Departamento
de Engenharia Eletrotécnica

Desenvolvimento de Ferramenta para Interligação de Dispositivos Utilizando Protocolos Industriais

Trabalho de Projeto apresentado para a obtenção do grau de Mestre em Engenharia Eletrotécnica – Área de Especialização em Automação e Comunicações em Sistemas Industriais

Autor

Francisco Almeida Nunes Brito Dias

Orientador

Doutor Inácio Fonseca

Instituição

Instituto Superior de Engenharia de Coimbra

Coimbra, setembro, 2015

AGRADECIMENTOS

O trajeto efetuado na elaboração deste Trabalho de Projeto foi extenso e nem sempre linear, pelo que, no momento da sua apresentação este não teria chegado a bom porto, se não fossem as sugestões, orientações e os apoios que foram proporcionados e que aqui refiro.

Em primeiro lugar gostaria de agradecer ao meu orientador, Professor Doutor Inácio Fonseca, o apoio e orientação ao longo deste trabalho e a contribuição na revisão do relatório deste Trabalho de Projeto.

Aos meus amigos e colegas Ana Ramos, Bruno Vieira, Danilo Mendes e Mikael Santos pelo apoio, incentivo e coragem transmitidos nesta fase.

Aos meus amigos e colegas que me incentivaram, e permitiram desfrutar de bons momentos e disposição durante o percurso académico.

Por último, mas nunca menos importantes, aos meus pais, António de Brito Dias e Carmelinda Brito, e aos meus irmãos, André Brito Dias e João Brito Dias, pelo apoio, por acreditarem em mim, e por serem uma família que me permite atingir os meus objetivos pessoais.

A todas estas pessoas, o meu sincero obrigado.

Francisco Brito Dias

RESUMO

Devido ao crescimento da aplicação de automação no meio industrial, a competitividade dos vários setores industriais tem vindo a aumentar muito os seus níveis de oferta aos clientes.

A gestão de todos os processos dentro de uma indústria é um trabalho de elevada complexidade. Como tal, recorre-se à utilização de ferramentas e tecnologias para o auxílio na execução dos diversos processos, tais como aplicações lógicas, que permitem uma *interface* gráfica ao utilizador para acesso à informação relevante sobre o estado das entradas e saídas de controlo do processo. Neste aspeto, o controlo, a supervisão, e os autómatos permitem o controlo sobre diversos equipamentos que estejam ligados na instalação utilizando protocolos de comunicação para um fluxo estável de informação entre as várias camadas de rede.

Os protocolos de comunicação *Ethernet* Industrial são cada vez mais uma alternativa no âmbito da automação industrial. Este tipo de protocolos de comunicação estabeleceram-se de uma forma positiva no mercado devido às suas comunicações em tempo-real, apresentando-se como uma solução credível em relação aos protocolos de *interface* série devido às suas elevadas taxas de transmissão de dados entre os ativos físicos, à utilização de topologias de rede mais flexíveis para comunicação entre os ativos físicos e a um número de nós superior ligados à rede.

O presente Trabalho de Projeto aborda a comunicação entre ativos físicos industriais, que envolve um PLC *(Programmable Logic Controller)*, uma consola interativa HMI (*Human Machine Interface*), e sistemas de microcontroladores. Foi desenvolvida uma biblioteca para implementação do protocolo FINS (*Factory Interface Network Service*) para comunicação entre os ativos físicos e terminais via *Ethernet*, sendo esta biblioteca para ambientes UNIX (Linux).

A biblioteca foi usada como controlador de unidades I/O distribuídas em Arduíno e PLC OMRON, e visualização em consola HMI.

Palavras-chave: PLC OMRON; Protocolo FINS; Ethernet; Consola HMI; Arduíno.

ABSTRACT

Given the growth of the application of automation in the industrial world, the competition among the many industrial divisions have been increasing the offer to their clients.

The management of all processes inside an industry is a highly complex piece of work. Thereby, tools and technologies are used to help in the execution of diverse processes, such as logical applications, which allow the user to have a graphic interface in order to access relevant information concerning the state of the inputs and outputs of the process control. Under this subject, the control, supervision, and automation allow the control of the several devices that are connected among themselves using communication protocols in order to have a stable data flow amongst the various network layers.

The Industrial Ethernet communication protocols are increasingly seen as an alternative in the ambit of industrial automation. This type of communication protocols have established themselves in a positive way on the market due to their real-time communication ability, presenting themselves as a believable solution regarding interface protocols in series given their high data transmission ratios between the physical assets, the usage of more flexible network topologies for communication between the physical assets and a number of superior nodes connected to the network.

This project addresses the communication between active industrial physical equipment, which involve a PLC (*Programmable Logic Controller)*, an interactive HMI (*Human Machine Interface*) console, and microcontroller systems. A library was developed to apply the FINS (*Factory Interface Network Service*) protocol so that the communication between the physical actives and terminals via Ethernet could occur, being this library for UNIX (Linux) software.

The library was used as an I/O unit controller distributed in Arduino and PLC OMRON, and the visualization is in an HMI console.

Key-words: PLC OMRON; FINS Protocol; Ethernet; HMI Console; Arduino.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

SIMBOLOGIA E ABREVIATURAS

CAMAC - *Computer Automated Measuremente and Control*

CAN - *Controller Area Network*

CIP - *Common Industrial Protocol*

CIM - *Computer Integrated Manufacturing*

CN - *Controlled Node*

DA1 - *Destination Node Address*

DA2 - *Destination Unit Address*

DCS – *Distributed Control System*

DNA - *Destination Network Address*

EtherCAT - *Ethernet for Control Automation Technology*

FINS – *Factory Interface Network Service*

FIP - *Factory Instrumentation Protocol*

GCT - *Gateway Count*

GPIB - *General Purpose Interface Bus*

Grafcet - *Graphe Fonctionnel de Command Etape Transition*

GCC - *GNU Compiler Collection*

HMI – *Human Machine Interface*

HTML - *HyperText Markup Language*

ICF – *Information Control Field*

IEC – *International Electrotechnical Commission*

IEEE - *Institute of Electrical and Electronics Engineers*

IP – *Internet Protocol*

ISO - *International Organization for Standardization*

LAN – *Local Area Network*

LD – *Ladder Diagram*

MAC - *Media Access Control*

MAP - *Manufacturing Automation Protocol*

MIL STD 1553 - *Military Standard 1553*

MN - *Managing Node*

NBS - *National Bureau of Standard*

ODVA - *Open DeviceNet Vendor Association*

OSI - *Open Systems Interconnection*

PCS - *Process Control System*

PLC - *Programmable Logic Controller*

PNO - *Profibus User Organization*

PROFIBUS - *PROcess Field BUS*

RSV - *Reserved by system*

SAP - *Service Access Point*

SA1 - *Source Node Address*

SA2 - *Source Unit Address*

SCADA - *Supervisory Control And Data Acquisition*

SFC - *Sequential Function Chart*

SID - *Service ID*

SMTP - *Simple Mail Transfer Protocol*

SNA - *Source Network Address*

SoC - *Start of Cycle Frame*

ST - *Structrured Text*

TCP – *Transmission Control Protocol*

TOP - *Technical and Office Protocol*

UDP – *User Datagram Protocol*

1 Introdução

1.1 Motivação e Contexto

Nos últimos anos, o aumento da capacidade de processamento e relação custo-eficácia dos sistemas eletrónicos tem causado um grande impacto nas empresas, influenciando os negócios das mesmas de modo a gerar um aumento de produção, redução de custos, melhoria da qualidade, segurança, e monitorização remota [1]. Este desempenho é possível nos dias de hoje devido a uma tecnologia bem conhecida, a automação. Segundo James Powell [4], a automação descreve a utilização da tecnologia para o controlo de sistemas através de um conjunto de eventos e de pouca ou nenhuma assistência humana quando em funcionamento. Genericamente existem duas categorias de automação: Automação de Processo e Automação de Fábrica.

A automação de processo menciona a automação para o uso de produção de bens através de uma fórmula ou sequência de eventos. A automação de fábrica refere-se à automação para o fabrico de objetos discretos, como é o caso de um carro ou de um computador.

Em muitas indústrias chega-se, por vezes a fazer uma combinação destes dois géneros de automação, como se observa na figura 1.

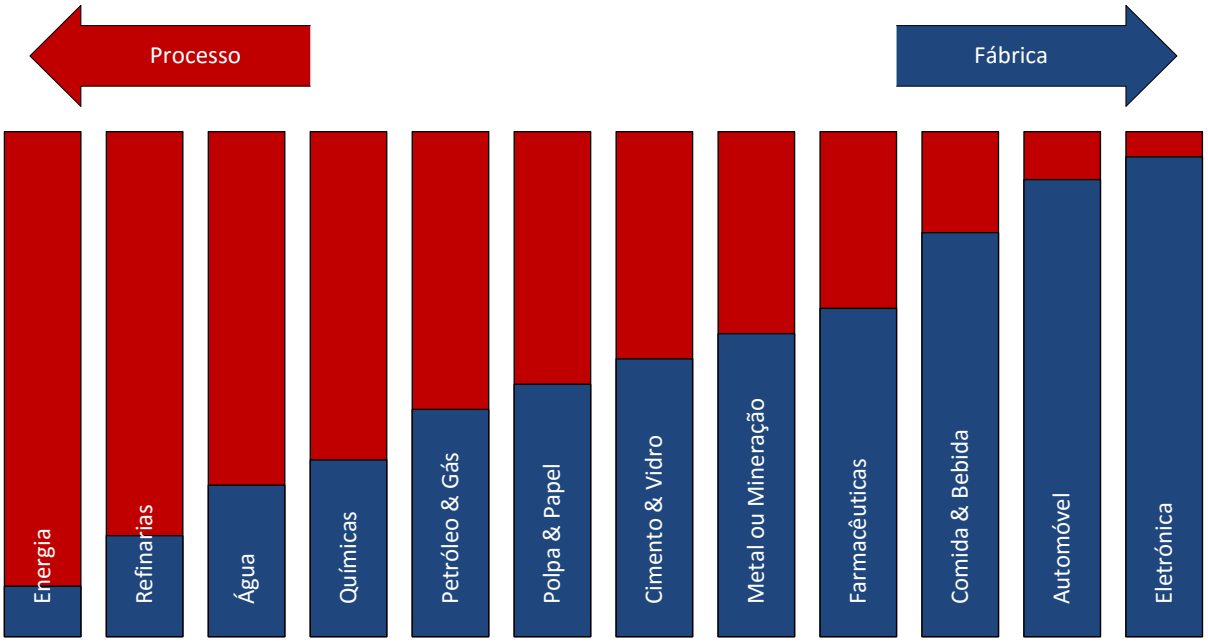


Figura 1 - Aplicação das automações nas diferentes indústrias; adaptado de [4].

Um dos principais responsáveis pela evolução da automação na indústria foi a inserção das redes industriais nas empresas. No início do século XX, os sistemas de controlo e os processos de fabrico nas empresas eram realizados através da tecnologia mecânica e equipamentos analógicos [3]. Tecnologias, tais como o controlo pneumático e a energia hidráulica tornaram possível o controlo de sistemas remotos através de um sistema de controlo centralizado.

Com a evolução dos equipamentos eletrónicos e o aumento da popularidade dos mesmos nos anos 60, os sistemas de controlo mecânico foram substituídos por sistemas de controlo eletrónico, que recorriam à utilização de transdutores, relés e circuitos de controlo com fio. Estes sistemas automatizavam espaços de grandes dimensões, sendo por vezes necessário quilómetros de cabo para permitir a ligação entre os equipamentos de campo (transdutores e atuadores) e os circuitos de controlo [2].

Em meados de 1970, o primeiro DCS (*Distributed Control System*) foi anunciado pela Honeywell como um sistema de controlo hierárquico com um largo número de microprocessadores [3], sendo este aplicado por muitos sistemas de automação industrial, tais como sistemas de fabrico automóvel, sistemas de controlo de potência, refinarias de petróleo e gás, entre outros. Deste modo, a funcionalidade de múltiplos sistemas analógicos podia ser desempenhada por um único controlador digital [2], com ligação aos dispositivos de entrada e de saída em modo ponto-a-ponto. Esta era a implementação mais comum de sistema de controlo naquele tempo [5].

Com a substituição dos controladores analógicos pelos controladores digitais e a evolução da velocidade de processamento dos computadores, em 1980 tornou-se popular a ligação entre computadores e dispositivos de automação, com recurso à LAN (*Local Area Network*). Este modelo de comunicação permitiu que os sistemas de automação sejam implementados com arquiteturas abertas distribuídas através da comunicação sobre redes de comunicação digitais [3], possibilitando a comunicação dos utilizadores de uma LAN com equipamentos ligados noutra LAN através de *gateways*.

Esta evolução na comunicação gerou um aumento do tamanho das redes e do número de dispositivos ligados às mesmas, o que levou à necessidade de implementar redes e protocolos de comunicação que permitissem a comunicação dos dados entre os dispositivos de campo e os controladores. Contudo, as soluções LAN não se apresentavam como as mais adequadas para a ligação dos controladores aos dispositivos de campo devido ao grande investimento que representavam e à carência de resposta em tempo real [3].

Assim sendo, foi desenvolvido um padrão específico para a comunicação entre os controladores industriais e os equipamentos de campo, o *Fieldbus*.

O *Fieldbus* surgiu com o propósito de substituir os esquemas de ligação em estrela ponto a ponto de modo a ligar controladores, sensores e atuadores através de um único sistema de barramento. De facto, este conceito de rede forneceu bastantes benefícios, nos quais se destacam o aumento de flexibilidade e modulação de instalações, aumento de inteligência nos dispositivos, facilidade na configuração do sistema e melhoria no comissionamento e manutenção [6].

Recentemente, os sistemas de controlo digital começaram a integrar redes a todos os níveis do controlo digital, ou seja, interligação de redes de negócio com equipamento industrial através da utilização de padrões *Ethernet*. Isto resultou num ambiente de rede que é semelhante às redes convencionais na camada física, mas com características significativamente diferentes [2].

1.2 Objetivos

O principal objetivo deste Trabalho de Projeto de mestrado corresponde ao desenvolvimento de comunicações entre equipamentos industriais e sistemas de microcontroladores. O projeto consiste no desenvolvimento de uma biblioteca baseada no protocolo FINS (*Factory Interface Network Service*) para comunicação entre os diversos dispositivos e terminais via uma rede *Ethernet*.

Inicialmente foi preciso efetuar um estudo sobre os vários protocolos industriais aplicáveis ao trabalho e que existem atualmente no mercado.

Este estudo culminou no desenvolvimento de uma biblioteca de comunicações para ambiente UNIX (Linux) *multi-threading*, sendo o núcleo escrito em linguagem C. Esta biblioteca requereu aquisição de dados em *sockets*, protocolos, e *threads* de modo a implementar sistemas cliente-servidor.

Foi igualmente necessário proceder ao estudo e programação de diversos dispositivos, implementar comunicações entre os mesmos, sendo estes dispositivos um autómato OMRON, uma Consola HMI (*Human Machine Interface*) OMRON e um Arduino Mega.

Por fim, o projeto foi concluído com a elaboração desta monografia.

1.3 Organização da monografia

Este Trabalho de Projeto está dividido em oito capítulos, tal como se sintetiza seguidamente:

- O primeiro capítulo contém a introdução ao Trabalho de Projeto, a contextualização, os objetivos, as metas a atingir e a organização da monografia;
- O segundo capítulo contém o Estado de Arte relativamente ao desenvolvimento do *Fieldbus* e a sua evolução histórica ao longo dos anos;
- O terceiro capítulo apresenta uma perspetiva do fluxo de informação dentro de uma empresa industrial, sendo apresentadas as diversas topologias de rede, os protocolos *Ethernet* Industrial mais utilizados e o protocolo utilizado neste trabalho, o protocolo FINS;
- O quarto capítulo apresenta os equipamentos OMRON utilizados neste trabalho, e as respetivas configurações de comunicação para trabalhar com o protocolo FINS;
- O quinto capítulo descreve a plataforma desenvolvida em ambiente Unix, para comunicação com os diversos equipamentos via protocolo FINS;
- O sexto capítulo descreve a plataforma eletrónica Arduíno utilizado neste trabalho, e a programação realizada no mesmo;
- O sétimo capítulo descreve o funcionamento entre os diversos capítulos realizados neste trabalho e a interação entre os mesmos;
- O oitavo capítulo apresenta as conclusões e trabalhos futuros que poderão ser desenvolvidos;
- O final deste Trabalho de Projeto é constituído pelas referências bibliográficas e os vários anexos mencionados ao longo dos capítulos anteriores.

2 Estado de Arte

Neste capítulo é apresentado o conceito do *Fieldbus*, sendo feita uma breve análise deste tipo de rede, através das suas origens históricas e as tecnologias das quais provém, assim como a evolução presenciada no mesmo conceito com o passar dos anos.

2.1 *Fieldbus*

Fazendo uma retrospetiva, é possível constatar que o *Fieldbus* foi um dos principais responsáveis pela evolução dos sistemas de automação.

De acordo com Richard Zurawski [6], a IEC (*International Electrotechnical Commission*) 61158 define o padrão *Fieldbus* como: "um barramento de dados digital, série, multiponto para comunicações com controlo e dispositivos de instrumentação industriais tais como – mas não apenas – transdutores, atuadores e controladores." Entretanto Nitaigour Mahalik [5] apresenta uma explicação mais detalhada sobre este tipo de sistemas: "O *Fieldbus* é uma rede de comunicações série, digital, bidirecional que interliga vários instrumentos, transdutores, controladores, elementos finais de controlo, analisadores de processos correntes, SCADA (*Supervisory Control And Data Acquisition*) e sistemas de controlo computorizados." O conceito de *Fieldbus* foi desenvolvido com o objetivo de substituir os sistemas de controlo centralizado de sinal analógico 4-20 mA por um sistema de controlo global distribuído [5].

Segundo John Park [8], existem bastantes benefícios na implementação de *Fieldbus*, tais como:

- Custos reduzidos em cablagem e terminais;
- Tempo de instalação e início de trabalho reduzido;
- Aumento da inteligência dos dispositivos;
- Aumento da interoperabilidade entre fabricantes.

A figura 2 demonstra um exemplo de comparação entre o sistema 4-20mA e o sistema *Fieldbus*.

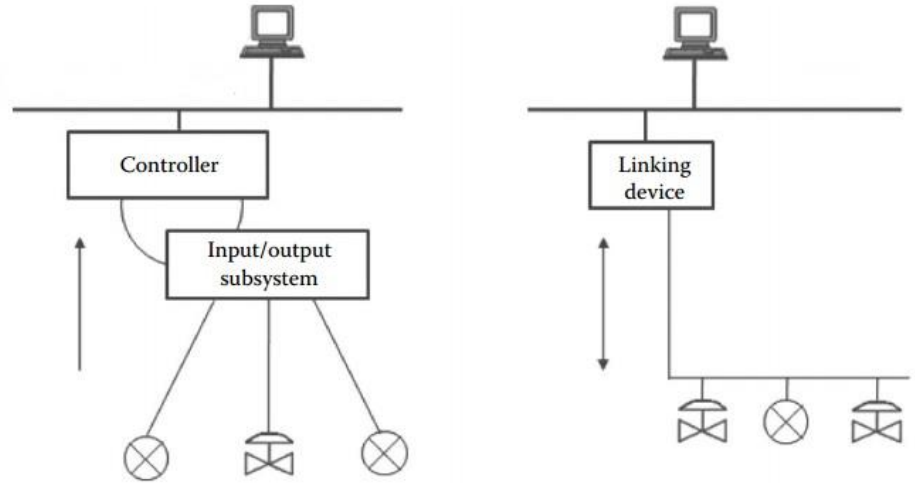


Figura 2 - Sistema 4-20mA vs Sistema *Fieldbus*; adaptado de [7].

2.1.1 Origens dos Sistemas *Fieldbus*

Apesar do termo *Fieldbus* ter surgido há cerca de 30 anos, este conceito de rede é muito mais antigo. As raízes da tecnologia *Fieldbus* moderna é uma combinação de engenharia eletrotécnica clássica com ciência de computação [6], [9].

Uma das bases da automação na transferência de dados é vista nas redes clássicas *telex* e também nos padrões de transmissões de dados por linhas telefónicas. Muitos dos padrões utilizados antigamente continuam a existir ainda hoje, como é o caso do V.21 (transmissão de dados através de linhas telefónicas) e X.21 (transmissão de dados através de linhas especiais). Naquele tempo, muitos protocolos foram definidos de certa forma devido à capacidade limitada de computação dos dispositivos daquela geração. Através da evolução dos microprocessadores e dos sistemas telefónicos, foi feita uma mudança gradual do sistema analógico para o sistema digital. Esta mudança proporcionou a oportunidade de transferir grandes quantidades de dados de um ponto de origem para um ponto de destino. Adicionando esta melhoria na transmissão a uma camada física aperfeiçoada, surgiram os primeiros protocolos para transmissão de dados, como é o caso do X.25 ou do SS7 [9].

Em simultâneo com o desenvolvimento nas telecomunicações, engenheiros de equipamento definiram *interfaces* para sistemas de computadores autónomos para efetuar ligações a dispositivos periféricos. A ideia essencial de ter *interfaces* padronizadas para dispositivos externos foi rapidamente alargada a equipamentos de controlo de processos e instrumentação. O problema a ser resolvido neste contexto seria a sincronização dos dados medidos dos diversos dispositivos distribuídos. Isso levou ao desenvolvimento de normas como o CAMAC

(*Computer Automated Measuremente and Control*) e o GPIB (*General Purpose Interface Bus*) também conhecido como IEEE (*Institute of Electrical and Electronics Engineers*) 488. Devido ao facto da velocidade limitada de processamento de dados e o requerimento de tempo real para a sincronização, estes sistemas de barramento tinham linhas de dados e controlo em paralelo, o que não é uma caraterística dos sistemas *Fieldbus*. No entanto, este tipo de barramentos já utilizava a estrutura típica multiponto.

Mais tarde, as ligações série dos periféricos dos computadores baseadas em RS-232 foram alargadas de modo a suportarem longas distâncias, possibilitando o entendimento multiponto.

A possibilidade de usar uma estrutura de barramento unicamente com duas ligações juntamente com a imunidade ao ruído fez do RS-485 um pilar para a tecnologia *Fieldbus* até aos dias de hoje.

2.1.2 Evolução dos Sistemas *Fieldbus*

A entrada dos sistemas *Fieldbus* nos assuntos da automação está relacionada com as tentativas de disponibilizar dados em todos os níveis de rede duma empresa. Um dos aspetos essenciais nesta estrutura foi a chamada "pirâmide de automação". Este modelo hierárquico foi concebido para estruturar o fluxo de dados em automação de fábrica e processo através da tecnologia de computação. O pensamento era desenvolver uma rede de vários níveis, transparente, que fosse capaz de acompanhar todas as etapas de um produto desde o seu projeto até ao seu fabrico. Este conceito é conhecido por CIM (*Computer Integrated Manufacturing*) e foi desenvolvido pela NBS (*National Bureau of Standard*) nos Estados Unidos da América [6], [11]. Tipicamente, este modelo é composto por cinco níveis. No entanto, as redes no nível superior da pirâmide já eram utilizadas quando esta foi definida, estando o nível de campo ainda com tecnologia ponto a ponto. Por isso, os sistemas *Fieldbus* foram igualmente desenvolvidos para mudar este nível. Esta pirâmide pode ser observada na figura 3.

Os *Fieldbuses* ocupam dois níveis nesta pirâmide: nível de campo e o nível de processo/célula. O nível de campo é composto por barramentos para ligação de dispositivos, tais como sensores e atuadores que fazem ligação com dispositivos da camada superior como, por exemplo os PLC (*Programmable Logic Controller*). No nível de célula o *Fieldbus* realiza as ligações entre os dispositivos de controlo, como por exemplo PLC e computadores.

A evolução destes sistemas teve como contributo chave a introdução do modelo OSI (*Open Systems Interconnection*) pela ISO (*International Organization for Standardization*) em 1978 [2], [6], [9], [11]. O modelo de sete camadas era e continua a ser uma referência para estes sistemas, tendo sido o ponto de partida para o desenvolvimento de muitos protocolos de comunicação complexos.

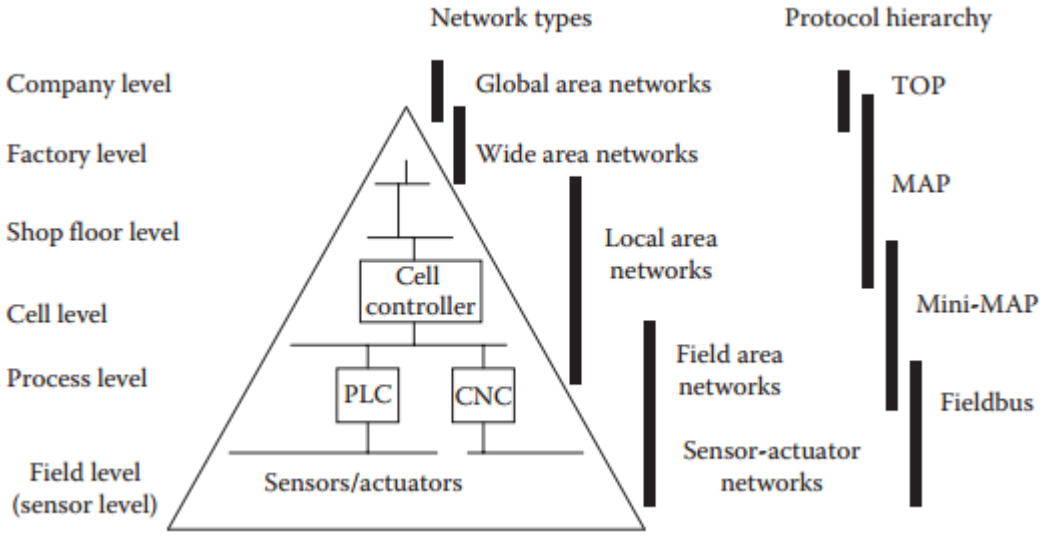


Figura 3 – Hierarquia de rede em automação e protocolos originais [6].

As camadas superiores na pirâmide de automação eram geridas com base em três protocolos: TOP (*Technical and Office Protocol*), MAP (*Manufacturing Automation Protocol*) e Mini-MAP.

