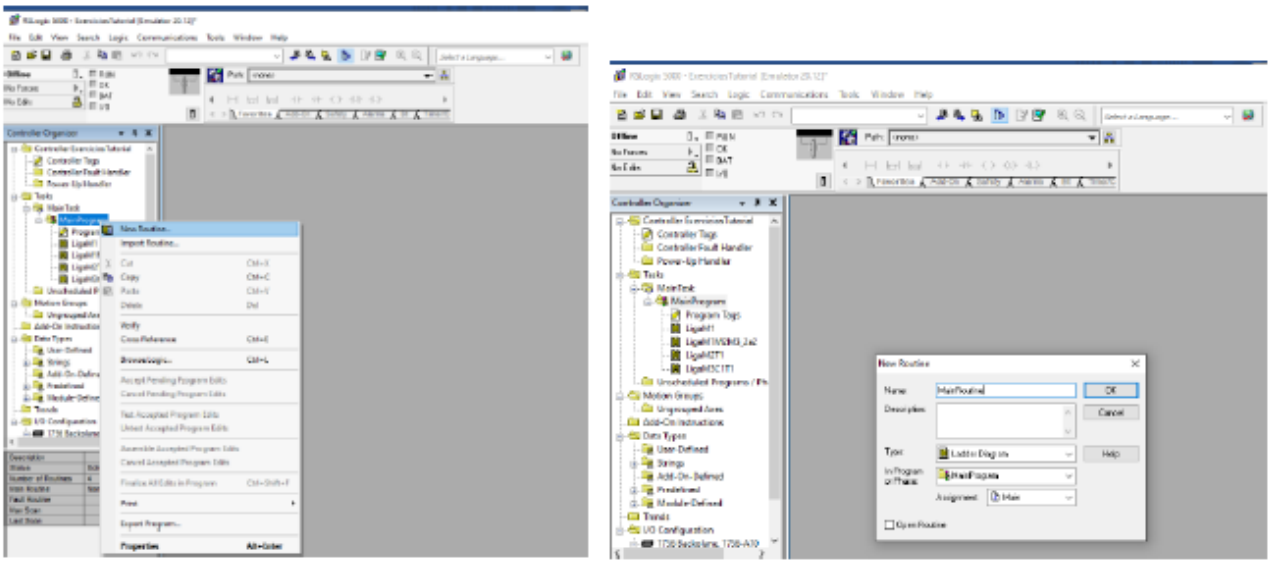
A próxima seção abordará a programação na linguagem diagrama de blocos.

3.3 PROBLEMAS DE AUTOMAÇÃO EM DIAGRAMA DE BLOCOS DE FUNÇÕES

A primeira etapa consiste em configurar o RSLogix 5000 emulate, criar o driver de comunicação no RSLinx e iniciar novo projeto no RSLogix 5000, conforme visto nas páginas anteriores.

A partir daqui, é preciso criar, como rotina principal, um código em linguagem Ladder, que realize um salto para o programa desejado, como na figura a seguir. O código pode ser criado clicando-se com o botão direito sobre a aba Main Program e, escolhendo a opção New routine (figura 162), originado a segunda ilustração da figura, onde se atribui o nome MainRoutine, caracterizada por ser um diagrama Ladder, com a função de programa principal.

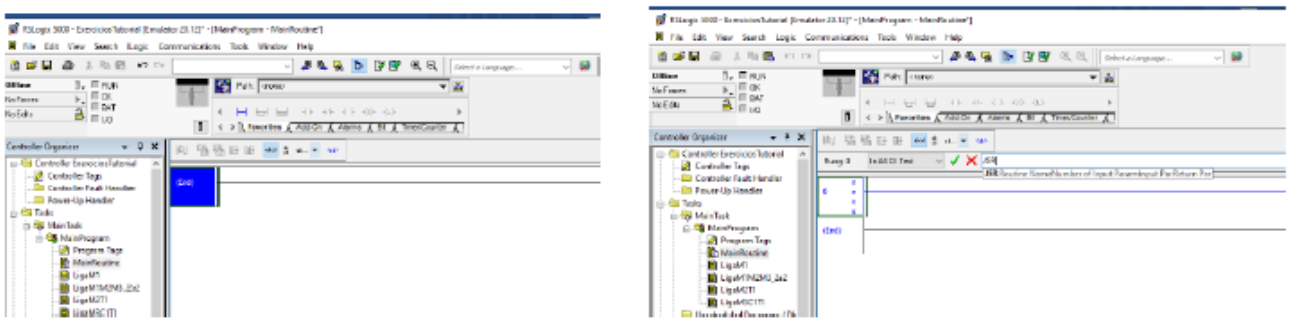
Figura 162 – Nomeando a rotina principal



Fonte: Própria

Após confirmar a criação da rotina pressionando o botão OK, surgirá um diagrama ladder, correspondente à primeira ilustração da figura 163. Clicando-se duas vezes sobre a linha inicial do programa proporcionará o aparecimento da segunda ilustração, onde se percebe uma linha para inserir instrução na linha 0, na qual se digita JSR.

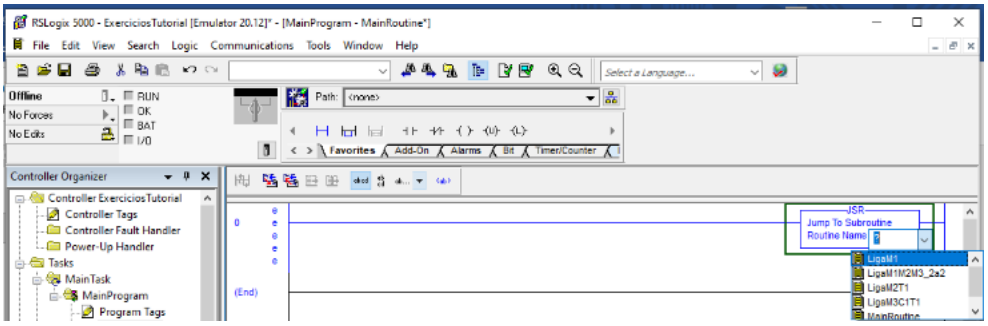
Figura 163 – Iniciando programação do diagrama Ladder



Fonte: Própria

Após teclar enter na segunda ilustração surgirá a figura 164, com a instrução JSR adicionada, possibilitando selecionar no argumento *Routine Name* o nome da rotina para a qual o fluxo de execução será desviado.

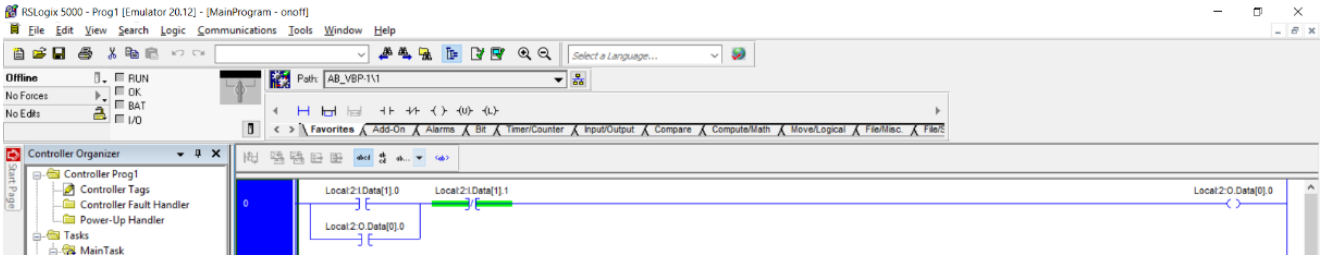
Figura 164 – Adicionando Instrução JSR



Fonte: Própria

A figura 165 a seguir apresenta um programa em linguagem ladder, para ligar ou desligar um motor. O objetivo consiste em utilizar o código como plataforma de apoio à escrita de programas em outras linguagens suportadas pelo ambiente de desenvolvimento integrado RSLogix 5000, que realizem as mesmas funções, ou seja, acionar ou desativar uma máquina.

Figura 165 – Programa Ladder para acionar motor



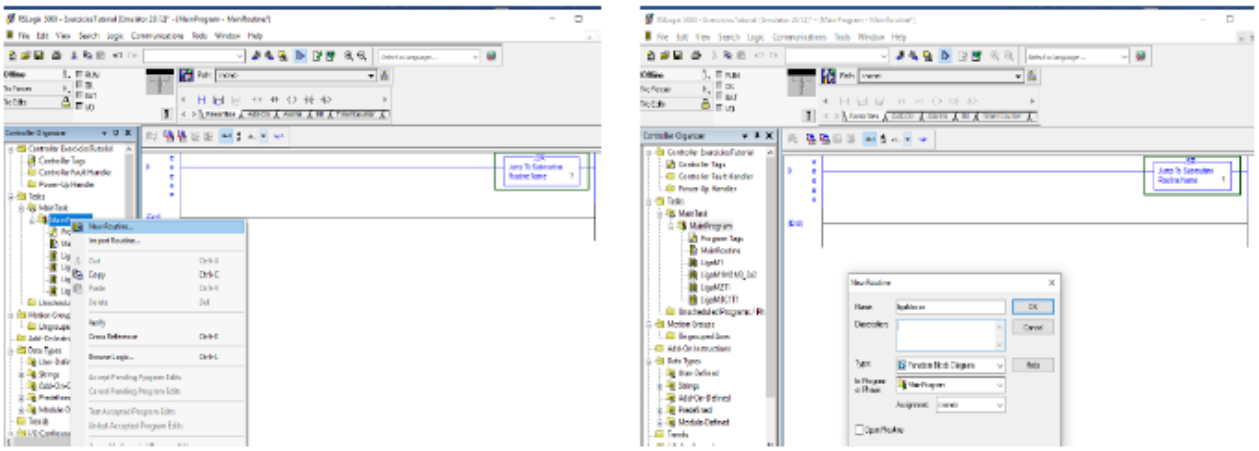
Fonte: Própria

EXERCÍCIOS

1) Desenvolver rotina em diagrama de blocos de funções (*FBD*) que possibilite ligar ou desligar um motor.

Para elaborar um programa equivalente em linguagem de blocos, que controle o acionamento de um motor, clica-se com o botão direito sobre *Main Program* e, na tela que surgir (figura 166), marca-se a opção New Routine, originando a segunda ilustração da figura, onde se atribui à rotina criada o nome ligablocos e o tipo *Function Block Diagram*.

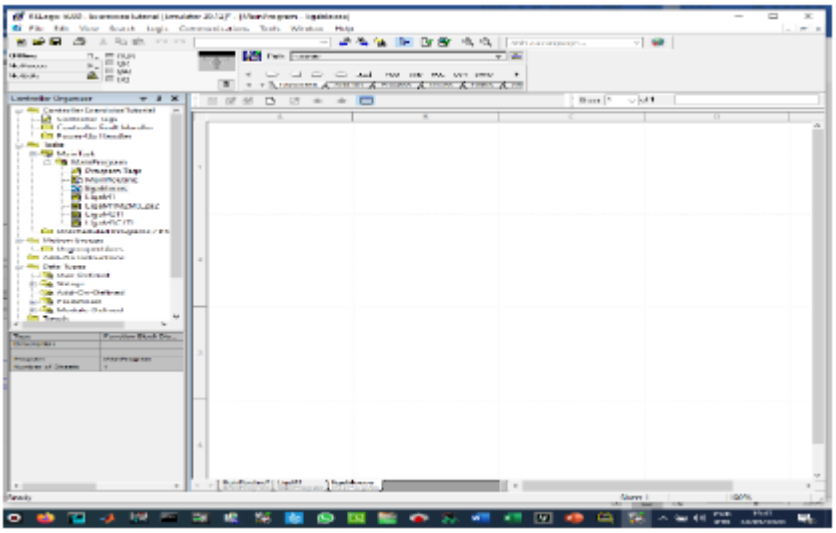
Figura 166 – Iniciando programação em diagrama de blocos de funções



Fonte: Própria

Após confirmar a criação da rotina, clica-se duas vezes sobre ela, nomeada ligablocos, para obter tela vista na figura 167

Figura 167 – Área de programação em diagrama de blocos de funções (FBD).

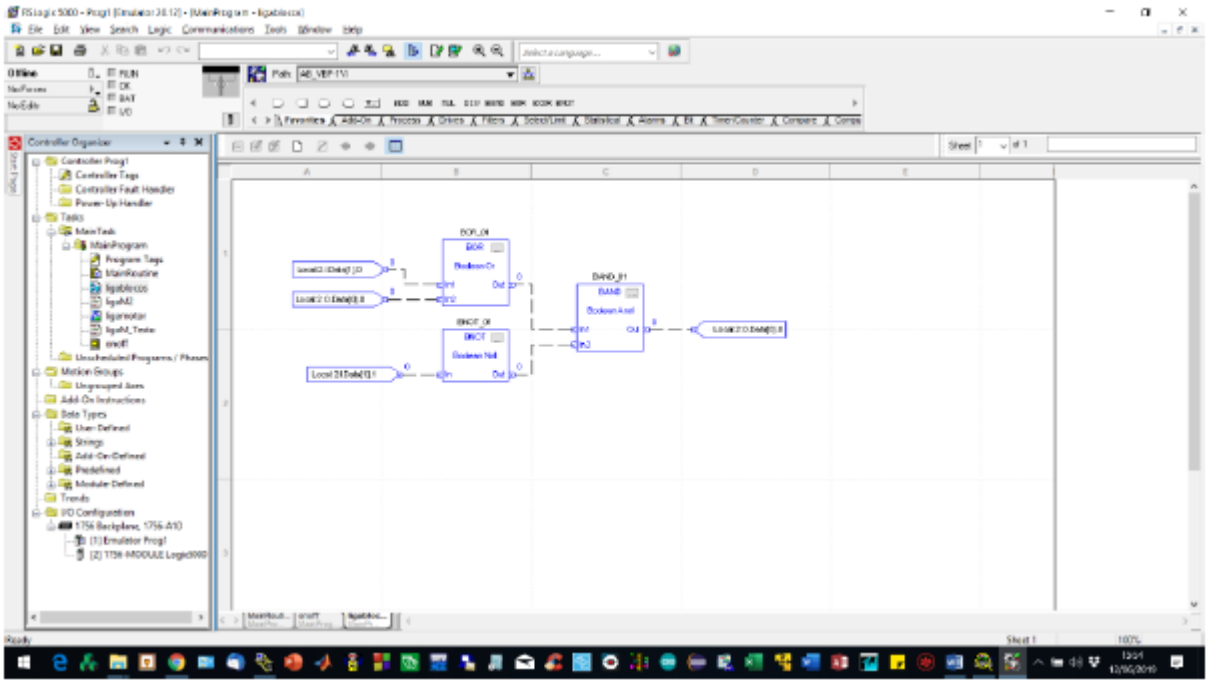


Fonte: Própria

A programação em linguagem de blocos é baseada em funções lógicas. Dessa forma, para reescrever o código ladder para controle do acionamento motor em linguagem de blocos funcionais (FBD), basta aplicar analogia apresentada em capítulos anteriores, onde se representa associação em paralelo de contatos virtuais pela função lógica OU (BOR). Analogamente, contatos virtuais interligados em série são representados pela função lógica E (BAND). Da mesma forma, contatos normalmente fechados são representados por meio de seus estados lógicos negados, através da função lógica NÃO (BNOT).

Segue código em linguagem de blocos que aciona um motor, conforme figura 168.

Figura 168 – Acionamento de motor em FBD.

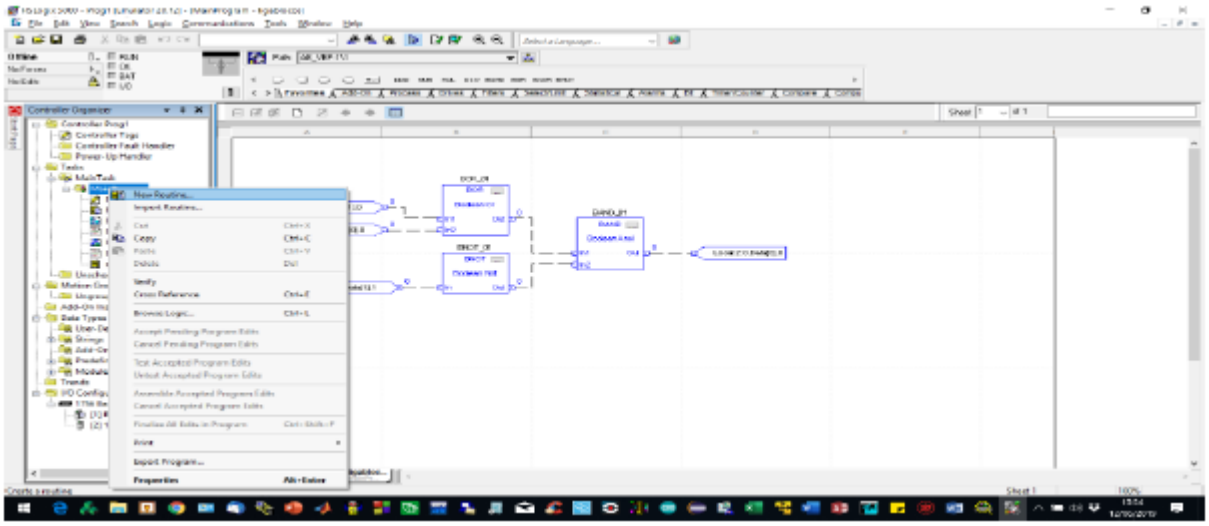


Fonte: Própria

O equivalente lógico do controle de acionamento motor em linguagem ladder é uma função definida por Y = ((ON1)OR(M1))AND(NOT(OFF1)) , onde ON1 é o contato virtual cujo endereço é o mesmo do botão liga (Local:2:I.Data[1].0), onde OFF1 é o contato virtual cujo endereço é o mesmo do botão desliga (Local:2:I.Data[1].1) e M1 representa a bobina de saída (endereçada por Local:2:O.Data[0].0) que controla o acionamento do motor, quando a lógica do programa ladder proporcionar circulação de corrente virtual até ela.

Para desenvolver por etapas um código em linguagem de blocos que controla o motor M1, cria-se nova rotina para acionar ou desligar um motor, conforme figura 169

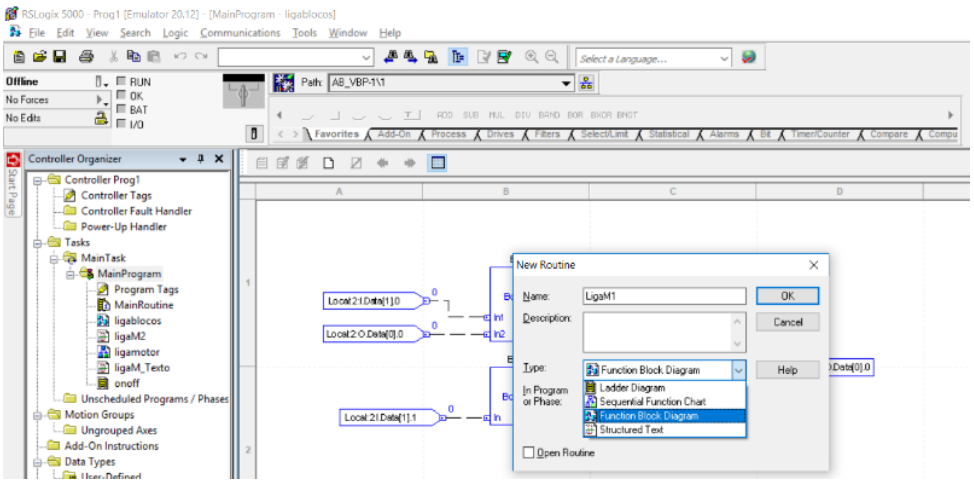
Figura 169 – Criação de nova rotina em FBD.



Fonte: Própria

A próxima etapa consiste em atribuir ao programa especificado o tipo diagrama de blocos de funções, o qual recebe o nome LigaM1, conforme figura 170.

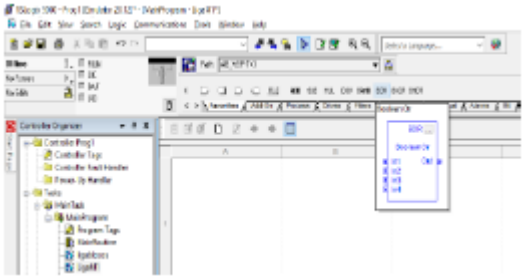
Figura 170 – Nomeando rotina em FBD



Fonte: Própria

Abre-se o programa LigaM1, selecionando-se o botão esquerdo do mouse o bloco BOR, para inserir no código as entradas referentes ao botão liga e ao contato selo, e implementar a associação em paralelo de ambas (fig. 171).

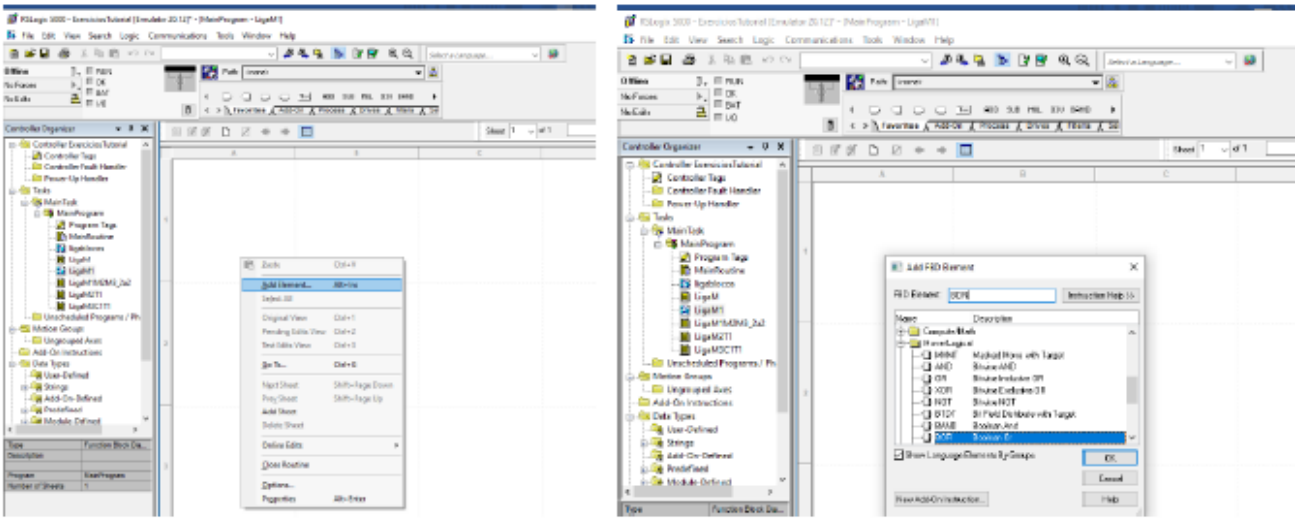
Figura 171 – Inserindo instrução BOR



Fonte: Própria

A figura 172 apresenta outra maneira de incluir uma instrução FBD, obtida clicando-se com o botão direito na área de programação, de acordo com sua primeira ilustração.

Figura 172 – Maneira alternativa de inserir instrução em FBD

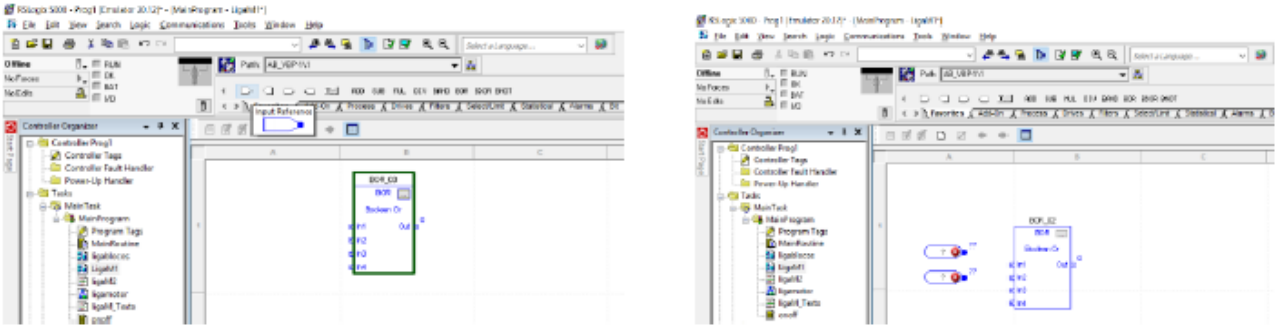


Fonte: Própria

A adição do elemento FBD conforme figura proporcionará condições ao surgimento de sua segunda ilustração, onde pode ser selecionada a função que se deseja inserir na folha de programação, realizando a confirmação do procedimento após pressionar o botão OK.

Após selecionar a função *BOR*, é necessário escolher entradas de referência, blocos que possibilitam inserir os endereços de entrada e saída necessários à operação lógica, os quais serão posteriormente interligados às entradas das portas através de conectores, conforme figura 173, onde se inserem duas funções *input reference*.

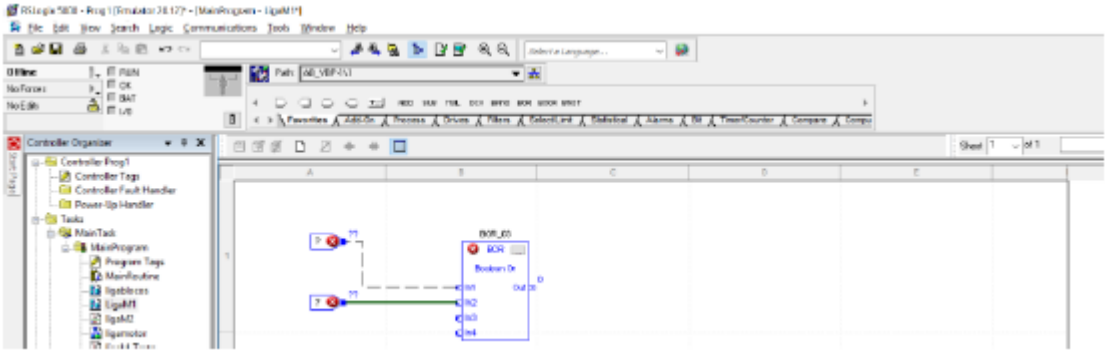
Figura 173 – Inserindo duas entradas de referência



Fonte: Própria

A próxima ação consiste em associar as duas entradas de referência ao bloco BOR, ligando suas saídas às respectivas entradas lógicas funcionais, conforme figura 174.

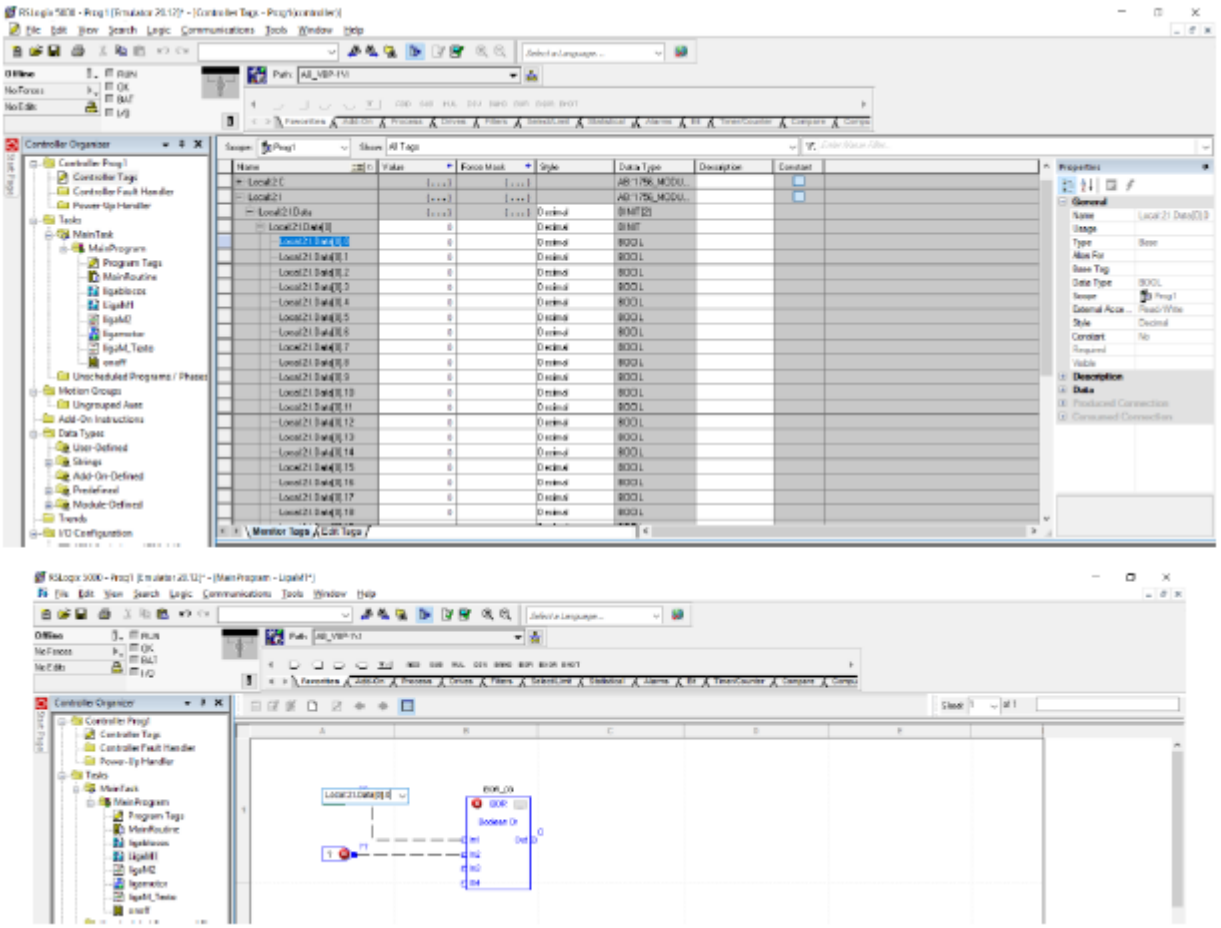
Figura 174 – Associando as entradas de referência ao bloco BOR



Fonte: Própria

Resta agora inserir os endereços do botão liga e do contato selo no interior das entradas de referência. Para tanto, clica-se duas vezes sobre controller tags, expandindo-se os endereços de entrada, com o intuito de copiar o primeiro deles, e colá-lo numa das entradas de referência do programa em linguagem de blocos, de acordo com os dois quadros da figura 175.

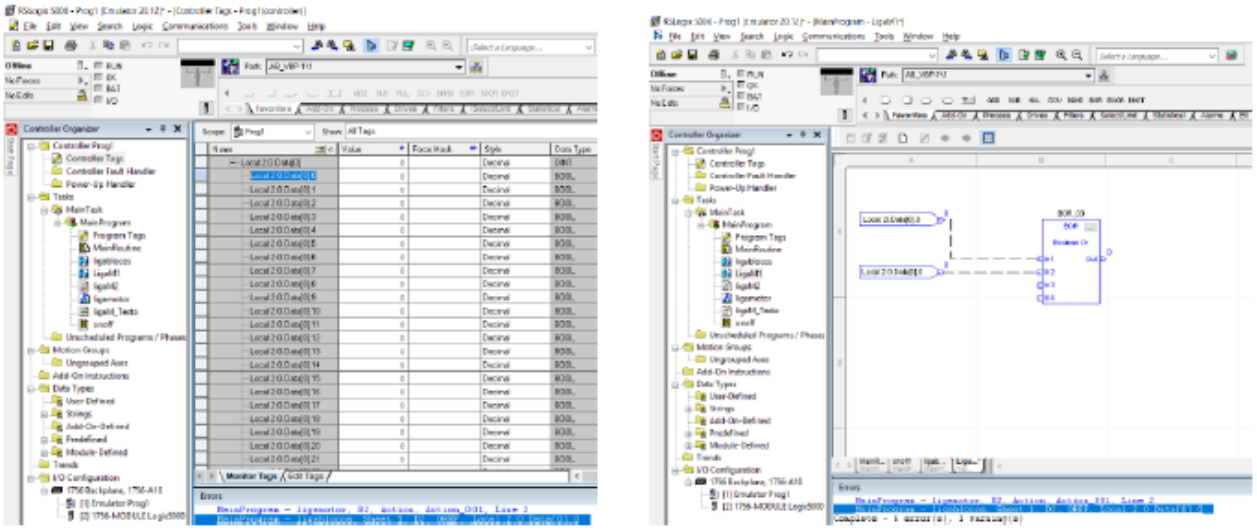
Figura 175 – Inserindo endereços dos botões liga e contato selo



Fonte: Própria

A partir deste ponto copia-se o endereço da saída, inserindo-o na segunda entrada de referência, conforme figura 176.

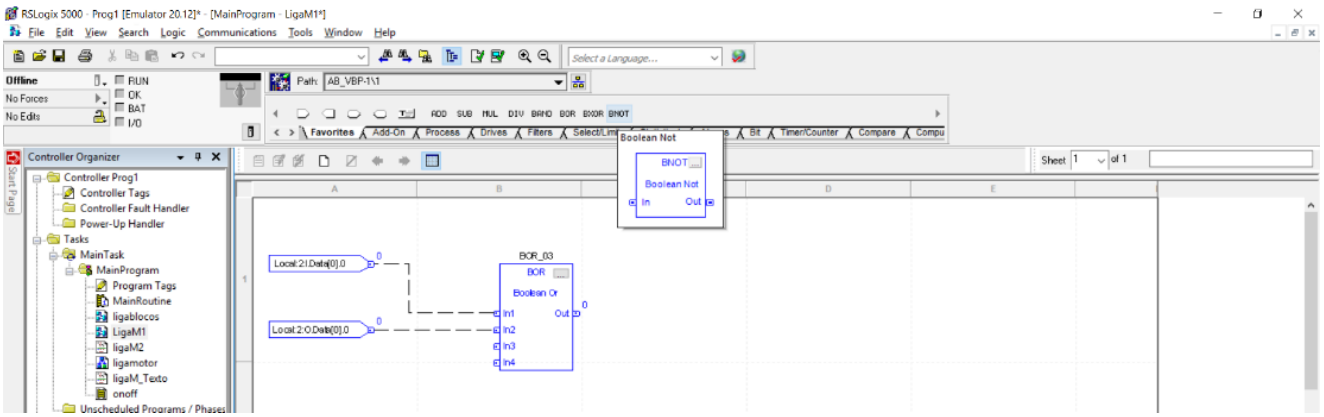
Figura 176 – Inserindo endereço da saída na entrada de referência do contato selo



Fonte: Própria

O próximo passo consiste em inserir um bloco lógico BNOT (boolean not), juntamente com sua entrada de referência, corretamente endereçada, conforme figura 177.

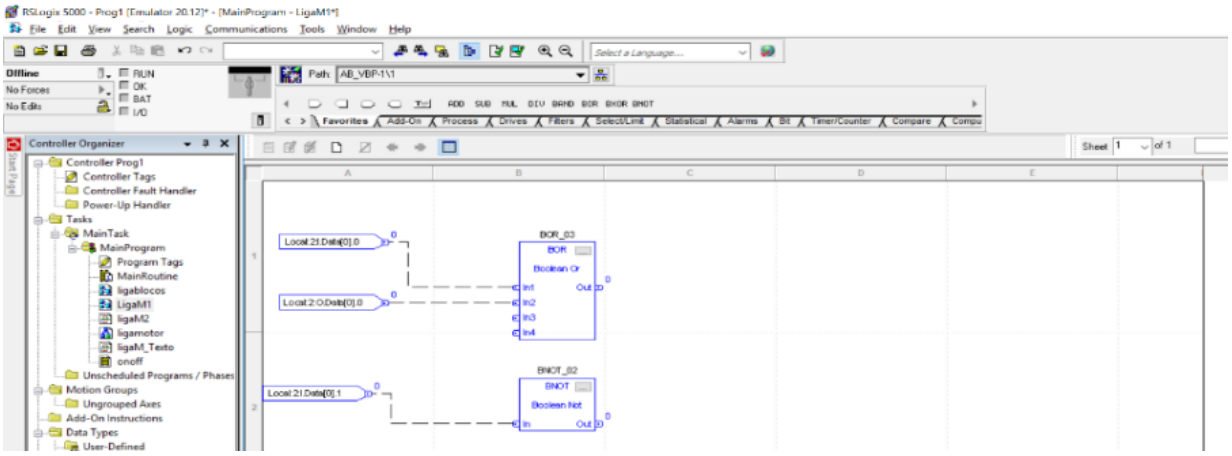
Figura 177 – Inserindo bloco BNOT e entrada de referência



Fonte: Própria

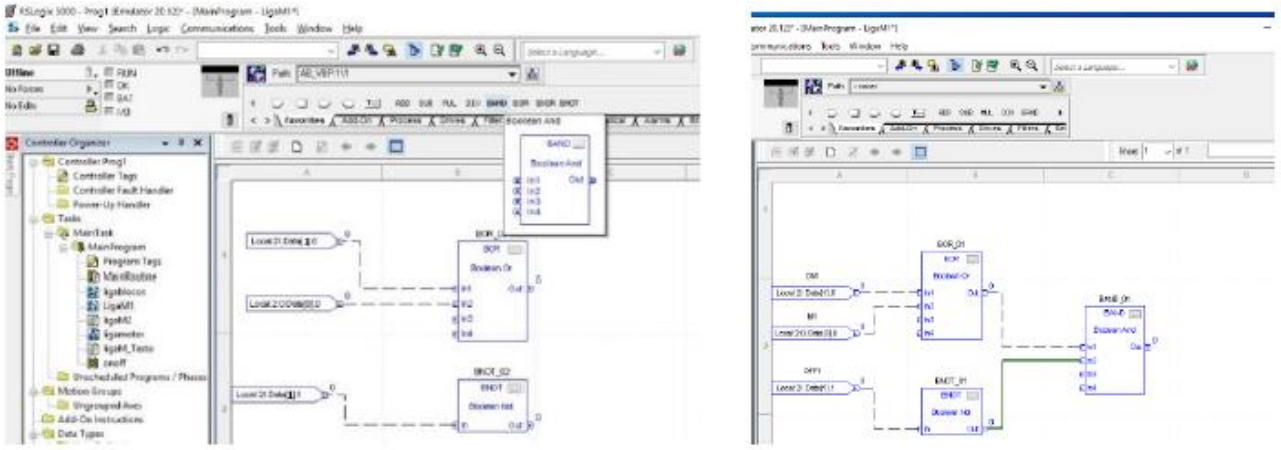
Em seguida se insere um bloco entrada de referência, atribuindo ao mesmo o endereço do botão desliga, para conectar sua saída à instrução BNOT (fig. 178), com o intuito de obter o equivalente ao contato virtual cujo estado de repouso é normalmento fechado (NF), cuja função consiste em desativar a bobina de saída através da interrupção do fluxo de corrente virtual até ela.

Figura 178 – Conectando entrada de referência a instrução BNOT



Fonte: Própria

Figura 179 – Inserindo instrução BAND

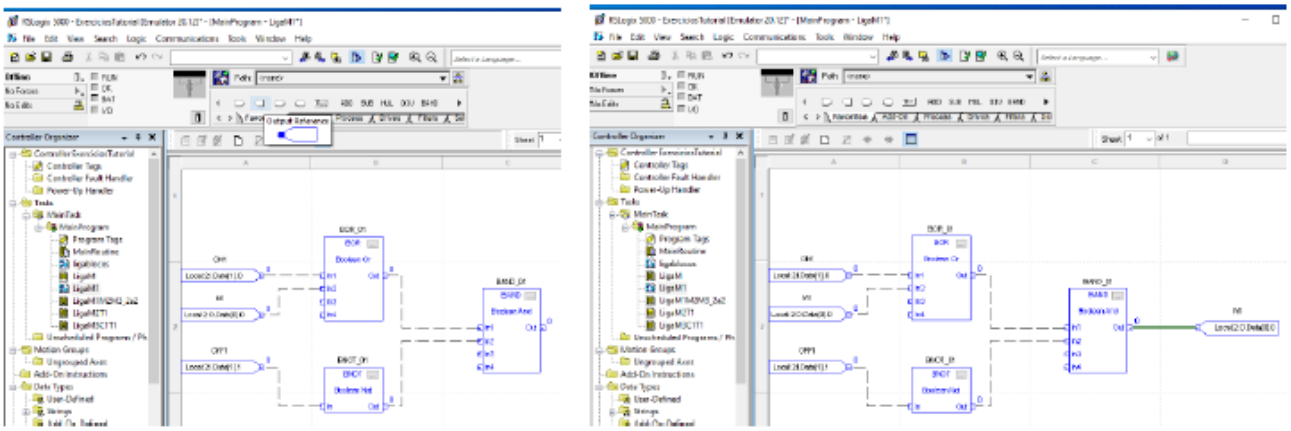


Fonte: Própria

A próxima etapa consiste na conexão em série do resultado das operações lógicas *BOR* (boolean *OR*) e *BNOT* (boolean *NOT*) para realizar, em blocos de funções, o equivalente à equação de saída do programa ladder, conforme figura. Uma associação em série requer a porta lógica *AND* (*BAND* – boolean and) inserida na folha de programação (Fig. 179).

Para finalizar a programação *FBD* deste problema, é necessário alimentar a função *Output reference* (que é o equivalente à instrução bobina de saída, em linguagem ladder) com o resultado da lógica que realiza a automação do controle motor. Dessa forma, o bloco referência de saída receberá o endereço da mesma, além de ser conectado à saída da porta *BAND* de acordo com a figura 180.

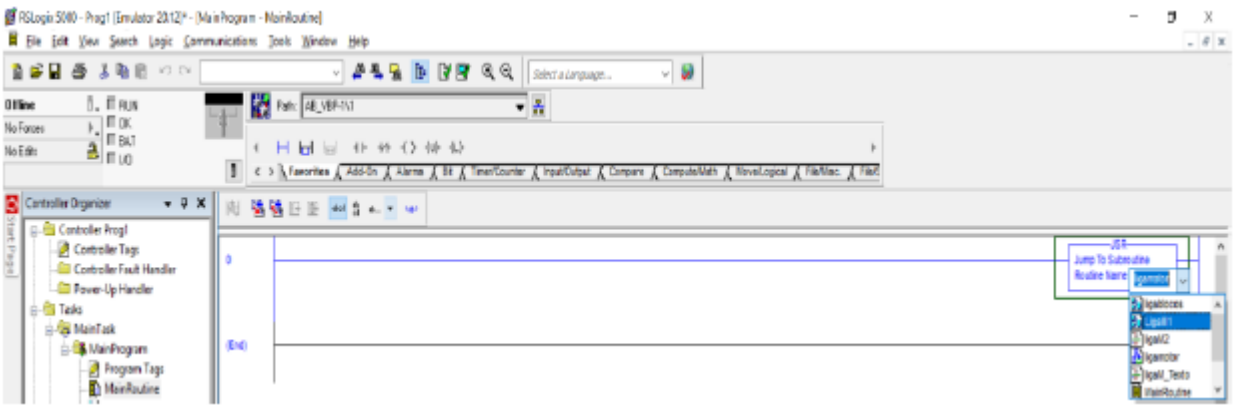
Figura 180 – Conectando Instrução saída de referência ao bloco BAND



Fonte: Própria

Para colocar em execução o programa finalizado, é preciso abrir o programa principal e, na instrução JSR, especificar como nome da rotina o código em linguagem de blocos, que é LigaM1, conforme figura 181.

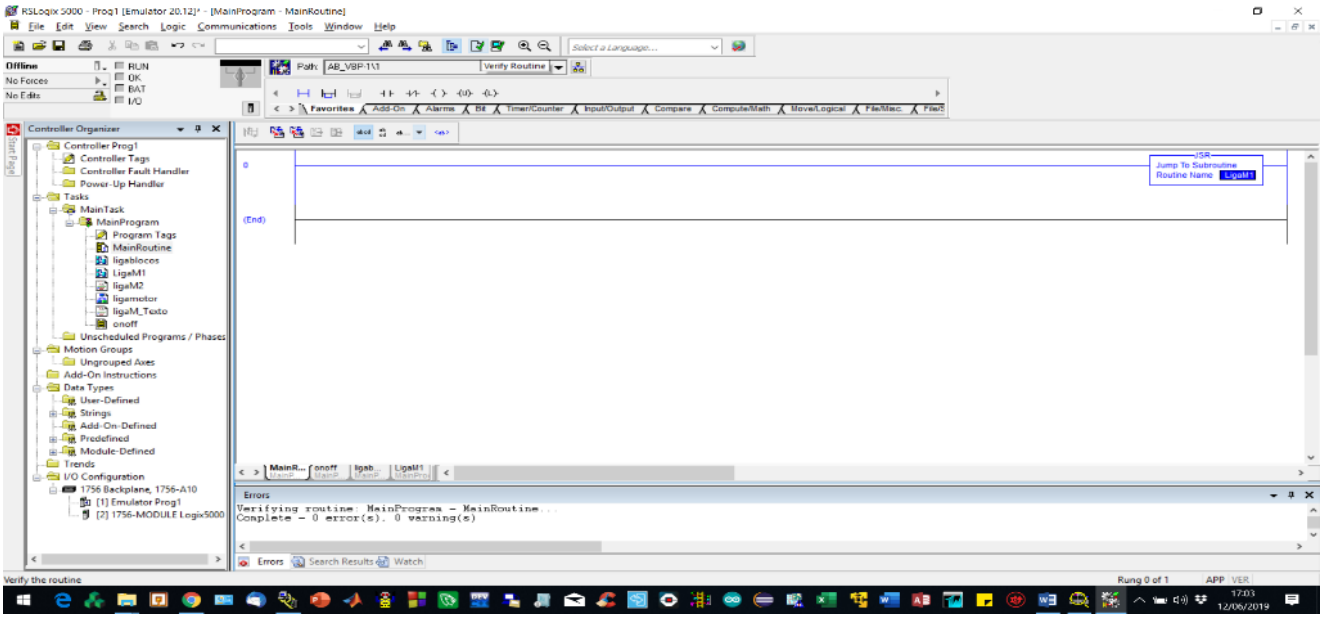
Figura 181 – Seleção do nome do programaa ser executado na instrução JSR



Fonte: Própria

Neste ponto é preciso verificar a integridade do código elaborado, o que é feito procedendo-se conforme figura 182, na qual se clica em Verify routine.

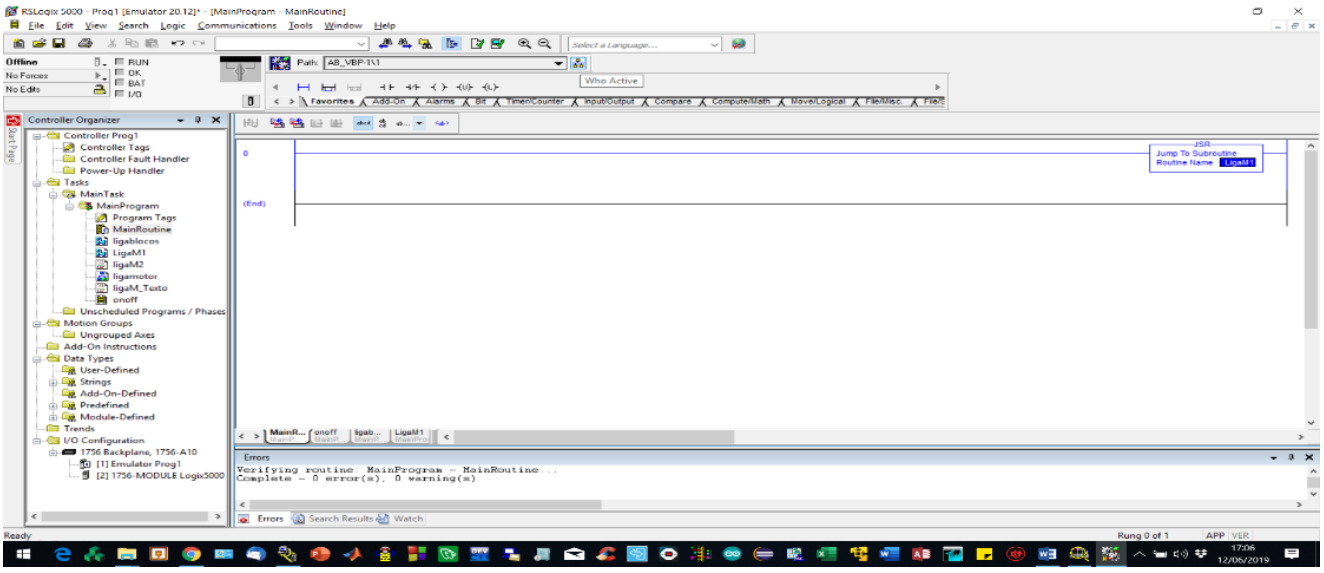
Figura 182 – Verificando integridade da rotina



Fonte: Própria

A seguir deve ser pressionado o botão Who Active, visto na figura 183.

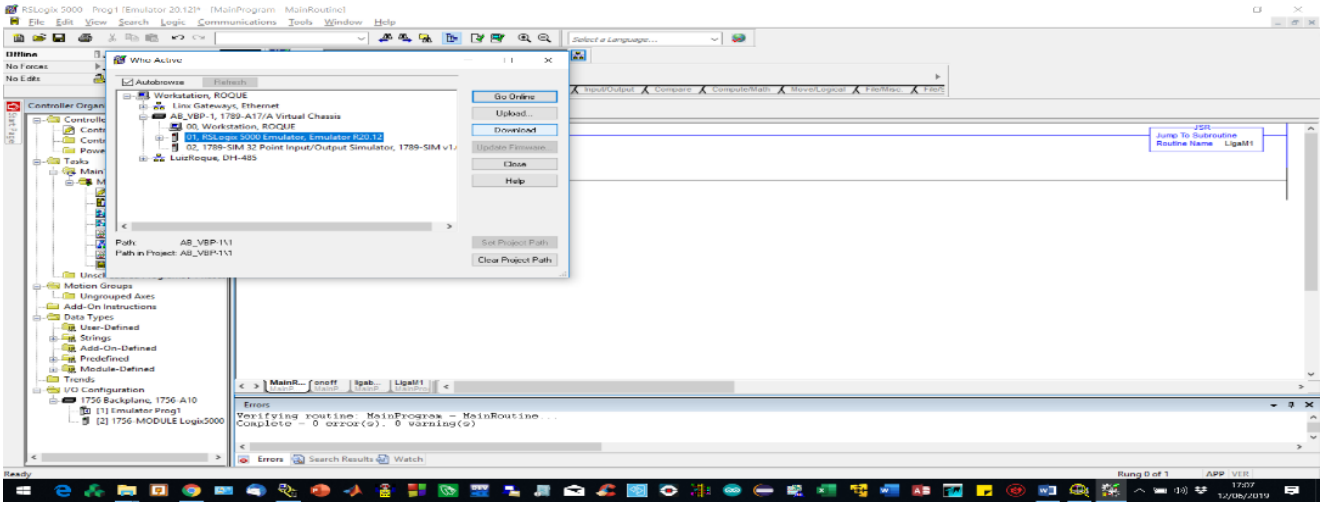
Figura 183 – Pressionando botão Who Active



Fonte: Própria

A supracitada ação originará a seguinte tela (fig. 184), na qual se efetua o download, após pressionar seu respectivo botão.

