UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Ilha Solteira

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

"LOGO2VHDL: MODELOS DESCRITOS EM VHDL A PARTIR DA LINGUAGEM DO LOGO!SOFT COMFORT DA SIEMENS"

RENATO CARDOSO DOS SANTOS

Orientador: Prof. Dr. Alexandre César Rodrigues da Silva

Co-orientador: Prof. Dr. Carlos Antonio Alves

Dissertação apresentada à Faculdade de Engenharia - UNESP – Campus de Ilha Solteira, para obtenção do título de Mestre em Engenharia Elétrica.
Área de Conhecimento: Automação.

Ilha Solteira – SP
setembro/2007

Dedico esse trabalho aos meus avós maternos José dos Santos e Gasparina Vieira, que sempre foram exemplos de amor, carinho, determinação e perseverança.

Agradecimentos

Este trabalho se deve em muito a algumas pessoas, por diversas razões, e eu gostaria de agradecer especialmente:

- *Aos meus tios, Washington Luiz Bráz e Maria de Fátima Braz, pelo grande incentivo e motivação, que me proporcionou a continuidade nos estudos, até a chegada a este Mestrado.*
- *Ao meu orientador, professor Alexandre César Rodrigues da Silva, sempre disposto a oferecer estímulos e indicando a direção a ser tomada, nos momentos de maior dificuldade. Agradeço, principalmente, pela confiança, depositada no trabalho de dissertação realizado.*
- *Ao companheiro Marco Antonini, que participou comigo em vários momentos importantes e que sempre me inspirou confiança nos momentos mais difíceis.*
- *A todos os docentes, funcionários e estagiários do Departamento de Engenharia Elétrica da Unesp.*
- *Aos meus colegas Rodrigo Sato, Rodrigo Serra Daltin e Carolina Tucunduva, pessoas com quem interagi tantos anos e que, sempre, se demonstraram dispostos e solidários no convívio em Ilha Solteira.*
- *Ao querido amigo Flávio Meno, pela sua amizade, companheirismo e uma irmandade toda especial, que sempre partilhamos.*

A todos agradeço profundamente e dedico o resultado deste trabalho.

Resumo

Neste trabalho é apresentada uma ferramenta de tradução, que converte sistemas de controle descritos na linguagem de automação LOGO!Soft, para um modelo VHDL correspondente. O software desenvolvido, denominado "LOGO^{2}VHDL", contém funções básicas e especiais disponíveis no LOGO!Soft. Nesta ferramenta, o usuário acostumado em programar o CLP LOGO!Soft pode facilmente obter uma descrição VHDL cujo modelo funcional, pode ser sintetizado, no ambiente QUARTUS II da Altera. Este trabalho teve como objetivo principal estudar uma nova metodologia, que visa o emprego de dispositivos lógicos programáveis (PLDs) como uma forma alternativa ao emprego dos controladores lógicos programáveis (CLPs) no controle automatizado de processos. A ferramenta foi avaliada através de estudos de casos descrevendo sistemas de controle simples e complexos. Em todos os casos, os resultados das simulações mostram a viabilidade desta nova abordagem em automatizar sistemas de controle.

Palavras Chave: VHDL, LOGO!Soft, Automação, CLP, Delphi.

Abstract

In this work it is presented a translation tool that converts control systems described in the automation language LOGO!Sof, for a model corresponding VHDL. The developed software, denominated "LOGO^{2}VHDL", contains basic and special functions available in LOGO!Soft. In this tool, the accustomed user in programming the CLP LOGO!Soft can easily obtain a description VHDL whose functional model can be synthecized in the environment QUARTUS II of the Altera. This work had as main objective to study a new methodology that seeks the employment of programmable logical devices (PLDs) as an alternative form to the programmable logical controllers' employment (CLPs) in the automated control of processes. The tool was evaluated through studies of cases describing simple and complex control systems. In all the cases, the results of the simulations show the viability of that new approach in automating control systems.

Key words: VHDL, LOGO!Soft, Automation, CLP, Delphi

Lista de Ilustrações

Sumário

Capítulo 1

Introdução Geral

1.1. Introdução

Um Controlador Lógico Programável (CLP) é um dispositivo eletrônico, que controla máquinas e processos. Utiliza uma memória programável, para armazenar instruções e executar funções específicas, como o controle de energização/desenergização, temporização, contagem, sequenciamento, operações matemáticas e manipulação de dados.

O desenvolvimento dos CLPs começou em 1968, em resposta, a uma necessidade da Divisão Hidramática da General Motors (GM). Na época, a empresa passava dias ou semanas alterando sistemas de controle baseados em relés, sempre que mudava um modelo de carro ou introduzia modificações, na linha de montagem. Para reduzir o alto custo de instalação decorrente destas alterações, a especificação de controle da GM necessitava, de um sistema de estado sólido, com flexibilidade e que pudesse ser programado e mantido pelos engenheiros e técnicos na fábrica. Além disso, era preciso, que suportasse o ar poluído, a vibração, o ruído elétrico e os extremos de umidade e temperatura, encontrados na empresa. Instalou-se desta forma os primeiros CLPs, com aplicações automobilísticas, que foram desenvolvidos e implantados em 1969 (01).

O Controlador Lógico Programável surgiu dentro da General Motors, no intuito de solucionar as dificuldades encontradas, nos painéis de comando das linhas de montagem. Criou-se, desta forma, um dispositivo, que refletia as necessidades, de muitos usuários de circuitos à relés, não só da indústria automobilística, como de toda a indústria manufatureira.

Nasceu assim, um equipamento bastante versátil, que vem se aprimorado constantemente, por diversos fabricantes (GE Fanuc, Unitronics, Atos, Dexter e outros), diversificando cada vez mais os setores industriais e suas aplicações. Dentre as empresas fabricantes de controladores lógicos programáveis, selecionou-se, para estudo neste trabalho, o LOGO!Soft Comfort (02) desenvolvido pela SIEMENS.

O LOGO! é um CLP, criado para aplicações de automação, não só na área industrial, como também na construção civil, comércio e até mesmo residências. Este CLP é utilizado em situações, onde há necessidade de acionar ou desativar dispositivos automáticos, como lâmpadas, portas, válvulas, sistemas de refrigeração dentre outros. (03).

Atualmente o LOGO! vem sendo muito usado no planejamento de instalações, em construção civil, como iluminação de escadarias, controle e iluminação de portões, controle de venezianas e vidros (persianas motorizadas) e sistemas, que necessitam, de controle de tempo, luminosidade e ventilação.

Da mesma forma, as aplicações fabris são também diversas, como acionamento e desligamento de guilhotinas de papel, prensas de sucata, controle de portas, controle de bombas d'água, controle de barreira entre outras.

De acordo com a definição de NATALE (04), um Controlador Lógico Programável pode automatizar uma grande quantidade de informações, substituindo assim o homem com mais precisão, confiabilidade, custo e rapidez.

O LOGO!, assim como qualquer CLP, de outros fabricantes, possui um microprocessador, que realiza quatro funções básicas:

- Processamento de programas definidos;
- Varredura das entradas no processo;
- Programação das memórias externas;
- Comunicação entre o computador e o CLP.

Desta maneira, os programas de automação desejados, são descritos em diagramas lógicos, listas de instruções ou linguagem ladder, e a partir de então, transferidos, para a memória RAM do CLP. Quando o mesmo é posto em operação, o conteúdo da memória é executado seqüencialmente, junto com os dados do sistema, realizando as tarefas de entrada e saída. O terminal de programação é o meio de comunicação entre o microcomputador e a memória do CLP.

Os CLPs disponíveis no mercado brasileiro, tipicamente, utilizam na sua arquitetura microprocessadores e circuitos integrados de aplicações específicas (ASICs - Application Specific Integrated Circuits). Esses microprocessadores para executarem os seus programas de controle necessitam realizar ciclos de busca e execução da instrução. O ciclo de busca da instrução não está diretamente relacionado com o processo no qual o CLP está inserido, mas é condição determinante, para o microprocessador executar o programa, que está carregado na memória. Esta necessidade de busca da instrução, demanda tempo do microprocessador, o qual poderia estar sendo utilizado, na execução das tarefas pertinentes ao processo.

Os microprocessadores são componentes extremamente flexíveis, devido a sua programabilidade. A sua programação permite aplicação em diversos tipos de controles industriais. A execução de um algoritmo depende de um software armazenado em memória,

que será executado em uma arquitetura tipo Von Neumann, por exemplo, com ciclos de busca e execução das instruções.

Na arquitetura baseada em dispositivo lógico programável, por exemplo FPGA (Field Programmable Gate Array), um algoritmo é implementado por hardware, sem precisar de ciclos de busca e execução de instruções (05).

Este trabalho de pesquisa tem como objetivo avaliar a viabilidade de se substituir o microprocessador utilizado nos atuais CLPs, por dispositivos lógicos programáveis (PLDs). Os PLDs são dispositivos, que implementam circuitos e sistemas digitais, podendo, desta forma, implementar os mesmos sistemas de controle, que os CLPs implementam de modo seqüencial.

Estudou-se como as funções disponíveis no CLP LOGO!Soft podem ser implementadas, em linguagem VHDL, podendo, desta forma, configurar uma FPGA, contendo um hardware equivalente, ao software executado pela CPU do CLP.

A VHDL, difere das linguagens ditas convencionais, pois tem como principal característica, a execução concorrente, ou seja, vários sinais que podem ser tratados simultaneamente, melhorando a resposta do sistema.

Considerando a grande utilização de CLPs, em automação de processos, a proposta de desenvolver um software conversor de LOGO!Soft para VHDL é uma iniciativa, na tentativa de substituir a arquitetura convencional empregada, em CLP por uma FPGA. Esta nova abordagem tem como vantagem o seguinte:

Expansão: Gerar um código VHDL à partir da descrição da linguagem LOGO!Soft do CLP – LOGO! irá expandir a possibilidade de automatizar sistemas de controle, para um amplo círculo de desenvolvedores e projetistas, que desconhecem as linguagens de descrição de hardware, como a VHDL que é uma linguagem padrão na síntese de sistemas digitais. Desta forma, um programador de controladores lógicos programáveis pode facilmente migrar os projetos descritos em linguagem de CLP, para um código VHDL correspondente.

Flexibilidade: Obtendo-se o código fonte de um sistema de controle em sintaxe VHDL, a transferência do mesmo, para uma outra linguagem, poderá ser facilmente conseguida, bastando somente adequar a lógica das funções à sintaxe da linguagem de programação desejada, possibilitando assim, o surgimento de novas ferramentas para a automação.

Desempenho: Os processos convertidos em VHDL podem ser testados e simulados da forma mais simplificada possível. As funções descritas de forma simplificadas aumentam

conseqüentemente o desempenho do sistema, como um todo, pois o tamanho das descrições tem impacto direto, sobre o tempo de varredura do sistema por completo.

Substituição do Software por Hardware: O programa de automação, uma vez gravado na memória do CLP, fica em constante processo de execução (RUNNING). Diante disto, obter uma descrição VHDL a partir do LOGO! Soft torna-se interessante, porque substitui um software, por um hardware, ou seja, substitui o programa compilando na memória do CLP, ganhando com isso o paralelismo inerente ao hardware cujo controle pode ser executado também em tempo real.

Paralelismo: A linguagem de descrição de hardware VHDL trabalha com sistemas concorrentes, o que permite uma análise de todas as entradas do circuito de forma paralela, e por conseqüência o tempo de resposta torna-se muito rápido. Em contrapartida, os programas descritos, para os CLPs trabalham, com a análise seqüencial, das descrições, o que torna o programa mais lento e a análise de suas rotinas mais complexas, para que haja uma resposta imediata do sistema (06).

Apresenta-se na próxima seção, um resumo sobre trabalhos desenvolvidos por outros autores, relacionados com as propostas deste trabalho. No capítulo 2, apresentam-se as funções básicas e especiais disponibilizadas, pelo LOGO!Soft, assim como os códigos VHDL e as simulações realizadas no ambiente QUARTUS II da Altera.

O software conversor desenvolvido é apresentado no capítulo 3. Alguns exemplos de uso do LOGO^{2}VHDL também são apresentados.

No capítulo 4 descrevem-se alguns casos, que foram selecionados. Finalmente, no capítulo 5 apresenta-se a conclusão sobre este trabalho e trabalhos futuros, que servirão para avaliar o desempenho da ferramenta desenvolvida.

1.2. Estado da Arte

A evolução das metodologias de projeto de hardware, apoiadas em poderosas ferramentas de software, em especial os dispositivos reconfiguráveis como FPGAs (Field-Programable Gate Arrays) abriu um novo horizonte, entre os extremos da computação e o hardware dedicado (07).

Hoje, é possível desenvolver um projeto de sistema digital empregando-se novas metodologias, como uma linguagens de descrição de hardware (HDLs), ferramentas de síntese lógica e simulação (08). Uma área promissora, para a aplicação de FPGAs, que está se

desenvolvendo, é a implementação de máquinas computacionais dedicadas e reprogramáveis dinamicamente.

Os primeiros trabalhos de automação, realizados com projetos de hardware utilizando a linguagem VHDL, para implementação em FPGA, foram desenvolvidos com o propósito de atender as necessidades na área da robótica, no desenvolvimento de algoritmos para controle de robôs móveis, que geralmente necessitavam de muitos recursos computacionais para execução em tempo real.

No Departamento de Engenharia Elétrica da Universidade da Califórnia - UCLA, foi construído o protótipo, de um sistema de reconhecimento, com hardware reconfigurável, que obteve significante economia de hardware, pelo auto-ajuste do circuito, para cada modelo comparado.

Uma outra abordagem interessante de aplicação de computação reconfigurável à robótica é o sistema de computação, desenvolvido pela NASA, chamado de processador matemático de robôs (RMP). Este sistema contém um grande número de elementos de processamento, conectados, em várias combinações paralelas e seriais que são reconfiguráveis via software. O RMP é uma arquitetura de propósito especial projetada, para resolver problemas computacionais diversos, em controle de robô, simulação, geração de trajetória, análise de ambiente de trabalho, dente outros (09).

Outro trabalho, desenvolvido por LIMA & CARDOSO (10), implementa um controlador, para câmera digital utilizando FPGA. Neste projeto, um robô em movimento captura imagens através de uma câmera de vídeo digital instalada. Métodos como este, visam à sintonia automática de controladores com interesses a diversas aplicações, em particular para a robótica. No trabalho, o processamento das imagens capturadas pelo robô é realizado à medida, que o mesmo se movimenta. Esta movimentação do robô de um ambiente, com muita luz, para outro com pouca luminosidade, requer que alguns dos parâmetros da câmera sejam alterados, de forma que as imagens continuem sendo capturadas com qualidade suficiente. A implementação deste controlador utiliza FPGA, com programação, em linguagem VHDL de forma, que as avaliações dos parâmetros de luminosidade fiquem a cargo de um processador incluído na própria FPGA.

Verifica-se, com os trabalhos realizados, que a tecnologia da computação reconfigurável, utilizando FPGA, tornou-se objeto de estudo principalmente pela habilidade de se modificar o hardware da arquitetura do sistema em tempo real, para se adequar à aplicação.

Além da Robótica, outras pesquisas a partir de então foram realizadas apoiadas na linguagem de descrição de hardware, como por exemplo, o desenvolvimento de ambientes para automação de sistemas de controle. A utilização de ferramentas de software, como é o caso da *VHDL* na implementação em *FPGAs*, tem simplificado e acelerado o desenvolvimento de projetos e assim obtido um papel de grande interesse entre pesquisadores no processo de substituição de CLPs por dispositivos Lógicos Programáveis.

WELCH & CARLETA (11), desenvolveram uma arquitetura, com alto desempenho em FPGA, para o controle de processos industriais. Trata-se de uma arquitetura, que utiliza como padrão a linguagem Ladder, onde foram desenvolvidos vários dispositivos, para a automação industrial, padrões de conexão, dispositivos de bloco de função terminais, e outros componente característicos de CLP mais usuais.

MIYAZAWA et al (12), também desenvolveram um tipo de controlador, que é executado, utilizando um dispositivo lógico programável, do tipo FPGA. Neste trabalho, o autor cita a importância da varredura cíclica de processos realizados em computadores pessoais e desta forma, descreve dois métodos para implementar esta execução cíclica. Um método envolvido utiliza pulsos de relógio, em linguagem de descrição de hardware – VHDL para análise de circuitos integrados, que exigem alta velocidade. O outro método traduz uma bobina automática em linguagem Ladder de forma equivalente à lógica da linguagem VHDL. Assim, como resultado da pesquisa, os dois métodos foram comparados e apresentados os benefícios do controlador novo, executado por FPGA.

Um dos primeiros trabalhos com implementação de sistema de controle, para aplicações domésticas foi descrito em (13). Esta pesquisa visa à automação de residências, integrando Hardware e Software via utilização de dispositivos lógicos programáveis do tipo FPGA. No trabalho, eletrodomésticos da casa são interligados, por um protocolo de controle, todo desenvolvido em VHDL e implementado, em um dispositivo lógico programável. O sistema é acessível via Internet, sendo que os usuários (moradores) controlam e administram seu lar usando um navegador Web comum.

O trabalho desenvolvido por HAFNER (14) implementa um medidor de qualidade de energia elétrica usando computação reconfigurável por hardware através de blocos funcionais que são programados em linguagem VHDL. O medidor analisa a quantidade e qualidade de energia elétrica de forma automática utilizando FPGA. A utilização de lógica reconfigurável por hardware minimiza a base de dados gerado por um medidor, permitindo uma análise mais rápida do comportamento global de um sistema composto por um grande número de medidores. Isto é possível devido ao processamento em tempo real de diversas funções usadas

na análise da qualidade da energia elétrica. Este processamento deve-se ao paralelismo e tempo de execução reduzido utilizado em lógica VHDL. O trabalho visa fornecer uma alternativa econômica e tecnologicamente viável na análise de quantidade e qualidade de sistemas energéticos domésticos.

Percebe-se, que vem surgindo vários trabalhos automatizando sistemas de controle, utilizando como referência, dispositivos lógicos programáveis, como forma alternativa aos projetos comumente realizados em controladores lógicos programáveis.

Na pesquisa apresentada em (15) descreve-se como periféricos e controladores podem ser aplicados a automação de sistemas. Neste, é implementado um projeto contendo seis periféricos de microcontroladores, os quais são mapeados em circuitos programáveis (FPGAs), com a respectiva programação em VHDL. Os autores apresentam, também, a importância do paralelismo existente em projetos programados em linguagem de descrição de hardware, podendo ser observado em um trabalho desenvolvido onde uma CPU especial executa as funcionalidades de periféricos de microcontroladores e um sistema que integra estes microcontroladores em FPGAs.

COSTA (16) lança também, uma bibliografia sobre tecnologias utilizadas em projeto de circuitos digitais, com utilização de ferramentas de software, como *EDA* e *VHDL*, no aperfeiçoamento do hardware reconfigurável dos dispositivos lógicos programáveis. Neste, o autor, apresenta conceitos teóricos e práticos relativos às tecnologias de projetos de circuitos digitais utilizando FPGAs e cita:

> *"Como resultado da pesquisa, considerava-se a possibilidade de se desenvolver um módulo de compilação que recebesse uma entrada escrita numa linguagem fonte (Ladder) e codificasse uma saída numa linguagem destino (VHDL), correspondente ao controlador FPGA. Durante o estudo realizado não foi encontrado o software mencionado".*

Em seu livro, o autor comenta não encontrar um software conversor da linguagem Ladder para a linguagem VHDL e como forma alternativa, opta por escolher um ambiente integrado de desenvolvimento, no caso o Quartus II, que permite o desenvolvimento de uma biblioteca de macroinstruções, baseada em símbolos gráficos. Assim, a bibliografia mostra a possibilidade de automatizar processos utilizando-se o editor de símbolos do Quartus II da Altera, criando desta forma símbolos gráficos equivalentes, aos utilizados nas instruções da linguagem ladder.

Uma metodologia para a especificação de sistemas de sínteses digitais, baseada em VHDL é apresentada em (17). Neste trabalho é proposto um método de conversão de um sistema mecatrônico implementado em ladder, para uma linguagem de descrição de hardware e posteriormente gravado, em uma FPGA. Trata-se de uma metodologia que converte um programa em ladder para VHDL, utilizam-se como referência o CLP OMNI-CNC.

Percebe-se na pesquisa de YAMAZAKI uma semelhança com o presente trabalho, porém aquele autor desenvolveu metodologias que transcrevem as funções em ladder para sintaxe de descrição de hardware. A grande diferença é que neste trabalho foi desenvolvida uma ferramenta própria denominada LOGO^{2}VHDL, que descreve automaticamente os códigos a partir de um sistema de controle.

Apresenta-se no próximo capítulo, os modelos VHDL das funções disponíveis na linguagem LOGO!Soft do CLP LOGO!.

Capítulo 2

Descrição das Funções do LOGO!Soft

2.1. Introdução

A lógica de Boole e a linguagem de programação utilizadas na automação permitem aos usuários representar, um processo ou uma situação de controle, em diagramas lógicos, linguagem Ladder ou em lista de instruções. Essas três representações compreendem hoje as linguagens de automação padrão entre os diversos fabricantes de CLPs (18).

De acordo com NATALE (04), automatizar um sistema significa fazer uso de funções lógicas representadas, por portas lógicas, que podem ser implementadas, independente do nível de sua tecnologia, ou seja, relé, diodo, transistor, circuito integrado, etc..

Desta maneira, as funções Booleanas, propostas na álgebra de Boole, fazem parte do grupo de instruções básicas da linguagem de descrição do LOGO!Soft, assim como qualquer outra linguagem de Controladores Lógicos Programáveis, desde os mais simples aos mais complexos.

Como o principal objetivo deste trabalho é obter um modelo VHDL a partir de funções lógicas básicas e especiais disponibilizadas pelo CLP LOGO!Soft, descreve-se a seguir estas funções que foram implementadas em sintaxe VHDL e suas respectivas simulações no QUARTUS II da Altera, tendo como referência o ambiente de desenvolvimento LOGO!Soft Comfort.

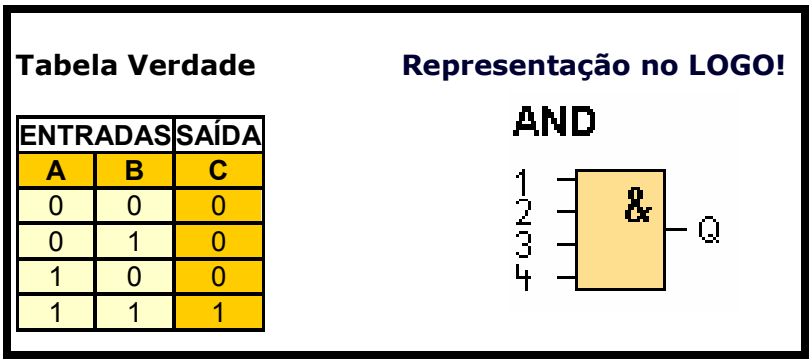
2.2. Listas de Funções Básicas

As funções básicas do LOGO!Soft correspondem aos elementos lógicos da álgebra booleana. Fazem parte, portanto, as funções lógicas: AND, NAND, OR, NOT, NOR e XOR (03).