O TOP foi definido pela empresa Boeing, e o MAP pela General Motors. O TOP tinha como objetivo a comunicação entre os serviços de negócio e os serviços técnicos estando situado entre o nível de empresa e de fábrica. O MAP tinha a tarefa de comunicar com os escritórios e fábricas, sendo idealizado inicialmente como uma estrutura de controlo de processos industriais a todos os níveis. Devido à complexidade e elevados custos do MAP, foi implementado o Mini-MAP para a comunicação entre controladores e máquinas. O MAP fica situado entre o nível de fábrica e o nível de célula, enquanto o Mini-MAP divide-se entre o nível de célula e de processo [11]. Independente das evoluções verificadas ao nível da ciência de computação, nos anos 70 e 80 houve um progresso significativo na microeletrónica que trouxe diferentes controladores integrados, e novas *interfaces* para interligar circuitos integrados de uma forma eficiente e barata. Na indústria da aviação e espacial, dada a necessidade de reduzir os cabos em tamanho e peso, foi desenvolvido o barramento MIL STD 1553 (*Military Standard 1553*), sendo

considerado ainda hoje o "primeiro" *Fieldbus*. Lançado em 1970, apresentava muitas propriedades dos sistemas *Fieldbus* modernos. Mais tarde, outras indústrias começaram aplicar este conceito, focando-se essencialmente nas duas últimas camadas do modelo OSI. Um exemplo clássico deste tipo de *Fieldbus* é a rede CAN (*Controller Area Network*). Em meados de 1980 presenciou-se uma grande evolução tecnológica na automação, principalmente com a implementação dos PLCS e dos dispositivos inteligentes [6]. A necessidade de redução da cablagem e o amadurecimento da microeletrónica serviram de base para o desenvolvimento de novos sistemas de comunicação. Foi igualmente o espaço de tempo em que o *Fieldbus* se afirmou como termo individual.

Neste espaço de tempo surgiram muitos sistemas *Fieldbus*, visto que muitas empresas de automação criavam o seu próprio *Fieldbus* para diferentes campos de aplicação. No entanto, muitos destes *Fieldbuses* não foram bem-sucedidos devido aos custos que implicavam. Na figura 4 é apresentado a linha de evolução dos sistemas *Fieldbus* e o ambiente em que eram aplicados.

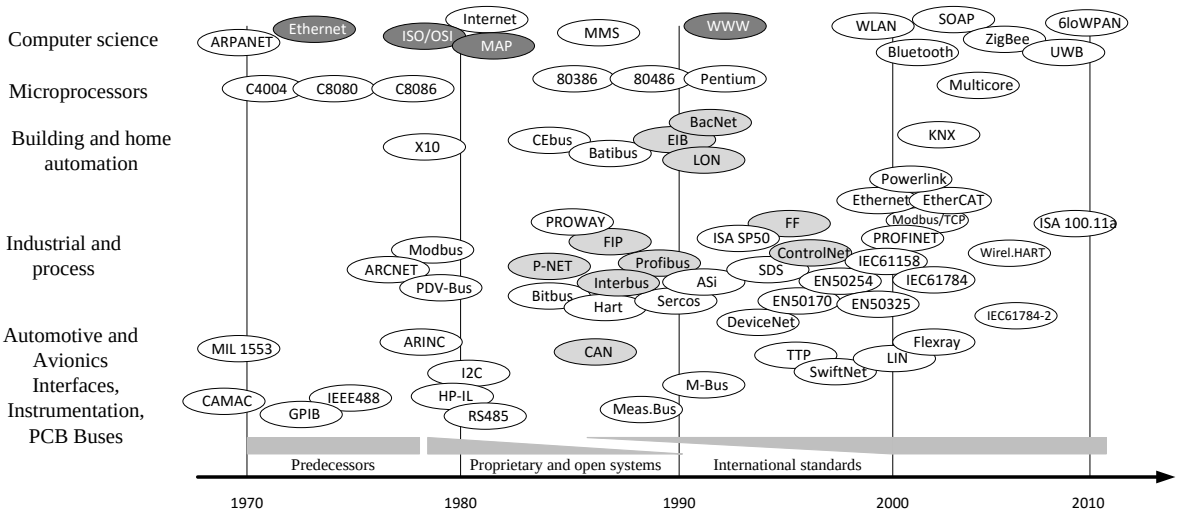


Figura 4 - Evolução dos *Fieldbuses*; adaptado de [6].

Depois de alguns anos de conflito e confusão tanto do lado dos fabricantes como do lado dos utilizadores, ficou visível que os únicos que tinham hipóteses de singrar no mercado eram os sistemas abertos. Os utilizadores fundaram organizações de modo a promover e definir os sistemas que eram independentes das empresas individuais. Este foi um dos primeiros avanços para o conceito *Fieldbus* [6]. O último passo para estabelecer o *Fieldbus* no mundo da automação foi a sua normalização internacional. O facto de o sistema estar normalizado

estabelece uma especificação formal e rígida, permitindo mudanças rápidas no mesmo, transmitindo confiança e credibilidade ao utilizador, assegurando uma posição segura no mercado. Reconhecendo esta necessidade, em 1985 com os esforços do IEC no subcomité técnico SC65C iniciou-se o projeto *Fieldbus* [6], que tinha o objetivo de criar uma norma *Fieldbus* universal que pudesse ser aplicada na automação de processo e de fábrica, com base em dois sistemas, nomeadamente o PROFIBUS (*PROcess Field BUS*) e o FIP (*Factory Instrumentation Protocol*). Contudo, os grandes investimentos já realizados em sistemas existentes e comprovados, juntamente com os interesses económicos de diversos países e empresas, a par das restrições e exigências, impediram o alcance da norma *Fieldbus* única. Por isso, ao fim de 14 anos de discordâncias políticas e técnicas, o objetivo original foi posto de lado com a implementação das normas IEC 61158 e IEC 61784 [9]. Outros sistemas foram entretanto desenvolvidos e normalizados, de tal modo, que o mundo *Fieldbus* hoje em dia consiste num conjunto de normas bem estabelecidas.

3 Arquitetura de Rede Industrial

3.1 Enquadramento

Atualmente, na indústria existem diversas empresas onde a automação é aplicada de modo a aumentar o desempenho nas suas tarefas, de forma alcançar as metas estabelecidas. Assim sendo, torna-se necessário que as empresas nesta área contenham uma estrutura de rede que permita a comunicação entre os diversos departamentos e o controlo e monitorização das atividades dos mesmos.

Este capítulo apresenta uma perspetiva do fluxo de informação dentro de uma empresa industrial, sendo demonstrado o exemplo de uma arquitetura de rede passível de implementação. São descritas algumas tecnologias e protocolos *Ethernet* Industrial que podem ser aplicados na mesma, sendo apresentado igualmente o protocolo estudado para a realização deste trabalho. Para mais informações acerca dos protocolos que interagem com os protocolos *Ethernet* Industrial e redes *Ethernet*, ver anexo A.

3.2 Arquitetura de Rede

Na indústria, seja esta de fabrico ou de processo, a rede industrial é estruturada em vários níveis hierárquicos, permitindo o controlo e o fluxo de informação ou dados desde o nível de campo até ao nível de empresa. Esta hierarquia pode ser estruturada em cinco níveis:

i. Nível de campo (sensores e atuadores);
ii. Nível de entrada e saída;
iii. Nível de controlo;
iv. Nível de gestão ou HMI (*Human-Machine Interface*);
v. Nível empresa [10].

A figura 5 apresenta um exemplo de uma hierarquia de redes nas empresas de ambiente industrial.

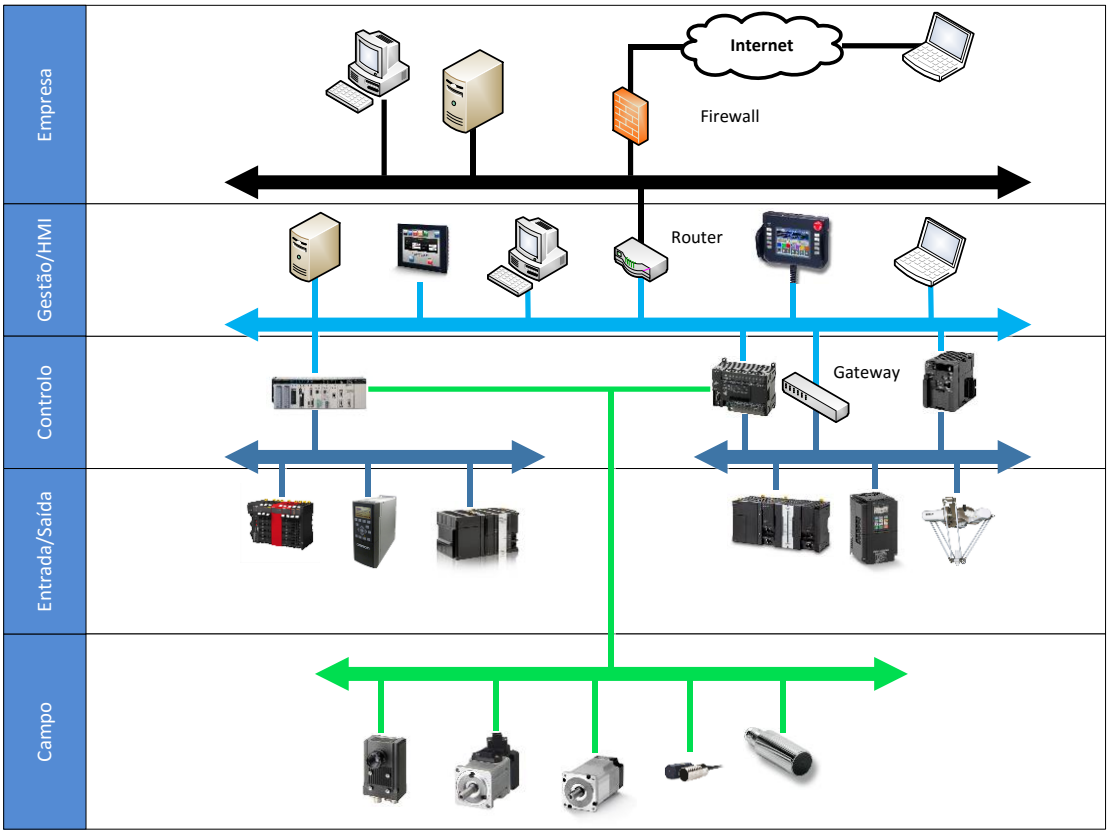


Figura 5 - Hierarquia nas comunicações industriais; adaptado de [19].

3.2.1 Níveis Hierárquicos

Segundo Béla Lipták [10], os níveis hierárquicos têm as seguintes tarefas:

- **Nível de campo**: abrange os atuadores e sensores que estão instalados, por exemplo em reservatórios, tanques ou linhas de produção. Os sensores fornecem as variáveis de processo, ou seja, a temperatura, nível, pressão, fluxo, entre outros e transferem os sinais para o nível de entrada e saída. Os atuadores recebem sinais do nível de entrada e saída e realizam uma tarefa com base no valor desse sinal;
- **Nível de entrada e saída**: A função principal do nível de entrada e saída é empacotar juntos os sinais de entrada e saída. Os sinais dos sensores são encaminhados para os controladores, e mediante a informação destes, os controladores enviam sinais para os atuadores;
- **Nível de controlo**: processa os sinais vindos do campo e gera os comandos para os atuadores. Normalmente, os controladores responsáveis por este processo sãos os PLC (*Programmable Logic Controller*) e os PCS (*Process Control System*);

- **Nível de gestão/HMI**: é responsável pela coleção dos dados recebidos do nível de controlo. Ferramentas de rede, fórmulas, gestão de ativos, documentação e programas de manutenção residem neste nível. Os operadores têm acesso à informação através do esquema de supervisão que é apresentado em monitor, sendo possível observar as variáveis de processo de qualquer parte da rede, sendo avisados quando um parâmetro ultrapassa o seu limite, podendo mudar a configuração de um dispositivo caso seja necessário. Este tipo de funcionalidade é proporcionado por programas SCADA (*Supervisory Control And Data Aquisition*), que podem operar em consolas ou em sistemas de controlo. Outras funções deste nível são o registro de sinais, acompanhamento de intervenções de operadores, e registo histórico de alarmes e eventos;
- **Nível empresa**: é a camada onde a informação ingressa para o mundo da gestão, ajudando na encomenda e faturação através de SAP (*Service Access Point*) ou no planeamento de produção. Esta é a camada de gestão do fornecimento em cadeia que intercala e avalia a informação, tais como as quantidades de material bruto em loja, a quantidade de água usada diariamente, a energia usada na refrigeração e aquecimento, as quantidades de produtos em fabrico e a quantidade que está disponível para ser vendida.

3.2.2 Topologias de Rede

A forma como os equipamentos são ligados de modo a conceber uma rede é conhecida como topologia [17]. Este conceito define o esboço físico da rede, especificando como os elementos são ligados uns aos outros eletricamente.

As topologias mais utilizadas são a estrela, anel, barramento e árvore.

A topologia em estrela consiste na ligação dos nós de rede a um ponto central, utilizando uma ligação ponto-a-ponto [12]. Normalmente, este tipo de redes utiliza um *switch* para a ligação entre dispositivos, como é demonstrado na figura 6 de forma radial em relação a esse ponto central [13].

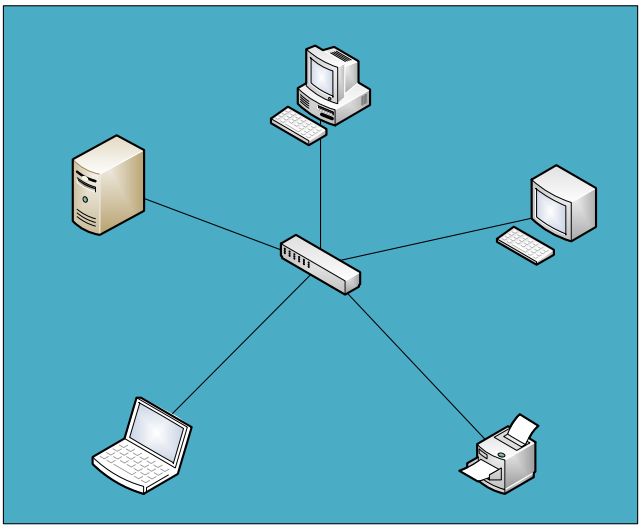


Figura 6 - Topologia em Estrela.

A topologia em anel é uma rede que se encontra ligada em formato circular, onde os dados são transmitidos num sentido único [12]. Nesta topologia, é importante que todos os equipamentos permaneçam ativos, de modo a permitir que os dados cheguem ao destino. A figura 7 apresenta esta topologia.

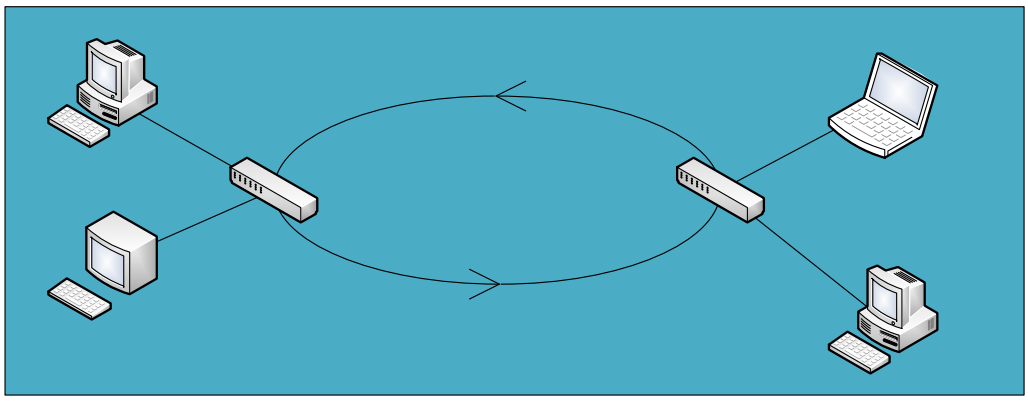


Figura 7 - Topologia em Anel.

A topologia em barramento, também conhecida como topologia em linha, é uma topologia onde os equipamentos são ligados a uma linha principal ou *trunk* [12], sendo este um barramento linear formado por vários pontos de acesso interligados [13]. Na figura 8 é possível observar esta topologia.

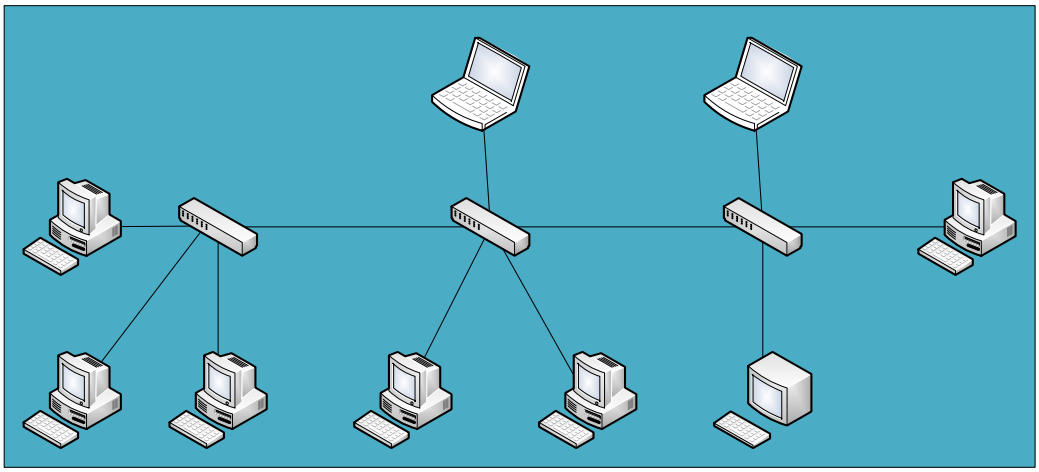


Figura 8 - Topologia em Barramento.

A topologia em árvore é formada por vários níveis hierárquicos, assumindo uma estrutura em forma de árvore com diversos níveis. Esta topologia resulta da interligação entre várias topologias em estrela, conforme observado na figura 9.

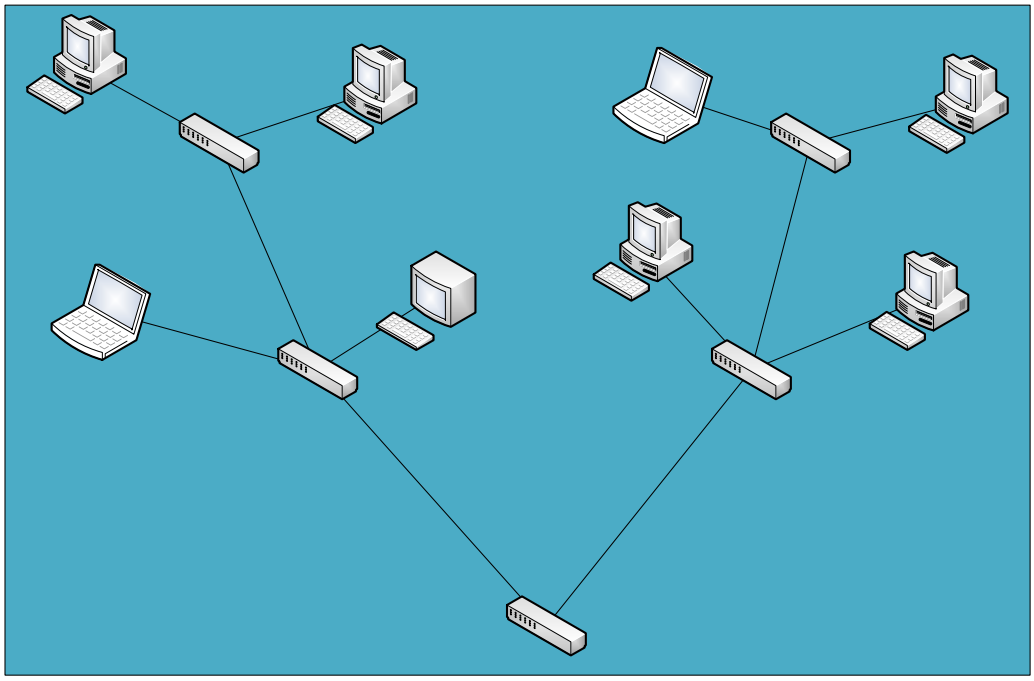


Figura 9 - Topologia em Árvore.

3.2.3 Componentes de Automação Industrial

Segundo Zhihong Lin [14], na automação industrial existe um conjunto de componentes que são essenciais para o desenvolvimento desta atividade na indústria: PLCs, painéis HMI, sensores e unidades industriais.

O PLC é um sistema de controlo utilizado na automação industrial que proporciona controlo de movimento, controlo de entradas e saídas de processos, sistemas distribuídos e controlo de rede. Este equipamento é utilizado frequentemente em condições ambientais adversas, onde é necessário um controlo preciso, em tempo-real e determinístico através de comunicações fidedignas. Na figura 10 é possível ver um destes aparelhos do fabricante OMRON.

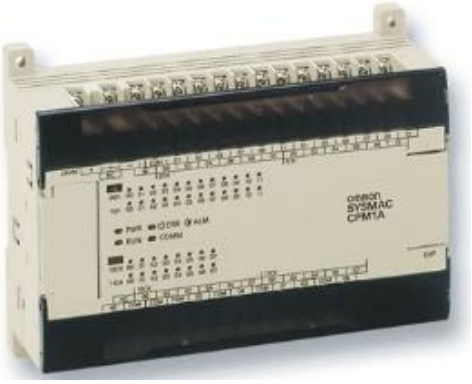


Figura 10 - PLC OMRON.

O HMI é uma interface gráfica de utilizador para controlo industrial [14], que permite a visualização, controlo, diagnóstico e gestão de processos [15]. Este equipamento (por exemplo, o ilustrado na Figura 11) está interligado com diversos sistemas industriais através de comunicações fidedignas [14].

Figura 11 - Painel HMI OMRON.

O sensor é um equipamento que fornece a entrada de dados para sistemas de controlo [16], sendo essencial para operações de monitorização, inspeções, medidas, entre outras tarefas que são realizadas em tempo-real (ver Figura 12) [14].

Figura 12 - Sensor OMRON.

As unidades industriais variadores, também conhecidas como *industrial drives*, são controladores de motores utilizados para operações de controlo otimizado de motores. Este tipo de equipamentos são usados numa grande variedade de aplicações industriais, podendo trabalhar com potências e tensões elevadas. Através dos mesmos, é possível monitorizar o comportamento dos motores e ajustar a *performance* do mesmo mediante a tarefa a realizar (ver Figura 13) [14].

Figura 13 - Exemplo de *industrial drive* OMRON.

3.2.4 Equipamentos de Interligação de Redes

A interligação das diversas redes dentro da arquitetura de rede industrial é um aspeto importante, visto que é necessário efetuar o controlo de todos os elementos que se encontram

ligados na mesma. Segundo Edmundo Monteiro [13], os equipamentos de interligação de redes permitem a ligação de sistemas terminais à rede, interligação de diversas porções de rede e interligação de redes distintas. Os equipamentos de interligação de redes utilizados na troca de dados entre redes são os *repeaters*, *hubs*, *bridges*, *switchs*, *routers*.

O *repeater* (repetidor) é um dispositivo cuja função é de intensificar ou melhorar os sinais físicos que recebem de um segmento de rede que interligam, retransmitindo os sinais para outro segmento. Este dispositivo é bidirecional, sem qualquer prática de armazenamento de bits ou inteligência, limitando-se apenas a repercutir os sinais que recebe nos seus portos para outro equipamento [13].

O *hub* (concentrador) é um equipamento que desempenha a função de vários *repeaters* num só equipamento. Este equipamento suporta um variado número de portas, todas com a mesma tecnologia, e cada uma suporta a ligação a um terminal ou servidor em topologia em estrela. As estações ligadas ao *hub* comunicam com outras estações de maneira transparente, transportando a informação para o meio físico, através da amplificação e repetição de sinal [13].

O *bridge* (ponte) é um dispositivo inteligente programável que interliga fisicamente várias redes LAN num único troço lógico LAN [18]. O *bridge* efetua ligações ao nível físico como os *repeaters* e desempenha funções ao nível da ligação de dados, ou seja, recebem pacotes de dados, processam-nos e retransmitem-nos para o terminal destino [13]. Este equipamento possibilita igualmente a ligação de redes com tecnologias diferentes.

O *switch* (comutador) é um equipamento de interligação, que tem semelhanças com os *hubs* e os *bridges*. Este dispositivo permite a ligação com vários terminais, servidores e equipamentos numa topologia em estrela a nível físico. A nível de dados, permite o isolamento do tráfego entre os vários troços, fazendo o encaminhamento e a comutação da informação apenas para o troço onde se encontra o dispositivo destino [13].

O *router* (encaminhador) permite a interligação entre redes distintas, através do encaminhamento e comutação da informação entre as sub-redes às quais está ligado [13]. Este equipamento desempenha o papel de *gateway* entre as redes.

3.2.5 Trama ou Mensagem

A trama ou mensagem são os termos utilizados para descrever o conjunto de dados transmitidos sobre a rede [19], [20]. É conhecida também como pacote de dados. A trama é utilizada para

transporte de informação entre dois dispositivos. A figura 14 apresenta o formato de uma trama IEEE 802 [19].

Figura 14 - Formato de uma trama.

3.2.6 Protocolo de Comunicação

As tramas transmitidas via rede *Ethernet* encontram-se normalmente sob o formato de algum protocolo de comunicação [20], o qual é considerado um conjunto de regras para um dado tipo de comunicação [19]. Este conceito define a constituição da trama e o controlo necessário para uma transferência correta de dados [20]. Qualquer protocolo de comunicação deve estar sempre em conformidade com o modelo OSI, o qual é dividido em sete camadas, sendo genérico no sentido em que descreve o funcionamento de um protocolo "ideal" independentemente do meio físico ou tipo de ligação. A tabela 1 apresenta um exemplo deste modelo. Inicialmente foi desenvolvido com o propósito de interligar sistemas abertos, ou seja, sistemas modulares independentes de fabricantes [13]. No entanto, esse objetivo não foi alcançado por razões de mercado, de execução e de desenvolvimento tecnológico. Apesar do insucesso, é considerado um modelo de referência rico em conceitos importantes no que diz respeito à comunicação entre sistemas, sendo aplicável de um modo geral a todas as arquiteturas de comunicação atuais [13].

Tabela 1- Modelo OSI de sete camadas.

Nº	Camada	Função
7	Aplicação	Aspetos de comunicação
6	Apresentação	Representação de dados
5	Sessão	Controlo de diálogo
4	Transporte	Transporte fiável
3	Rede	Encaminhamento de informação
2	Ligação de Dados	Controlo de fluxo e de erros
1	Física	Envio e receção de bits

As camadas têm as seguintes funções:

- **Camada Física**: constitui a *interface* com o meio físico de transmissão e estabelece o modo como a representação lógica dos dados – os *bits*, com o valor lógico 0 ou 1 – são transformados em dados físicos – tensões ou correntes elétricas, sinais óticos, ondas eletromagnéticas, que viajam no meio físico usado [13]. Esta camada é responsável por aspetos referentes ao meio físico, caso dos conetores, transmissão e receção de sinais físicos, e também por aspetos independentes do meio físico, caso da descodificação e codificação de conjuntos de bits recebidos ou enviados;
- **Camada de Ligação de Dados**: é responsável pela criação, transmissão e receção de pacotes de dados [21], podendo fornecer mecanismos locais de controlo de fluxo de informação e de controlo de erros [13]. Proporciona serviços para diversos protocolos na camada de rede e utiliza a camada física para a transmissão ou receção de informação [21];
- **Camada de Rede**: esta camada assegura a interligação entre os vários sistemas terminais, sem ter em conta a localização desses sistemas ou o número e tipo de sub-redes [13]. As suas tarefas prioritárias dizem respeito ao encaminhamento de dados através da rede, através de um complexo conjunto de mecanismos e protocolos;
- **Camada de Transporte**: garante a fiabilidade da comunicação aos sistemas terminais sem ter em conta o tipo ou qualidade da sub-rede usada, através de mecanismos de deteção e recuperação de erros, controlo de fluxo e controlo de sequência [13];
- **Camada de Sessão**: é responsável pelo controlo e sincronização do diálogo entre as aplicações comunicantes [13]. Esta camada estabelece e mantém a comunicação entre dois dispositivos durante o tempo em que estes estiverem a comunicar [20];
- **Camada de Apresentação**: tem como tarefa a representação dos dados, de modo a que todos os sistemas abrangidos entendam a informação [13];
- **Camada de Aplicação**: é a camada que fornece o acesso das aplicações do utilizador à rede, permitindo a execução de diversas tarefas como, por exemplo, transferência de dados, gestão de rede, entre outros [21].