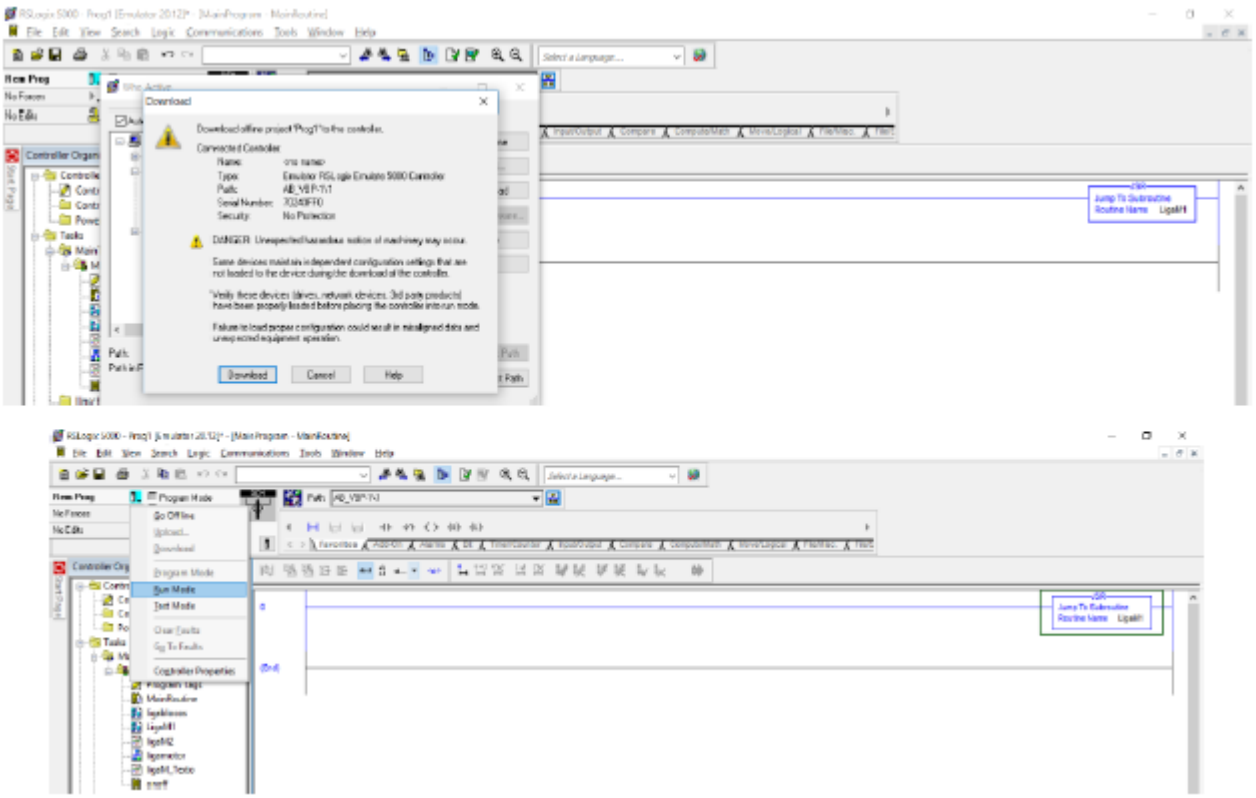
Figura 184 – Realizando Download



Fonte: Própria

Com isso surgirá a primeira ilustração da figura 185, onde surge a oportunidade de confirmar o download.

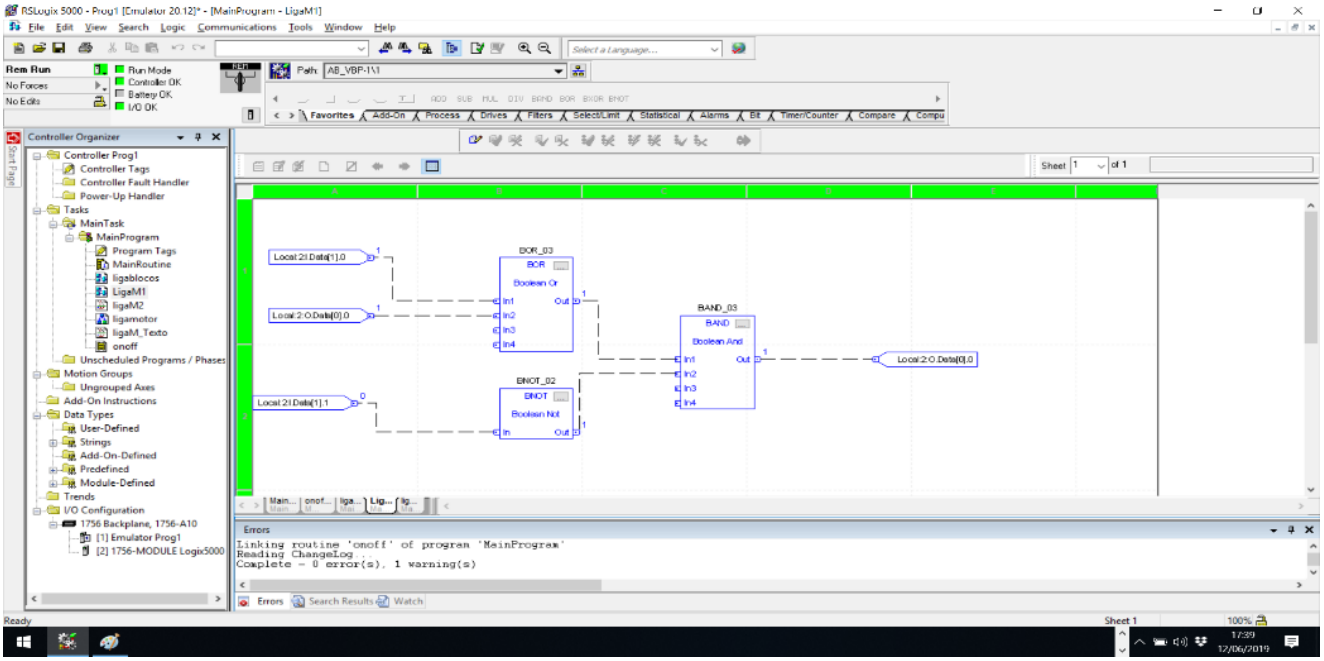
Figura 185 – Tela de Download



Fonte: Própria

Clica-se sobre *Run Mode* no canto superior esquerdo da tela, marcado pela cor azul clara, ao lado da mensagem **Rem Prog,** de acordo a segunda ilustração da figura 185. Dessa forma, torna-se possível abrir o programa em linguagem de blocos, clicando-se duas vezes sobre LigaM1 (fig. 186).

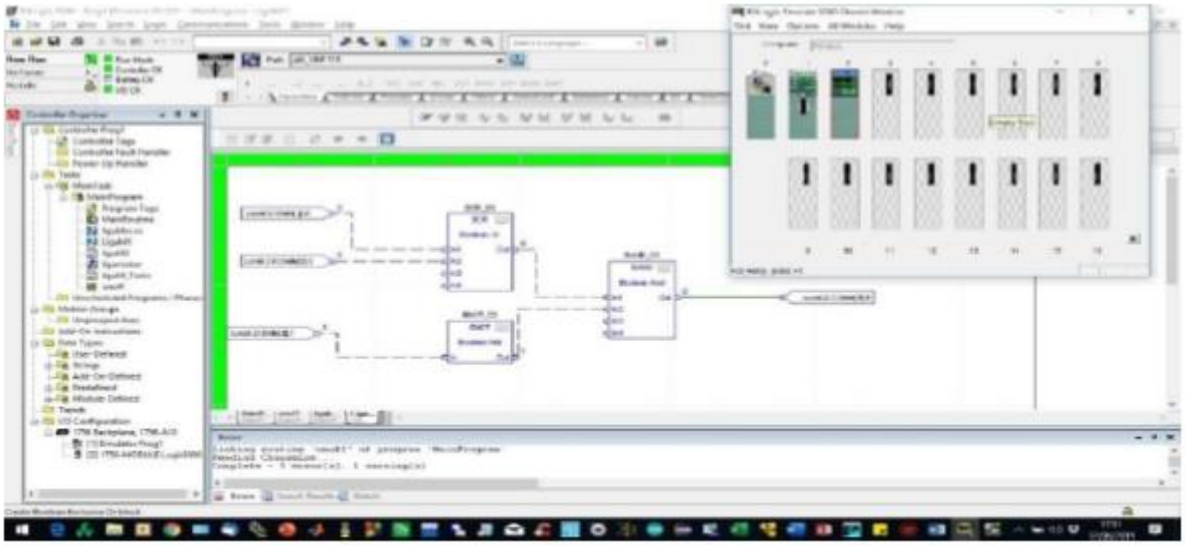
Figura 186 – Programa FBD para acionar motor em execução



Fonte: Própria

A seguir é preciso tornar o RSLogix 5000 emulate visível, para controlar os botões de entrada do programa LigaM1, elaborado em FBD, de acordo com a figura 187.

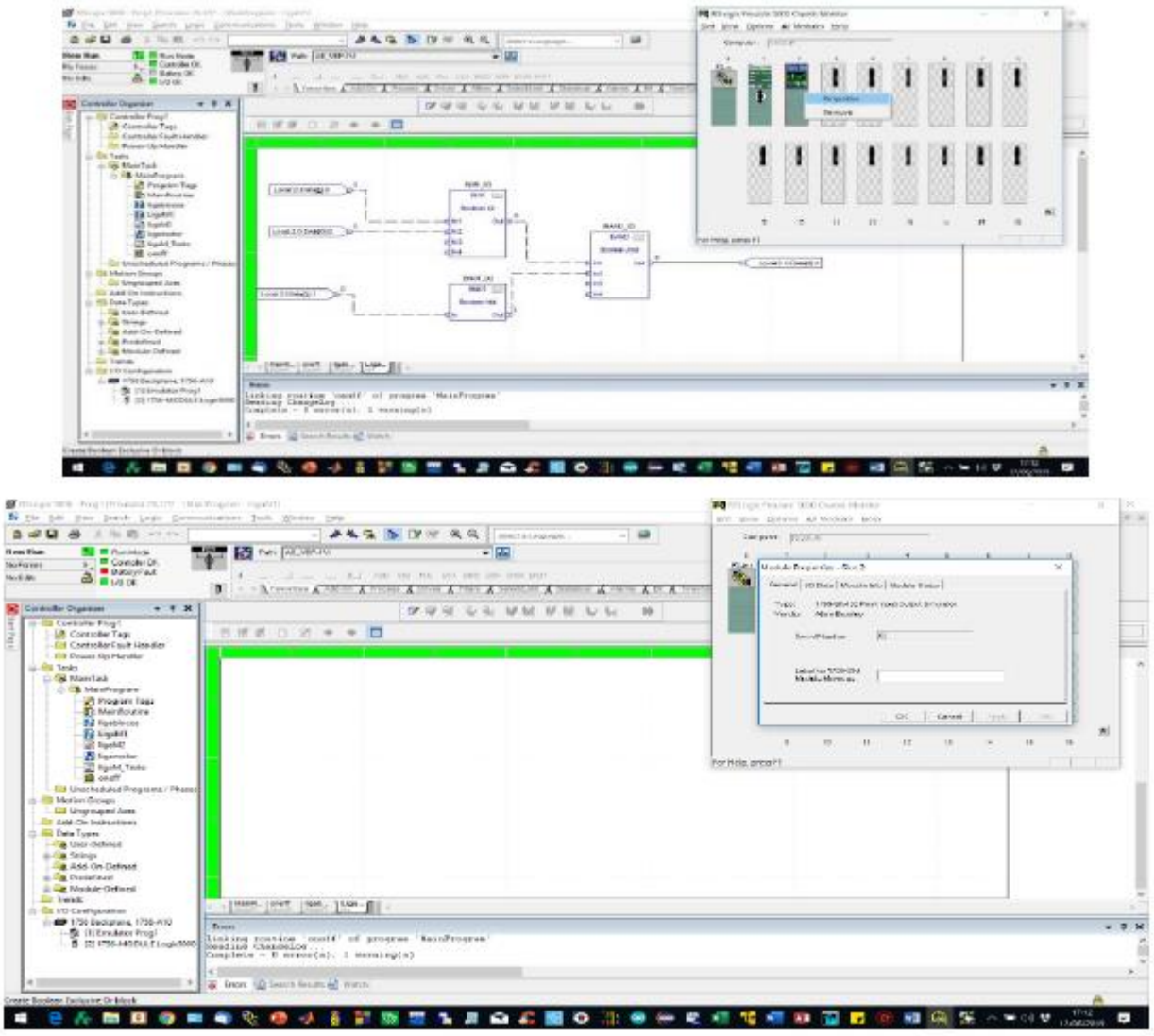
Figura 187 – Acionamento do programa FBD através do RSLogix Emulate 5000



Fonte: Própria

Clica-se com o botão direito sobre slot 2, para acessar a opção properties, segundo conteúdo apresentado na primeira ilustração da figura 188. Isso proporcionará o surgimento da segunda ilustração da figura 188.

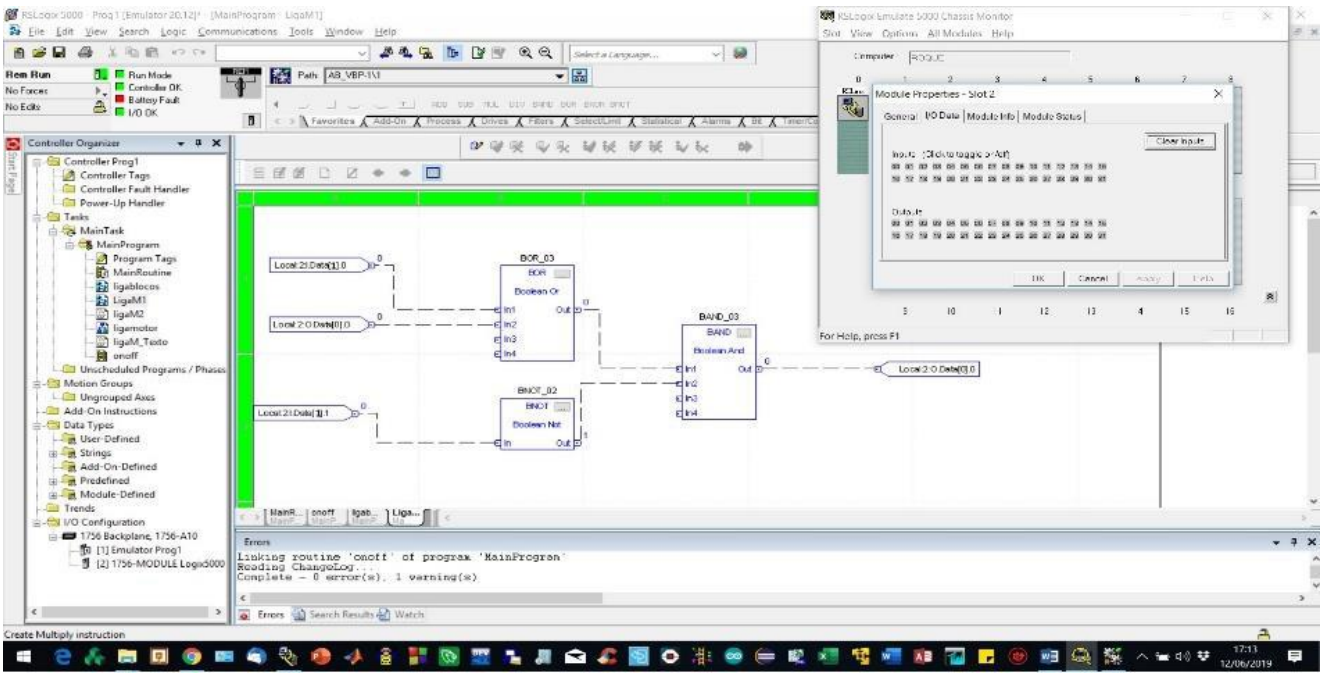
Figura 188 – Configurando o acionamento do FBD através do Emulate 5000



Fonte: Própria

Seleciona-se a aba I/O Data, originando a figura 189, onde é possível visualizar a disposição de entradas e saídas.

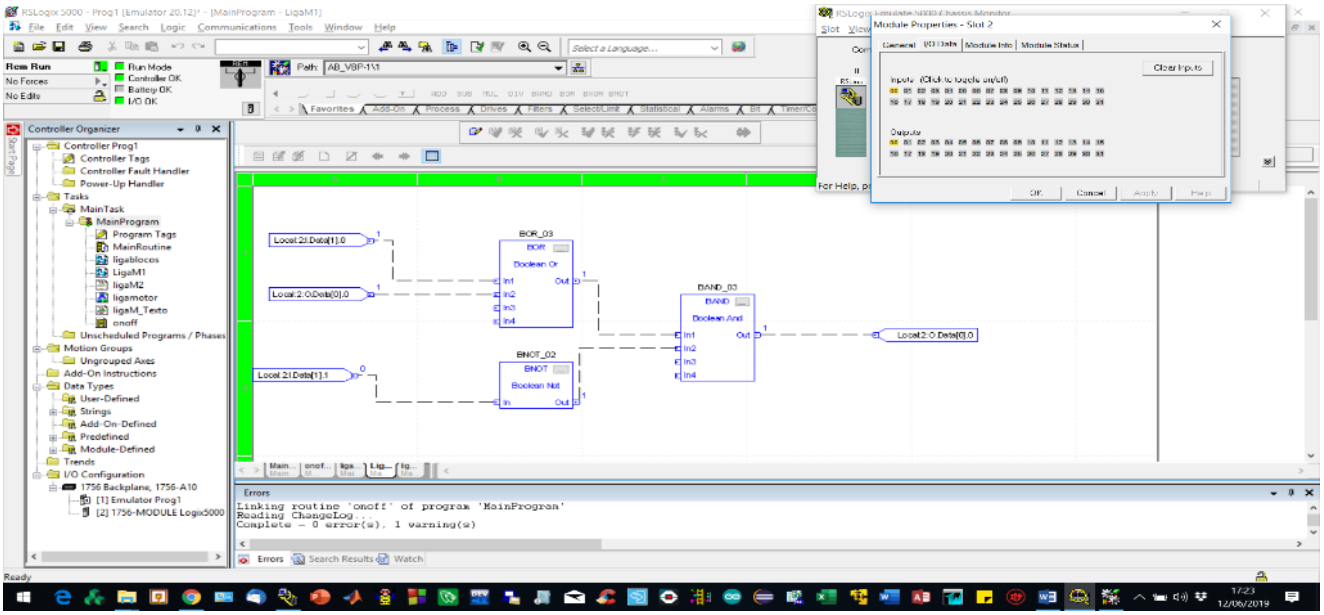
Figura 189 – Visualizando disposição de entradas e saídas



Fonte: Própria

Simula-se o pressionar do botão liga apertando na entrada 00, endereçada por Local:2:I.Data[1].0, o que provocará o acionamento da saída 00, cujo endereço é Local:2:O.Data[0].0, de acordo com a figura 190.

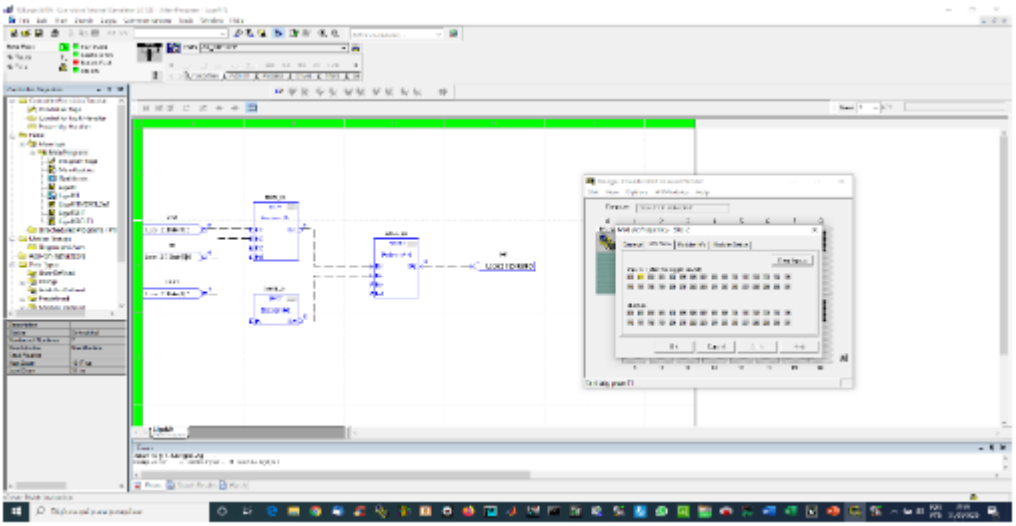
Figura 190 – Acionando o botão liga pelo RSLogix Emulate 5000



Fonte: Própria

Para desativar o motor, é preciso desacionar o botão liga, clicando novamente sobre a entrada 00, desativando a luz amarela que indica o acionamento da mesma. Percebe-se que o selo representado pelo contato normalmente aberto (NA) endereçado pela saída (e associado em paralelo ao botão liga) mantém o motor acionado, mesmo após a entrada 00 retornar ao estado de repouso. Em seguida, basta pressionar sobre o botão desliga apertando na entrada 01, endereçada por Local:2:I.Data[1].1, conforme figura para interromper a circulação de corrente virtual, devido à comutação do estado normalmente fechado (NF) para aberto, conforme figura 191.

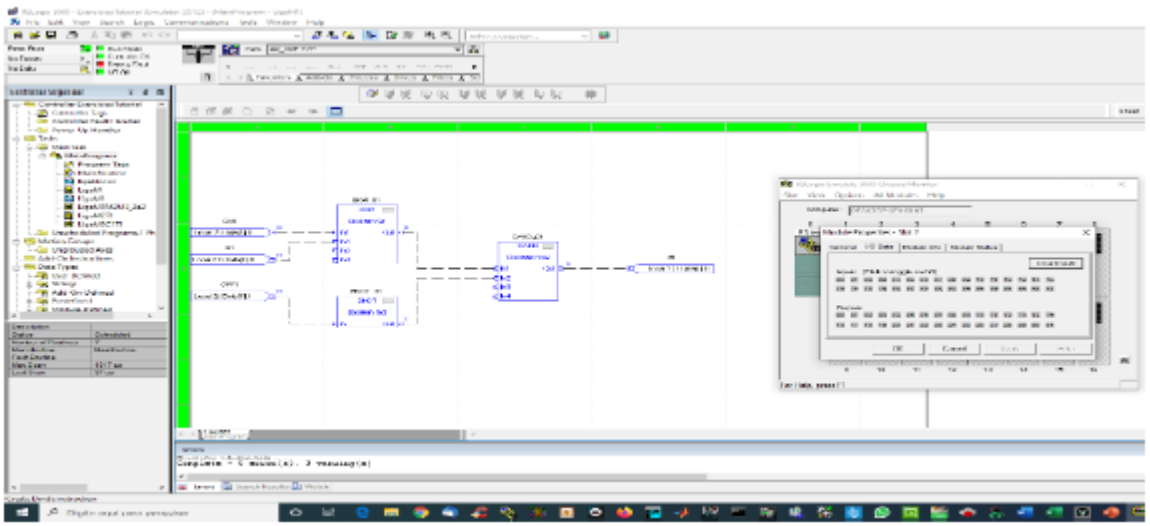
Figura 191 – Acionando o botão liga pelo RSLogix Emulate 5000



Fonte: Própria

Para acionar novamente o motor, é preciso proporcionar condições para o retorno da circulação de corrente virtual até a saída, o que ocorre pressionando novamente o botão da entrada 01 (condição verificada pelo apagar da luz a ela associada), para que ela retorne ao estado de repouso NF, viabilizando a operação da máquina, quando o botão liga for acionado (fig. 192).

Figura 192 – Habiliatando o programa para novo acionamento

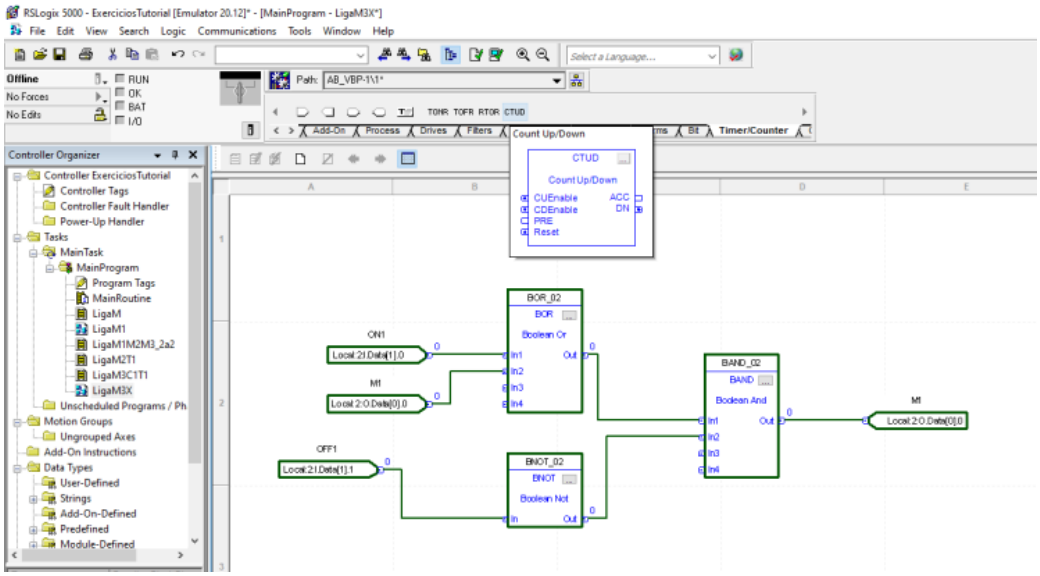


Fonte: Própria

2) solicita-se elaborar programa em diagrama de blocos que possibilite o controle do acionamento de um motor, desligando-o automaticamente após entrar em operação por duas vezes consecutivas. Dessa forma, a máquina será desligada pelo código FBD quando o botão liga for pressionado por três vezes.

O código básico é análogo ao anterior, possibilitando ligar ou desligar o motor. A diferença consiste na inserção de bloco contador CTUD, obtido na aba Timer/Counter, conforme figura 192.

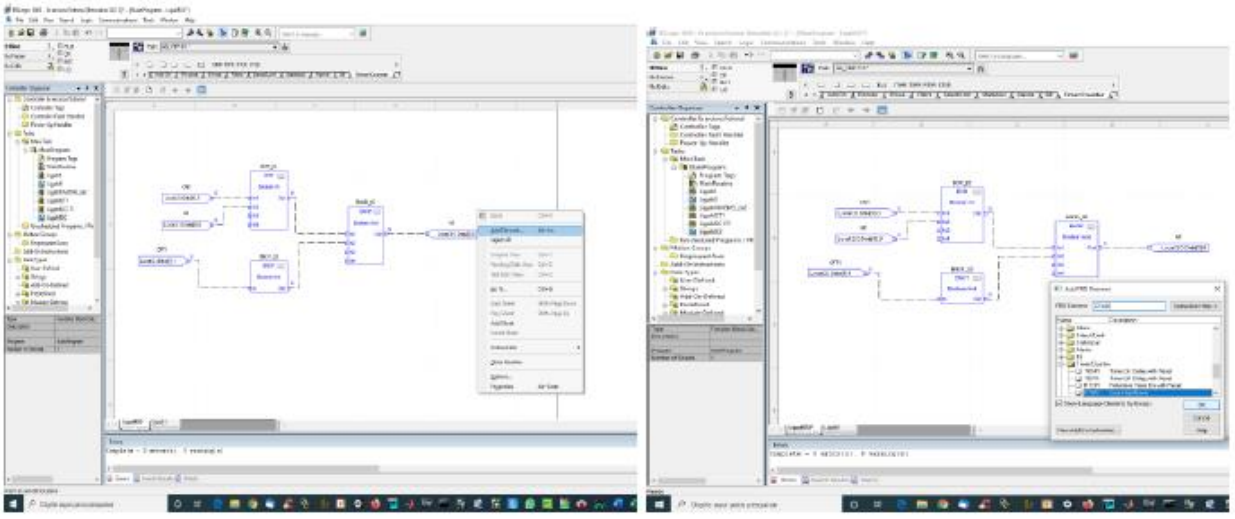
Figura 192 – Programa completo em FBD



Fonte: Própria

Outra forma de inserir a instrução em diagrama de blocos de funções (FBD) pode ser obtida clicando-se com o botão direito sobre a folha de programação e marcando a opçãp *Add Element*, conforme primeira ilustração da figura 193, o que originará a segunda tela, onde se digita o nome do bloco a ser inserido, ou seja, a instrução CTUD.

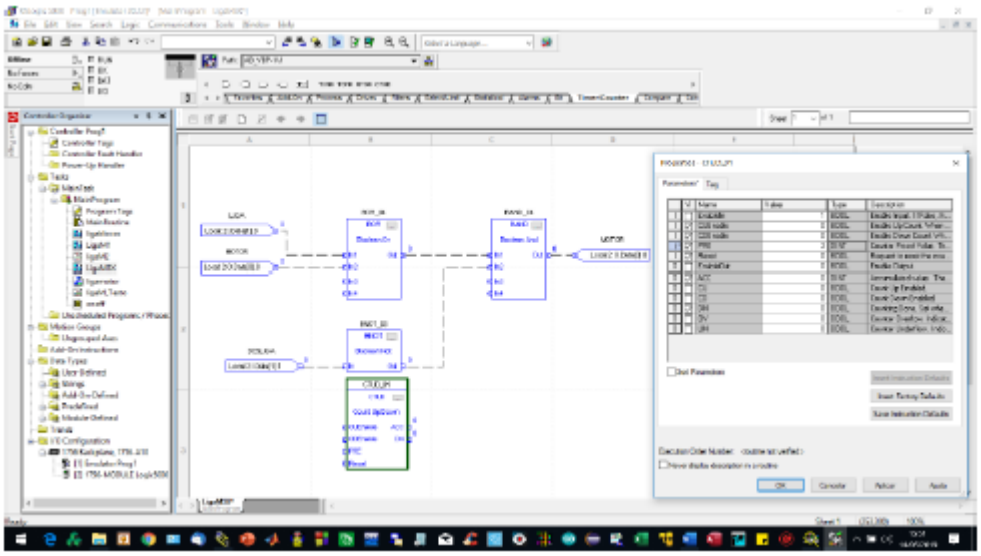
Figura 193 – Inserindo instrução CTUD



Fonte: Própria

Clicando-se duas vezes sobre a instrução contadora, pode-se ajustar o *Preset*, ou limite de contagem, digitando o número 3. Isso permitirá 2 acionamentos do motor, desligando-o automaticamente na terceira tentativa de ativação, conforme figura 194.

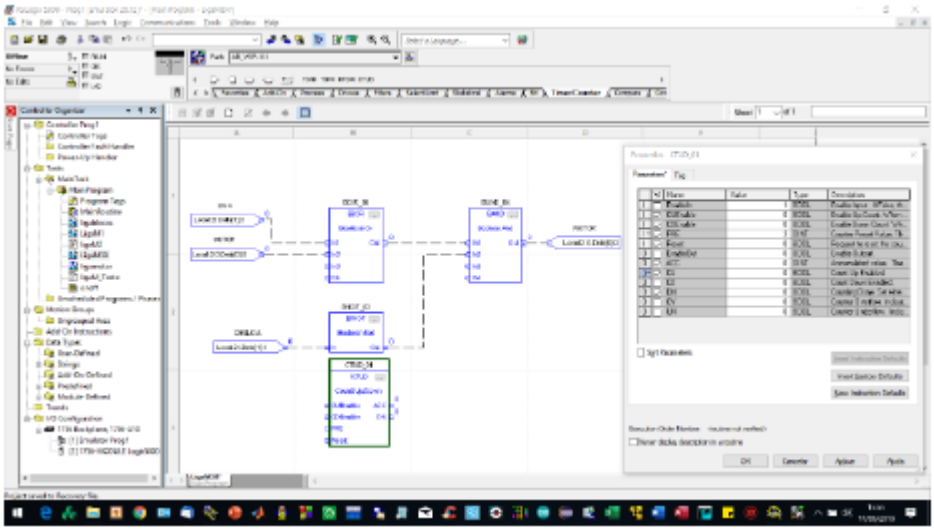
Figura 194 – Desativação do motor após dois acionamentos



Fonte: Própria

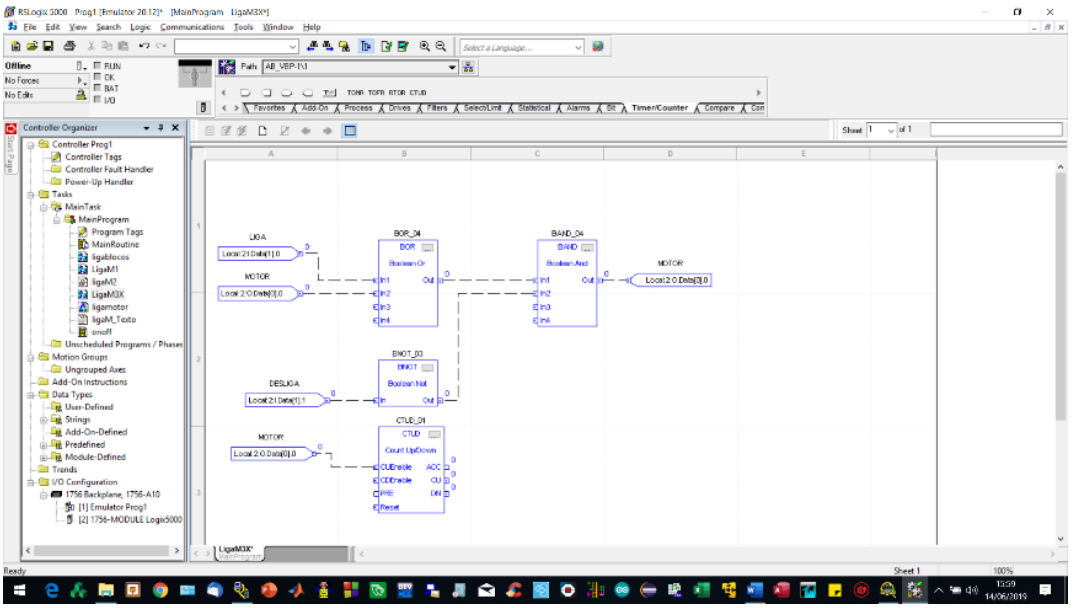
O próximo passo consiste em habilitar contagem crescente, o que é conseguido marcando-se no botão de checagem da entrada CU (Count Up), segundo mostra a figura 195.

Figura 195 – Ativando a entrada Count Up da instrução CTUD



Fonte: Própria

Figura 196 – Configurando a entrada CUEnable

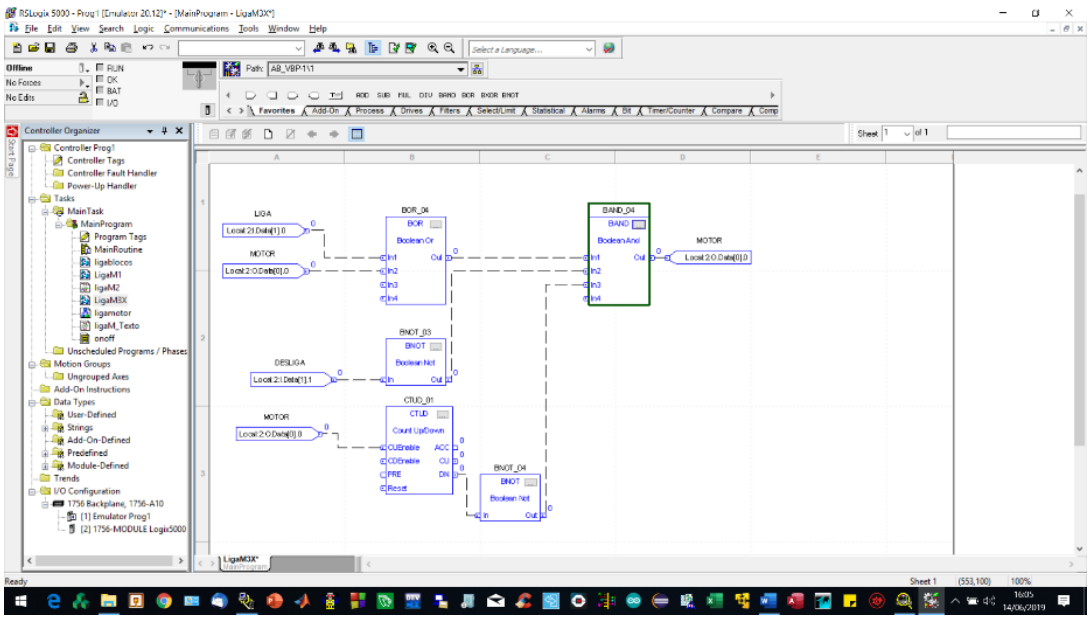


Fonte: Própria

Conforme figura 196, insere-se na entrada *CUEnable* (Habilita contagem crescente) do contador um bloco *Input Reference* (referência de entrada), atribuindo ao mesmo o endereço de saída (Local:2:O.Data[0].0), para contabilizar os acionamentos motores.

De acordo com a figura 197, é necessário conectar a saída *DN* do contador em série com um bloco *BNOT*, encontrado na aba *Favorites*. Isso ocorre porque enquanto *ACC* for menor que *PRESET*, a saída *DN* será igual a 0. Assim, *DN* precisa ter seu valor barrado, para se tornar 1, e proporcionar condições ao acionamento do motor, após ser conectado a uma das entradas do bloco *BAND*.

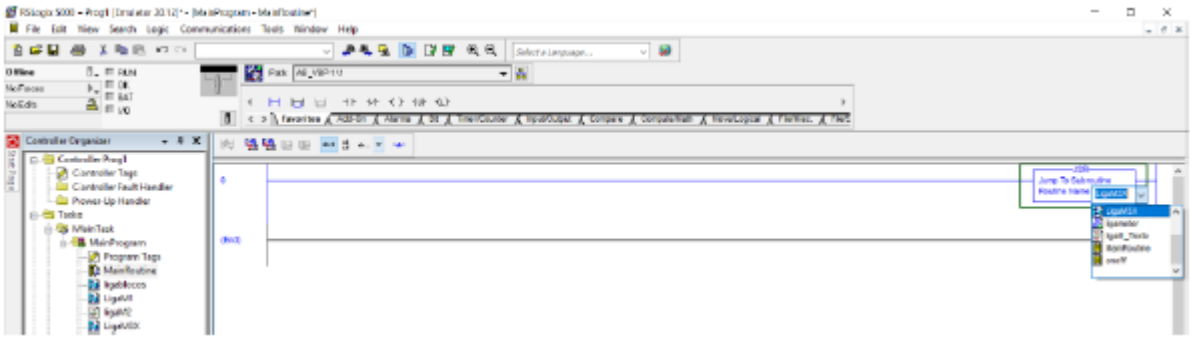
Figura 197 – Conexão da saída DN ao bloco BNOT



Fonte: Própria

O próximo passo consiste em Acessar o código da rotina principal (MainRoutine), selecionando na instrução JSR (Jump to Subroutine) o nome do código em blocos que acabou de ser criado, ou seja, ligaM3X, conforme visto na figura 198.

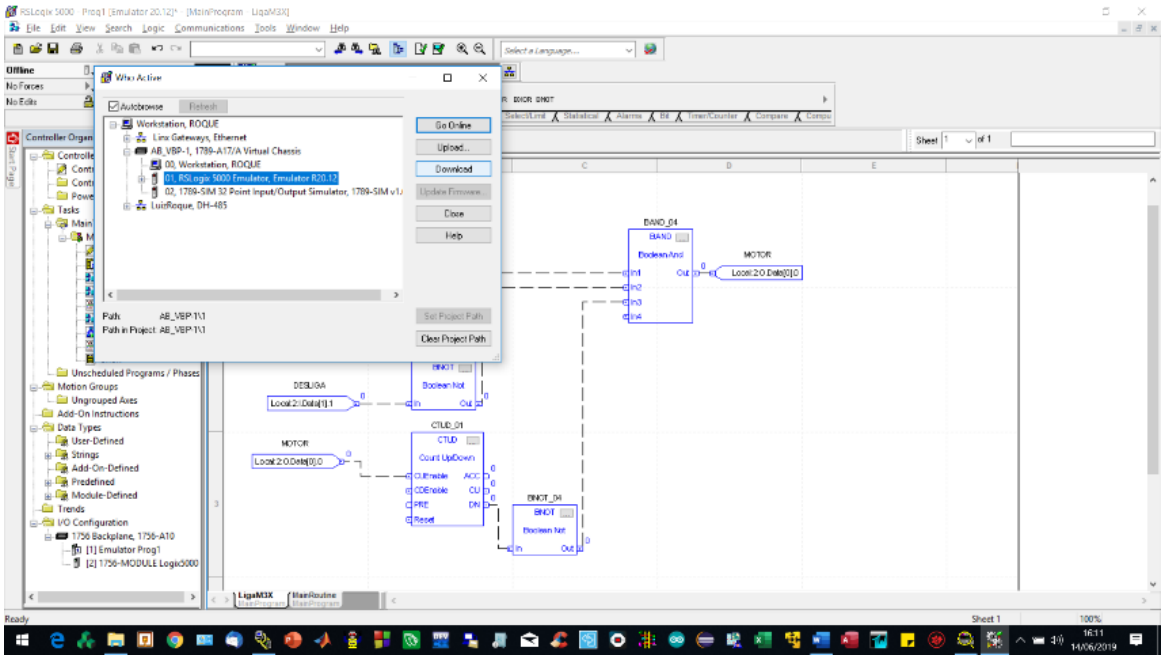
Figura 198 – Seleção do programa a ser executado pela instrução JSR



Fonte: Própria

Surge o momento de efetuar o download, e executar o código (fig. 199).

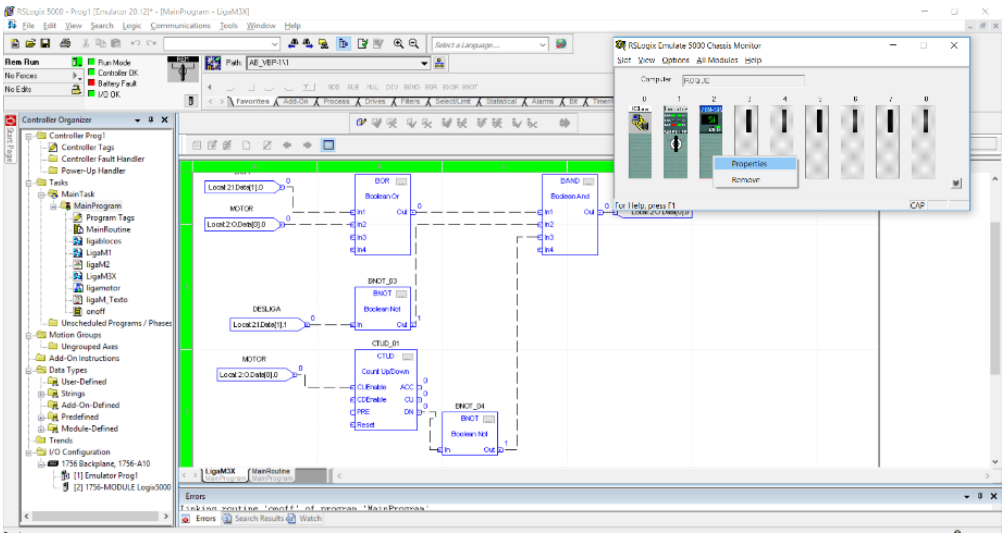
Figura 199 – Efetuando download do programa no emulador



Fonte: Própria

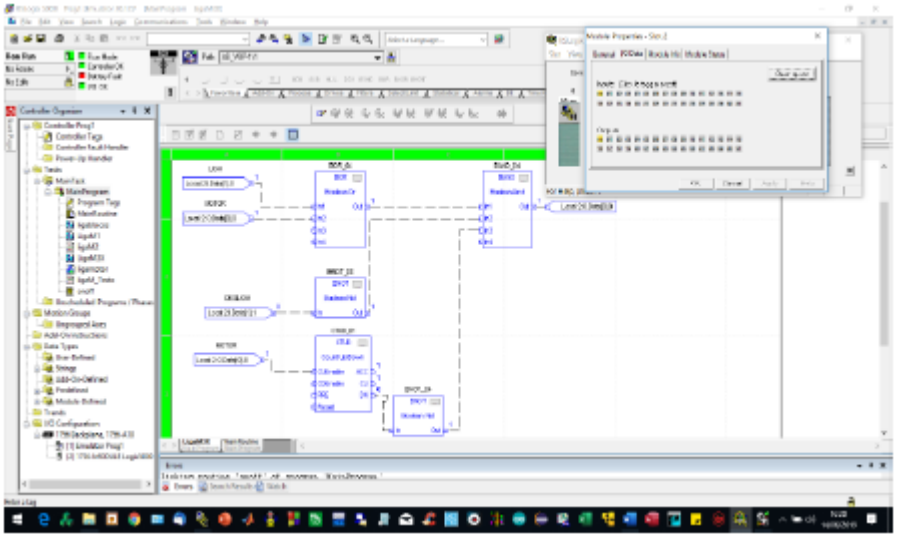
Conforme figura, deve-se selecionar o modo de execução (run mode), maximizar o RSLogix5000 Emulate, clicar com o botão direito sobre o slot 2, e escolher a opção Properties.

Figura 200 – Executando o programa



Fonte: Própria

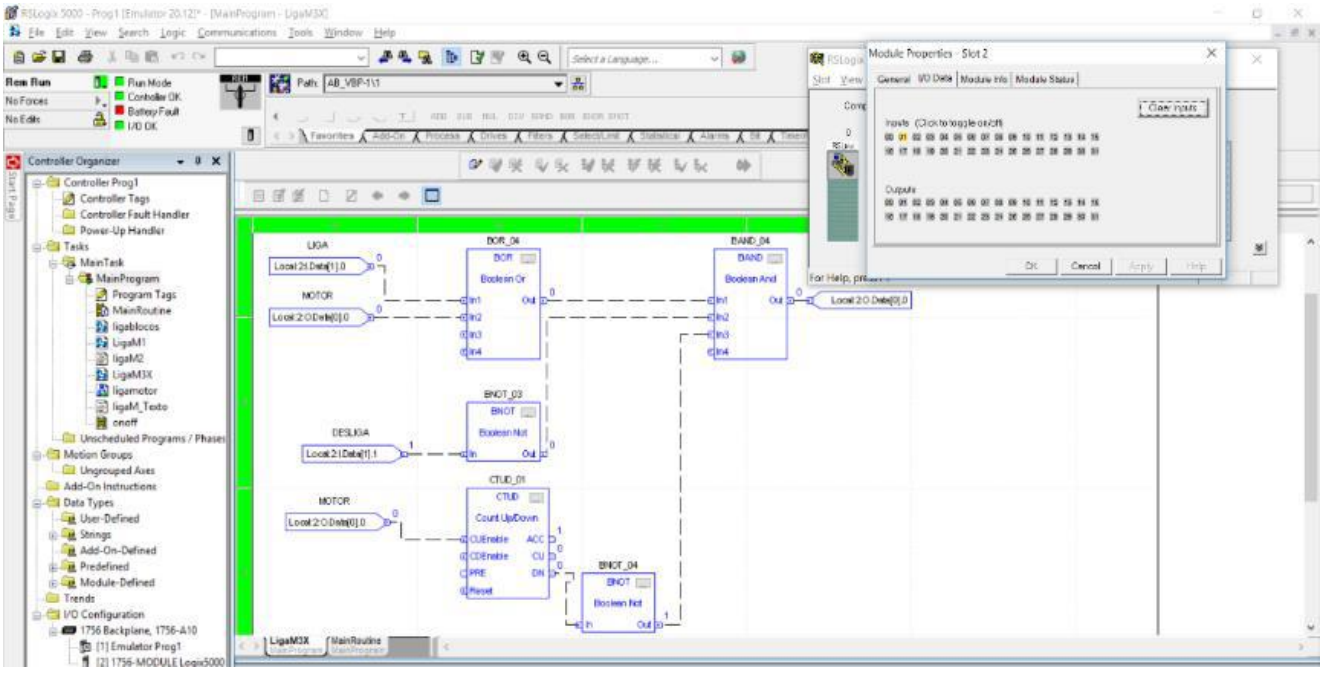
Figura 201 – Ativando o motor



Fonte: Própria

Clica-se sobre a aba I/O Data, para iniciar o controle do processo. O primeiro acionamento do motor é alcançado ativando a entrada zero no emulate, conforme figura 201, o que acenderá o sinalizador da saída zero. Percebe-se na figura que a saída ACC do contador já contabilizou um acionamento.

Figura 202 – Desativando o motor

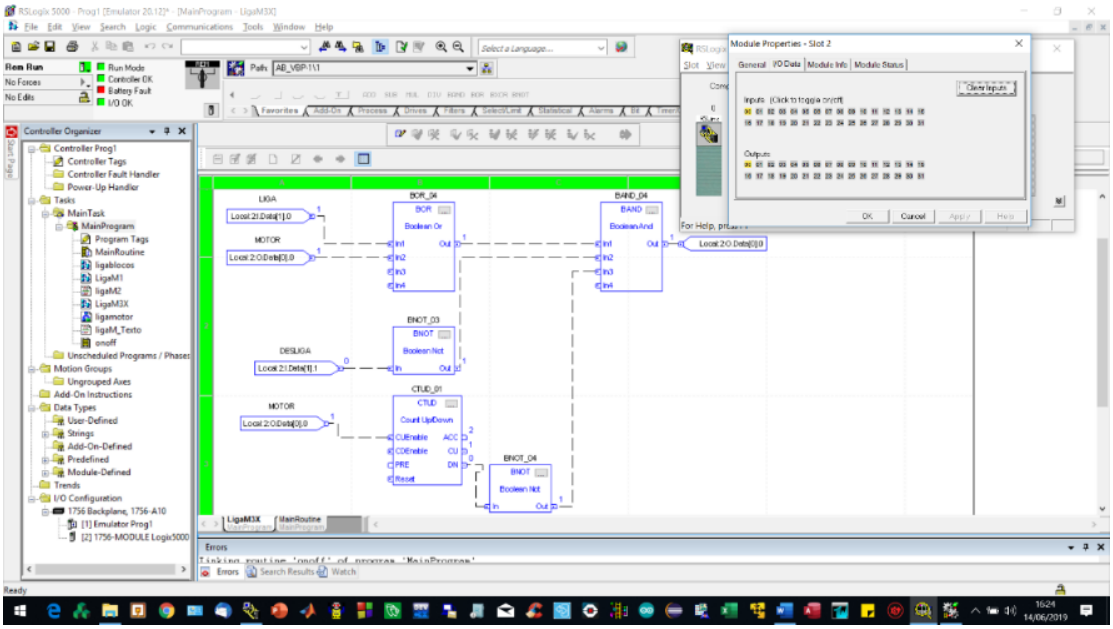


Fonte: Própria

Para desligar o motor é preciso soltar o botão da entrada zero no RSLogix5000 *Emulate*, referente ao acionamento, e ativar a entrada 01, que desligará a máquina, de acordo com a figura 202.

O segundo acionamento pode ser conseguido clicando novamente sobre a entrada 01, para desativar o botão desliga no Emulate. Em seguida, pressiona-se novamente o botão da entrada 00, inserido nível lógico alto na primeira entrada da porta *BOR*, fazendo com que todas as entradas do bloco *BAND* estejam ativadas, proporcionando acionamento da saída 00 e, consequemente, do motor. Verifica-se que o registrador ACC passa a ser igual a 2, no código em blocos apresentado na figura 203.

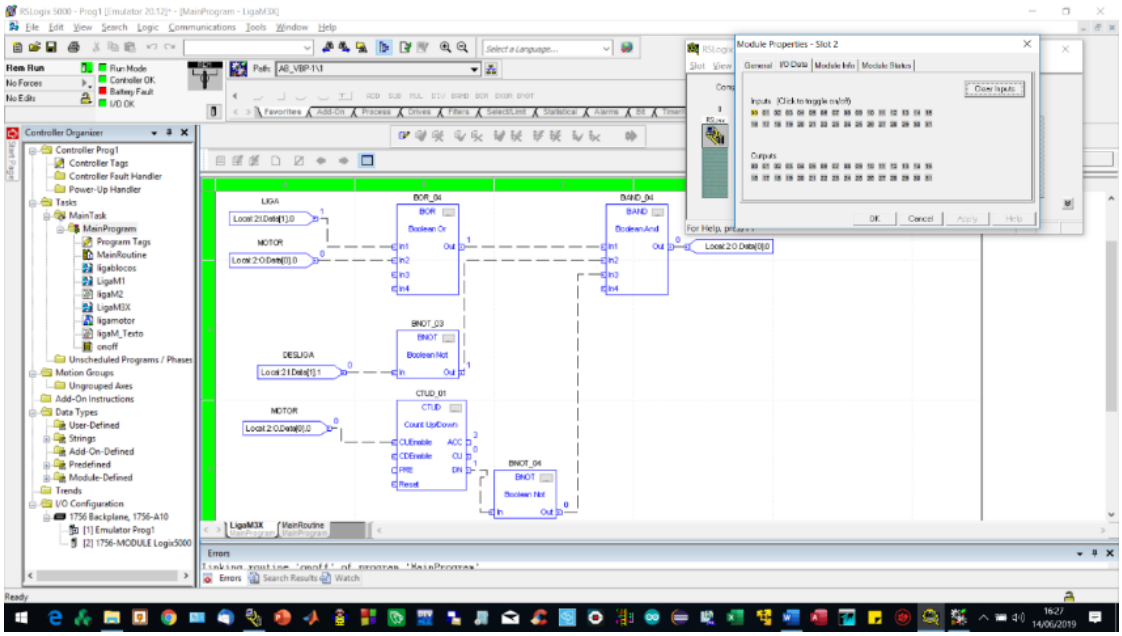
Figura 203 – Contabilizando segundo acionamento motor



Fonte: Própria

É preciso desativar a entrada 00 (referente ao botão liga) clicando-se novamente sobre ela, para desligar o motor através do acionamento da entrada 01 (referente ao botão desliga). Feito isso, inicia-se a terceira tentativa de acionar o motor, o que é efetuado pressionando-se a entrada 00. Percebe-se na figura 204 que a saída foi automaticamente desligada, porque DN passou a ser igual a 1, uma vez que ACC igualou o valor de PRESET (três). Esse novo valor de DN é barrado, fazendo com que uma das entradas do bloco BAND seja zero, desativando automatimente a saída.