2.2.1. Função And

A função AND combina dois ou mais sinais de entrada de modo, que somente haverá o nível lógico 1 na saída, se em todas as entradas houverem o nível lógico 1. Pode-se

comparar uma função AND a interruptores ligados em série, conforme apresentado na figura 2.1. Somente há condução quando todos os interruptores estiverem fechados.



ENTRADAS		SAÍDA
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Figura 2.1 - Tabela verdade e representação da função AND no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada, no ambiente QUARTUS II é apresentado na figura 2.2.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity porta_and4 IS
PORT(
        I1,I2,I3,I4 : IN std_logic;
        Q : OUT std_logic
        );
End porta_and4;
Architecture porta of porta_and4 IS
BEGIN
        Q  <= (I1 AND I2 AND I3 AND I4);
END porta;
```

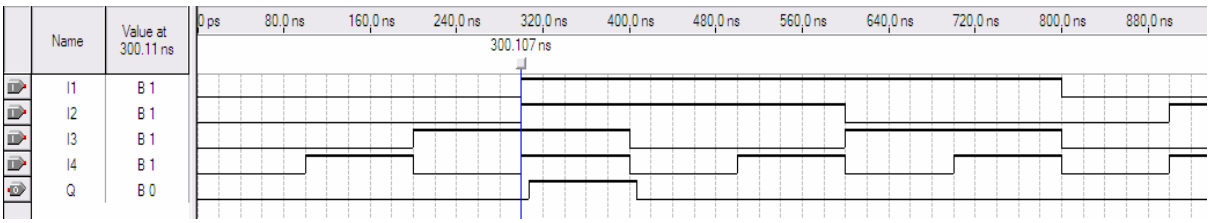


Figura 2.2 – Simulação do código VHDL referente à função AND

Percebe-se que a saída somente possui nível lógico 1 quando todas as entradas também possuírem nível lógico 1. Verifica-se também, que é gerado um pequeno sinal de atraso na saída do circuito, estes sinais de atrasos são inerentes à tecnologia utilizada, mas que no entanto, não interferem no real funcionamento da função.

2.2.2. Função Not

A função NOT funciona como um circuito em chaves. Esta função tem como objetivo complementar o sinal de entrada, ou seja, se na entrada possuir o nível lógico 1, a saída terá o nível lógico 0 e vice-versa, conforme apresentado na figura 2.3.

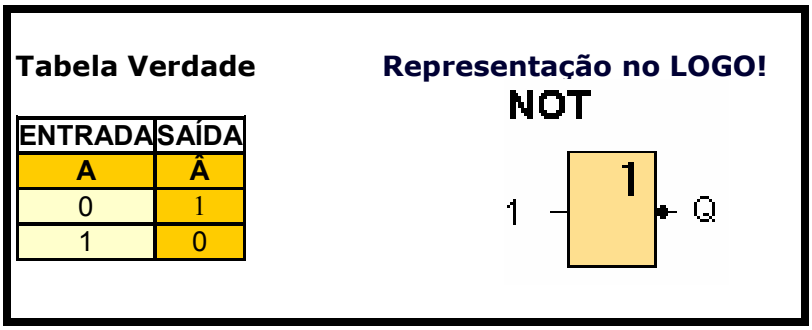


Figura 2.3 - Tabela verdade e representação da função NOT no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada, no ambiente QUARTUS II é apresentado na figura 2.4.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity porta_not IS
PORT(
        I1   : IN std_logic;
        Q  : OUT std_logic
        );
End porta_not;
Architecture porta of porta_not IS
BEGIN
        Q  <= not I1 ;
END porta;
```

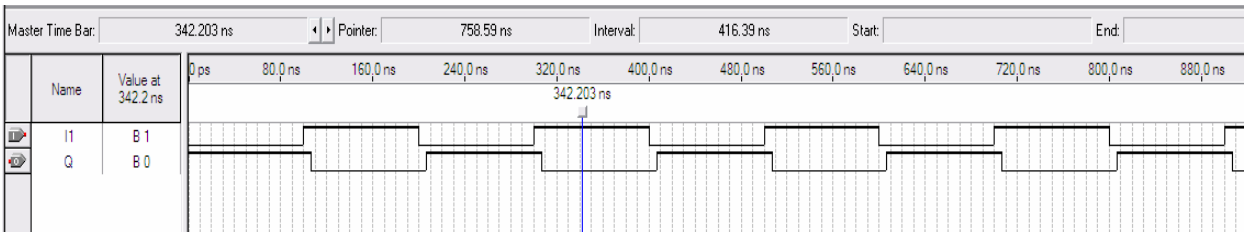


Figura 2.4 – Simulação do código VHDL referente à função NOT

Percebe-se que a saída somente possui nível lógico 1 quando a entrada possuir nível lógico 0 e vice-versa.

2.2.3. Função Nand

A função NAND é uma função AND seguida de um inversor (função NOT). Haverá sempre na saída NAND o inverso do que se tem na saída da função AND. A tabela verdade da função e sua representação no LOGO!Soft é mostrada na figura 2.5.

Tabela Verdade

ENTRADAS		SAÍDA
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Representação no LOGO!

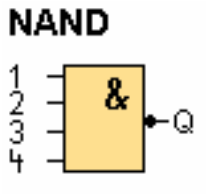


Figura 2.5 - Tabela verdade e representação da função NAND no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada, no ambiente QUARTUS II é apresentado na figura 2.6.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity porta_nand4 IS
PORT(
        I1,I2,I3,I4 : IN std_logic;
        Q : OUT std_logic
        );
End porta_nand4;
Architecture porta of porta_nand4 IS
BEGIN
        Q  <= not(I1 AND I2 AND I3 AND I4);
END porta;
```

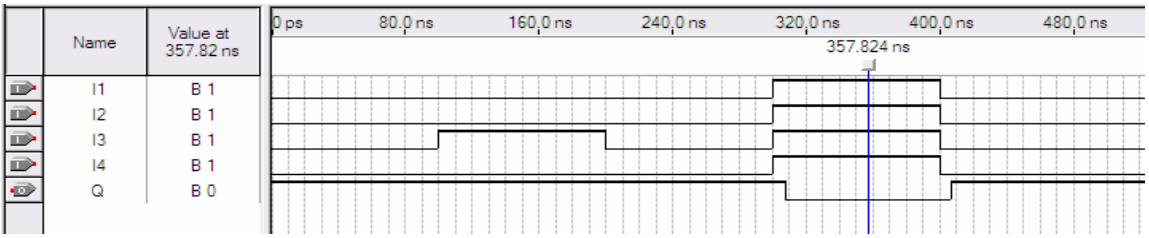
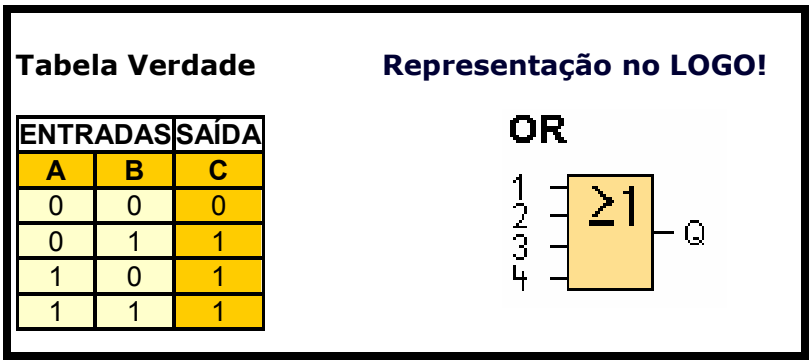


Figura 2.6 – Simulação do código VHDL referente à função NAND

Percebe-se que a saída somente possui nível lógico 0 quando todas as entradas possuírem nível lógico 1.

2.2.4. Função Or

A função OR possui o nível lógico 1 na saída quando em qualquer de suas entradas houver o nível lógico 1. Pode-se compará-la a dois interruptores em paralelo, conforme verifica-se na figura 2.7.



ENTRADAS		SAÍDA
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

Figura 2.7 - Tabela verdade e representação da função OR no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada no ambiente QUARTUS II é apresentado na figura 2.8.

```
 Library IEEE;
USE ieee.std_logic_1164.all;
Entity porta_or4 IS
PORT(
        I1,I2,I3,I4 : IN std_logic;
        Q : OUT std_logic
        );
End porta_or4;
Architecture porta of porta_or4 IS
BEGIN
        Q  <= (I1 OR I2 OR I3 OR I4);
END porta;
```

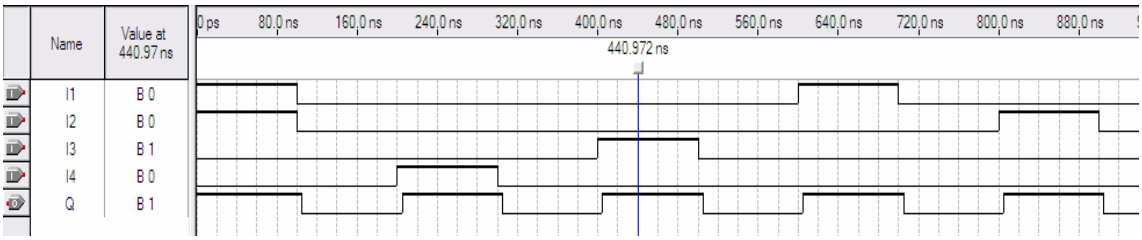


Figura 2.8 – Simulação do código VHDL referente à função OR

Observa-se que a saída possui nível lógico 1 quando pelo menos uma das entradas possuírem nível lógico 1.

2.2.5. Função Nor

A função NOR é uma função OR seguida da função NOT, o que significa dizer que a saída desta função é o complemento da saída de uma função OR. A tabela verdade da função e sua representação no LOGO!Soft pode ser verificada conforme ilustrado na figura 2.9.

Tabela Verdade

ENTRADAS		SAÍDA
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Representação no LOGO!

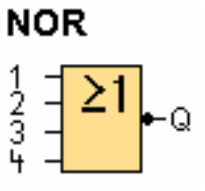


Figura 2.9 - Tabela verdade e representação da função NOR no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada no ambiente QUARTUS II é apresentado na figura 2.10.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity porta_nor4 IS
PORT(
        I1,I2,I3,I4 : IN std_logic;
        Q : OUT std_logic
        );
End porta_nor4;
Architecture porta of porta_nor4 IS
BEGIN
        Q  <= not(I1 OR I2 OR I3 OR I4);
END porta;
```

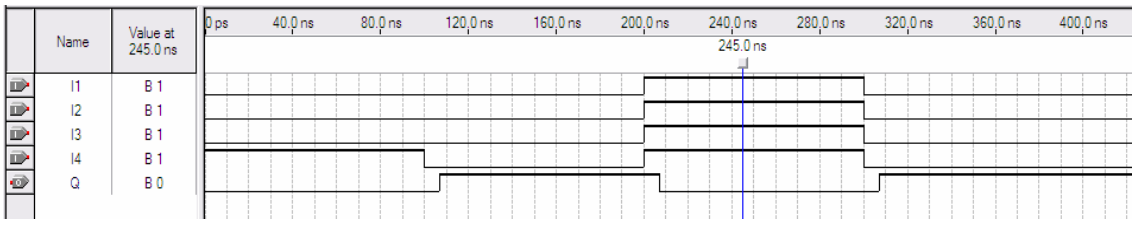


Figura 2.10 – Simulação do código VHDL referente à função NOR

Constata-se que a saída somente possui nível lógico 1 quando todas as entradas possuírem nível lógico 0.

2.2.6. Função Xor

A função XOR produz na saída o nível lógico 0 quando os bits na entrada forem iguais e produz na saída o nível lógico 1 quando os bits de entrada forem diferentes. A função XOR é dada pela expressão $A.\overline{B} + \overline{A}.B$ e é representada no LOGO!Soft conforme ilustrado na figura 2.11.

Tabela Verdade

ENTRADAS		SAÍDA
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Representação no LOGO!

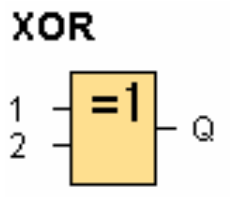


Figura 2.11 - Tabela verdade e representação da função XOR no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada no ambiente QUARTUS II é apresentado na figura 2.12.

```
 Library IEEE;
USE ieee.std_logic_1164.all;
Entity porta_xor2 IS
PORT(
        I1,I2 : IN std_logic;
        Q : OUT std_logic
        );
End porta_xor2;
Architecture porta of porta_xor2 IS
BEGIN
        Q  <= (I1 and not(I2)) or (not(I1) and I2);
END porta;
```

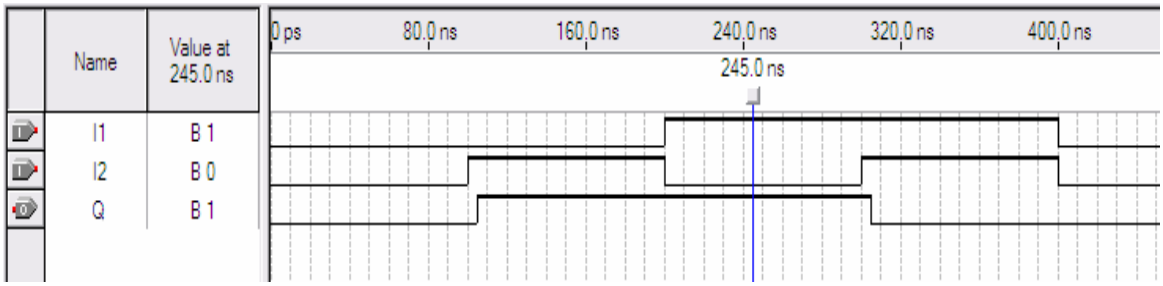


Figura 2.12 – Simulação do código VHDL referente à função XOR

Observa-se que a saída somente possui nível lógico 0 quando os bits de entrada possuem o mesmo nível lógico. A saída possui nível lógico 1 quando os bits na entrada possuírem diferentes níveis.

Além das funções lógicas básicas o LOGO!Soft disponibiliza um grupo de funções denominadas funções especiais. Estas são funções mais complexas e na sua maioria utilizam conceitos descritos na teoria de circuitos lógicos seqüenciais e permitem a parametrização de algumas variáveis.

2.3. Listas de Funções Especiais

As funções especiais diferenciam-se das funções básicas devido à necessidade de parametrização das entradas. Estas funções contêm parâmetros de tempo utilizando contadores e temporizadores possibilitando assim, adaptar um programa às necessidades individuais (03).

O LOGO!Soft disponibiliza as funções: retardamento de ligação, retardamento do desligamento, retardamento de ligação/desligamento, retardamento de ligação a memorizar, relé de passagem, comandado por flanco, gerador de ciclos assíncrono, gerador de sinal aleatório, interruptor de luz da escada, interruptor conforto, temporizador semanal, temporizador anual, contador crescente e decrescente, contador de horas de serviço, interruptor de valor limiar, analógico interruptor de valor limiar, analógico interruptor de valor limiar de diferença, comparador analógico, amplificador analógico, amplificador analógico, relé de auto-retenção, relé de impulso corrente, texto de aviso, softkey, registrador de deslocamento.

Nas variantes LOGO! 24, LOGO! 24o, LOGO! 12/24 RC e LOGO! 12/24RCo existe a possibilidade de parametrizar as entradas através de funções analógicas. Para essa programação o LOGO!Soft disponibiliza as funções: analógico interruptor de valor limiar, analógico interruptor de valor limiar de diferença, comparador analógico, monitorização do valor analógico e amplificador analógico. Salienta-se que trabalhou-se somente, com funções digitais.

Adotou-se que cada pulso de relógio corresponde a 1 segundo. Essa temporização é utilizada no tratamento referente à parametrização de cada uma dessas funções.

A seguir serão apresentadas as funções especiais, seguidos de código VHDL e simulação a partir da descrição VHDL. Ressalta-se que os programas foram devidamente

testados, compilados e simulados de acordo com o funcionamento correspondente oferecido pela linguagem LOGO!Soft.

2.3.1. Retardamento de Ligação

Nesta função, conforme representado na figura 2.13, a saída Q só será ligada após decorrido um tempo, que é passível de parametrização. Se o estado na entrada Trg mudar de 0 para 1, começa a decorrer o tempo parametrizado. Se o estado na entrada Trg permanecer nível lógico 1 durante o tempo parametrizado, a saída será colocada em nível lógico 1 decorrido o tempo. Se o estado na entrada Trg mudar novamente para 0 antes de esgotado o tempo parametrizado, a temporização será reinicializada. A saída será definida novamente em 0, se houver o estado 0 na entrada Trg.

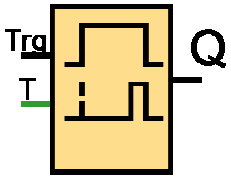
Representação no LOGO!Soft	Denominação da Função Especial
	Retardamento de Ligação

Figura 2.13 - Representação da função retardamento de ligação no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada no ambiente QUARTUS II é apresentado na figura 2.14.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity ret_lig IS
        Port(
                clk : in std_logic;
                Trg : in std_logic;
                saida : out std_logic);
        End ret_lig;
Architecture funcao of ret_lig IS
Begin
        process(Trg, clk)
                variable tempo : integer;
                Begin
                if (clk'EVENT and clk = '1') Then
                        if (Trg = '1') then
                          tempo := tempo + 1;
                                else
```

```
                        tempo := 0;
                end if;
                if (tempo >= 5) then
                        saida <= '1';
                        else
                        saida <= '0';
                end if;
        end if;
    end process;
End funcao;
```

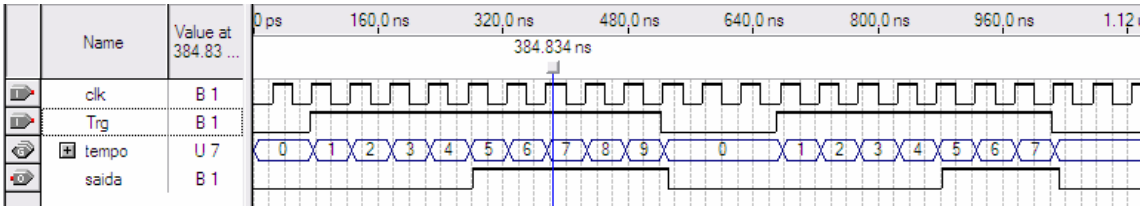


Figura 2.14 – Simulação do código VHDL referente à função retardamento de ligação

Constata-se que quando o Trg transita para o nível lógico 1, começa a decorrer o tempo parametrizado, que no caso é de 5 pulsos de relógio. Quando o contador atingir o valor parametrizado, a saída assume e permanece em nível lógico 1, enquanto o sinal em Trg permanecer em nível lógico 1. No término da contagem a saída possuía nível lógico 0, assume o nível lógico 1. Se o estado na entrada Trg mudar novamente para 0 o tempo parametrizado é reposto. A saída será definida novamente, em nível lógico 0, quando houver o estado 0 na entrada Trg.

2.3.2. Retardamento do Desligamento

Neste bloco, conforme representado na figura 2.15, se a entrada Trg mudar para o estado 1, a saída também mudará imediatamente para o estado 1. Se o estado em Trg mudar de 1 para 0, então inicia a contagem do tempo parametrizado. Quando o tempo parametrizado for alcançado, a saída será redefinida para o estado 0 (retardo do desligamento). Toda vez que a entrada Trg ligar e desligar, será reiniciado a contagem do tempo. Através da entrada R (reset) é possível colocar o tempo e a saída na posição inicial, antes mesmo, que a contagem do parâmetro tenha se esgotado.

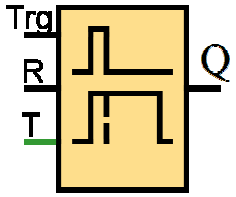
Representação no LOGO!Soft	Denominação da Função Especial
	Retardamento do desligamento

Figura 2.15 - Representação da função retardamento do desligamento no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada no ambiente QUARTUS II é apresentado na figura 2.16.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity ret_deslig IS
        Port(
                clk, reset, Trg : in std_logic;
                saida : out std_logic);
        End ret_deslig;
Architecture funcao of ret_deslig IS
Begin
                process(Trg, clk)
                        variable tempo: integer;
                        Begin
                        if (clk'EVENT and clk = '1') Then
                                if (reset = '1') then
                                        tempo := 0;
                                        saida <= '0';
                                else
                                         if (Trg = '1') then
                                                saida <= '1';
                                                tempo := 0;
                                                else
                                                tempo := tempo + 1;
                                                 if (tempo = 5) then
                                                        saida <= '0';
                                                 end if;
                                        end if;
                                end if;
                        end if;
                end process;
End funcao;
```

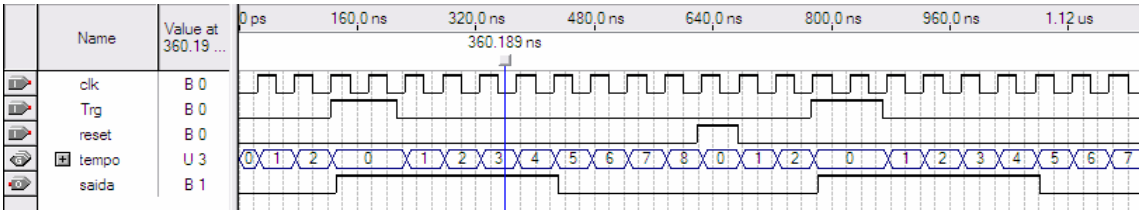


Figura 2.16 – Simulação do código VHDL referente à função retardamento do desligamento

Observa-se que quando a entrada Trg muda para o nível lógico 1, a saída também muda para o nível lógico 1. Quando o estado Trg muda seu nível lógico de 1 para 0, então inicia a contagem do tempo parametrizado. Quando o tempo parametrizado for alcançado, que neste caso é de 5 pulsos de relógio, a saída será redefinida para o estado 0. Toda vez que a entrada Trg ligar e desligar, será reiniciado a contagem do tempo.

2.3.3. Retardamento de Ligação / Desligamento

Nesta função, conforme representado na figura 2.17, a saída é ligada após um tempo parametrizado. Se o estado na entrada Trg mudar de 0 para 1, então começa a decorrer o tempo, se o estado na entrada Trg permanecer em 1, durante o tempo parametrizado, a saída será colocada em 1 (retardamento de ligação). Se o estado da entrada Trg mudar novamente para 0 antes de ter decorrido, o tempo parametrizado então o tempo é reposto. Se o estado na entrada Trg mudar novamente para 0, então começa a decorrer o tempo. Se o estado na entrada Trg permanecer em 0 durante o tempo parametrizado, a saída será colocada em 0 (retardamento de desligamento) após decorrer o tempo parametrizado. Se o estado na entrada Trg mudar para 1 antes de ter decorrido o tempo, então o tempo é reposto a 0.