3.3 Ethernet Industrial

Durante décadas os fabricantes têm introduzido os sistemas de controlo em rede nos seus equipamentos. No entanto, as condições de mercado e tecnológicas influenciaram este tipo de

propriedade, obrigando desta forma a que diferentes géneros de rede fossem interligados para tornar o controlo mais centralizado sobre os ativos. A *ethernet* é a escolha esmagadoramente aceite pelas redes de dados, fazendo sentido para as comunicações industriais. Esta tecnologia é bem entendida e apoiada, oferecendo um grande intervalo de endereços, e transmite muito mais dados que as velhas comunicações série, permitindo assim um controlo mais preciso [24].

Segundo Perry S. Marshall [23], a *ethernet* industrial é definida como "uma aplicação bem-sucedida das normas IEEE 802.3 com sistemas de fios, conectores e *hardware* que atendam ao ruído elétrico, vibração, temperatura, e requisitos de durabilidade do equipamento de fábrica, e protocolos de rede que forneçam interoperabilidade e controlo do tempo-crítico dos dispositivos e máquinas inteligentes. Este conceito é uma força em crescimento nas redes industriais devido ao amadurecimento das camadas de *hardware* e de s*oftware*" [22]. A *ethernet* industrial é baseada na norma *ethernet* que, geralmente é encontrada em ambiente de escritório, mas tem que ser modificada para entender os requisitos das aplicações, tais como:

- Entrega de dados determinísticos;
- Suporte a eventos acionados por tempo;
- Topologias que garantam segurança e fiabilidade.

Todos estes fatores são especificados para a *ethernet* industrial [24].

Os protocolos *ethernet* industrial utilizam uma camada MAC (*Media Access Control*) modificada para alcançarem latência muito baixa e respostas determinísticas. A *ethernet* possibilita ainda topologias de rede flexíveis e um número flexível de elementos no sistema.

3.3.1 EtherCAT

O *EtherCAT* (*Ethernet for Control Automation Technology*) é um protocolo de comunicação de tecnologia tempo-real *Ethernet* Industrial que foi desenvolvido originalmente pela *Beckhoff Automation*. Este protocolo é descrito na norma IEC61158 sendo adequado para requisitos de tempo real duros e leves (*hard and soft*) na tecnologia de automação, em medição e teste, entre muitas outras aplicações [25].

O mestre *EtherCAT* envia um telegrama para a rede que passa através de cada nó. Cada nó escravo *EtherCAT* lê os dados que lhe são dirigidos e insere os dados na trama enquanto a trama está em movimento a jusante. O último elemento num segmento ou ramo deteta uma porta aberta e envia o telegrama de volta para o mestre através da utilização caraterística *full-duplex*

da tecnologia *Ethernet*. O mestre *EtherCAT* é o único nó numa rede que pode enviar uma trama *EtherCAT*, enquanto os outros nós simplesmente encaminham as tramas a jusante. Este conceito previne os atrasos imprevisíveis e garante as capacidades em tempo-real.

O *EtherCAT* é um protocolo de camada MAC, sendo transparente para qualquer camada elevada de protocolos *Ethernet* como o TCP/IP (*Transmission Control Protocol/Internet Protocol*), UDP (*User Datagram Protocol*), entre outros. Pode ligar até 65535 nós numa rede, e o mestre *EtherCAT* é um controlador padrão *Ethernet*, simplificando assim a configuração da rede. O *EtherCAT* entrega soluções de *ethernet* industriais flexíveis, baixo custo, e compatíveis com redes devido à baixa latência de cada nó escravo [14].

3.3.2 Ethernet/IP

O *Ethernet/IP* é um protocolo *Ethernet* Industrial projetado pela *Rockwell Automation* [14], e gerido nos dias de hoje pela ODVA (*Open DeviceNet Vendor Association*) [26]. Este protocolo aplica o CIP (*Common Industrial Protocol*) sobre a norma *Ethernet* e tecnologias TCP/IP. O CIP utiliza o modelo produtor/consumidor em vez do modelo cliente/servidor. O modelo produtor/servidor diminui o tráfego de rede e aumenta a velocidade de transmissão.

O protocolo CIP implementa um caminho para o envio de mensagens dos dispositivos produtores do sistema para os dispositivos consumidores [27].

O *Ethernet/IP* utiliza dois tipos de comunicação para o envio de mensagens: explícito e implícito.

O método explícito utiliza o modelo pedido/resposta, sendo um tipo de comunicação para dados não reais. Pode ser usado por um HMI para recolher dados por uma ferramenta de programação de dispositivos. Falando em termos de CIP, com a mensagem explícita, é possível requisitar um serviço de um objeto em particular, como por exemplo um serviço ou leitura. Este tipo de mensagem utiliza o protocolo TCP (*Transmission Control Protocol*) para a comunicação, podendo ser utilizado com ou sem a ligação CIP.

O método implícito utiliza uma natureza referida como tempo crítico e "I/O". Este tipo de comunicação é utilizado para troca de dados em tempo real, onde a velocidade e a baixa latência são essenciais. Com este tipo de mensagem estabelece-se uma ligação (ligação CIP) entre dois dispositivos e produz mensagens implícitas entre os mesmos. A mensagem implícita utiliza o

protocolo UDP (*User Datagram Protocol*) para comunicação, podendo ser *multicast* ou *unicast* [26].

O Ethernet/IP utiliza uma ligação ponto-a-ponto para a transferência de dados entre o transmissor e o recetor [27].

3.3.3 POWERLINK

O *POWERLINK* é um protocolo *Ethernet* Industrial desenvolvido pela B&R, sendo caraterizado pelos ciclos de tempo na gama dos microssegundos, aplicabilidade universal, e máxima flexibilidade na configuração de rede. Este protocolo fornece todas as caraterísticas da norma *Ethernet*, incluindo cruzamento de tráfego e escolha livre da topologia de rede [28].

O *POWERLINK* utiliza uma mistura de procedimentos *timeslot* e *polling* para atingir a transferência de dados isócrono. De modo a garantir coordenação, um PLC é designado para ser o MN (*Managing Node*). Este gestor obriga a impor os tempos de ciclo, que servem para sincronizar todos os dispositivos e os controlos cíclicos da comunicação de dados. Os outros dispositivos operam como CN (*Controlled Node*). No decorrer de um ciclo de relógio, o MN envia gradualmente mensagens "*Poll Requests*" para cada CN, que transmite dados do MN para todo o CN que se encontre ligado. Todo o CN responde imediatamente a este pedido com um "*Poll Response*", que todos os outros elementos podem escutar. O ciclo *POWERLINK* consiste em três períodos:

- O primeiro período é conhecido como "*Start Period*", em que o MN envia um SoC (*Start of Cycle Frame*) para sincronizar todos os dispositivos CN;
- O segundo período é onde tem lugar a troca de dados isócrono cíclico, onde é otimizada a largura de banda por multiplexação;
- O terceiro período marca o início da fase assíncrona, que permite a transferência de grandes pacotes de dados *non-time-critical*.

Este protocolo faz distinção entre os domínios de tempo-real e não tempo-real. Dado que a transferência de dados é em períodos assíncronos, suporta tramas de norma IP, os *routers* separam os dados de forma segura e transparente dos dados de tempo-real.

3.3.4 PROFINET

O *PROFINET* é um protocolo de *Ethernet* industrial desenvolvido com os contributos chave da *Siemens* e outros membros da organização PNO (*Profibus User Organization*). O sistema de comunicação deste protocolo especifica a transferência de dados entre controladores de entradas/saídas, a parametrização, o diagnóstico e a implementação de uma rede.

Este protocolo pode ser dividido em classes de desempenho dependendo dos requisitos de tempo: *PROFINET RT* para não tempo-real ou tempo-real suave e *PROFINET IRT* para tempo-real duro.

O *PROFINET RT* envia carga útil e dados em não tempo-real crítico dentro de um ciclo, adotando o princípio produtor/consumidor. Um canal tempo-real é reservado para carga útil de alta prioridade que é transmitida diretamente via protocolo *Ethernet*. No entanto, a configuração de dados e diagnósticos são enviados via protocolo UDP/IP. Assim, ciclos de tempo de 10 milissegundos podem ser realizados para aplicações de entradas/saídas. Aplicando a multiplexação por divisão de tempo baseada na gestão de comutadores, o PROFINET IRT fornece tempos de ciclo relógio sincronizado abaixo de 1 milissegundo, que são utilizados em aplicações, por exemplo, de controlo de movimento [28].

3.3.5 *Modbus*/TCP

O *Modbus*/TCP é uma extensão do protocolo *Modbus* que foi desenvolvido pela *Modicon* (hoje em dia é uma divisão da *Schneider Electric*). Implementa os mesmos serviços e o mesmo modelo objeto que as anteriores versões, utilizando a norma *Ethernet* para a transferência de pacotes de dados via TCP/IP [28]. As mensagens são suportadas por redes configuradas em cliente/servidor e ponto-a-ponto. Todas as mensagens objeto são enviadas entre dois elementos para executar serviços de transação, os quais são associados a um serviço pedido e consistem numa mensagem pedido e a mensagem resposta correspondente [29]. Normalmente, as aplicações *Modbus* operam em modelo cliente/servidor, onde um cliente pesquisa o seu servidor por informação. Devido à capacidade ponto-a-ponto, o *Modbus*/TCP pode suportar outros métodos de comunicação, tais como servidores reportando assincronamente em cíclico ou mudança de estado utilizando o serviço de notificação. No entanto, o determinismo pode ser comprometido quando se recorrer a métodos de reportação assíncronos [29].

3.4 Comunicações FINS

3.4.1 Visão Geral das Comunicações FINS

O serviço de comunicações FINS (*Factory Interface Network Service*) é um protocolo de rede criado pela OMRON com o propósito de proporcionar um caminho de dados consistente entre os PLCS e os computadores de diversas redes ou na mesma rede. Este serviço é compatível com outros tipos de rede, como é o caso da *Ethernet*, *Host Link*, *Controller Link*, *SYSMAC LINK*, *SYSMAC WAY* e *Toolbus* [30].

A comunicação FINS permite ao utilizador executar operações no PLC de leitura e escrita de dados na memória, não havendo necessidade de configurar o PLC para responder a estes comandos. Através deste protocolo é também possível carregar/ler programas dos PLCS OMRON. Deste modo, o PLC recorre a um dispositivo *Ethernet* que utiliza uma porta UDP/IP (*User Datagram Protocol/Internet Protocol*) para realização do envio e receção de mensagens FINS [31].

3.4.2 Comunicações em Rede *Ethernet*

Na rede *Ethernet*, os dispositivos comunicam através de pacotes UDP. Para identificar os equipamentos em redes IP é necessário configurar os seguintes campos:

- Endereço IP;
- Porta.

No caso de dispositivos FINS, além destes dois são adicionados outros dois campos:

- Endereço de nó FINS;
- Número de CPU.

Na figura 15 está apresentado um esquema que demonstra a troca de pacotes UDP entre os dispositivos em rede Ethernet [31].

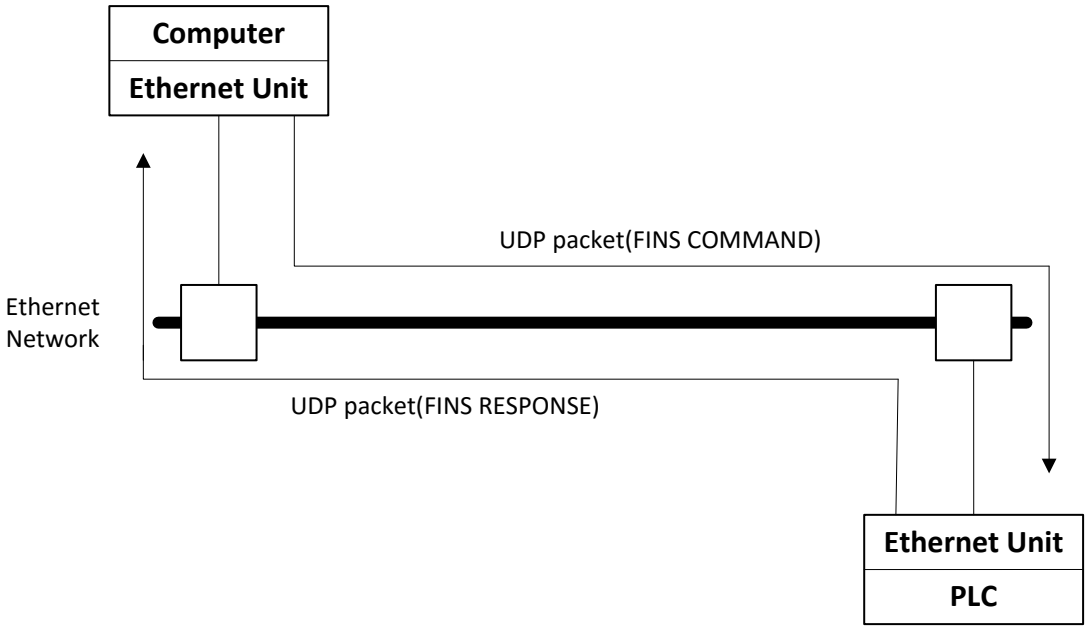


Figura 15 - Diagrama de Troca de Mensagens FINS.

3.4.3 Tramas FINS

O funcionamento do serviço de comunicações FINS é realizado através da troca de tramas de comando FINS e as correspondentes tramas de resposta. No entanto, existem tramas de comando que não exigem uma resposta.

As tramas são definidas da seguinte forma: cabeçalho FINS (FINS *Header*), campo de comando FINS (FINS *Command*) e parâmetro FINS/dados (FINS *parameter/data*). O cabeçalho FINS é responsável pelo armazenamento da informação de controlo de transferência do comando; o campo de comando FINS define a instrução de comando a ser executada; e o parâmetro FINS/dados regista os parâmetros de dados de transmissão ou resposta.

Nas figuras 16 e 17 é possível visualizar a constituição de uma trama de comando e de uma trama de resposta.

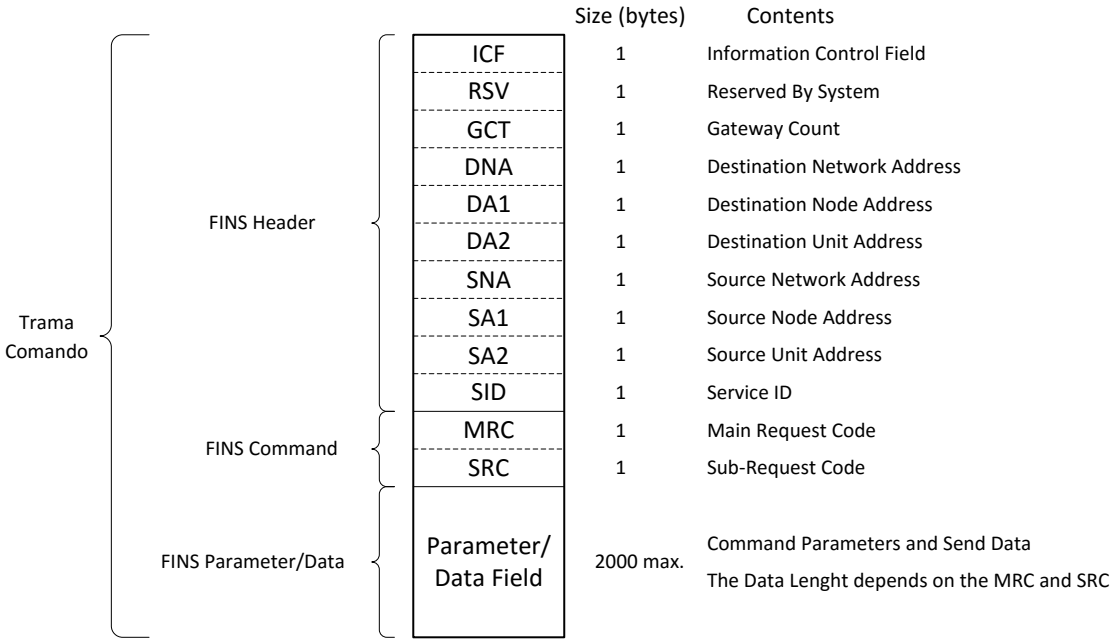


Figura 16 - Constituição da Trama de Comando.

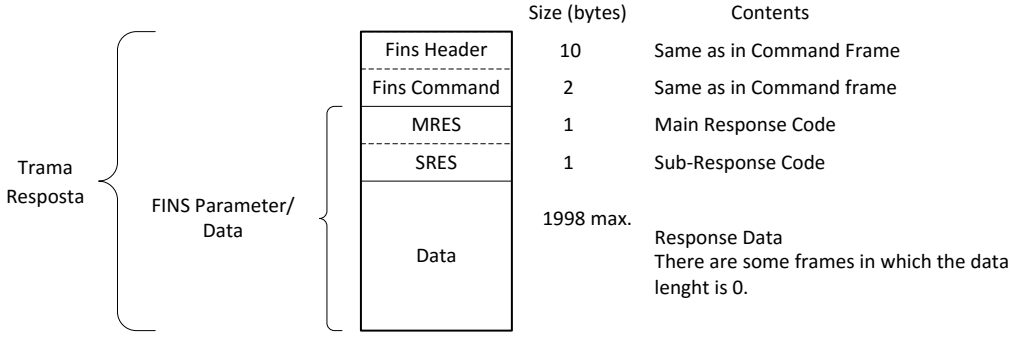


Figura 17 - Constituição da Trama de Resposta.

De modo a garantir que a trama seja entregue no endereço destino, a informação do cabeçalho FINS deve ser composta da seguinte forma [31]:

- ICF (*Information Control Field*): é definida de acordo com a figura 18;
- RSV (*Reserved by system*): Este campo encontra-se a 0, sendo utilizado pelo sistema;
- GCT (*Gateway Count*): É o campo responsável pela quantidade de redes para as quais a trama pode ser enviada. Este parâmetro é definido com o valor 2;
- DNA (*Destination Network Address*): parâmetro que especifica a rede onde se encontra nó de destino. Este endereço pode ser especificado de 1 a 127;

- DA1 (*Destination Node Address*): parâmetro que especifica endereço do nó para o qual é enviado o comando, sendo este usado para endereço FINS. No caso de a rede apresentar diversos nós de comunicação, o DA1 especifica o endereço do nó pretendido na rede. Este endereço pode ser especificado de 1 a 126;
- DA2 (*Destination Unit Address*): parâmetro que especifica o número do equipamento (CPU) pretendido no endereço destino. Este parâmetro é definido com o valor 0;
- SNA (*Source Network Address*): identifica a rede onde está localizado o nó de origem. Este parâmetro apresenta o mesmo intervalo de endereços que o campo DNA;
- SA1 (*Source Node Address*): identifica o endereço do nó de origem. Este parâmetro apresenta o mesmo intervalo de endereços que o campo DA1;
- SA2 (*Source Unit Address*): identifica o equipamento (CPU) no nó de origem. Este parâmetro é definido com o valor 0;
- SID (*Service ID*): Este campo é utilizado para identificar a trama, isto é, o número atribuído a este campo numa trama de comando é o mesmo na trama de resposta de maneira a identificar a tarefa que foi desempenhada. Este valor varia entre 0 e 255.

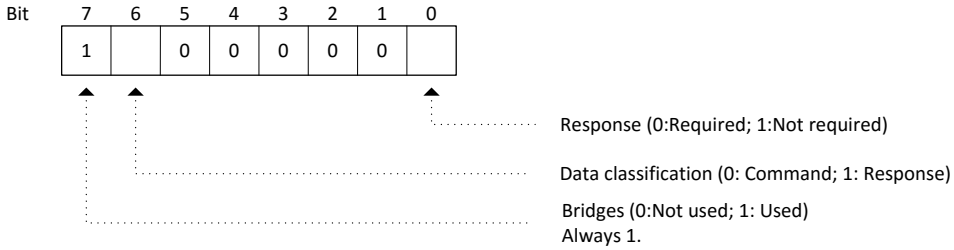


Figura 18 - Campo ICF.

Na figura 19 é apresentado um exemplo da constituição do comando para leitura de entradas e saídas de memória.

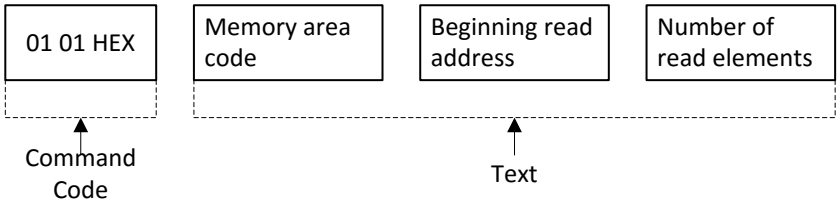


Figura 19 - Constituição de Comando para Leitura de Memória.

A figura 20 apresenta o comando descrito na figura anterior para realizar a leitura de 10 *words* começando pela memória DM 10 (000A00 HEX), onde o endereço é composto por três *bytes.* Os dois primeiros correspondem à *word* da memória e o terceiro corresponde ao número usado para os *bits*.

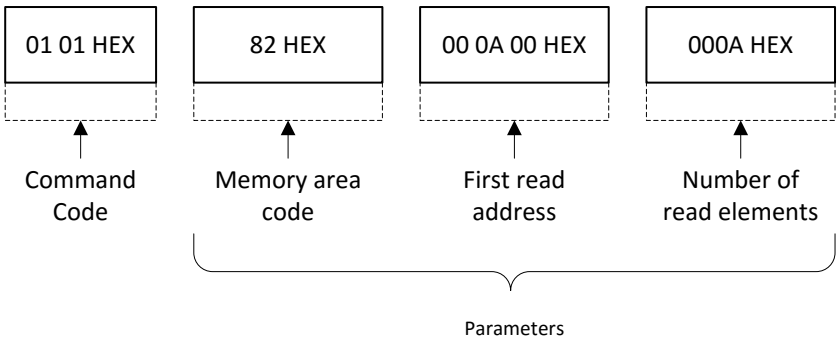


Figura 20 - Comando para Leitura de 10 *words*.

Na figura 21 é apresentado o formato de resposta à leitura de entradas e saídas de memória.

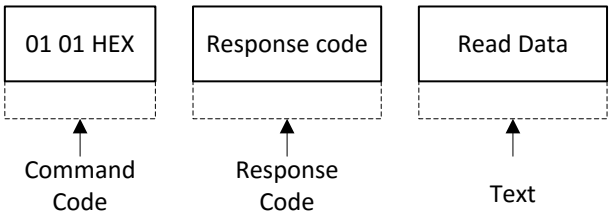


Figura 21 - Constituição da Resposta ao Comando de Leitura de Memória.

3.4.4 Método FINS/UDP

O método FINS/UDP é o formato da comunicação FINS que recorre ao protocolo UDP/IP para a troca de dados. O protocolo UDP/IP é um protocolo de comunicações sem ligação, ou seja, cada mensagem é independente da seguinte. Os nós detêm caraterísticas iguais mas não possuem uma ligação clara. É possível afirmar então que o protocolo UDP é como a entrega de um conjunto de folhas em mão a outra pessoa, ou seja, embora o protocolo UDP permita transmissões rápidas, esta forma de comunicação de dados não é tão fiável como o protocolo TCP, porque não garante que o pacote de dados chegue ao destino e não garante a estrutura inicial do mesmo. De forma a contornar esta questão, quando se pretende transmitir com um tamanho significativo, torna-se necessário a implementação de roteamento e definir medidas

para nova requisição de dados às aplicações caso seja necessário. Desta forma aumenta-se a fiabilidade da transmissão de dados entre os nós e a segurança dos mesmos.

Na figura 22 é apresentado um diagrama de troca de dados através do método FINS/UDP. Os pacotes são enviados em sentido único, não existindo a confirmação de receção dos mesmos.

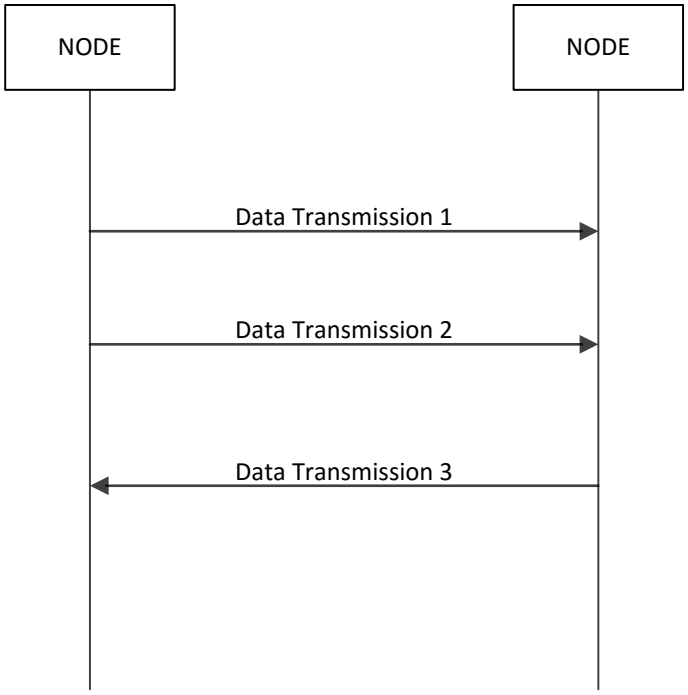


Figura 22 - Transmissão de dados em protocolo UDP/IP.

Resumindo, o método FINS/UDP possui as seguintes caraterísticas:

- Utiliza um protocolo sem ligação, logo não existe limite para o número de correções;
- Pode ser usado em *broadcasting*;
- Caso os dados sejam enviados através de uma rede IP com múltiplas camadas, a fiabilidade das comunicações desce.

3.4.5 Formato da Trama FINS/UDP

A figura 23 demonstra a estrutura de um pacote de dados UDP usado para o método FINS/UDP para envio e receção de dados numa rede *Ethernet*. O pacote é estruturado da seguinte forma: *Ethernet* Ver.2, Trama IP, Trama UDP e Trama FINS. A secção de dados UDP (Trama FINS) que exceda 1472 bytes é dividida em pacotes UDP para transmissão. Os dados UDP divididos são adicionados automaticamente à camada do protocolo UDP/IP. Normalmente, não há

necessidade de prestar atenção à camada de aplicação para esta divisão, no entanto, pode não ser possível o envio de 1472 bytes de pacotes UDP através de rede IP com múltiplas camadas [32].