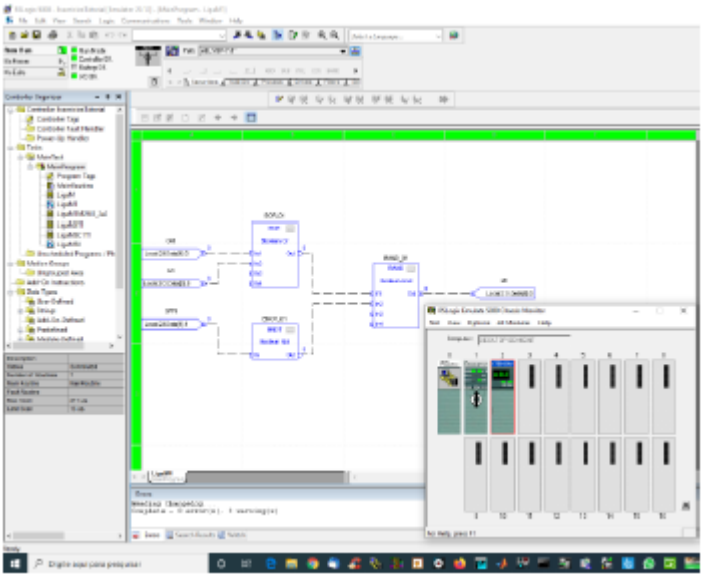
Figura 204 – Desativação automática do motor



Fonte: Própria

3) pede-se complementar o programa elaborado no exercício 1, desenvolvendo um sistema supervisório para o mesmo.

Figura 205 – Solução da automação em FBD do exercício 1



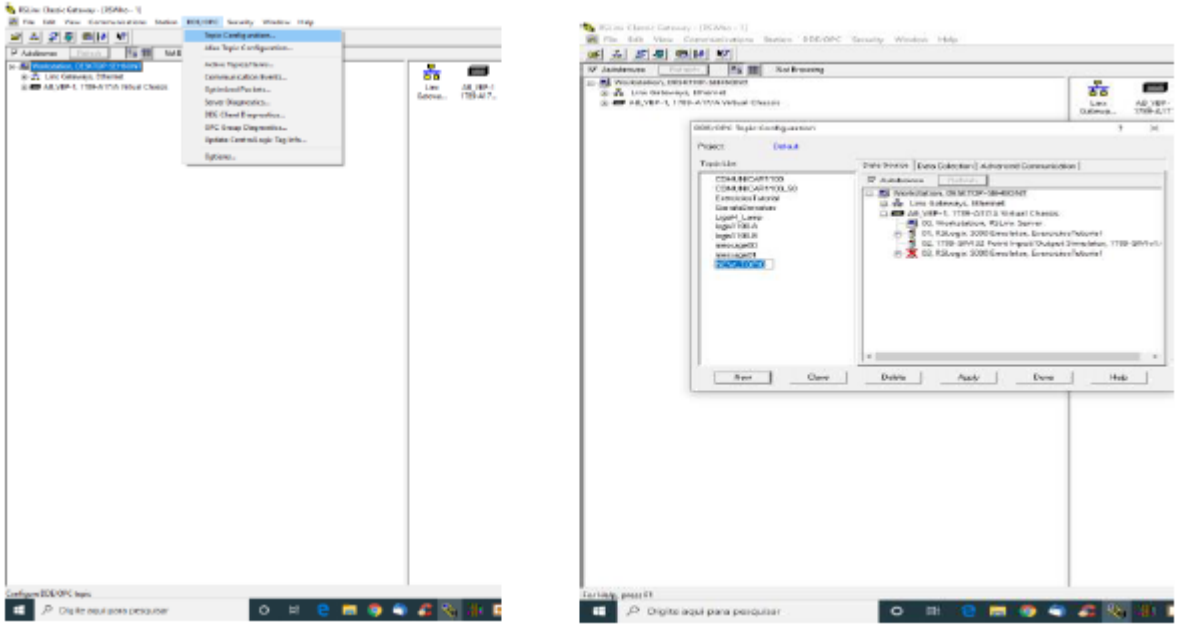
Fonte: Própria

Antes de iniciar o desenvolvimento do sistema supervisório pedido, é preciso criar um tópico no software RSLinx, para compartilhar dados entre os ambientes de desenvolvimento integrados (IDE – acrônimo para *Integrated Development Environment*) utilizados através do protocolo de intercâmbio

dinâmico de dados (DDE – acrônimo para Dynamic Data Exchange), que é nativo do sistema operacional *Windows.* Para tal fim, é preciso deixar o programa em execução no IDE RSLogix 5000, conforme figura 205, juntamente com o RSLogix 5000 Emulate.

A seguir, maximixa-se o RSLinx, selecionando as opções DDE/OPC e Topic Configuration, de acordo com a figura 206, o que originará a segunda tela da mesma, na qual se clica sobre o botão New, para criar novo tópico (NEW_TOPIC).

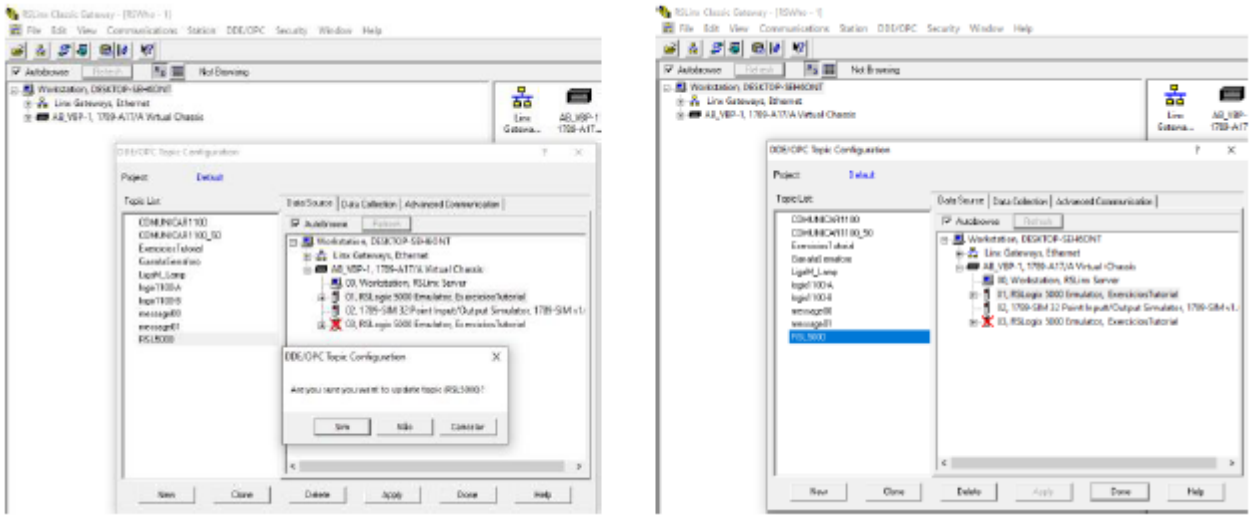
Figura 206 – Configuração de tópico DDE no RSLinx



Fonte: Própria

Modifica-se o nome NEW_TOPIC para RSL5000, aplicando-o em seguida ao nó 01, associado às informações RSLogix 5000 Emulator, ExercíciosTutorial, conforme figura 207.

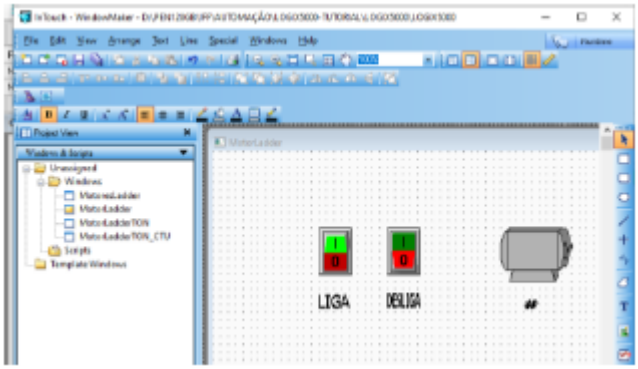
Figura 207 – Aplicando tópico criado ao nó 1



Fonte: Própria

Em seguida, pressiona-se o botão Done na tela da figura, minimiza-se o RSLinx e abre-se o sistema supervisório In Touch, para elaborar a supervisão do processo, de acordo com a figura 208, na qual se percebem três principais objetos, dos quais dois são botões (para ligar e desligar), enquanto o terceiro é o motor que será controlado por seus elementos acionadores. Os objetos restantes são os textos onde se observam as informações LIGA e DESLIGA, para destacar a função de cada botão, e tornar a operação do sistema supervisório mais clara ao usuário. Encerrando a descrição dos objetos, percebe-se a cerquilha abaixo do motor, configurada para apresentar duas mesagens distintas, que são ativado ou desativado, conforme a condição em que se encontre a operação da máquina controlada.

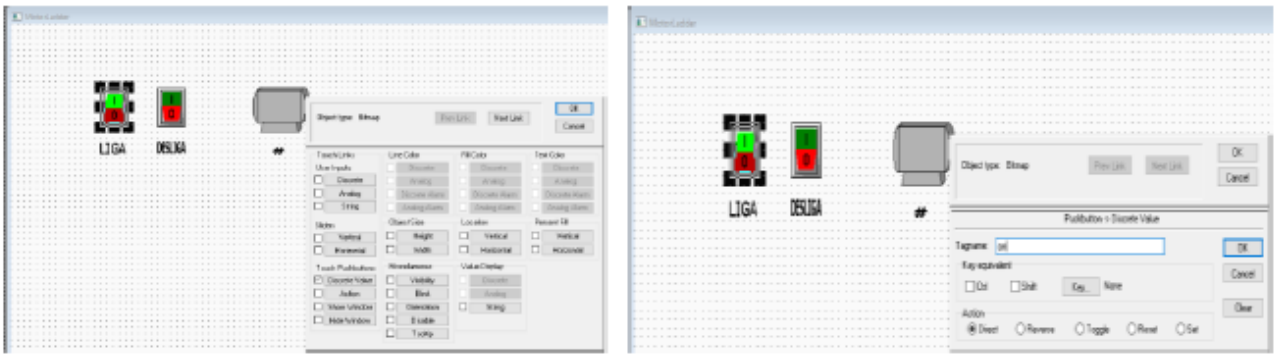
Figura 208 – Sistema supervisório do problema



Fonte: Própria

Para configurar o botão liga no sistema supervisório, clica-se duas vezes sobre o mesmo, proporcionando a obtenção da figura 209, na qual se marca a propriedade *Touch Pushbuttons – Discrete Value*, cujas definições de *tagname* são vistas na segunda tela.

Figura 209 – Configurando as propriedades do botão liga

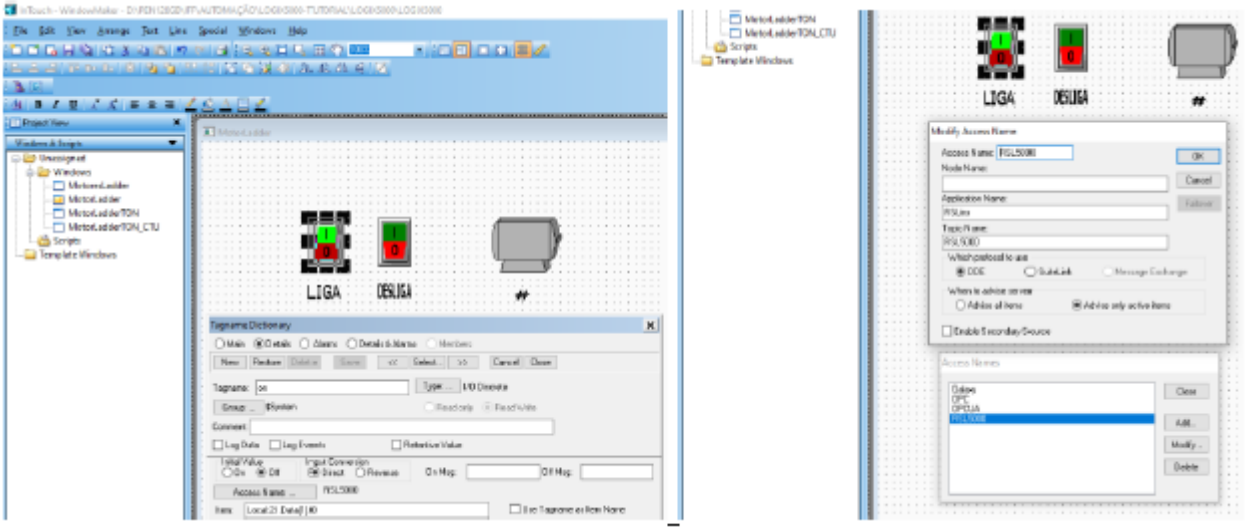


Fonte: Própria

Com isso percebe-se que a tagname on controla o acionamento do motor, e sua ação é direta (Direct), proporcionando à mesma um comportamento análogo à chave de impulso, cuja dinâmica de funcionamento é semelhante a uma campainha doméstica, ou seja, seu contato comuta da posição de repouso somente enquanto o botão estiver pressionado. Tão logo o botão seja solto, o contato retornará ao estado de repouso.

Quando o botão OK for pressionado na segunda tela da figura, surgira a pergunta Define Tagname? Após confirmar a criação da variável, surgirá o dicionário de tagnames, onde são definidos os parâmetros da Tagname on, conforme figura 210.

Figura 210 – Configurando as propriedades da tagname on



Fonte: Própria

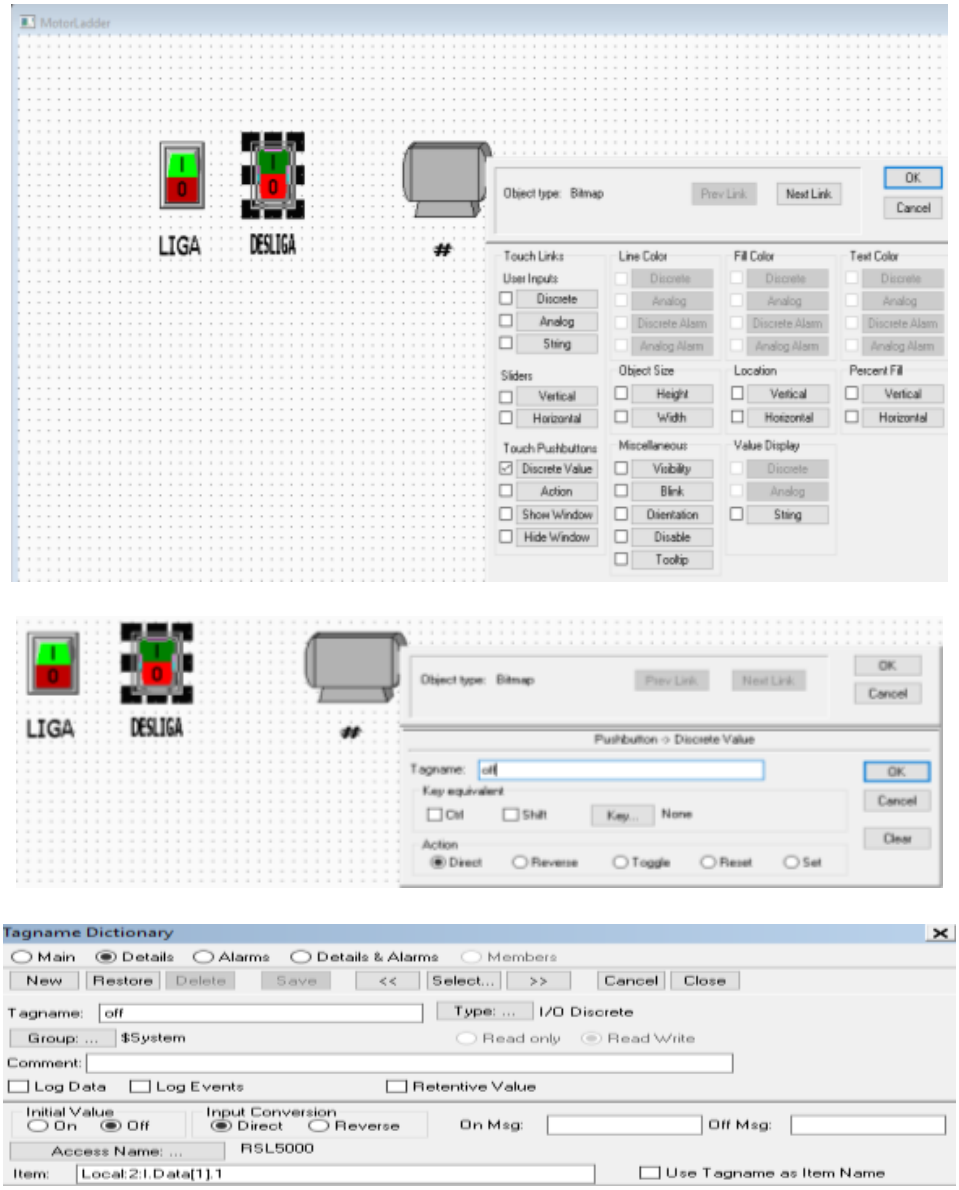
Percebe-se que foi atribuído à tagname *on* o tipo *I/O Discrete*, indicando que a mesma será binária ou booleana, e se comunicará com elemento do programa ladder, cujo endereço é especificado em *Item* como sendo o mesmo do botão liga, ou seja, Local:2:I.Data[1].0 .

Após clicar sobre o botão *Access Name* na primeira tela da Figura, surgirá a segunda tela da mesma, na qual se atribuem as configurações de nome de Accesso apresentadas. Embora não seja obrigatório, recomenda-se designar o mesmo nome do tópico que foi criado no RSLinx, conforme pode ser verificado na caixa de texto *Access Name*, identificada na segunda tela da figura por RSL5000. O campo *Node Name ficará vazio,* pois a aplicação não está rodando em um nó específico.

Salienta-se a obrogatoriedade de atribuir nome de tópico idêntico ao criado no RSLinx. Por esse motivo, o campo *Topic Name* também é designado por RSL5000.

As propriedades do botão desliga são obtidas clicando-se duas vezes sobre o mesmo, conforme primeira tela da figura 211, na qual se observa assinalada a opção *Touch Pushbuttons - Discrete Value*. Clicando-se no botão ao lado da propriedade assinalada, aparecerá na segunda tela da figura a *tagname* que controlará a propriedade de toque, definida como *off*, juntamente com a ação atribuída a mesma, identificada por *Direct*.

Figura 211 – Configurando as propriedades do botão desliga

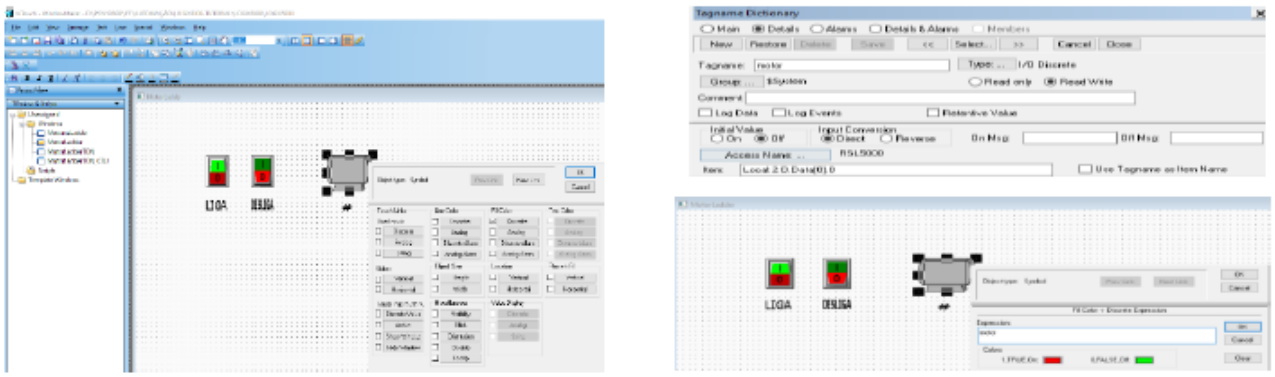


Fonte: Própria

Após confirmar a criação da *tagname*, surge a terceira tela da figura, onde se percebem as configurações da variável *off*, cujo tipo é I/O Dicrete, para lembrar que se trata de elemento de entrada e saída, que pode se comunicar com o programa em execução, através do endereço especificado no campo *Item* (Local:2:I.Data[1].1).

A ação *Direct* fará com que o contato virtual endereçado pela *tagname off* no programa para controlar o motor se comporte como um botão do tipo impulso. Isso significa que o contato virtual do programa ladder cujo endereço é Local:2:I.Data[1].1 comutará para o estado aberto apenas enquanto o botão desliga for pressionado, para interromper a circulação de corrente virtual e desativar a bobina de saída, retornando em seguida a sua situação de repouso, que é normalmente fechado (NF).

Figura 212 – Configurações do objeto motor



Fonte: Própria

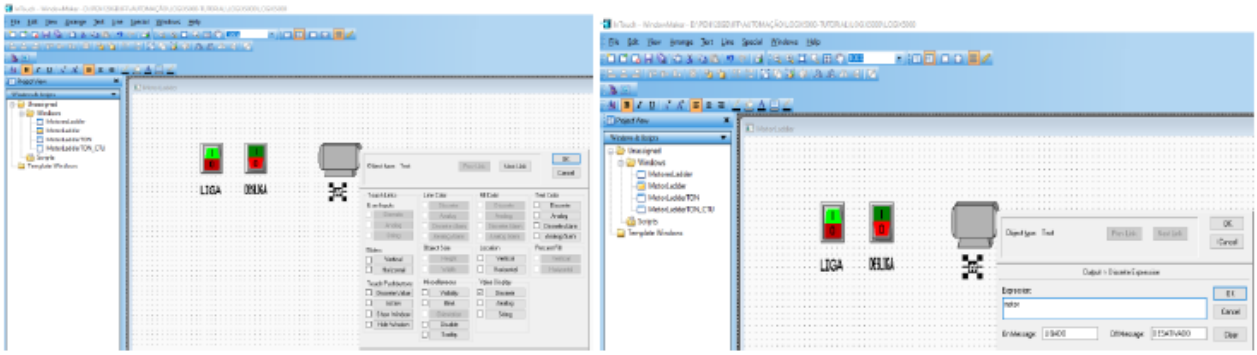
A propriedade utilizada no objeto motor é apresentada na figura 212, juntamente com a *tagname* que a controla, acompanhada das respectivas configurações. Percebe-se nas ilustrações da figura 212 que a propriedade *Fill* Color Discrete é a única associada o objeto, enquanto se estabeleceu a tagname motor como variável de controle para o preenchimento automático de suas cores.

A cor do motor será vermelha enquanto o valor de sua *tagname* homônima for zero ou falsa, conferindo ao mesmo o estado desativado. Por outro lado, a cor do motor será verde enquanto o valor de sua tagname homônima for um ou verdadeiro, conferindo ao mesmo o estado ativado. Observam-se as propriedades da tagname motor na terceira ilustração da figura, definida como tipo I/O Discrete, *Access Name* RSL5000 e Item Local:2:O.Data[0].0, indicando que a variável recebe o endereço da bobina de saída no programa.

Cabe destacar que a partir do momento em que se cria e atribui determinado nome de Accesso a um objeto numa tela do supervisório In Touch, os demais elementos inseridos receberão automaticamente este mesmo Access Name.

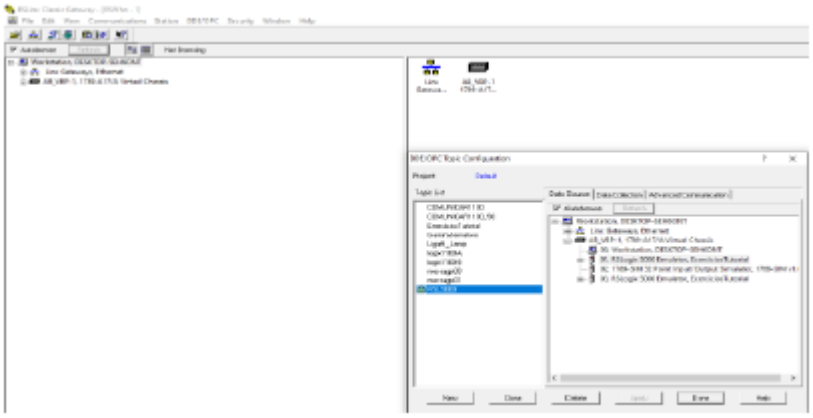
Por último, clica-se duas vezes para Acessar as configurações do objeto cerquilha, originando a primeira ilustração da figura 213, onde se percebe a propriedade Value *Display Discrete* como a única selecionada. Ao pressionar o botão Discrete assinalado na propriedade da cerquilha, aparecerá a segunda tela da figura, onde se define como expressão a tagname motor, que definirá o texto a ser exibido. Quando o motor estiver em operação, a cerquilha exibirá a mensagem LIGADO. Por outro lado, a cerquilha exibirá a mensagem DESATIVADO enquanto o motor estiver inoperante.

Figura 213 – Configurações do objeto cerquilha



Fonte: Própria

Figura 214 – Configuração do tópico de comunicação DDE com o sistema supervisório

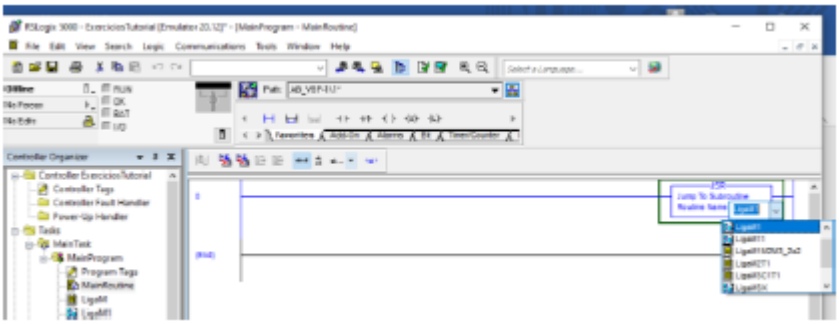


Fonte: Própria

Para confirmar se a comunição entre o sistema supervisório e o programa *FBD* está funcionando corretamente, precisa-se Acessar o software *RSLinx*, selecionar o menu DDE/OPC e, em seguida, marcar a opção *Topic Configuration*, proporcionando o surgimento da figura 214, na qual se percebe um cadeado amarelo ao lado esquerdo do tópico criado, RSL5000, indicando operação adequada do intercâmbio de dados entre os amboientes de desenvolvimento integrado (IDE) utilizados.

A partir deste ponto, clica-se duas vezes sobre *MainRoutine*, que é a rotina principal através da qual a instrução JSR realiza salto quando o programa estiver em operação, para selecionar o nome do programa que será executado pelo RSLogix 5000, que é LigaM1, conforme figura 215.

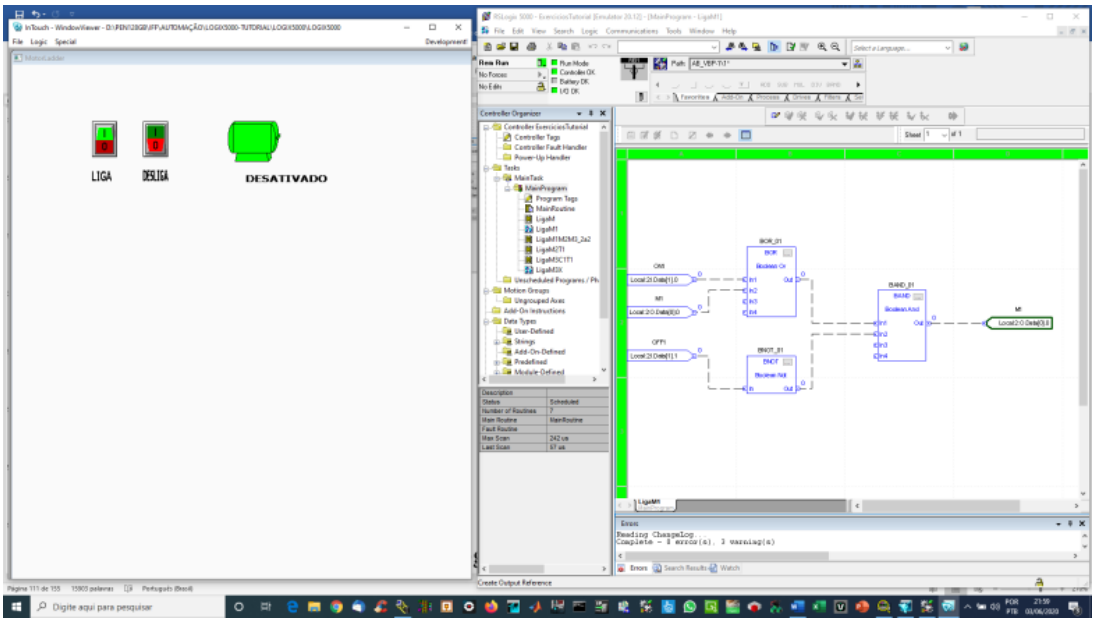
Figura 215 – Selecionando rotina a ser executada pela instrução JSR



Fonte: Própria

A seguir, efetua-se o download do programa LigaM1, colocando-o em modo de execução, juntamente com seu respectivo sistema supervisório, para observar o funcionamento do conjunto. A figura 216 apresenta o controle do programa em diagrama de blocos de funções (FBD) através da tela de supervisório desenvolvida.

Figura 216 – Programa em FBD controlado pelo sistema supervisório

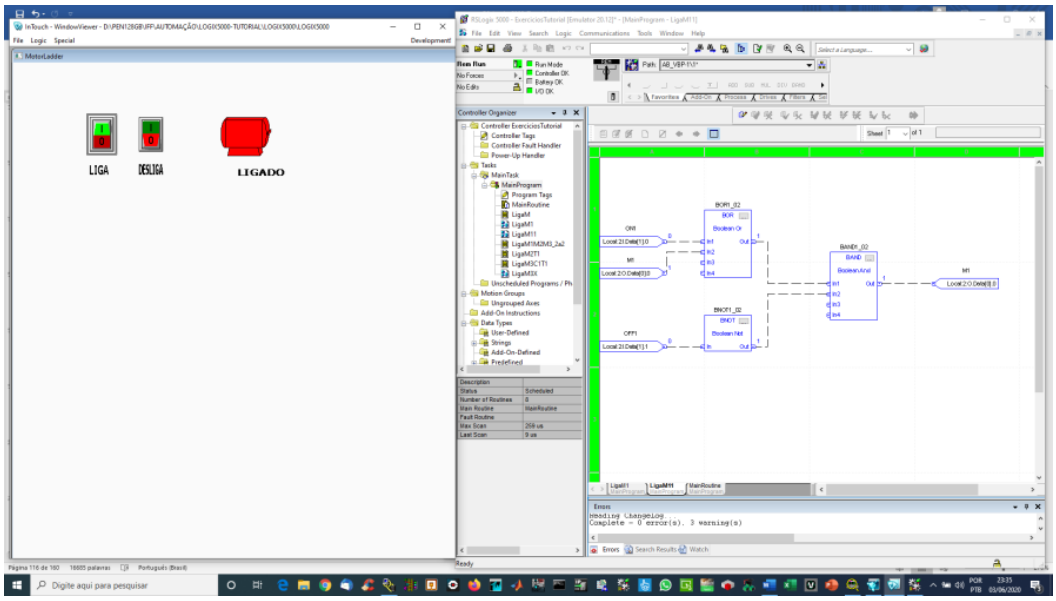


Fonte: Própria

Após pressionar o botão liga no sistema supervisório, a primeira entrada do bloco BOR passará a ter nível lógico alto, pois ambos os elementos possuem o mesmo endereço, especificado no campo *Item* (Local:2:I.Data[1].0) da tela à esquerda, e no bloco *Input Reference* ON1, visto na ilustração direita (fig. 217),conforme já explanado anteriormente. Isso fará com que a operação lógica OU leve a saída do bloco BOR ao nível lógico alto, a qual alimentará a primeira entrada da porta BAND. Como o motor

está desativado, o inversor BNOT leva nível lógico elevado à segunda entrada da porta *BAND*, fazendo com que a operação lógica resultante seja nível lógico alto, direcionado à saída de referência, terminando por ativar o motor.

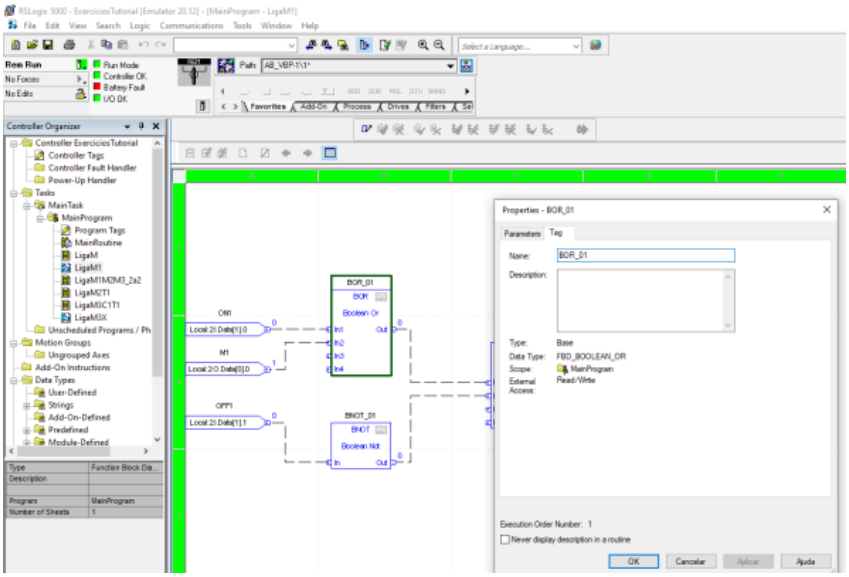
Figura 217 – Acionamento do motor pelo sistema supervisório



Fonte: Própria

Percebe-se na figura 217 que o motor passa a ter cor vermelha, e apresentar a mensagem LIGADO, pois a tagname homônima (que controla as propriedades de preenchimento dele e de exibição de mensagens da cerquilha) se tornou verdadeira, devido ao acionamento do motor pelo programa em diagrama de blocos de funções.

Figura 218 – Propriedades do bloco BOR

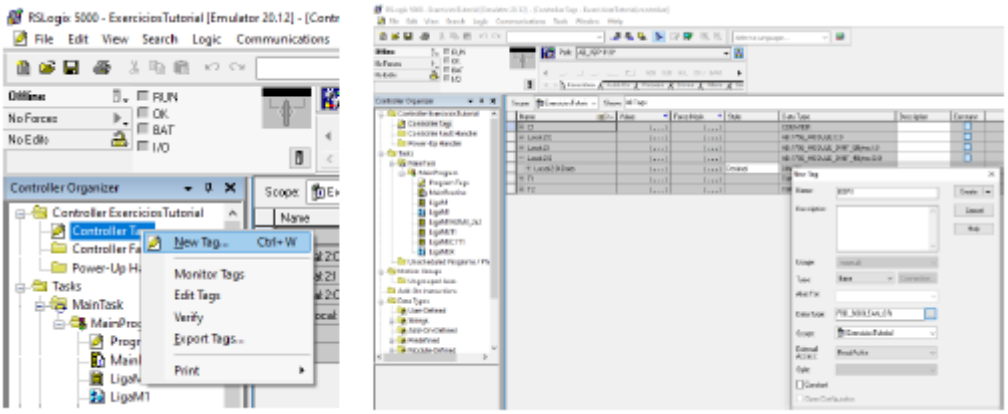


Fonte: Própria

Clicando-se duas vezes sobre um dos blocos lógicos, como a função BOR, para Acessar as propriedades de suas tags, conforme figura 218, percebe-se que as mesmas têm escopo restrito ao programa em que atuam, ou seja, em diagrama de blocos de funções.

Caso seja necessário atribuir escopo global às tags dos blocos de funções, levam-se os programas *In Touch* e RSLogix 5000 aos modos de desenvolvimento e *offline*, respectivamente. A seguir, clica-se duas vezes sobre Controller Tags e, no menu que surgir, seleciona-se a opção New Tag, originando a segunda ilustração da figura 219, onde se nomeia a tag como BOR1, atribuindo-se à mesma o tipo de dados FBD_BOOLEAN_OR e escopo ExercíciosTutorial, indicando que seu alcance engloba todo o projeto desenvolvido. Confirma-se a criação da tag BOR1 pressionando-se o botão Create.

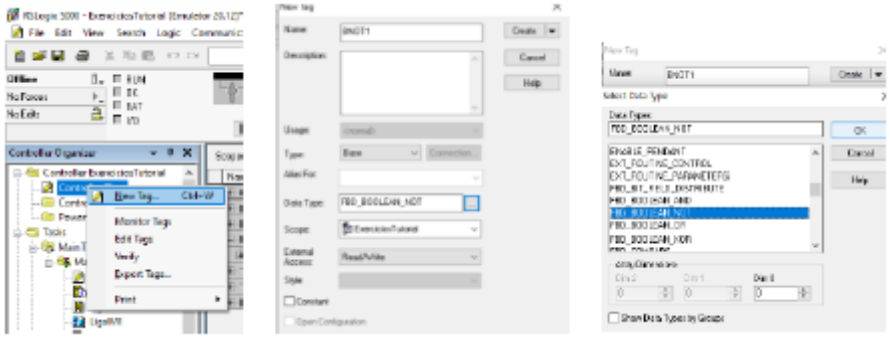
Figura 219 – Criando tag BOR1



Fonte: Própria

Para criar a tag da função BNOT com escopo global, clica-se novamente duas vezes sobre *Controller* Tags e, no menu que surgir, seleciona-se a opção *New Tag*, originando a segunda tela da figura 220, onde se definem as configurações da tag como nome (BNOT1), tipo de dados FBD_BOOLEAN_NOT e escopo ExercíciosTutorial, indicando que seu alcance engloba todo o projeto desenvolvido. Confirma-se a criação da tag BNOT1 pressionando-se o botão Create.

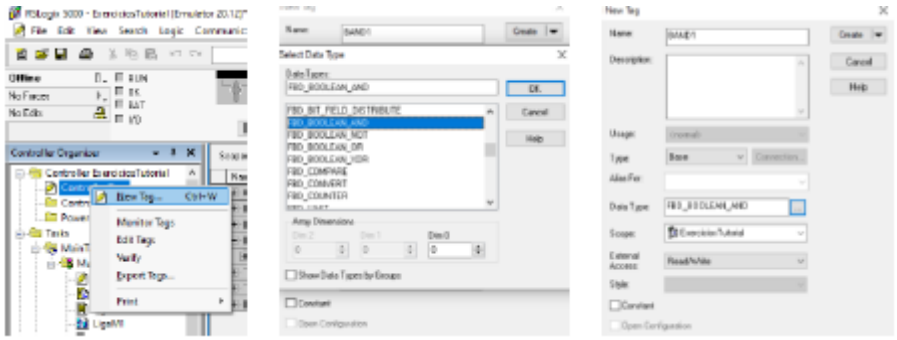
Figura 220 – Criação da tag BNOT1



Fonte: Própria

Por último, procede-se à criação da tag BAND1, conforme figura 221, que traz as respectivas configurações a ela atribuídas.

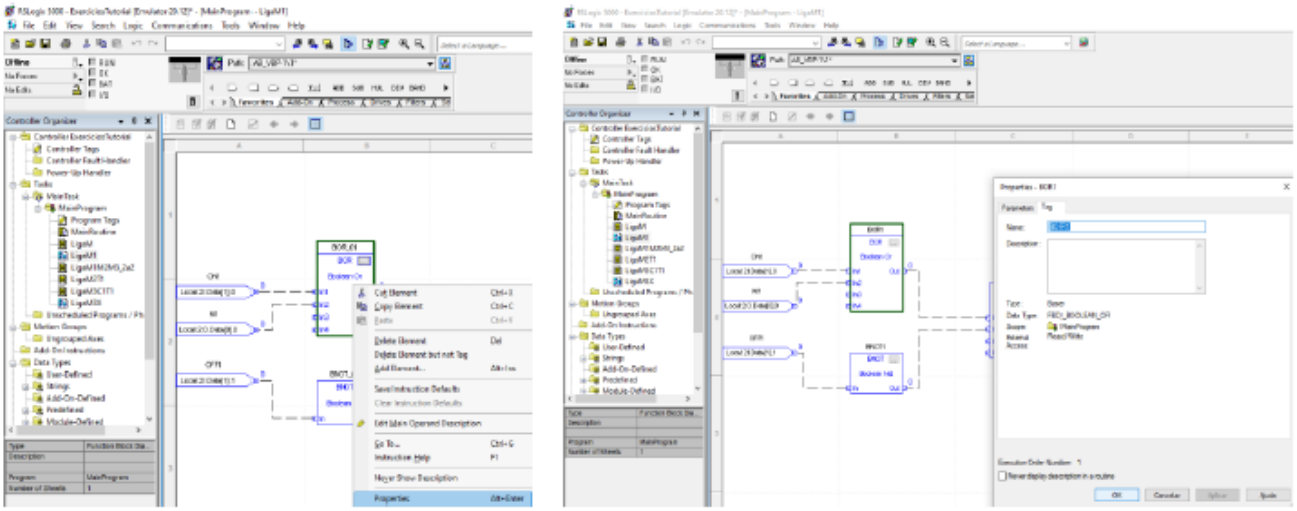
Figura 221 – Criação da tag BAND1



Fonte: Própria

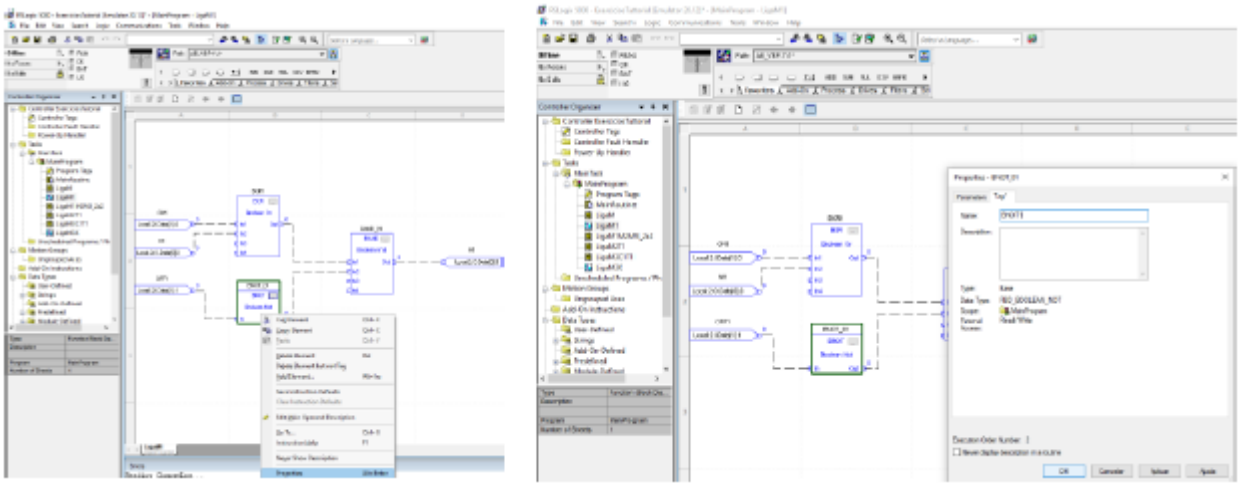
A próxima etapa consiste em abrir o programa FBD LigaM1, e associar aos blocos de funções os nomes de escopo global que foram criados (fig. 222).

Figura 222 – Associação de nomes das tags aos respectivos blocos



Fonte: Própria

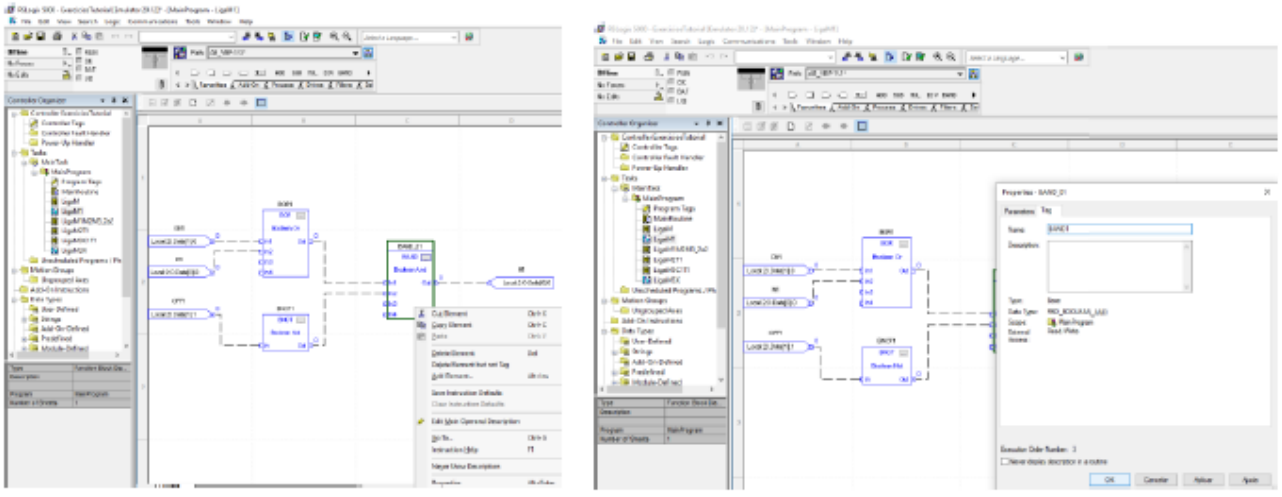
Figura 223 – Associando tagname BNOT 1 ao bloco BNOT_01



Fonte: Própria

O Acesso às configurações do bloco BOR é alcançado clicando-se sobre o mesmo com o botão direito, originando a primeira tela da figura 222, cuja seleção da opção *Properties* proporcionará o surgimento da segunda ilustração, na qual se marca a aba tag, para digitar o nome BOR1 sobre a mesma, pressionando-se em seguida o botão OK. Procede-se conforme supracitado com os demais blocos, de acordo com as figuras 223 e 224.

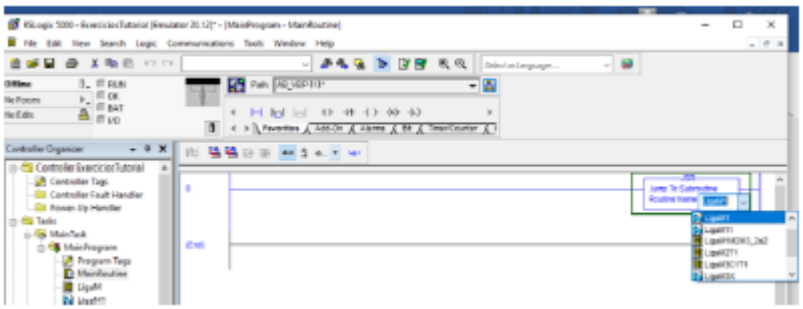
Figura 224 – Associando tagname BAND1 ao bloco BAND_01



Fonte: Própria

A partir deste ponto, clica-se duas vezes sobre MainRoutine, que é a rotina principal através da qual a instrução JSR realiza salto quando o programa estiver em operação, para selecionar o nome do programa que será executado pelo RSLogix 5000, que é LigaM1, conforme figura 225.

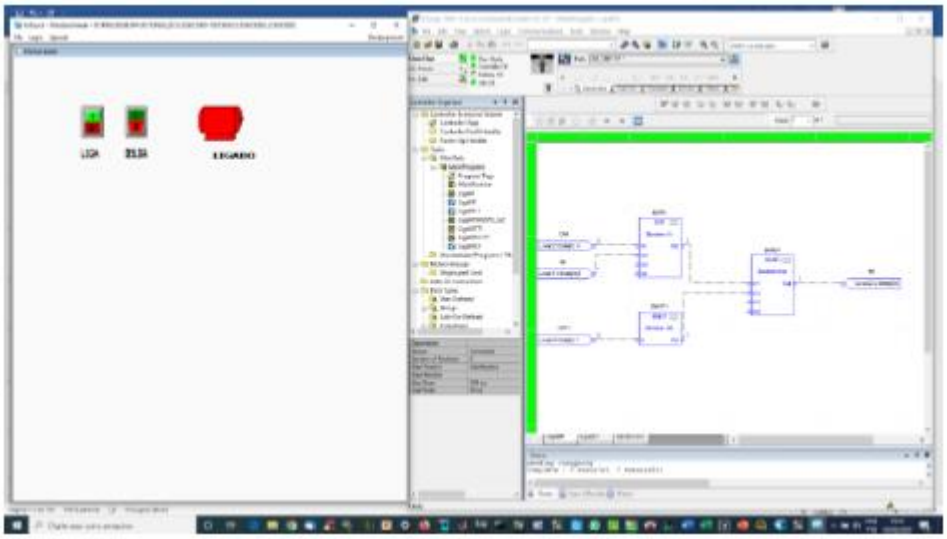
Figura 225 – Seleção da rotina LigaM1 para execução na instrução JSR



Fonte: Própria

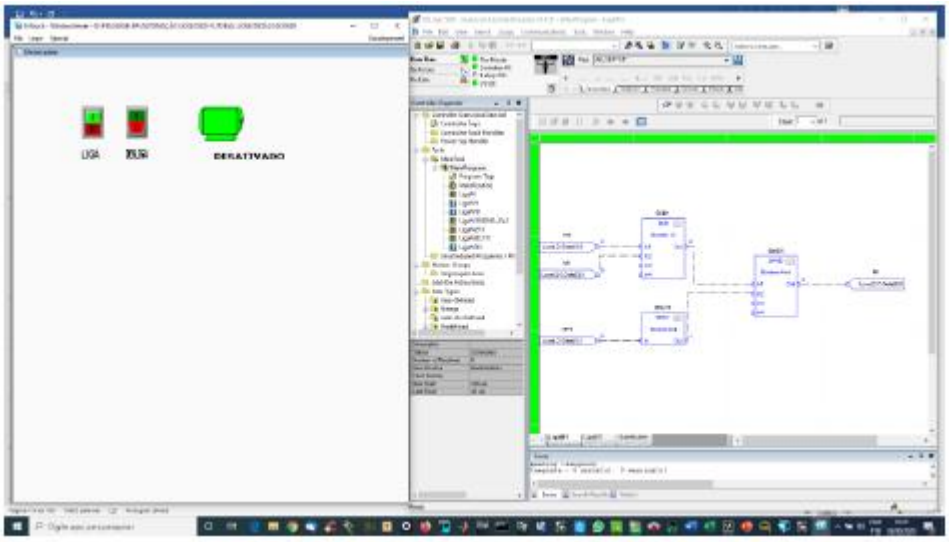
A seguir, efetua-se novamente o download do programa LigaM1, colocando-o em modo de execução, juntamente com seu respectivo sistema supervisório, para observar o funcionamento do conjunto, conforme figuras 226 e 227, onde se observam as situações para os botões liga e desliga, respectivamente acionados através da tela do sistema supervisório. Percebe-se a validade das explicações anteriormente realizadas, com relação ao comportamento da rotina.