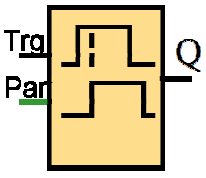
Representação no LOGO!Soft	Denominação da Função Especial
	**Retardamento de ligação / desligamento**

Figura 2.17 - Representação da função retardamento de lig. / deslig. no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada, no ambiente QUARTUS II é apresentado na figura 2.18.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity ret_lig_deslig IS
        Port(
                clk : in std_logic;
                Trg : in std_logic;
                saida : out std_logic);
        End ret_lig_deslig;
Architecture funcao of ret_lig_deslig IS
Begin
        process(Trg, clk)
        variable tempol, tempoh : integer;
        begin
                 if (clk'event and clk = '1') Then
                        if (Trg = '1') then
                           tempol := 0;
                           if (tempoh < 5) then
                              tempoh := tempoh + 1;
                           end if;
                         else
                           tempoh := 0;
                           if (tempol < 5) then
                                    tempol := tempol + 1;
                                end if;
                        end if;
                        if (tempoh = 5) then
                                saida <= '1';
                         end if;
                         if (tempol = 5) then
                                saida <= '0';
                         end if;
                end if;
        end process;
End funcao;
```

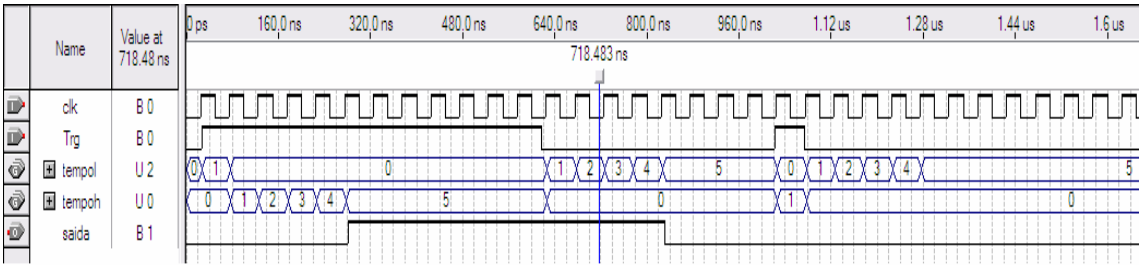


Figura 2.18 – Simulação do Código VHDL referente à função retardamento de lig. / deslig.

Percebe-se que a saída é ligada de acordo com o tempo parametrizado, que no caso é de 5 pulsos de relógio. Para esta função foram criados dois contadores: um para realizar o retardo de ligação (tempoh) em função do tempo parametrizado e outro para realizar o retardo do desligamento (tempol). Se o estado na entrada Trg mudar de 0 para 1, então começa a decorrer o tempoh, se o estado na entrada Trg permanecer em 1, durante o tempo parametrizado, a saída será colocada em 1. Se o estado na entrada Trg mudar novamente para 0, então começa a decorrer o contador tempol. Se o estado na entrada Trg permanecer em 0 durante o tempo parametrizado, a saída será colocada em 0 (retardamento de desligamento).

2.3.4. Retardamento de Ligação a ser Memorizado

Nesta função, conforme representado na figura 2.19, se o estado na entrada Trg mudar seu estado, começa então a correr o tempo parametrizado. Alcançando o tempo, a saída Q será colocada em 1. Uma nova mudança na entrada Trg não tem qualquer influência sobre o tempo que está sendo contado. A saída e o tempo do parâmetro só serão novamente recolocadas em 0, se na entrada R assumir o estado 1.

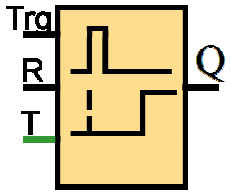
Representação no LOGO!Soft	Denominação da Função Especial
	**Retardamento de ligação a ser memorizado**

Figura 2.19 - Representação da função retardamento de ligação a ser memorizado no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada no ambiente QUARTUS II é apresentado na figura 2.20.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity ret_lig_mem IS
                Port(
                        reset : in std_logic;
                        clk   : in std_logic;
                        Trg   : in std_logic;
                        saida : out std_logic );
```

```
        End ret_lig_mem;
Architecture funcao of ret_lig_mem IS
Signal aux : std_logic := '0';
Begin
        process(Trg, clk)
                variable tempo : integer;
                Begin
                if (clk'EVENT and clk = '1') Then
                        if (reset = '1') then
                                saida <= '0';
                                tempo := 0;
                        end if;
                        if (trg = '1') then
                                aux <= '1';
                        end if;
                        if (aux = '1') then
                                tempo := tempo + 1;

                                if (tempo >= 10) then
                                        saida <= '1';
                                        aux <= '0';
                                else
                                        saida <= '0';
                                end if;
                        end if;
                end if;
end process;
End funcao;
```

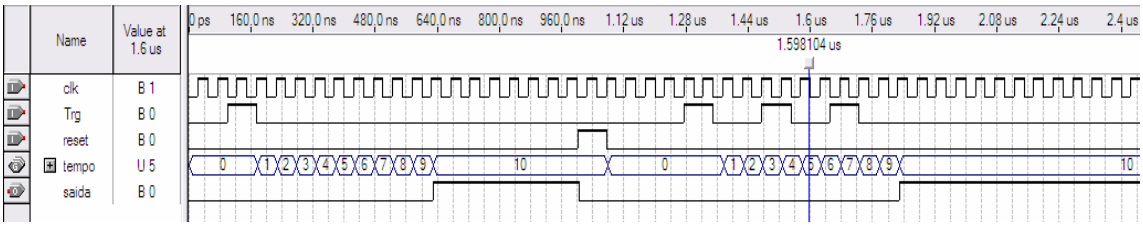


Figura 2.20 – Simulação do Código VHDL referente à função retardamento de ligação a ser memorizado

Observa-se que se a entrada Trg muda seu estado, automaticamente, o contador é incrementado até atingir o seu valor parametrizado, que para este caso é de 10 pulsos de relógio. Neste momento a saída que possuía nível lógico 0 passa a adquirir o nível lógico 1. Percebe-se também, na simulação, que uma outra comutação seguinte a entrada Trg, não influencia no tempo que está sendo contado. A entrada reset por sua vez tem como função

zerar o contador e colocar a saída em nível lógico 0, e desta forma o sistema aguarda uma nova comutação na entrada Trg.

2.3.5. Relé de Passagem

Nesta função, conforme representado na figura 2.21, se a entrada Trg assumir o estado 1, a saída Q passa para o estado 1. Simultaneamente inicia–se a contagem do tempo parametrizado. Quando o tempo alcançar o valor ajustado, a saída será redefinida, para o estado 0. Se, antes de esgotar o tempo especificado, a entrada Trg mudar de 1 para 0, a saída também mudará imediatamente de 1 para 0.

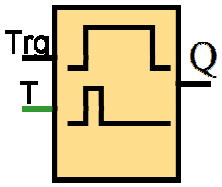
Representação no LOGO!Soft	Denominação da Função Especial
	Relé de passagem

Figura 2.21 - Representação da função relé de passagem no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada no ambiente QUARTUS II é apresentado na figura 2.22.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity rel_pas IS
        Port(
                clk : in std_logic;
                Trg : in std_logic;
                saida : out std_logic);
        End rel_pas;
Architecture funcao of rel_pas IS
Begin
        process(Trg, clk)
                variable tempo : integer;
                Begin
                if (clk'EVENT and clk = '1') Then
                        if (Trg = '1') then
                          tempo := tempo + 1;
                          saida <= '1';
                          else
```

```
                    tempo := 0;
                    saida <= '0';
                end if;
                if (tempo >= 5) then
                    saida <= '0';
                end if;
                end if;
        end process;
End funcao;
```

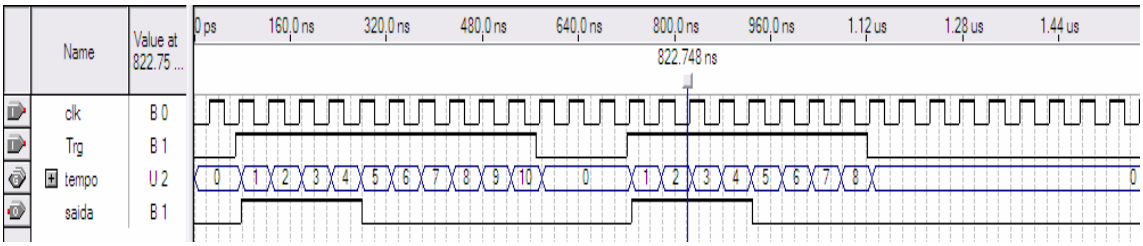


Figura 2.22 – Simulação do código VHDL referente à função relé de passagem

Observa-se que se a entrada Trg assumir o estado 1, a saída passa para o nível lógico 1. Simultaneamente inicia–se a contagem do tempo parametrizado, que no exemplo é de 5 pulsos de relógio. Quando o tempo for alcançado, a saída será redefinida para o estado 0.

2.3.6. Gerador de Ciclos Assíncrono

Nesta função, conforme representado na figura 2.23, existem dois parâmetros de entrada que medem um intervalo programado de forma cíclica, TH (Time High) e TL (Time Low). A entrada INV é um parâmetro, que permite a inversão da saída, se uma outra entrada denominada EN, que funciona como uma chave, estiver ativada.

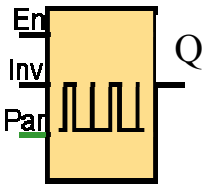
Representação no LOGO!Soft	Denominação da Função Especial
	Gerador de ciclos Assíncrono

Figura 2.23 - Representação da função gerador de ciclos assíncrono no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada no ambiente QUARTUS II é apresentado na figura 2.24.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity ger_cic_assin IS
Port(
        inv, en, clk : in std_logic;
        saida : out std_logic);
End ger_cic_assin;
Architecture funcao of ger_cic_assin IS
Begin
        process(clk)
        variable x: std_logic;
        variable th, tl: integer;
        Begin
        if (clk'EVENT and clk = '1') Then
                if (en = '1') and (inv = '0') then
                                x := '1';
                                if (th >= 10) then              --tempo th parametrizado
                                        x := '0';
                                        tl := tl +1;
                                        if (tl >= 5) then       -- tempo tl parametrizado
                                                x := '1';
                                                th := 0;
                                        end if;
                                end if;
                                if (x = '1') then
                                        saida <= '1';
                                        th := th + 1;
                                        tl := 0;
                                else
                                        saida <= '0';
                                end if;
                ELSE
                if (en = '1') and (inv = '1') then
                                x := '0';
                                if (th >= 10) then              --tempo th parametrizado
                                        x := '1';
                                        tl := tl +1;
                                        if (tl >= 5) then       -- tempo tl parametrizado
                                                x := '0';
                                                th := 0;
                                        end if;
                                end if;
                                if (x = '1') then
                                        saida <= '1';
                                else
                                        saida <= '0';
                                        th := th + 1;
                                        tl := 0;
                                end if;
                else
```

```
                        if (en = '0') then
                                th := 0;
                                tl := 0;
                        end if;
                end if;
        end if;
        end if;
        end process;
        End funcao;
```

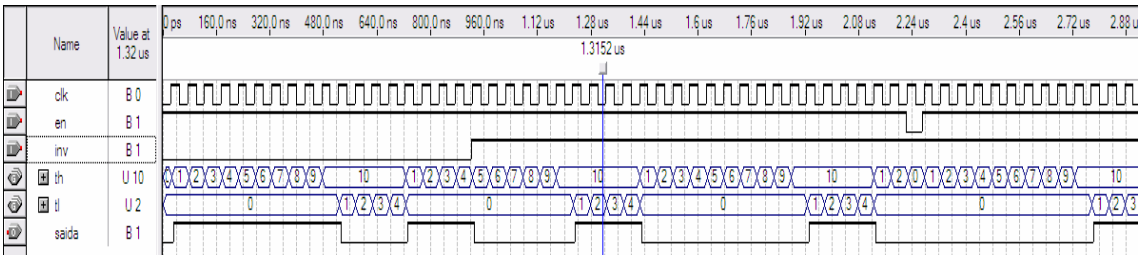


Figura 2.24 – Simulação do código VHDL referente à função gerador de ciclos assíncrono

Pode-se observar que por meio do parâmetro th (time high) e tl (time low) pode ser ajustada à duração do impulso. Definiu-se como exemplo th = 10 e tl = 5 pulsos de relógio. Percebe-se que uma vez a entrada En ativa (nível lógico 1) liga o gerador de impulso assíncrono, inicializando os contadores dos parâmetros th e tl, caso a entrada En esteja em nível lógico 0 reinicializa o contador, que está sendo incrementado no momento. Verifica que a saída do gerador depende do sinal INV na entrada, que por sua vez inverte o sinal de saída da função.

2.3.7. Interruptor de Luz da Escada

Nesta função, conforme representado na figura 2.25, se houver uma comutação na entrada Trg, a saída será colocada em nível lógico 1. Se o estado em Trg mudar de 1 para 0, então inicia–se o contador e a saída fica colocada em nível lógico 1. Se o tempo atingir o valor parametrizado, então a saída é reposta ao nível lógico 0. Antes de decorrer o tempo de retardamento do desligamento pode ser indicado um pré–aviso de desligamento, que repõe a saída a 0, durante o tempo de pré–aviso de desligamento. Se a entrada Trg for novamente ligada e desligada, enquanto o tempo do contador estiver sendo incrementado, então o tempo é reposto a zero (Retrigger).

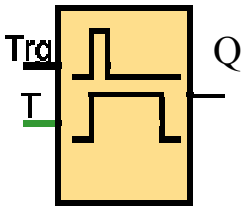
Representação no LOGO!Soft	Denominação da Função Especial
	**Interruptor de luz da escada**

Figura 2.25 - Representação da Interruptor de luz da escada no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada no ambiente QUARTUS II é apresentado na figura 2.26.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity int_luz_esc IS
Port(
        clk, trg : in std_logic;
        saida : out std_logic);
End int_luz_esc;
Architecture funcao of int_luz_esc IS
Begin
        process(Trg, clk)
        variable T: integer;
        variable x : std_logic;
        Begin
        if (clk'EVENT and clk = '1') Then
                if (Trg = '1') then
                                x := '1';
                                T := 0;
                                else
                                T := T + 1;
                                if (T >= 5) then        --pre aviso de desligamento
                                        if (T <= 7) then    --tempo do desligamento
                                                x := '0';
                                                else
                                                        x := '1';
                                        if (T >= 10) then         --tempo de parametrizacao
                                                x := '0';
                                end if;
                        end if;
                end if;
            end if;
                if (x = '0') then
                                saida <= '0';
                        else
                                saida <= '1';
```

```
                end if;

            end if;
end process;
End funcao;
```

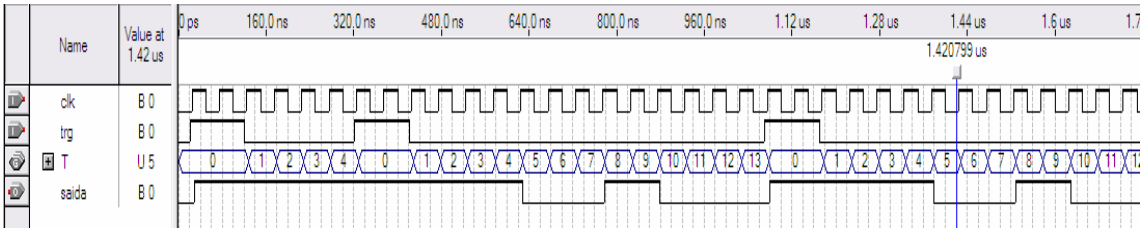


Figura 2.26 – Simulação do código VHDL referente à função interruptor de luz da escada

Pode-se observar que quando há uma mudança de estado na entrada Trg, a saída é colocada em nível lógico 1, e neste momento o contador é incrementado. Percebe-se que quando o tempo do contador atinge o valor parametrizado, que para este exemplo é de 10 pulsos de relógio a saída que possuía nível lógico 1 passa a adquirir o nível lógico 0. No entanto, antes que este procedimento seja realizado um pré-aviso de desligamento pode ser acionado. Definiu-se o intervalo do 5º ao 7º pulso de relógio, que posiciona a saída em nível lógico 0 durante este período. Verifica-se também, que quando a entrada Trg é novamente comutada enquanto o contador está sendo incrementado, neste momento, o mesmo é reposto ao valor zero.

2.3.8. Interruptor Conforto

Nesta função, conforme representado na figura 2.27, se na entrada Trg o nível lógico 0 mudar para o nível lógico 1, a saída será colocada em nível lógico 1. Se a saída possuir nível lógico 0 e a entrada Trg mudar do nível lógico 1 para o nível lógico 0 antes do tempo TL, então é ativada a função de luz permanente e a saída liga para permanente. Se o estado na entrada Trg mudar, antes do decorrer do tempo parametrizado, então é iniciado o tempo de retardamento do desligamento T. Se o tempo decorrido atingir o valor parametrizado, então a saída é reposta ao nível 0. Antes do decorrer do tempo de retardo do desligamento pode ser indicado um pré–aviso de desligamento, que repõe a saída para o nível lógico 0 durante o tempo de pré–aviso de desligamento. Uma nova ligação na entrada Trg repõe o tempo e o

mesmo procedimento é realizado. Através da entrada reset o tempo T é redefinido e repõe a saída a zero.

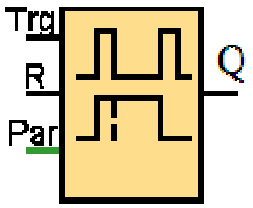
Representação no LOGO!Soft	Denominação da Função Especial
	**Interruptor Conforto**

Figura 2.27 - Representação da função interruptor conforto no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada no ambiente QUARTUS II é apresentado na figura 2.28.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity int_con1 IS
Port(
        clk, reset, trg : in std_logic;
        saida : out std_logic);
End int_con1;
Architecture funcao of int_con1 IS
Begin
        process(Trg, clk)
        variable T, TL: integer;
        variable x : std_logic;
        Begin
        if (clk'EVENT and clk = '1') Then
                if (reset = '1') then
                        T := 0;
                        x := '0';
                else
                        if (Trg = '1') then
                                TL := TL + 1;
                                x := '1';
                                T := 0;
                                else
                                T := T + 1;
                                if (TL <= 4) then              -- tempo para ativar a luz permanente
                                   if (T >= 5) and (T <= 7) then   -- pré-aviso de desligamento
                                           x := '0';                       -- e tempo do desligamento
                                           else
                                             x := '1';
                                     if (T >= 10) then            -- tempo de parametrizacao
                                          x := '0';
```

```
                                    TL := 0;
                                end if;
                                    end if;
                            end if;
                    if (x = '0') then
                            saida <= '0';
                        else
                            saida <= '1';
                    end if;
            end if;
        end if;
    end if;
end process;
End funcao;
```

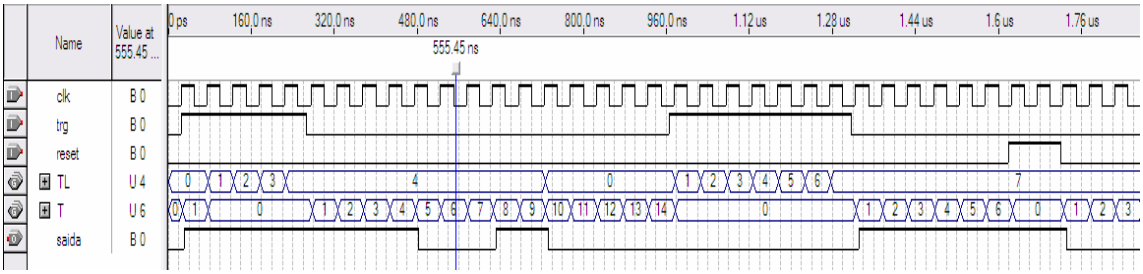


Figura 2.28 – Simulação do código VHDL referente à função interruptor conforto

Na simulação pode-se observar que quando a entrada Trg muda do nível lógico 0 para o nível lógico 1, a saída passa a adquirir o nível lógico 1 e o contador TL é incrementado. Este contador tem como função predefinir o tempo de duração máximo, para ativar a função de luz permanente. Neste mesmo instante se a entrada Trg mudar novamente de estado para o nível lógico 0 antes do valor parametrizado em TL, que para este caso é de 4 pulsos de relógio então é ativada a função de luz permanente, ou seja, a saída liga para permanente (nível lógico 1) durante o período T parametrizado, que para este caso é de 10 pulsos de relógio. Quando o contador T atinge o tempo pré-definido, a saída que possuía nível lógico 1 passa a adquirir o nível lógico 0. Verifica-se também, que antes do decorrer do tempo parametrizado, um indicativo de pré–aviso de desligamento, pode ser realizado, que repõe a saída a nível lógico 0 durante este pré–aviso de desligamento, intervalo pré-definido compreendido entre o 5º ao 7º pulso do relógio. Uma nova ligação na entrada Trg repõe o tempo e realiza o processo novamente. Percebe-se que quando a entrada Trg permanece em nível lógico 1 em tempo maior do que o valor parametrizado no contador TL a saída permanece em nível lógico 1. Um último caso pode ser observado, onde a entrada reset coloca a saída, em nível lógico 0 e reinicializa o contador T.

2.3.9. Contador Crescente / Decrescente

Nesta função, conforme representado na figura 2.29, se o valor do contador for menor do que o valor do tempo parametrizado, então a saída será 0. Se o contador for igual ou maior do que o tempo parametrizado, então a saída será 1. O sentido da contagem pode ser alterado através da entrada Dir. Se Dir = 0 o contador será crescente, se Dir = 1 o contador será decrescente. Se a entrada R (Reset) for igual a 1, a saída recebe 0 lógico e o contador é zerado e permanece sem contar até que R seja igual a 0.