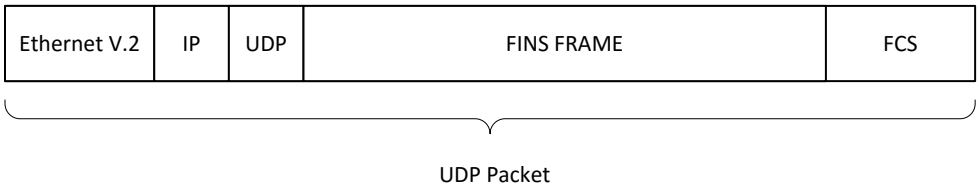


Figura 23 - Estrutura de pacote de dados FINS/UDP.

3.4.6 Números de Porta UDP para FINS/UDP

O número da porta UDP é responsável pela identificação da camada de aplicação. Quando as comunicações estão a ser executadas pelo protocolo UDP/IP, este número de porta deve ser alocado para o serviço de comunicações FINS.

Normalmente, o número que é usado para definir a porta de comunicações FINS é o 9600. Porém, é possível usar outro número desde que este seja atribuído nas configurações respetivas.

O número da porta UDP para a aplicação do utilizador funciona de forma diferente das portas que são usadas para os endereços, dependendo da configuração do método de conversão de endereços. Basicamente, a mesma porta pode ser usada como número de porta UDP num dispositivo *Ethernet*, mas o número não precisa ser o mesmo se respeitar as seguintes condições:

- Enviar comandos da aplicação de utilizador quando o método de geração automático é usado como método de conversão de endereços da unidade *Ethernet*;
- Enviar comandos da aplicação de utilizador sem registar os endereços IP na tabela de endereços IP quando o método de tabela de endereços IP é usado como método de conversão de endereços do dispositivo *Ethernet*;
- Enviar comandos da aplicação do utilizador sem registar os endereços IP na tabela de endereços IP quando o método combinado é usado como método de conversão dos endereços do dispositivo *Ethernet*.

Para cada método de conversão de endereços a ser utilizado, quando os comandos são enviados da unidade *Ethernet*, usa-se o mesmo valor de porta UDP para o dispositivo *Ethernet* e para a aplicação do utilizador [32].

3.4.6 Conversão de Endereços em Comunicações FINS

Utilizando as comunicações FINS, os nós de comunicação devem ser configurados com o sistema de endereços FINS, para que os dados sejam transmitidos em rede *Ethernet*, mas em formato de endereço IP. Assim sendo, são usados métodos de conversão de endereços, onde é possível emparelhar um endereço IP com endereço de nó FINS.

Existem três métodos para este efeito:

i. Método de Geração Automática;
ii. Método de Tabela de Endereços IP;
iii. Método Combinado (Tabela IP + Método de Geração Automática).

Método de Geração Automática:

O método de geração automática utiliza como número do nó FINS o número de identificação do equipamento do endereço IP que corresponde ao último *byte* do endereço. O número de identificação de rede no endereço IP é utilizado como número de rede FINS, correspondendo este ao terceiro *byte* do endereço IP [33].

A geração automática usa o seguinte tipo de endereço IP remoto, configurado através do endereço IP local, máscara de rede e o número de nó FINS:

Endereço IP Remoto = (Endereço IP Local AND Máscara de Rede) OR Número de nó FINS.

Através deste método é fácil entender a relação entre os endereços FINS e os endereços IP. Contudo, apresenta algumas limitações:

- Este método só é aplicável a endereços que se encontrem na mesma rede;
- O número de utilizadores atribuído a endereços remotos tem um intervalo de nós FINS que deve ser respeitado, de 1 a 126;
- O número de nó FINS definido no equipamento e o número de utilizador no endereço IP (último byte do endereço) devem ser definidos com o mesmo valor [33].

Método de Tabela de Endereços IP:

Ao passo que o método de geração automática obtém o endereço IP através do número do nó FINS, o método de tabela de endereços IP converte o número do nó FINS para endereço IP com base na definição da tabela de endereços, como é apresentada na figura 24.

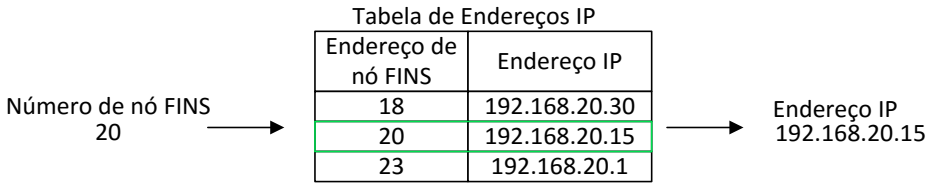


Figura 24 - Tabela de Relação de Endereços.

O método de tabela de endereços IP simplesmente prepara a conversão da tabela, apresentando como vantagem o emparelhamento de nós FINS com endereços IP livremente. Contudo, têm a desvantagem de o registo de endereços na tabela ser limitado a 32 endereços, ou seja, 32 nós, incluindo o nó local onde é definida [33].

Método Combinado (Tabela IP +Método de Geração Automática):

O método combinado é a junção dos dois métodos referidos anteriormente.

Ao usar este método, a tabela de endereços IP é consultada primeiro, e se o endereço FINS estiver incluído na tabela, o endereço IP correspondente é facultado. Caso o endereço FINS não esteja incluído, o endereço IP é calculado recorrendo ao método de geração automática, tal como é apresentado na figura 25 [33].

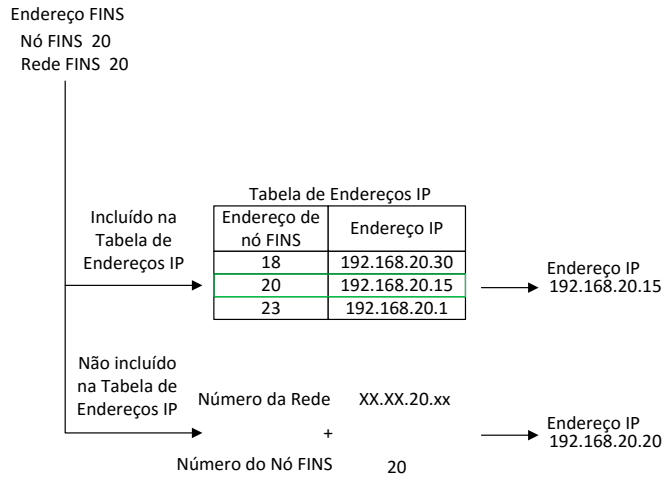


Figura 25 - Método Combinado.

3.5 Sumário

Neste capítulo foi apresentada uma perspetiva geral da arquitetura de rede industrial aplicada em automação industrial, assim como as possíveis topologias de rede a aplicar e os equipamentos utilizados.

Foram ainda apresentados os protocolos *Ethernet* Industrial mais utilizados na indústria assim como o protocolo FINS que foi aplicado neste trabalho.

4 Equipamentos OMRON

4.1 Enquadramento

Este capítulo abrange os equipamentos OMRON utilizados neste trabalho. É feita a descrição de cada equipamento e as suas caraterísticas, assim como uma breve demonstração das configurações a serem realizadas nos respetivos equipamentos para comunicação via *Ethernet* através do protocolo FINS.

4.2 Autómato OMRON

4.2.1 Visão Geral de um Autómato

Um autómato programável industrial (PLC) é um equipamento eletrónico, programável em linguagem não informática, desenvolvido para o controlo em tempo real de processos sequenciais em ambiente industrial [34].

Este elemento controla, com base na informação disponibilizada pelos sensores e programa lógico interno, atuando de seguida nos atuadores da instalação (ver Figura 26).

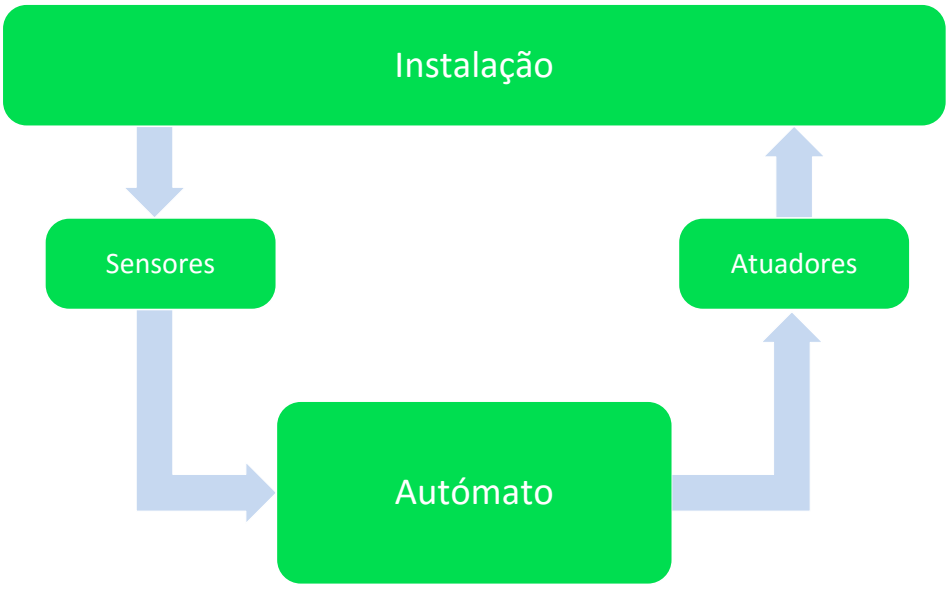


Figura 26 - Ciclo de controlo; adaptado de [34].

Na realização deste trabalho utilizou-se um autómato OMRON modelo CJ1M-CPU22, com um módulo *Ethernet* CJ1W-ETN11 de modo a entender o funcionamento do protocolo FINS.

A figura 27 apresenta o equipamento em questão utilizado.

4.2.2 Modelo CJ1M-CPU22 OMRON

Figura 27 - Autómato CJ1M OMRON com módulo *Ethernet*.

O autómato CJ1M-CPU22 é uma mais-valia para o controlo de processos devido ao seu tamanho reduzido (90 mm de altura e 65mm de largura) [35], e apresenta um formato do tipo modular, que permite acrescentar módulos ou cartas que são inseridas em *rack* [34]. Desta forma é possível controlar os custos de instalação deste equipamento e tem uma fácil adaptação ao processo a controlar.

Este autómato destaca-se pela capacidade de ligar 10 unidades (expansão *rack*), execução de programas de 10000 passos, e apresenta um tempo de execução de instruções em *ladder* de 0.1 microssegundos. Consoante o processo a controlar, é possível adaptar este equipamento para o efeito, sendo possível programar este equipamento através da norma IEC 61131-3, através das seguintes linguagens de programação: LD (*Ladder Diagram*), SFC (*Sequential Function Chart*), e ST (*Structrured Text*).

O LD é uma linguagem de programação gráfica, baseada no princípio de um diagrama de contatos elétricos. Este tipo de programação permite o controlo discreto e sequencial, e usa blocos para funções de controlo regulatório e funções especiais [36]. As variáveis lógicas, caso das entradas ou variáveis intermédias, são representadas por contatos e as saídas por relés ou lâmpadas. O diagrama é formado por uma ou várias linhas horizontais que se organizam verticalmente.

A figura 28 apresenta um exemplo de demonstração da linguagem de programação *Ladder*.

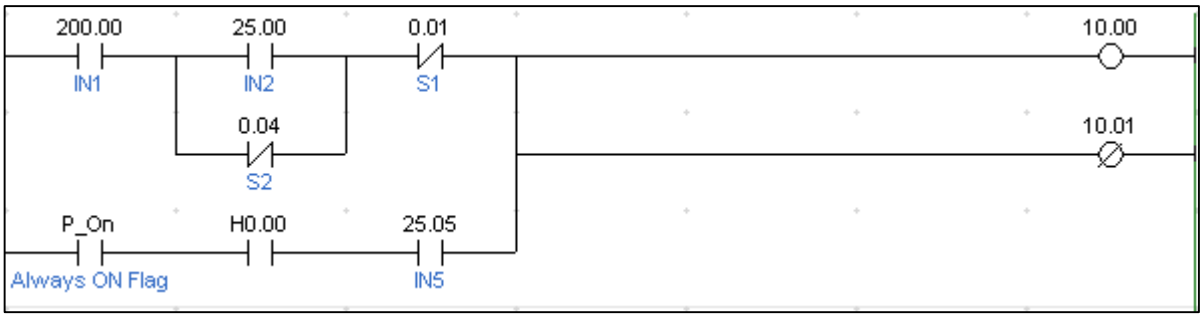


Figura 28 - Exemplo de demonstração de diagrama em *Ladder*.

O SFC é uma linguagem de programação gráfica que foi desenvolvida em França e tem como base as redes de Petri e o Grafcet (*Graphe Fonctionnel de Command Etape Transition*).

Este tipo de programação permite planificar a estrutura de um programa, decompor o programa por fases, e confere uma visão global do sistema a controlar [36].

A filosofia desta programação baseia-se na explicação do automatismo a projetar através de etapas e transições. Nas etapas são postas em prática ações podendo estas não ser realizadas na etapa se a condição não se verificar. Em cada momento, no diagrama projetado, só uma etapa se encontra ativa.

De modo a ocorrer transição de uma etapa para outra é necessário que se confirme uma ou mais condições de transição. Este procedimento é conhecido por recetividade.

A figura 29 demonstra um exemplo desta programação realizada em autómatos.

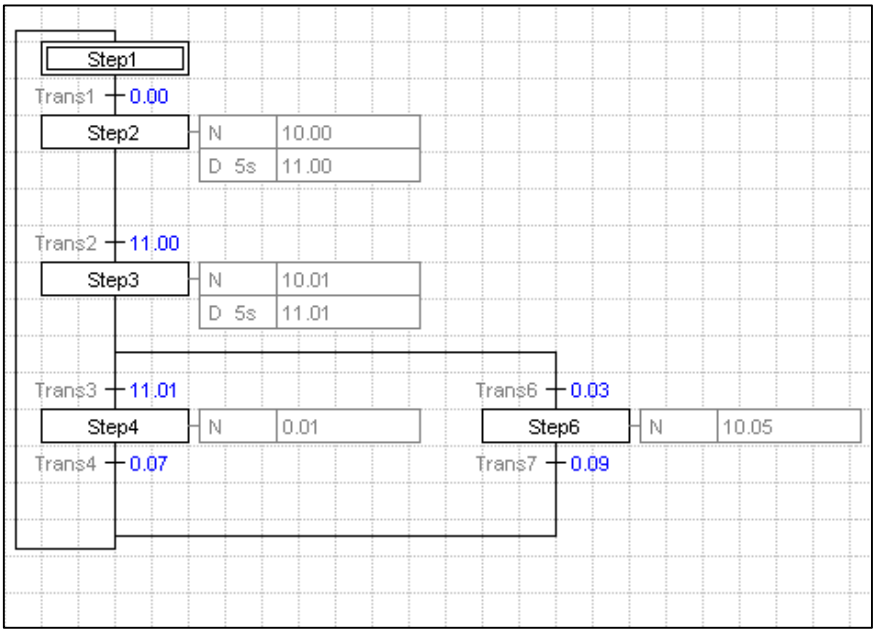


Figura 29 - Exemplo de demonstração SFC.

O ST é igualmente uma linguagem textual, sendo considerada uma linguagem de alto nível que permite a programação estruturada. A principal vantagem desta programação está no uso de sub-rotinas para a execução de diferentes partes de uma função de controlo [36]. Esta programação é um bom modelo para tomada de decisões, cálculos, implementação de algoritmos, entre outras tarefas (ver Figura 30).

```
IF value < 7 THEN
WHILE value < 8 DO
value := value +1 ;
END_WHILE;
END_IF;
```

Figura 30 - Exemplo de demonstração de ST; adaptado de [37].

Na gama de equipamentos OMRON, todos as memórias apresentam registos de 16 posições ou *bits*. Estes registos permitem reter a informação lógica do programa. Este endereçamento é feito no formato xxxx.yy, onde xxxx corresponde ao número de registo ou canal, e o yy ao número de *bit* ou relé (entre 0 e 15) [34].

O autómato permite a utilização das seguintes áreas de memória: área CIO (*Core I/O Area*), área WR (*Work Area*), área HR (*Holding Area*), área AR (*Auxiliary Area*), Temporizador (*Timer Area*), Contador (*Counter Area*), área DM, área IR (*Index Registers*), e área TK (*Task Flag Area*):

- A memória CIO é responsável pelo endereçamento das entradas e saídas físicas, apresentando um intervalo de canais (canal⇔ *word*) entre 0000 e 6143;
- A memória WR é a memória de trabalho, sendo utilizada para a programação de etapas do equipamento. Esta memória apresenta um espaço de canais entre 000 e 511;
- A memória HR é a memória de retenção, sendo utilizada para o controlo da execução do programa, e em caso de falha de alimentação do equipamento o estado lógico mantém-se. Esta memória tem um intervalo de registos entre 000 e 511;
- A memória AR é a memória auxiliar, sendo utilizada para controlo e informação de operações do equipamento, e está dividida em duas partes: uma só de leitura e outra de leitura e escrita. Esta memória apresenta um intervalo de canais de 000 a 959;
- O temporizador é a memória utilizada para controlar o tempo de execução de uma dada tarefa. Esta memória tem um intervalo de 0000 a 4095;

- O contador é a memória usada para controlar o número de vezes que uma dada tarefa é executada. Esta memória tem um intervalo de 0000 a 4095;
- A memória DM é uma memória utilizada para leitura e escrita de dados, sendo igualmente utilizada para funções especiais. Esta memória encontra-se dividida num intervalo que vai de 00000 a 32767;
- A memória IR é a memória utilizada para registo de índices, sendo utilizada para endereços indiretos, contento a área e o endereço pretendido. Esta memória encontra-se organizada entre o intervalo de registos 0 e 15;
- A memória TK é a memória utilizada para leitura de *flags* de tarefas cíclicas, sinalizando o estado em que se encontram (ligada ou desligada). Esta memória tem um intervalo de canais entre 00 e 31 [34], [35].

4.2.3 Módulo de Comunicação Ethernet

O módulo *Ethernet* CJ1W-ETN11 permite ao autómato efetuar a comunicação via *Ethernet* através de protocolos de comunicação, como é o caso da comunicação de dados via *sockets* TCP/IP e UDP/IP, protocolo FINS (protocolo da OMRON), protocolo FTP (*File Transfer Protocol*) e SMTP (*Simple Mail Transfer Protocol*) [31].

Sendo o objetivo deste trabalho a utilização do protocolo FINS para a comunicação entre equipamentos é necessária a configuração do autómato para o efeito através da utilização dos métodos de conversão de endereços referidos no capítulo 3.

O diagrama da figura 32 demonstra todo o processo de configuração de endereços remotos.

O método de geração automática requer que o utilizador configure o número do nó FINS no módulo *Ethernet* e do número do barramento CPU da *rack* a ser utilizado.

4.2.4 Configuração no *Cx-Programmer*

O *Cx-Programmer* é um *software* desenvolvido pela OMRON para a programação da sua gama de autómatos. Esta ferramenta permite a programação conforme a norma IEC 61131-3 dos autómatos e tem um ambiente de desenvolvimento amigável para compreensão do utilizador (ver Figura 31). As versões mais recentes deste *software* já permitem a simulação do comportamento do autómato, tornando-se uma mais-valia para a realização de testes por parte do utilizador antes da transferência do programa para o equipamento de controlo.

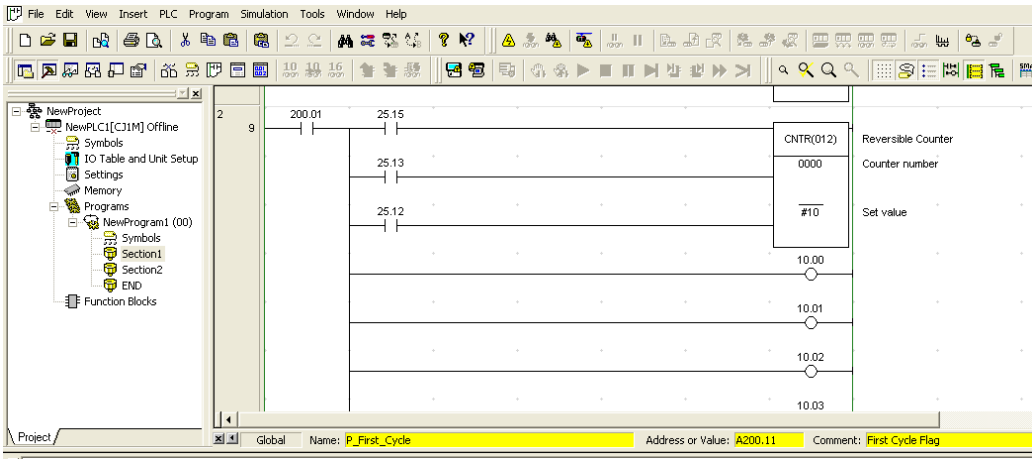


Figura 31 - Exemplo de programa desenvolvido em *Cx-Programmer*.

As configurações de comunicação via *Ethernet* para o protocolo FINS são igualmente realizadas no *Cx-Programmer*, acedendo à tabela de entradas e saídas do PLC e atribuindo o respetivo módulo na *Rack*. É possível definir estes parâmetros acedendo ao campo *IO Table and Unit Setup*, e depois no campo *Main Rack* indica-se o módulo pretendido.

A figura 33 apresenta os parâmetros que foram atribuídos ao módulo *Ethernet*.

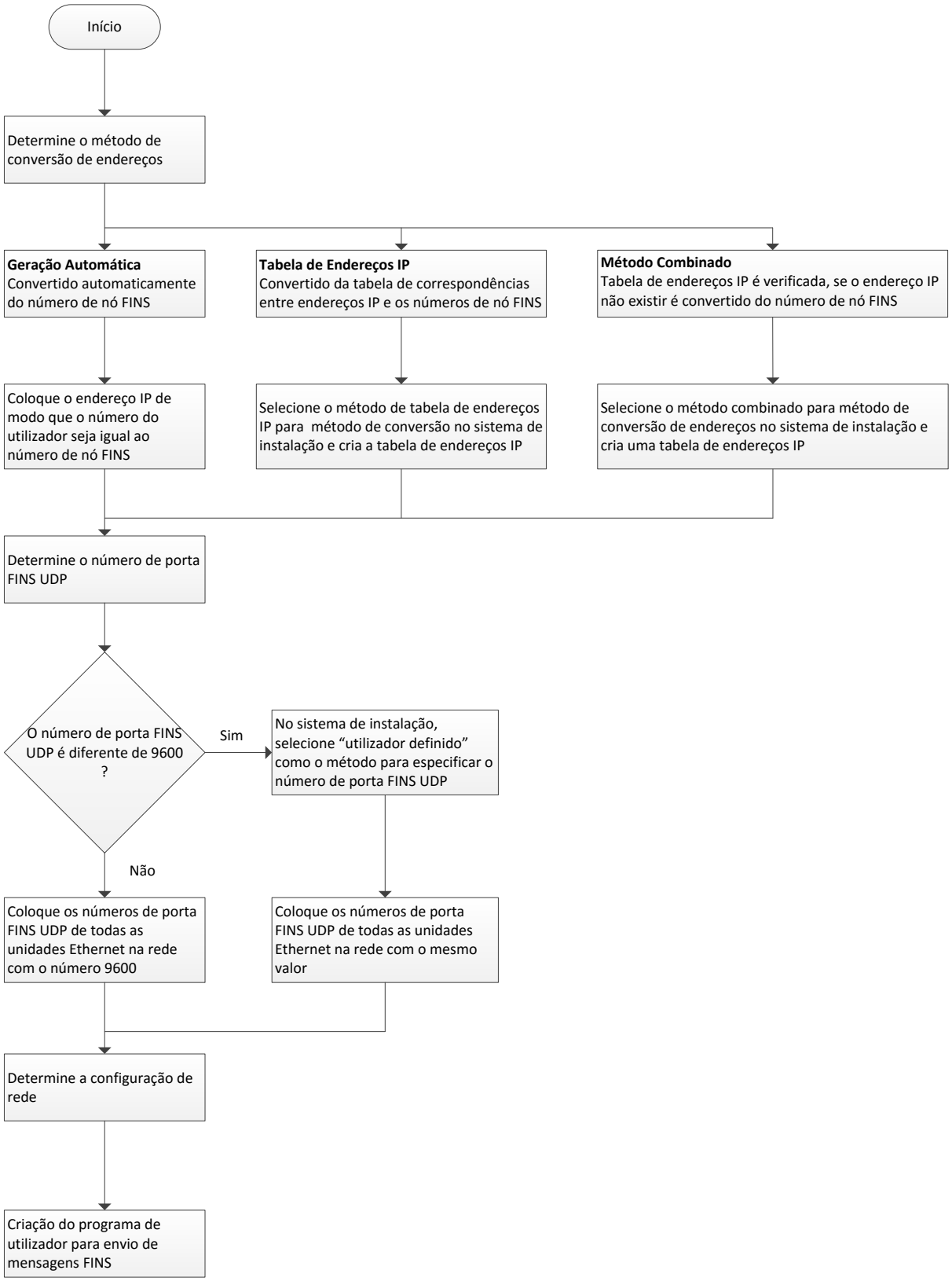


Figura 32 - Diagrama de configuração dos parâmetros de comunicação FINS no *Cx-Programmer*; adaptado de [31].

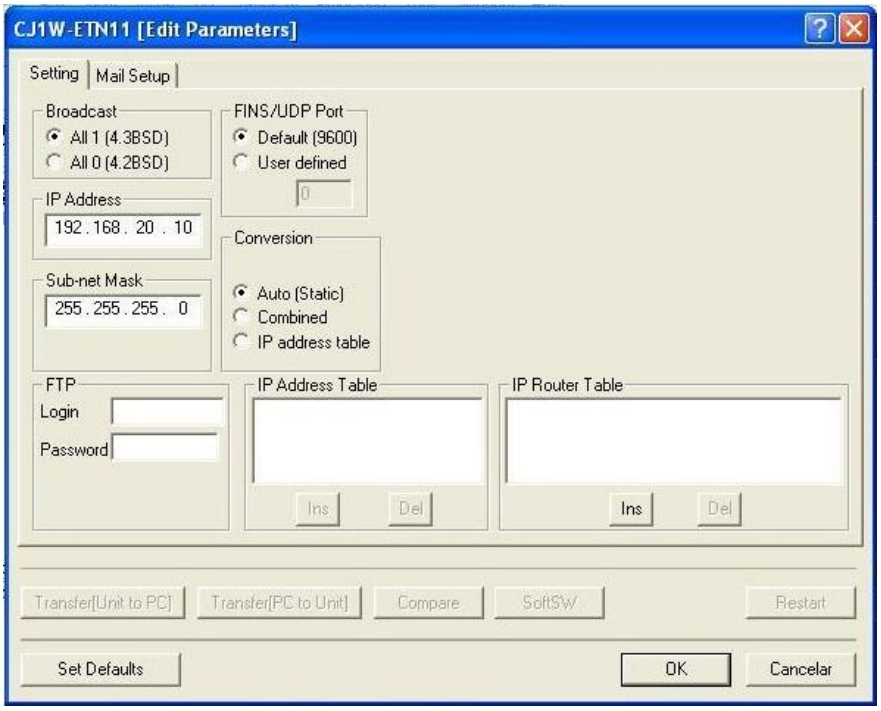


Figura 33 - Edição de parâmetros para comunicação no *Cx-Programmer*.