Figura 226 – Ativação do programa LigaM1 através do sistema supervisório



Fonte: Própria

Figura 227 – Desativação do programa LigaM1 através do sistema supervisório



Fonte: Própria

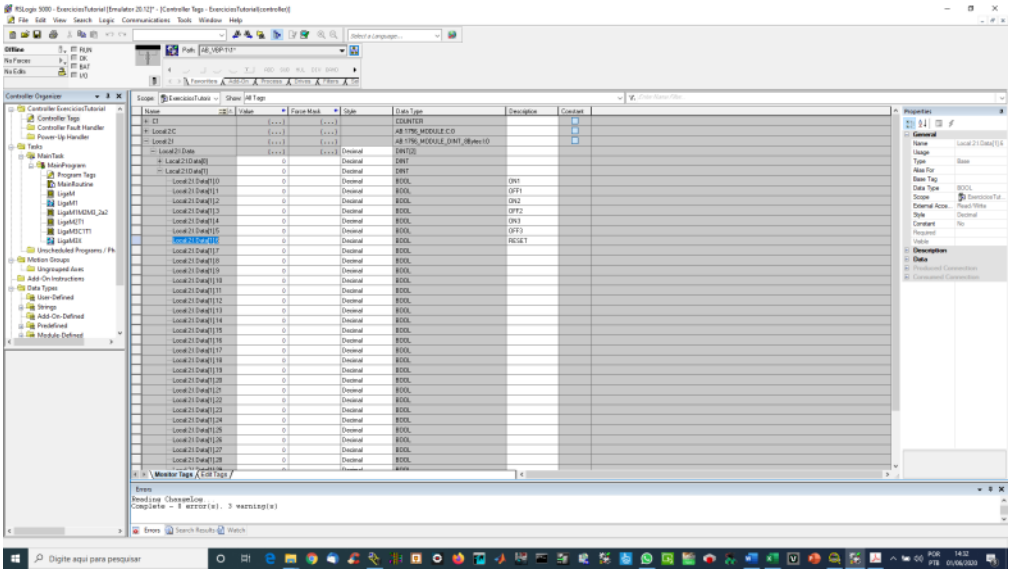
Entretanto, cabe salientar que para os problemas de automação tratados neste livro, não será preciso atribuir às tags de blocos de funções escopo global, cuja abrangência envolve todo o projeto de automação desenvolvido no RSLogix 5000.

4) Pede-se alterar o código do programa desenvolvido no exercício 1, para que o mesmo possibilite RESET manual do bloco contador.

Para cumprir os pedidos do enunciado, é preciso criar *tagname* ou variável, que seja responsável por endereçar o botão relacionado ao *RESET* do contador. Há duas formas de criar *tagnames*. A primeira delas será relembrada. Para tanto, clica-se duas vezes sobre *Controller Tags*, de acordo com a figura 228, onde se localiza o primeiro endereço de entrada do tipo *BOOL* (booleano ou discreto) disponível, que é Local:2:I.Data[1].6, inserindo no campo *Description* do mesmo a palavra *RESET*, para associá-lo à entrada *Reset* da instrução *CTUD*, no programa desenvolvido em diagrama de blocos de funções.

A vantagem de criar *tagnames* desta forma é que as mesmas terão escopo global no projeto. O proximo passo consiste em copiar o endereço do botão *RESET* (Local:2:I.Data[1].6) da tabela de tags do controlador (*Controller Tags*).

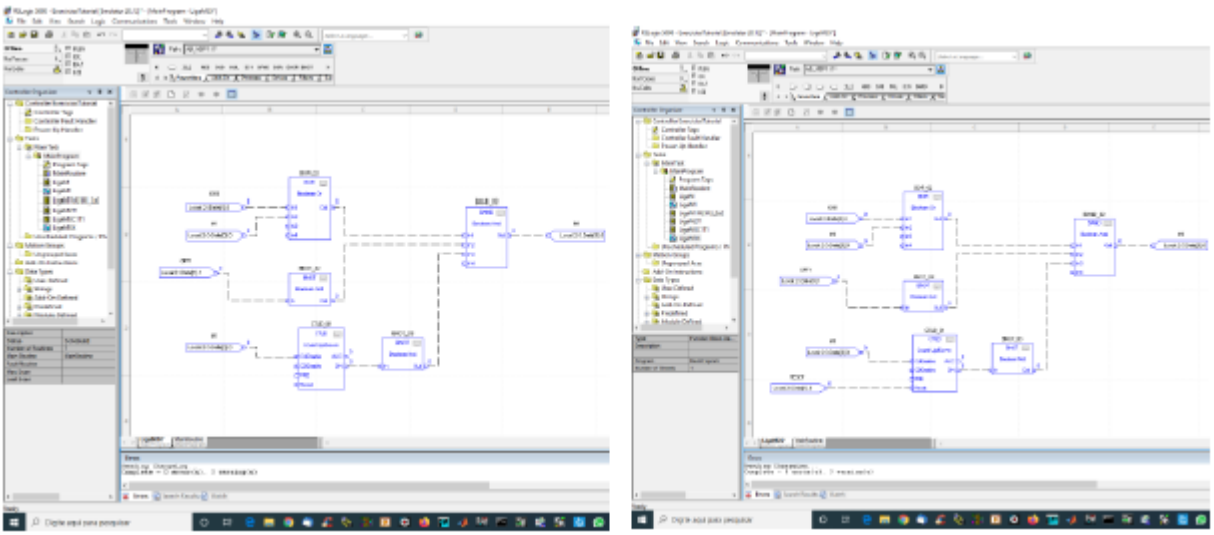
Figura 228 – Criação da *tagname* Reset



Fonte: Própria

Em seguida, abre-se o programa LigaM3X clicando-se duas vezes sobre o mesmo, para inserir o bloco Input Reference (lembrando-se de digitar em seu interior o endereço Local:2:I.Data[1].6) na entrada Reset do contador, de acordo com as ilustrações da figura 229.

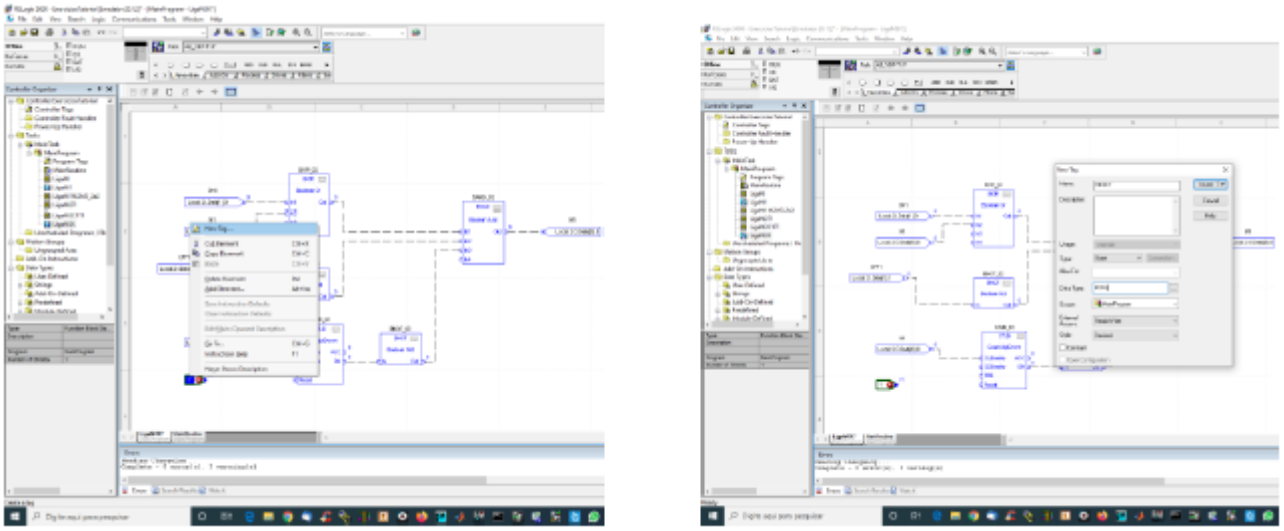
Figura 229 – Inserindo entrada de referência para o botão Reset



Fonte: Própria

A segunda forma de criar uma entrada *RESET* começa por inserir um bloco referência de entrada, clicar com o botão direito do mouse sobre ele e, no quadro de opções que surgir, selecionar *New Tag*, conforme primeira ilustração da figura 230.

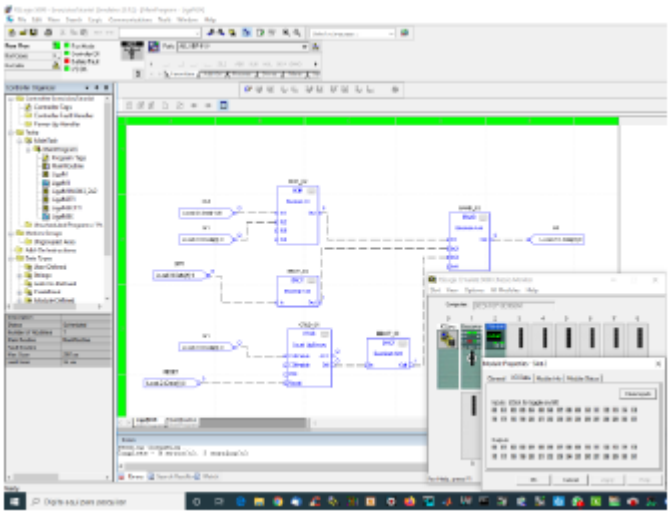
Figura 230 – Inserindo entrada de referência para o botão Reset



Fonte: Própria

Após selecionar a opção *New Tag* na figura 230, surgirá a segunda tela da mesma, onde se atribuem à nova variável nome, tipo e escopo, que são *RESET*, *BOOL* e *MainProgram*, respectivamente. Isso significa que a *tagname RESET* poderá assumir somente dois valores, 0 ou 1, conforme base numérica binária, sobre a qual se fundamenta a lógica discreta. Além disso, o escopo de RESET será restrito aos programas elaborados, não tendo âmbito global no projeto, o que não ocorreria se esta *tagname* fosse elaborada de acordo com a primeira forma, anteriormente explanada. Para homologar a criação da tag, basta pressionar o botão create.

Figura 231 – Execução do programa LigaM3X através do RSLogix Emulate 5000

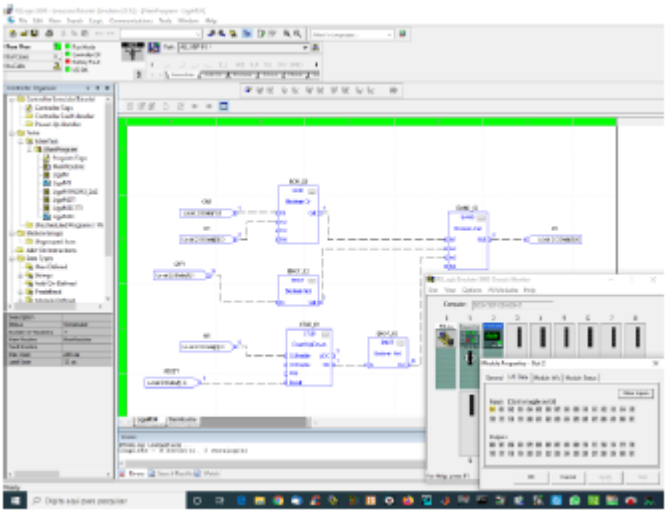


Fonte: Própria

Após as modificações, o código adquirirá a configuração da figura 231, acompanhada da inclusão de nova entrada, e das propriedades do módulo associadas ao slot 2, no RSLogix 5000 *Emulate,* para simular a execução do programa LigaM3X.

De acordo com o programa, o motor é automaticamente desativado na terceira vez em que entra em operação, pois a saída DN do contador recebeu nível lógico alto nesta situação, uma vez que o conteúdo de ACC alcançou o valor de três acionamentos, especificados no campo PRESET do contador, conforme figura. Isso ocorre porque entre a saída DN do contador e a entrada da porta BAND há um inversor BNOT, que converte o nível alto gerado ao final da contagem para zero. A tabela verdade da operação lógica E determina que a saída seja zero quando pelo menos uma das entradas estiver em nível lógico baixo, situação que ocorre ao final da contagem, confome figura 232.

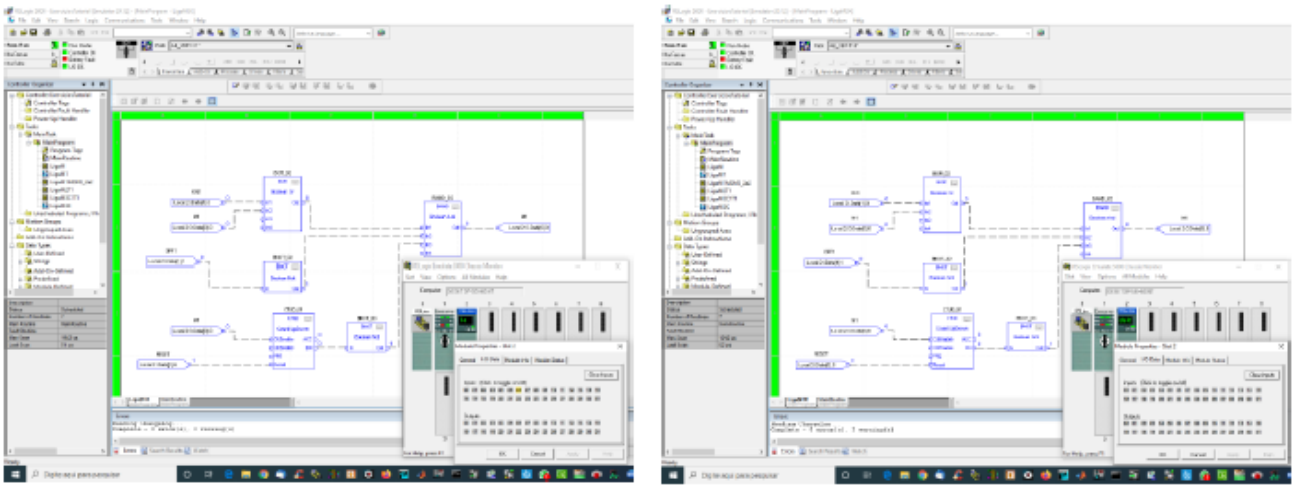
Figura 232 – Primeiro acionamento do motor



Fonte: Própria

Para restabelecer a possibilidade de acionar o motor, é preciso desativar a saída DN do contador, situação que ocorre com a ativação de sua entrada Reset, que zera os valores do campo ACC. Consegue-se essa situação clicando-se sobre o botão 00, para desativar a entrada liga e, em seguida, pressionando-se o botão 06, realizando o Reset do contador. Por último, desativa-se novamente a entrada 06, do contrário o contador permanecerá em estado de reinicialização, sem efetuar contagem alguma (fig. 233).

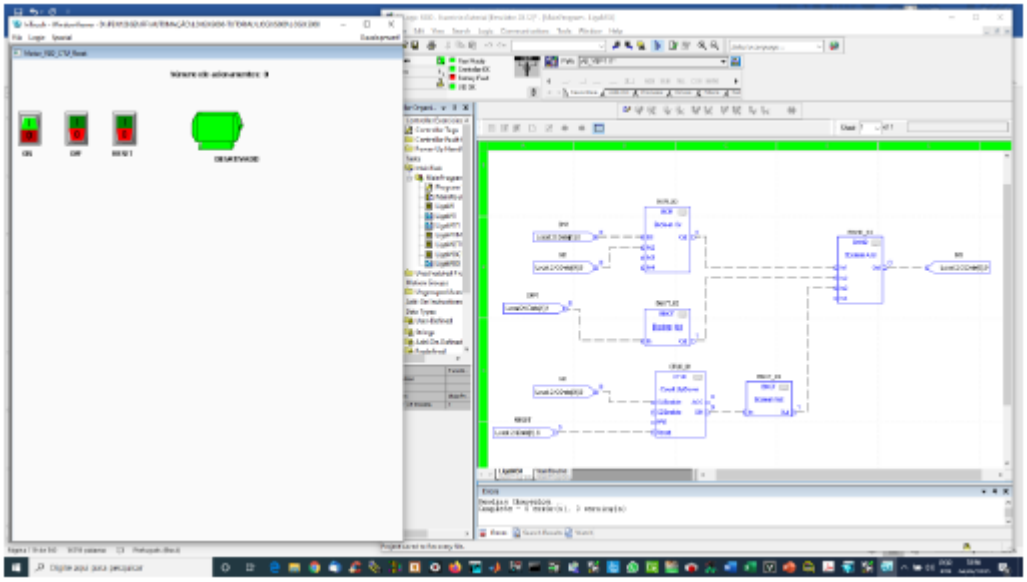
Figura 233 – Ativação da entrada Reset



Fonte: Própria

5) Solicita-se elaborar sistema supervisório para o problema anterior, identificado como o quarto dos exercícios integrantes desta seção, referente à programação em diagrama de blocos de função (FBD), conforme tela da figura 234.

Figura 234 - Sistema supervisório controlando o código do processo em FBD



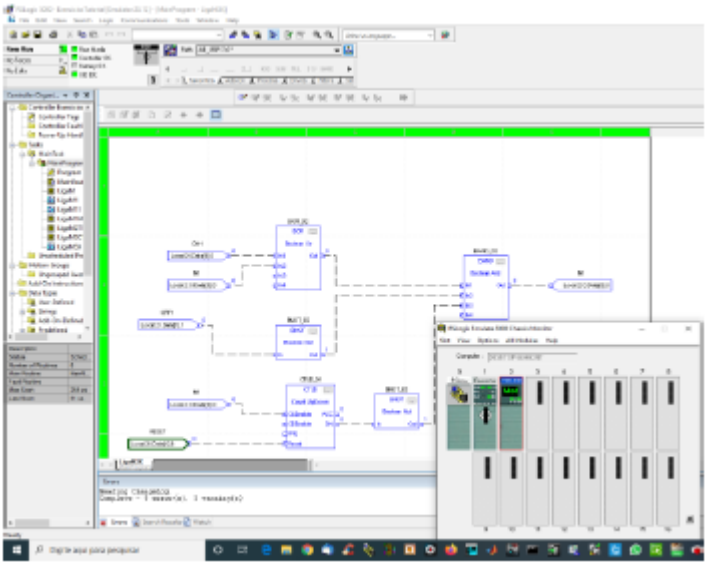
Fonte: Própria

Antes de iniciar o desenvolvimento do sistema supervisório pedido, é preciso criar um tópico no software RSLinx, para compartilhar dados entre os ambientes de desenvolvimento integrados (*IDE* – acrônimo para *Integrated Development Environment*) utilizados através do protocolo de intercâmbio

dinâmico de dados (*DDE* – acrônimo para *Dynamic Data Exchange*), que é nativo do sistema operacional *Windows*.

Para tal fim, é preciso deixar o programa em execução no IDE RSLogix 5000, conforme figura 235, juntamente com o RSLogix 5000 Emulate.

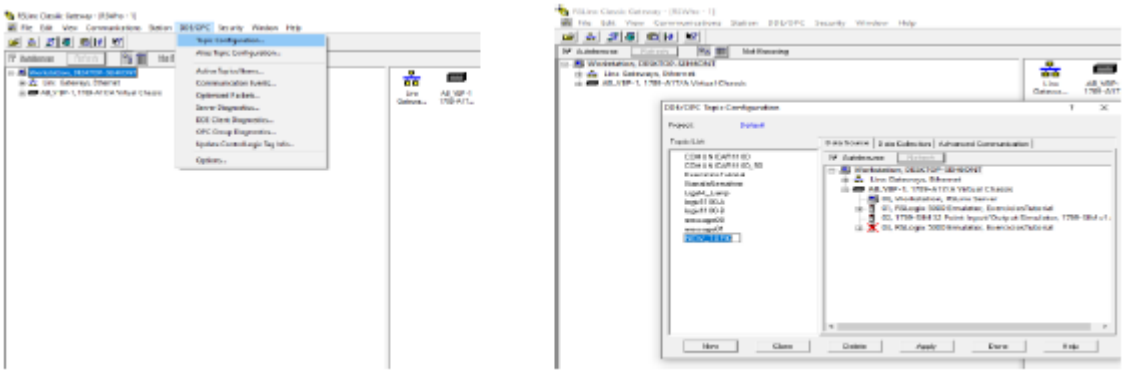
Figura 235 – Programa em execução com o RSLogix 5000 Emulate



Fonte: Própria

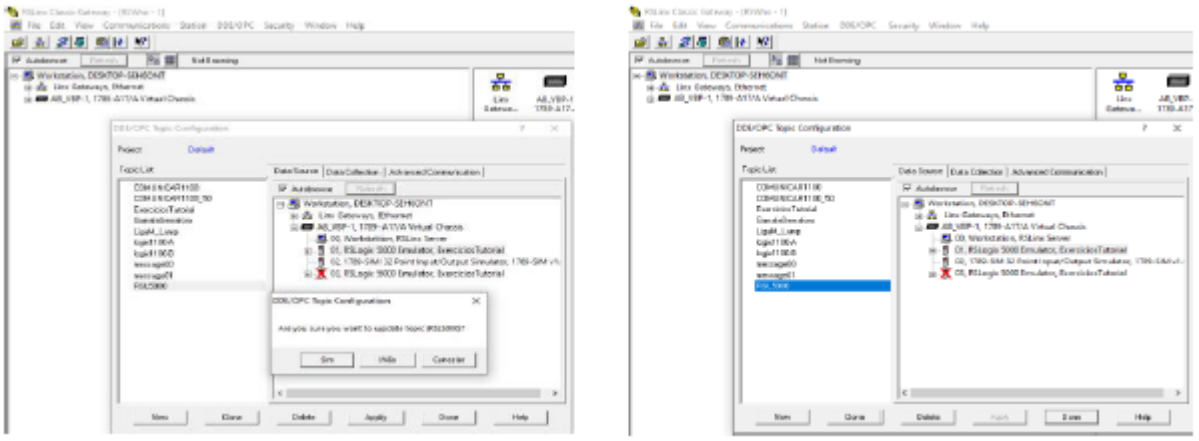
A seguir, maximixa-se o RSLinx, selecionando as opções DDE/OPC e Topic Configuration, de acordo com a figura 236, o que originará a segunda tela da mesma, na qual se clica sobre o botão New, para criar novo tópico (NEW_TOPIC).

Figura 236 – Configurando tópico de comunicação com o sistema supervisório



Fonte: Própria

Figura 237 – Configurando tópico de comunicação com o sistema supervisório

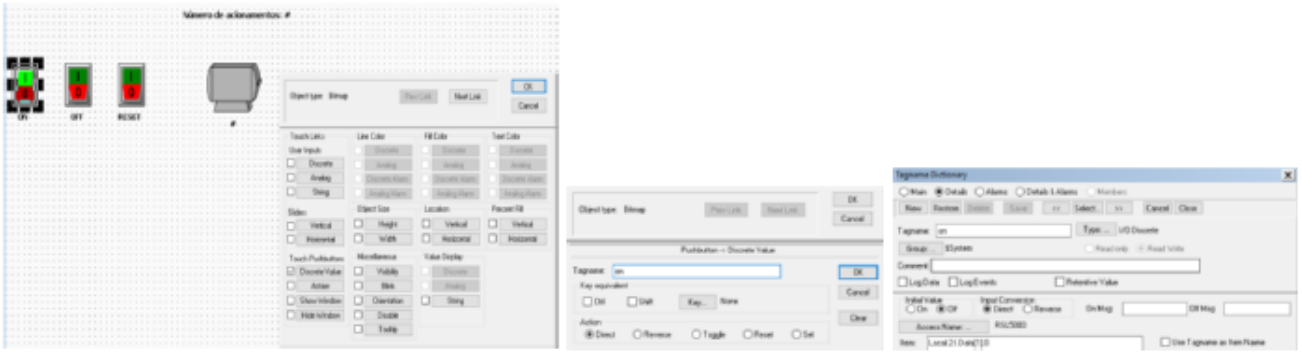


Fonte: Própria

Modifica-se o nome NEW_TOPIC para RSL5000, aplicando-o em seguida ao nó 01, associado às informações RSLogix 5000 Emulator, ExercíciosTutorial (fig. 237). Em seguida, pressiona-se o botão *Done* na tela2 da figura, minimiza-se o RSLinx e abre-se o sistema supervisório In Touch, para elaborar a supervisão do processo.

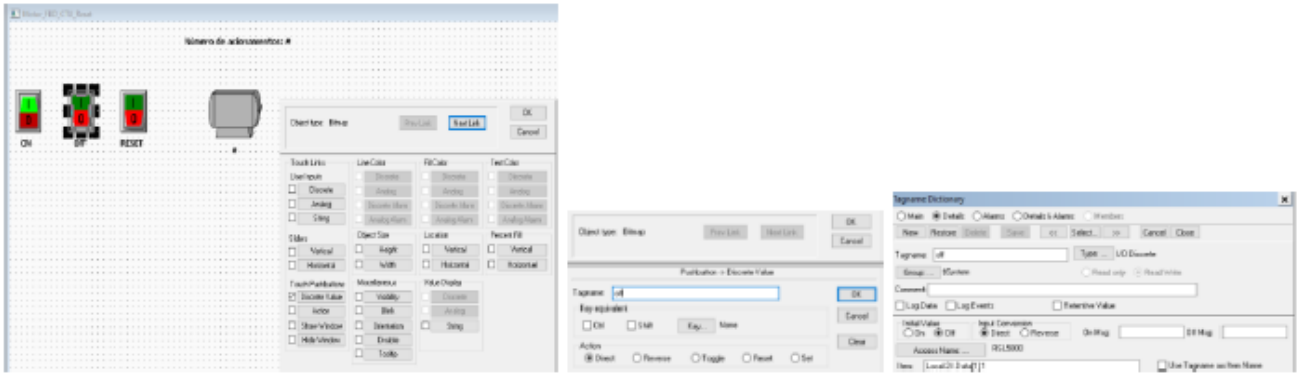
O sistema supervisório compreende três botões, para ligar o motor, desativar a máquina e reiniciar o contador zerando o conteúdo do campo acumulador de acionamentos (*ACC*), respectivamente designados por *ON*, *OFF* e *RESET*. Para esses três botões se comunicarem com as referências de entradas (*ON1*, *OFF1* e *RESET*) da rotina *FBD*, nomeada LigaM3X, é preciso que eles tenham suas respectivas propriedades *Touch Pushbutton – Discret value* controladas por *tagnames* cujo endereço seja o mesmo daqueles blocos, ou seja, Local:2:I.Data[1].0 (*ON1*), Local:2:I.Data[1].1 (*OFF1*) e Local:2:I.Data[1].6 (*RESET*), conforme figuras 238, 239 e 240.

Figura 238 – Configurando botão ON no sistema supervisório



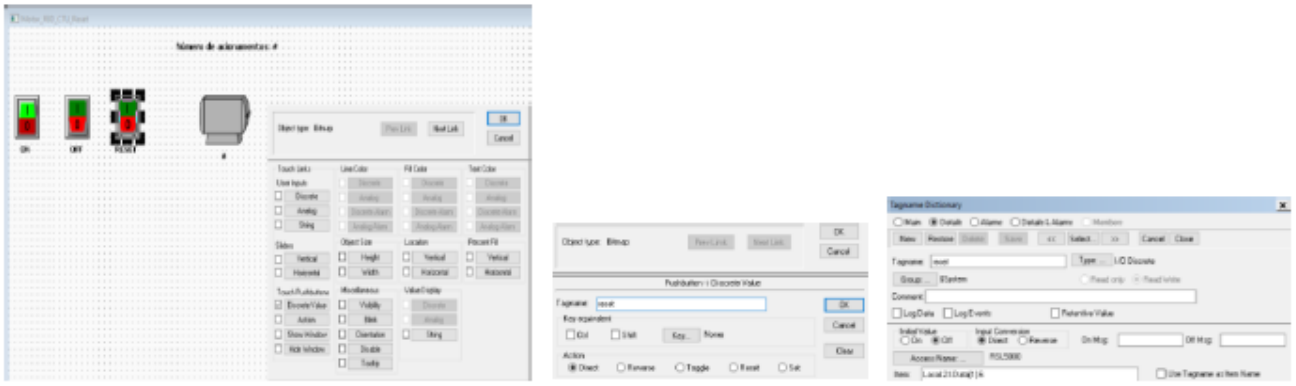
Fonte: Própria

Figura 239 – Configurando botão OFF no sistema supervisório



Fonte: Própria

Figura 240 – Configurando botão RESET no sistema supervisório

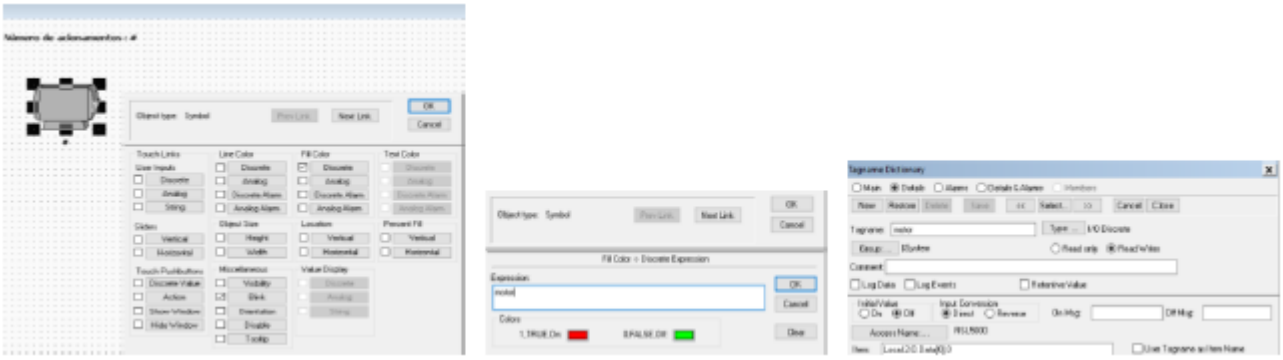


Fonte: Própria

Além disso, as ações das *tagnames* associadas às propriedades *Touch Pushbutton – Discret value* devem ser do tipo *Direct*, indicando comportamento análogo a botões do tipo impulso quando se clicar sobre eles. Isso significa que os contatos virtuais do programa *FBD* (cujos endereços são descritos no campo Item das *tagnames* do sistema supervisório) comutarão de estado somente enquanto seus respectivos botões da tela supervisora estiverem pressionados, comportando-se de forma análoga a campainhas domésticas.

A próxima etapa consiste em analisar as configurações do motor, que podem ser vistas nas figuras 241 e 242.

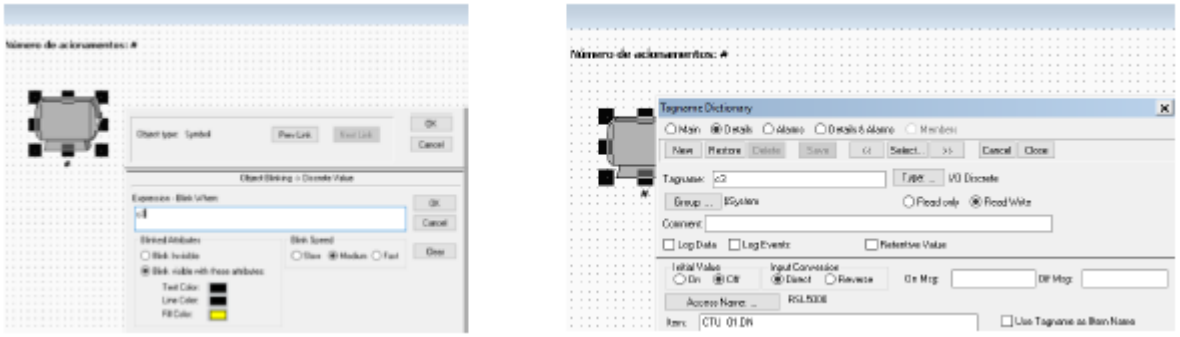
Figura 241 - Configurações da propriedade *Fill Color Discrete*



Fonte: Própria

Percebe-se nas ilustrações da figura 241 que somente as propriedades *Fill Color Discrete* e *Miscellaneous Blink* estão associadas ao objeto, enquanto se estabeleceu a *tagname* motor como variável de controle para o preenchimento automático de suas cores. A cor do motor será vermelha enquanto o valor de sua *tagname* homônima for zero ou falsa, conferindo ao mesmo o estado desativado. Por outro lado, a cor do motor será verde enquanto o valor de sua *tagname* homônima for um ou verdadeiro, conferindo ao mesmo o estado ativado. Observam-se as propriedades da *tagname* motor na terceira ilustração, definida como tipo I/O Discrete, Access Name RSL5000 e Item Local:2:O.Data[0].0, indicando que a variável recebe o endereço da bobina de saída no programa.

Figura 242 - Configurações da propriedade Miscellaneous Blink



Fonte: Própria

A figura 242 define a tagname c3 como a expressão que controla a propriedade *Miscellaneous Blink* do motor. Dessa forma, quando a variável c3 for verdadeira ou estiver em nível lógico alto (*On*), o motor piscará com a cor de preenchimento amarelo.

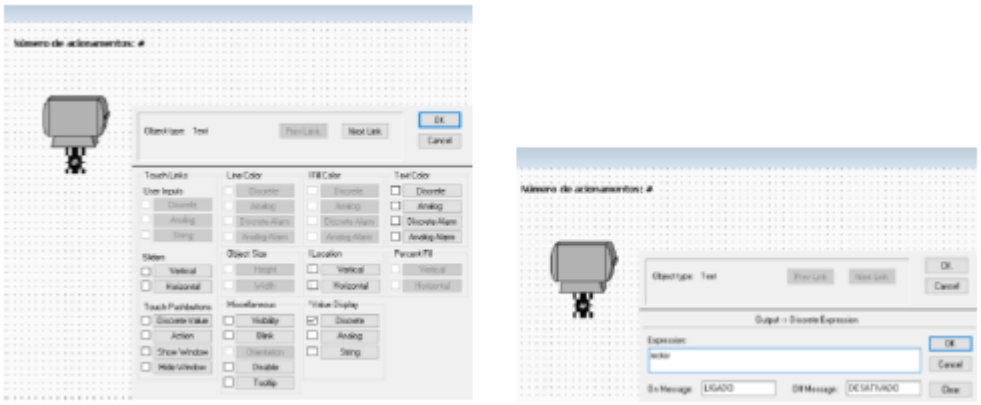
A segunda ilustração da figura 242 apresenta as definições da *tagname* c3, que é do tipo *I/O Discrete*, como nome de acesso RSL5000 e endereço especificado no campo Item por CTU_01.DN.

Dessa forma, c3 terá em nível lógico alto (*On*) quando o bloco contador detectar que seu registro de contagens (*ACC*) alcançou o valor estabelecido em *PRESET* no programa *FBD* LigaM3X, o que acarretará na comutação da saída DN para o valor discreto igual a 1. Nessas condições c3 se torna verdadeiro, porque seu endereço é o mesmo da saída DN do contador, fazendo com que o motor pisque com a cor amarela, até que o botão reset seja pressionado, limpando o valor acumulado pelo contador. Com isso, DN retornará ao valor zero ou desativado, porporcionando condições para novos acionamentos do motor.

Cabe destacar que a partir do momento em que se cria e atribuí determinado nome de Accesso a um objeto numa tela do supervisório *In Touch*, os demais elementos inseridos receberão automaticamente este mesmo *Access Name*.

Por último, clica-se duas vezes para acessar as configurações do objeto cerquilha, originando a primeira ilustração da figura, onde se percebe a propriedade *Value Display Discrete* como a única selecionada. Ao pressionar o botão *Discrete* assinalado na propriedade da cerquilha, aparecerá a segunda tela da figura 243, onde se define como expressão a tagname motor, que definirá o texto a ser exibido. Quando o motor estiver em operação, a cerquilha exibirá a mensagem LIGADO. Por outro lado, a cerquilha exibirá a mensagem DESATIVADO enquanto o motor estiver inoperante.

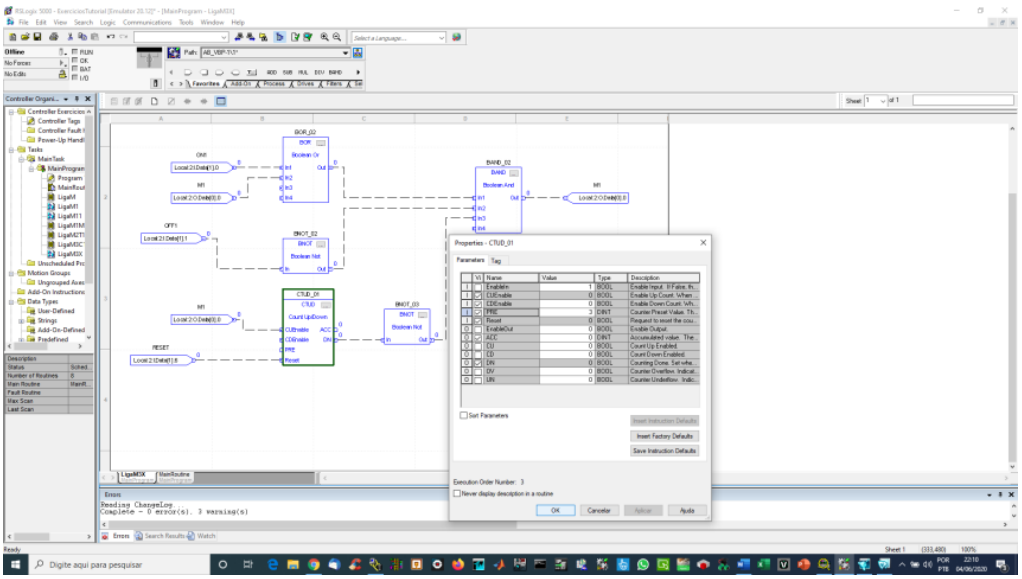
Figura 243 – Configurações do objeto cerquilha e sua propriedade Value Display Discrete



Fonte: Própria

Cabe ressaltar a importância de se definir um limite para contagem na instrução CTUD, pois um dos erros de programação comuns é justamente o esquecimento de se estabelecer até que ponto o bloco contador registrará transições de borda de subida (no caso crescente) ou descida (no caso descrescente) em sua entrada. Para tanto, clica-se duas vezes sobre a instrução contadora no programa LigaM3X, procedendo-se ao ajuste do *Preset*, ou limite de contagem, digitando o número 3. Isso permitirá 2 acionamentos do motor, desligando-o automaticamente na terceira tentativa de ativação (fig. 244).

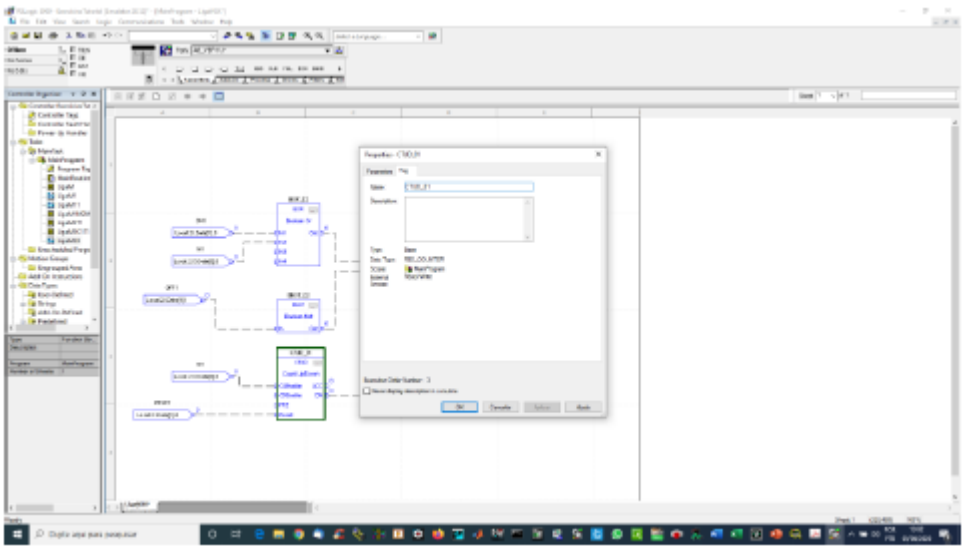
Figura 244 – Ajuste do campo Preset na instrução CTUD



Fonte: Própria

Outra consideração importante sobre o programa LigaM3X diz respeito ao escopo da tag CTUD_01, cujo caráter é local, ou seja, engloba somente o conjunto de programas disponibilizados na aba MainProgram (fig. 245).

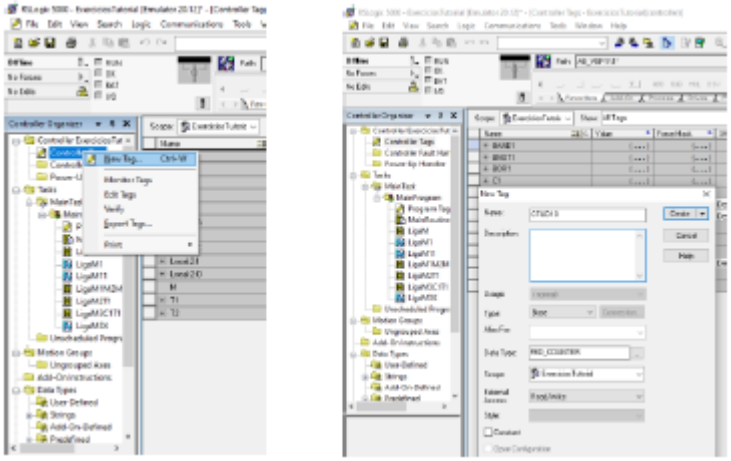
Figura 245 – Escopo local da tag CTUD_01



Fonte: Própria

O problema de *tags* com escopo local é que algumas de suas propriedades não são disponibilizadas para troca de informações através do protocolo *Dinamyc Data Exchange* (*DDE*), nativo do sistema operacional *windows*, comprometendo a atuação de sistemas supervisórios sobre programas destinados a automação de processos.

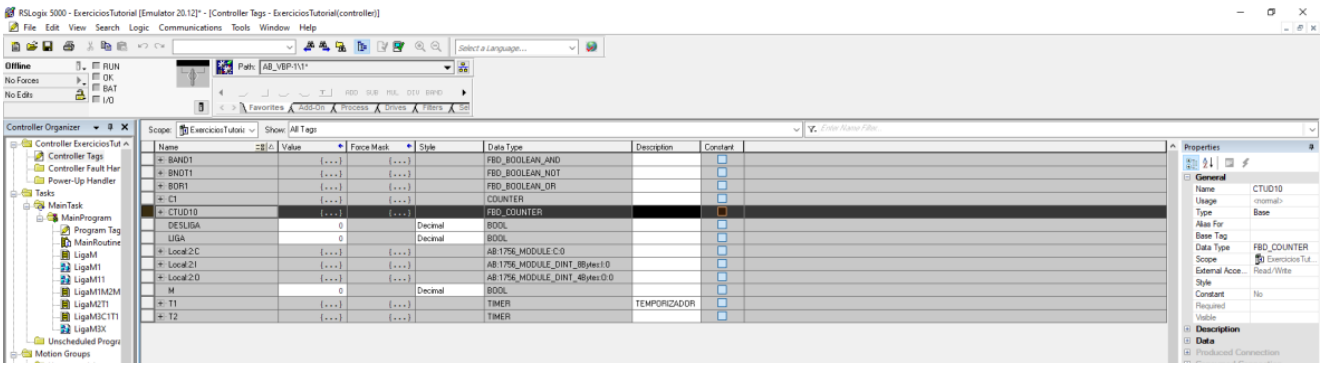
Figura 246 – Criação de tag contador com escopo global em blocos de funções (*FBD*)



Fonte: Própria

A solução consiste em estabelecer variável contadora com escopo global. Para tanto, clica-se com o botão direito sobre *Controller Tags* e, no quadro de opções que surgir, selecionar *New Tag*, conforme figura 246, cuja segunda ilustração traz as configurações atribuídas à *tag* criada, que são: *Name* (Nome – CTUD10), *Data_Type* (Tio de dados – FBD_COUNTER) e *Scope* (Escopo – ExercíciosTutorial, que é o nome completo do projeto criado no RSLogix 5000. Após clicar sobre o botão *Create*, a lista de *tags* do controlador ficará de acordo com a figura 247.

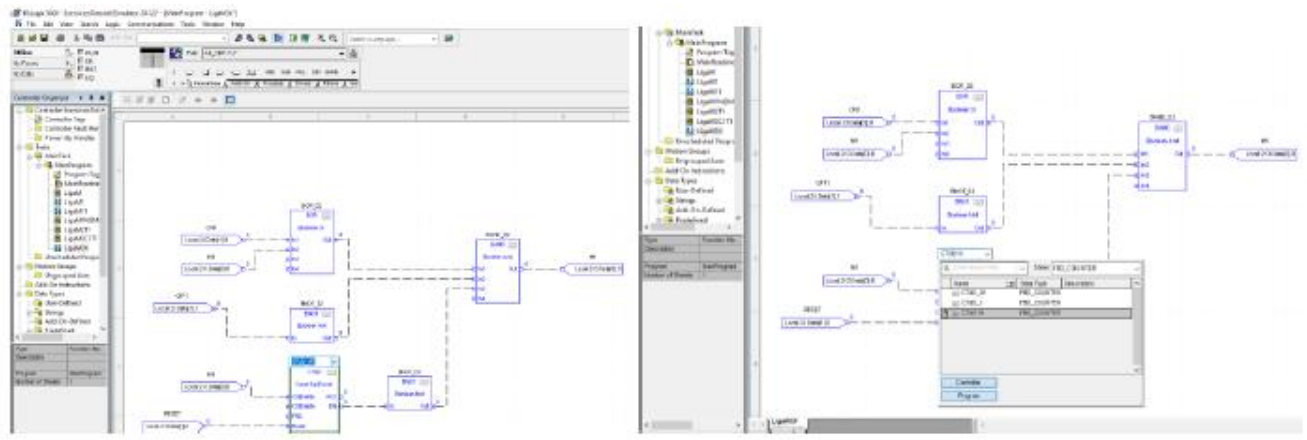
Figura 247 – Lista de Tags do Controlador



Fonte: Própria

O próximo passo consiste em abrir o programa FBD LigaM3X, clicar duas vezes sobre a tag associada ao contador, CTUD_01, abrir o Menu de tags e substituir a mesma pela variável de escopo global recém criada, que é CTUD10, segundo figura 248.

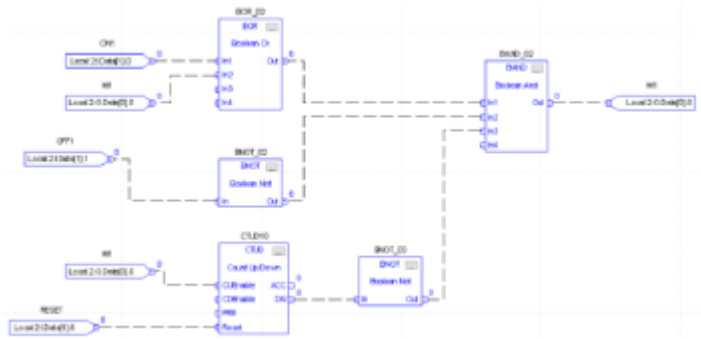
Figura 248 – Substituição da tag do bloco contador



Fonte: Própria

Dessa forma, o programa FBD LigaM3X adquire o aspecto da figura 249.

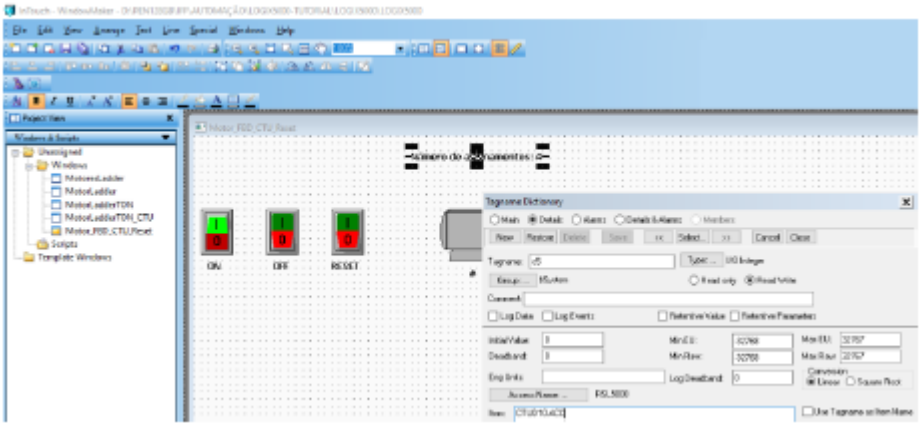
Figura 249 – Programa LigaM3X com tag de escopo global CTUD10



Fonte: Própria

A próxima etapa consiste em modificar as configurações dos objetos integrantes do sistema supervisório, cujos endereços de *tagnames* referenciavam a variável de escopo local (CTUD_01), atualizando as referências para a tag contadora de escopo global, criada no programa FBD (CTUD10). Para tanto, clica-se duas vezes sobre a cerquilha responsável por informar o número de acionamentos do motor, para selecionar a propriedade *Value Display Analog*, e acessar as definições da *tagname* c5 que a controla, modificando o endereço do Item para CTUD10.ACC (fig. 250).

Figura 250 - Atualização do endereço do Item da *tagname* c5 para CTUD10.ACC



Fonte: Própria

Em seguida acessa-se a propriedade *Miscellaneous Blink* clicando-se duas vezes sobre o motor, com o intuito de atualizar sua expressão de controle, conforme primeira ilustração da figura 251. Percebe-se que o motor começará a piscar com o fundo amarelo quando a *tagname* c5 (que registra o número de acionamentos do motor) for igual à *tagname* c33, cujas configurações são apresentadas na segunda ilustração.

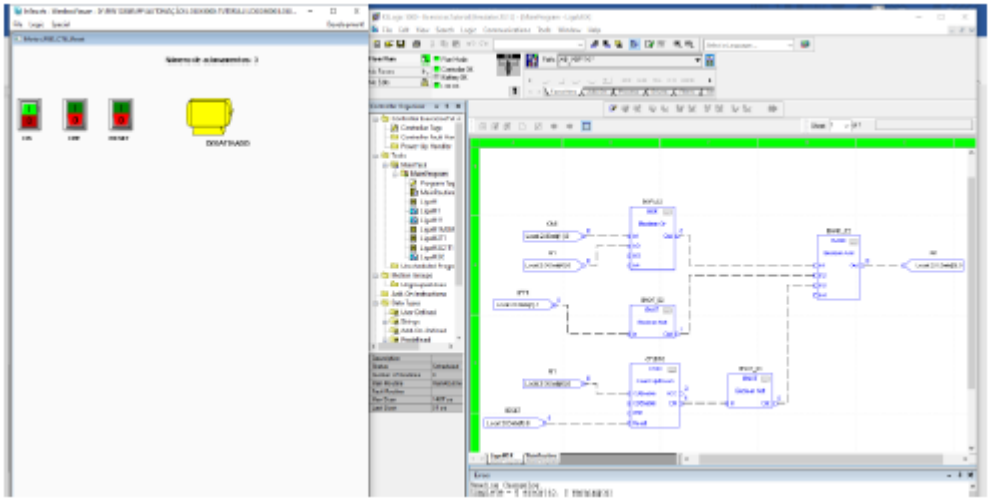
Figura 251 – Atualização da propriedade *Miscellaneous Blink* do motor



Fonte: Própria

A tagname c33 é do tipo I/O Integer, indicando que se comunicará com elemento do programa FBD que trabalha com dados inteiros, cujo endereço é especificado no campo Item como CTUD10.PRE, o qual armazena o conteúdo do campo PRESET do bloco de contagem. Dessa forma, sempre que o conteúdo do acumulador armazenado na variável c5 for igual ao valor do campo PRESET, acondicionado na tagname c33, a expressão da figura 249 será verdadeira, fazendo com que a cor de preenchimento do motor comece a piscar com o fundo amarelo. Essa situação é observada na figura 252, onde a contabilização de acionamentos chegou ao máximo definido (três), desligando automaticamente o motor.

Figura 252 – Funcionamento do motor com três acionamentos

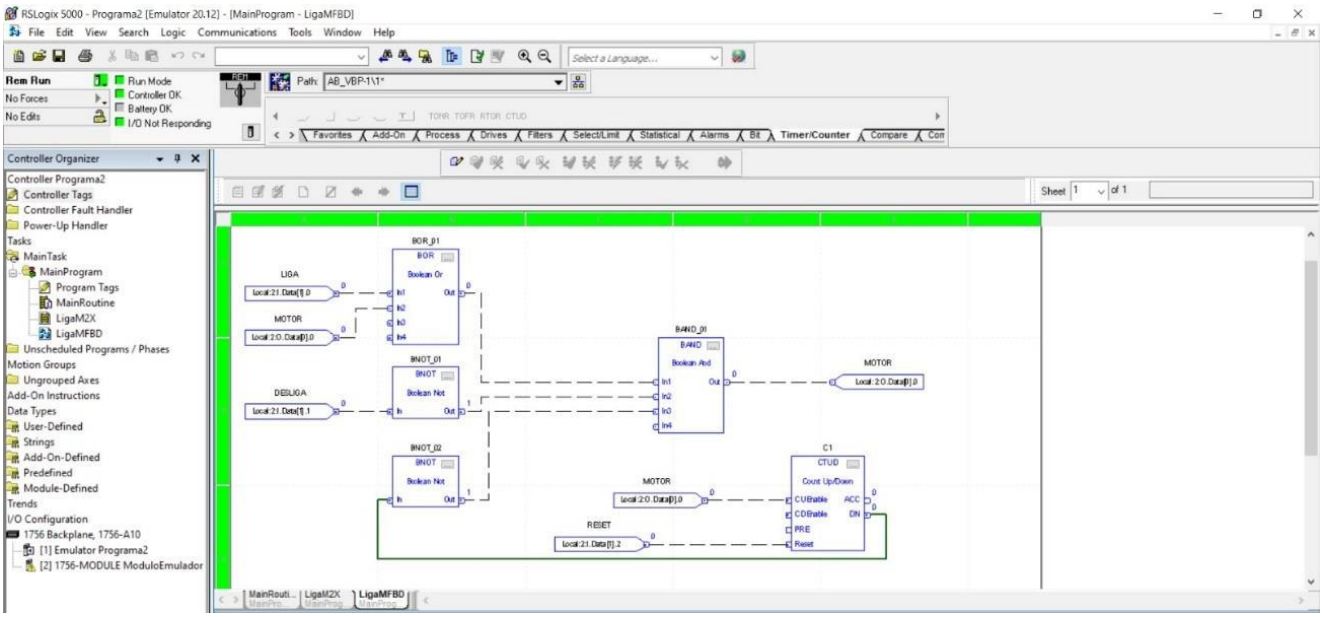


Fonte: Própria

O sistema supervisório poderá controlar novamente o processo quando for pressionado o botão Reset, para zerar o campo ACC no contador do programa FBD. Isso zerará a saída DN, fazendo com que o bloco inversor BNOT_03 realize a inserção de nível lógico alto na terceira entrada do bloco BAND_02, viabilizando novos acionamentos do motor através do botão ON.