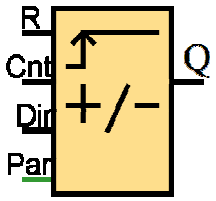
Representação no LOGO!Soft	Denominação da Função Especial
	Contador crescente / decrescente

Figura 2.29 - Representação da função contador crescente / decrescente no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada no ambiente QUARTUS II é apresentado na figura 2.30.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity cont_up_down2 IS
Port(
        reset, dir, clk : in std_logic;
        Q : out std_logic);
End cont_up_down2;
        Architecture Contador of cont_up_down2 IS
        Begin
                process(clk)
                variable cont, direcao : integer;
                Begin
                        if (dir = '0') then
                                direcao := 1;
                                else
                                direcao := -1;
                        end if;
                        If ((clk'event) and (clk = '1')) then
                                if (reset = '1') then
                                        cont := 0;
                                        else
```

```
                        cont := cont + direcao;
                end if;
                if (cont >= 5) then
                        Q <= '1';
                         else
                        Q <= '0';
                end if;
        end if;
    End process;
End Contador;
```

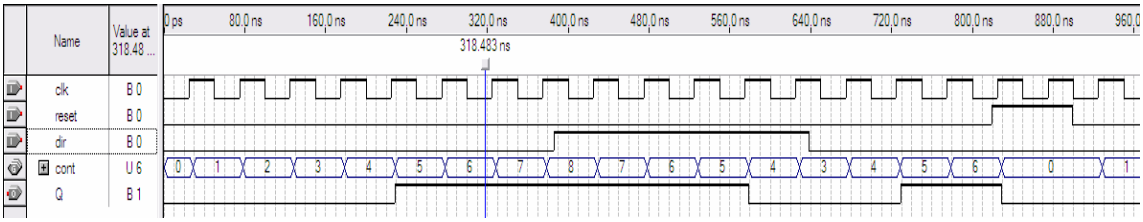


Figura 2.30 – Simulação do código VHDL referente à função contador crescente / decrescente

Constata-se, que enquanto a contagem for menor que o tempo parametrizado (5 pulsos de relógio), então a saída Q possui nível lógico 0, quando o contador for igual ou maior do que o tempo parametrizado, então a saída possuirá nível lógico 1. Percebe-se também, que o sentido da contagem pode ser alterado, através da entrada Dir. Se Dir = 0 o contador será crescente, se Dir = 1 o contador será decrescente. Em último caso, quando o Reset é ativo (nível lógico 1), a saída recebe o nível lógico 0 e o contador é zerado.

2.3.10. Contador de Horas de Serviço

Nesta função, conforme representado na figura 2.31, o contador das horas de serviço monitora a entrada En. Se a entrada En possuir nível lógico 1, um contador Mn que é inicialmente parametrizado é inicializado. Se o tempo do contador Mn atingir o valor parametrizado, a saída que possuía o nível lógico 0 passa a adquirir o nível lógico 1. Neste bloco de função existe um outro contador denominado Ot que inicialmente recebe o valor zero, sendo incrementado, enquanto a entrada En possuir o nível lógico 1. A entrada reset repõe a saída para o nível lógico 0 e reinicializa o contador Mn. A entrada Ral por sua vez repõe os dois contadores Ot e Mn e também a saída para o nível lógico 0.

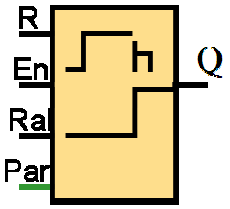
Representação no LOGO!Soft	Denominação da Função Especial
	Contador de horas de serviço

Figura 2.31 - Representação da função contador de horas de serviço no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada no ambiente QUARTUS II é apresentado na figura 32.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity cont_horas_servico1 IS
        Port(
                clk, reset, ral, En : in std_logic;
                Q : out std_logic
                );
End cont_horas_servico1;
Architecture Contador of cont_horas_servico1 IS
Begin
process(clk)
variable Mn, Ot: integer;
variable x : std_logic;
Begin
        If ((clk'event) and (clk = '1')) then
                if (ral = '1') then
                        x := '1';
                        Ot := 0;
                        Mn := 0;
                else
                if (reset = '1') then
                  x := '1';
                  Mn := 0;
                end if;
                        if (En = '1') then
                                Ot := Ot + 1;
                                Mn := Mn + 1;
                                if (Ot <= 99999) then --valor limite p/o cont. de horas de serviço
                                        if (Mn >= 5) then   -- valor parametrizado
                                                x := '0';
                                        else
                                          x := '1';
                                        end if;
```

```
                        end if;
                end if;
        end if;
        if (x = '1') then
                Q <= '0';
                else
                Q <= '1';
        end if;
    end if;
End process;
End Contador;
```

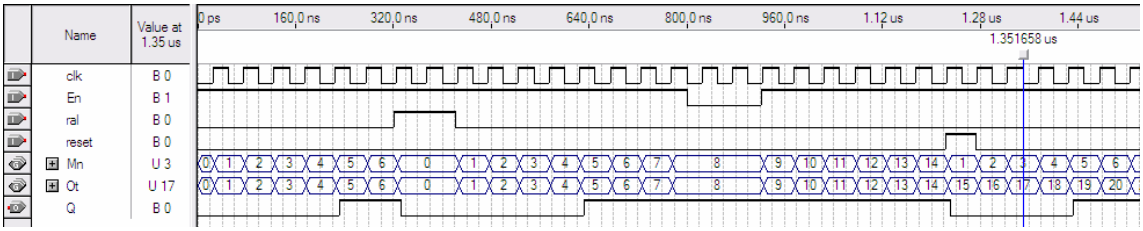


Figura 2.32 – Simulação do código VHDL referente à função contador de horas de serviço

Através da simulação pode-se verificar, que quando a entrada En possuir nível lógico 1, o contador Mn é inicializado. Percebe-se que quando o tempo do contador Mn atingir o valor parametrizado, que neste caso é de 5 pulsos do relógio, a saída que possuía o nível lógico 0 passa a adquirir o nível lógico 1. Verifica-se ainda, que existe um outro contador denominado Ot que inicialmente recebe o valor zero e é incrementado enquanto a entrada En possuir o nível lógico 1. Esta entrada tem como função realizar a contagem do tempo de serviço total decorrido no sistema. Observa-se que a entrada Ral repõe os dois contadores Ot e Mn e também a saída para o nível lógico 0. Finalmente a entrada reset repõe a saída para o nível lógico 0 e reinicializa somente o contador Mn. Um novo processo se inicia caso a entrada En continue em nível lógico 1.

2.3.11. Relé de Auto-Retenção

Nesta função, conforme representado na figura 2.33, a saída será definida em nível lógico 1, quando a entrada S (Set) também possuir nível lógico 1. Quando a outra entrada R (Reset) possuir nível lógico 1, a saída será reposta em nível lógico 0. Se S e R possuírem ao mesmo tempo o nível lógico 1, dá–se a prioridade para o R, ou seja a saída será 0.

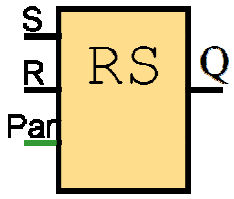
Representação no LOGO!Soft	Denominação da Função Especial
	Relé de auto-retenção

Figura 2.33 - Representação da função relé de auto-retenção no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada no ambiente QUARTUS II é apresentado na figura 2.34.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity re_au_ret IS
Port(
        R, S : in std_logic;
        Q : out std_logic
        );
End re_au_ret;
Architecture funcao of re_au_ret IS
Begin
process (S, R)
Begin
        if (S = '1') AND (R = '0') then
                Q <= '1';
                elsif (S = '0') AND (R = '1') then
                        Q <= '0';
                else
                        Q <= '0';
        end if;
end process;
end funcao;
```

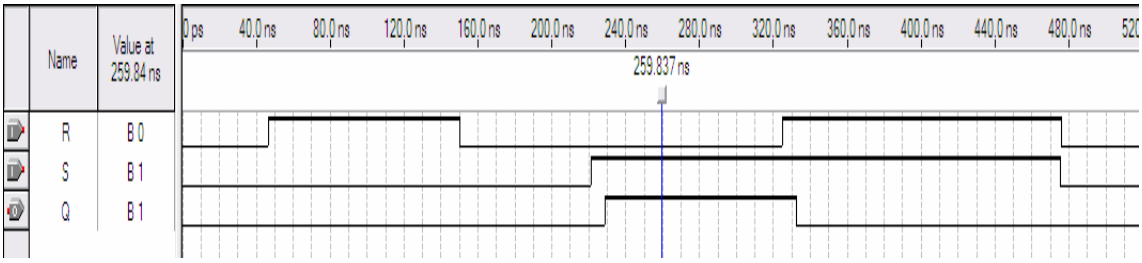


Figura 2.34 – Simulação do código VHDL referente à função relé de auto-retenção

Constata-se que se a entrada S possuir o nível lógico 1 a saída será definida em nível lógico 1. Se a entrada R possuir nível lógico 1, a saída será reposta a 0. Quando as duas entradas, S e R possuírem ao mesmo tempo o nível lógico 1, a saída possuirá o nível lógico 0. No entanto, se em último caso as duas entradas S e R possuírem o nível lógico 0 na entrada, a saída também possuirá o nível lógico 0.

2.3.12. Relé de Impulso Corrente

No relé de impulso corrente, conforme representado na figura 2.35, se o estado na entrada Trg mudar de 0 para 1 e as entradas S e R forem iguais a 0, a saída muda o seu estado, ou seja, a saída é ligada ou desligada. A entrada Trg não influencia esta função quando S = 1 ou R = 1. Através da entrada S, a saída é colocada em 1. Através da entrada R a saída é colocada a 0. Se tanto R quanto S possuírem níveis lógicos equivalentes a 1, este bloco de função permite, que o mesmo seja parametrizado definindo-se prioridades, a entrada R tem prioridade em relação à entrada S, colocando a saída em nível lógico 0 ou a entrada S tem prioridade em relação à entrada R, colocando a saída em nível lógico 1.

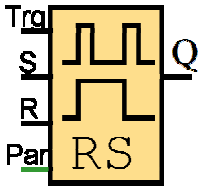
Representação no LOGO!Soft	Denominação da Função Especial
	Relé de impulso corrente

Figura 2.35 - Representação da função relé de impulso de corrente no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada no ambiente QUARTUS II é apresentado na figura 2.36.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity re_im_cor IS
        PORT(
                S, R, Trg : in std_logic;
                par : in std_logic;
                Q  : out std_logic);
End re_im_cor;
Architecture funcao of re_im_cor IS
```

```
BEGIN
process (Trg, S, R)
variable vs, vr : std_logic;
variable x : std_logic := '0';
 BEGIN
        if (R = '1') and (S = '1') then
          if (par = '0') then
                        vr := '1';
                        vs := '0';
         end if;
          if (par = '1') then
                        vr := '0';
                        vs := '1';
          end if;
        end if;
        if (R = '1') and (S = '0') then
                vr := '1';
                vs := '0';
        end if;
        if (R = '0') and (S = '1') then
                vr := '0';
                vs := '1';
        end if;

        if (R = '0') and (S = '0') then
                vr := '0';
                vs := '0';
        end if;

        if (vr = '1') then
                x := '0';
                elsif (vs = '1') then
                        x := '1';
                        elsif (Trg'event and Trg = '1') then
                                x := not (x);
        end if;
        Q <= x;
END process;
END funcao;
```

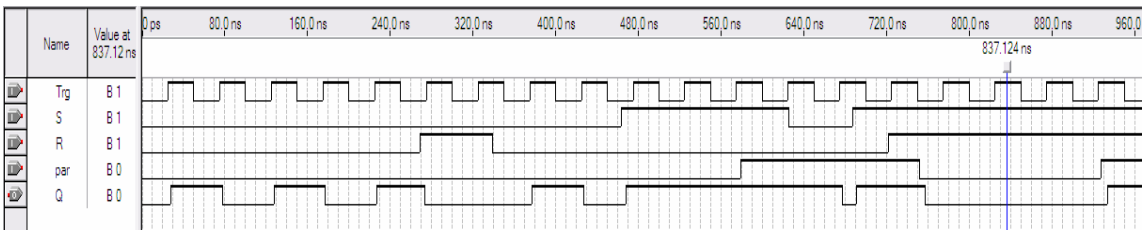


Figura 2.36 – Simulação do código VHDL referente à função relé de impulso corrente

Na simulação apresentada percebe-se, que a cada mudança na entrada Trg e quando ambas as entradas S e R possuem níveis lógicos igual a 0, a saída alterna seus impulsos entre níveis lógicos 0 e 1. A entrada S (set) quando possui nível lógico 1 coloca a saída em nível lógico 1, enquanto que a entrada R (reset) repõe a saída em nível lógico 0. Nota-se que quando ambas as entradas S e R possuem níveis lógicos equivalentes a 1, este bloco de função analisa a entrada par, cuja função é a de definir qual será a prioridade no momento. Se a entrada par neste instante possuir nível lógico 0, a entrada R tem prioridade em relação à entrada S, colocando a saída em nível lógico 0, caso a entrada par neste instante possuir o nível lógico 1, a entrada S tem prioridade em relação à entrada R, colocando a saída em nível lógico 1.

2.3.13. Softkey

Esta função, conforme representado na figura 2.37, funciona de forma similar a um botão ou interruptor mecânico. Se a entrada En (Enable) for colocada em 1 e o parâmetro "Switch" estiver ligado para a posição "On" (1 lógico), a saída liga–se. A saída é colocada em nível lógico 0 se o estado na entrada En mudar de 1 para 0 ou se o parâmetro "Switch" tiver sido comutado para a posição "Off", ou seja se estiver em nível lógico 0.

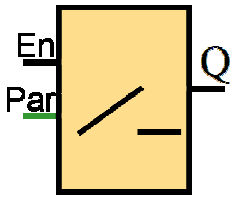
Representação no LOGO!Soft	Denominação da Função Especial
	Softkey

Figura 2.37 - Representação da função softkey no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada no ambiente QUARTUS II é apresentado na figura 2.38.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity softkey IS
Port(
        En, Switch : in std_logic;
        Q : out std_logic
        );
```

```
End softkey;
Architecture funcao of softkey IS
        BEGIN
        process (En, Switch)
        BEGIN
        if (En = '1' and Switch = '1') then
                        Q <= '1';
                elsif (En = '0' or Switch = '1') then
                        Q <= '0';
                else
                Q <= '0';
        end if;
End process;
End funcao;
```

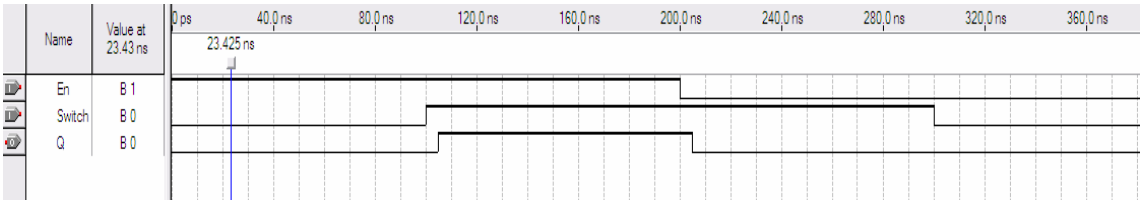


Figura 2.38 – Simulação do código VHDL referente à função softkey

Observa-se que se a entrada En for colocada em 1 e o parâmetro Switch' estiver ligado (nível lógico 1), a saída é ligada (possuirá nível lógico 1). Percebe-se que a saída é colocada em nível lógico 0 se o estado na entrada En mudar de 1 para 0 ou se o parâmetro 'Switch' tiver sido comutado para a posição 'Off' (nível lógico 0).

2.3.14. Registrador de Deslocamento

Esta função, conforme representado na figura 2.39, na mudança do nível lógico de 0 para 1 na entrada Trg, lê a função do valor da entrada In. Em função do sentido de deslocamento este valor é aceito no bit do registrador de deslocamento S1 ou S8. O registrador possui dois tipos de deslocamento: Deslocamento ascendente: S1 assume o valor da entrada In; o valor anterior de S1 é deslocado para S2; o valor anterior de S2 é deslocado para S3; etc. Deslocamento descendente: S8 assume o valor da entrada In; o valor anterior de S8 é deslocado para S7; o valor anterior de S7 é deslocado para S6 e assim por diante. A saída indica o valor do bit do registrador de deslocamento parametrizado.

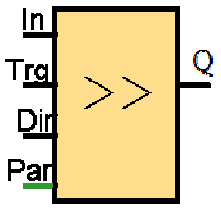
Representação no LOGO!Soft	Denominação da Função Especial
	**Registrador de deslocamento**

Figura 2.39 - Representação da função registrador de deslocamento no LOGO!Soft

O código VHDL implementado para esta função é apresentado a seguir e o resultado da simulação executada no ambiente QUARTUS II é apresentado na figura 2.40.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity reg_desloc IS
        Port(
                Trg : in std_logic;
                dir: in std_logic;
                sel: in integer range 1 to 8;
                inn : in std_logic;
                Q : out std_logic);
End reg_desloc;
Architecture Contador of reg_desloc IS
Signal S1, S2, S3, S4, S5, S6, S7, S8 : std_logic := '0';
Begin
        process(Trg, inn, dir)
        variable VQ1, VQ2, VQ3, VQ4, VQ5, VQ6, VQ7, VQ8 : bit;
                Begin
                case dir is
                        when '0' =>
                        VQ8 := inn;
                        VQ7 := S8;
                        VQ6 := S7;
                        VQ5 := S6;
                        VQ4 := S5;
                        VQ3 := S4;
                        VQ2 := S3;
                        VQ1 := S2;
                        when '1' =>
                                VQ8 := S7;
                                VQ7 := S6;
                                VQ6 := S5;
                                VQ5 := S4;
                                VQ4 := S3;
                                VQ3 := S2;
```

```
                    VQ2 := S1;
                    VQ1 := inn;

            end case;
            if (Trg'event and Trg = '1') then
            S1 <= VQ1;
                    S2 <= VQ2;
                    S3 <= VQ3;
                    S4 <= VQ4;
                    S5 <= VQ5;
                    S6 <= VQ6;
                    S7 <= VQ7;
                    S8 <= VQ8;
            end if;
    End process;
            With sel select
                Q <= S1 when 1,
                S2 when 2,
                S3 when 3,
                S4 when 4,
                S5 when 5,
                S6 when 6,
                S7 when 7,
                S8 when 8;
End Contador;
```

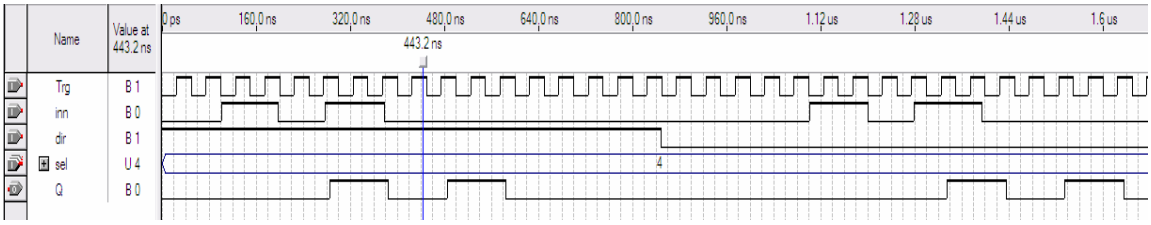


Figura 2.40 – Simulação do código VHDL referente à função registrador de deslocamento

Observa-se que quando a entrada dir possui nível lógico 1 o deslocamento é ascendente e a saída possui nível lógico 1 quando atinge o valor do bit do registrador de deslocamento parametrizado (sel = 4). Quando a entrada dir possuir nível lógico 0 o deslocamento é descendente e a saída possui nível lógico 1, quando atinge o valor do bit do registrador de deslocamento parametrizado (sel = 4). Uma melhor visualização deste funcionamento pode ser observada na figura 2.41.

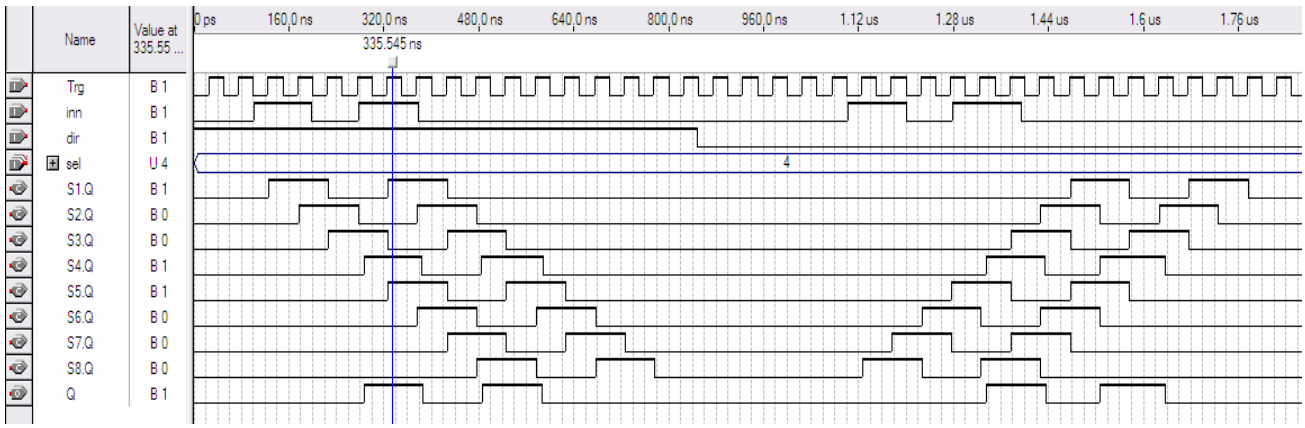


Figura 2.41 – Simulação do código VHDL referente à função registrador de deslocamento

Este capítulo apresentou-se como as funções básicas e as funções especiais disponíveis no LOGO!Soft foram modeladas através da linguagem VHDL. Apresenta-se no próximo capítulo o programa tradutor de LOGO!Soft para VHDL, denominado LOGO^{2}VHDL.

Capítulo 3

Tradutor LOGO²VHDL

3.1. Introdução

O *software* desenvolvido em ambiente Delphi foi denominado de "LOGO²VHDL". Esta ferramenta possibilita traduzir sistemas de controle utilizando, como referência expressões booleanas básicas e especiais oferecidas pelo LOGO!Soft, em modelos descritos na linguagem VHDL.

No desenvolvimento do LOGO²VHDL preocupou-se em cumprir basicamente quatro etapas: descrever em VHDL as funções oferecidas no LOGO!Soft; implementar no Delphi um ambiente com as funções programadas; inter-relacionar essas funções de acordo com as exigências e necessidades do usuário, descrever um sistema de controle em sintaxe VHDL completo com todas as declarações e bibliotecas necessárias.

Na primeira etapa foi necessário estudar a linguagem de descrição VHDL, as funções de automação do LOGO!Soft, e assim descrever de forma textual em linguagem de descrição de hardware cada uma das funções de forma independente.

A segunda etapa teve como objetivo realizar uma interface gráfica, que pudesse realizar modelos, utilizando as funções básicas e especiais estudadas no software LOGO!Soft.