O método de conversão de endereços usado é o de geração automática que se encontra indicado no campo *Conversion* com a opção "*Auto (Static)*". O número do equipamento do endereço IP (último *byte* do endereço) é igual para o endereço de nó FINS, ou seja, o nó toma o valor 10.

Estando estes parâmetros editados, resta definir o endereço do nó de rede FINS de modo a estabelecer uma rede FINS para comunicar com outros equipamentos. Estes parâmetros são configurados no *Cx-NET Network*, uma ferramenta usada para atribuir um endereço de rede, no caso presente para o protocolo FINS, sendo necessário reservar uma unidade do CPU para atribuir o endereço de rede FINS (ver Figura 34). Assim sendo, cria-se um projeto com os parâmetros do autómato e depois configura-se a rede no campo *Routing Table* e escolhe-se a opção *FINS Network Net*.

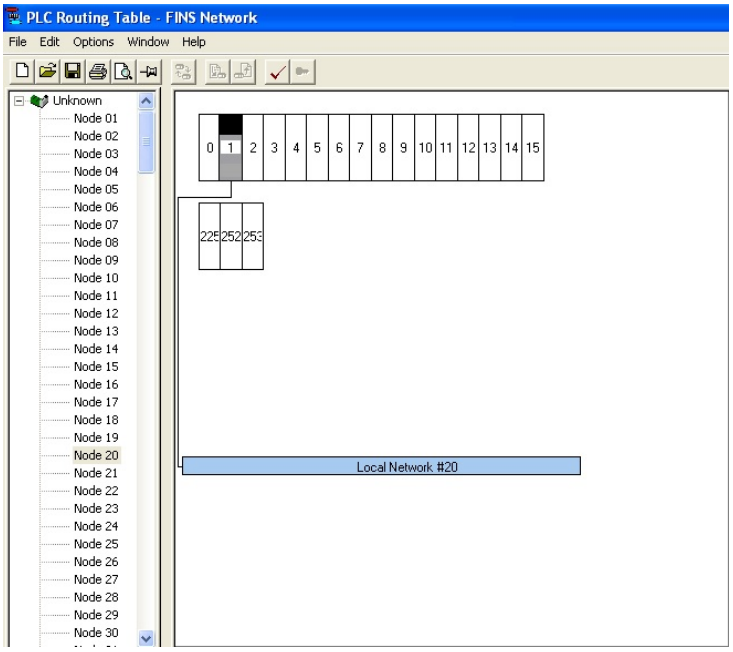


Figura 34 - Edição do endereço de rede FINS em *Cx-NET Network*.

Desta forma, tem-se o autómato preparado para comunicar com os restantes elementos que se encontrem ligados à rede *Ethernet* com recurso às mensagens FINS.

4.3 Consola Interativa OMRON

4.3.1 Visão Geral de uma Consola

A consola interativa é um terminal programável que funciona como HMI. Este equipamento permite criar janelas de interruptores de modo a enviar dados para um autómato e janelas que disponibilizam a informação contida no autómato ao operador. Através deste tipo de consola, é possível realizar as seguintes operações [33]:

- Transferência de Janelas de Dados: a janela de dados desenvolvida é transferida para o terminal programável via *Ethernet*;
- Apresentação de Janelas: O terminal programável apresenta os dados ao utilizador conforme o desenvolvimento e organização dos ecrãs. As janelas mostram a informação relativa aos comandos do autómato e aos comandos executados pelo terminal;
- Leitura de Dados do Autómato: A leitura de dados é automaticamente feita ao autómato através de métodos de comunicação. Neste caso é feito através de *Ethernet*;

- Escrita de Dados no Autómato: As operações executadas no terminal através de comutadores táteis são enviadas para o autómato, também através de *Ethernet*.

Na realização deste trabalho foi utilizada uma consola de modelo NS5-MQ01-V2 da OMRON que permite o desempenho das tarefas referidas (ver Figura 35).

Figura 35 - Consola Interativa.

4.3.2 Configuração no *Cx-Designer*

O *Cx-Designer* é um *software* HMI utilizado para a criação de dados de ecrã para terminais programáveis da série NS da OMRON. Este programa pode observar a operação de dados dos ecrãs desenvolvidos, permitindo ainda ao utilizador a criação, simulação e lançamento de projetos de ecrãs. Na figura 36 é apresentado o ambiente de desenvolvimento deste *software*.

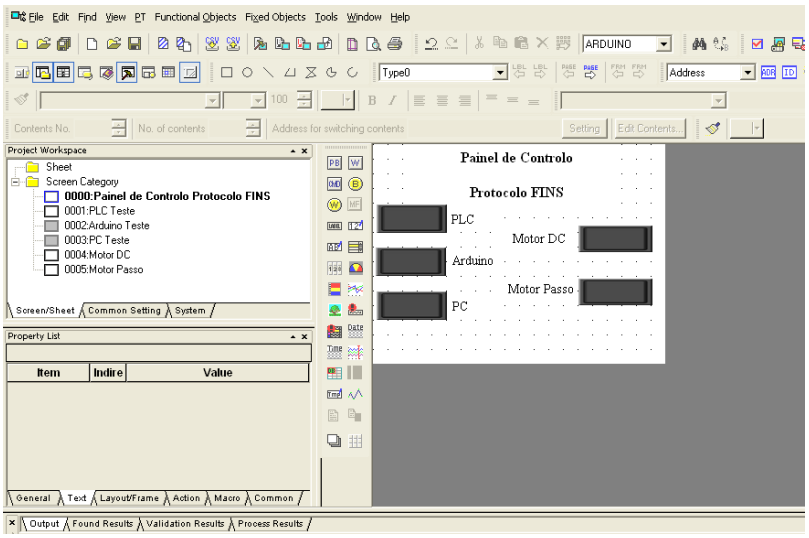
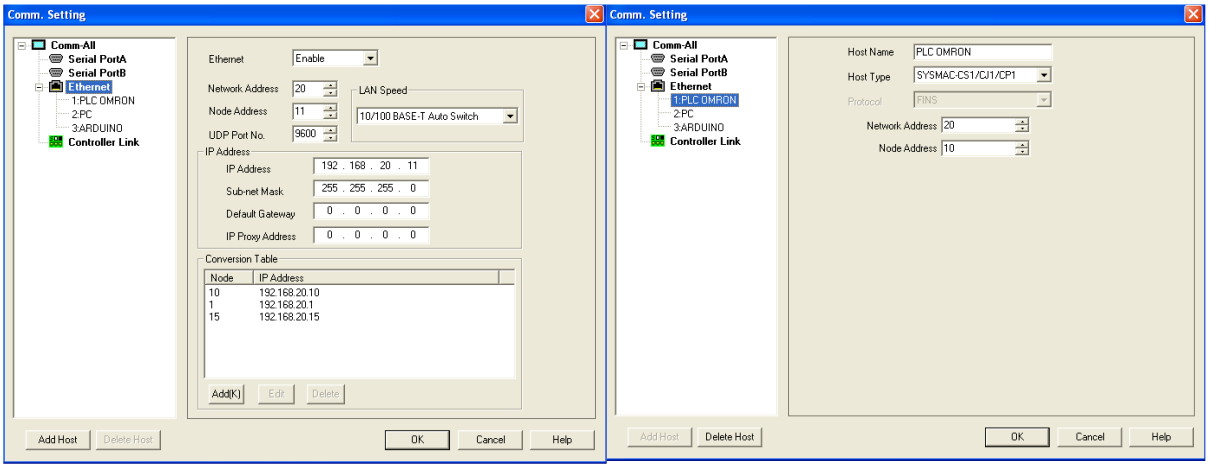


Figura 36 - Exemplo de ecrã desenvolvido em *Cx-Designer*.

Para o terminal poder comunicar com outros equipamentos, é importante estabelecer os parâmetros de comunicação. Como a comunicação pretendida é via *Ethernet* e com recurso ao protocolo FINS, definiram-se os parâmetros de comunicação da consola e os endereços FINS dos equipamentos com os quais a consola poderá comunicar em eventos futuros (ver Figura 37 a) e b)).



a) b)

Figura 37 - a) Configuração de Endereços para a Consola; b) Configuração de Endereços para comunicação com outro equipamento.

A figura 37 a) apresenta a configuração da consola para comunicar na rede *Ethernet*, com os endereços referidos anteriormente. A consola comunica com outro equipamento através do método de conversão de tabela de endereços IP, sendo este indicado no campo *Conversion Table*.

Na figura 37 b) são apresentados os endereços de nó e rede FINS para o qual a consola efetua as operações de leitura e escrita de dados.

4.4 Sumário

Neste capítulo foram descritos os equipamentos OMRON utilizados para a realização deste trabalho, bem como a programação utilizada e as ferramentas que existem atualmente no mercado para o desenvolvimento dos programas. Foi igualmente explicado o processo de configuração de comunicações via *Ethernet* para a utilização da troca de mensagens do protocolo FINS nos equipamentos.

5 Desenvolvimento de Biblioteca em *Linux* para Comunicações FINS

Este capítulo aborda o desenvolvimento de uma biblioteca, de modo a efetuar comunicações com diversos equipamentos e terminais. São explicados os conceitos implementados, assim como o ambiente em que foi desenvolvida, e os modos de funcionamento que foram criados para a interligação com os vários equipamentos.

5.1 Enquadramento

A biblioteca desenvolvida foi projetada para ambiente *Unix* (*Linux*), de modo a implementar o protocolo FINS para comunicação com diversos equipamentos: sistemas microprocessadores, terminais, autómatos, entre outros. A biblioteca foi escrita em linguagem de programação C e compilada com recurso ao seguinte conjunto de ferramentas:

- GCC (*GNU Compiler Collection*): distribuição integrada de compiladores para diversas linguagens de programação [38];
- *Ubuntu* (*Linux*): Sistema operativo *Linux* onde foi desenvolvida e testada a biblioteca;
- GNU *Make*: ferramenta que controla a geração de programas executáveis, através de ficheiros de programas previamente desenvolvidos pelo utilizador [39].

5.2 Conceitos Implementados

5.2.1 Biblioteca Estática

No desenvolvimento de programas, estes são sempre ligados por uma ou mais bibliotecas [40]. Estas são uma importante componente para a utilização de uma ou várias funções e variáveis no desenvolvimento da estrutura de um programa. No programa desenvolvido, para a ligação das bibliotecas de modo a gerar no final o executável da plataforma FINS foi utilizada a biblioteca estática, a qual corresponde a um arquivo no qual são guardados ficheiros objeto.

Na compilação de um programa, caso seja fornecida uma biblioteca estática ao mesmo, este pesquisa os ficheiros objeto necessários, extrai-os, e vincula-os ao programa desenvolvido pelo utilizador. Estas bibliotecas são criadas através do comando "ar" e a extensão destes ficheiros é um ".a" [40].

5.2.2 Makefile

A compilação e *linkagem* de códigos fonte é uma tarefa árdua quando são utilizados diversos ficheiros fonte e bibliotecas para gerar o programa. Um meio de tornar simples este processo é recorrendo à utilização do *Makefile*.

O *Makefile* é um ficheiro que auxilia na compilação e ligação dos diversos ficheiros, gerando um executável para correr o programa. Este é chamado através do utilitário *make*, um programa criado para ler arquivos *Makefile* [41].

O *Makefile* é organizado com base num grupo de regras de compilação em termos de objetivos (caso dos executáveis), e dependências (por exemplo ficheiros fonte e objeto). Em cada objetivo, o comando *make* verifica as dependências correspondentes e decide se os objetivos precisam de ser compilados novamente [42].

Na construção deste ficheiro são utilizadas algumas regras implícitas para que ficheiros de extensão ".o" possam ser adquiridos através da compilação de ficheiros ".c", e um executável seja gerado através da ligação de ficheiros ".o". Estas regras são descritas por variáveis pré-definidas pelo utilizador, como por exemplo [42]:

- CC: compilador de linguagem C;
- CFLAGS: opções de compilação dos programas em C.

A figura 38 ilustra o *Makefile* desenvolvido para a compilação da biblioteca projetada para comunicações FINS. Esta implementa comandos de compilação de modo a criar os ficheiros objetos das funções que serão implementadas e agrega as mesmas numa biblioteca estática. A completar o processo, os executáveis são gerados com base na biblioteca estática criada e de dois ficheiros que funcionam como ficheiros principais e de ficheiros cabeçalho com extensão ".h".

```
CC=gcc

LIB=./lib
INCLUDE=./include
SRC=./src
OBJ=./bin

LIBFLAGS = -lsoftplc_fins
CFLAGS = -Wall
THREAD = -lpthread
demo:  softplc_fins demo_plc demo_noplc

demo_plc:
     @ echo "A compilar Funcoes..."
     @$(CC)  $(SRC)/demo_plc.c $(CFLAGS) -I$(INCLUDE) -L$(LIB) $(LIBFLAGS) $(THREAD) \
    -o demo_plc

demo_noplc:
     @ echo "A compilar Funcoes..."
     @$(CC)  $(SRC)/demo_noplc.c $(CFLAGS) -I$(INCLUDE) -L$(LIB) $(LIBFLAGS) $(THREAD) \
    -o demo_noplc

softplc_fins:
    @$(CC) -c $(SRC)/sockets_fins.c $(CFLAGS) -I$(INCLUDE) -o $(OBJ)/sockets_fins.o
    @$(CC) -c $(SRC)/fins.c $(CFLAGS) -I$(INCLUDE) -o $(OBJ)/fins.o
    @ar -cru $(LIB)/libsoftplc_fins.a $(OBJ)/fins.o $(OBJ)/sockets_fins.o

clean:
    @rm -f demo_plc demo_noplc $(OBJ)/*o $(LIB)/*a
```

Figura 38 – *Makefile* desenvolvido para a biblioteca FINS.

5.2.3 Sockets UDP

A biblioteca projetada para comunicar com os diversos equipamentos que estejam ligados na rede, tem na base da sua comunicação o *Socket*. O *Socket* é um mecanismo que permite a aplicações, programas, entre outros, enviar e receber dados através de uma rede e comunicar com outras aplicações que estejam ligadas na mesma rede [43]. A biblioteca implementa o *Socket* Datagrama, que possibilita o envio de mensagens em ambiente de rede sem efetuar ligação com o equipamento destino. O equipamento origem cria o pacote de dados com a informação da máquina destino e envia para a mesma. Este tipo de *Socket* é semelhante ao sistema de correio postal, e utiliza o protocolo UDP [44].

A figura 39 demonstra um exemplo do modelo cliente/servidor dos *Sockets* UDP.

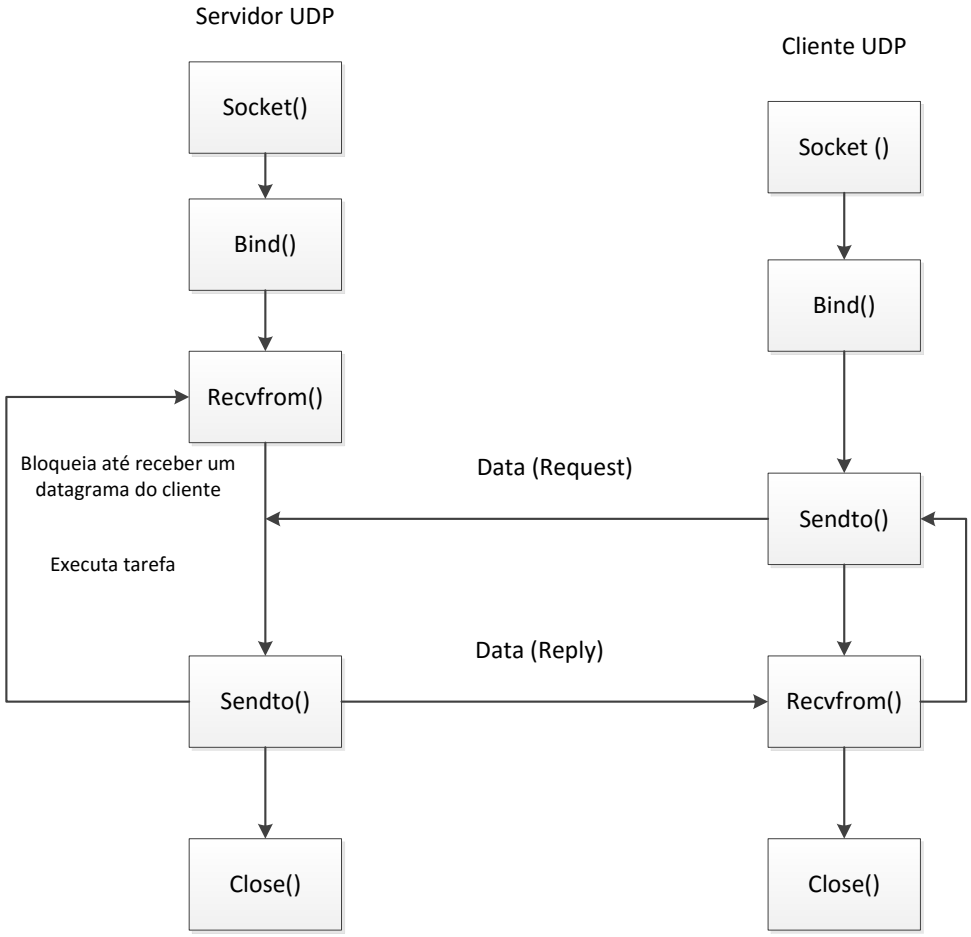


Figura 39 – Modelo cliente/servidor do *Socket* UDP; adaptado de [45].

Para a utilização do modelo cliente/servidor do *Socket* UDP, existe um conjunto de tarefas que devem ser realizadas, sendo estas as seguintes [46]:

- Criação do *Socket*;
- Identificação do *Socket*;
- Envio de mensagem;
- Receção de mensagem;
- Fecho do *Socket*.

Criação do *Socket*:

O *socket* é criado com a seguinte chamada ao sistema:

int s = socket(domínio, tipo, protocolo).

- **Domínio ou Endereço de Família**: trata-se do domínio de comunicação em que o *socket* deve ser criado. Como é o endereço IP que interessa, este campo é ocupado por AF_INET;
- **Tipo**: é o tipo de serviço que é usado conforme as propriedades da aplicação. Como é utilizado o protocolo UDP, este campo é preenchido com o SOCK_DGRAM;
- **Protocolo**: indica um protocolo específico para o apoio às operações do *socket*. Este campo é importante, visto que há nós de rede que utilizam mais que um protocolo de comunicação, e desta forma pode fazer-se a distinção entre eles. Visto que o tipo de serviço a usar é o de *sockets* UDP/IP, este campo é identificado com o parâmetro IPPROTO_UDP. Este campo pode ser igualmente definido com a constante 0, visto que os dois parâmetros anteriores já definem o tipo de *socket* para a comunicação [43], [46].

Em termos de programação, este é um possível exemplo:

```
if ((fd = socket( AF_INET, SOCK_DGRAM,0)) <0){
        perror (“impossível criar socket”);
        return -1;
```

Identificação do *Socket*:

Quando se fala em identificação/nomeação de um *socket*, é o mesmo que dizer que se atribuí um endereço de transporte ao *socket*, ou seja, uma porta na rede IP. A operação usada para este efeito é conhecida como *binding an address*. Segue a mesma analogia que uma chamada telefónica, ou seja, para falar com uma pessoa é preciso o número de telefone para conseguir colocá-la em comunicação com a outra pessoa. No caso do cliente UDP, o *bind* é opcional.

O endereço de transporte é definido através de uma estrutura de endereços *socket*, visto que os *sockets* trabalham com diferentes tipos de comunicação. Como o sistema usado é um UNIX, o *bind* permite especificar o endereço local do *socket*.

O sistema de chamada *bind* é criado da seguinte forma:

*int bind(int socket, const struct sockaddr *address, socklent_t address_len);*

O primeiro parâmetro é o *socket* que foi criado anteriormente.

O segundo parâmetro é uma estrutura genérica *sockaddr* que permite ao sistema operativo a leitura de dados identificados no endereço de família. O endereço de família determina os campos relevantes para o tipo de comunicação que se pretende usar.

Para a rede IP, é usada a estrutura sockaddr_in:

```
struct sockaddr_in {
        __uint8_t       sin_len;
        sa_family_t     sin_family;
        in_port_t       sin_port;
        struct  in_addr sin_addr;
        char            sin_zero[8];
};
```

Desta estrutura, pretendem usar-se três campos:

- **sin_family**: é o endereço de família utilizado quando se define o *socket*, no caso presente usou-se o AF_INET;
- **sin_port**: O número de porta (endereço de transporte), atribuída pelo sistema para efetuar a comunicação;
- **sin_addr**: Trata-se do endereço IP usado no *socket*. Em alguns casos pode usar-se um endereço especial, 0.0.0.0, que é definido na constante INADDR_ANY, sendo a utilização desta constante vantajosa, caso a máquina possua mais que um endereço IP (Ex:router), [46], [47], [48].

O terceiro parâmetro especifica o comprimento da estrutura, ou seja, trata-se do tamanho da estrutura de endereço, *sizeof(struct sockaddr_in)*.

O código para o *bind* é feito da seguinte forma:

```
memset((char *)&wsaddr, 0, sizeof(wsaddr));
myaddr.sin_family = AF_INET;
myaddr.sin_addr.s_addr = htonl(INADDR_ANY);
myaddr.sin_port = htons(UDP_PORT);
if (bind(fd, (struct sockaddr *)&wsaddr, sizeof(wsaddr)) < 0) {
        perror("bind falhou");
        return -1;
}
```

Envio de Mensagem:

Como o *socket* UDP usa um protocolo sem ligação, é possível enviar a mensagem sem estabelecer uma ligação com o servidor. Visto que não existe ligação, a mensagem tem que ser

endereçada para o destino através do endereço IP do servidor. Deste modo, é utilizado o comando *sendto* que permite definir o destino para qual se deve enviar a mensagem.

O endereço é identificado através da estrutura *sockaddr*, tal como no tópico anterior.

O código *sendto* é estruturado desta forma:

*int sendto(int socket, const void *buffer, size_t length, int flags, const struct sockaddr *dest_addr, socklen_t dest_len).*

O primeiro parâmetro, *socket*, é criado através do sistema de chamadas *socket* e identificado através do *bind*. O segundo parâmetro, *buffer* fornece a mensagem a transmitir. O campo *length* é o comprimento de dados que é enviado. O campo *flag* assume o valor zero, visto que não é usado em *sockets* UDP.

O parâmetro *dest_addr* identifica o endereço de destino e o número da porta para enviar a mensagem. Utiliza a mesma estrutura *sockaddr_in* que é usada para a identificação do *socket*.

O parâmetro *dest_len* define o comprimento da estrutura do endereço, usando a configuração *sizeof(struct sockaddr_in)* [46].

O código é definido da forma seguinte:

```
memset((char*)&destaddr, 0, sizeof(destaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(UDP_PORT);
if(inet_aton( server,& destaddr.sin_addr)==0){
        fprintf(stderr,"inet_aton falhou\n");
        exit(1);
}
if(sendto(fd,msg,sendlen,0,(struct sockaddr *)&destaddr,addrlen) ==1{
        perror("sendto falhou");
        exit(1);
}
```

Receção de Mensagem:

A receção de mensagem é conseguida através da chamada à função de sistema, *recvfrom*. Este método é usado para obter o datagrama de resposta de um determinado endereço de transporte, ou seja, endereço IP e número de porta. No caso do servidor UDP, este comando é executado até receber alguma mensagem de um cliente.

O *recvfrom* utiliza a seguinte sintaxe:

*int recvfrom(int socket, void *restrict buffer, size_t length, int flags, struct sockaddr *restrict src_addr, socklen_t *restrict *src_len).*

O parâmetro *socket* é o campo que foi criado anteriormente. O *buffer* disponibiliza os dados recebidos do servidor. O *length* é o comprimento de dados que se podem receber do servidor. O parâmetro *flag* não é utilizado em s*ocket* UDP e toma o valor zero.

O parâmetro *src_addr* é o ponteiro alocado na estrutura *sockaddr* para guardar a identificação da origem da mensagem. Por fim, o *src_len* é o comprimento desta estrutura.

Em termos de código:

*recvlen = recvfrom(fd, resp,MAX_MSG,0, (struct sockaddr *)&origaddr, &addrlen);*

Fecho do *Socket*:

Este campo é responsável pelo fecho do *socket* quando o cliente ou servidor UDP já concluíram as tarefas pretendidas.

O fecho do *socket* é realizado pela função *close*():

int close(int socket).

O *close* inicia todas as ações necessárias para encerrar as comunicações e desaloca qualquer recurso que esteja associado ao *socket* [43].

5.2.4 Comandos FINS

O protocolo FINS apresenta uma grande variedade de comandos que possibilitam efetuar diversos acessos a um equipamento. No desenvolvimento desta ferramenta foram implementados comandos que permitem a leitura e escrita de dados nas várias memórias dos equipamentos (ver anexo B), nomeadamente: *Memory Area Read*, *Memory Area Write*, *Memory Area Fill*, *Multiple Memory Area Read*, *Multiple Area Transfer*, *Forced Set/Reset*.

Estes comandos são configurados através de sistema de numeração hexadecimal.

Memory Area Read:

Este comando permite a leitura de conteúdos de forma consecutiva das áreas de memória [49]. A figura 40 apresenta um exemplo da utilização deste código de comando, onde é enviado uma

mensagem FINS a consultar o valor que se encontra presente na *word* CIO 0, especificando que só tem interesse naquele valor (apenas ler uma *word*).

A trama resposta é devolvida com o respetivo valor solicitado, neste caso 9. Este comando permite a leitura de memórias em formato de *word* e *bit*.