Segue solução alternativa para esta programação, vista na figura 253.

Figura 253 – Acionamento motor limitado por duas vezes em FBD

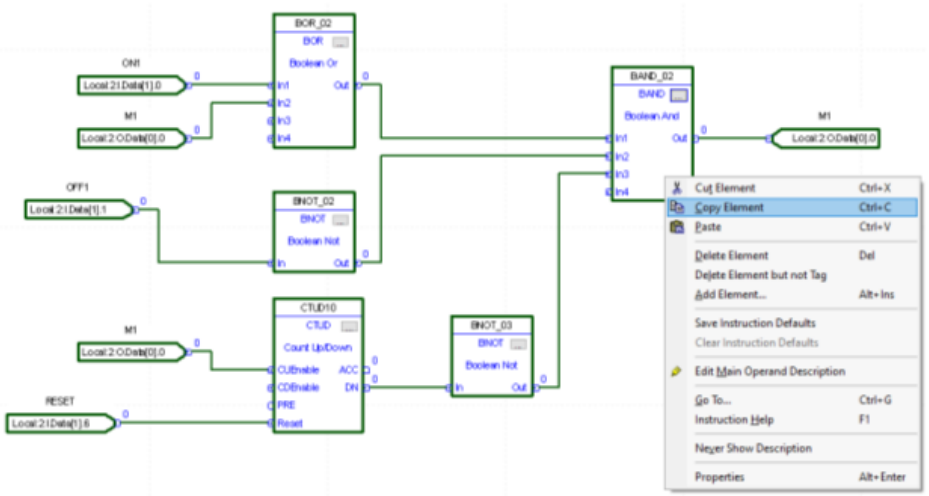


Fonte: Própria

6) Fazer alterações no programa anterior, para que haja obrigatoriamente um intervalo mínimo de 7 segundos entre cada acionamento.

A solução será elaborada a partir do código anterior, onde são selecionadas todas as suas instruções através do mouse, para realizar a cópia das mesmas (fig. 254).

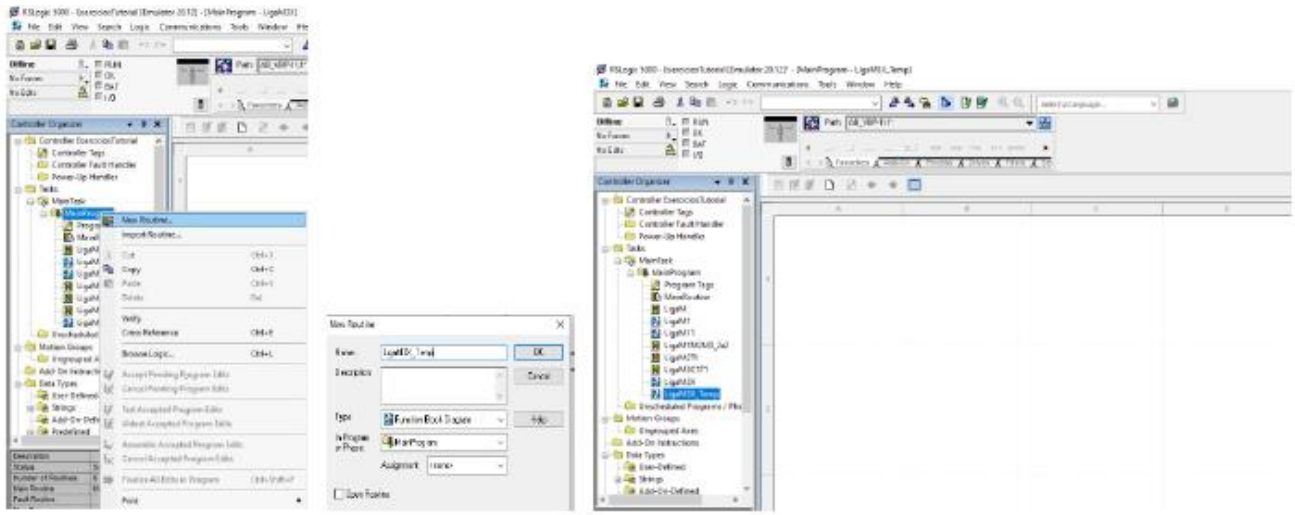
Figura 254 – Cópia das instruções de LigaM3X



Fonte: Própria

A seguir, clica-se com o botão direito sobre Main Program, para criar nova rotina em diagrama de blocos de funções, conforme primeira ilustração da figura 255. Na tela seguinte, descreve-se o nome da rotina (LigaM3X_Temp) e o tipo de linguagem nela utilizada (*Function Block Diagram*). Após confirmar a criação da rotina, basta clicar duas vezes sobre ela, para acessar sua página de código em branco, cujo conteúdo será preenchido na fig. 254, após colar o código copiado para a memória, da rotina LigaM3X.

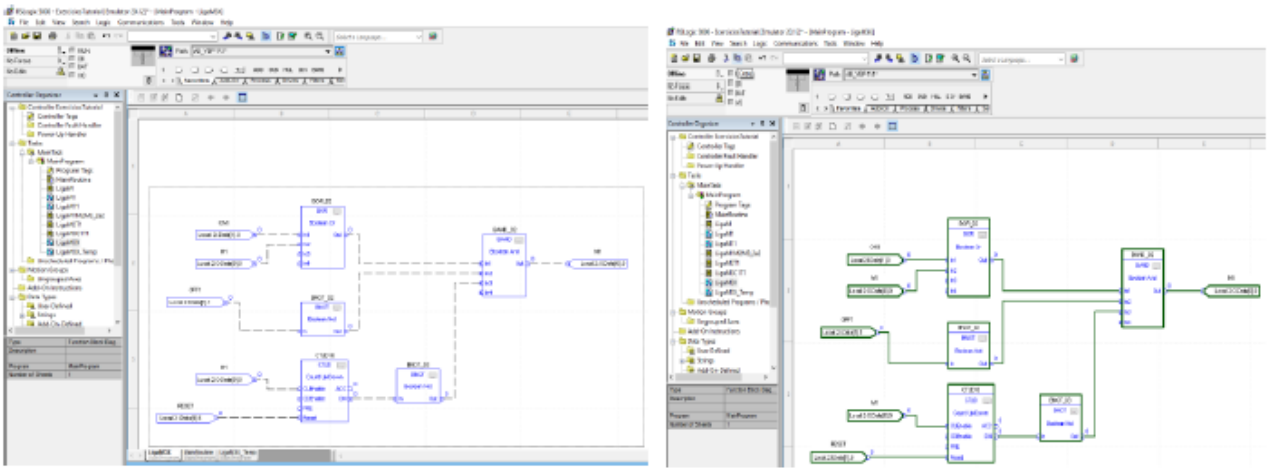
Figura 255 – Criação da rotina LigaM3X_Temp



Fonte: Própria

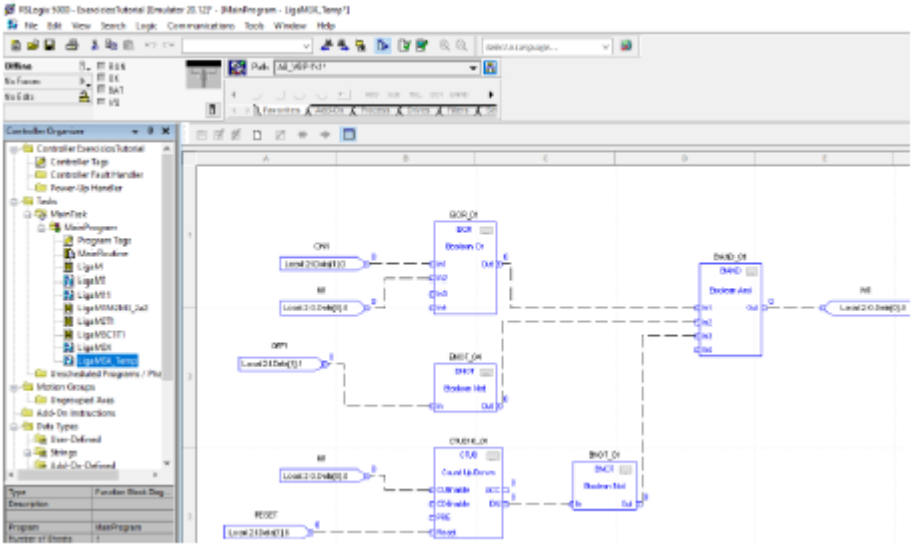
A figura 256 apresenta forma alternativa para copiar conteúdo do código da rotina LigaM3X, que ocorre após seleção dos blocos pelo mouse, e clicar sobre o botão Copy. A figura 257 mostra o código de LigaM3X_Temp com instruções coladas de LigaM3X.

Figura 256- Forma alternativa para copiar conteúdo de código *FBD*



Fonte: Própria

Figura 257 – Código de LigaM3X_Temp com instruções coladas de LigaM3X

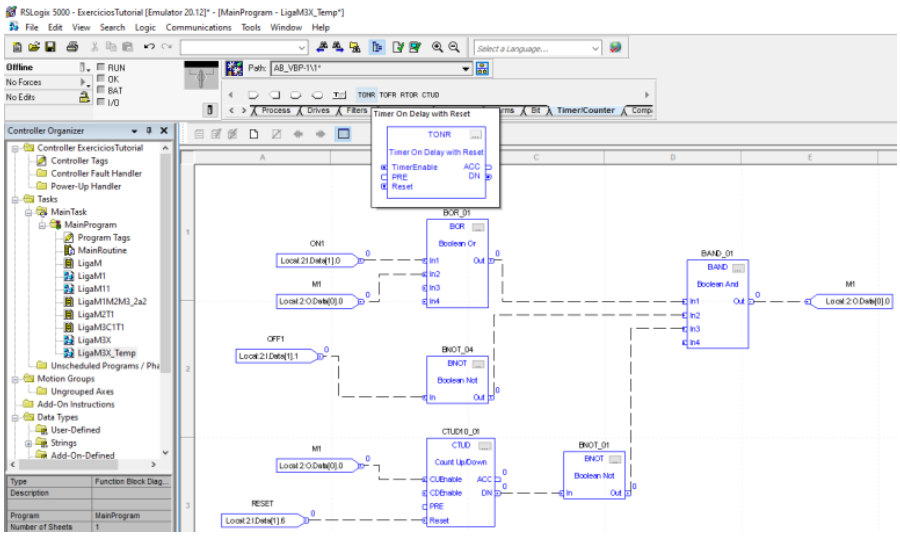


Fonte: Própria

Para resolver o que se pede no enunciado deste exercício, é preciso adicionar um bloco temporizador, conforme figura 258, obtida na aba *Timer*/*Counter*.

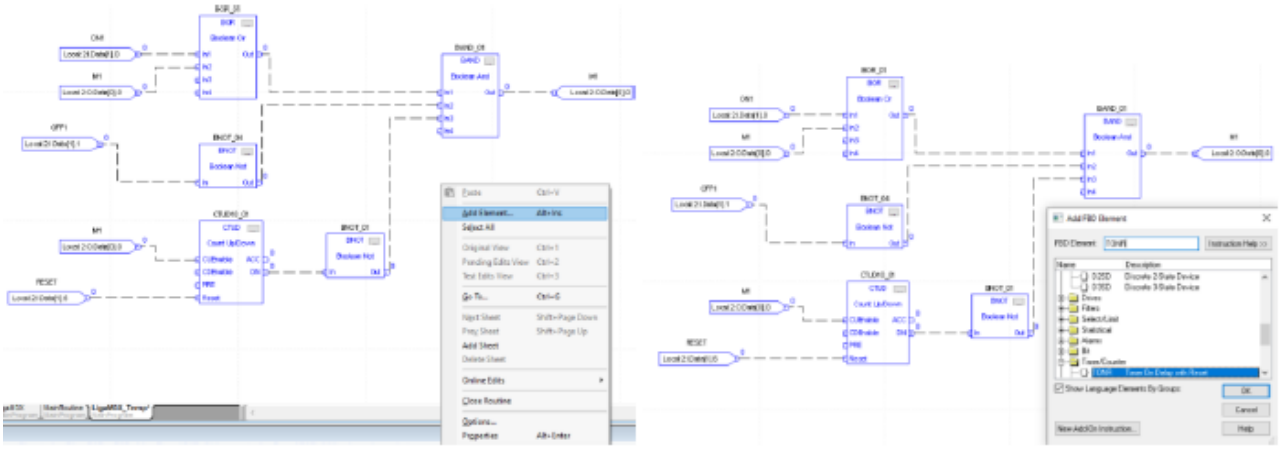
Outra forma de adicionar instruções consiste em clicar com o botão direito do mouse sobre a folha de programação, de acordo com a figura 259, e selecionar a opção *Add Element* e, no menu que surgir, digitar o nome do bloco que se deseja adicionar, confirmando a inserção do mesmo após pressionar o botão OK.

Figura 258 – Inserção da instrução TONR



Fonte: Própria

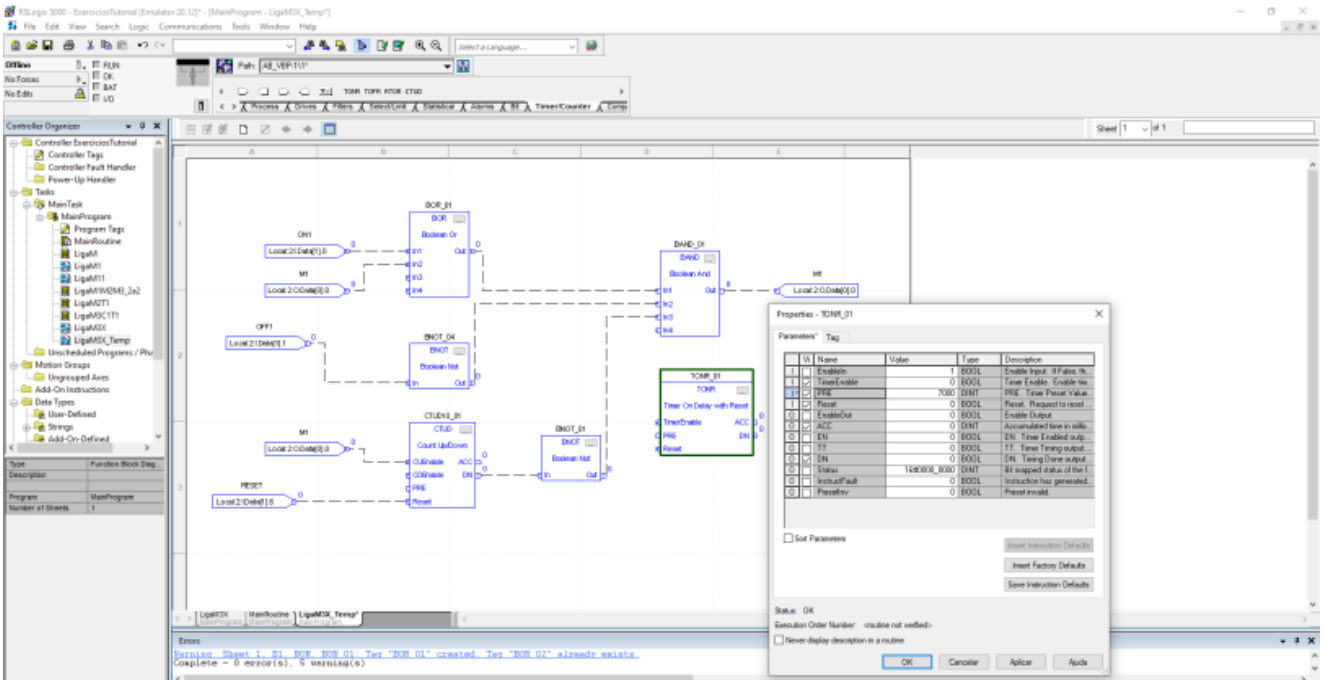
Figura 259 – Outra maneira de inserir a instrução TONR



Fonte: Própria

Clicando-se duas vezes sobre o bloco TONR é possível ajustar o preset da instrução, digitando 7000 milissegundos sobre o mesmo (figura 260), para determinar o limite máximo para cronometrar o tempo decorrido, após o qual se ativará a saída DN.

Figura 260 – Ajuste do preset de TONR

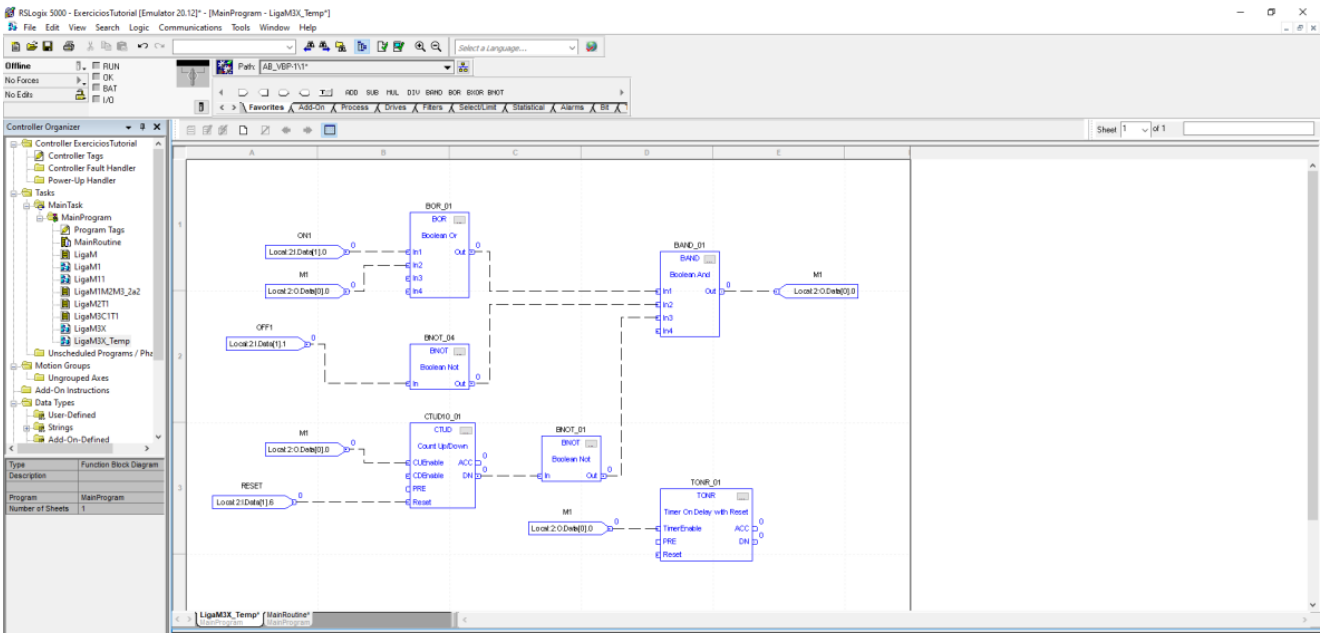


Fonte: Própria

A partir deste ponto, insere-se uma entrada de referência, com o endereço da saída do motor, para ser conectada junto à entrada Timer enable. Isso significa que toda a vez que o motor for ligado, haverá

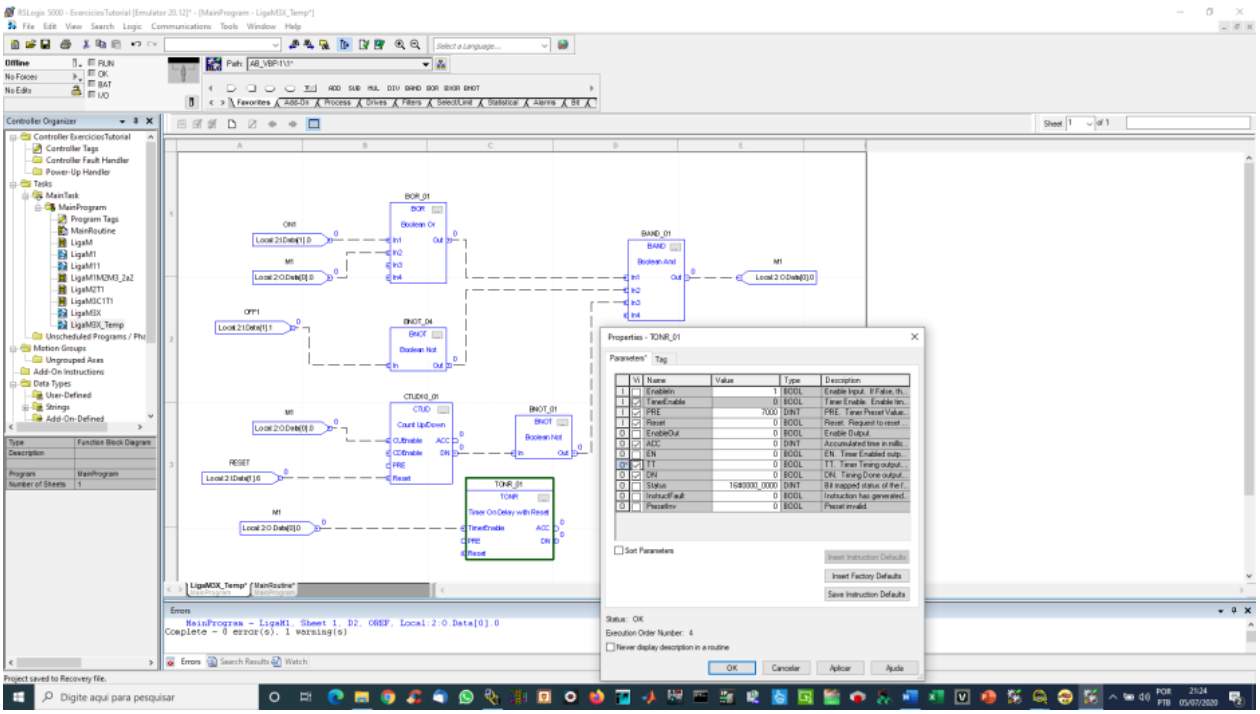
transição de borda de subida junto à entrada do temorizador, iniciando temporização de 7 segundos, após a qual será ativada a saída DN da instrução TONR, conforme figura 261.

Figura 261-Entrada de referência MOTOR inserida na instrução TONR



Fonte: Própria

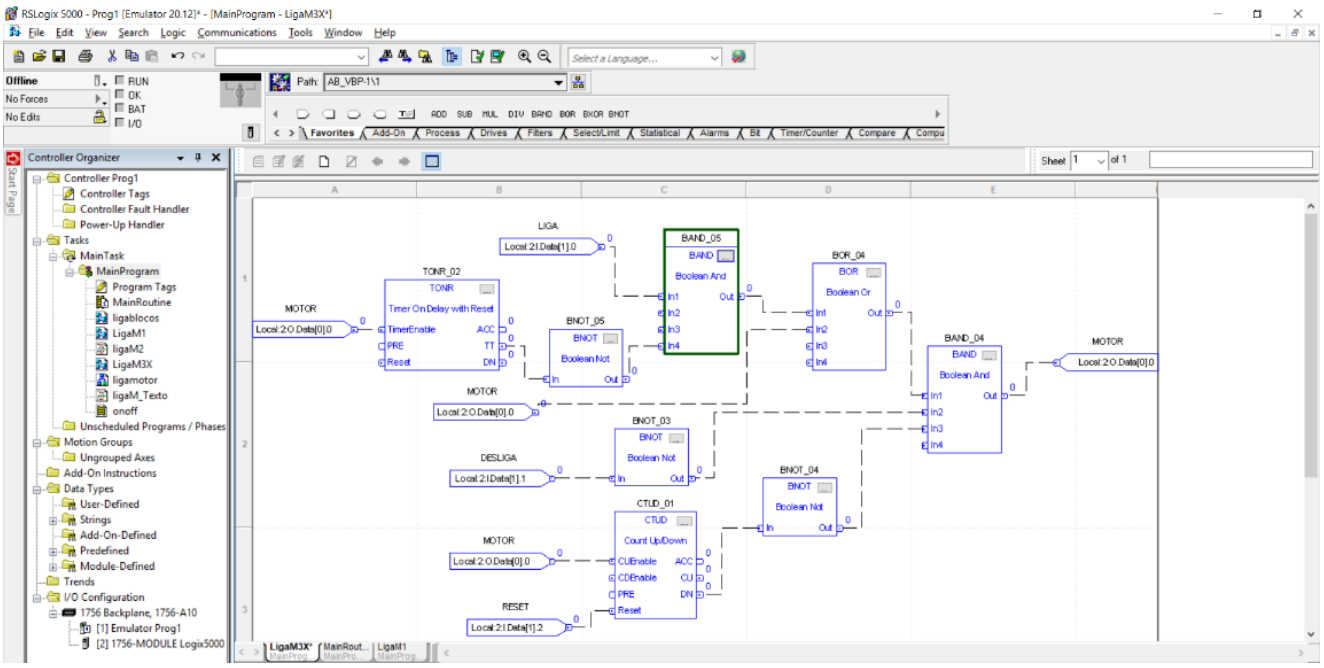
Figura 262 – Habilitando saída TT na instrução TONR



Fonte: Própria

Pretende-se habilitar o próximo acionamento somente após 7 segundos. Uma idéia para realizar esta funcionalidade seria associar a entrada de referência do botão liga a um bloco BAND. A outra entrada desta função lógica AND seria associada à saída TT do temporizador, cujo valor é baixo antes de iniciar a temporização. Para fazer com que a saída TT surja na instrução TONR, clica-se duas vezes sobre este bloco, conforme figura 262 e, na caixa de opções que surgir, marca-se TT. Dessa forma, uma entrada com valor barrado, relacionada à saída TT do temporizador, conectada em série ao botão liga, permitiria o acionamento do motor. Entretanto, após ligar a máquina, o temporizador passa a contabilizar o tempo decorrido, levando a saída TT ao nível alto, cujo valor barrado impede novos acionamentos, devido ao bloco BNOT, cuja saída passa a ser zero.

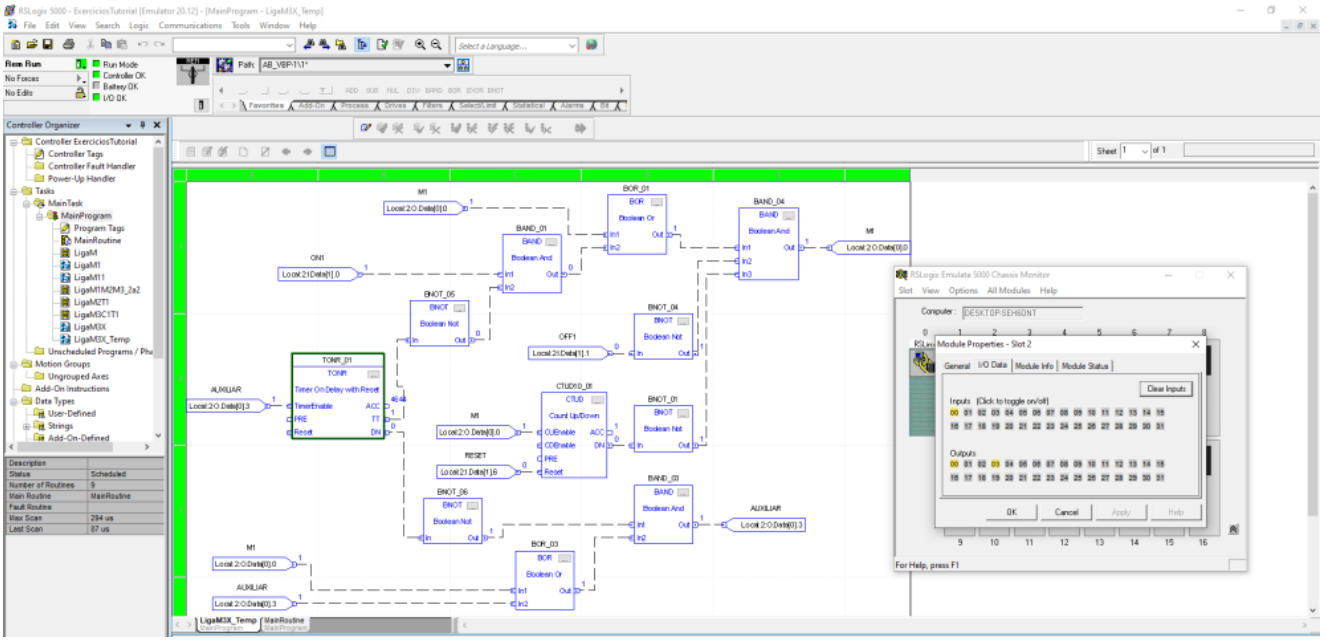
Figura 263 – Solução para o problema proposto



Fonte: Própria

A figura 263 apresenta o programa com o problema proposto finalizado. Entretanto, a solução tem um problema. Se o motor for desligado antes de decorridos 7 segundos, a entrada TT será desabilitada, pois cessará a temporização.

Figura 264 – Solução aprimorada



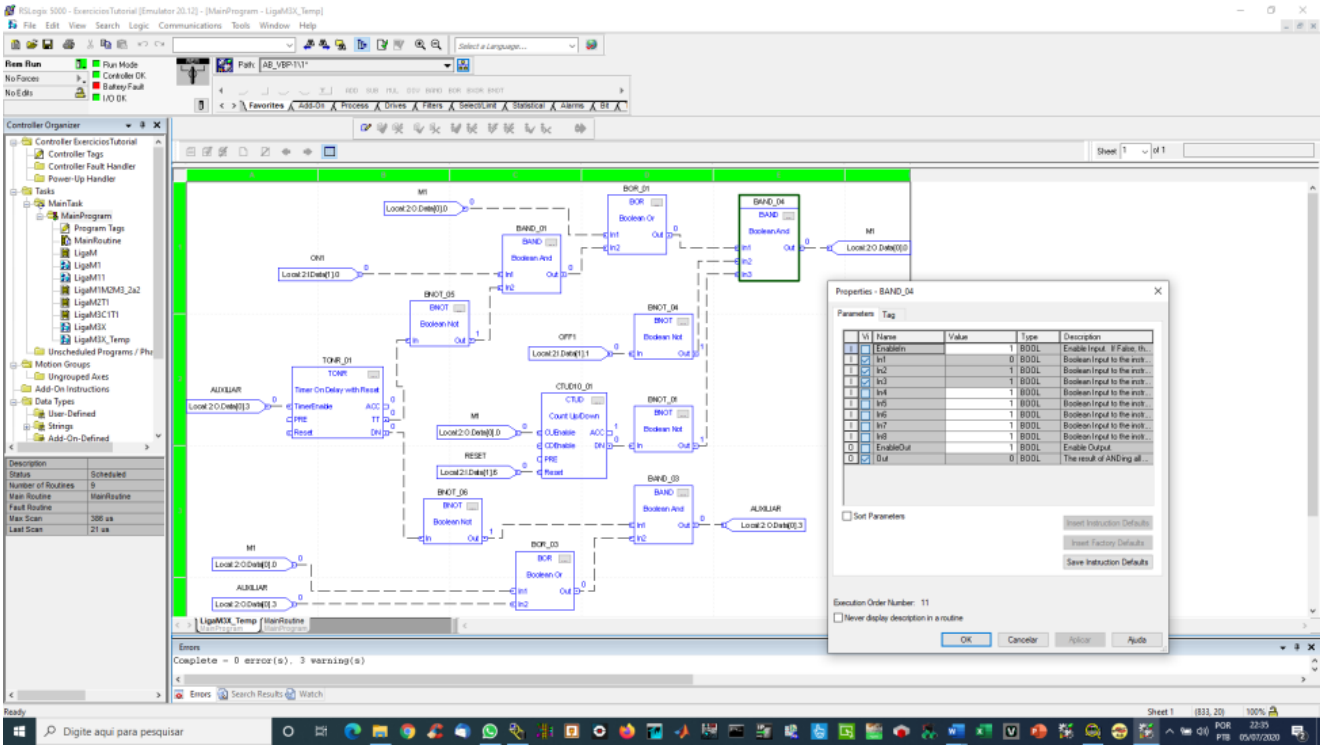
Fonte: Própria

Uma ideia para o problema descrito no parágrafo anterior seria criar uma saída auxiliar, que se manteria acionada, mesmo após o desligamento do motor, proporcionando continuidade à temporização. A solução aperfeiçoada está apresentada na figura 264, acompanhada da devida explicação.

Após contabilizar o tempo decorrido, o conteúdo do registro ACC da instrução TONR igualará o valor estabelecido no campo PRESET, ativando a saída DN, cujo valor barrado insere nível lógico 0 no bloco BAND_03, desativando a saída auxiliar Local:2:O.Data[0].3, o que desabilitará o temporizador. Isso levará a saída TT a zero, a qual passa por um bloco BNOT_05, invertendo seu valor para nível lógico alto, habilitando novamente o acionamento do motor através do botão liga.

Para selecionar as entradas a serem exibidas nos respectivos blocos de instruções, basta clicar sobre um deles, como o BAND_04 na figura 265, e marcar os campos que aparecerão na instrução.

Figura 265 – Seleção de campos para exibir na instrução BAND_04

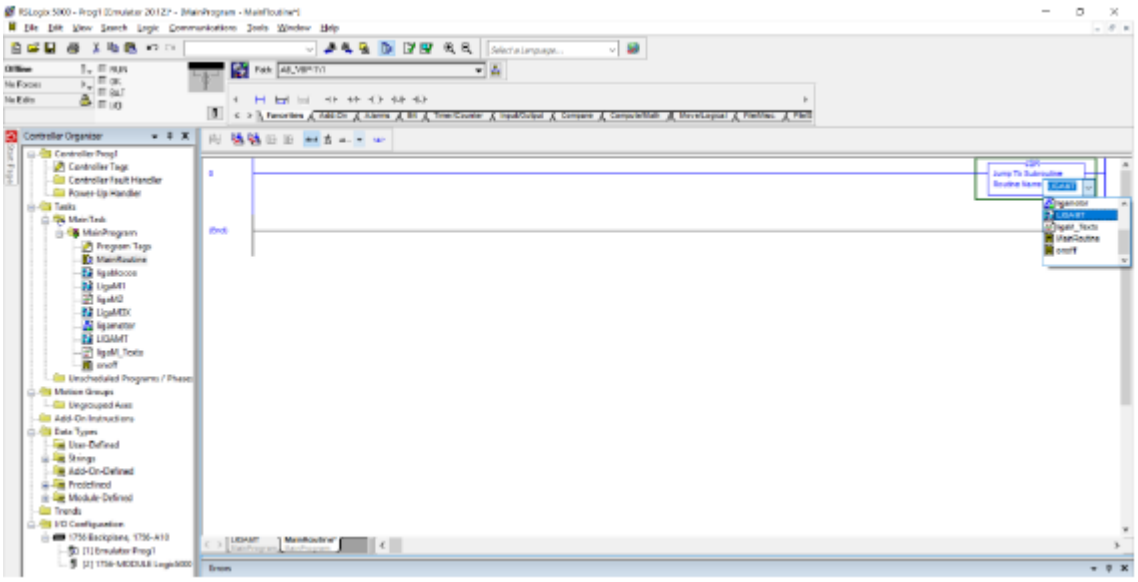


Fonte: Própria

7). Elabore programa em linguagem de blocos para acionamento de motor, que permita ligar manualmente uma segunda máquina, decorridos 10 segundos após entrar em funcionamento.

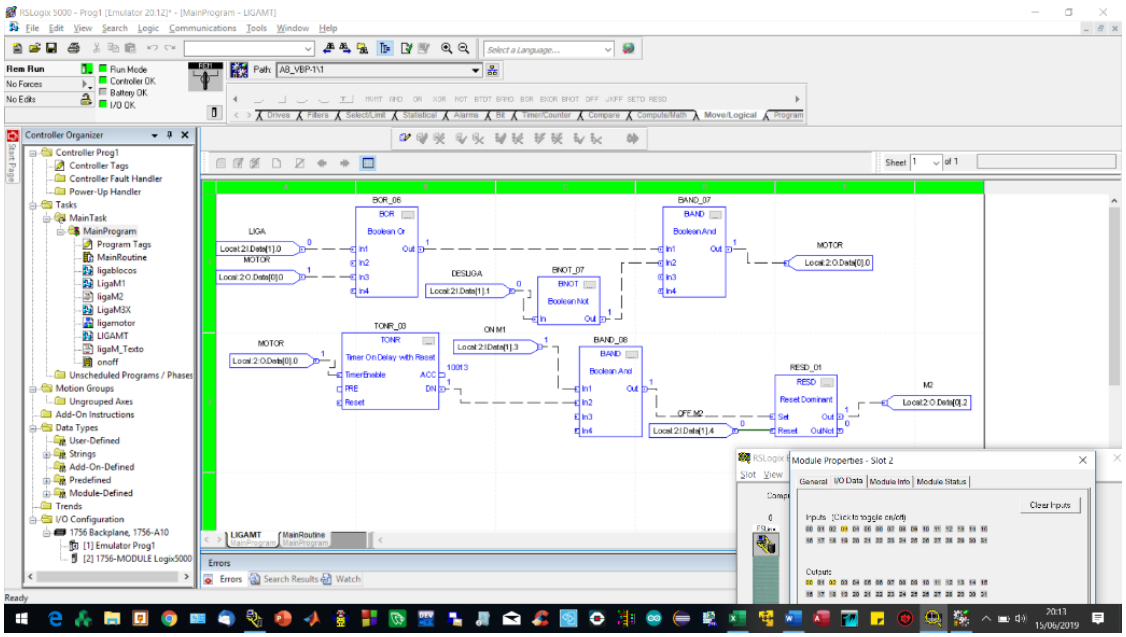
O programa foi nomeado ligaMT. Procede-se à seleção do arquivo criado, para ser carregado pela rotina principal, antes de efetuar o download (fig. 266).

Figura 266 – Seleção do programa a ser carregado através da instrução JSR



Fonte: Própria

Figura 267 – Programa completo do sétimo enunciado



Fonte: Própria

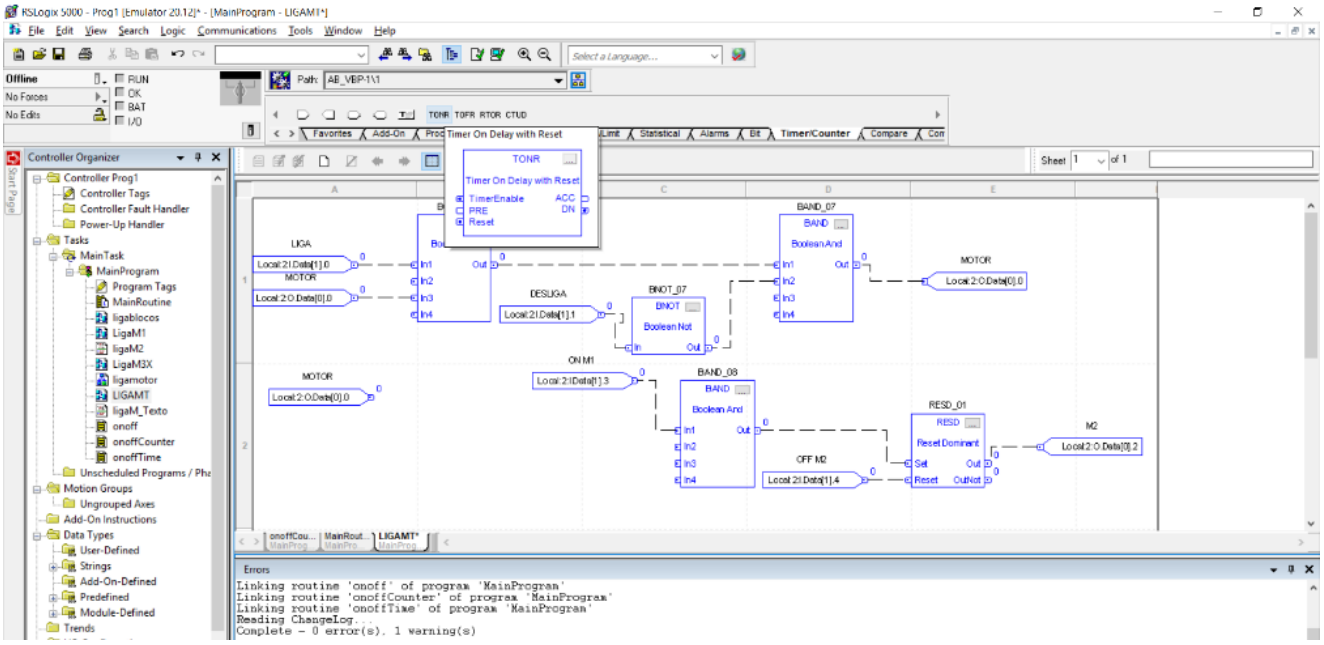
A figura 267 apresenta código que realiza o pedido do sétimo enunciado. O programa utiliza a instrução auto retentiva RESD para acionar o segundo motor, porque o valor inicial da saída deve ser desativada ou zero. A saída do segundo motor será acionada apenas após o temporizador contabilizar 10 segundos, o que ativará sua saída DN, conectada a uma das entradas da porta BAND_08, habilitando a ativação de M2 pelo botão associado ao endereço de entrada Local:2:I.Data[1].3

O estado de repouso do botão desliga do primeiro motor é normalmente fechado. Dessa forma, o mesmo deve passar por um bloco lógico BNOT, antes de se conectar à porta BAND_07, cujo resultado acionará M1.

8). Elaborar um aplicativo no sistema supervisório In Touch, que realize comunicação com o programa anterior em linguagem de blocos.

Para tal fim, é preciso criar instrução temporizadora em linguagem de blocos, que tenha escopo global. Dessa forma, antes de proceder ao desenvolvimento do sistema supervisório, apaga-se o temporizador de escopo local, criado para o programa anterior, inserindo-se novo bloco temporizador com abrangência global, conforme figura 268.

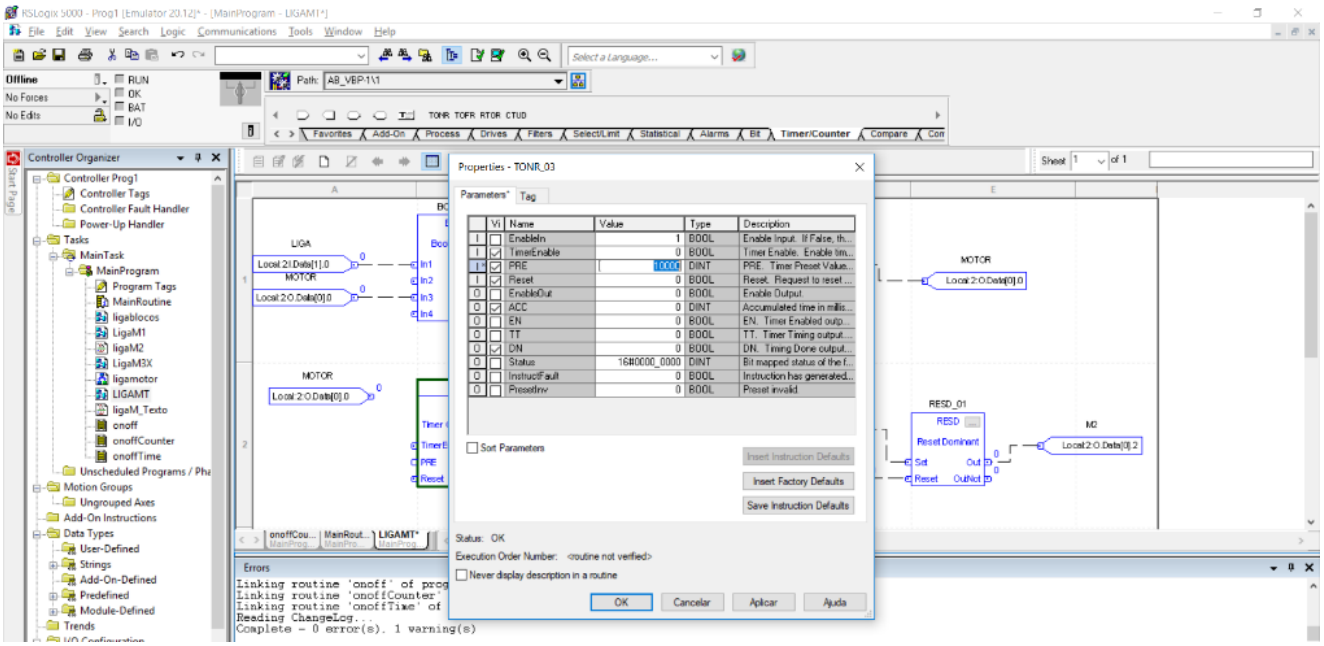
Figura 268 – Inserindo temporizador com escopo global



Fonte: Própria

Clicando-se duas vezes sobre o temporizador, seu preset pode ser configurado para 10000 milissegundos (fig. 269).

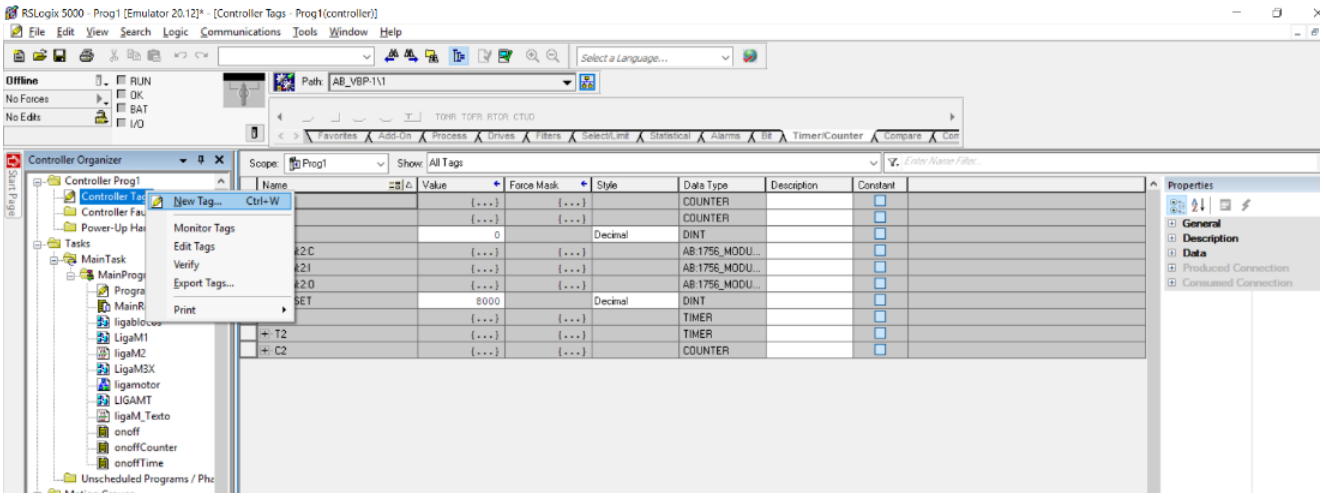
Figura 269 – Configurando Preset do temporizador



Fonte: Própria

Abre-se Controller Tags, para criar temporizador global (fig. 270).

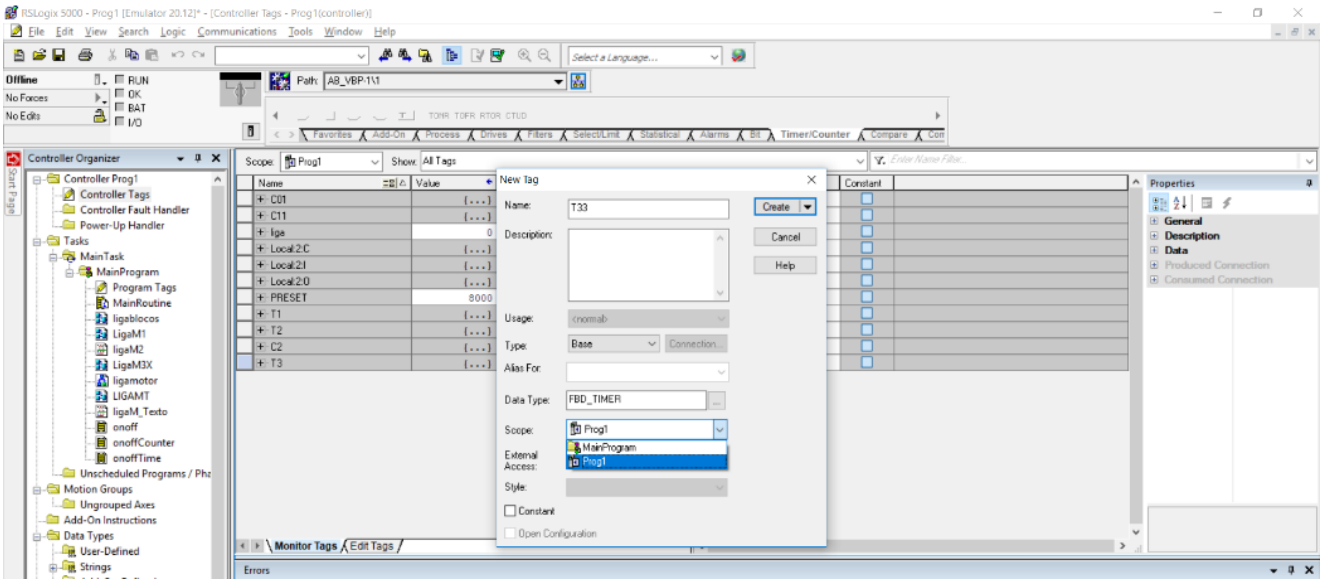
Figura 270 – Configurando temporizador global



Fonte: Própria

Nomeia-se o temporizador de T33, conferindo ao mesmo escopo global, e tipo de dados FBD_TIMER, que é o temporizador da linguagem de blocos de funções (fig. 271).

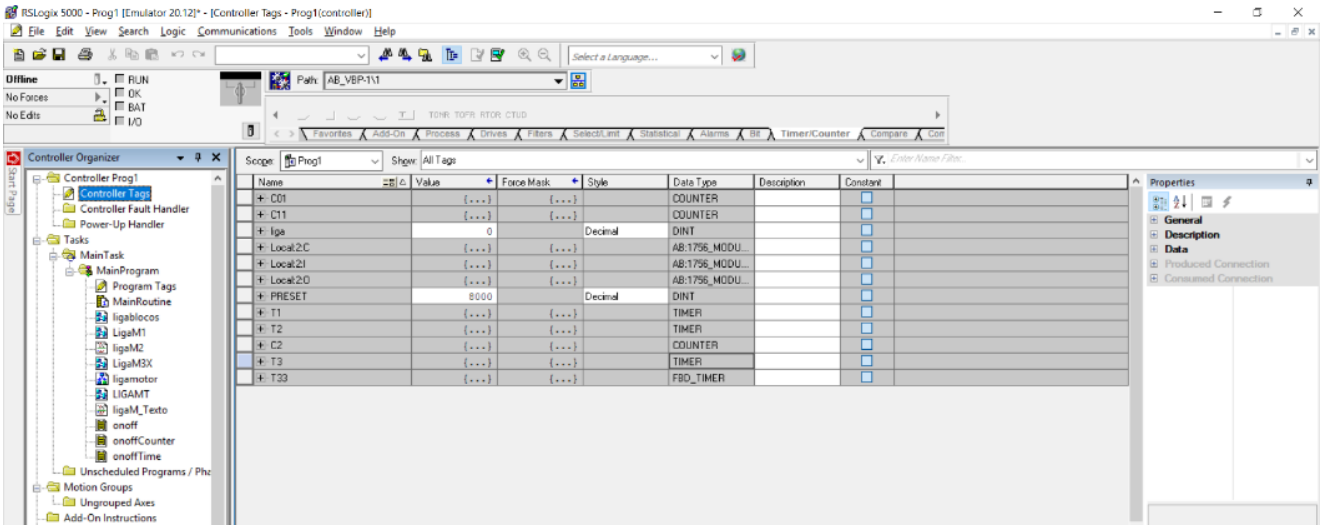
Figura 271 – Atribuindo escopo global ao temporizador



Fonte: Própria

Verifica-se a criação da tag após pressionar em Create (fig. 272).