Na terceira etapa realizou-se o relacionamento entre as funções, tento em vista, que para se automatizar um sistema de controle faz-se necessárias várias funções de parametrização, contadores e temporizadores. Desta forma, houve a necessidade de se criar a possibilidade de relacionar uma função a qualquer outra, de acordo com as especificidades do sistema de controle a ser desenvolvido e conseqüentemente realizar testes a fim de verificar possíveis erros.

A quarta etapa refere-se ao último passo executado dentro do software conversor. Ele tem como função enviar para a caixa de texto do formulário principal, o código produzido para uma sintaxe formatada, em uma descrição VHDL, com todas as suas declarações realizadas, bibliotecas e demais estruturas necessárias para execução e compilação no ambiente QUARTUS II da Altera.

O esquema da figura 3.1, ilustra o diagrama de fluxo do LOGO²VHDL no qual o formulário principal é responsável em exibir as funções contidas no programa. O formulário

de definição de parâmetros tem como função exibir os parâmetros de entrada e saída que serão selecionados para cada função programada. Quando todas as funções são implementadas, o formulário principal é exibido novamente com o código VHDL final para simulação.

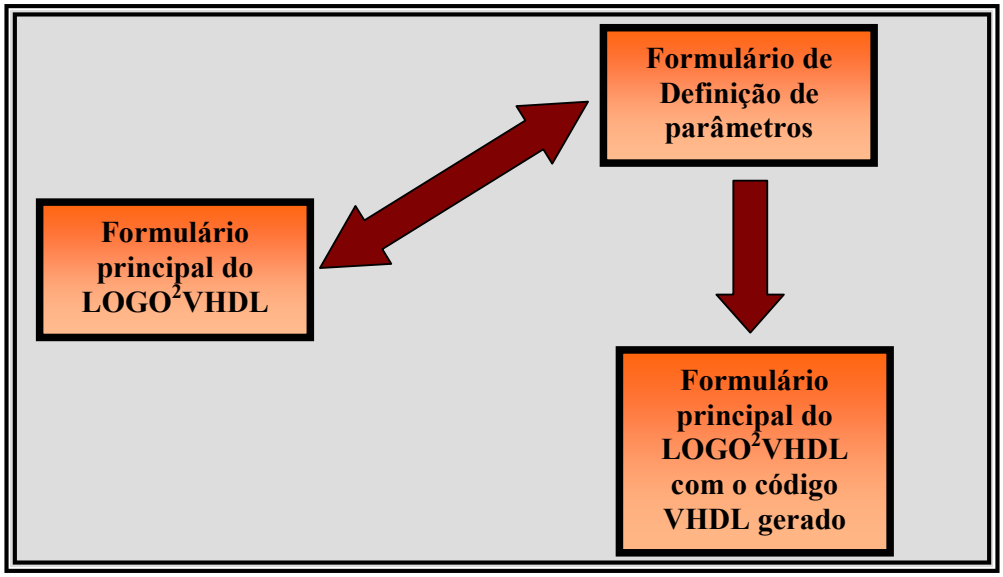


Figura 3.1 – Fluxograma do sistema LOGO²VHDL

Para o desenvolvimento do software conversor, foi necessário criar dois formulários no Delphi, um para apresentação das funções e outro para definição dos parâmetros a ser inseridos pelo usuário.

A figura 3.2 apresenta o formulário principal.

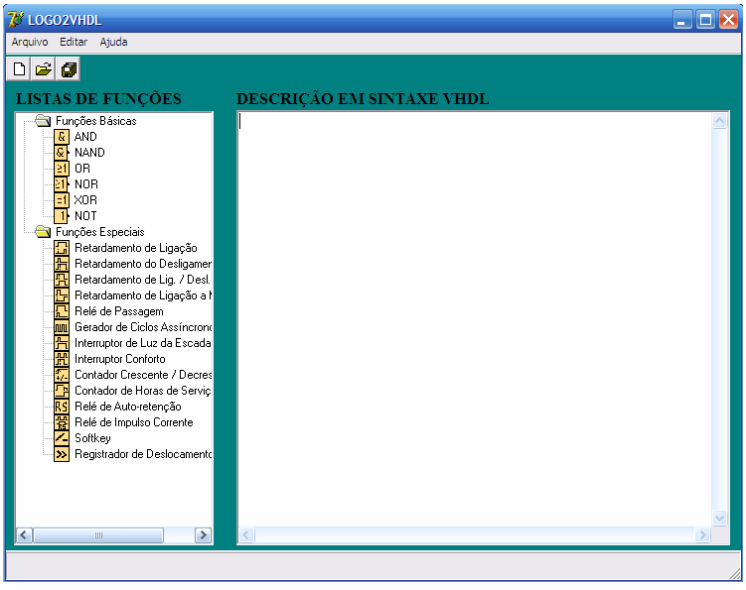


Figura 3.2 - Ilustração do formulário principal do LOGO²VHDL

O formulário principal é composto por uma barra de menus e de ferramentas construídos para apresentação, edição e demais funções necessárias para se trabalhar com edição de texto, no caso a sintaxe descritiva VHDL.

Foram construídos os seguintes menus com suas respectivas funções conforme descrito a seguir:

Menu Arquivo

Novo – Limpa a caixa de texto;

Abrir – Abre um arquivo já salvo;

Fechar – Fecha o programa em execução;

Salvar – Salva o arquivo, se for a primeira vez especifica o diretório;

Salvar Como – Especifica o diretório que irá salvar o arquivo;

Sair – Sai do programa em execução.

Menu Editar

Recortar – Recorta um texto selecionado;

Copiar – Copia um texto selecionado;

Colar – Cola um texto selecionado.

Menu Ajuda

Sobre – Exibe informações sobre o programa.

Foram desenhados os símbolos de cada uma das funções, pois preocupou-se, em apresentar estes símbolos conforme são oferecidas no LOGO!Soft, para que o projetista tenha uma semelhança no desenvolvimento do seu projeto.

Do lado esquerdo do formulário principal foram construídas as listas de funções básicas e funções especiais, que serão utilizadas, no processo de descrição dos sistemas a serem realizados e a parte central, contém uma caixa de texto, onde é apresentado o modelo VHDL correspondente a cada função.

Na figura 3.3 apresenta-se o formulário, para a especificação dos parâmetros de entrada e saída, assim como as conexões.

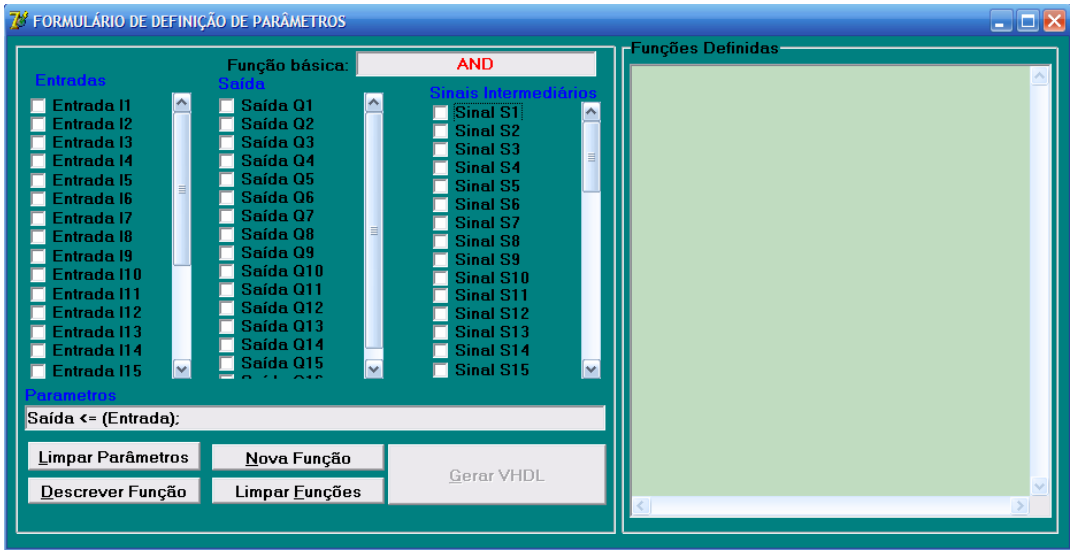


Figura 3.3 - Ilustração do formulário de definição de Parâmetros do LOGO^{2}VHDL

Este formulário é chamado quando uma função básica ou especial é acionada através de uma seleção, com o botão do mouse. Sua função é de descrever os parâmetros de entrada e saída referente à função que está em uso no momento. Após realizar a descrição, o código, já em sintaxe VHDL, é enviado para uma caixa de texto (Funções Definidas) e armazenado.

Neste formulário, foram criados alguns componentes necessários, para a definição dos parâmetros de entrada e saída e botões, para a manipulação dos códigos VHDL gerados.

Neste *layout* são oferecidos 24 componentes, para o usuário definir as entradas da função, 16 para definição de parâmetros de saída e 50 para definição dos sinais intermediários. Define-se como sinal intermediário, o elo de ligação entre a saída de uma função e a entrada de outra, ou seja, se a saída de uma função AND é a entrada de uma função OR, por exemplo, define-se esta ligação como sendo um Sinal Intermediário.

Estabeleceu-se 24 possibilidades, para as entradas e 16 para as saídas, devido ao fato de que o Controlador Lógico Programável LOGO possui exatamente essa limitação física de parâmetros de entradas e saídas, porém a quantidade de sinais internos torna-se variável de acordo, com a estrutura do programa a ser desenvolvido. Diante deste fato, criou-se 50 sinais intermediários, inseridos dentro de um componente, que contém uma barra de rolagem, oferecendo ao projetista a possibilidade de utilizar, uma grande quantidade de sinais, para o sistema e ao mesmo tempo oferecer um formulário de tamanho reduzido.

Neste formulário existem também dois painéis. Um painel ilustra a função, que foi acionada, para a definição dos parâmetros e o outro mostra a função básica, que será descrita.

Após a descrição das funções, as mesmas são enviadas, para um componente de texto, que posteriormente irá gerar o código VHDL completo desta estrutura.

Existe no formulário de definição dos parâmetros, cinco botões cujas funções são descritas a seguir:

- **Limpar Parâmetros** – Este botão desabilita os componentes marcados nos parâmetros de entrada e saída e limpa também a expressão exibida no painel de Parâmetros;
- **Descrever Função** – Uma vez definidas corretamente as entradas e saídas da função, este botão, quando acionado envia a informação contida no painel de Parâmetros para a caixa de texto que se encontra em uma janela denominada "Funções Definidas";
- **Nova Função** – Caso uma função esteja relacionada a outra, este botão uma vez acionado, oferece a possibilidade de se programar uma nova função e relacioná-la com a anterior;
- **Limpar Funções** – O botão "Limpar Funções" tem como objetivo limpar todos os códigos descritos na caixa de texto referente às funções definidas;
- **Gerar VHDL** – Este botão refere-se ao último passo a ser executado. Uma vez acionado, a função será descrita em VHDL com toda a estrutura necessária para a simulação e síntese visando a implementação em dispositivo lógico programável.

Além destes botões existe uma outra função neste formulário que inverte um parâmetro de entrada. Trata-se da função NOT, que funciona como inversor de uma entrada a medida que um componente (I ou S) é selecionado e habilitado na cor vermelha do formulário de definição de parâmetros.

3.2. Exceções Tratadas

Foi necessário tratar algumas situações especiais, tendo em vista, que o software "LOGO^{2}VHDL", se relaciona com dois outros ambientes, o LOGO!Soft e o QUARTUS II da Altera. De acordo com o LOGO!Soft cada função básica, com exceção do XOR, tem que possuir no mínimo uma entrada e no máximo quatro. Desta forma, as seguintes situações foram tratadas:

- Se um sinal intermediário é marcado como saída da função, as saídas (Q) serão desabilitadas, tendo em vista que cada função pode ter somente uma saída;
- Não é possível descrever uma função básica ou especial sem ser definida uma saída. Quando esta situação ocorre a mensagem apresentada na figura 3.4 é gerada;

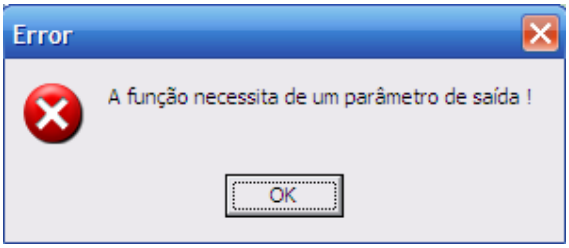


Figura 3.4 – Mensagem para definição de parâmetro de saída

- Não é possível descrever uma função básica com somente uma entrada. Quando isso ocorre aparecerá uma mensagem conforme apresentado na figura 3.5.

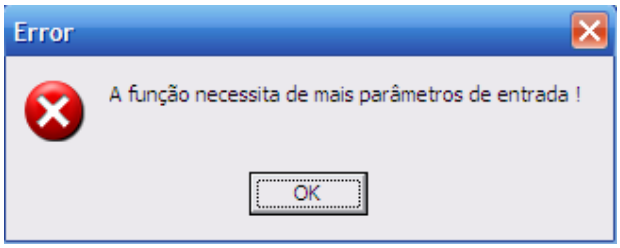


Figura 3.5 – Mensagem para definição de parâmetro de entrada

- Não é possível definir mais de quatro parâmetros de entrada para uma função básica. Quando isso ocorre aparecerá a mensagem da figura 3.6 e o componente não é habilitado.

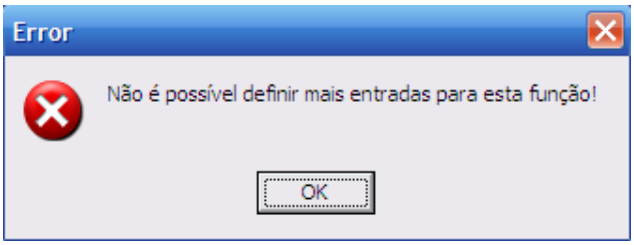


Figura 3.6 – Mensagem para excesso de parâmetros de entrada

- Para a função XOR não é possível selecionar somente um parâmetro de entrada. Caso o usuário tente descrever a função com somente uma entrada, aparecerá a mensagem da figura 3.5. No entanto, só é possível marcar duas entradas. Caso o usuário tente marcar três ou mais, aparecerá a mensagem conforme ilustrado na figura 3.6.
- Existem funções especiais que possuem limites para inserção de parâmetros de entrada, ou seja, há funções especiais que possuem uma, duas ou até três entradas. Tendo em vista estas diferenças, foi realizado o tratamento individual para cada uma

das funções especiais, estabelecendo o limite de entradas de acordo com a característica de cada uma delas.

Os exemplos apresentados a seguir foram importantes para realizar basicamente três etapas: relacionar uma função básica à outra, realizar o elo de ligação entre todas as funções básicas e relacionar funções básicas e especiais conjuntamente.

3.3. Exemplos

Para ilustrar o uso do LOGO^{2}VHDL alguns testes foram realizados e são apresentados a seguir. Alguns circuitos simples foram implementados, com o objetivo de testar o funcionamento do programa tradutor e as funções mais complexas, mostram as potencialidades do programa, com o intuito de posteriormente tratar sistemas de controle de uso prático.

3.3.1. Exemplo 01

O circuito apresentado na figura 3.7 é um circuito contendo três entradas I1, I2, e I3 e uma saída Q1. O Circuito implementa a função Q1 = (I1 . I2) + I3.

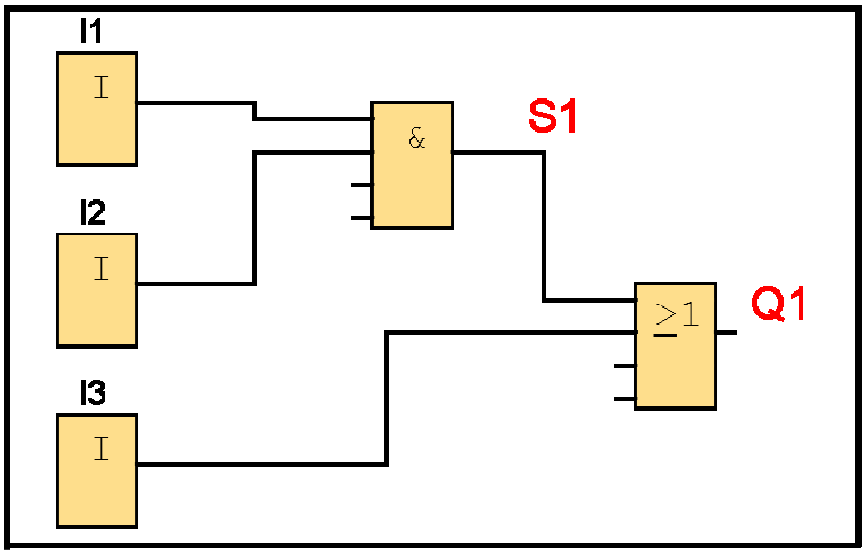


Figura 3.7 – Circuito implementando a função Q1 = (I1 . I2) + I3

Uma vez programadas todas as funções básicas de forma independente, é necessário relacionar uma função à outra, justamente para realizar o tratamento dos sinais internos. Este exemplo ilustra uma função AND relacionada a uma função OR.

Percebe-se através da figura 3.8 que são geradas suas linhas de códigos para a estrutura da figura 3.7. Desta maneira, a próxima etapa é gerar o código VHDL completo para este circuito.

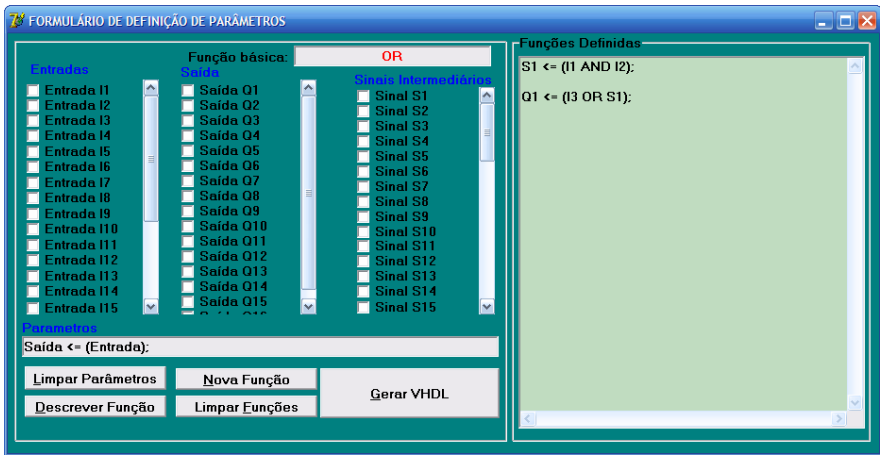


Figura 3.8 – Linhas de código definidas para a função Q1 = (I1 . I2) + I3

A figura 3.9 apresenta o código VHDL, com todas as declarações realizadas, bibliotecas e demais estruturas necessárias em sintaxe VHDL, para compilação do programa no QUARTUS II, conforme observa-se na figura 3.10, e posterior implementação em FPGA.

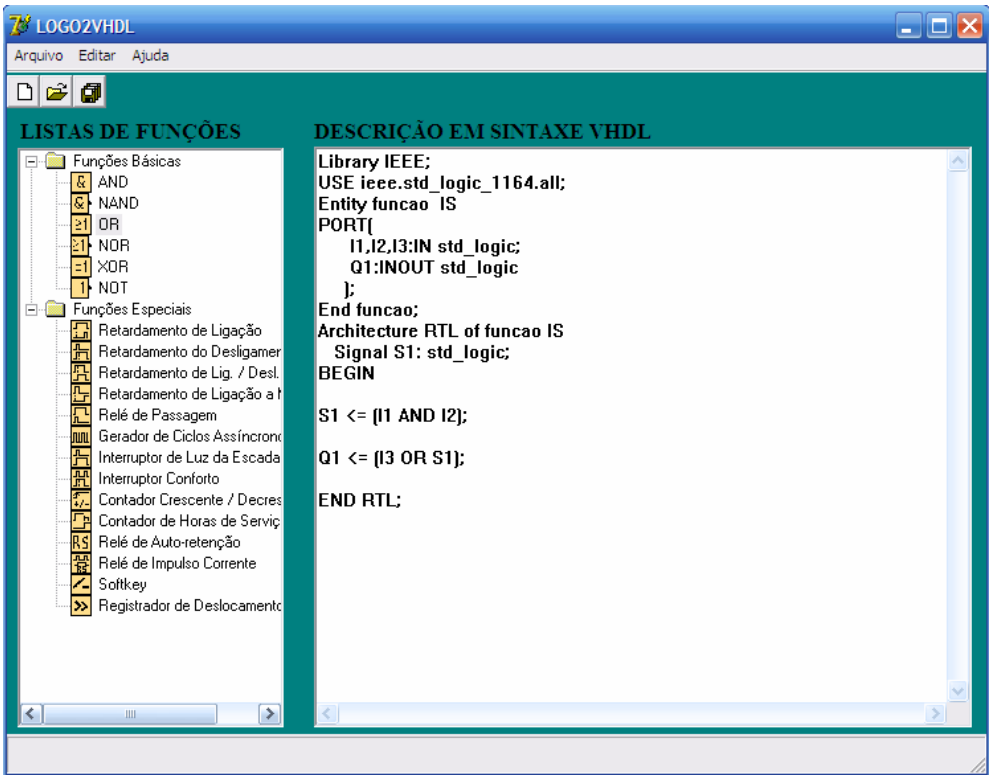


Figura 3.9 – Código VHDL completo para a função Q1 = (I1 . I2) + I3

A figura 3.10 mostra que a simulação do circuito obtido através da sintaxe VDHL gerada pelo LOGO^{2}VHDL descreve corretamente a função Q1 = (I1 . I2) + I3.

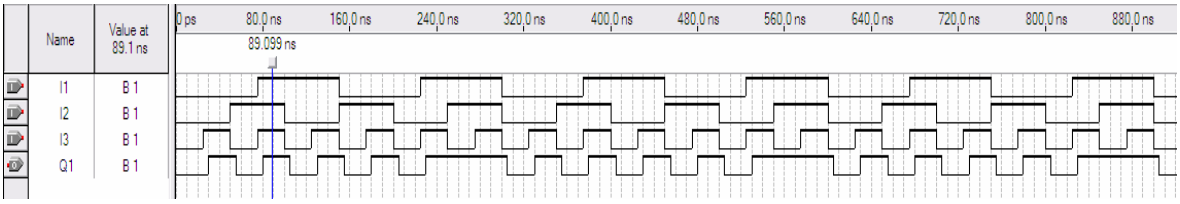


Figura 3.10 – Simulação do Código VHDL para a função Q1 = (I1 . I2) + I3

3.3.2. Exemplo 02

O circuito apresentado na figura 3.11, trata-se de um circuito mais complexo, com vários sinais de entrada e utilizando-se de mais portas lógicas (AND, OR, NAND e NOR, NOT e XOR). Este exemplo teve como objetivo integrar uma maior quantidade de funções básicas.