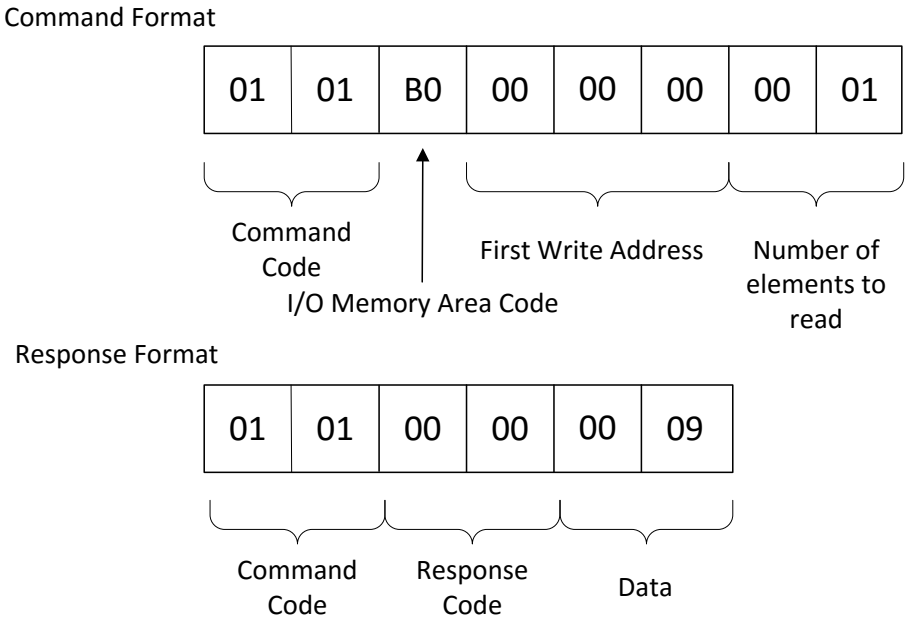


Figura 40- Exemplo de comando e resposta *Memory Area Read*.

Memory Area Write:

Este comando é responsável pela escrita de conteúdos num formato consecutivo nas áreas de memória [49]. Na figura 41 é possível observar um exemplo deste comando, onde o utilizador pretende alterar o estado lógico do bit 0 da memória CIO 0.0, sendo pretendido só a alteração de um bit. A resposta é composta pelo código de comando e de resposta, onde o valor 0000 significa que o comando foi executado com sucesso. Este comando permite a escrita de memórias em formato *word* e *bit*.

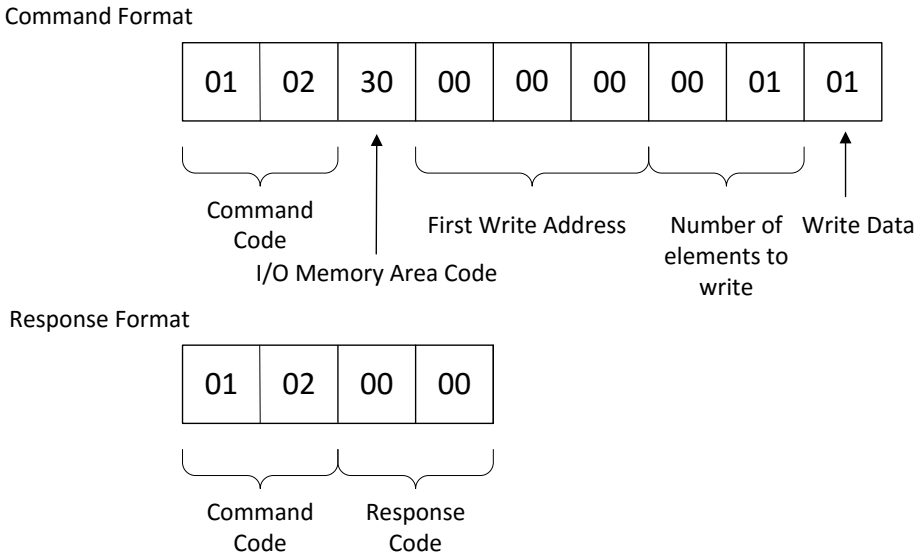


Figura 41 - Exemplo de comando e resposta *Memory Area Write*.

Memory Area Fill:

Este comando permite a escrita dos mesmos dados de forma consecutiva numa área de memória, sendo somente utilizado para escrita no formato *word* [49]. A figura 42 demonstra um exemplo onde é escrito o valor 0020 em hexadecimal nas áreas de memória DM, iniciando no registo 0 e terminando a escrita no registo 2. A resposta é composta pelo código de comando e de resposta, tal como o comando anterior. Este comando permite a escrita de dados nas memórias em formato *word*.

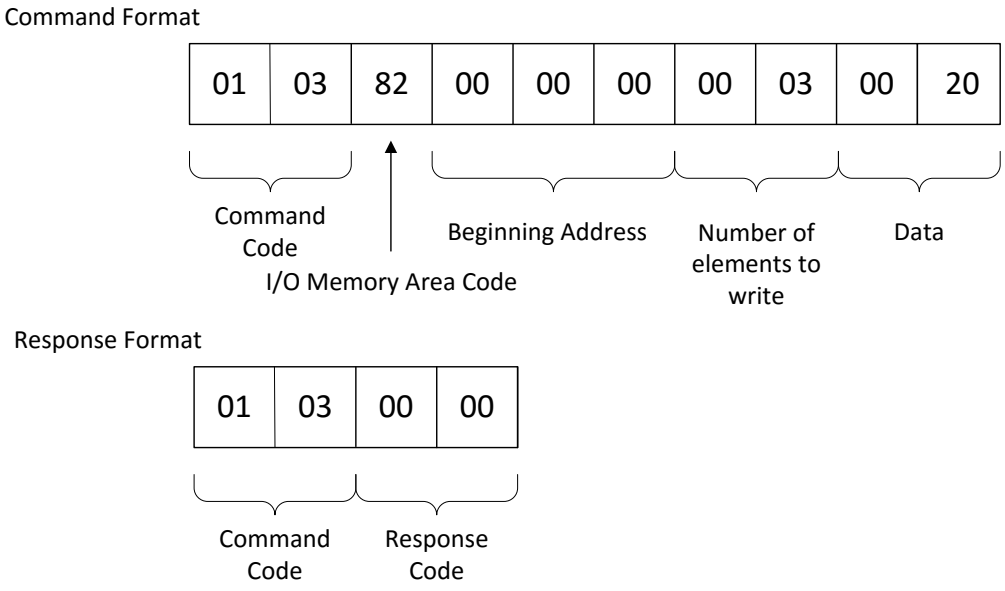


Figura 42 - Exemplo de comando e resposta *Memory Area Fill*.

Multiple Memory Area Read:

Este comando permite a leitura de um conjunto de áreas de memórias de uma forma não consecutiva [49]. Na figura 43 é apresentado um exemplo deste comando, onde é efetuada a leitura das áreas de memória DM 0000 e CIO 0000. Na resposta é possível visualizar os dados que foram lidos das respetivas memórias. Este comando permite a leitura de dados das memórias em formato *word* e *bit*.

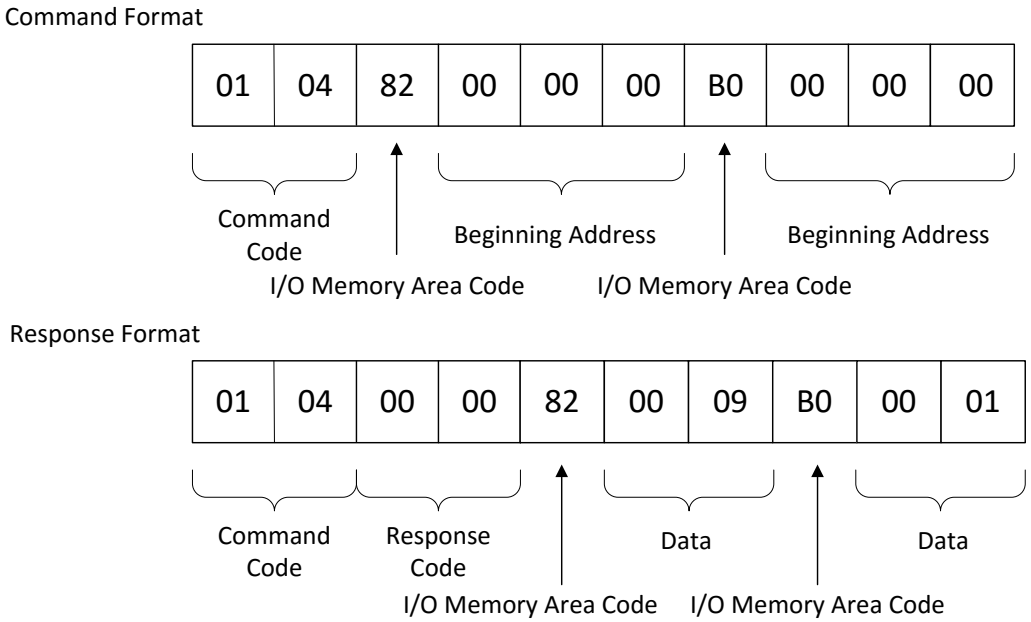


Figura 43 - Exemplo de comando e resposta *Multiple Memory Area Read*.

Memory Area Transfer:

Este comando permite copiar e transferir o conteúdo de um endereço de memória para outra área de memória de forma consecutiva [49]. Na figura 44 é visualizado um exemplo deste comando, onde é transferido o valor da CIO 0000 para os endereços de memória 0000 e 0001 da memória DM. Este comando permite a escrita em memórias em formato *word*.

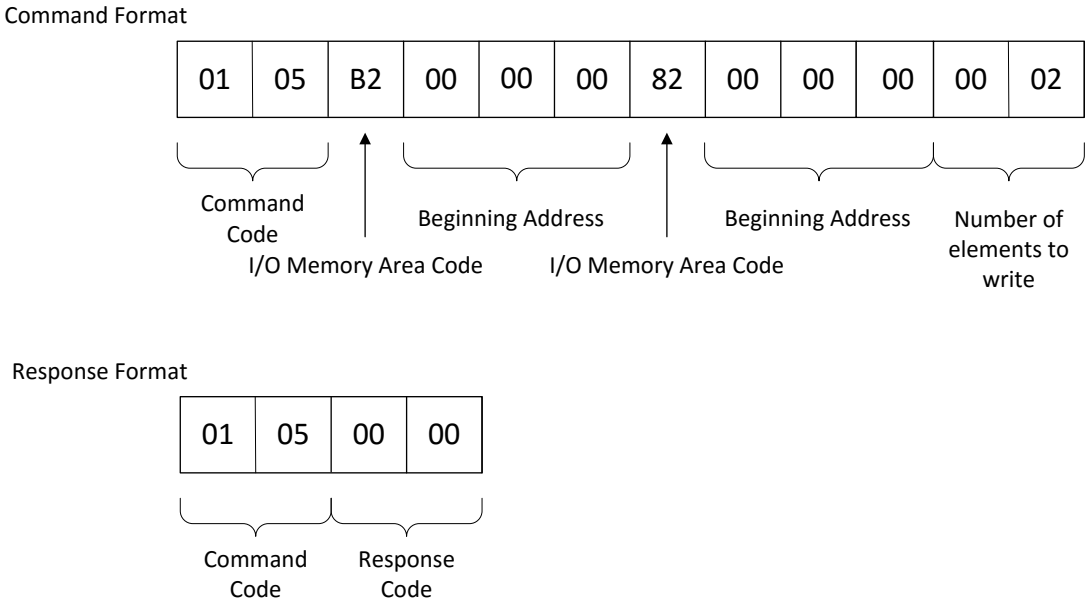


Figura 44 - Exemplo de comando e resposta do comando *Memory Area Transfer*.

Forced Set/Reset:

Este comando força a alteração do estado do bit de uma área de memória. Este valor pode ser colocado a ON/OFF (1 ou 0) [49]. Na figura 45 é possível visualizar um exemplo deste comando, onde o *bit* 15 da área de memória CIO 0025 é colocado a 1.

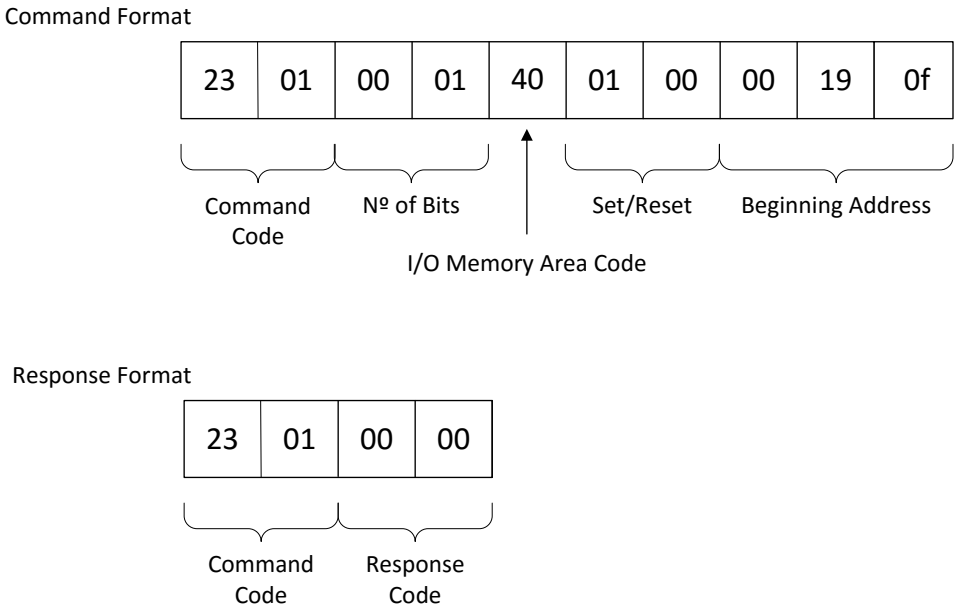


Figura 45 – Exemplo de comando e resposta *Forced Set/Reset.*

5.2.5 Threads

A plataforma desenvolvida tem que estar preparada para a receção de pacotes de dados dos diversos equipamentos que se encontram ligados à rede *Ethernet*.

Para responder, por exemplo a dois equipamentos que enviem mensagens para a biblioteca e recebam a devida resposta, foi implementado o conceito *Thread*. Uma *Thread* é um mecanismo que permite a um programa ou aplicação a execução de várias tarefas em simultâneo. Na mesma aplicação podem ser executadas várias *Threads* de forma independente, e todas elas partilham a mesma memória global [50]. Quando um sistema operativo *Linux* inicia um programa, é criado um novo processo que gera uma *Thread* simples. Esta *Thread* é responsável por correr o programa sequencialmente, podendo criar *Threads* adicionais. Estas *Threads* são executadas dentro do mesmo programa, no mesmo processo, mas cada *Thread* pode desempenhar uma tarefa diferente em qualquer altura [40].

O GNU implementa a norma *POSIX Thread API*, conhecida também por *pthread*, e todas as funções e variáveis *thread* encontram-se declaradas no ficheiro cabeçalho <*pthread.h*> [40].

Quando se utiliza este mecanismo, há duas tarefas essenciais a realizar:

- Criação da *Thread*;
- Terminação da *Thread.*

Criação da Thread:

Na execução de um programa, o processo que resulta é constituído por uma *thread* única, sendo esta a *thread main*. Para criar mais *threads* é utilizada a função *pthread_create()* [50]:

*int pthread_create (pthread_t *thread, const pthread_attr_t *attr, void *(*start)(void *), void *arg);*

- O primeiro parâmetro **thread* é um ponteiro para a variável *pthread_t*, no qual o número de identificação da nova *thread* é armazenado [40];
- O segundo parâmetro *attr é um ponteiro para o objeto atributo *thread*. Este objeto controla os detalhes de como a *thread* irá interagir com o resto do programa. Caso este parâmetro seja definido com *NULL*, a *thread* é criada com atributos padrão;
- O terceiro parâmetro é um ponteiro para a função *thread* que será realizada [40];
- O último parâmetro **arg* é um argumento que é enviado para dentro da *thread*. Este parâmetro é definido como *void** para ser possível a passagem do ponteiro para qualquer tipo de objeto no início da função. Caso seja necessário a passagem de

diversos argumentos, é possível o envio dos argumentos em formato de estrutura com os campos separados [50].

Terminação da *Thread*:

- Para terminar a execução da *thread* criada, é utilizada a função *pthread_exit()*. Esta função termina a *thread* criada, retornando um valor que pode ser obtido pela função *pthread_join()*.
- *void pthread_exit (void *retval);*
- Esta função é equivalente ao comando *return*, e pode ser chamada de qualquer função que tenha inicializado uma *thread*.
- O parâmetro *retval* define o valor a devolver a função que criou a *thread*. Este valor não deve ser definido na pilha de *thread*, visto que o conteúdo é desconhecido após a terminação desta. [50].

A figura 46 apresenta um exemplo de execução de *threads* dentro dum programa.

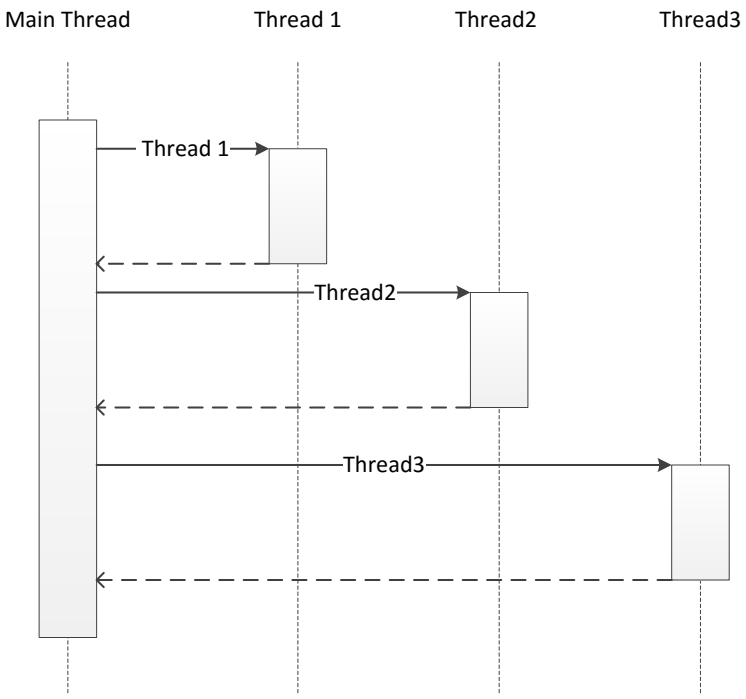


Figura 46 - Exemplo de execução de *Threads* dentro dum programa.

5.3 Funcionalidades implementadas na biblioteca FINS

5.3.1 Enquadramento

A interligação com os diversos equipamentos em ambiente industrial é uma prática fundamental para o controlo dos ativos físicos na instalação. De modo a implementar supervisão e controlo sobre os vários equipamentos recorre-se a protocolos de comunicação para a troca de informação, consulta de informação, alteração do estado destes ativos. A biblioteca FINS permite pôr em prática as tarefas referidas, sendo possível a sua execução através dos modos de funcionamento desenvolvidos. Os respetivos modos são: interativo, servidor e PLC. A figura 47 demonstra um esquema simplificado das funcionalidades da biblioteca FINS. De salientar que esta biblioteca utiliza uma estrutura de memórias semelhante às memórias utilizadas pelos equipamentos OMRON, sendo apresentadas no anexo C as que foram utilizadas na biblioteca FINS. No anexo D são apresentadas algumas das funções desenvolvidas para a implementação dos modos de funcionamento desta biblioteca.

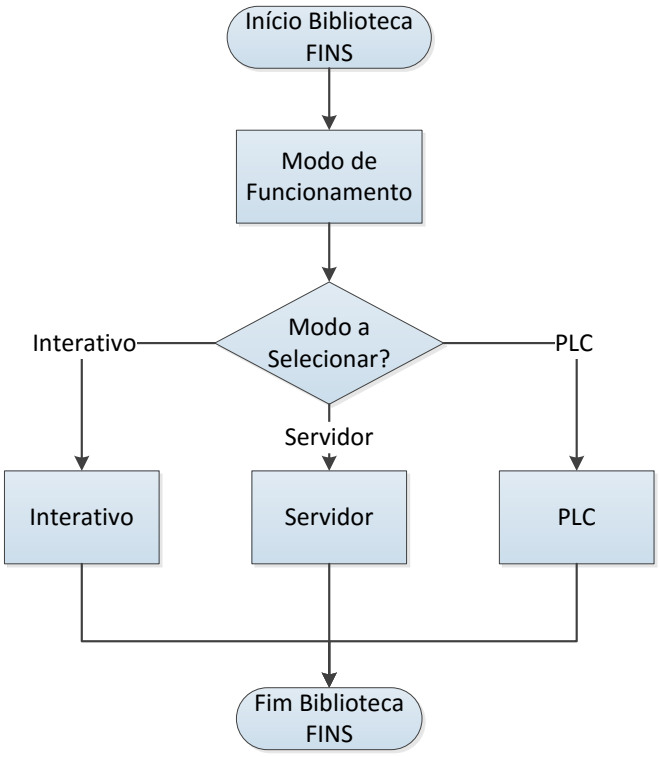


Figura 47 - Esquema simplificado do funcionamento da Biblioteca FINS.

5.3.2 Modo Interativo

Este modo de funcionamento foi criado com o intuito de permitir ao utilizador a consulta de dados e escrita de dados nas áreas de memória dos diversos equipamentos que permitam a comunicação FINS. O desenvolvimento do modo interativo foi a primeira etapa na criação desta biblioteca, de forma a entender o acesso aos dados de um equipamento que funcione como servidor. A figura 48 apresenta um diagrama a explicar o funcionamento deste modo de funcionamento.

O programa ao ser iniciado apresenta um menu, no qual o utilizador pode escolher as opções para trabalhar sobre o ficheiro com a informação dos equipamentos que estão ligados na rede ou a opção comandos FINS.

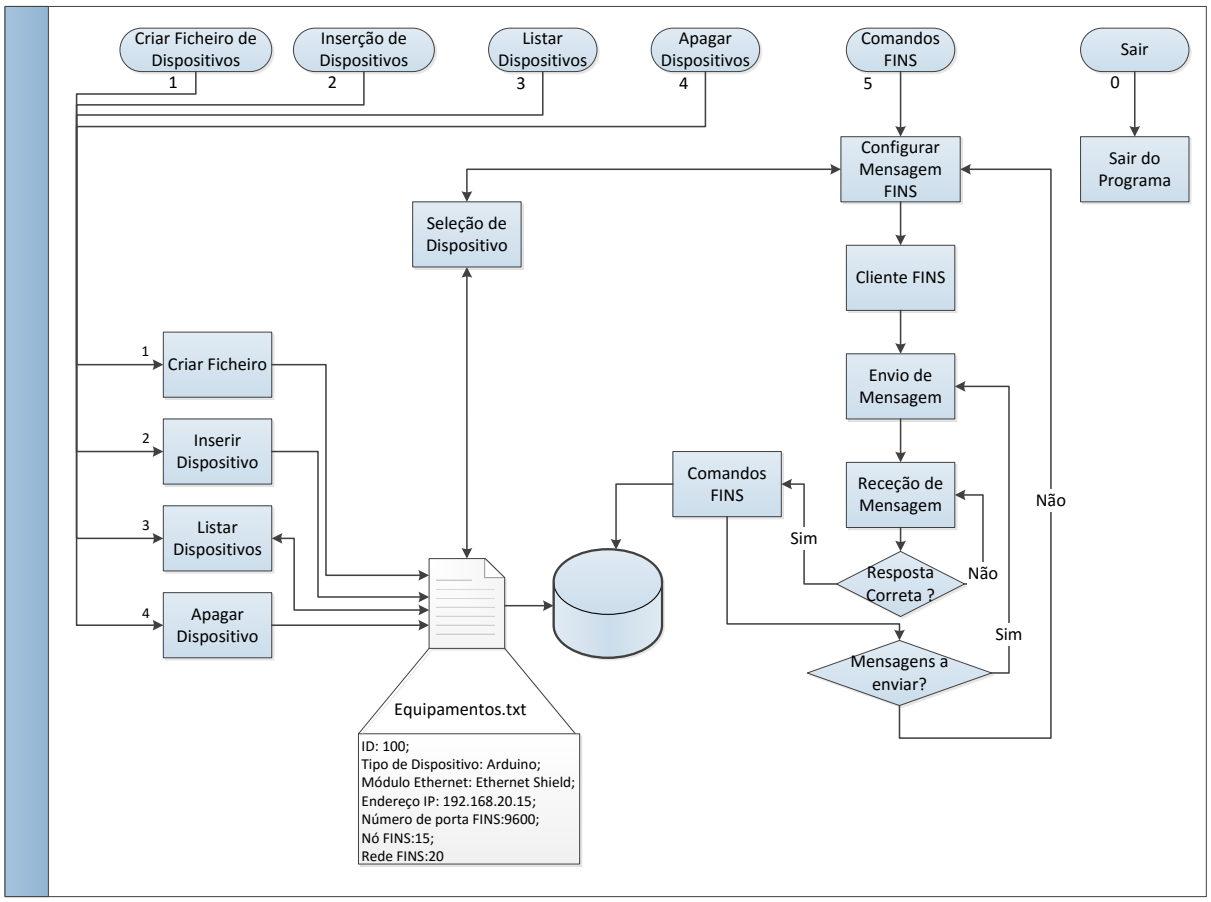


Figura 48 – Diagrama de funcionamento do Modo Interativo da Biblioteca FINS.

As opções para trabalhar sobre o ficheiro são as seguintes:

- **Criar Ficheiro**: Esta opção é responsável por gerar o ficheiro para armazenar os dados de identificação dos equipamentos na rede;
- **Inserir Dispositivo**: Este campo permite ao utilizador a inserção dos dados de identificação de um equipamento que se encontre ligado na rede em ficheiro. A informação é inserida pela seguinte ordem: Número de Identificação; Tipo de Dispositivo (*Arduino*, PLC,..); Módulo *Ethernet*; Endereço IP, Número de porta FINS, Endereço de nó FINS, Endereço de rede FINS;
- **Listar Dispositivo**: Esta opção disponibiliza ao utilizador todos os equipamentos armazenados em ficheiro em formato de lista;
- **Apagar Dispositivo**: Este campo é responsável por eliminar a informação de um equipamento que já não seja necessária do ficheiro. A seleção do equipamento a eliminar é feita através do seu número de identificação.

Na opção comandos FINS o utilizador tem disponíveis os comandos FINS que poderão ser utilizados na configuração da mensagem FINS. Esta opção permite a utilização dos comandos FINS referidos na secção 5.2.4, com exceção do comando *Forced Set/Reset* que só será utilizado nos restantes modos de funcionamento. Na configuração da mensagem FINS, o utilizador pode selecionar o equipamento com o qual pretende enviar dados FINS através do seu endereço IP. O endereço IP é verificado no ficheiro Equipamentos.txt e, caso exista, os respetivos dados para comunicação FINS são selecionados. Nos comandos FINS de leitura de dados, o utilizador poderá escolher o número de vezes que pretende executar o comando.

Após estas configurações procede-se à comunicação com o respetivo equipamento ligado na rede. A biblioteca FINS procede ao envio da trama FINS para o equipamento destino na rede. Na receção da resposta FINS, a plataforma FINS faz uma comparação entre endereços FINS de origem e destino e dos números de identificação da trama enviada e recebida. Caso os valores não estejam corretos, a mensagem recebida é descartada. Para a resposta correta, os dados são armazenados numa estrutura de áreas de memória. O *socket* aplicado neste modo de funcionamento está implementado como sendo "não-bloqueante", ou seja, se o equipamento destino por alguma razão não esteja ligado à rede Ethernet, o *socket* está preparado para não ficar preso naquele comando, voltando às opções de comando FINS.