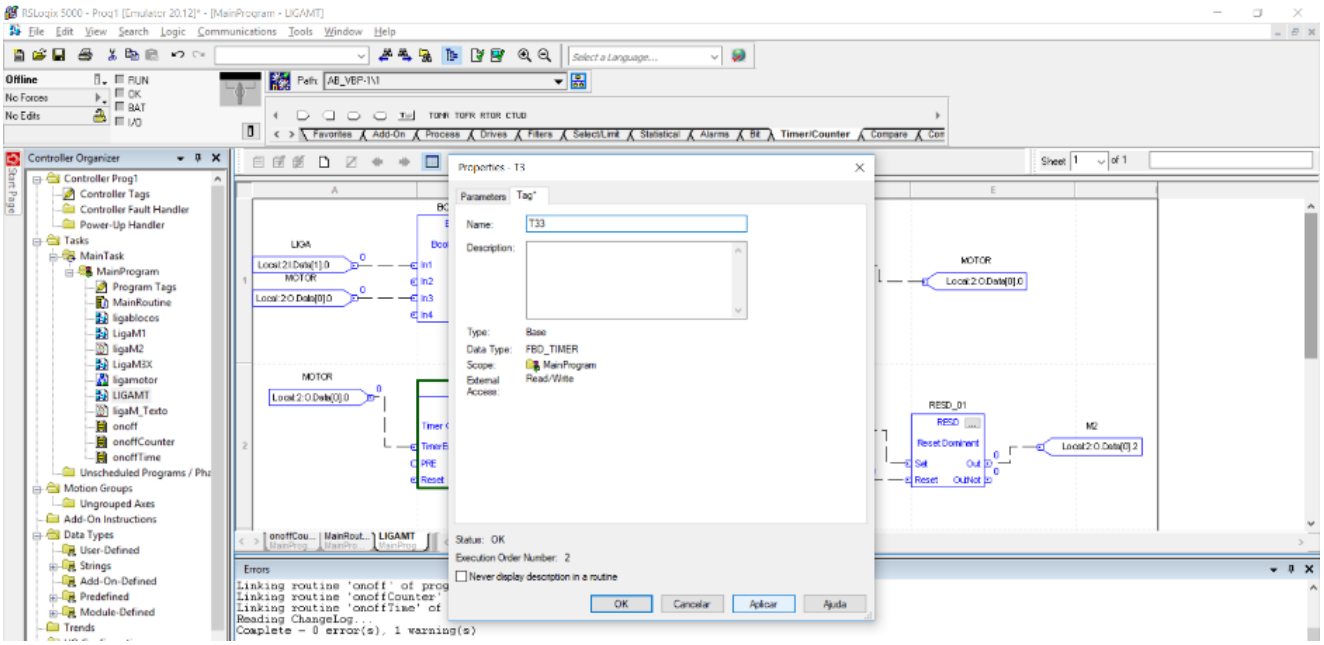
Figura 272 – Finalizando criação do temporizador T33



Fonte: Própria

Abre-se o programa LIGAMT, para acessar as propriedades do temporizador TONR3, e clicar sobre a aba tag, com o intuito de trocar seu nome para a variável T33, cujo escopo é global (fig. 273).

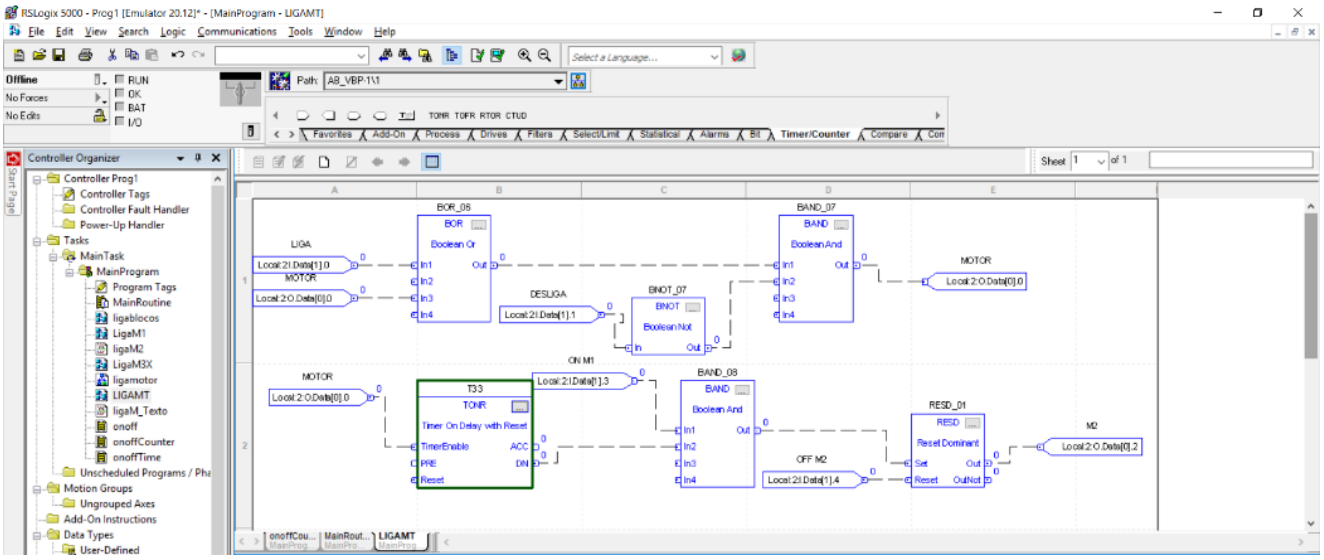
Figura 273 – Substituindo nome do temporizador TONR3



Fonte: Própria

Refazem-se as conexões de entradas e saídas de TONR3 (fig. 274).

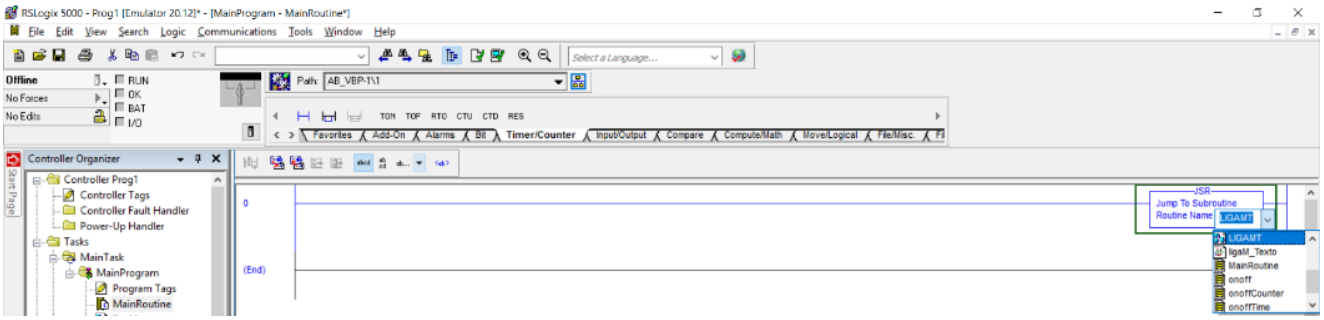
Figura 274 – Refazendo conexões de entrada e saída de TONR3



Fonte: Própria

A partir da figura 274 o programa está adequado à comunicação com o sistema supervisório. Resta acessar o código principal e modificar o programa a ser carregado pela instrução JSR (fig. 275).

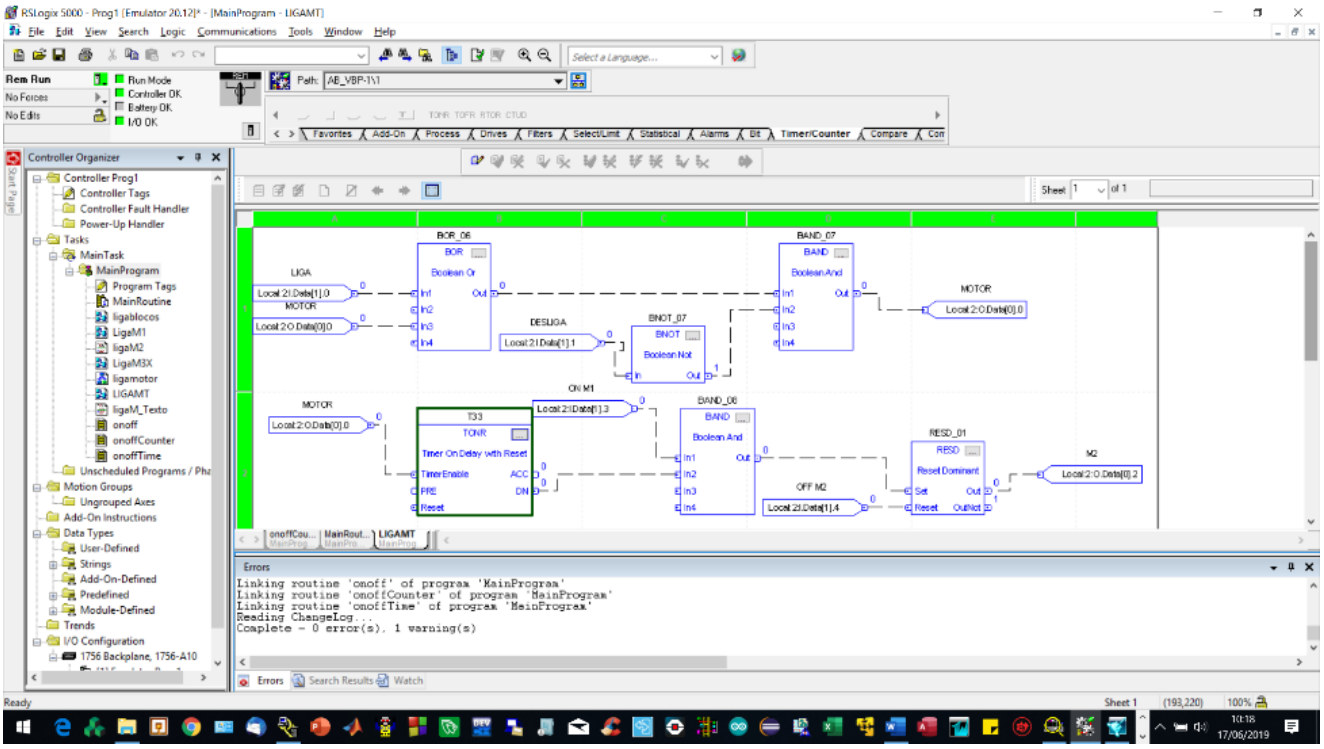
Figura 275 – Selecionado o programa LIGAMT para carga através da instrução JSR



Fonte: Própria

Procede-se em seguida ao download, e à execução do código (Fig. 276).

Figura 276 – Execução do programa

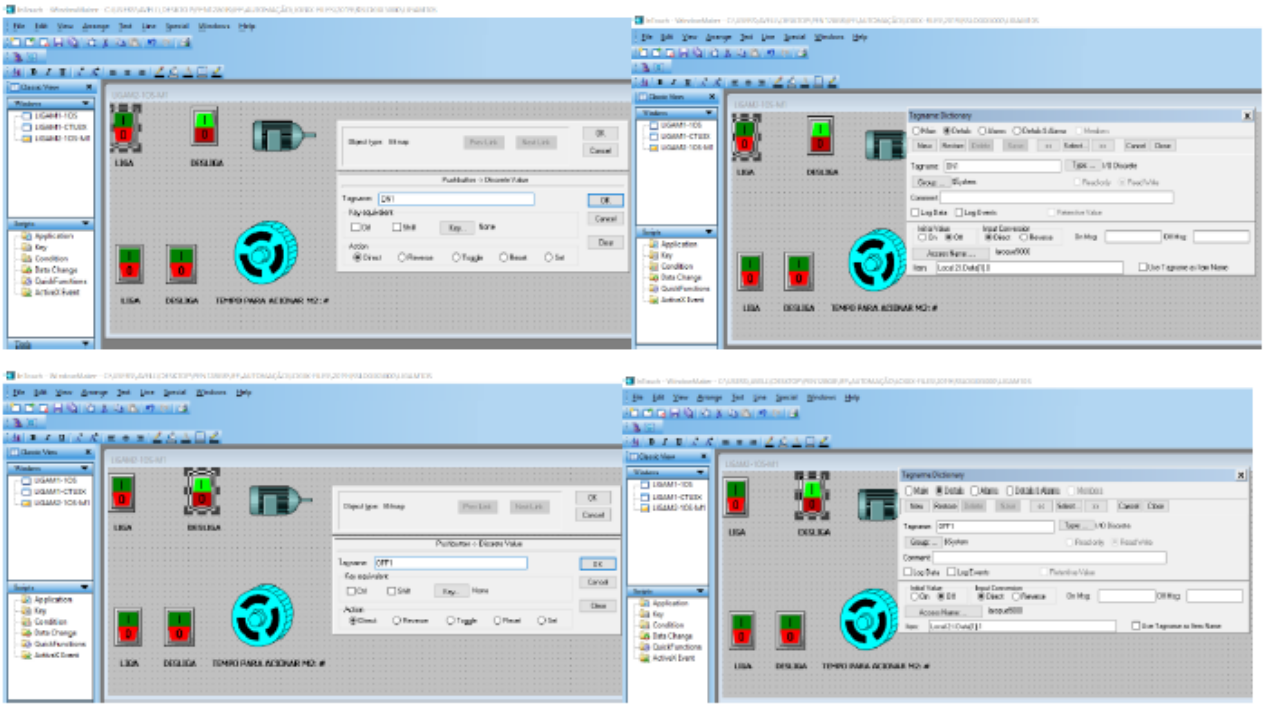


Fonte: Própria

As telas do sistema supervisório são apresentadas nas próximas figuras, juntamente com as respectivas configurações de objetos que as acompanham.

Vê-se na figura 277 as tags ON1 e OFF1, que se comunicam com o programa FBD pelos endereços Local:2.I.Data[1].0 e Local:2.I.Data[1].1, especificados no campo Item do dicionário de Tagnames.

Figura 277 – Configurações dos botões liga e desliga do primeiro motor

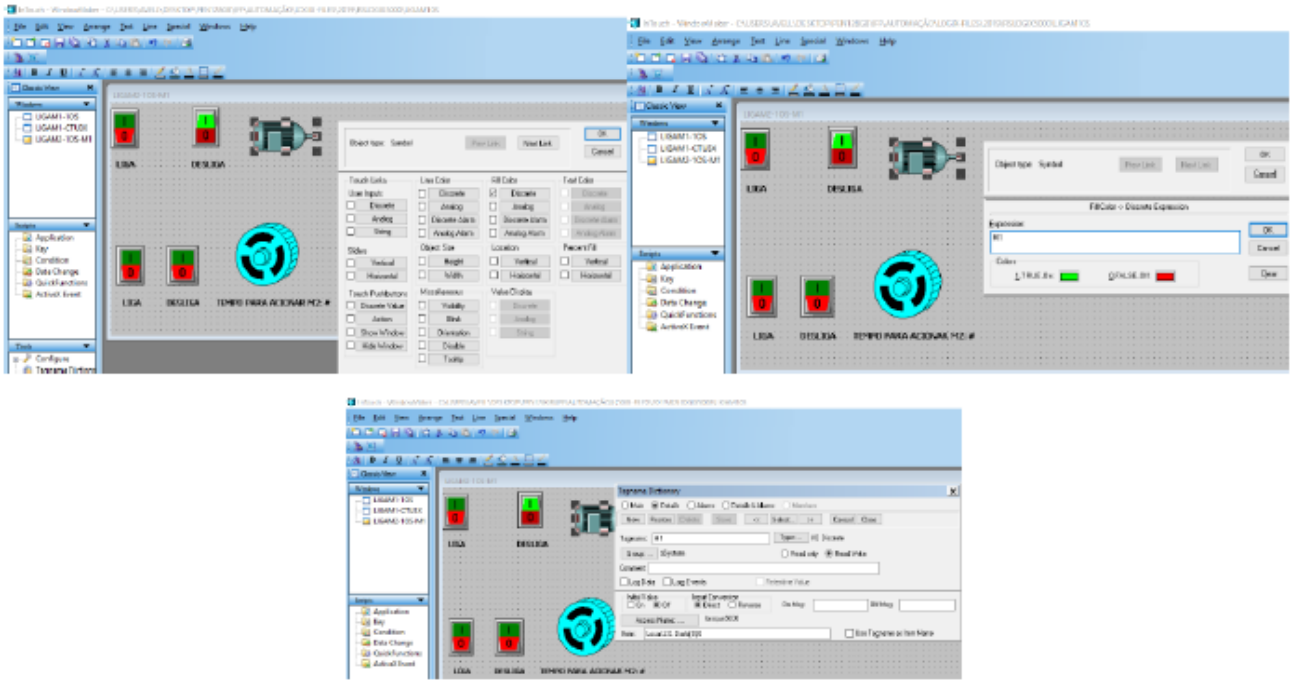


Fonte: Própria

ON1 e OFF1 estão configuradas como botão do tipo impulso, com comportamento análogo a campainha doméstica, pois no campo *Action* selecionou-se a opção *Direct*.

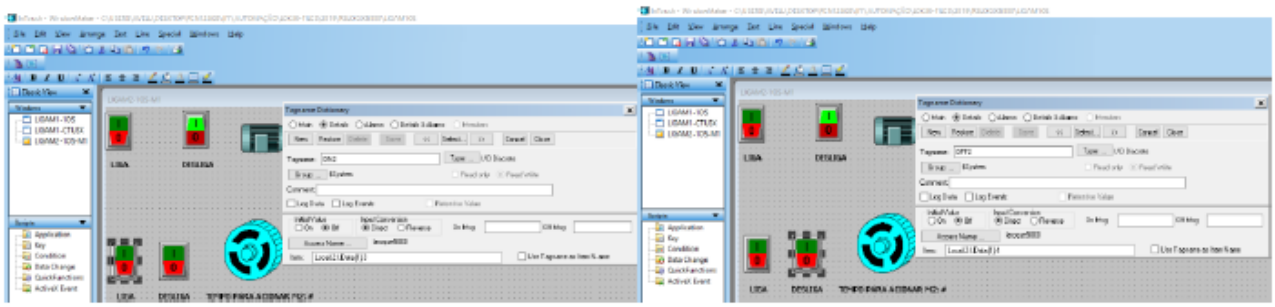
Isso significa que os contatos do programa endereçados no campo Item somente comutarão dos respectivos estados de repouso enquanto as chaves do sistema supervisórios estiverem acionadas. As configurações do primeiro motor (fig. 278) são controladas pela *tagname* M1, que controla cores de preenchimento e se comunica com o seguinte endereço de saída especificado no campo Item: Local: 2.O.Data[0].0

Figura 278 – Configurações do primeiro motor



Fonte: Própria

Figura 279 – Configurações dos botões liga e desliga do primeiro motor

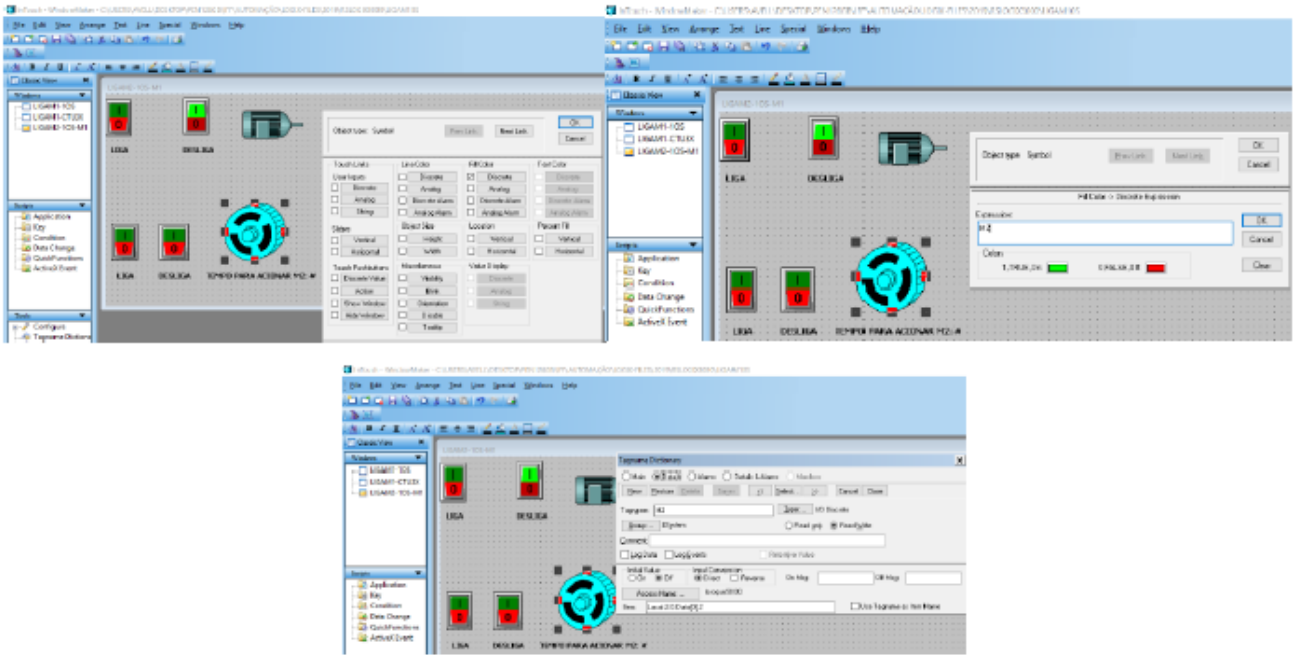


Fonte: Própria

A figura 279 mostra as configurações dos botões Liga e Desliga do segundo motor, respectivamente controlados pelas *tagnames* ON2 e OFF2, cujos endereços de comunicação com o programa FBD são especificados nos campos Item como Local:2.I.Data[1].3 e Local:2.I.Data[1].4

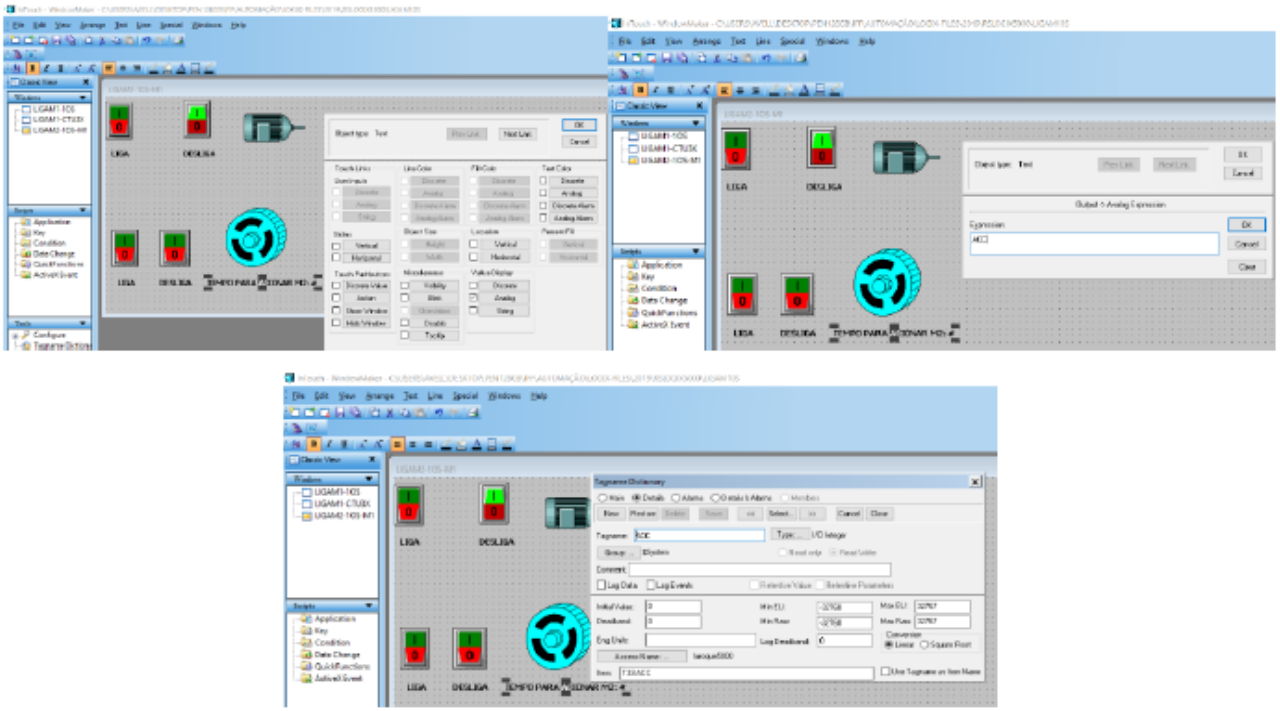
As configurações do segundo motor (fig. 280) são controladas pela *tagname* M2, que controla cores de preenchimento e se comunica com o seguinte endereço de saída especificado no campo Item: Local: 2.O.Data[0].2

Figura 280 – Configurações do segundo motor



Fonte: Própria

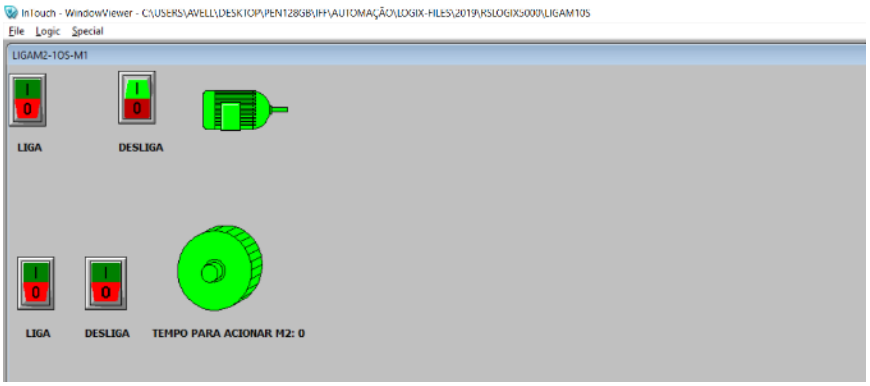
Figura 281 – Configurações de exibição do tempo restante para acionar o segundo motor



Fonte: Própria

Segue tela do sistema supervisório em execução (fig. 282).

Figura 282 – Tela do sistema supervisório em execução



Fonte: Própria

3.4 PROBLEMAS DE AUTOMAÇÃO EM LINGUAGEM GRAFCET

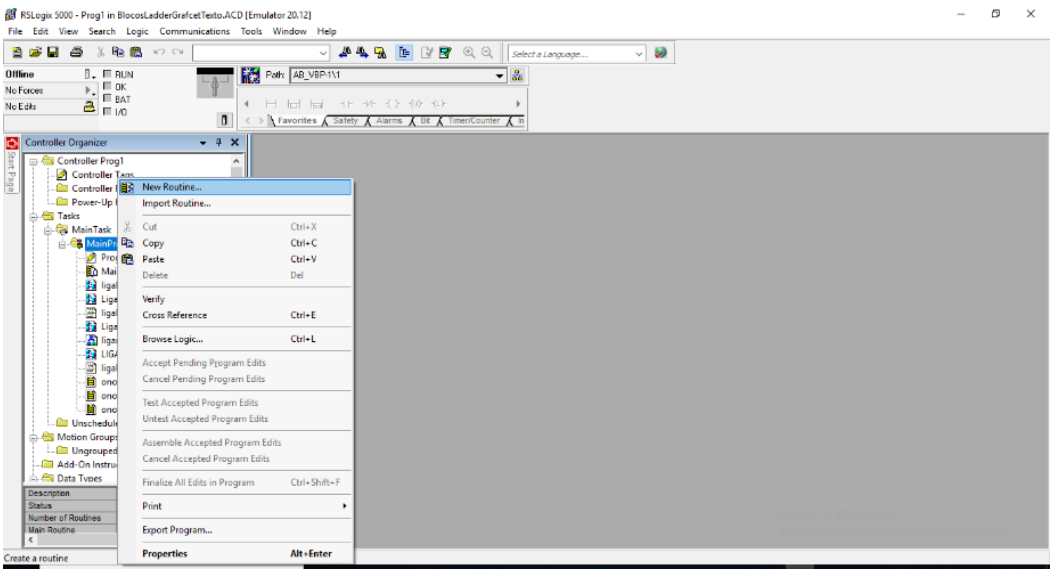
Grafcet é uma linguagem visual, também conhecida como *Sequential Function Chart*, sendo constituida por etapas, ações e transições.

Cada etapa tem uma ação associada a ela. Para mudança entre etapas, é preciso que a transição entre elas seja verdadeira. A seguir serão desenvolvidos diversos problemas de automação em *grafcet*.

1) Pede-se criar código em grafcet, que ligue ou desligue um motor. Para tanto, pode-se pensar inicialmente em duas etapas, respectivamente nomeadas on e off.

Clica-se com o botão direito em New program, selecionando-se a opção new routine, conforme figura 283.

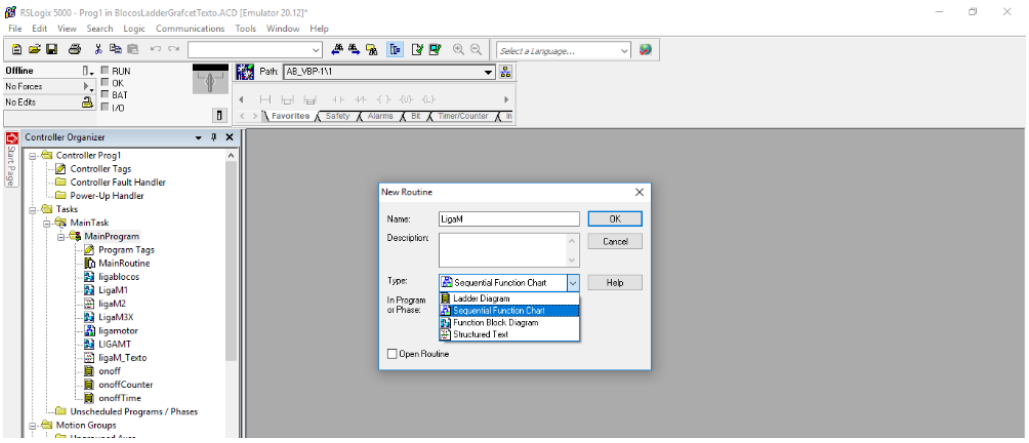
Figura 283 – Criando nova rotina



Fonte: Própria

Escolhe-se o nome LigaM, e o tipo de linguagem Sequential Function Chart (fig. 284).

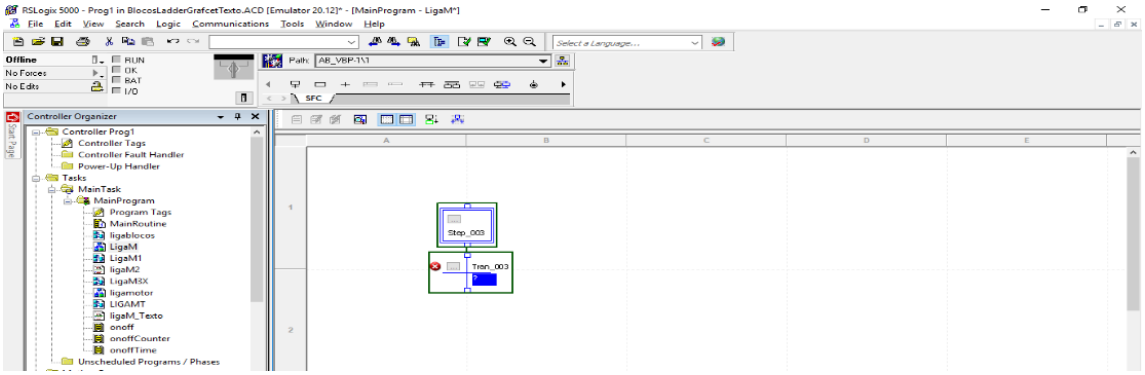
Figura 284 – Selecionando a linguagem de programação em LigaM



Fonte: Própria

Pressiona-se Ok, e surge figura 285, na qual são automaticamente criadas uma etapa e uma transição.

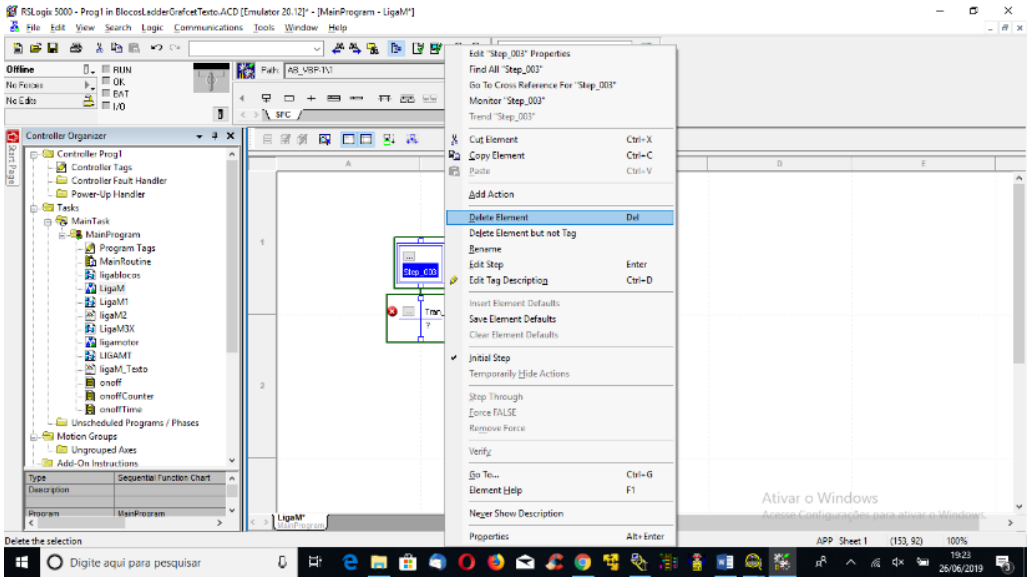
Figura 285 – Criação de etapa e transição



Fonte: Própria

É possível criar etapas e transições com escopo global. Para tanto, clica-se com o botão direito sobre as etapas e transições automaticamente, com o intuito de apagá-las, selecionando a opção *Delete Element* (fig. 286).

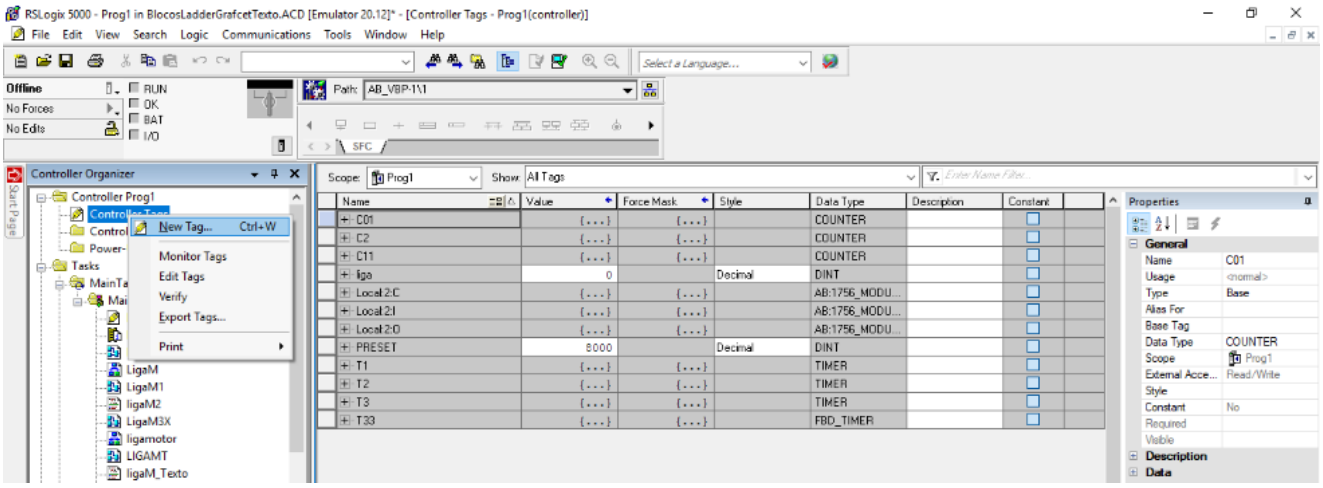
Figura 286 – Apagando elemento criado



Fonte: Própria

Clica-se duas vezes sobre a aba Controller Tags, selecionando a opção New Tag (fig. 287).

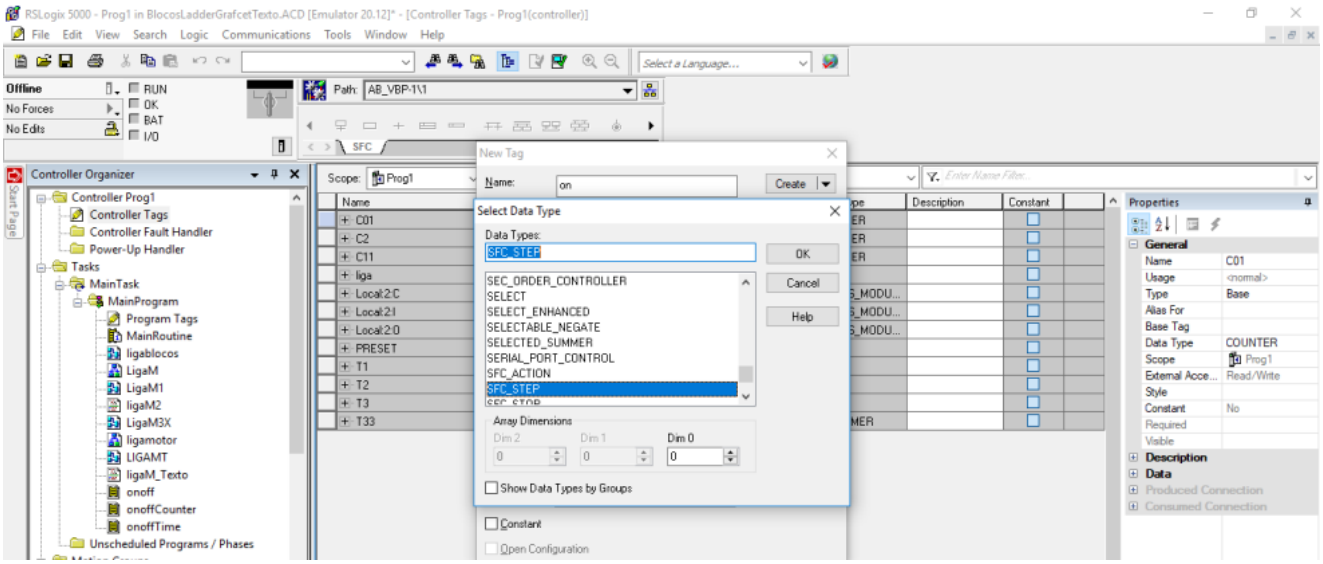
Figura 287 – Inserindo nova Tag



Fonte: Própria

Em seguida procede-se à elaboração de etapa com escopo global, chamando-a de on e selecionando seu tipo de dado como SFC_STEP (fig. 288).

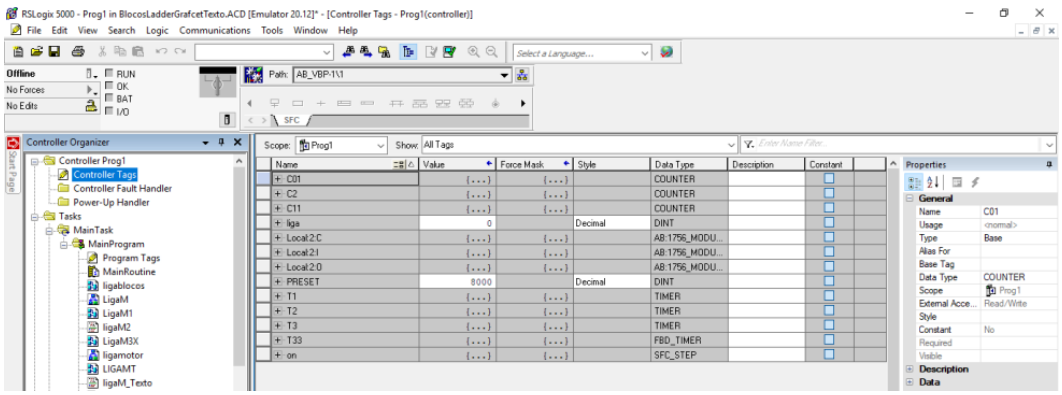
Figura 288 – Criação da Tag SFC on



Fonte: Própria

Pressiona-se OK e, em seguida, aperta-se o botão Create, verificando-se a criação da Tag on, conforme figura 289.

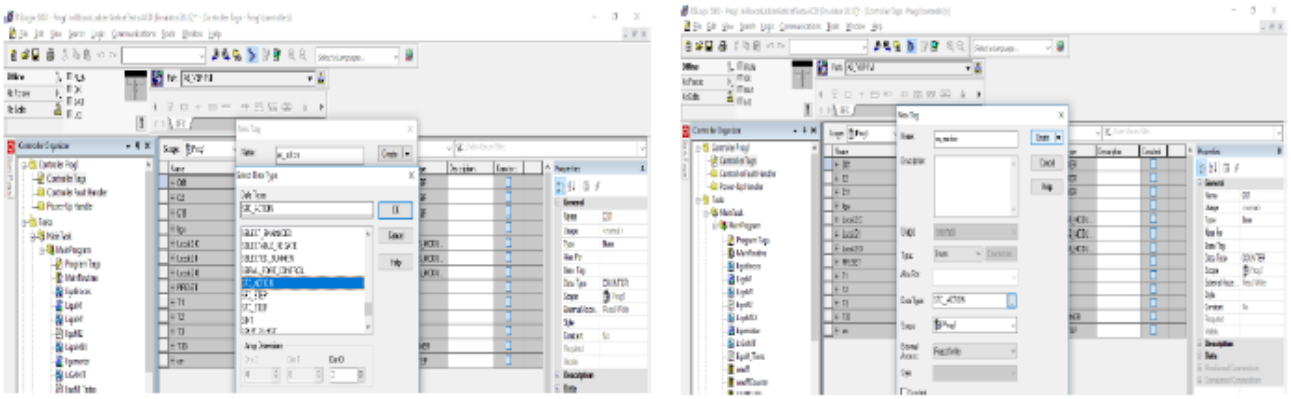
Figura 289 – Criação da Tag SFC on



Fonte: Própria

Cria-se nova *tag* de escopo global, clicando-se novamente sobre controller tags, a qual será nomeada *on_action*, selecionando seu tipo de dado como SFC_ACTION (fig. 290).

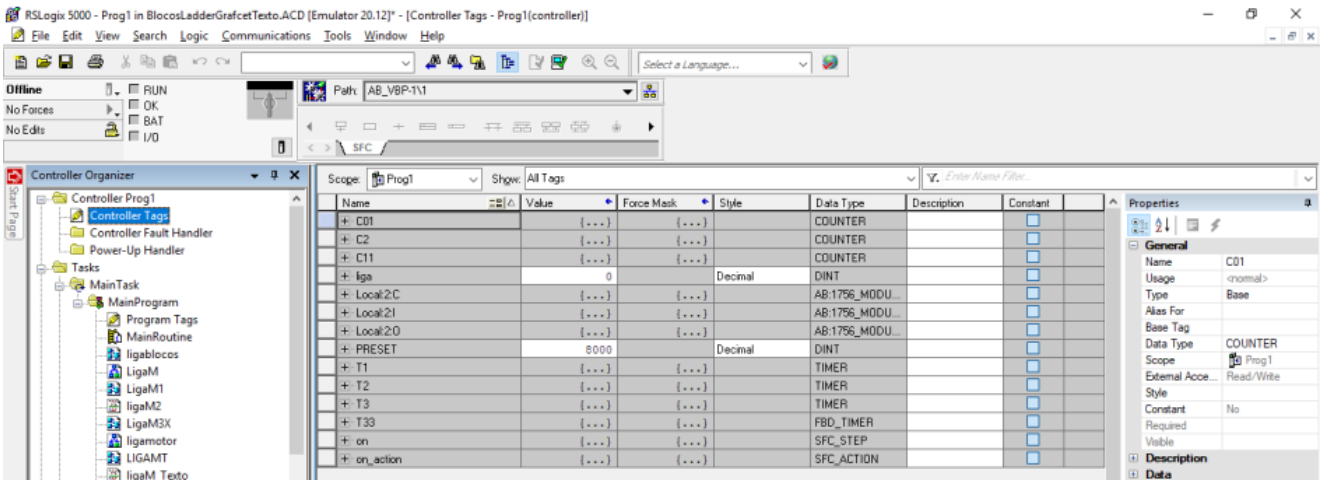
Figura 290 – Criação da *tag on_action*



Fonte: Própria

Finaliza-se a criação da tag na figura 291.

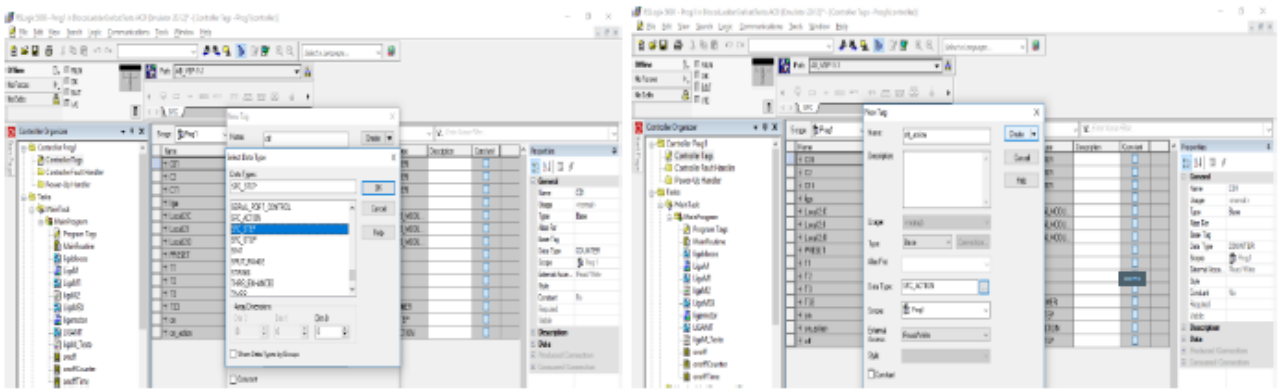
Figura 291 – Criação da tag on_action



Fonte: Própria

Em seguida criam-se mais duas tags de escopo global, para representar a etapa desligada e ação a ela associada, respectivamente nomeadas off e off_action, conforme figura 292.

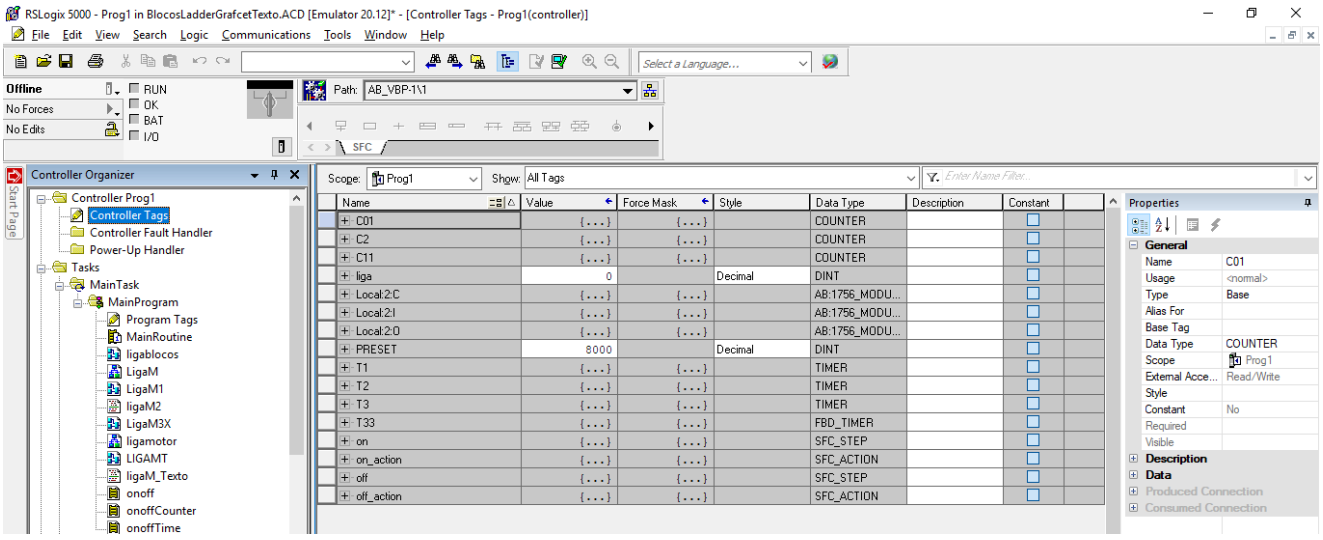
Figura 292 – Criação das tags off e off_action



Fonte: Própria

Verifica-se a criação das 4 tags na figura 293.

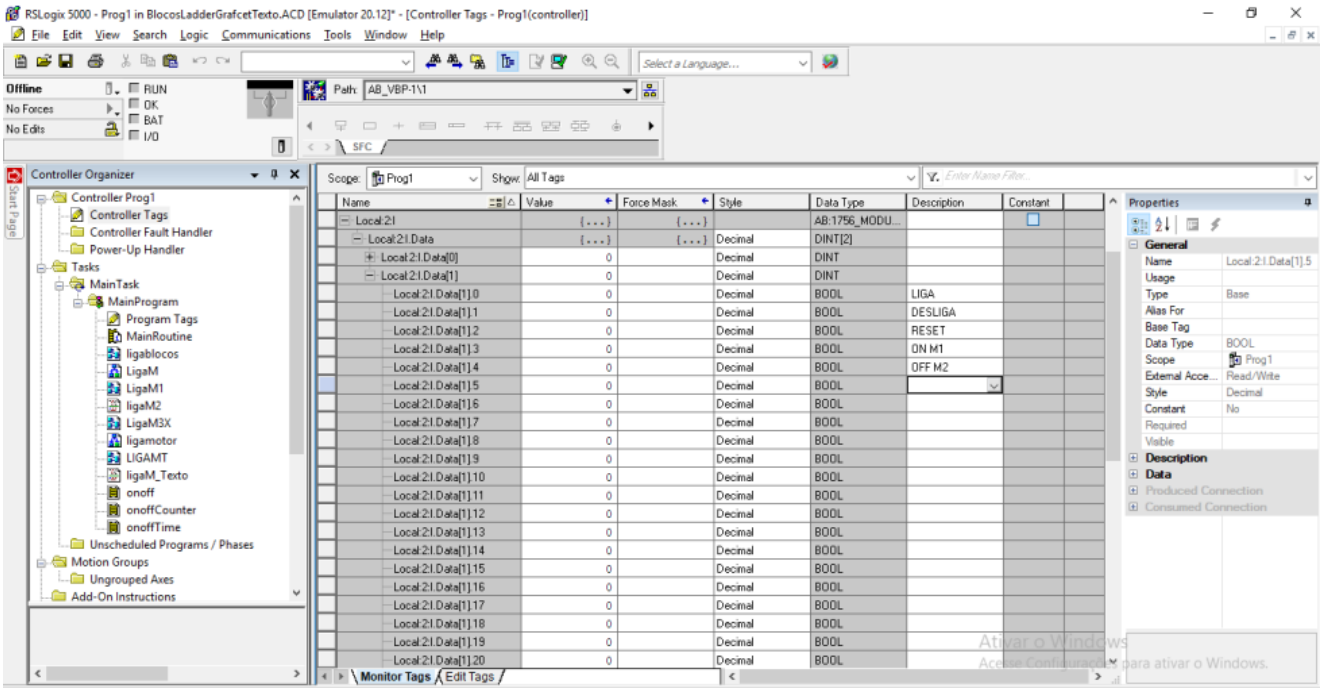
Figura 293 – Criação das tags on_action, on, off e off_action



Fonte: Própria

Agora é preciso definir variáveis de escopo global, associadas aos botões de liga e desliga. Para tanto, clica-se duas vezes sobre Controller Tags, abrindo-se o segundo conjunto de dados de entradas locais, conforme figura 294.