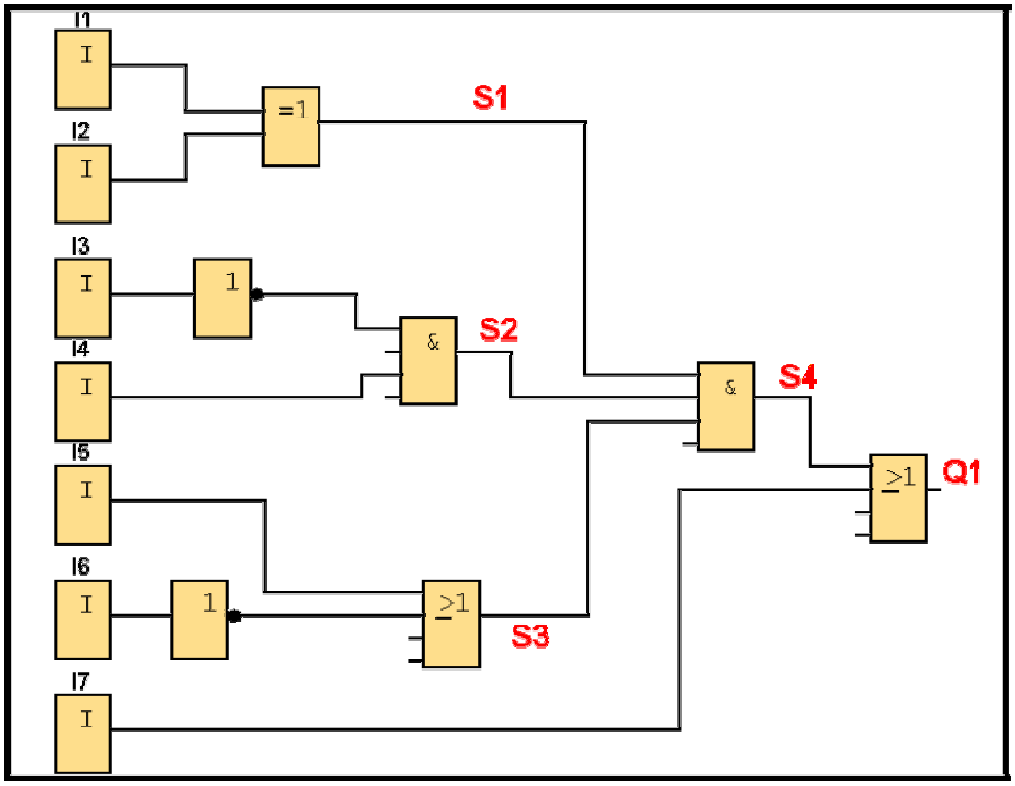


Figura 3.11 – Circuito implementando a função Q1 = (I7 or ((I1 and not (I2)) or (not(I1) and I2)) and ((not I3) and I4) and ((I5 or (not I6))))

Na figura 3.12 são geradas as linhas de códigos para a estrutura da figura 3.11. Após esta etapa, será gerado o código VHDL completo para o circuito.

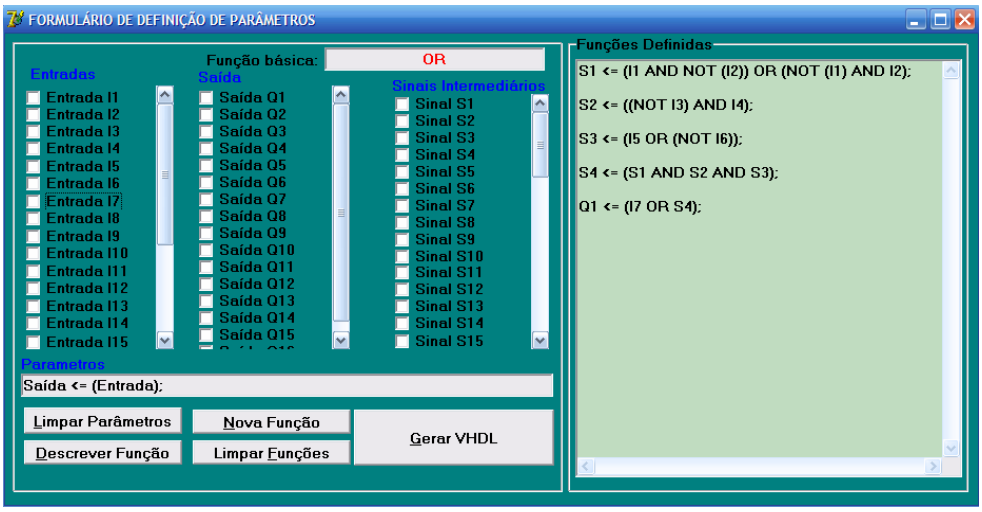


Figura 3.12 – Linhas de código definidas para a função Q1 = (I7 or ((I1 and not (I2)) or (not(I1) and I2)) and ((not I3) and I4) and ((I5 or (not I6))))

A figura 3.13 apresenta o código VHDL com todas as declarações realizadas, bibliotecas e demais estruturas necessárias em sintaxe VHDL para compilação do programa no QUARTUS II como observa-se na figura 3.14.

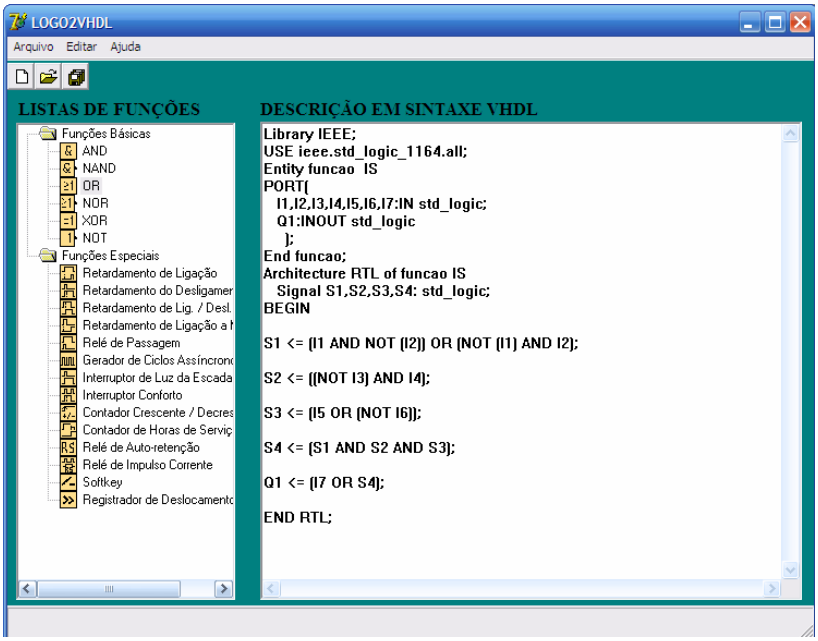


Figura 3.13 – Código VHDL completo para a função Q1 = (I7 or ((I1 and not (I2)) or (not(I1) and I2)) and ((not I3) and I4) and ((I5 or (not I6))))

A figura 3.14 mostra que a simulação do circuito obtido através da sintaxe VDHL gerada pelo LOGO^{2}VHDL descreve corretamente a função Q1 = (I7 or ((I1 and not (I2)) or (not(I1) and I2)) and ((not I3) and I4) and ((I5 or (not I6)))).

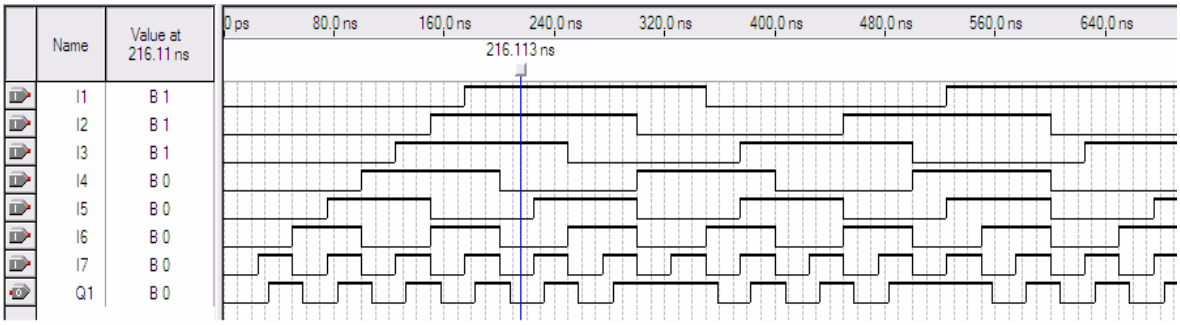


Figura 3.14 – Simulação do Código VHDL para a função Q1 = (I7 or ((I1 and not (I2)) or (not(I1) and I2)) and ((not I3) and I4) and ((I5 or (not I6))))

3.3.3. Exemplo 03

Uma vez concluído os testes realizados nas funções básicas, realizou-se testes das funções especiais. Esta primeira implementação, conforme ilustrado na figura 3.15 foi importante tendo em vista, que as funções especiais devem se relacionar, com as funções básicas.

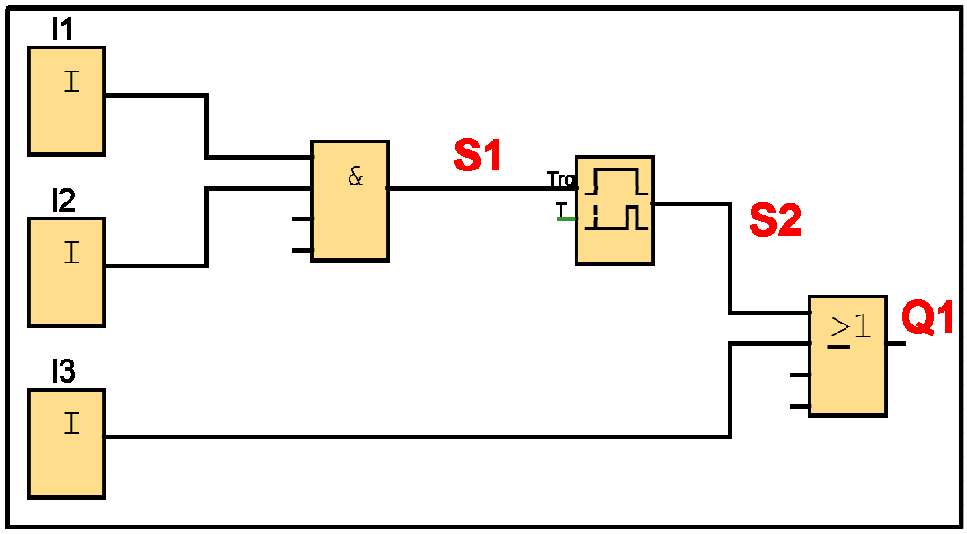


Figura 3.15 – Esquemático de uma função especial (Retardamento de Ligação) relacionada a duas funções básicas (And e Or)

Observa-se através da figura 3.16 que são geradas as linhas de códigos para a estrutura da figura 3.15. Neste exemplo, verifica-se que foi descritos a função And, a função retardo de ligação e posteriormente a função Or. A próxima etapa é gerar o código VHDL completo para esta estrutura.

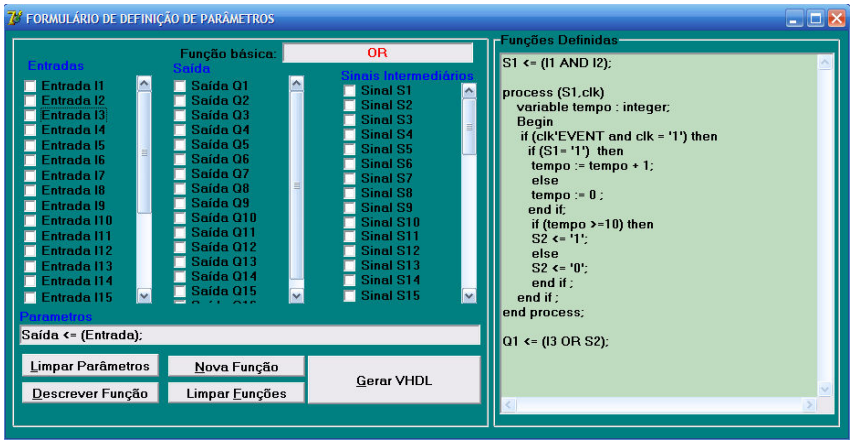


Figura 3.16 – Linhas de código definidas para o esquemático de uma função especial (Retardamento de Ligação) relacionada a duas funções básicas (And e Or)

A figura 3.17 mostra a última etapa realizada, o código VHDL final com todas as declarações realizadas, bibliotecas e demais estruturas necessárias em sintaxe VHDL. E a figura 3.18 a compilação do programa no QUARTUS II.

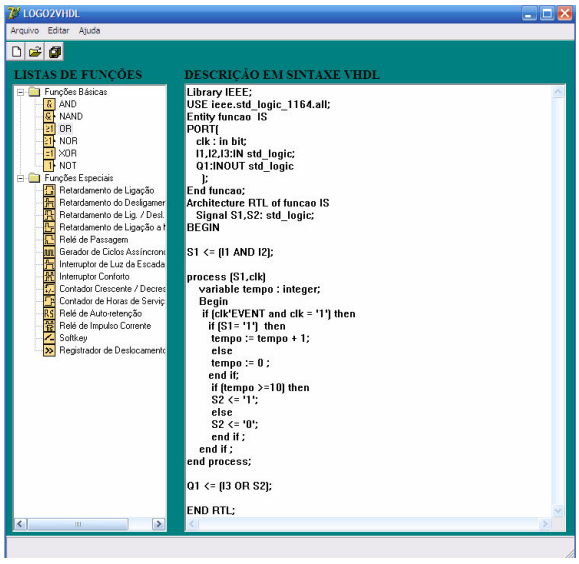


Figura 3.17 – Código VHDL completo para o esquemático de uma função especial (Retardamento de Ligação) relacionada a duas funções básicas (And e Or)

A figura 3.18 mostra que a simulação do circuito obtido através da sintaxe VDHL gerada pelo LOGO^{2}VHDL descreve corretamente a função especial Retardamento de Ligação relacionada a duas funções básicas And e Or.

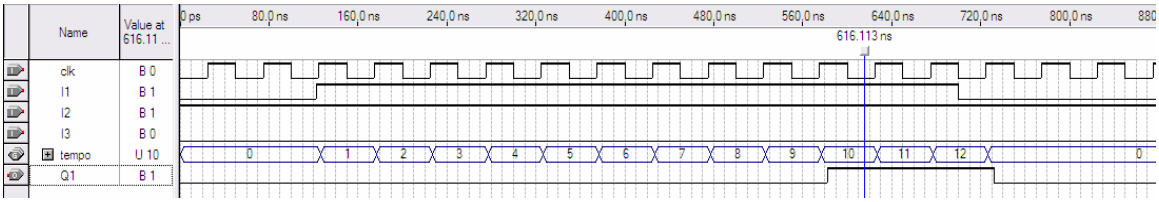


Figura 3.18 – Simulação do Código VHDL ilustrando o funcionamento de uma função especial (Retardamento de Ligação) relacionada a duas funções básicas (And e Or)

No exemplo pode-se verificar que o retardo de ligação é ativado, quando as duas entradas I1 e I2 possuírem o nível lógico 1, desta forma, o contador inicializa o processo de contagem e finaliza quando o tempo parametrizado é atingido, que para este exemplo é de 10 pulsos de relógio. Quando o contador atingir o valor parametrizado, a saída que possuía o nível lógico 0 passará a assumir o nível lógico 1.

Apresenta-se no próximo capítulo, alguns estudos de casos extraídos da literatura, cujo objetivo é avaliar o sistema desenvolvido em controle de plantas reais.

Capítulo 4

Estudo de casos

4.1. Introdução

Após a realização de vários testes, conforme ilustração dos exemplos citados no capítulo anterior, a ferramenta desenvolvida denominada LOGO^{2}VHDL foi avaliada através de sistemas de controle descritos na literatura. Os exemplos foram extraídos dos catálogos do fabricante de CLP Siemens.

4.2. Porta Automática

O sistema apresentado na figura 4.1 possui quatro entradas denominadas de I1, I2, I3 e I4. As entradas I1 e I2 funcionam como detectores de movimento de pessoas nos dois lados da porta, interno e externo, respectivamente, que são acionados no circuito através de sensores. As entradas I3 e I4 são interruptores limites (fim de curso) para porta fechada e aberta respectivamente, trata-se de entradas que indicam as condições favoráveis ao sistema para a porta ser aberta e também ser fechada.

O circuito possui duas saídas Q1 e Q2, onde Q1 ativo, indica que a porta se encontra aberta e Q2 ativo indica que a porta se encontra fechada.

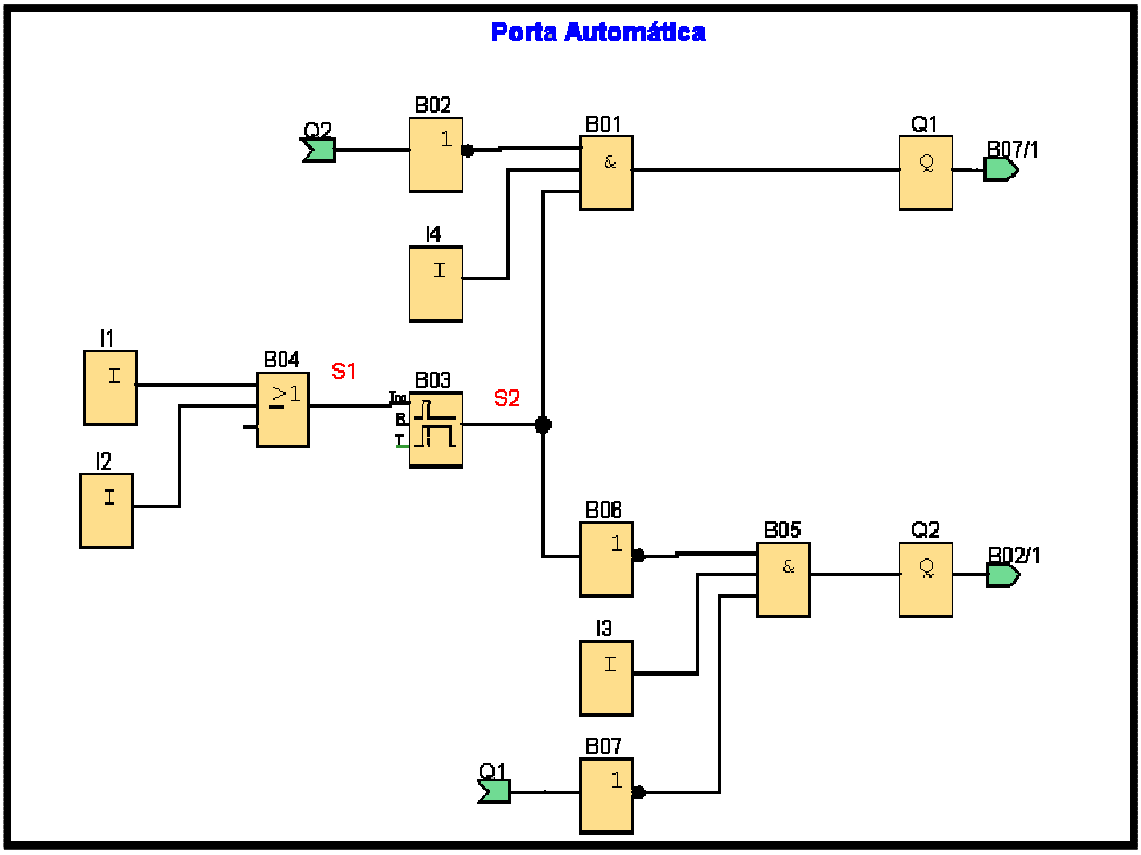


Figura 4.1 - Ilustração de uma porta automatizada

Percebe-se que para o funcionamento da porta automatizada, é necessário que pelo menos um dos dois sensores (I1 e I2) detecte a passagem de uma pessoa, ou seja, pelo menos uma das duas entradas devem estar ativadas. É necessário também, que os interruptores para o funcionamento da porta na condição “abrir” e “fechar” estejam devidamente acionados (I3 e I4 habilitados).

Este exemplo possui um parâmetro de tempo, para o funcionamento da porta. A medida que um indivíduo é identificado, a porta se abre e permanece aberta durante um período de tempo pré-definido. Estabeleceu-se uma parametrização de 10 pulsos de relógio para a permanência da porta aberta.

A figura 4.2, ilustra a simulação deste sistema de controle no software LOGO!Soft Comfort da Siemens onde o intervalo do circuito em cor vermelha indica a passagem de corrente.

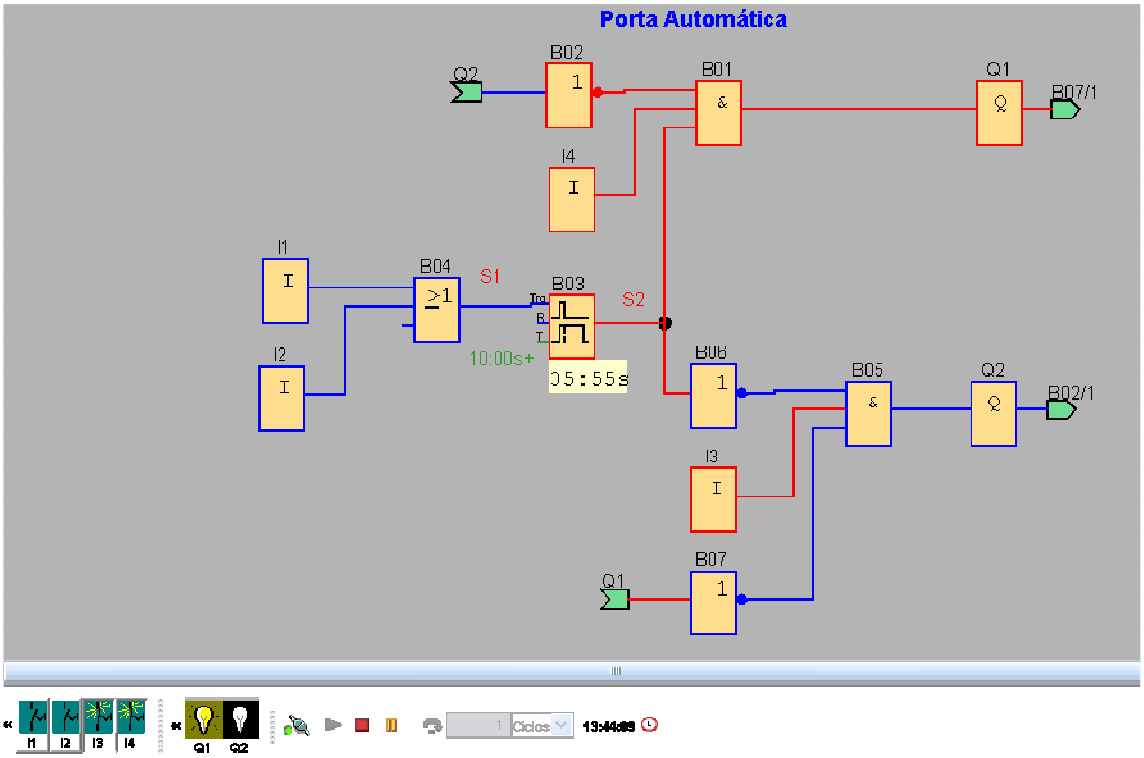


Figura 4.2 – Simulação da porta automática no LOGO!Soft Comfort

Pode-se notar através dos sensores representados pelas entradas I3 e I4 que a porta está em condições de ser acionada (nível lógico 1). Na ilustração a entrada I2 foi acionada e desligada novamente inicializando o sistema. Neste momento, verifica-se que a saída Q1 se encontra ativa, informando que a porta se encontra aberta. Verifica-se também que o contador está sendo incrementado. No momento que este contador atingir o valor parametrizado, que é de 10 pulsos de relógio, a saída Q1 é desabilitada e a saída Q2 adquire o nível lógico 1 indicando que a porta se encontra fechada.