5.3.3 Modo Servidor

Este modo de funcionamento foi criado com o objetivo de ter a biblioteca FINS a funcionar como um autêntico servidor, ou seja, permite a receção de dados dos vários dispositivos que se encontrem ligados na rede *Ethernet*, e possibilita a equipamentos de supervisão a consulta dos dados que foram anteriormente transmitidos por outros ativos físicos que se encontrem instalados na rede (ver Figura 49).

O modo servidor implementa o conceito de *threads*, visto ser necessário a receção de vários pacotes de dados de diversos equipamentos e efetuar a resposta aos mesmos. Os únicos equipamentos com possibilidade de acesso à informação do servidor ou transmitir dados para o mesmo são apenas os equipamentos que tenham sido previamente registados em ficheiro no modo interativo.

O modo servidor implementa todos os comandos FINS descritos previamente, permitindo assim a comunicação com equipamentos de execução de tarefas, e com equipamentos de supervisão e controlo desses equipamentos, como por exemplo a consola HMI.

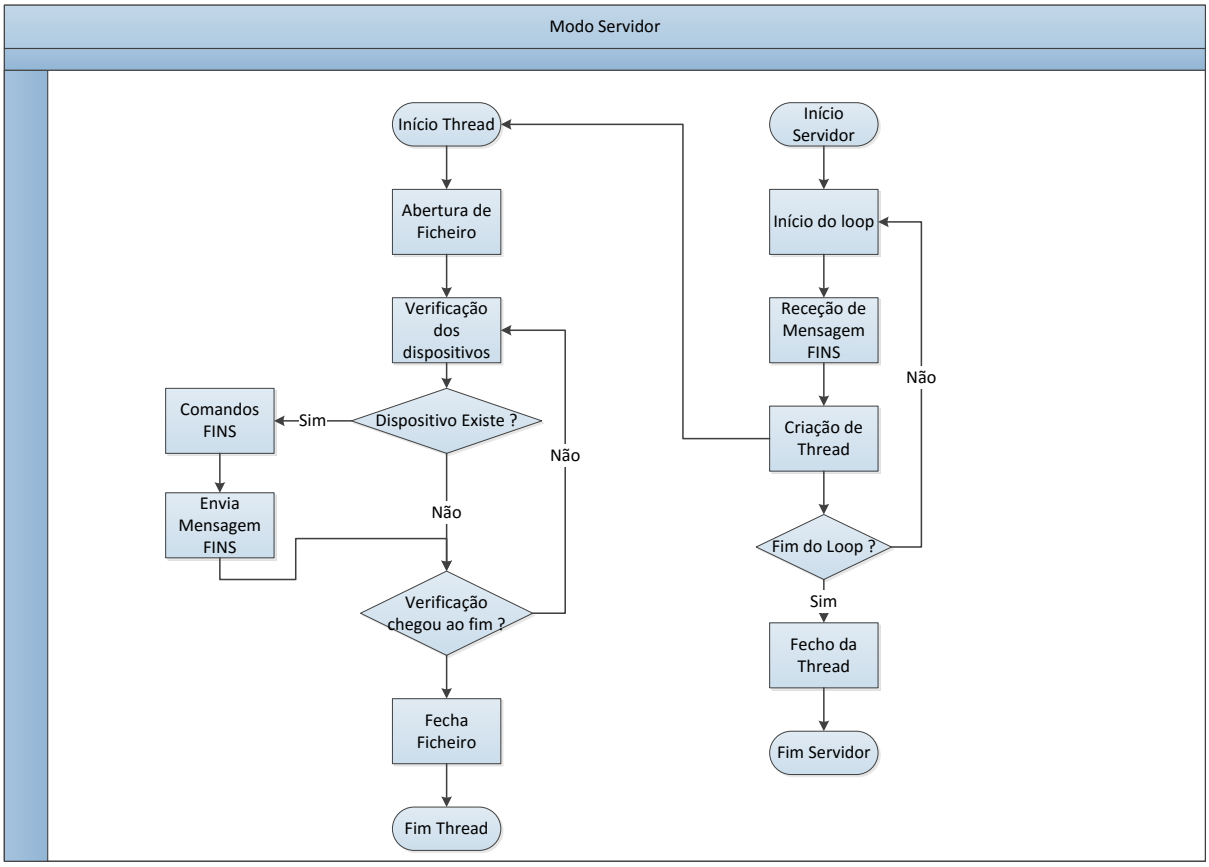


Figura 49 - Diagrama de funcionamento do Modo Servidor da Biblioteca FINS.

5.3.4 Modo PLC

Este modo de funcionamento foi desenvolvido com o intuito de simular o comportamento de um PLC na biblioteca FINS. Neste modo de funcionamento são aplicados os conceitos previamente desenvolvidos no modo interativo e no modo servidor.

Este modo, ao ser iniciado apresenta um menu, no qual o utilizador pode escolher as opções para trabalhar sobre o ficheiro ou a opção do modo PLC.

O ficheiro armazena o endereço IP do equipamento de origem da ligação juntamente com tipo de memória e o endereço da mesma para o qual pode escrever na biblioteca FINS, e os endereços (IP e FINS) do equipamento de destino juntamente com a memória destino e o respetivo endereço.

As opções para trabalhar sobre o ficheiro são as seguintes:

- **Criar Ficheiro**: Esta opção é responsável por gerar o ficheiro para armazenar os dados de ligações que podem ser realizadas;
- **Inserir Comunicação**: Este campo permite ao utilizador a inserção das comunicações que a plataforma FINS poderá executar no modo PLC. A informação é inserida pela seguinte ordem: Número de Identificação; Equipamento de entrada (*Arduino*, Consola,..); Memória de entrada; Endereço de memória de entrada, Endereço IP de saída, Endereço de nó FINS de saída, Endereço de rede FINS de saída, Memória de saída, Endereço de memória de saída;
- **Listar Comunicação**: Esta opção disponibiliza ao utilizador todos as ligações armazenadas em ficheiro em formato de lista;
- **Apagar Comunicação**: Este campo é responsável por eliminar os dados de uma ligação que já não seja necessária do ficheiro. A seleção da ligação a eliminar é feita através do seu número de identificação.

No modo PLC existem um conjunto de etapas definidas que são ativadas mediante o equipamento externo que envie mensagem para a biblioteca FINS. Uma etapa é realizada se a memória responsável pela sua execução for ativada e após a sua realização há a possibilidade de enviar um comando FINS para ativar uma saída noutro equipamento que esteja ligado na rede *Ethernet*. O envio ou receção de mensagem FINS é realizado através de uma *Thread* desenvolvida para efetuar as comunicações com os equipamentos que se encontrem interligados via rede *Ethernet*. Esta *Thread* encontra-se preparada para trabalhar como um *Socket* UDP servidor ou como *Socket* UDP cliente, sendo esta distinção feita mediante a ativação ou não das

etapas de tarefas no modo PLC. Em caso de execução de uma etapa é transferido para a *Thread* o parâmetro a indicar o modo de trabalho *Socket* UDP cliente, a mensagem FINS a enviar ao equipamento destino, e os endereços de comunicação que dizem respeito ao mesmo.

Na ausência de ativação de etapas, o modo PLC envia para a *Thread* um parâmetro a indicar o modo de trabalho *Socket* UDP servidor, ficando à escuta de novos pacotes de dados FINS da rede *Ethernet* para ativação das etapas.

Para distinção entre os equipamentos que têm autorização para ativar as etapas e enviar dados para outros equipamentos, procede-se ao registo das comunicações que poderão ser efetuadas neste modo de funcionamento num ficheiro (ver Figura 50).

Durante a receção de dados como servidor, o modo PLC só guarda os dados recebidos dos equipamentos que tenham sido previamente registados no ficheiro Equipamentos.txt no modo interativo.

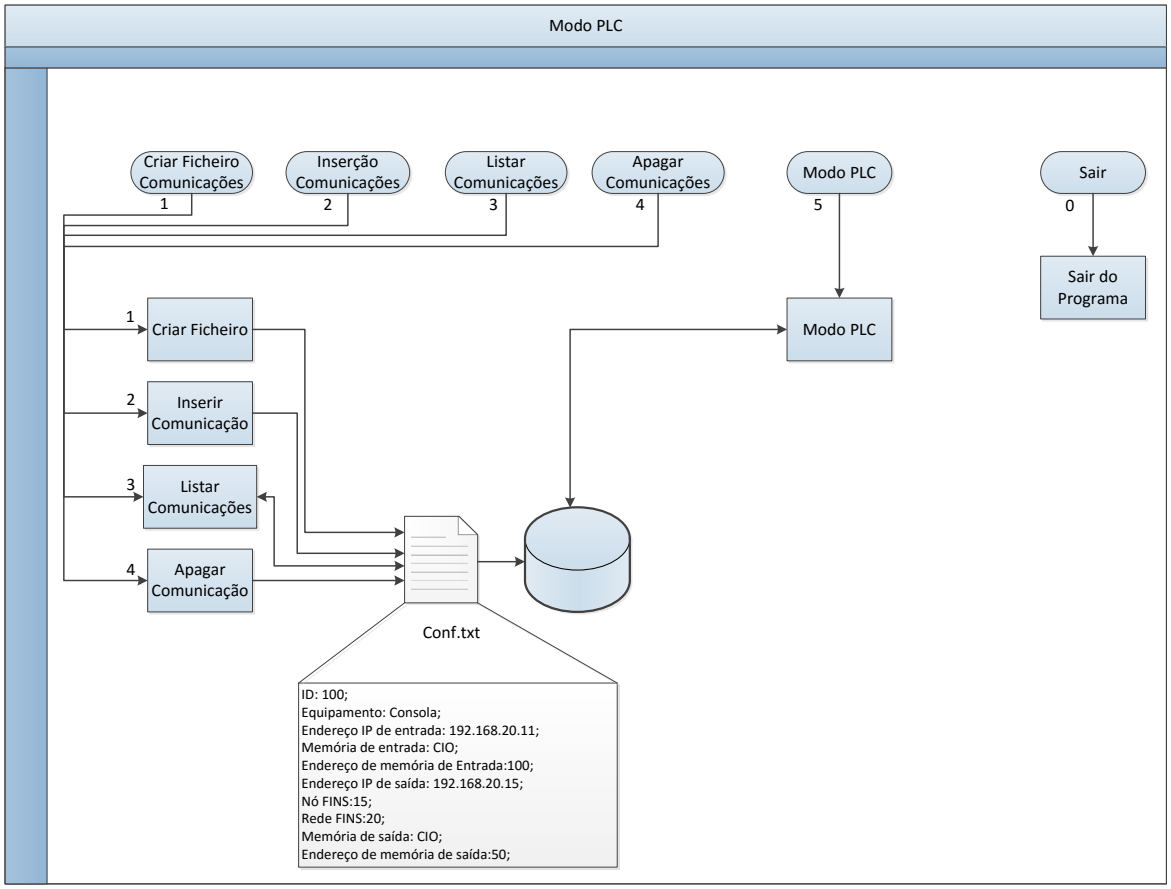


Figura 50 – Diagrama de funcionamento do Modo PLC da Biblioteca FINS.

Resumindo, este modo de funcionamento permite a simulação de um PLC na biblioteca FINS, e possibilita ainda a leitura e escrita de dados de equipamentos que estejam registados.

É possível afirmar então que este modo de funcionamento desempenha funções de servidor, e ainda pode fazer o papel de ponte de ligação de dados entre os equipamentos que se encontrem ligados na rede *Ethernet*, bem como codificar algoritmos de controlo que serão executados como de um programa de autómato se tratasse.

A figura 51 apresenta um diagrama das comunicações do modo PLC.

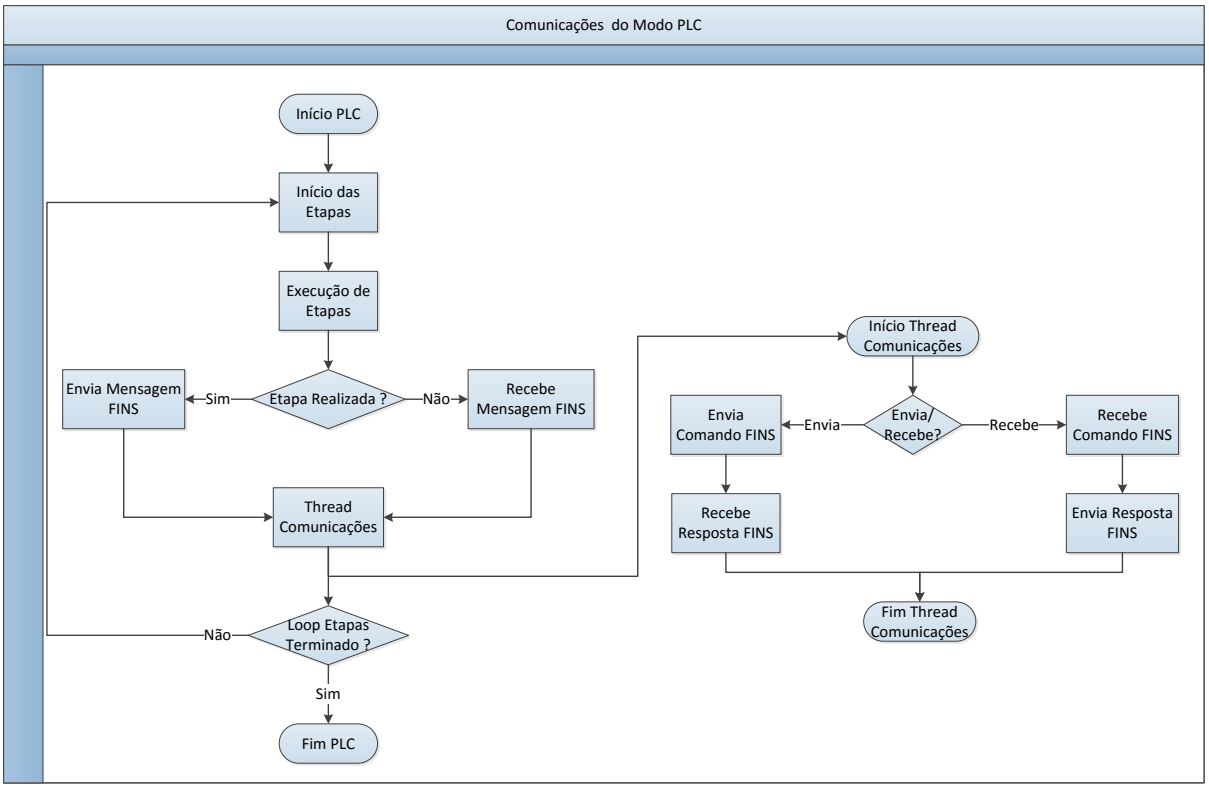


Figura 51 – Comunicações do modo PLC da Biblioteca FINS.

5.4 Sumário

Neste capítulo foi abordado o desenvolvimento da biblioteca FINS, sendo apresentados os conceitos utilizados no desenvolvimento desta plataforma, assim como as funcionalidades que foram desenvolvidas.

6 Desenvolvimento de *Firmware* para Comunicação FINS e Página *Web* em Plataforma *Arduino*

6.1 Enquadramento

Este capítulo aborda o desenvolvimento do *Firmware* necessário para parametrização e comunicação via *Ethernet* com uma plataforma eletrónica *Arduino*, permitindo assim ao utilizador monitorizar e executar tarefas no meio ambiente através deste sistema microprocessador. Para mais informações, consultar o anexo D.

6.2 Plataforma *Arduino*

6.2.1 O que é um *Arduino*?

O *Arduino* é uma plataforma *open-source* de prototipagem eletrónica, baseada em *software* e *hardware* de fácil entendimento para o utilizador, sendo destinado a qualquer pessoa que pretenda aprender a criar ambientes interativos.

Esta plataforma consegue interagir facilmente com o meio ambiente através de receção de sinais, por meio de interruptores, sensores, entre outros dispositivos de medida e mediante esta informação atua sobre os equipamentos, tais como luzes, motores, entre outros atuadores.

O microcontrolador deste tipo de placa é programado por linguagem de programação *Arduino*, sendo esta linguagem baseada em linguagem *Wiring* e o ambiente de desenvolvimento (*Arduino IDE*) baseia-se no ambiente *Processing* [51].

6.2.2 Arduino Mega 2560

Na gama de *Arduinos* disponíveis para a realização deste trabalho optou-se pela utilização do *Arduino Mega* 2560 (ver Tabela 2 e Figura 52). Esta placa é operada a uma tensão de 5 V, podendo ser alimentada via ligação USB ou alimentação externa, e disponibiliza uma corrente DC de 20 mA nos pinos digitais. A nível de entradas e saídas, o *Arduino Mega* 2560 possui 54 pinos digitais que podem ser configurados como entradas ou saídas, sendo que, 15 destes pinos podem trabalhar como saídas PWM e possui 16 entradas analógicas que fornecem uma resolução de 10 bits, ou seja, 1024 valores diferentes.

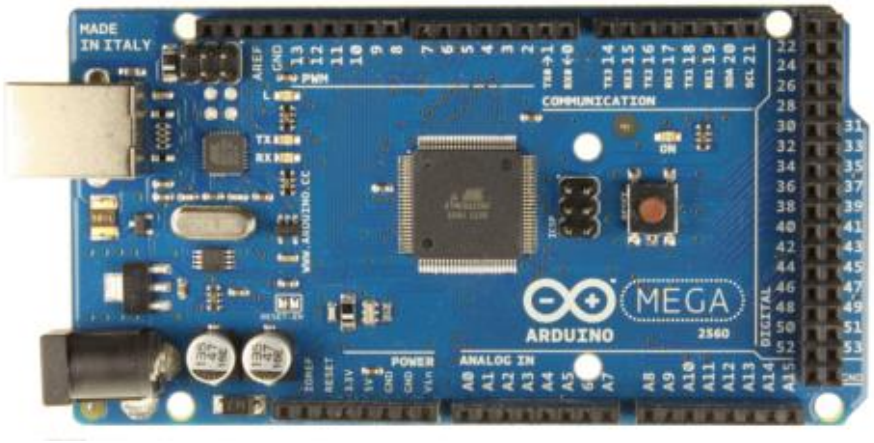


Figura 52 – *Arduino Mega* 2560.

Tabela 2 - Especificações Técnicas do *Arduino Mega*2560.

Especificações Técnicas	
Microcontrolador	**ATmega2560**
Tensão Operacional	**5V**
Tensão de Entrada (recomendada)	**7-12V**
Tensão de Entrada (limite)	**6-20V**
Pinos Digitais I/O	**54 dos quais 15 fornecem saídas PWM**
Pinos Entradas Analógicas	**16**
Corrente DC por Pino I/O	**20 mA**
Corrente DC no Pino 3.3V	**50 mA**
Memória Flash	**256 KB dos quais 8KB são para o bootloader**
SRAM	**8 KB**
EEPROM	**4 KB**
Velocidade Relógio	**16 MHz**
Comprimento	**101.52 mm**
Largura	**53.3 mm**
Peso	**37 g**

A nível de memória, o microcontrolador que este *Arduino* utiliza, o ATmega2560, disponibiliza 256KB de memória *flash* para armazenar o programa desenvolvido, 2KB de SRAM e 4 KB de EEPROM que permitem a leitura e escrita de dados [51], [52].

A nível de comunicação, o *Arduino Mega* 2560 consegue comunicar com computadores, outra placa ou até mesmo outro microcontrolador. O microcontrolador ATmega2560 fornece 4 portas UARTs para comunicação série via TTL. É possível igualmente monitorizar os dados do *Arduino* via ligação USB que fornece uma porta COM virtual que disponibiliza os dados num monitor série que vem incluído no *software* de desenvolvimento desta placa, o *Arduino IDE* [51].

6.2.3 Arduino Ethernet Shield

Tendo em conta o objetivo de efetuar comunicações via *Ethernet* com a placa *Arduino Mega* 2560, é utilizado para este efeito um *Arduino Ethernet Shield* (ver Figura 53).

Esta *Shield* permite efetuar a ligação à *internet* de uma placa *Arduino* em poucos minutos, e baseia-se no *chip ethernet Wiznet* W5100.

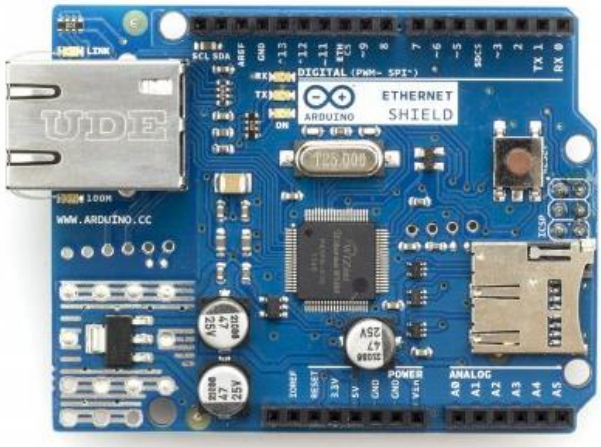


Figura 53 - *Arduino Ethernet Shield*.

O *Wiznet* W5100 fornece a capacidade de comunicação via *ethernet* através dos protocolos TCP e UDP, sendo capaz de suportar 4 comunicações *socket* em simultâneo. Através da biblioteca *Ethernet* é possível escrever o código necessário para a ligação à rede utilizando esta *Shield*.O *Arduino Ethernet Shield* liga-se ao barramento *Ethernet* através de ligação RJ45, e possui ainda um *slot* de cartões micro-SD, que pode ser usado para guardar ficheiros para servir toda a rede [51].

6.3 *Firmware* desenvolvido no *Arduino*

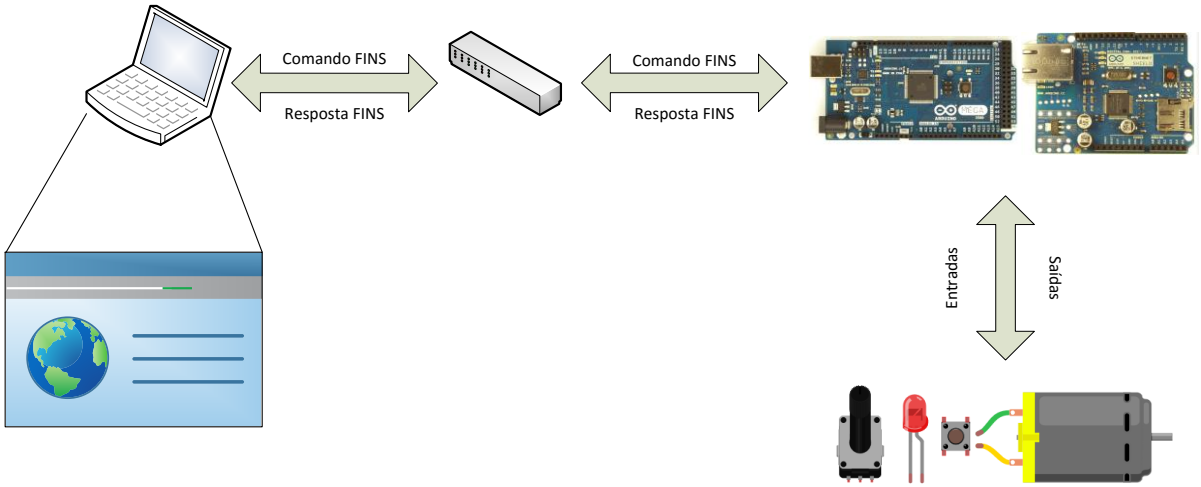


Figura 54 - Diagrama Ilustrativo da Comunicação com o Arduino.

Na figura 54 está presente um diagrama que representa a forma como o utilizador estabelece comunicação com o *Arduino*, e parametrização de entradas e saídas.

O *firmware* desenvolvido para o *Arduino* permite a implementação de um servidor FINS que permite a troca de dados via *ethernet* com outros equipamentos através do protocolo FINS, e um servidor *Web* para o mapeamento de entradas e saídas do *Arduino* e dos endereços de comunicação do *Arduino* via página *web*.

O *firmware* foi projetado com recurso à programação em linguagem C, programação em HTML (*HyperText Markup Language*), e foram utilizadas as bibliotecas que são apresentadas na tabela 3.

Tabela 3 – Bibliotecas utilizadas na programação do *Arduino*.

Biblioteca	Descrição
EEPROM	**Funções para leitura e escrita na memória EEPROM.**
Ethernet	**Funções para o *Arduino* ser compatível com a *Shield Ethernet*.**
SPI	**Funções para comunicação com barramento periférico série.**

6.3.1 Servidor FINS *Arduino*

A figura 55 fornece uma perspetiva do comportamento do *Arduino* no que toca à interação com o *hardware* (Entradas e Saídas) e na receção e envio de tramas FINS.

Quando o *Arduino* entra em funcionamento, existe um conjunto de fatores que são necessários configurar para a boa execução do restante programa: Alocação de memória, Alocação de Endereços de Comunicação, Alocação das Entradas e Saídas.

Na alocação de memória, o *Arduino* configura as memórias utilizadas para o registo dos dados num formato de estrutura. Esta estrutura indica um formato semelhante de algumas memórias utilizadas pelos equipamentos OMRON (CIO,DM e HR) e cada uma permite até 100 registos.

A alocação de endereços de comunicação permite ao *Arduino* definir os seus endereços de comunicação, nomeadamente os endereços *Ethernet* e os endereços FINS, assim como os endereços do equipamento para quem envia mensagem, caso ocorra alteração do estado lógico das suas entradas.

Na alocação de entradas e saídas, esta plataforma eletrónica permite atribuir as entradas e saídas que serão utilizados para interação com o meio ambiente. São utilizadas entradas analógicas, e entradas e saídas digitais.

No programa principal, o *Arduino* encontra-se à escuta de mensagens, e caso receba alguma mensagem FINS, é realizado o processo de descodificação e envio de resposta FINS para o equipamento de origem. Este processo pode ser visualizado na figura 56.

Por último, é feita uma verificação contínua do estado das entradas e saídas do *Arduino* alocadas anteriormente e, em caso de alteração é realizado o processo de leitura/escrita de pinos (ver Figura 57).