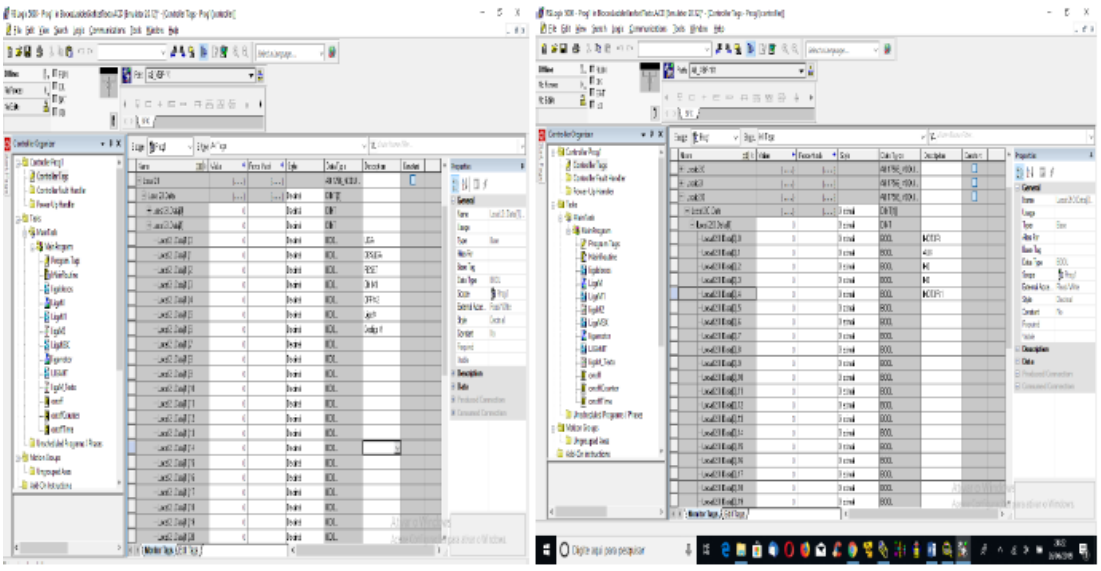
Figura 294 – Definindo as variáveis dos botões



Fonte: Própria

Como há outros endereços ocupados, escolhem-se os seguintes para assumir as posições de liga e desliga, respectivamente Local:2:I.Data[1].5 e Local:2:I.Data[1].6, descrevendo suas funções na coluna Description como sendo Liga M e Desliga M. Minimizam-se os conjuntos de entrada, para abrir a tabela de dados de saída, atribuindo-se ao endereço Local:2:O.Data[0].4 a descrição MOTOR1 (fig. 295).

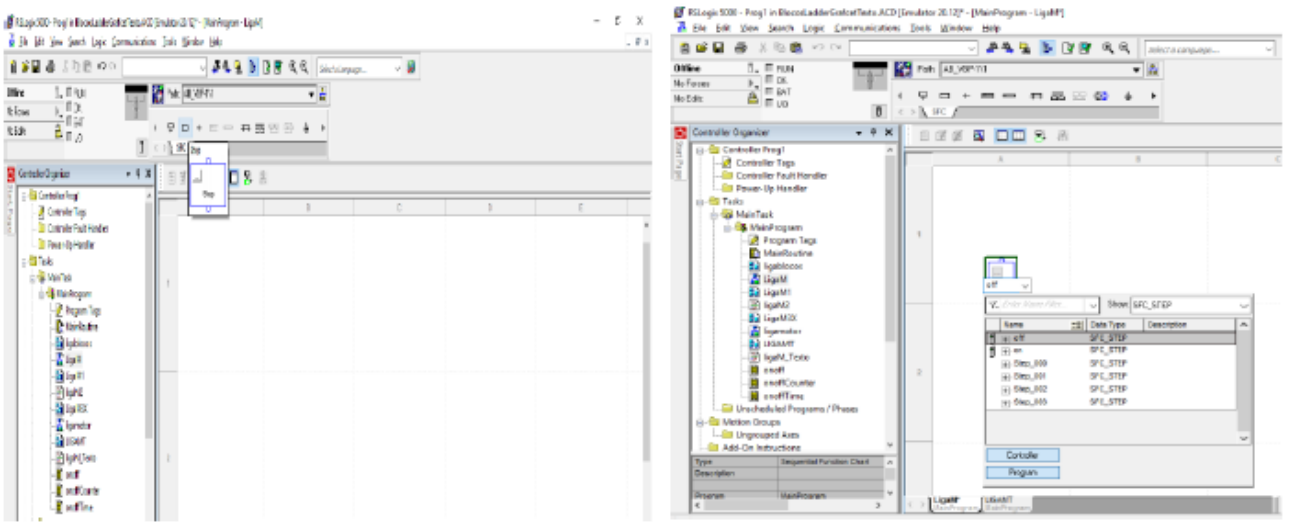
Figura 295 – Definindo as variáveis dos botões



Fonte: Própria

O próximo passo consiste em abrir a rotina LigaM, e selecionar um passo para ser inserido. Clica-se duas vezes sobre o passo, selecionando-se a tag on, conforme figura 296.

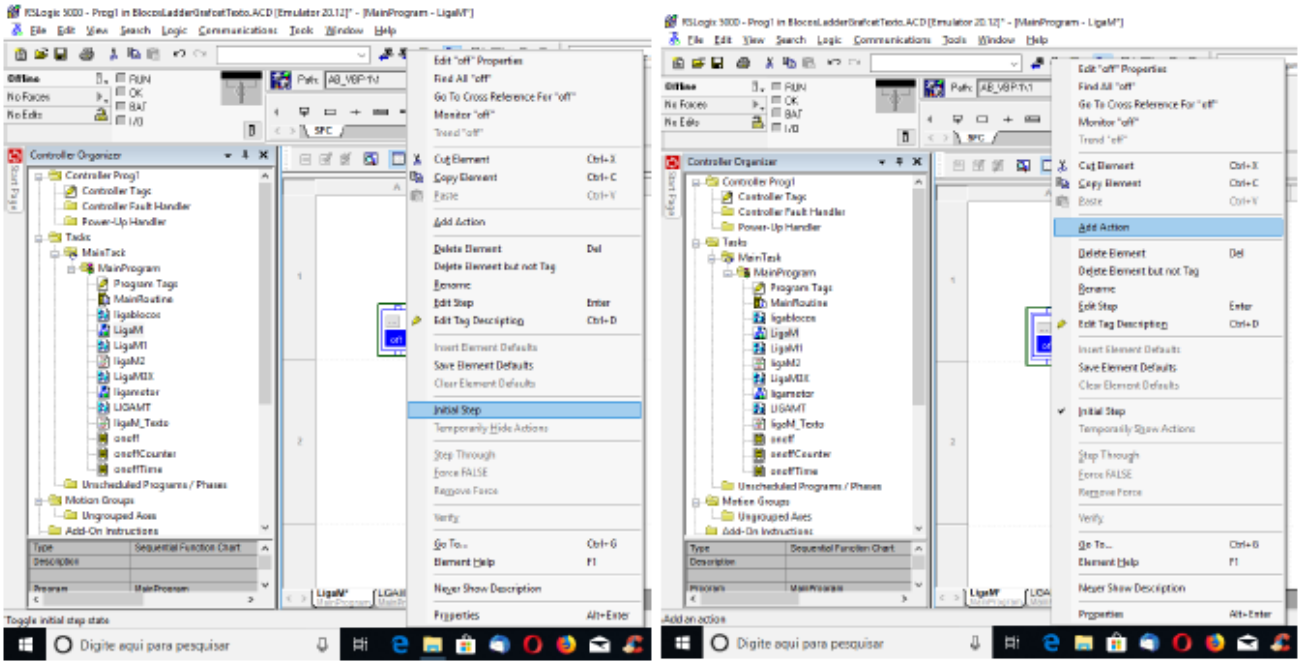
Figura 296 – Inserção de passo e atribuição da respectiva tag



Fonte: Própria

Clica-se com o botão direito sobre o passo, definindo-o como passo inicial. Pressiona-se o botão direito sobre o passo, para adicionar uma ação associada a ele, conforme figura 297.

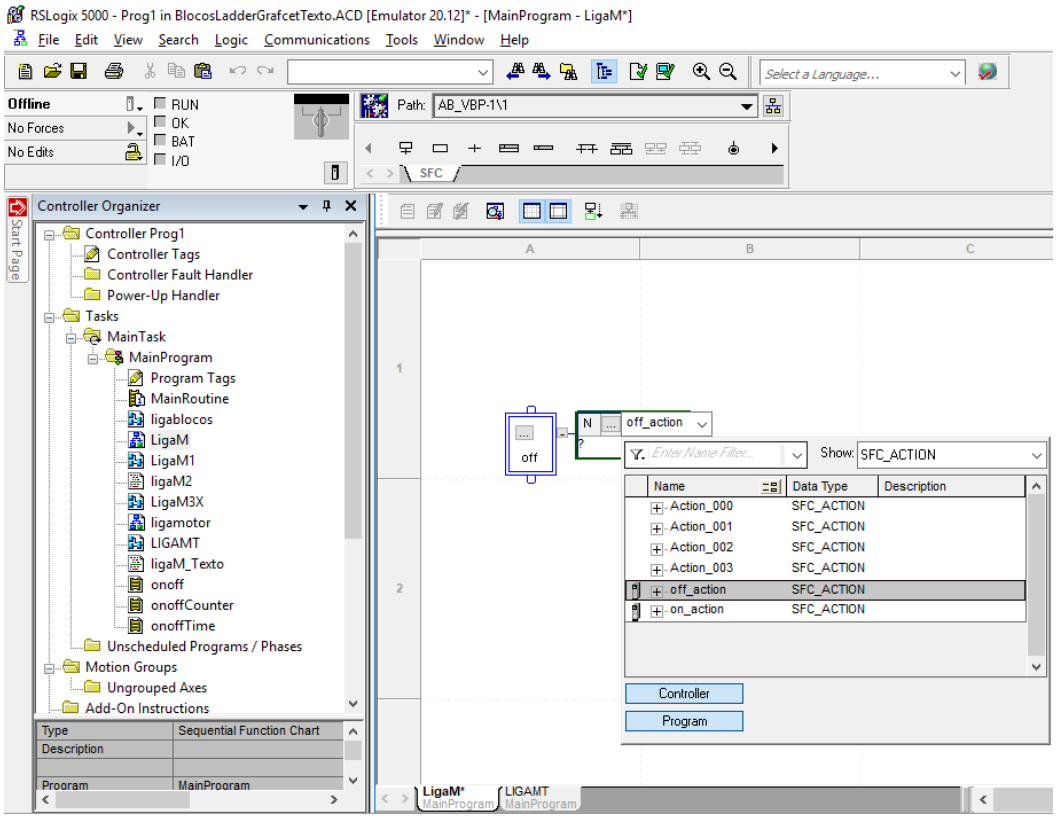
Figura 297 – Adicionando ação ao passo



Fonte: Própria

Seleciona-se a ação off_action, anteriormente definida como tag do controlador (fig. 298).

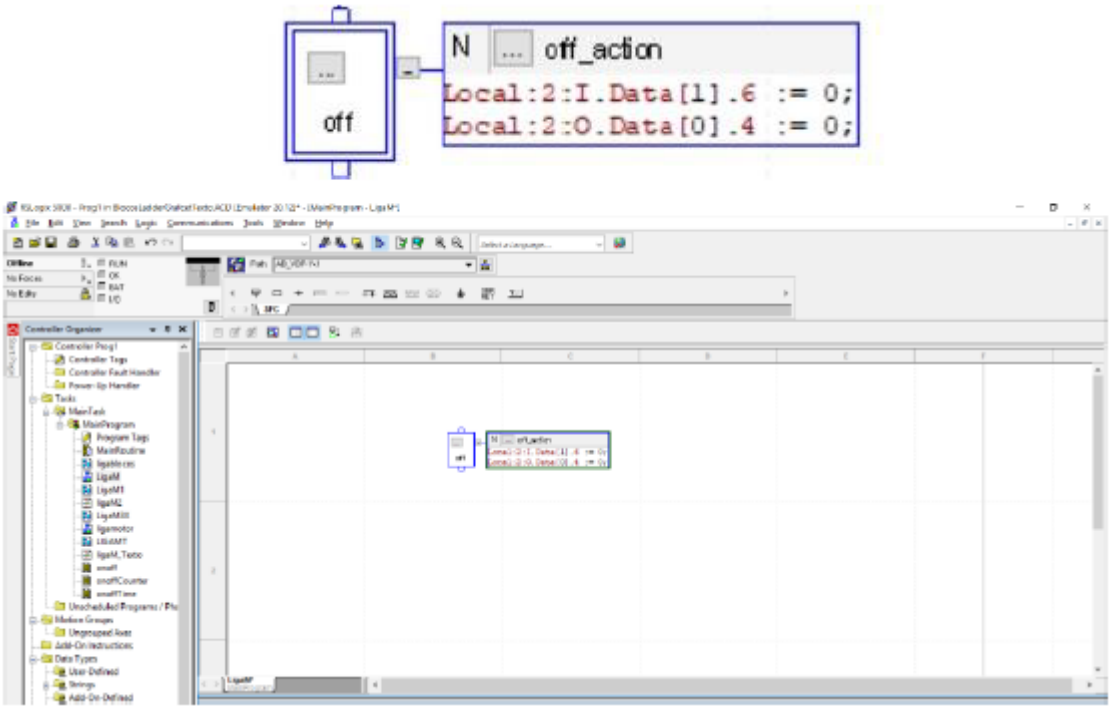
Figura 298 – Selecionando a ação off_action



Fonte: Própria

Em *off_action*, definem-se os estados dos botões que determinam a desativação do motor, clicando-se no campo ? (interrogação), para digitar o estado do botão desliga e da saída como desativados, respectivamente Local:2:I.Data[1].6 e Local:2:O.Data[0].4, igualando-os a zero (fig. 299).

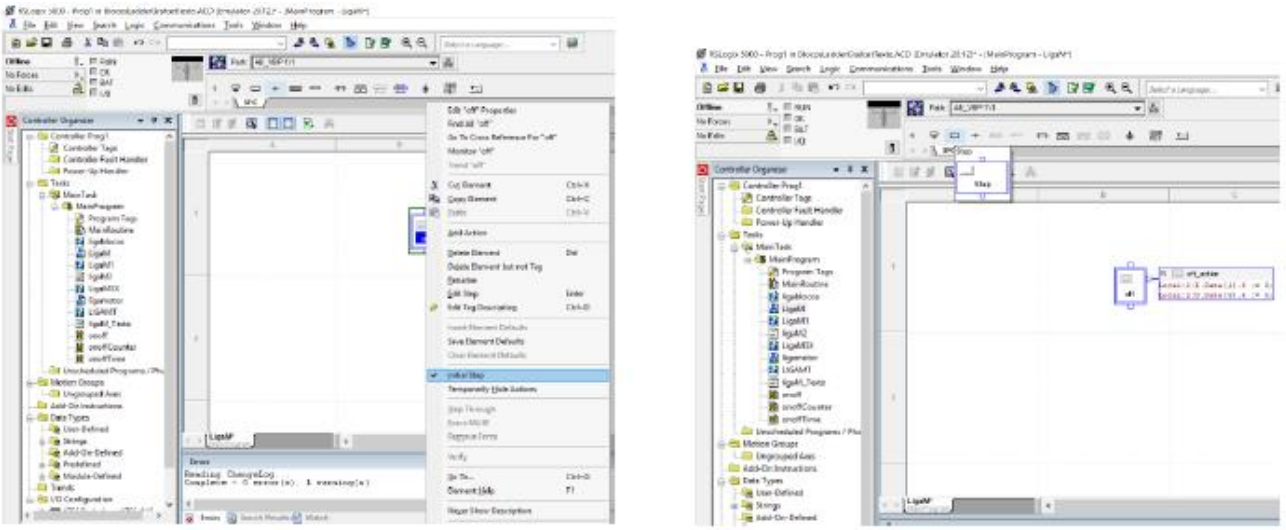
Figura 299 – Definição dos estados dos botões de off_action



Fonte: Própria

Clica-se com o botão direito sobre a etapa off, para defini-la como inicial. conforme figura 300.

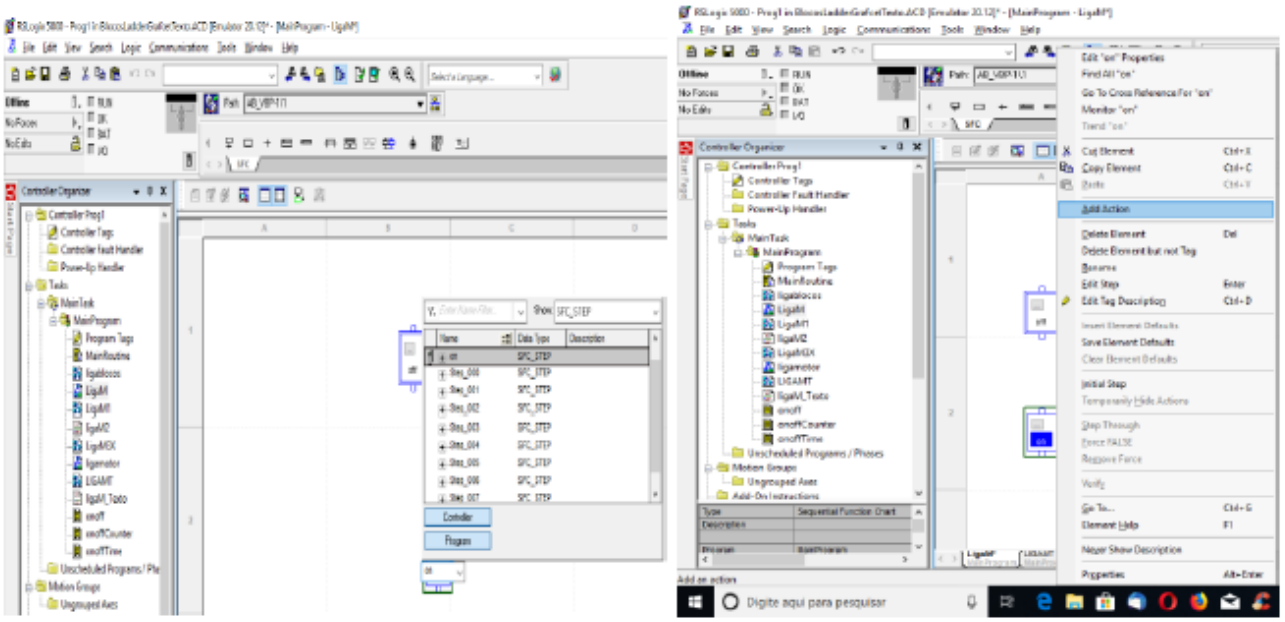
Figura 300 - Definição da etapa off



Fonte: Própria

Procede-se agora à criação da etapa on. Em seguida, seleciona-se o nome da etapa. Clica-se agora em Add action (fig. 301).

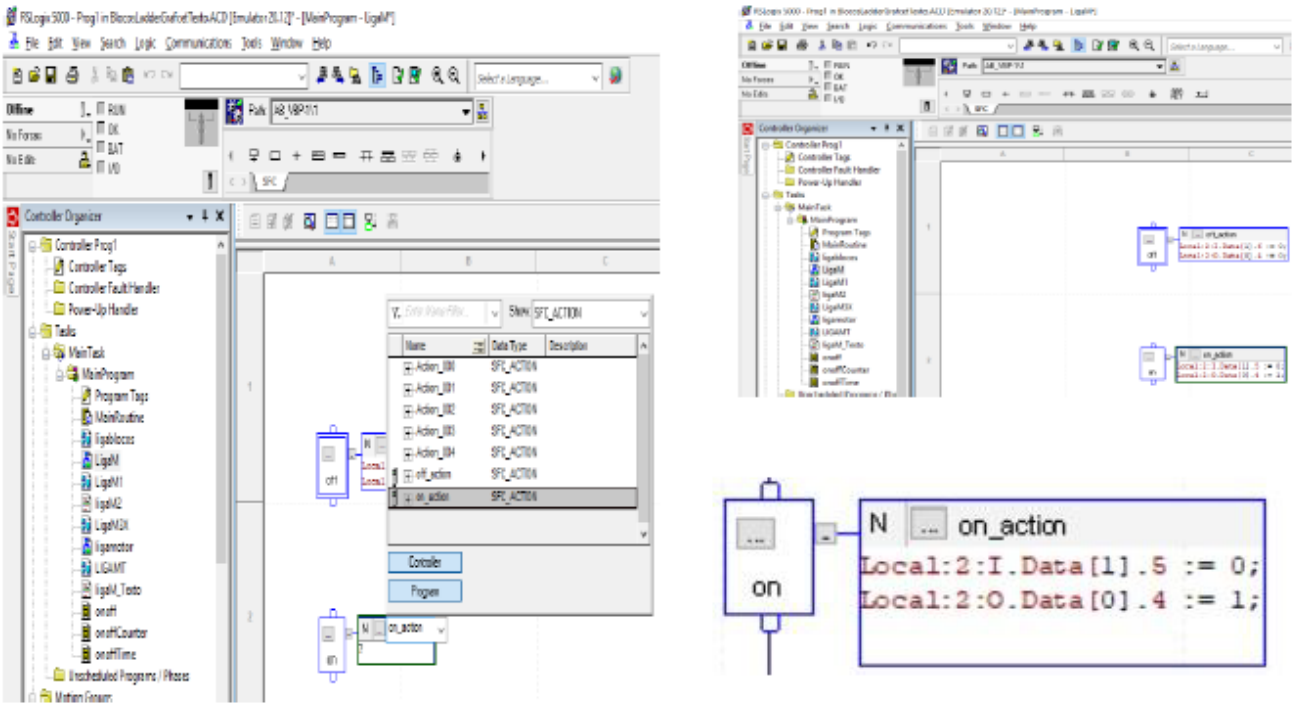
Figura 301 - Definição da etapa on



Fonte: Própria

Seleciona-se *on_action* (fig. 302), escolhendo como tipo de dados SFC_ACTION. Em on_action, definem-se os estados dos botões que determinam a ativação do motor, clicando-se no campo ? (interrogação), para digitar o estado do botão liga como desativado, igualando-o a zero (Local:2:I.Data[1].5 := 0), e da saída como ativada, igualando-a a um (Local:2:O.Data[0].4 := 1).

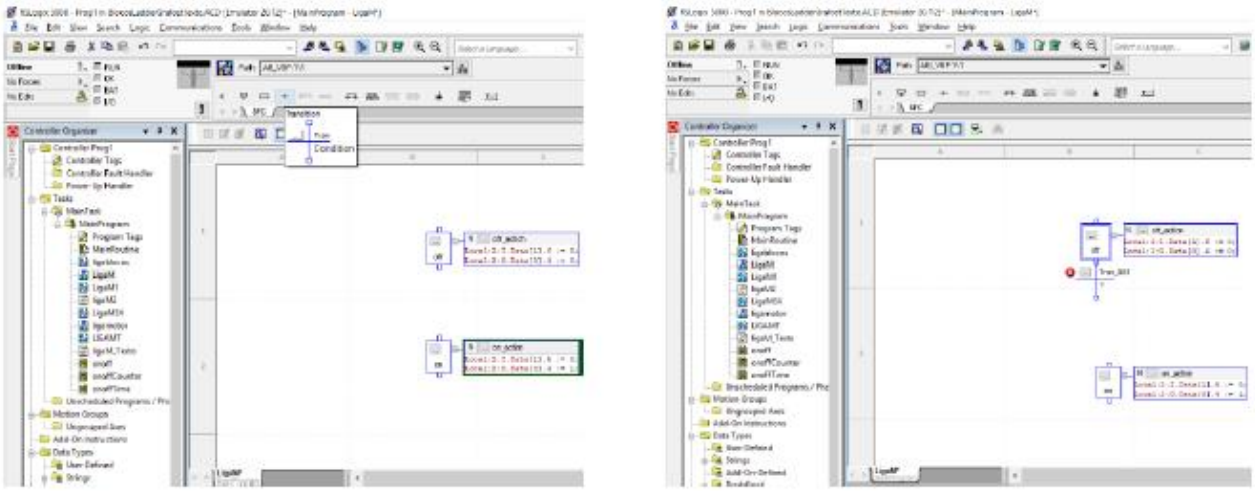
Figura 302 – Atribuindo tipo de dados à tag on_action



Fonte: Própria

O próximo passo consiste em adicionar uma transição entre os estados *off* e on. A transição foi interligada automaticamente à primeira etapa, que é a *off*, conforme figura 303.

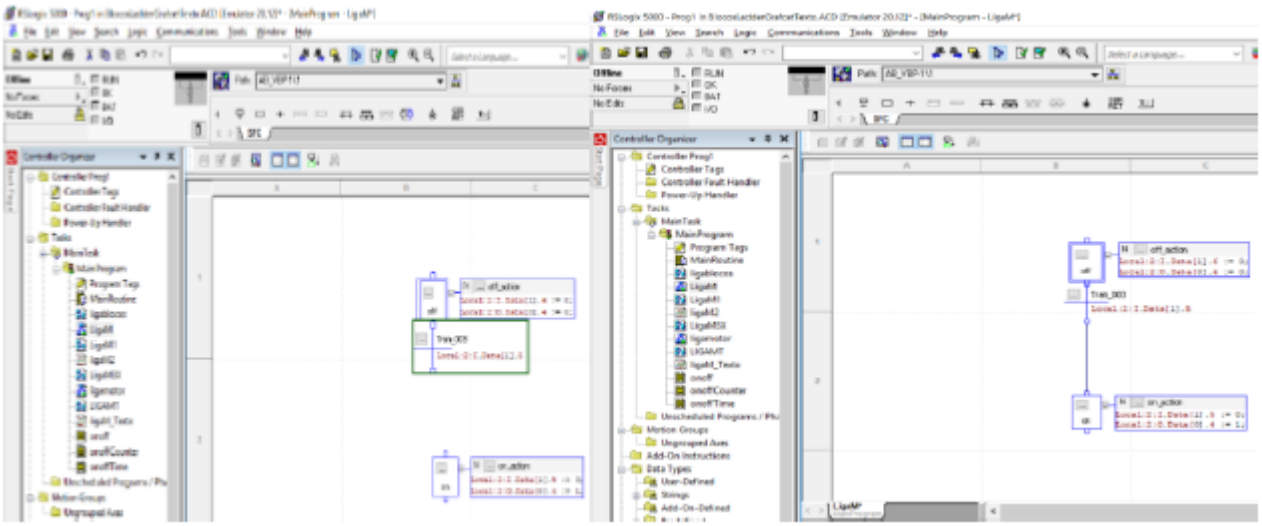
Figura 303 – Adicionando transição entre estados



Fonte: Própria

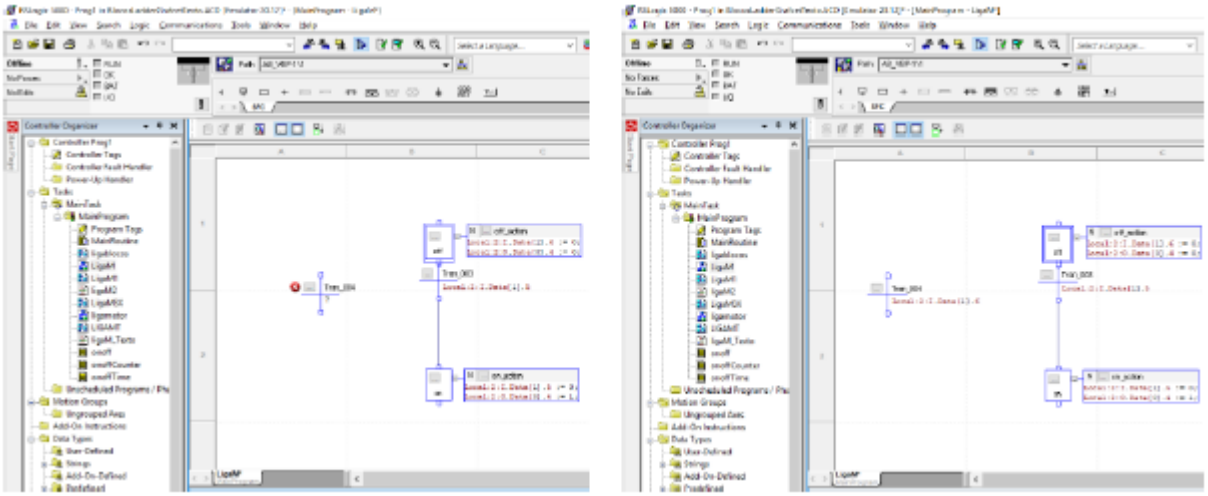
No campo interrogação (?) da transição, digita-se a condição que a torna verdadeira, ou seja, o pressionar do botão liga, definida por Local:2:I.Data[1].5 assumindo valor lógico verdadeiro (fig. 304). Interliga-se a parte de baixo da transição ao próximo estado.

Figura 304 – Atribuição das condições de transição



Cria-se agora nova transição, que levará o estado on ao off (fig. 305).

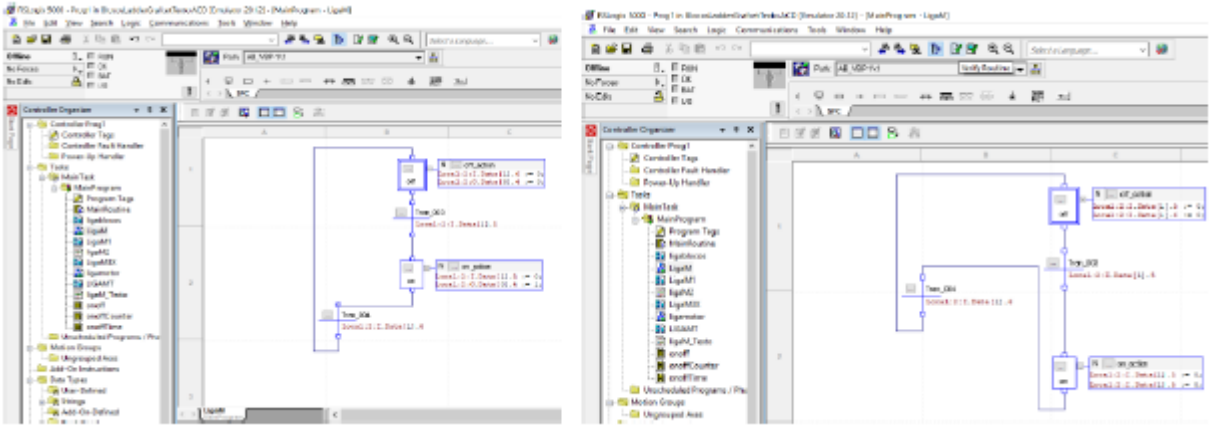
Figura 305 – Criando transição Tran_004Fonte: Própria



Fonte: Própria

A condição de ocorrência desta transição é que seja verdadeiro o pressionar do botão desliga, definido por Local:2:I.Data[1].6, conforme figura 306. Interliga-se agora a transição às etapas *on e off*. A partir deste ponto basta rodar a verificação do código, clicando na aba Verify Routine.

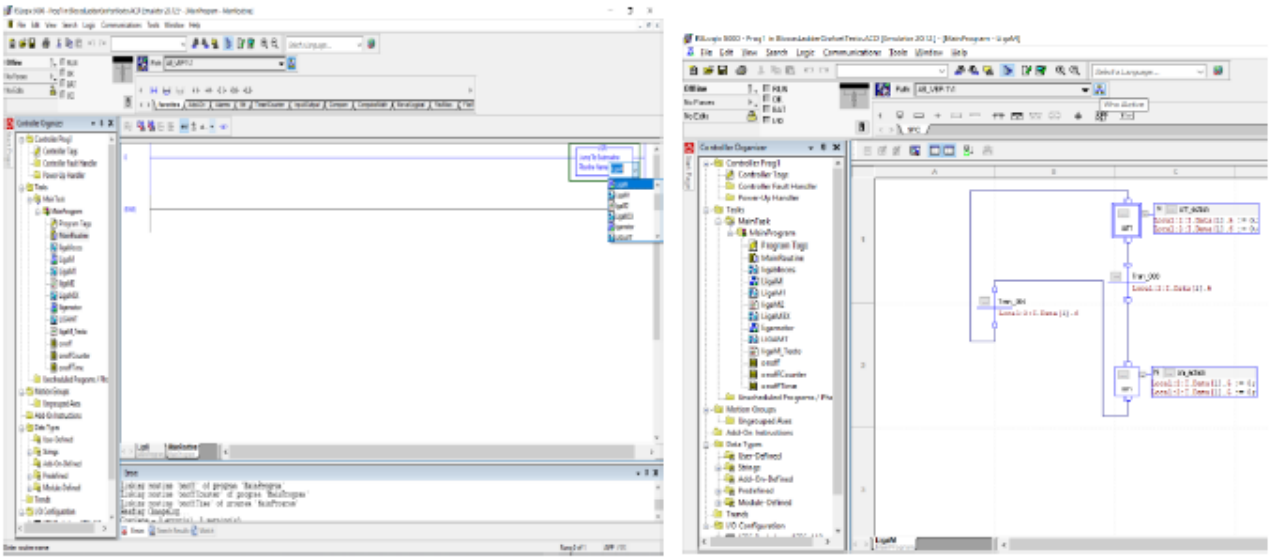
Figura 306 – Interligando Tran_004 às respectivas etapas



Fonte: Própria

Deve-se agora informar à rotina principal em linguagem ladder o nome da subrotina em SFC (que é LigaM), para o qual deve ocorrer o salto quando a mesma for ativada, através da instrução JSR (Jump to Subroutine). O próximo passo consiste em clicar no botão who active, conforme figura 307.

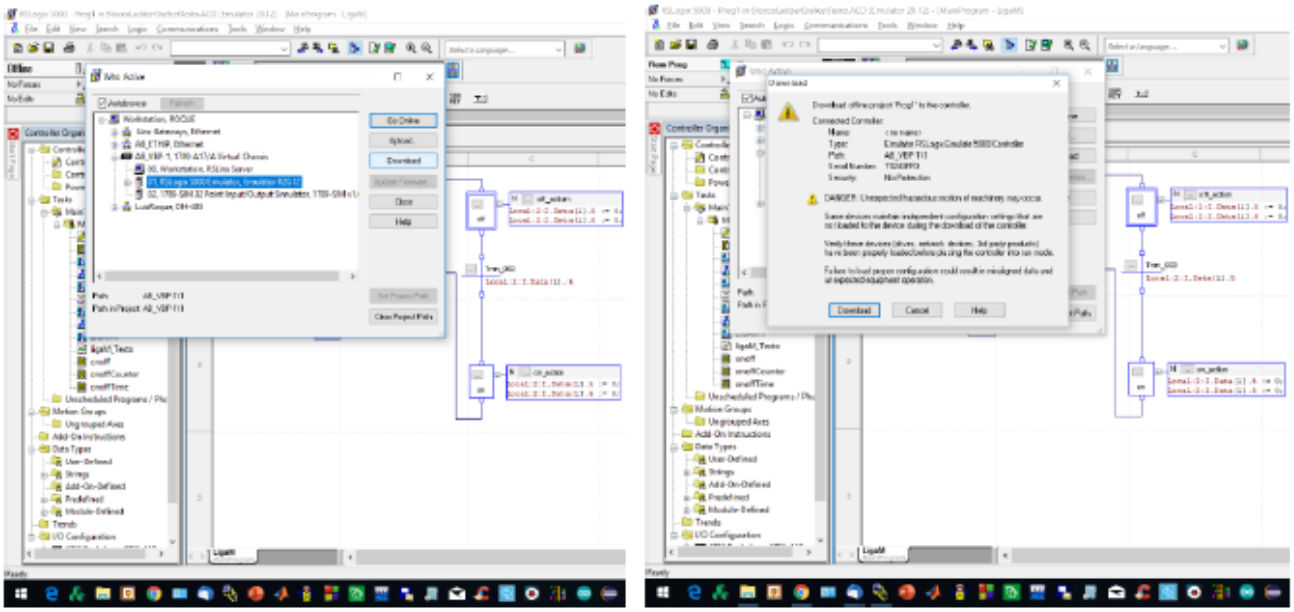
Figura 307 – Carregando o programa em grafcet na instrução JSR



Fonte: Própria

Seleciona-se o emulador para efetuar download, conforme figura 308

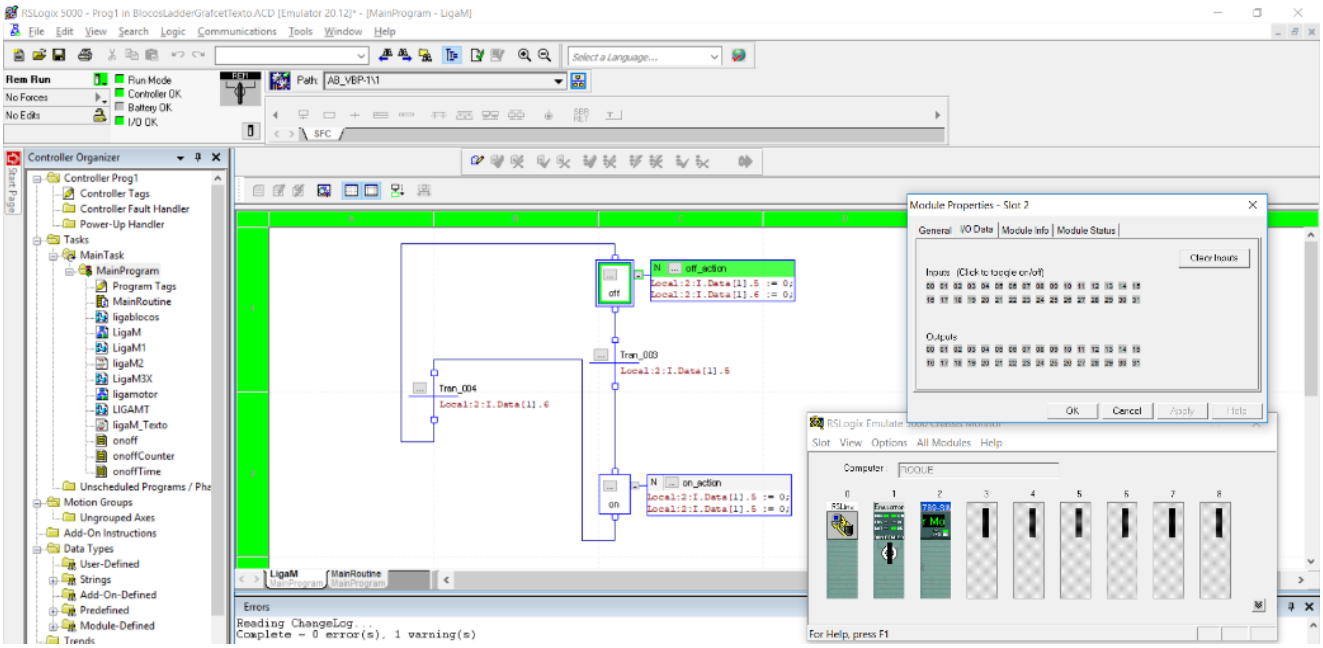
Figura 308 – Efetuando download do programa



Fonte: Própria

Basta selecionar Run Mode, após verificar que o programa está correto (fig. 309).

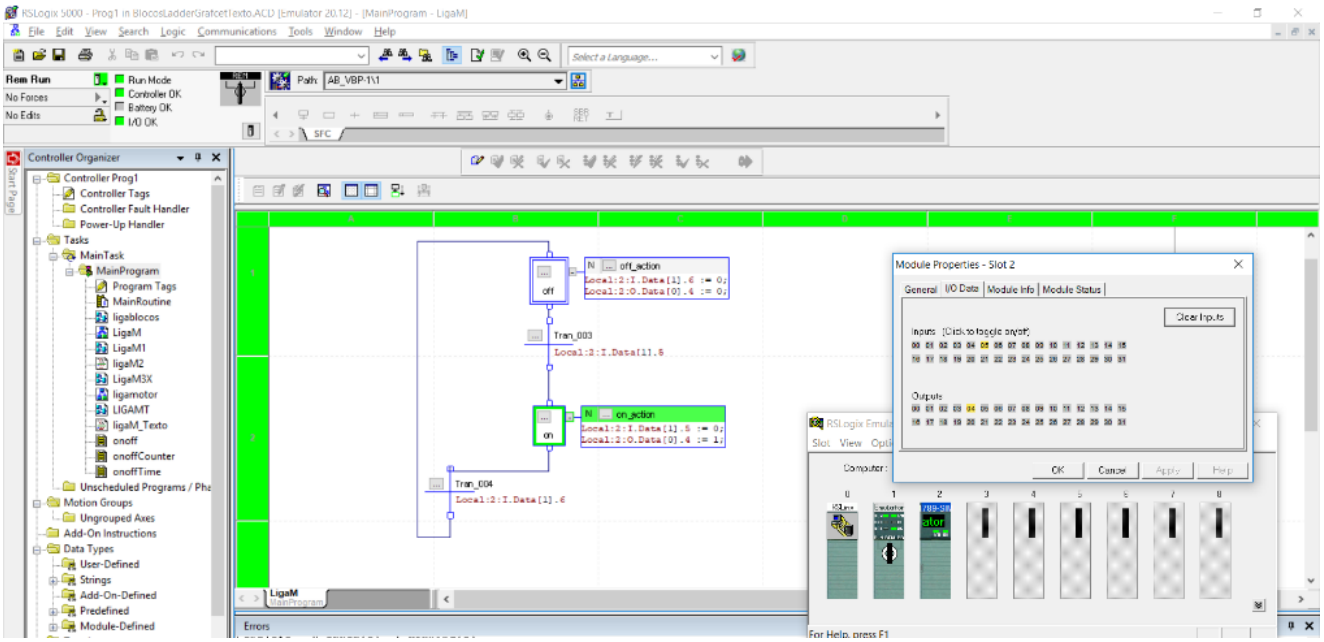
Figura 309 – Execução do programa Grafcet LigaM



Fonte: Própria

A etapa esverdeada é a inicial, cuja transição ocorre após o pressionar do botão liga, identificado pelo endereço Local:2:I.Data[1].5, conforme figura 310.

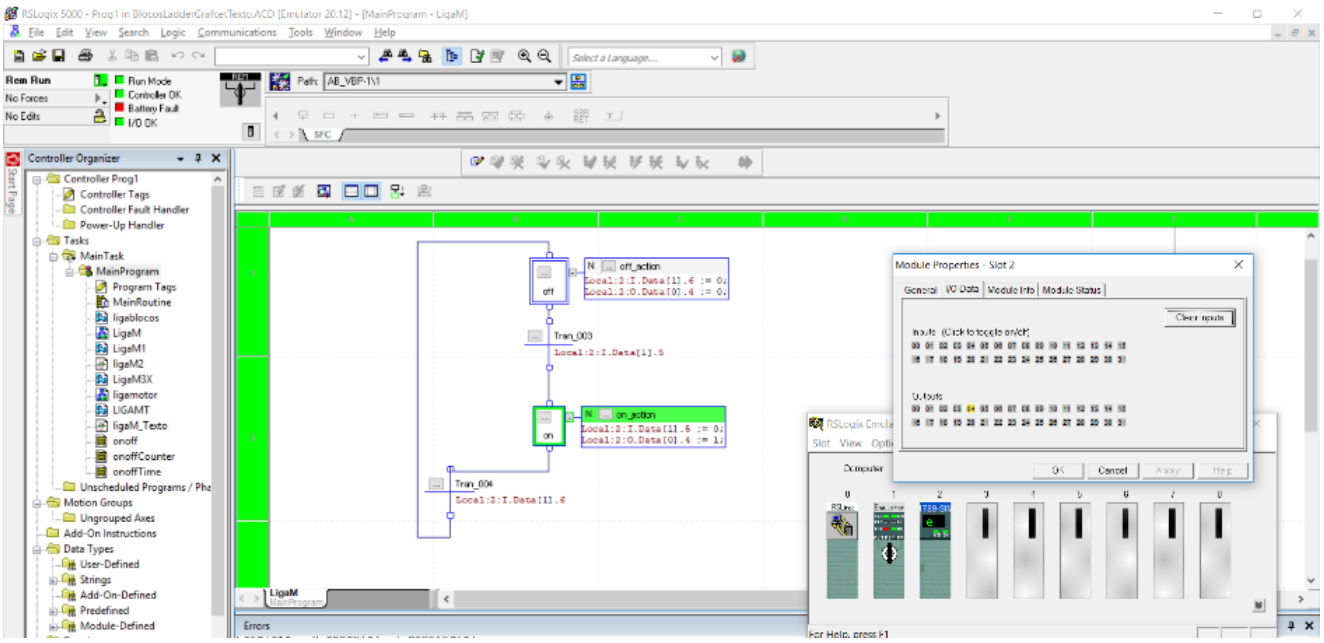
Figura 310 – Acionando o motor



Fonte: Própria

Solta-se então o botão liga, de acordo com a figura 311.

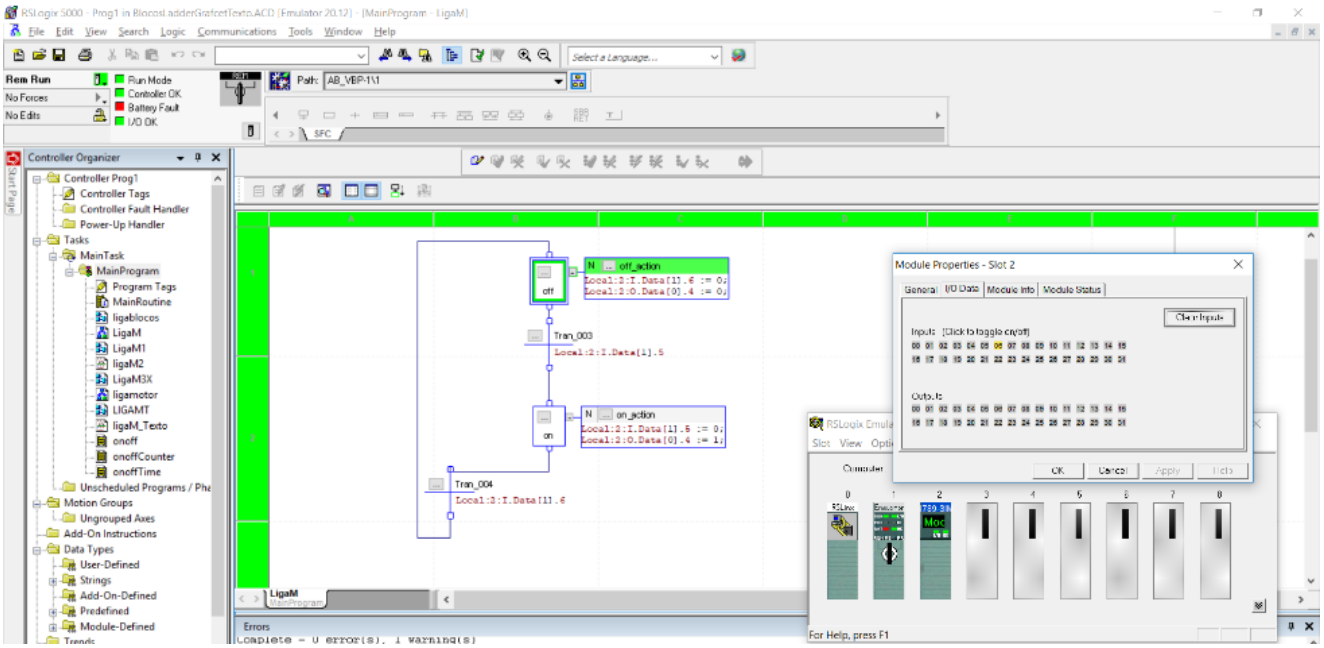
Figura 311 – Soltando o botão liga



Fonte: Própria

Para desligar o motor, basta pressionar o botão desliga, identificado pelo endereço Local:2:I.Data[1].6, conforme figura 312.

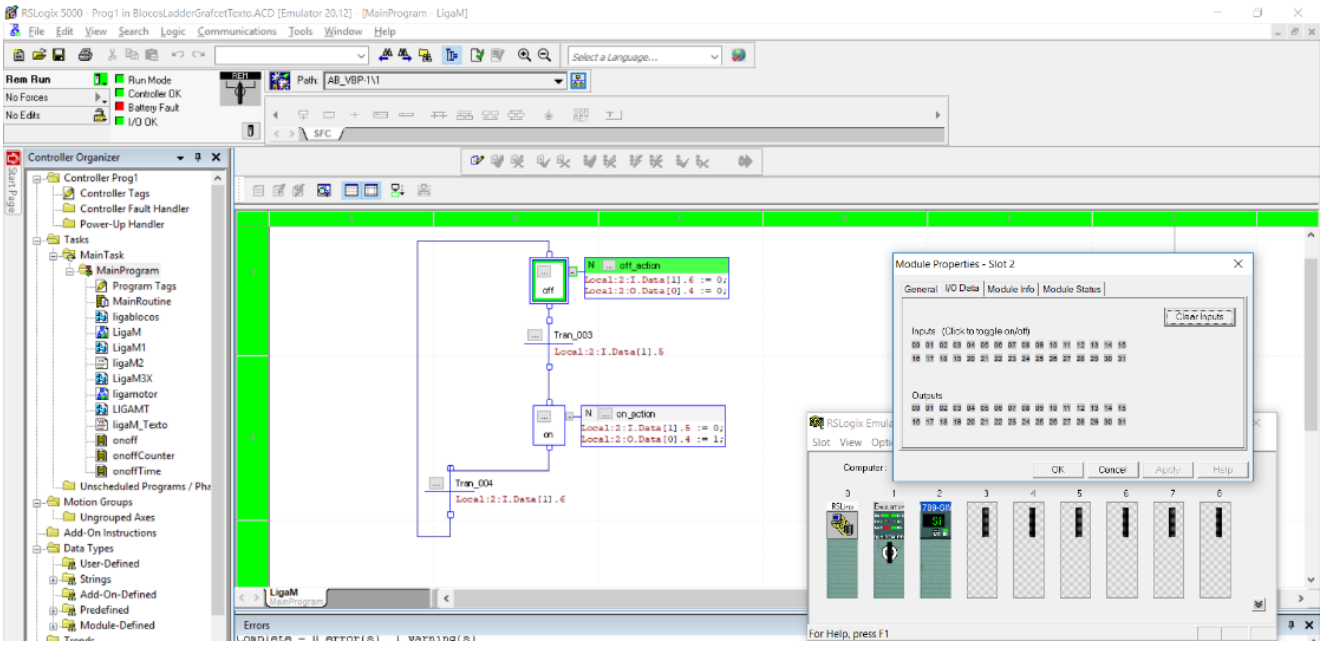
Figura 312 – Pressionando o botão desliga



Fonte: Própria

Basta agora soltar o botão desliga, de acordo com a figura 313.

Figura 313 – Soltando o botão desliga



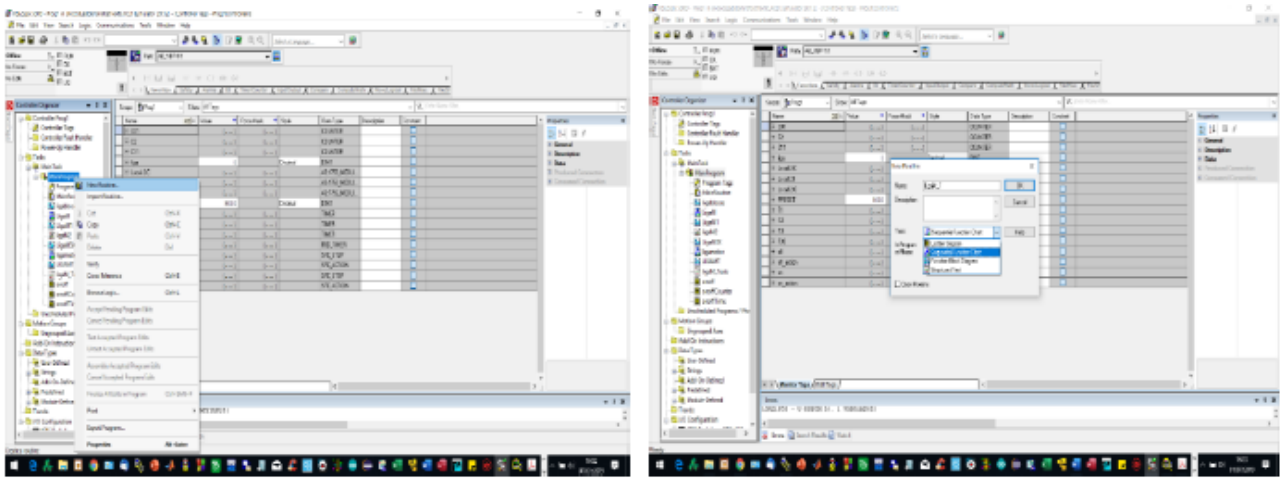
Fonte: Própria

2). Como próximo exercício em SFC, solicita-se elaborar código que permita ligar motor, desativando-o mesmo através de botão desliga ou automaticamente, decorridos dez segundos de seu

acionamento. O programa também deve ser automaticamente desligado após 2 acionamentos consecutivos.