Esta mesma estrutura foi gerada em VHDL, podendo-se observar que é possível converter este sistema de controle, para linguagem de descrição de hardware utilizando a ferramenta LOGO^{2}VHDL. Descreve-se a seguir este sistema de controle em linguagem de descrição de hardware.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity port_autom IS
        Port(
                clk : in bit;
                Q1, Q2 : inout std_logic;
```

```
                I1, I2, I3, I4 : in std_logic
                );
        End port_autom;
Architecture funcao of port_autom IS
Signal S1, S2: std_logic;

Begin

S1 <= (I1 OR I2);

process (S1,clk)
  variable tempo : integer;
  begin
   if (clk'EVENT and clk = '1') then
      if S1 = '1' then
      S2 <= '1';
      tempo := 0;
     else
      tempo := tempo + 1;
      if (tempo = 10) then
      S2 <= '0';
    end if;
   end if;
  end if ;
end process;

Q1 <= (I4 AND S2 AND (NOT Q2));

Q2 <= (I3 AND (NOT S2) AND (NOT Q1));

End funcao;
```

Pode-se notar na descrição apresentada, que o código VHDL possui todas as declarações realizadas, bibliotecas e demais estruturas necessárias, para a compilação da descrição no ambiente de síntese Quartus II da Altera.

Através da forma de onda, ilustrado na figura 4.3, verifica-se que o sistema de controle funciona de acordo com o especificado.

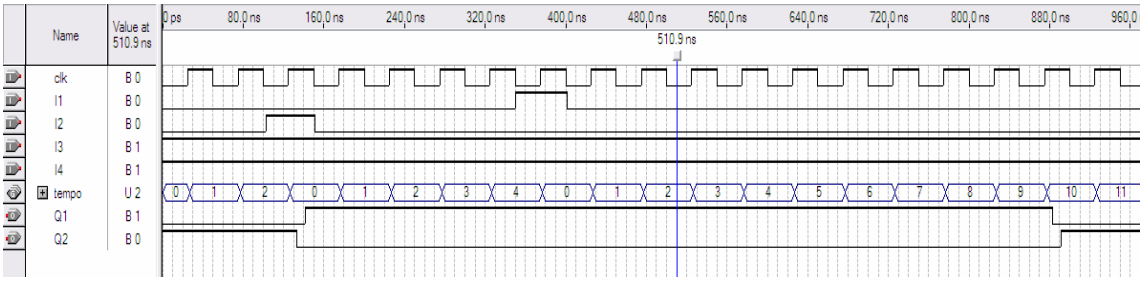


Figura 4.3 - Simulação da porta automática no Quartus II da Altera

Através da simulação pode-se perceber o funcionamento do circuito descrito em linguagem de descrição de hardware. Assim verifica-se, que as entradas I3 e I4 devem estar ativas (nível lógico 1) para indicar, que as condições para a porta abrir e fechar estão adequadas.

Verifica-se também, que quando uma das entradas I1 ou I2 são acionadas, o contador é incrementado e durante o tempo parametrizado que é de 10 pulsos de relógio a saída Q1 possui nível lógico 1 indicando que a porta se encontra aberta. Após este período de parametrização a saída Q2 possuirá nível lógico 1 indicando, que a porta foi fechada. Neste momento o sistema aguarda uma nova comutação em uma das entradas I1 e I2, indicando que um indivíduo deseja passar pela porta novamente.

Pôde-se verificar duas descrições diferentes através da figura 4.1 apresentada no LOGO!Soft Comfort da Siemens e da descrição em VHDL proposta para o sistema de controle, porém com uma funcionalidade correspondente. É possível perceber também, que apesar de serem formas de programação diferente todas as entradas (I1, I2, I3 e I4) e saídas (Q1 e Q2) realizam a mesma lógica de programação para o funcionamento do sistema.

A figura 4.4 mostra a planta baixa deste sistema de controle implementado em FPGA.

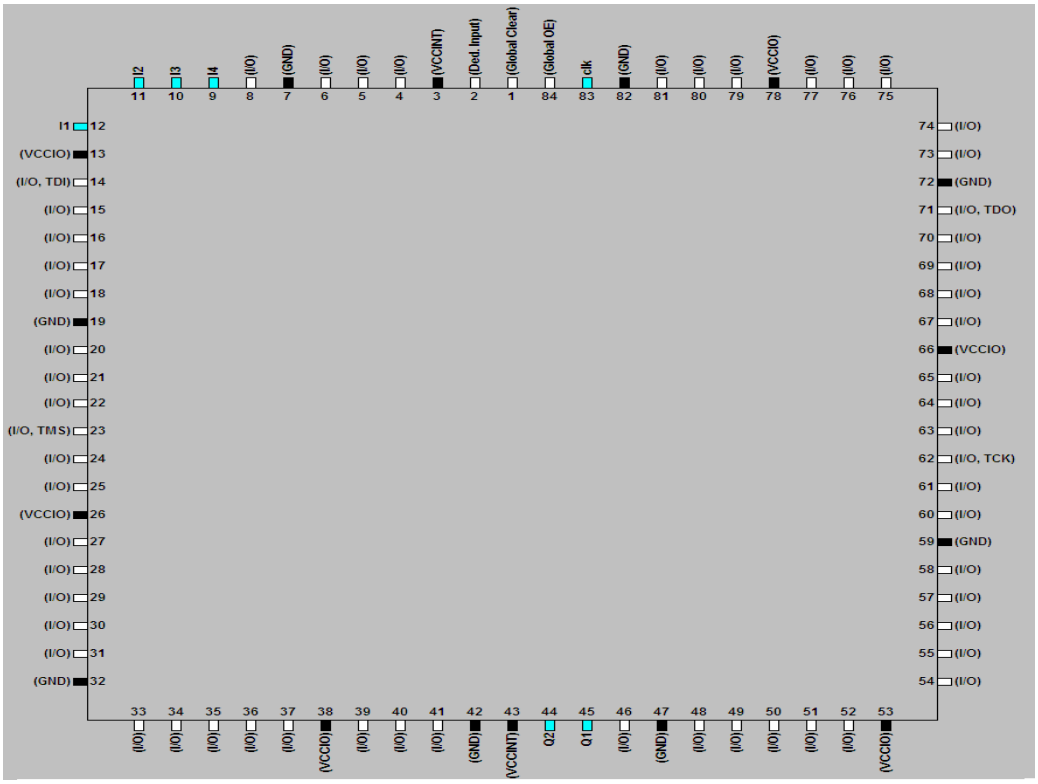


Figura 4.4 - Ilustração da planta baixa do sistema de irrigação

Constata-se que todos os processos relativos ao sistema de controle foram descritos, a simulação observar-se que a sintaxe em VHDL proposta pelo software LOGO^{2}VHDL descreve o sistema de controle de acordo com o esperado e o mesmo foi programado em um dispositivo lógico programável – FPGA.

4.3. Sistema de Irrigação Automatizado

Este circuito, conforme ilustrado na figura 4.5, possui quatro entradas, que funcionam como interruptores I1, I2, I3 e I4. O interruptor I1 funciona como uma chave que habilita o bombeamento da água canalizada durante o período parametrizado de 20 pulsos de relógio. O interruptor I2 é uma chave que interrompe a passagem de água nos canos principais de água. O interruptor I3 é uma chave que indica as condições favoráveis ou não de proteção externa do sistema de irrigação. Da mesma forma, I4, indica as condições internas do sistema.

Existem duas saídas no esquemático, Q1 e Q2, onde Q1 ativo, indica que o sistema está bombeando água e Q2 ativo informa, que se encontra água nos canos principais. A saída Q2 desativa quando a entrada I2 se encontra ativa, indica que a passagem de água no reservatório se encontra temporariamente interrompida, porém quando a saída Q2 encontra-se desativa durante o processo de bombeamento (I1, I3 e I4 ativos), indica que a água está sendo transferida nos canos principais do sistema.

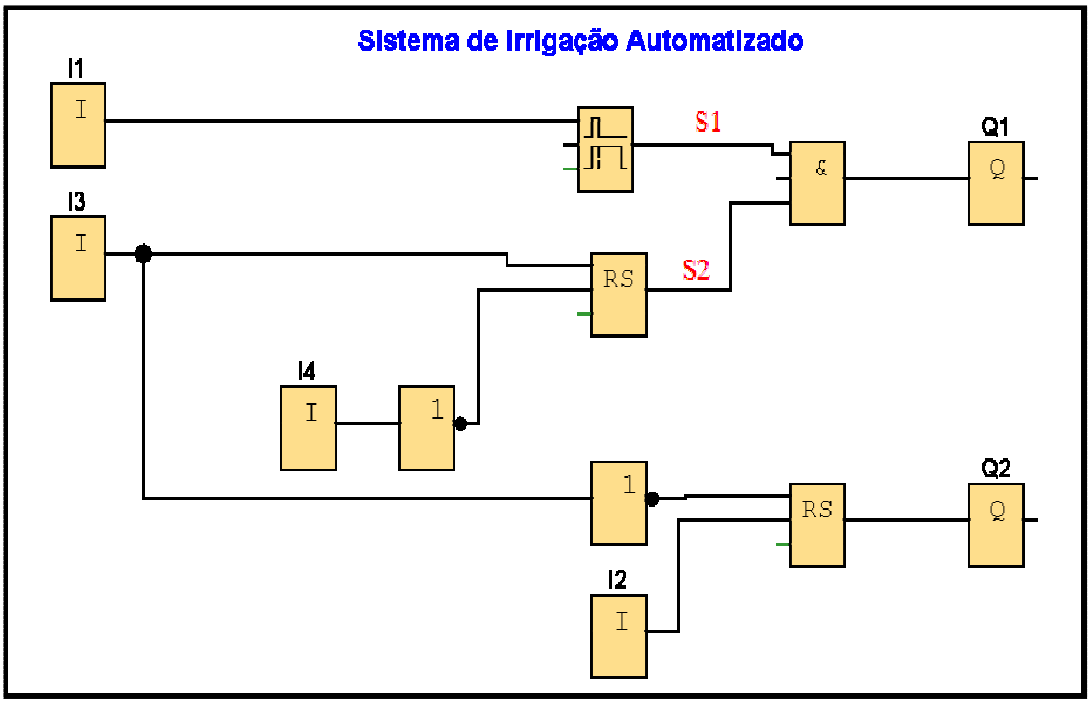


Figura 4.5 - Ilustração de um sistema de irrigação automatizado

Assim, percebe-se que para o funcionamento do sistema de irrigação, deve-se habilitar o bombeamento de água (I1 ativo), deve-se indicar a passagem de água no reservatório (I2 desabilitado) e os fatores externos e internos ao sistema (condições físicas de instalação) estejam adequados (I3 e I4 habilitados) conforme ilustra a figura 4.6.

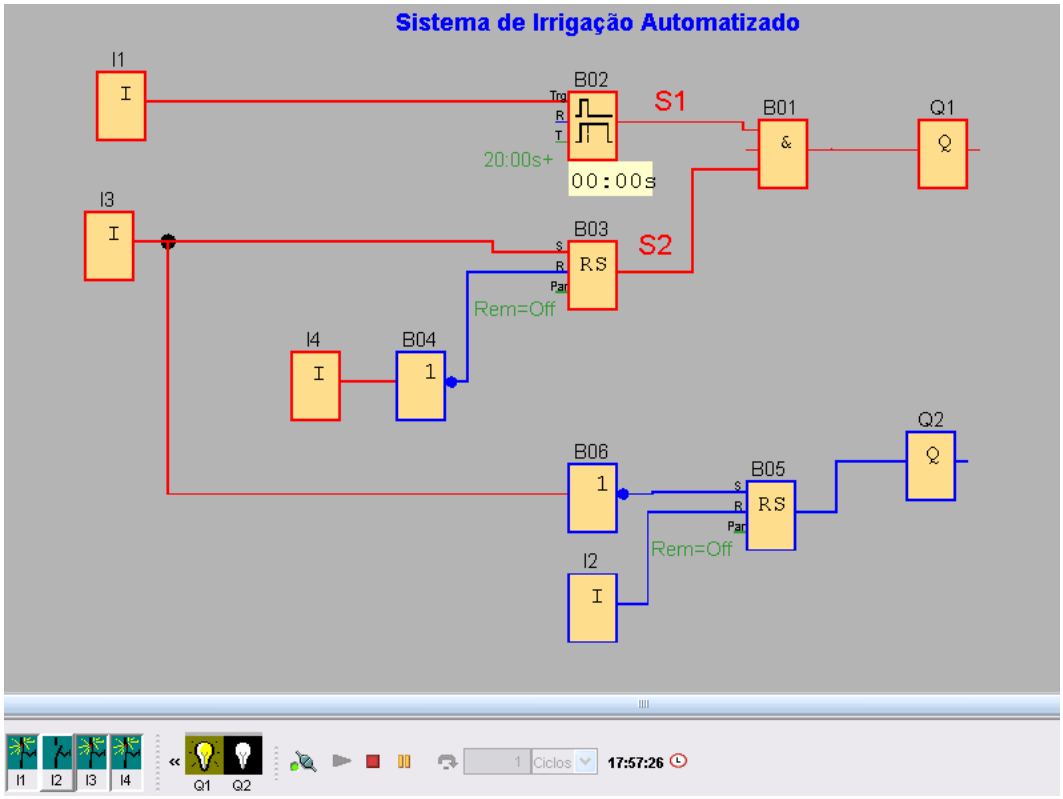


Figura 4.6 – Simulação do Sistema de Irrigação automatizado no LOGO!Soft Comfort

Verifica-se que as chaves I3 e I4 estão habilitadas, indicando que as condições externas e internas, respectivamente, se encontram favoráveis para o início do bombeamento de água. Percebe-se também que a entrada I2 se encontra desabilitada, indicando que a água está sendo transferida nos canos do sistema. Por sua vez, a entrada I1 habilita o bombeamento. Constata-se que neste instante as saídas Q1 se encontra ativa, indicando o bombeamento de água e Q2 se encontra desativado, indicando a passagem de água na canalização.

Através deste exemplo, observa-se a codificação do sistema de controle de forma estrutural, para forma descritiva em sintaxe VHDL torna-se possível utilizando o LOGO^{2}VHDL. Segue abaixo a descrição deste sistema de controle em linguagem de descrição de hardware.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity exemplo IS
Port(
        clk, I1, I2, I3, I4 : in std_logic;
        Q1, Q2 : out std_logic
          );
End exemplo;
Architecture funcao of exemplo IS
Signal S1, S2: std_logic;
BEGIN
  process(I1, clk)
  variable tempo: integer;
  Begin
     if (clk'EVENT and clk = '1') Then
             if I1 = '1' then
                   S1 <= '1';
                   tempo := 0;
                   else
                     tempo := tempo + 1;
               if (tempo = 20) then
                   S1 <= '0';
               end if;
             end if;
       end if;
end process;
process (I3, I4)
BEGIN
   if (I3 = '1') AND (Not I4 = '0') then
         S2 <= '1';
    elsif (I3 = '0') AND (Not I4 = '1') then
         S2 <= '0';
    else
         S2 <= '0';
  end if;
end process;
Q1 <= (S1 and S2);
process (I3, I2)
BEGIN
 if (Not I3 = '1') AND (I2 = '0') then
         Q2 <= '1';
  elsif (Not I3 = '0') AND (I2 = '1') then
         Q2 <= '0';
  else
         Q2 <= '0';
  end if;
end process;
End funcao;
```

Nota-se na descrição apresentada que o código VHDL pode ser compilado no ambiente de síntese Quartus II da Altera, sem erros de lógica e sintaxe. Através da formas de onda, ilustrado na figura 4.7, pode-se verificar também que o sistema de controle funciona de forma correspondente ao especificado.

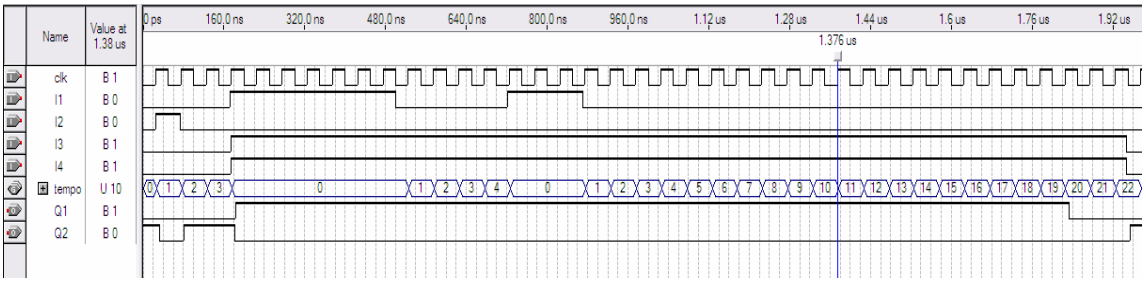


Figura 4.7 - Simulação do sistema de irrigação no Quartus II da Altera

Observa-se que se a entrada I2 é acionada a saída Q2 permanece em nível lógico 0 indicando que se encontra água no reservatório no entanto através desta entrada é informada uma interrupção temporária da água nos canos principais.

Percebe-se também que quando as três entradas I1, I3 e I4 são acionadas imediatamente a saída Q1 é habilitada e Q2 desabilitada, ou seja, o reservatório é bombeado e a água é transferida nos canos do sistema.

Verifica-se também que após uma nova comutação na entrada I1 um contador é disparado e o sistema bombeia água durante todo o período parametrizado no caso 20 pulsos de relógio, pois ativa a função retardo do desligamento e as condições externas e internas do sistema se encontram favoráveis (I3 e I4 habilitados). Quando o contador encerra este período, o sistema de irrigação finaliza o bombeamento de água.

Este modelo de irrigação automatizado visa estabelecer e definir uma metodologia de funcionamento com uma uniformidade de distribuição de água. A descrição deste sistema de controle em linguagem de descrição de hardware torna-se interessante principalmente devido às interrupções e saídas que são exigidas em um curto período de tempo, no caso 20 segundos, o que é facilmente tratado em VHDL devido ao paralelismo existente.

4.4. Sistema Automático para Preenchimento de Silo

Este sistema, conforme ilustrado na figura 4.8, colhe os dados necessários para o controle do silo, processando e controlando estes dados, de forma a preencher o reservatório conforme as necessidades do agricultor.

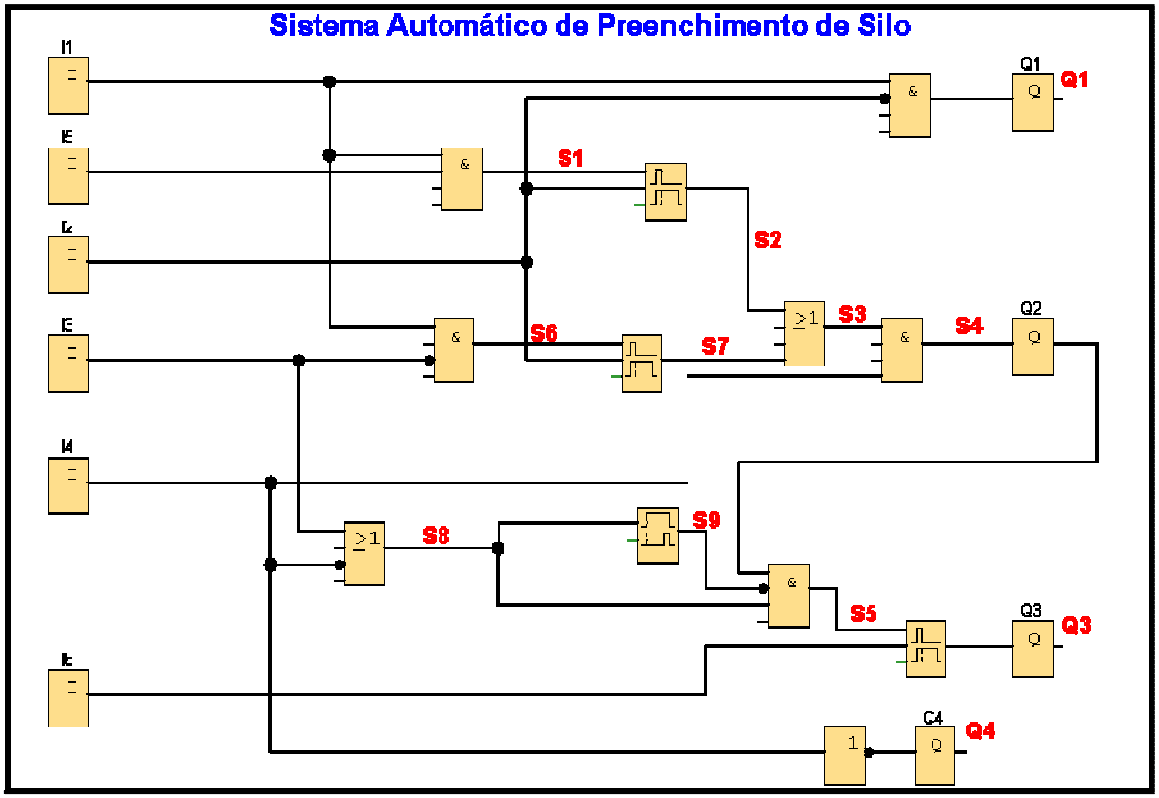


Figura 4.8 – Esquemático de um sistema de controle automático de preenchimento de silo

O Sinal intermediário, conforme visto anteriormente, é um conceito proposto neste trabalho, trata-se de um elo de ligação, entre uma função e outra. Assim, de posse do sistema de controle a primeira tarefa é descrever todos os sinais intermediários envolvidos no projeto. Para este caso, conforme se observa em vermelho na figura 4.8, foram descritos nove sinais intermediários (S1 a S9).

A simulação deste sistema de controle no LOGO!Soft Comfort é visualizado na figura 4.9.

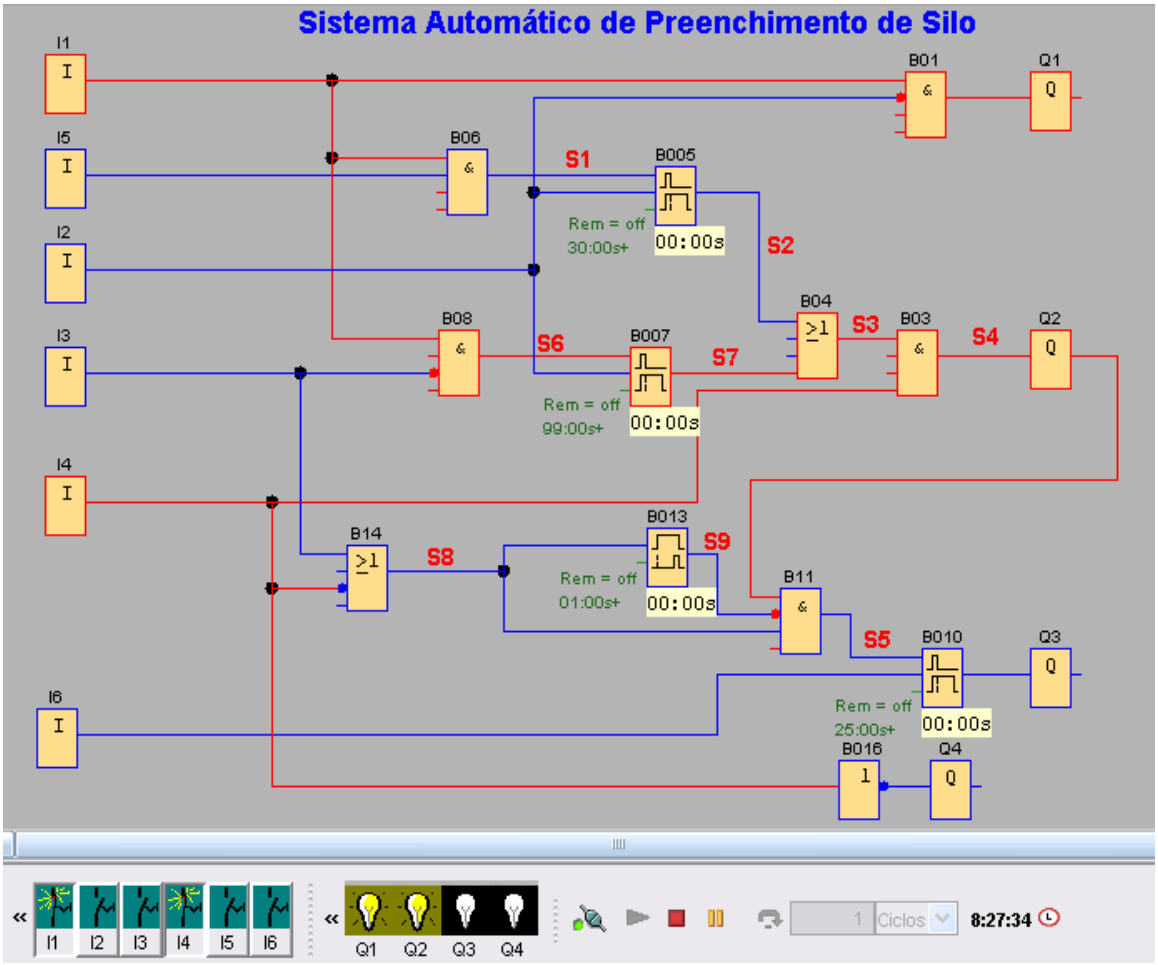


Figura 4.9 – Simulação do sistema de controle automático de preenchimento de silo no LOGO!Soft Comfort

O sistema só pode ser iniciado quando o botão I1 está habilitado e a mangueira para transferência dos grãos está corretamente conectada. Um contato dos grãos no reservatório é sinal de que a mangueira está conectada corretamente no silo. Este sinal no sistema de controle é representado pela entrada I2 em caráter desabilitado. Desta forma, a válvula de compressão em Q2 é então aberto, o filtro em Q1 é ativado simultaneamente.

O filtro tem que permanecer ativo ao longo do processo “enchendo”, para serem bombeados os grãos no silo. O indicador nivelado em I3 sinaliza quando o silo estiver cheio. Um sinal de alarme indicado através da saída Q3 informa que o processo permanece 99 pulsos de relógio até realizar a finalização automática do processo. A válvula no caminhão é fechada dentro deste período, para permitir o esvaziamento da mangueira.

O alarme pode ser reajustado manualmente pelo acionamento da entrada I6, caso não seja acionado, o sistema será desligado automaticamente depois de 25 pulsos de relógio. Se a

mangueira não for esvaziada a tempo, em 30 pulsos de relógio um procedimento de emergência pode ser ativado acionando o botão I5. O monitoramento de pressão no silo também termina automaticamente o procedimento de preenchimento no reservatório. Este caso é sinalizado através do indicador Q4.

O código fonte abaixo mostra a descrição em sintaxe VHDL de todo o esquemático acima citado, gerado pelo LOGO^{2}VHDL.

```
Library IEEE;
USE ieee.std_logic_1164.all;
Entity exemplo IS
        Port(
                clk : in std_logic;
                I1, I2, I3, I4, I5, I6 : in std_logic;
                Q1, Q3, Q4 : out std_logic);
        End exemplo;
Architecture funcao of exemplo IS
Signal S1, S2, S3, S4, S5, S6, S7, S8, S9: std_logic;
BEGIN
Q1 <= (I1 and (not I2));
S1 <= (I1 and I5);
        process(S1, clk)
                variable tempo: integer;
                Begin
                if (clk'EVENT and clk = '1') Then
                        if (I2 = '1') then
                        tempo := 0;
                        S2 <= '0';
                  else
                        if (S1 = '1') then
                                S2 <= '1';
                                tempo := 0;
                else
                                tempo := tempo + 1;
                 if (tempo = 30) then
                                  S2 <= '0';
                    end if;
                end if;
            end if;
        end if;
        end process;
S6 <= (I1 and (not I3));
        process(S6, clk)
                variable tempo: integer;
                Begin
                if (clk'EVENT and clk = '1') Then
                        if (I2 = '1') then
                        tempo := 0;
                        S7 <= '0';
```

```
                        else
                         if S6 = '1' then
                                S7 <= '1';
                                tempo := 0;
                else
                                tempo := tempo + 1;
                if (tempo = 99) then
                                  S7 <= '0';
                             end if;
                        end if;
                  end if;
                end if;
end process;
S3 <= (S2 or S7);
S4 <= (S3 and I4);
S8 <= (I3 or (not I4));
        process(S8, clk)
                variable tempo : integer;
                Begin
                if (clk'EVENT and clk = '1') Then
                        if (S8 = '1') then
                           tempo := tempo + 1;
                          else
                                tempo := 0;
                        end if;
                        if (tempo >= 1) then
                                S9 <= '1';
                        else
                                S9 <= '0';
                        end if;
                end if;
                end process;
S5 <= (S4 and (not S9) and S8);
        process(S5, clk)
                variable tempo: integer;
                Begin
        if (clk'EVENT and clk = '1') Then
                        if (I6 = '1') then
                                tempo := 0;
                                Q3 <= '0';
                        else
                         if (S5 = '1') then
                                Q3 <= '1';
                                tempo := 0;
                        else
                                tempo := tempo + 1;
                        if (tempo = 25) then
                                  Q3 <= '0';
                        end if;
                end if;
```

```
            end if;
          end if;
end process;
Q4 <= not I4;
End funcao;
```

Constata-se que todos os processos relativos ao sistema de controle foram descritos e sua simulação pode ser observada conforme mostra a figura 4.10. Pode-se observar através da simulação que a sintaxe em VHDL proposta pelo software "LOGO^{2}VHDL" descreve o sistema de controle de acordo com o esperado.

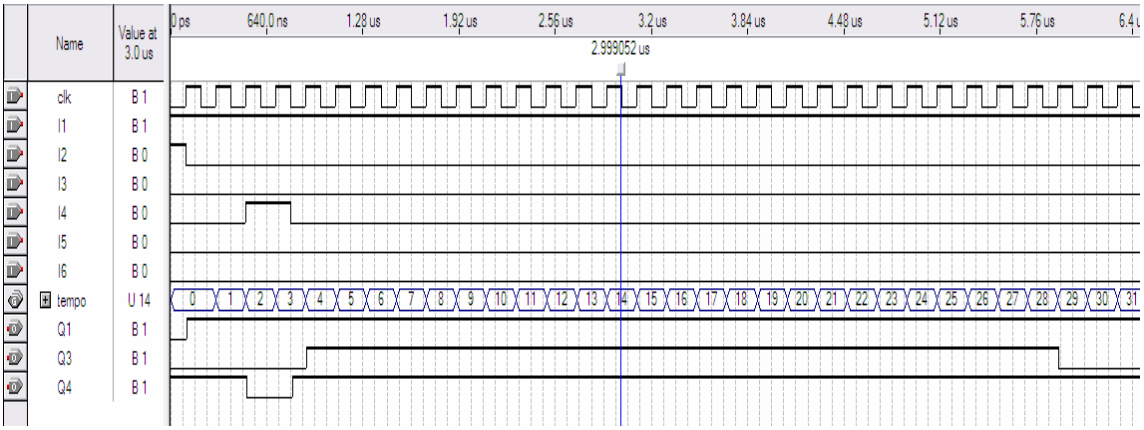


Figura 4.10 – Simulação do código VHDL do sistema de controle automático de preenchimento de silo

Na simulação verifica-se que o filtro (Q1) não é ativo quando o botão I1 não está habilitado ou quando I1 e I2 está habilitado simultaneamente, ou seja, a mangueira para transferência dos grãos não está corretamente conectada, através do acionamento de I2. Neste momento o sinal intermediário S4 funciona como a saída Q2, onde o mesmo é aberto, significando que a válvula de compressão para despejar o silo está acionado.

O interruptor I4 aciona a função retardo do desligamento e inicia o timer do contador, para o sistema ser interrompido em 25 pulsos de relógio após o desligamento deste botão.

Em I3, o sistema sinaliza quando o silo estiver cheio. Trata-se de uma válvula de finalização automática do processo, devendo se encontrar portanto desativado (nível lógico 0).

O alarme pode ser reajustado manualmente pelo acionamento da entrada I6, e se a mangueira não pôde ser esvaziada a tempo, em 30 pulsos de relógio um procedimento de emergência pode ser ativado acionando o botão I5.

No QUARTUS II, assim como qualquer outro sistema de síntese, é importante especificar a tecnologia para desenvolvimento. Inicialmente utilizou-se o componente da Família EPM7256 que dispõe e 257 células para integrar o sistema no Circuito Integrado.

Observou-se que era necessário especificar um outro componente para códigos VHDL que exigiam um número maior de células de acordo com as especificações do sistema de controle. Utilizou-se, portanto a família FLEX10KE, que apesar de ser mais lenta e volátil, dispõe de uma quantidade maior de silício.

Salienta-se que nos exemplos realizados foram utilizadas além das funções básicas, as funções especiais: retardamento de ligação, retardamento do desligamento e relé de auto-retenção.

Capítulo 5

Conclusões Gerais

5.1. Conclusões

Percebe-se que a utilização do hardware configurável, utilizando dispositivos lógicos programáveis vem se tornou-se uma tendência crescente, na área da robótica, síntese e atualmente em automação de sistemas de controle.

Verificou-se que um sistema de controle automatizado é comumente descrito em uma linguagem de programação, podendo ser do tipo estruturado, Ladder ou blocos lógicos. Essas três abordagens são ditas linguagens-padrão para automatizar sistemas de controle para quaisquer aplicações, sejam elas industriais, residenciais, comerciais dentre outras. Assim, verifica-se, com o trabalho apresentado, uma abordagem nova para automatizar sistemas de controle utilizando a linguagem de descrição de hardware – VHDL, que é uma linguagem, que descreve circuitos eletrônicos digitais e que também não é propriamente utilizado para automação de processos.

Esta nova metodologia visa utilizar dispositivos lógicos programados apoiados na linguagem de descrição de hardware, no caso VHDL, em caráter alternativo às metodologias normalmente usadas com controladores lógicos programáveis. A partir deste novo conceito de programação de sistemas de controle, surge a motivação para automatizar processos baseados em CLPs por dispositivos lógicos programáveis, que sejam dinamicamente reprogramáveis e que possuam estrutura adequada a implementação de algoritmos via hardware.

A princípio, pensou-se em decodificar os programas de automação, que eram descritos no ambiente de programação LOGO!Soft, porém, com o estudo do software, percebeu a indisponibilidade da linguagem estruturada (lista de instruções) no programa. De acordo com o fabricante, a linguagem STL (Instruction List), é uma linguagem possível apenas no Software Step 7, que programa o CLP S7-200/300/400 da Siemens. Verificou-se que não seria possível realizar a decodificação, devido o alto nível de codificação gerado pelo compilador do programa.

Tendo em vista o problema encontrado, desenvolveu-se um software em ambiente visual denominado LOGO^{2}VHDL que a partir da leitura de um esquemático descrito no

LOGO!Soft, torna-se possível gerar um arquivo correspondente em linguagem de descrição VHDL, utilizando esta ferramenta como intermediadora.

Desta forma, ao invés de decodificar os programas realizados no LOGO!Soft, foi desenvolvido uma linguagem de programação própria com interface gráfica que implementa as funções de automação, básicas e especiais, à partir do LOGO!Soft e posteriormente gera um arquivo descrito em sintaxe VHDL.

Uma vez concluído o ambiente gráfico, as funções do LOGO!Soft foram testadas, e conforme ilustra os exemplos dos sistemas de controles citados no trabalho, conclui-se que existe uma grande tendência desta metodologia, para o crescimento na área de automação, utilizando principalmente FPGA implementando processos, em máquinas computacionais dedicadas e que podem ser reprogramadas dinamicamente.

Sabe-se que os PLDs são circuitos integrados que podem ser configurados pelo próprio usuário podendo simplificar e acelerar o ciclo de um sistema automatizado. Fatores como este, contribui para o desenvolvimento de uma área promissora em aplicações de PLDs em substituição aos microprocessadores.

O trabalho vem, portanto fornecer uma nova ferramenta no processo de automação, bem como possibilitar a expansão da linguagem de programação de automação para um amplo grupo de programadores e projetistas que sabem programar em VHDL, no entanto, desconhecem as linguagens normalmente utilizadas para automação. Dessa mesma forma o trabalho poderá ser disponibilizado para todos os tipos de aplicações, principalmente àqueles em que o tempo de resposta seja rápido, e para processos onde o uso de dispositivos seqüenciais não seja indicado devido à natureza de seu processamento.

Sugere-se como trabalho futuro implementar as funções analógicas oferecidas pelo ambiente LOGO!Soft, em linguagem VHDL, bem como melhorar o software LOGO^{2}VHDL de forma gráfica, através da utilização de recursos *drag-and-drop,* para realizar as descrições dos sistemas de controle, de forma mais ilustrativa, em caráter de substituição aos formulários que foram propostos neste trabalho.

Uma outra abordagem interessante como trabalho futuro, seria o desenvolvimento de um software que pudesse decodificar os sistemas de controle descritos no LOGO!Soft sem precisar descrevê-lo em outro aplicativo. Um programa com este comportamento, possibilitaria ao projetista gerar o código VHDL correspondente de forma ainda mais automática que o proposto no trabalho.

O hardware configurável apresenta-se, como uma alternativa tecnológica, que pode adaptar-se a aplicação de sistemas de controle com a facilidade de um processador de

finalidade geral, enquanto mantém as vantagens de desempenho do hardware dedicado. Com essa grande velocidade e adaptabilidade, a computação reconfigurável tem um grande potencial a ser explorado, especialmente em aplicações que necessitam de alto desempenho como arquiteturas paralelas, aplicações de tempo real, dentre outras.

Referências

(01) RIBEIRO, J. M. S; GARCIA, J. P. F. **Apostila de controladores lógicos programáveis em processos industriais.** Ilha Solteira: Faculdade de Engenharia de Ilha Solteira, 2005. (Apostila).

(02) SOFTWARE LOGO! Soft Comfort da Siemens. Disponível em: **<http://www.automation.siemens.com/logo/index_76.html>**. Acesso em: 12 out. 2007.

(03) SIEMENS. **Manual de instruções do LOGO!** Edição 06/2003. disponível em: **<www.siemens.com>**. Acesso em: 15 mar. 2005.

(04) NATALE, F. **Automação industrial.** São Paulo: Érica. 2000.

(05) MECATRONICA ATUAL. **Automação industrial de processos e manufatura**. Disponível em: **<www.mecatronicaatual.com.br/edicoes.asp?comando=27_54&dettaglio=27>**. Acesso em: 18 abr. 2007.

(06) PEREIRA, M. H. R.; SILVA, A. C. R. **Acionamento de um motor de indução com o emprego de microcontrolador.** Trabalho de Formatura. (Engenharia Elétrica) – Faculdade de Engenharia de Ilha Solteira, Universidade Estadual Paulista, Ilha Solteira, 2004.

(07) BROWN, S.; ROSE, J.; FRANCIS, R.; VRANESIC, Z. **Field programmable gate arrays.** Boston: Kluwer Academic Publisher, 1997.

(08) HARDING, B. HDLs: a high-powered way to look at complex design. **Computer Design**, Boston, v. 29, n.10, p. 74-84, 1990.

(09) ELETRONICA.org. Computação reconfigurável aplicada a robótica. Disponível em: **<www2.eletronica.org/artigos/eletronica-digital/computacao-reconfiguravel-aplicada-a-robotica-fpga/view>**. Acesso em: 18 abr. 2007.

(10) LIMA, J.; CARDOSO, J. M. P. **Estudo e implementação de um controlador para câmara digital**. Disponível em: **<www.deei.fet.ualg.pt/projectos/controlador_JL1.pdf>**. Acesso em: 15 jul. 2007.

(11) WELCH, J. T. ; CARLETTA, J. **A direct mapping FPGA architecture for industrial process control applications**. The proceedings of the 2000 IEEE International

Conference on Computer Design: VLSI in Computers & Processors. Disponível em: **<http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/7044/18963/00878352.pdf>**. Acesso em 15 jul. 2007.

(12) MIYAZAWA, I. N.; FUKAGAWA, M. I.; SEKIGUCHI, T. KANAGAWA, I. **Implementation of ladder diagram for programmable controller using FPGA**. Disponível em: **<http://ieeexplore.iee.org/xpl/freeabs_all.jsp?arnumber=813150>**. Acesso em: 15 jul. 2007.

(13) MORAES, F. G.; AMORY, A. M.; PETRINI JÚNIOR, J. **Sistema integrado e multiplataforma para controle remoto de residências.** Porto Alegre: Pontífica Universidade Católica do Rio Grande do Sul, 2001.

(14) HAFNER, A.; LIMA C. R. E.; LOPES, H. S. **Implementação de um medidor de qualidade de energia usando computação reconfigurável por hardware**. Disponível em: **<www.cpgei.cefetpr.br/~hslopes/publicacoes/2005/sbai2005b.pdf>**. Acesso em: 15 jul. 2007.

(15) ORDONEZ, E. D. M.; PENTEADO, C. G.; SILVA, A. C. R. **Microcontroladores e FPGAs: aplicações em Automação**. São Paulo: Novatec. 2005

(16) COSTA, C. **Projetando Controladores Digitais com FPGA.** São Paulo: Novatec, 2006.

(17) YAMAZAKI, K. **A study on programmable logic controller by direct logic synthesis**. IMS – Mechatronics. University of Calofornia, Davis. Disponível em: **<http://ims.engr.ucdavis.edu/MemberPosterFile/QingWang.pdf>**. Acesso em: 15 mar. 2005.

(18) GEORGINI, M. **Automação aplicada: descrição e implementação de sistemas seqüenciais com PLCs**. São Paulo: Érica, 2000.

(19) LAWRENCE T. A. **Automation system for control and data acquisition.** USA: Instrument Society of America. 1992.

(20) ALTERA CORPORATION. Manual do MaxPlus: programmable logic development system. Disponível em: **<http://www.altera.com/literature/ds/dsmii.pdf>.** Acesso em 15 jul. 2007.

(21) BONFATTI, F.; MONARI, P. A.; SAMPIERI, U. **IEC1131-3 Programming Methodology - Software engineering methods for industrial automated systems.** France: CJ International, 1997.

(22) BRUCIAPAGLIA, A. H. ; PAGANO, D. J. **Instrumentação em controle.** Santa Catarina: Universidade Federal de Santa Catarina, 1994. (Apostila).

(23) CANTÚ, M. **Dominando o Delphi 5:** a Bíblia. São Paulo: Makron Books, 2000.

(24) CARVALHO, F. F. **Programação orientada a objetos usando o Delphi 3.** São Paulo: Érika, 1998.

(25) CHAN, P.; MOURAD, S. **Digital design using field programmable gate arrays.** Englewood Cliffs: Prentice Hall, 1994.

(26) CHANG, K. L. **Digital design and modeling with VHDL and synthesis**. Los Alamitos: IEEE Computer Society Press, 1997.

(27) CHONGASAMETHAWORN, S.; SUITJARRITTHAMMAKURN, S. **Design PLC using FPGA**. Disponível em: **<http://www.geocities.com/yanohc/hardware/cpufpga/index.html>**. Acesso em: 18 jul. 2006.

(28) CLEMENTS, K ; JEWERY, W. J. **The PLC workbook.** London: Prentice Hall, 1996.

(29) CRISPIN, A. J. **Programmable logic controllers and their engineering applications.** São Paulo: McGraw-Hill, 1990.

(30) ENGO, F. **Como programar em Delphi 3.** São Paulo: Makron Books, 1997.

(31) FCCM'98. Top 10 Preditcions for FCCMs in 2003. Disponível em: **<http://www.fccm.org/rellinks.php>**. Acesso em: 12 jul.2006.

(32) PERRY, D. L. **VHDL.** 2.ed. New York: MacGraw-Hill, 1993. (Series on Computer Engineering).

(33) FELIPE, E. R. **Delphi 5 - Fundamentos.** Belo Horizonte: SENAC, 2000.

(34) SILVEIRA, P. R.; SANTOS, W. E. **Automação e controle discreto.** 2ª. ed. São Paulo: Érica, 1998.

(35) TEIXEIRA, S; PACHECO, X. **Borland Delphi 6: guia do desenvolvedor**. Rio de Janeiro: Campus, 2002.