Para tal fim, cria-se nova rotina em SFC, clicando-se com o botão direito sobre Main Program. A seguir, seleciona-se a opção New Rotine (figura 314), onde se escolhe a linguagem SFC, nomeando o código de LigaM_T.

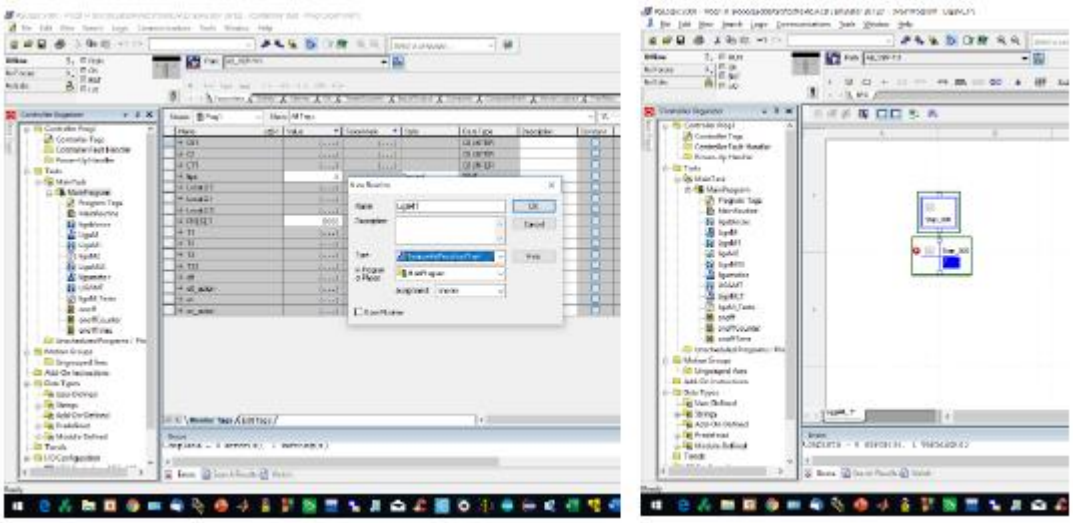
Figura 314 – Criando nova rotina em SFC



Fonte: Própria

Pressiona-se OK para obter a figura 315, na qual se apagam transição e etapa iniciais, cradas por padrão.

Figura 315 – Criando nova rotina em SFC

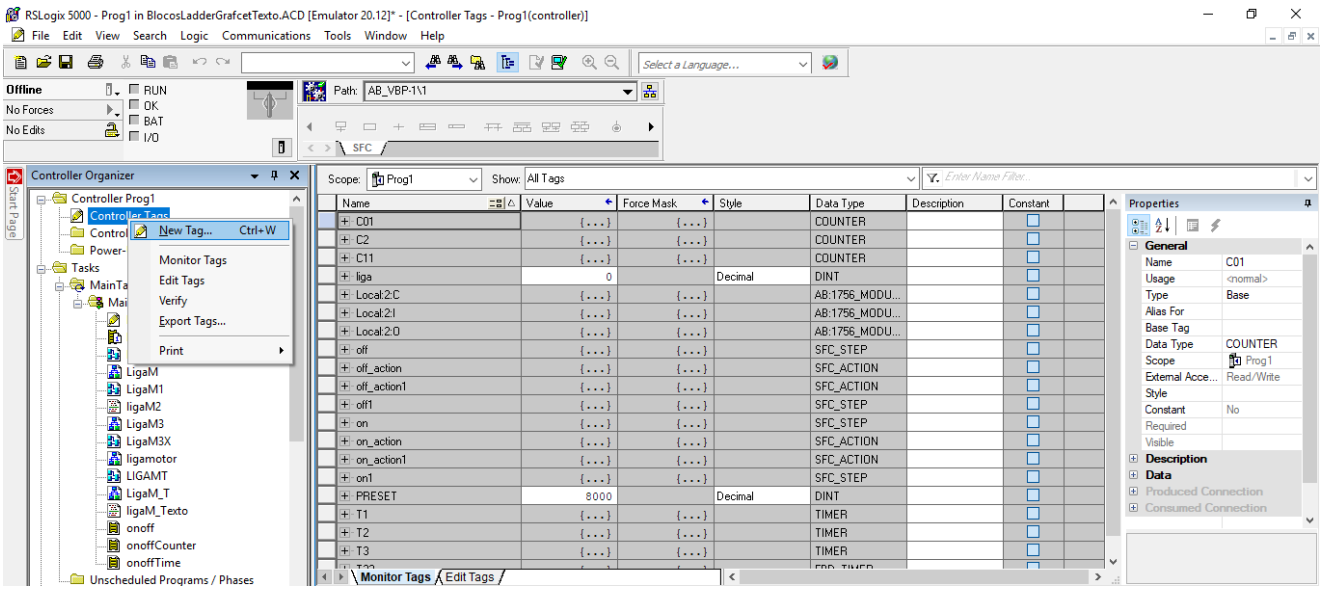


Fonte: Própria

A partir do problema anterior, cria-se novamente um código para ligar ou desligar um motor. Para tanto, criam-se duas instruções SFC_STEP no controlador, nomeando-as off2 e on2. Suas ações serão nomeadas off2_action e on2_action.

É preciso definir variável que contabilize os acionamentos da saída. Com esta finalidade, clica-se em *Controller Tags*, para selecionar a opção *New Tag*, conforme figuras 316 e 317.

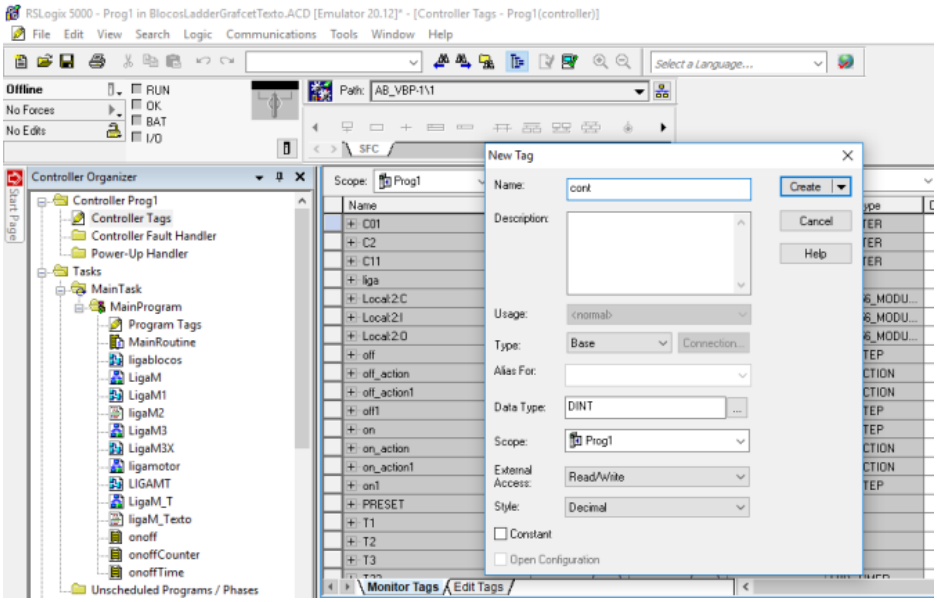
Figura 316 – Criando nova Tag



Fonte: Própria

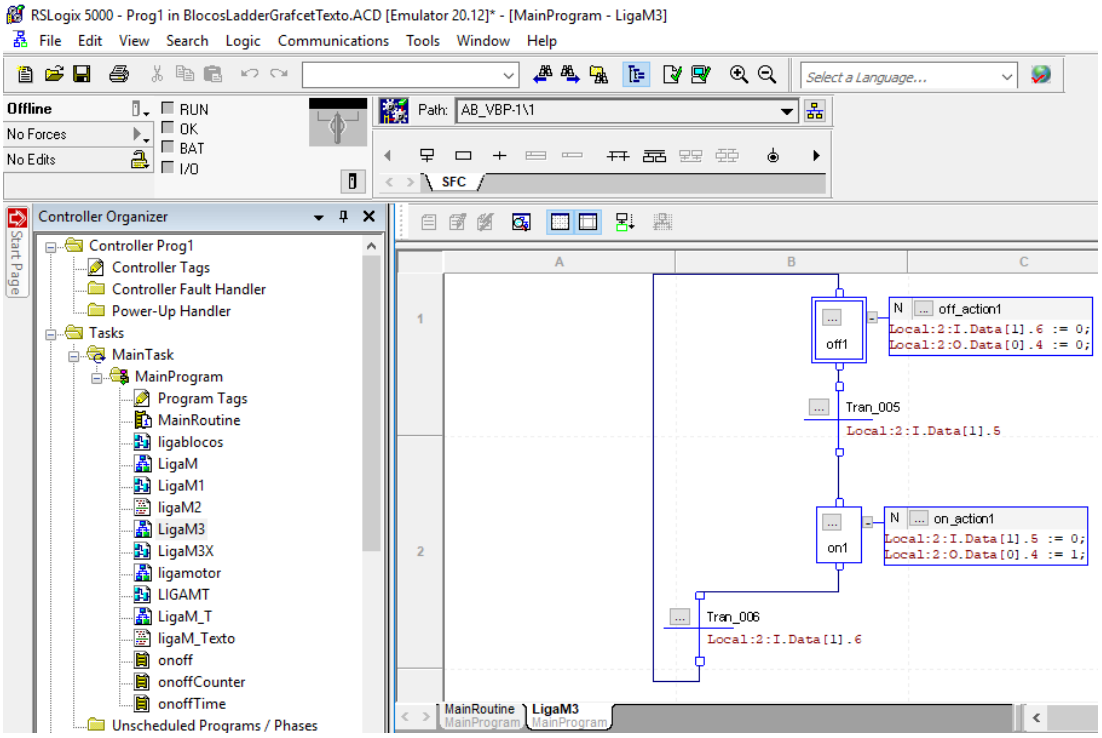
Deve-se selecionar variável do tipo DINT, que recebe o nome cont, criada após o pressionar do botão create.

Figura 317 – Criando Tag cont



Fonte: Própria

Figura 318 – Programa Liga3M

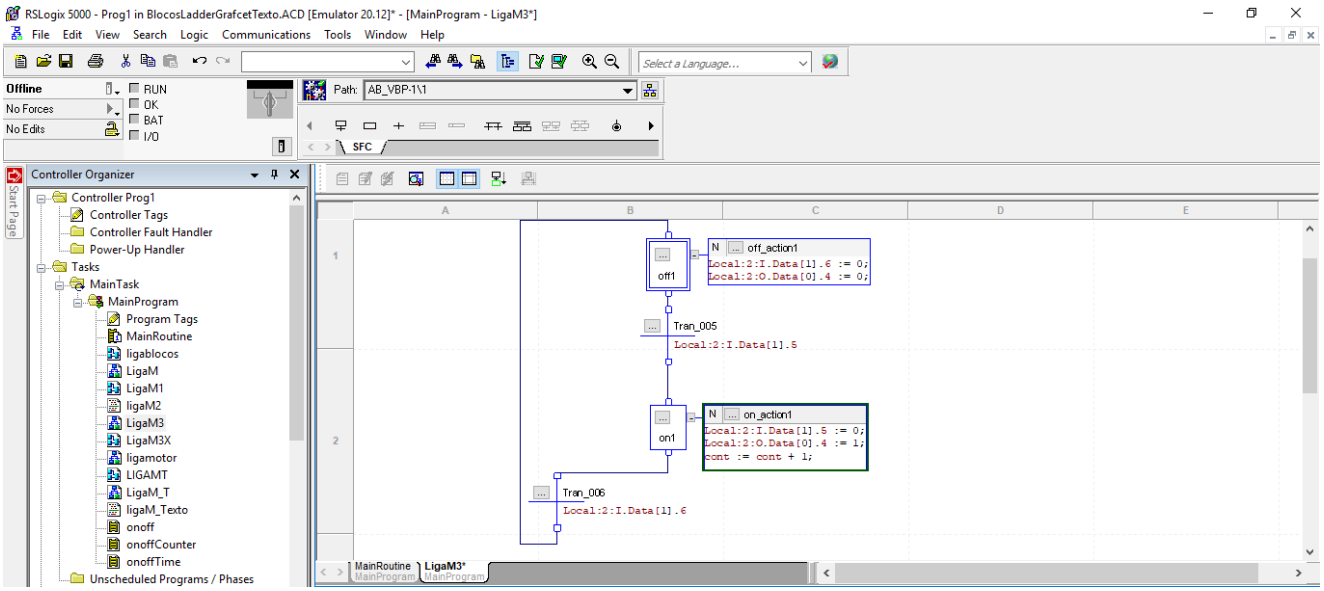


Fonte: Própria

Copia-se o programa do enunciado anterior (LigaM), colando-o com o nome LigaM3, para efetuar as alterações necessárias, conforme figura 318.

Toda vez que o motor atigir o estado on1, a variável cont será incrementada de uma unidade, conforme definido na ação on_action1 (fig. 319).

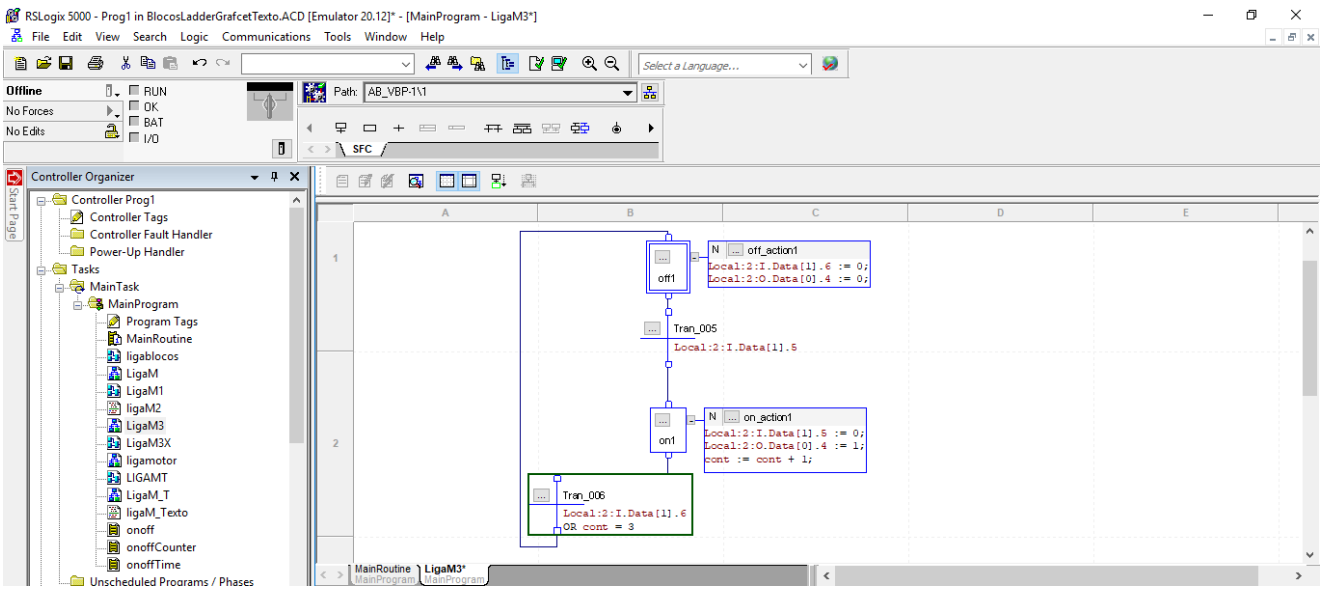
Figura 319 - Programa Liga3M com variável cont



Fonte: Própria

O próximo passo consiste em modificar a transição Tran_006 (fig. 320), de forma que a mesma inclua a possibilidade de desativar a saída, quando o botão 6 for pressionado, ou se cont for igual a 3, ou seja: Local:2:I.Data[1].6 OR cont = 3.

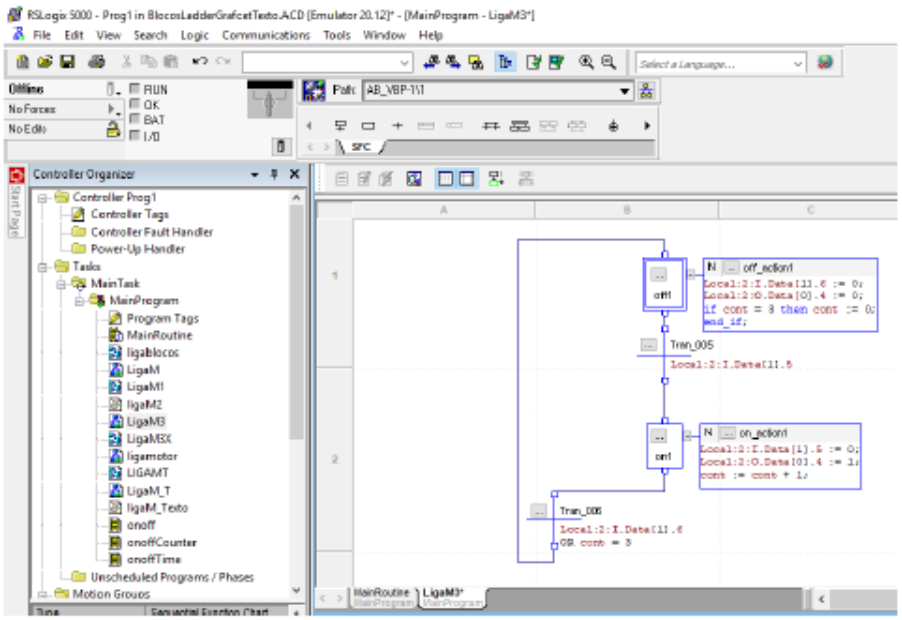
Figura 320 - Programa Liga3M com transição Tran_006 atualizada



Fonte: Própria

Para finalizar o código, é necessário incluir na ação associada à etapa inicial linhas que verifiquem se cont = 3 e, caso isto seja verdadeiro, a variável deve ser inicializada, recebendo valor zero. Dessa forma, a ação desativa o botão desliga e a saída, ficando conforme figura 321.

Figura 321 – Versão final do programa Liga3M

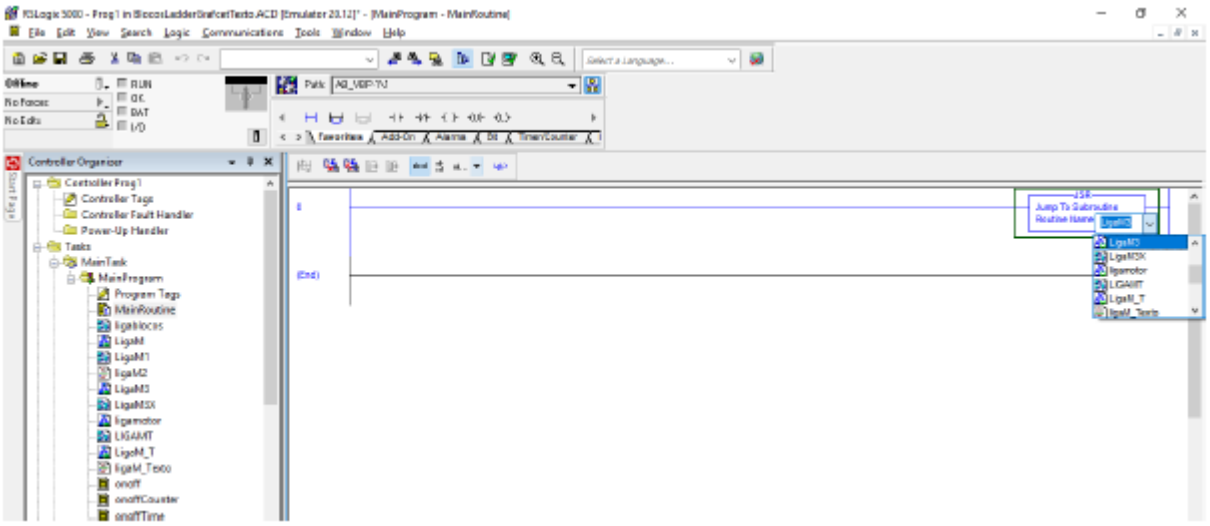


```
Local:2:I.Data[1].6 := 0;
Local:2:O.Data[0].4:= 0;
if cont = 3 then cont :=
0;
end_if;
```

Fonte: Própria

Em seguida é preciso modificar a rotina principal para o programa ladder realizar salto para o programa elaborado em SFC, cujo nome é LigaM3 (fig. 322).

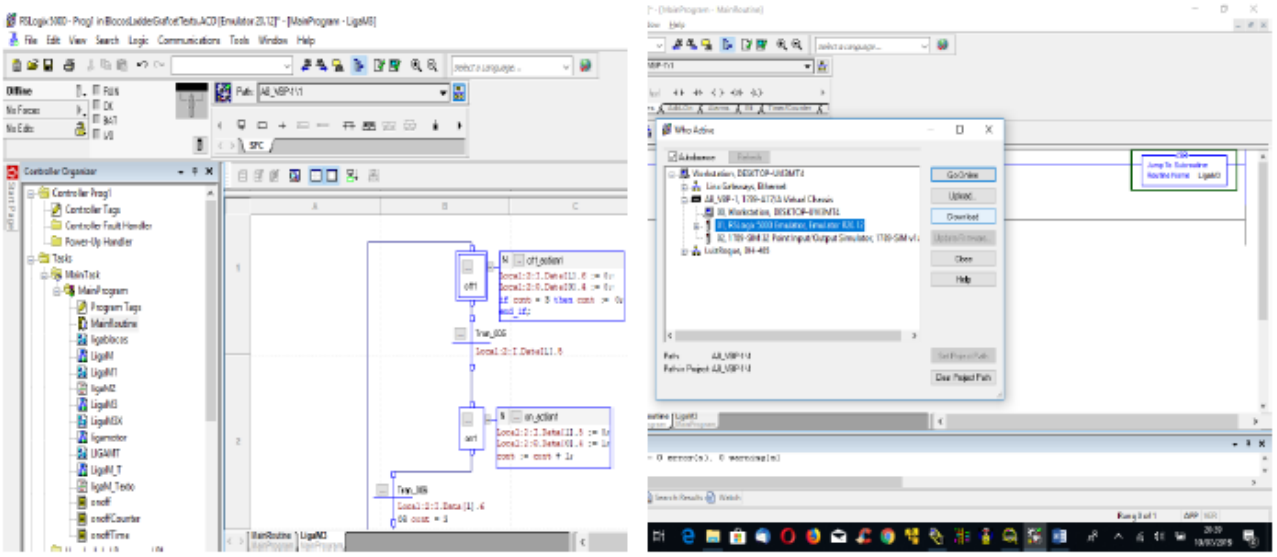
Figura 322 – Selecionando programa LigaM3 para carga na instrução JSR



Fonte: Própria

A partir desse ponto, basta realizar verificação de erros no código, efetuar download e rodar o programa desenvolvido em SFC,, de acordo com a figura 323.

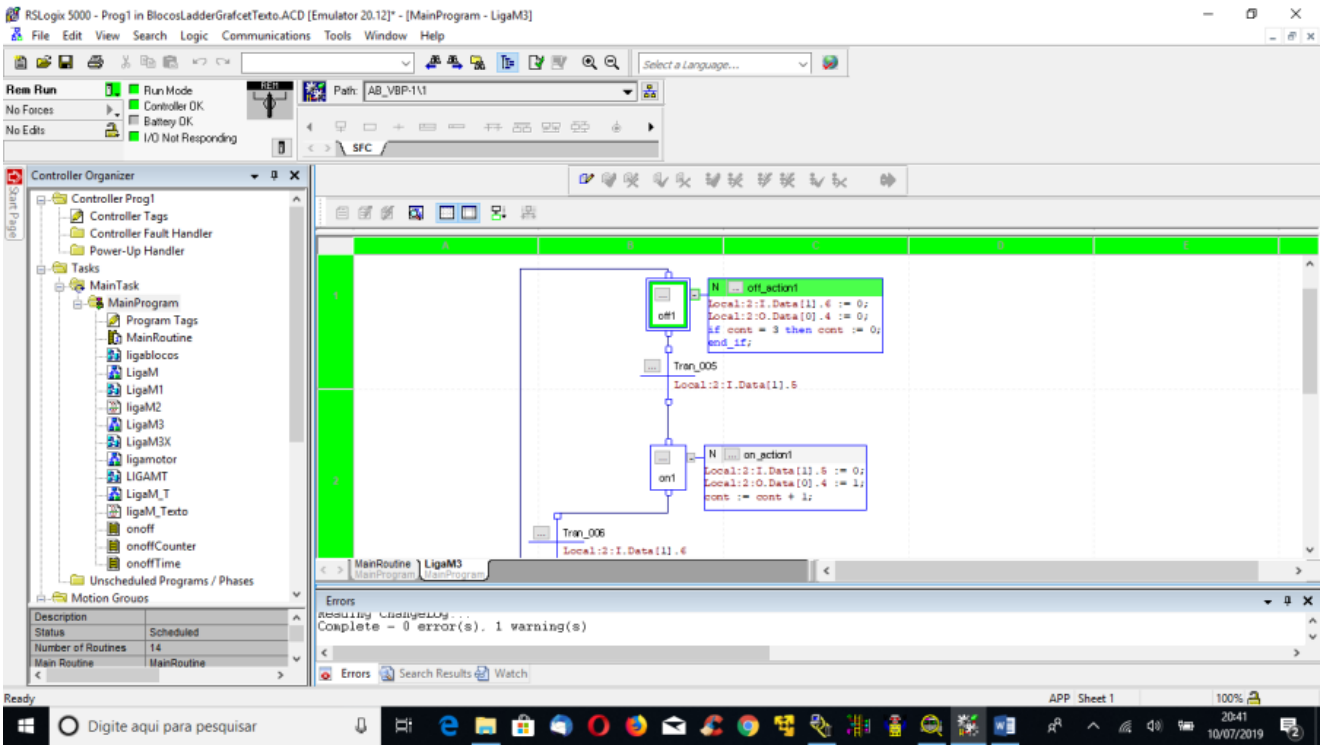
Figura 323 – Download do programa Liga M3



Fonte: Própria

A execução do programa pode ser vista na figura 324.

Figura 324 – Execução do programa LigaM3



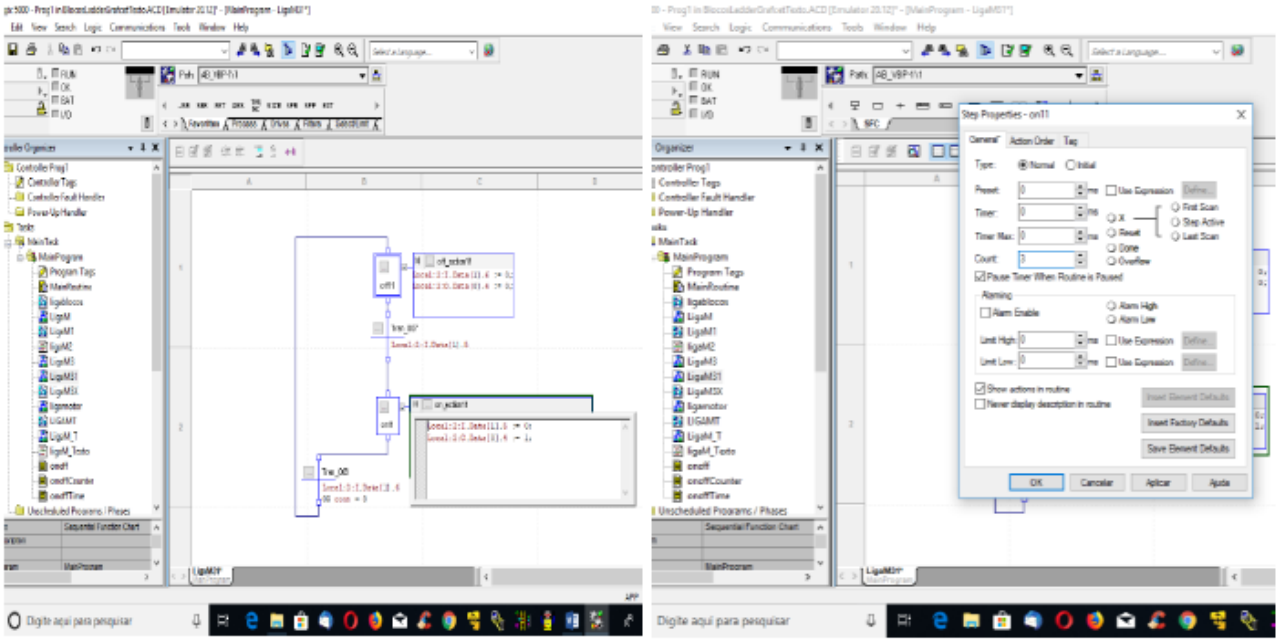
Fonte: Própria

Há um problema com essa forma de realizar contagem, pois a saída fica comutando entre acionada e desativada, após pressionar o botão de liga, mesmo enquanto não forem atingidos três acionamentos.

Essa falha pode ser resolvida abordando o problema de outra forma, de acordo com o próximo exercício, que será nomeado LigaM31, conforme figura 325, na qual são retiradas as estruturas condicionais (if) das ações. Clica-se duas vezes sobre a etapa on_11, definindo-se sua propriedade Count como igual a três. Modificam-se as condições da transição Tran_008 para:

Local:2:I.Data[1].6 OR on11.Count >= 3

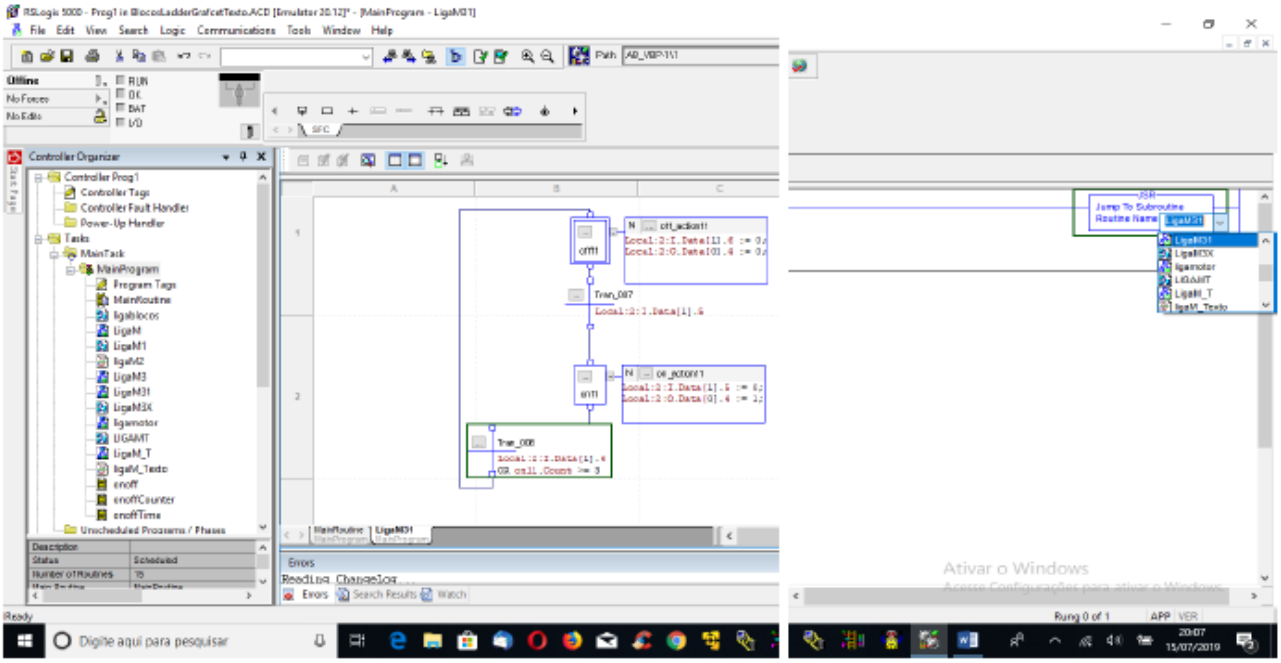
Figura 325 – Corrigindo comportamento inesperado do programa



Fonte: Própria

Isso significa que o motor será desativado quando o botão desliga for pressionado, ou assim que o terceiro acionamento for detectado (fig. 326). Deve-se agora clicar em Main routine, e selecionar na instrução JSR (do programa ladder principal) o código LigaM31 para realizar o salto.

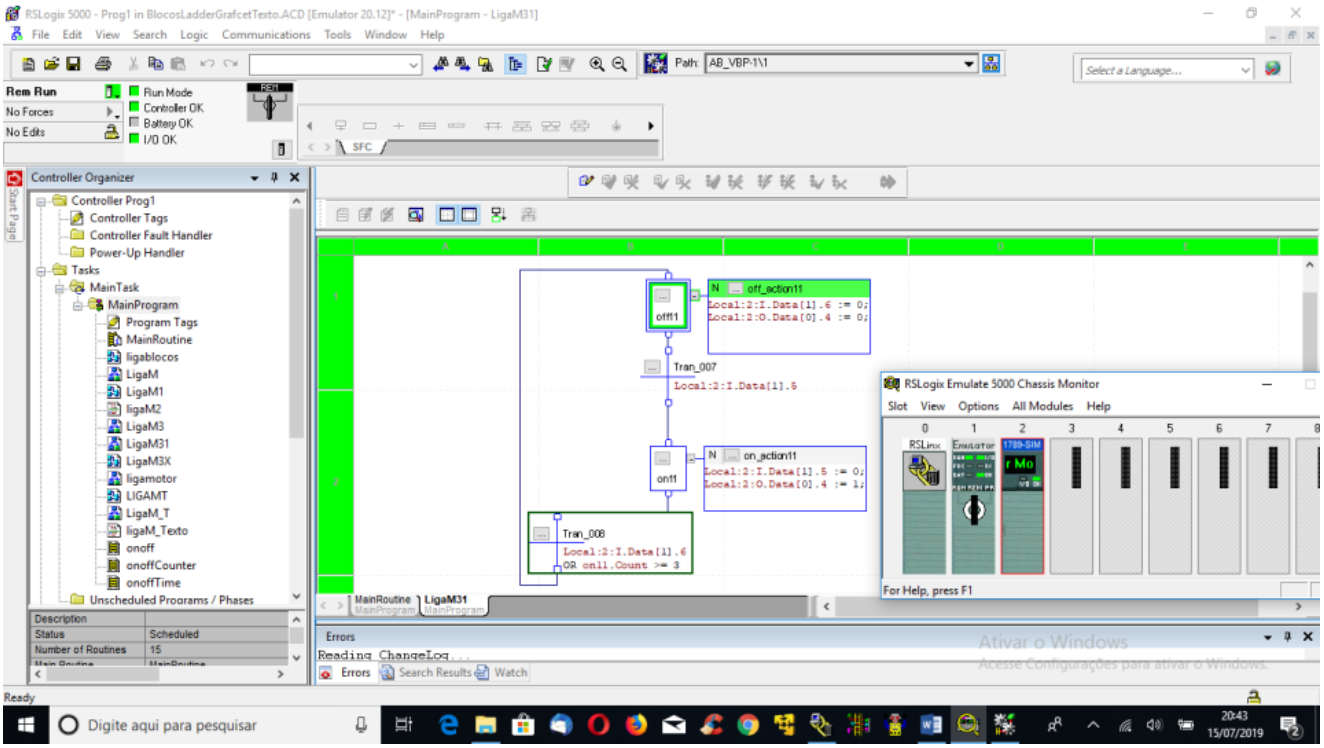
Figura 326 – Versão final do programa LigaM31



Fonte: Própria

Basta agora efetuar o download, de acordo com a figura 327.

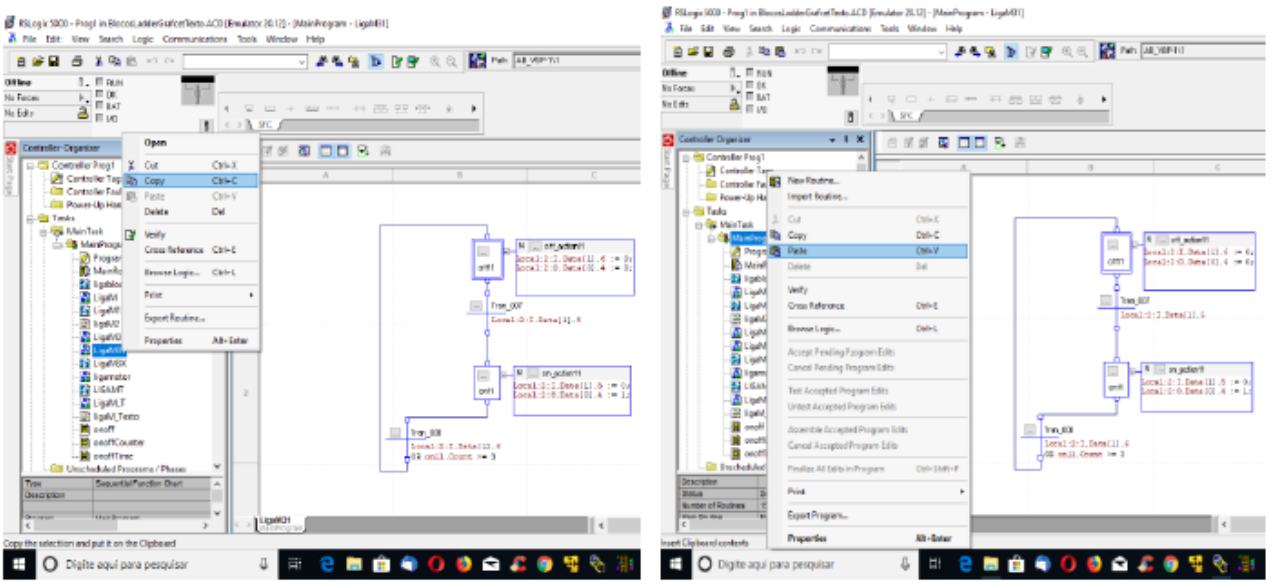
Figura 327 – Programa LigaM31 em execução



Fonte: Própria

O exercício anterior poderia ser modificado para desativar automaticamente a saída, após 30 segundos de funcionamento da mesma. Para tanto, clica-se sobre o programa LigaM31, selecionando-se a opção copy (fig. 328). Clica-se agora com o botão direito sobre a aba Main Program, marcando-se a opção Paste.

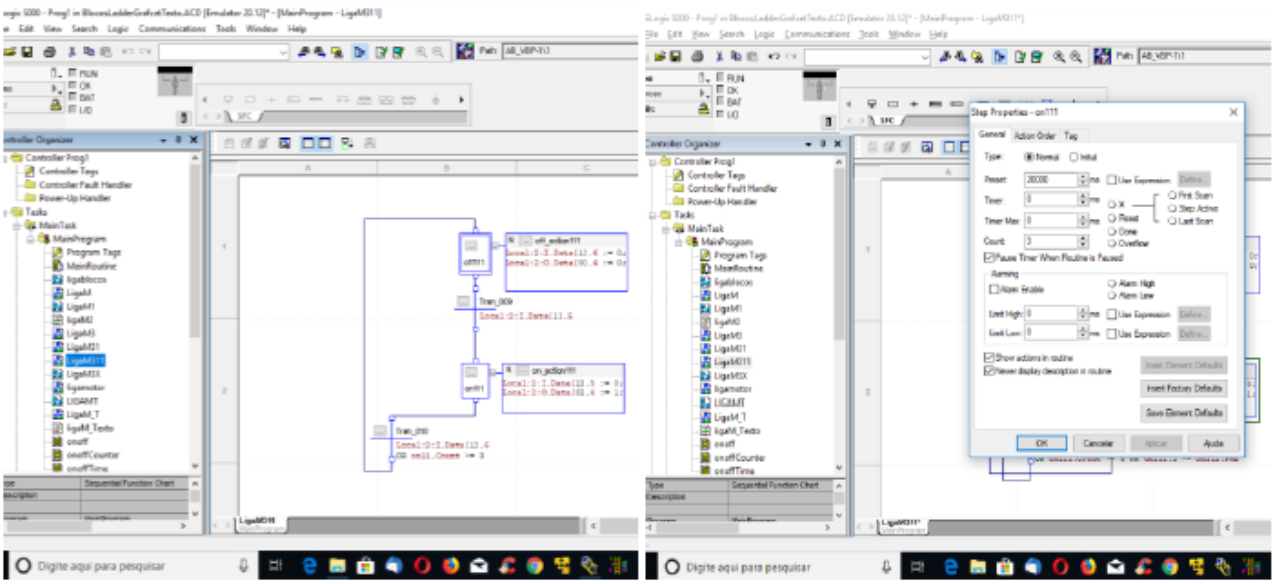
Figura 328 – Salvando programa Liga311



Fonte: Própria

Isso fará com que o arquivo colado receba o nome de LigaM311. Clica-se duas vezes sobre a etapa on111, referente à ativação do motor, e ajusta-se o tempo de funcionamento dela para 20000mS, ou vinte segundos (fig. 329).

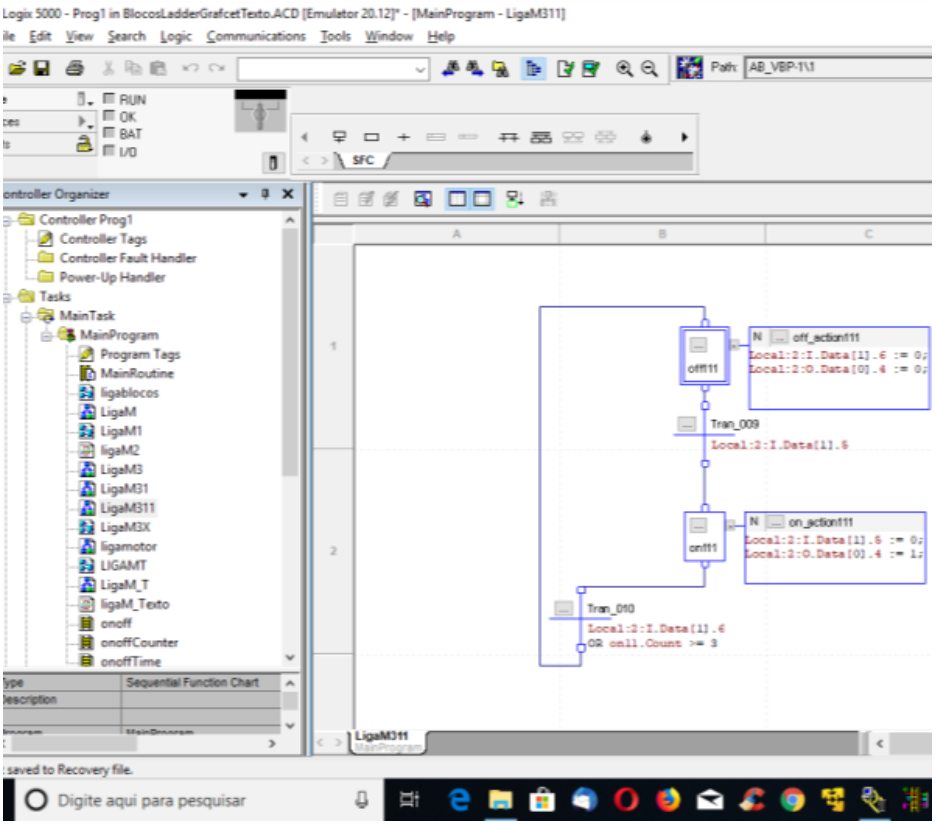
Figura 329 – Programa LigaM311



Fonte: Própria

Clica-se em OK, obtendo a figura 330.

Figura 330 – Programa LigaM311



Fonte: Própria

Em seguida, modifica-se a transição Tran_010 para:

Local:2:I.Data[1].6 OR on111.Count >= 3 OR on111.T >= on111.PRE

Isso significa que o motor pode ser desligado por três condições:

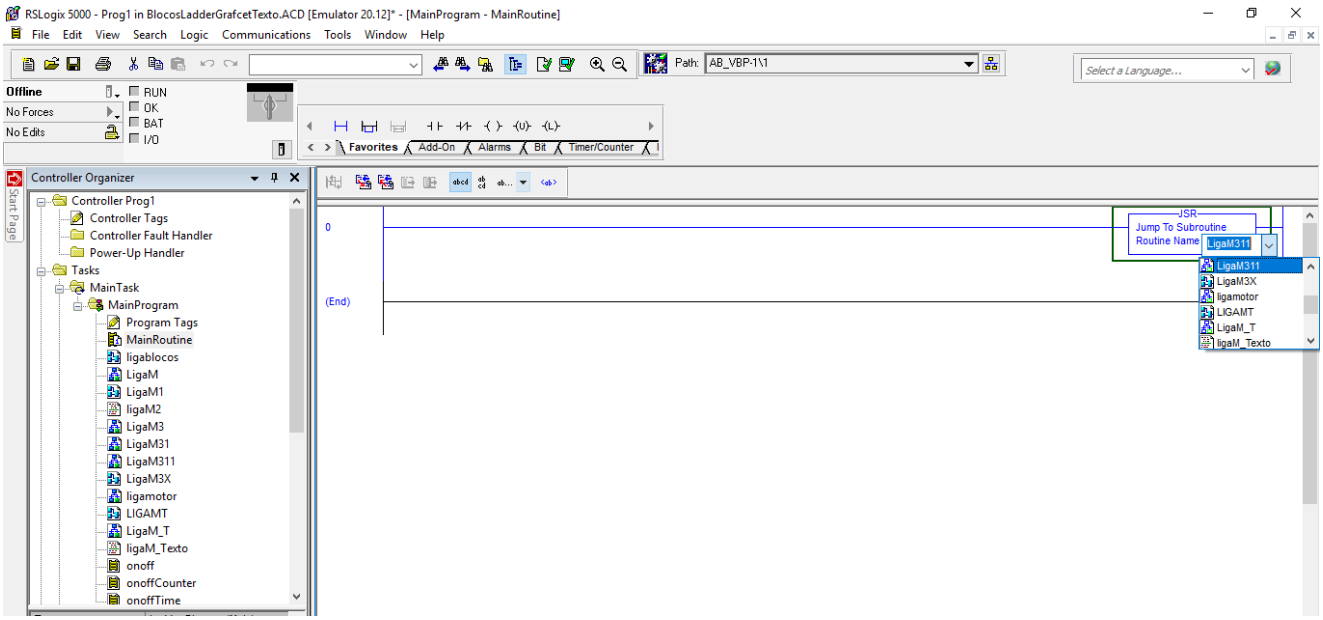
1) Pressionar o botão desliga (Local:2:I.Data[1].6).

2) Ligar o motor três vezes (on111.Count >= 3).

3) Depois que o tempo decorrido atingir o limite estabelecido em Preset, de 20 segundos (on111.T >= on111.PRE).

A partir daqui abre-se a rotina principal e, na instrução JSR, seleciona-se a rotina desenvolvida em grafcet, ou seja, LigaM311.

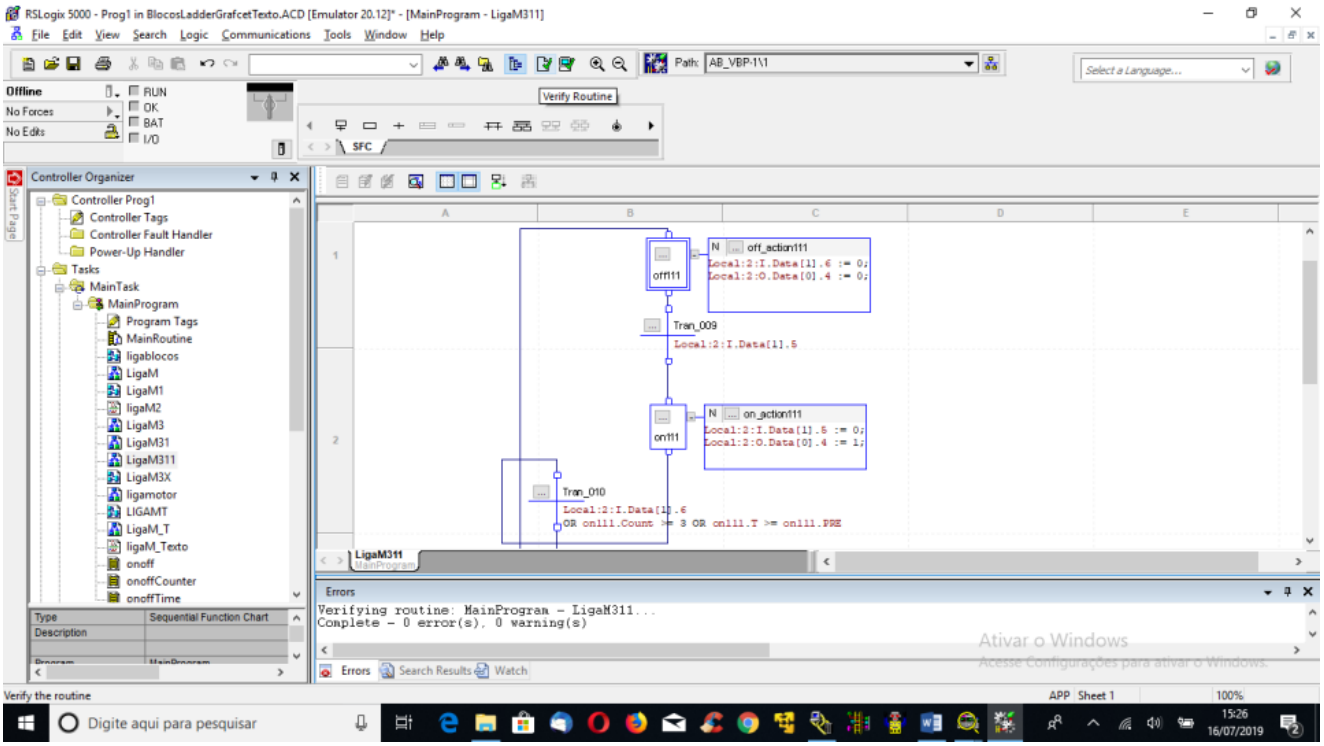
Figura 331 – Carregando programa LigaM311 na instrução JSR



Fonte: Própria

A partir deste ponto verifica-se o projeto e, caso não haja erro, efetua-se download (Fig. 332).

Figura 332 – Programa LigaM331 em execução



Fonte: Própria

4 CONSIDERAÇÕES FINAIS

Pretendeu-se nestas laudas apresentar problemas típicos de automação de processos, envolvendo linguagens Ladder, Diagramas de blocos de funções (FBD) e Grafcet. Relevante parcela dos programas desenvolvidos foi integrada com o sistema supervisório In Touch.

O ambiente de desenvolvimento integrado (IDE) utilizado foi o RSLogix 500, atualmente conhecido como Logix Studio, que permite programar nas três supracitadas linguagens, e também numa quarta, conhecida por texto estruturado (ST). Trata-se de uma linguagem literal, semelhante à programação em C. Dessa forma, propõe-se como publicações futuras a inclusão da automação de processos industriais em texto estruturado, juntamente com sua integração ao sistema supervisório intouch.

A próxima obra lançada também abrangerá automação de processos no IDE Ecoxtruture, desenvolvido pela multinacional de origem francesa Schneider Electric, englobando programação em linguagens Ladder e Lista de Instruções (IL). IL é semelhante à linguagem de montagem (Assembly Language), de caráter literal, onde as operações são realizadas por mnemônicos, em função do conteúdo armazenado em registro de memória.

Esse livro é o quinto do autor, que também pretende incluir em sua sexta obra já em desenvolvimento, a programação de controladores lógicos programáveis (CLP) da fabricante chinesa Haiwell, realizada através do IDE Haiwell Happy, juntamente com seu respectivo sistema supervisório, Haiwell Scada.

EDITORA CONHECIMENTO LIVRE

conhecimentolivre.org/home
contato@conhecimentolivre.org
editoraconhecimentolivre



AUTOMAÇÃO DE PROCESSOS COM AS LINGUAGENS DEFINIDAS PELA NORMA IEC 61131-3, ATRAVÉS DOS AMBIENTES DE DESENVOLVIMENTO INTEGRADOS RSLOGIX 5000 E INTOUCH

Luiz Alberto Oliveira Lima Roque