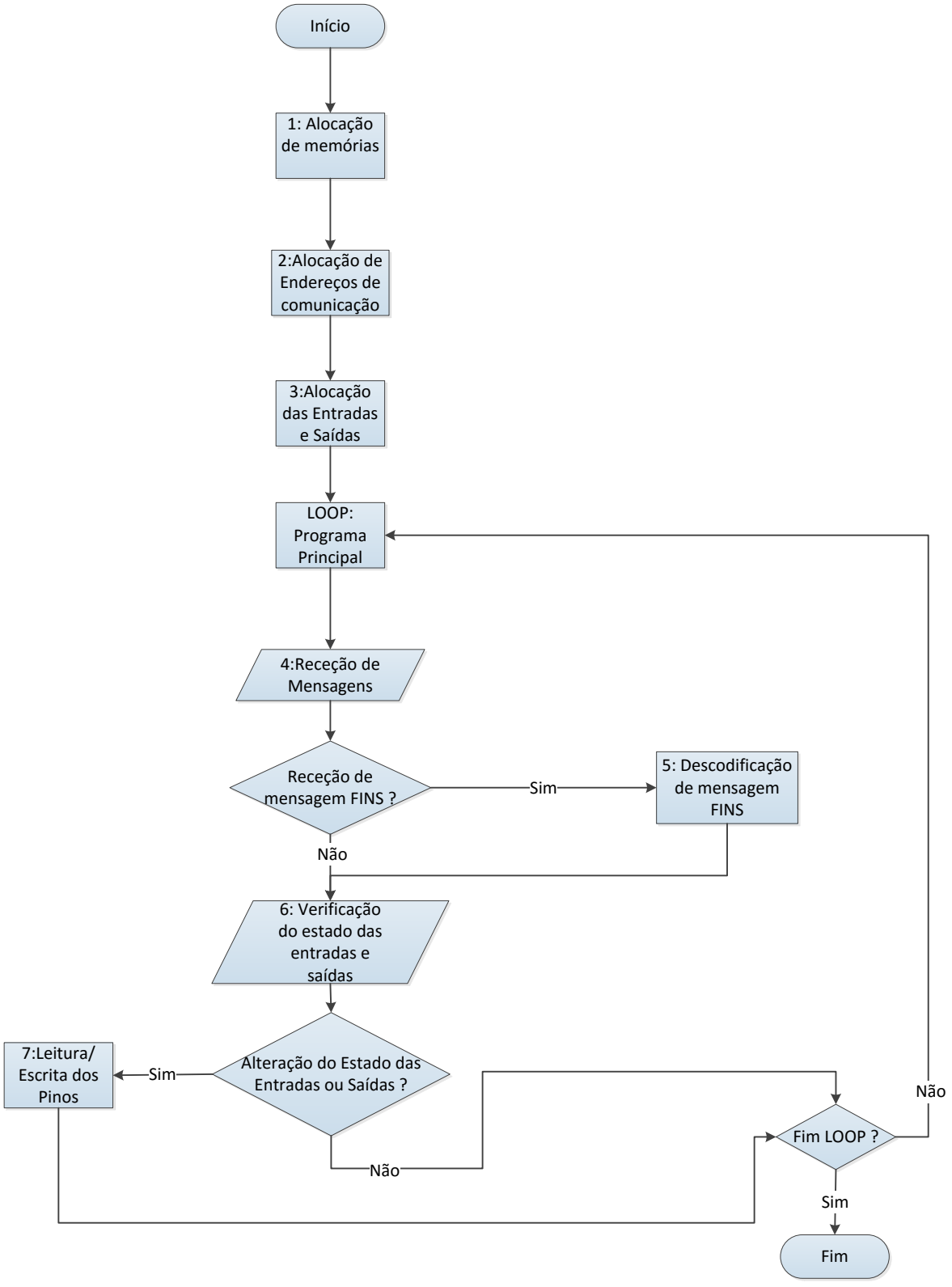


Figura 55 – Esquema simplificado do funcionamento do FINS no *Arduino*. Os pontos 5 e 7 são especificados nas figuras 6.5 e 6.6 respetivamente.

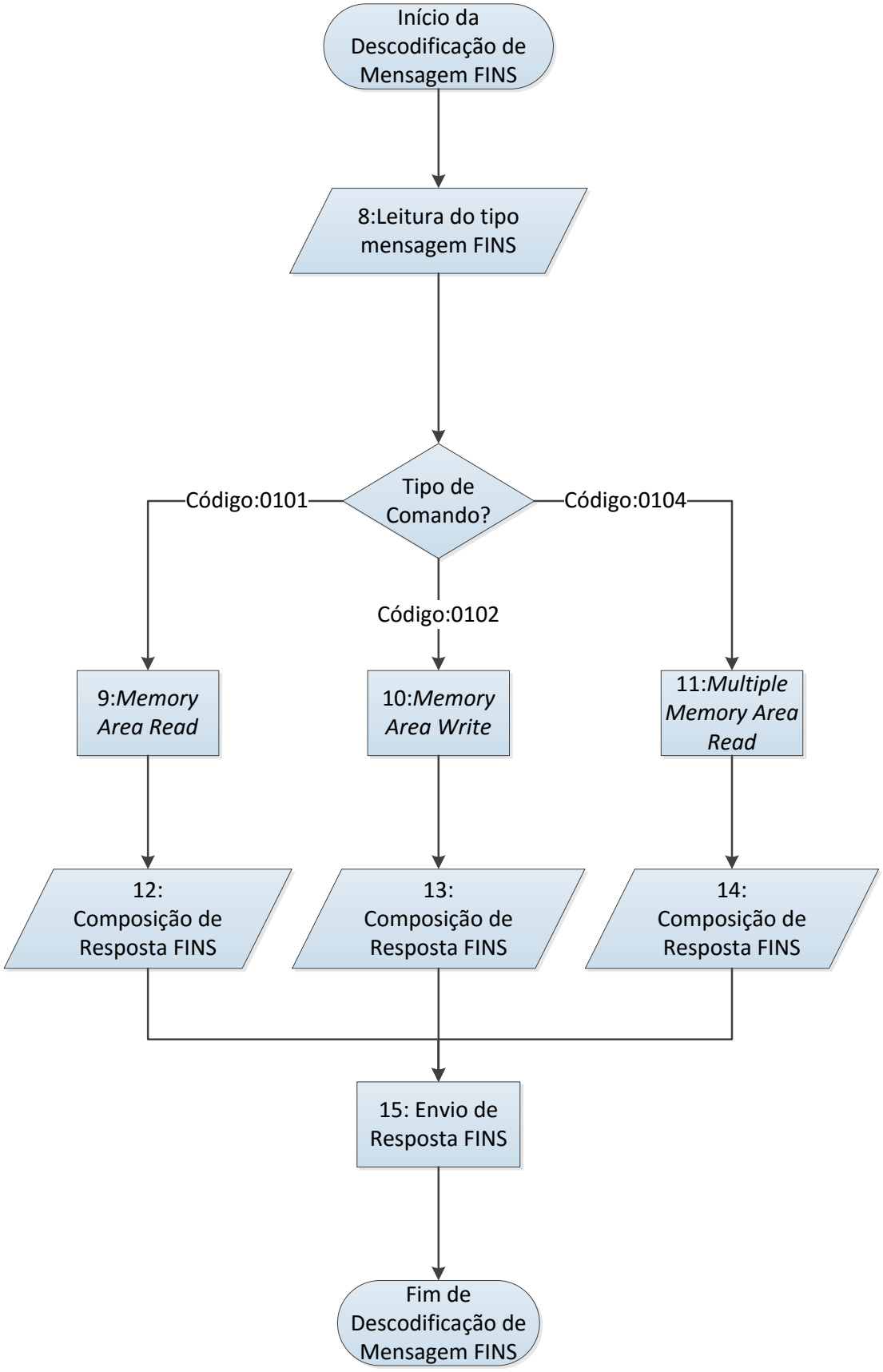


Figura 56 – Descodificação de Mensagem FINS.

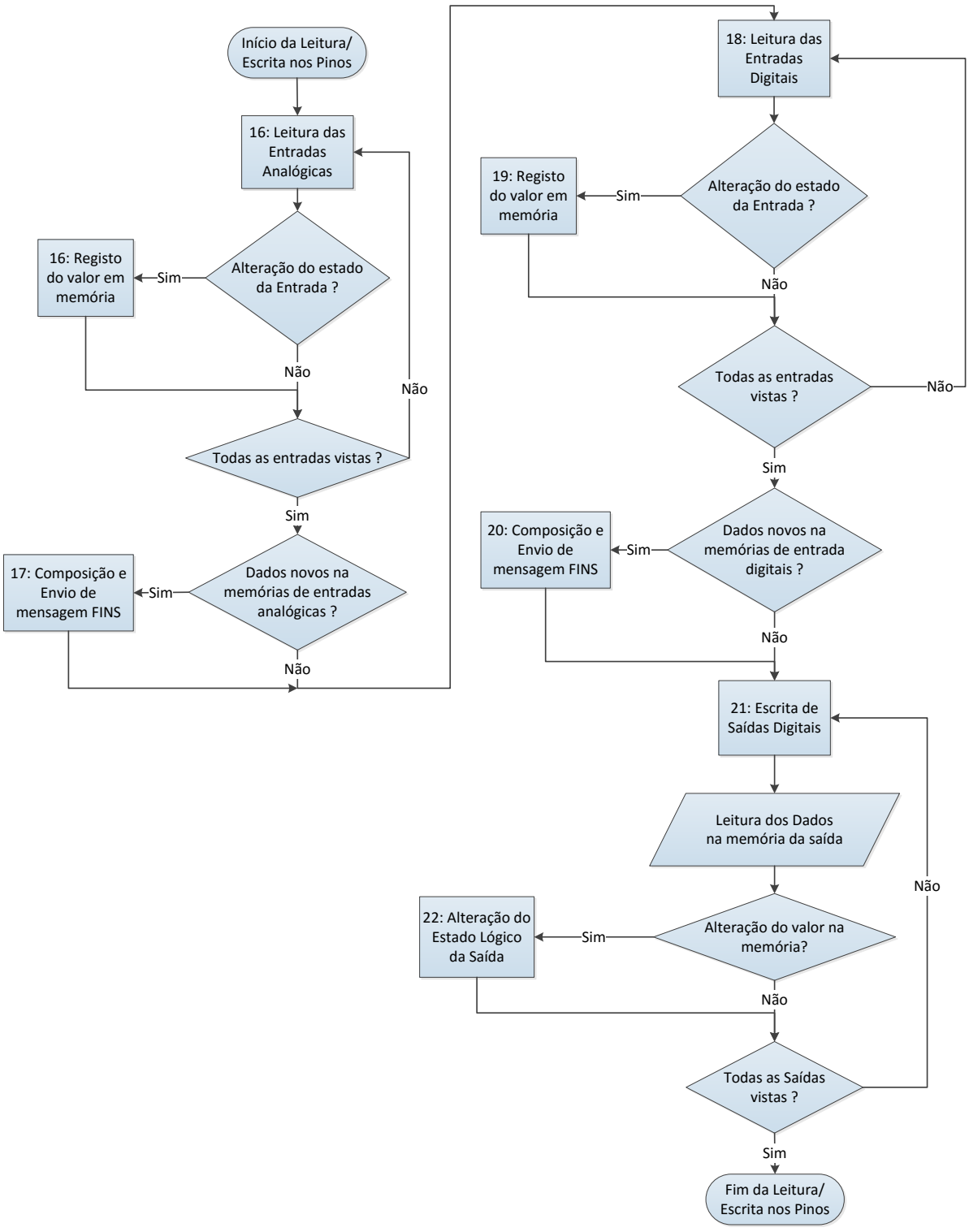


Figura 57 – Leitura ou Escrita dos Pinos do *Arduino*.

Descodificação de mensagem FINS:

Neste processo, o *Arduino* analisa a mensagem FINS recebida via *Ethernet*. Esta plataforma consegue analisar três tipos de mensagens FINS (*Memory Area Read*, *Memory Area Write*, *Multiple Memory Area Read*):

- O *Memory Area Read* permite a um equipamento externo a leitura de um ou mais conteúdos consecutivos de uma memória. Em resposta, o *Arduino* compõe a resposta FINS com todos dos dados solicitados e envia mensagem FINS para o equipamento de origem;
- O *Memory Area Write* é o comando usado para escrita de dados, sendo possível escrever em um ou mais conteúdos de uma memória de forma consecutiva. Em resposta a este comando, o *Arduino* envia uma resposta FINS a informar que o comando FINS foi bem-sucedido;
- O *Multiple Memory Area Read* permite a um equipamento externo a leitura de diversas memórias do *Arduino*, num formato não consecutivo das mesmas. O *Arduino* em resposta FINS coloca os dados solicitados e retorna a informação pedida.

Leitura ou Escrita dos Pinos:

Este processo permite a monitorização dos valores nas entradas analógicas e digitais, assim como permite a alteração do estado das saídas. As entradas e saídas do *Arduino* são mapeadas nos endereços de memória OMRON:

- Nas entradas analógicas foram utilizadas 8 entradas, em que o seu valor varia entre 0 e 1023, sendo estes valores registados em endereços consecutivos da mesma memória;
- Nas entradas digitais e saídas digitais foram utilizados 32 pinos para esse efeito, sendo possível guardar o valor de 16 pinos num endereço de memória em formato *bit* a *bit*, começando no *bit* 0 e terminando no *bit* 15. Estes valores assumem os valores 0 ou 1;
- Nas entradas analógicas e digitais é feita uma verificação de todas as entradas e, caso exista alteração nas mesmas, o valor é registado no respetivo endereço de memória. Após esta ação e, se for verificada alguma ou conjunto de memórias alteradas, o *Arduino* cria uma mensagem FINS com os respetivos dados e envia para um equipamento servidor.

As saídas digitais alteram o seu estado lógico mediante informação que tenha sido escrita anteriormente nas memórias do *Arduino*. É feita uma verificação do valor na memória *bit* a *bit*, e o valor da saída varia mediante o valor do *bit* nesse endereço de memória.

6.3.2 Servidor *Web Arduino*

No decorrer da programação realizada sobre o *Arduino*, além de um servidor FINS foi implementado igualmente um servidor *Web* (ver Figura 58). As páginas *Web* foram construídas com base em programação *Arduino*, HTML e HTTP, e os dados armazenados em memória EEPROM do *Arduino*.

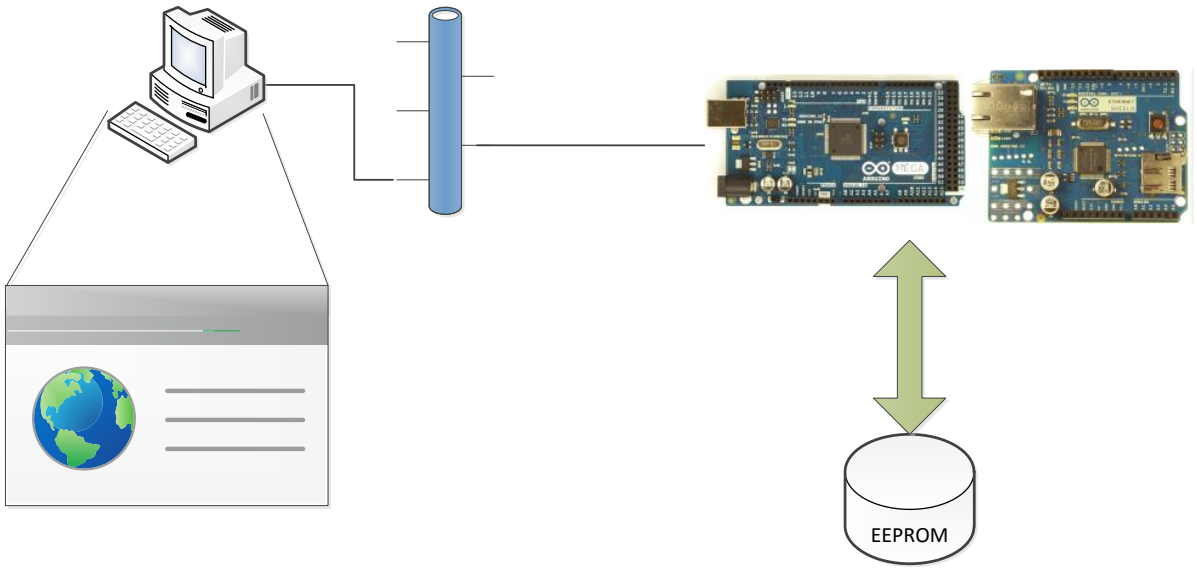
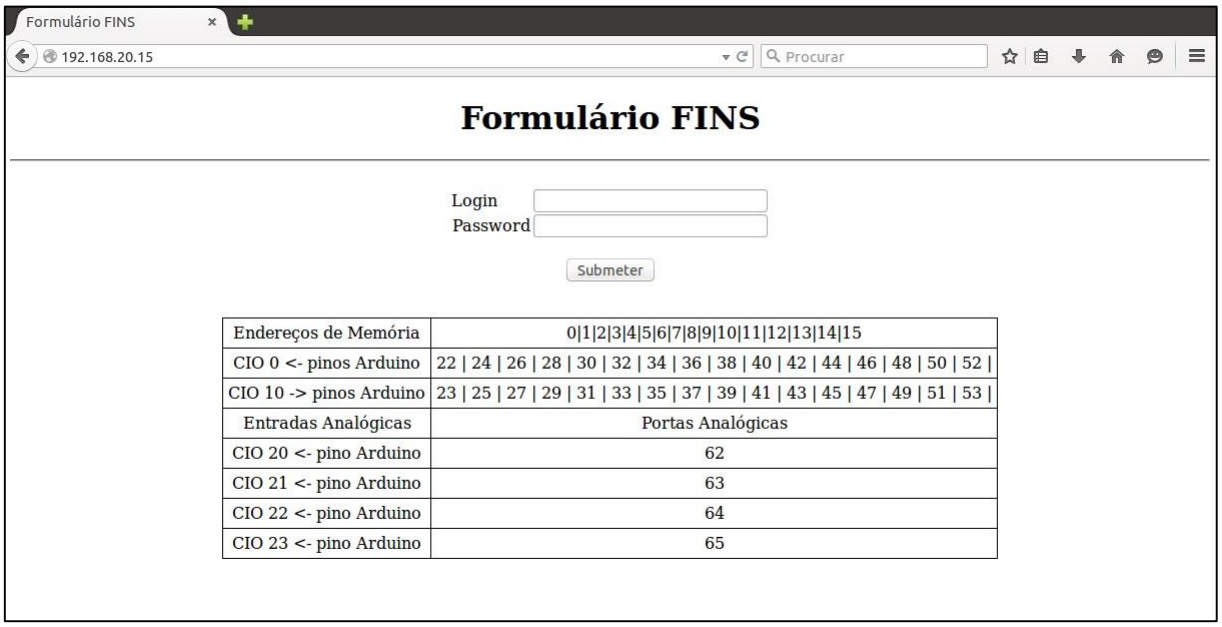


Figura 58 – Diagrama de comunicação com servidor *Web Arduino*.

O servidor *Web* disponibiliza ao utilizador duas páginas, uma página formulário e uma página de configuração. A primeira página permite ao *Arduino* verificar se o utilizador tem permissão para aceder à página de configuração. Esta verificação é feita através do campo *login* que identifica o utilizador, e de *password* que verifica a senha do utilizador. Caso os campos estejam de alguma forma incorretos, o utilizador mantém-se na página formulário.

Nesta mesma página é disponibilizada uma tabela que informa o utilizador do número de entradas e saídas que se encontram configuradas e os respetivos endereços de memória no *Arduino*. É possível mapear 32 pinos digitais (entradas e saídas) e 8 entradas analógicas em

endereços de memória. A figura 59 permite observar o formato da página Formulário FINS e a figura 60 o fluxograma da página formulário.



Endereços de Memória	0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|15
CIO 0 <- pinos Arduino	22 | 24 | 26 | 28 | 30 | 32 | 34 | 36 | 38 | 40 | 42 | 44 | 46 | 48 | 50 | 52 |
CIO 10 -> pinos Arduino	23 | 25 | 27 | 29 | 31 | 33 | 35 | 37 | 39 | 41 | 43 | 45 | 47 | 49 | 51 | 53 |
Entradas Analógicas	Portas Analógicas
CIO 20 <- pino Arduino	62
CIO 21 <- pino Arduino	63
CIO 22 <- pino Arduino	64
CIO 23 <- pino Arduino	65

Figura 59 – Página Formulário FINS.

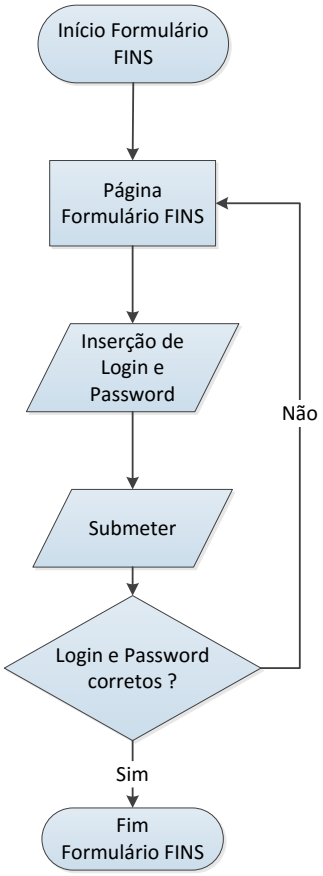


Figura 60 – Fluxograma da página *Web Formulário FINS*.

A figura 61 demonstra o formato da página de configuração do *Arduino*.

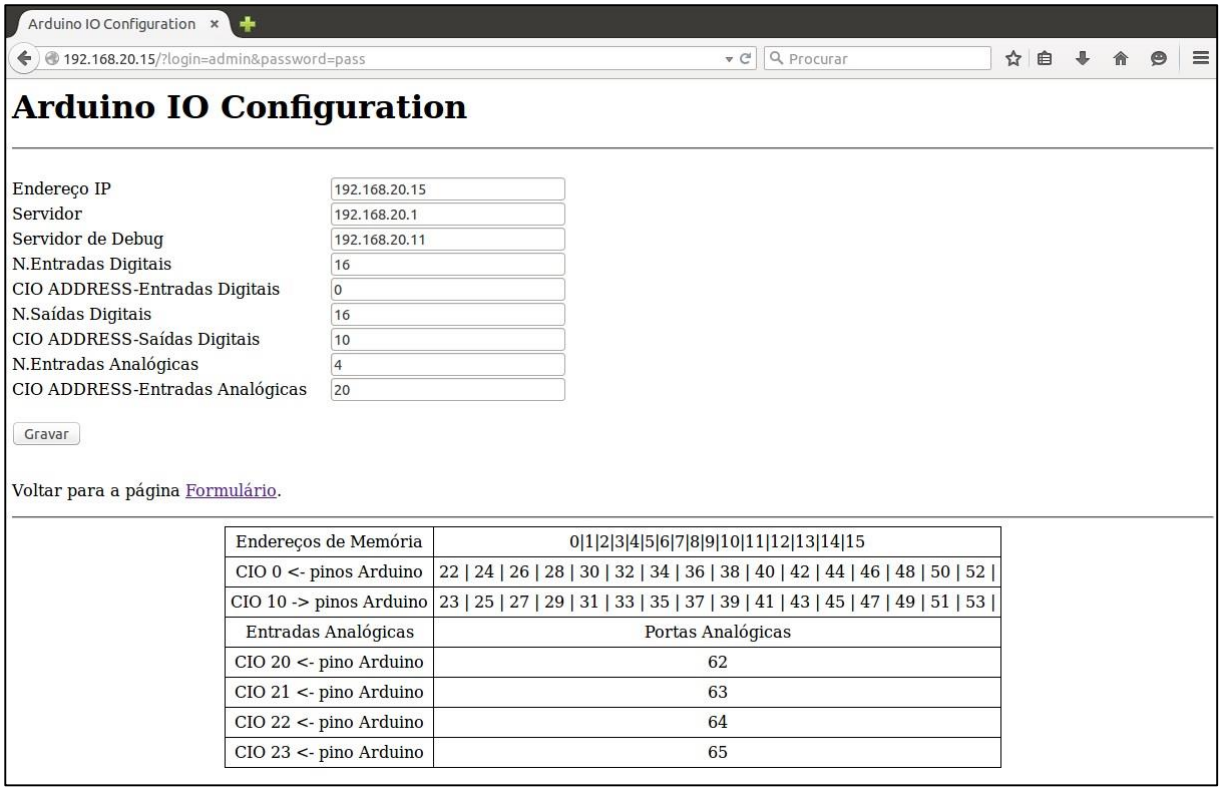


Figura 61 – Página Web de Configuração do Arduino.

Na página de configuração do *Arduino*, o utilizador tem oportunidade de parametrizar um conjunto de dados do *hardware Arduino*. Neste conjunto de dados é possível definir os endereços de comunicação do *Arduino*, ou seja, endereço IP próprio e os endereços para os quais envia as mensagens FINS. O operador pode também proceder à mudança do número de pinos digitais e entradas analógicas a utilizar e alterar o endereço de memória para o qual as entradas e saídas estão mapeadas. Esta informação fica disponível em formato de tabela, e os parâmetros carregados na memória EEPROM ficam disponíveis também nos campos de inserção dos dados para o utilizador ter um exemplo de preenchimento dos dados.

Os endereços FINS serão atribuídos com base no método de geração automática no servidor *Web*, ou seja, o endereço de nó FINS assumirá o valor do equipamento no endereço IP (último *byte* do endereço) e o endereço de rede FINS é atribuído pelo valor da rede no endereço IP (terceiro *byte* do endereço).

Na figura 62 é possível visualizar o fluxograma da página *Web* de configuração do *Arduino*.

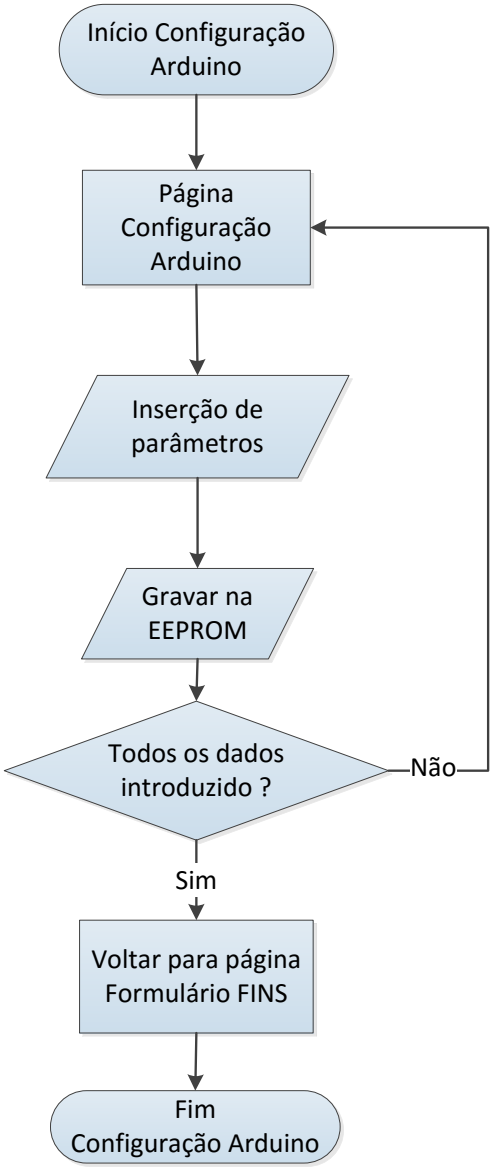


Figura 62 – Fluxograma da página Web Configuração do *Arduino*.

6.4 Sumário

Neste capítulo foi abordada a plataforma *Arduino* e o *firmware* que foi implementado no mesmo para este funcionar como um servidor FINS e servidor *Web*. Foi apresentado o modelo do *Arduino* utilizado assim como as estruturas de programação desenvolvidas para o funcionamento do *Arduino* em formato servidor.

7 Validação Experimental

7.1 Enquadramento

Este capítulo aborda os testes realizados com os equipamentos descritos anteriormente, juntamente com a biblioteca FINS que foi desenvolvida neste trabalho. Os equipamentos são ligados a um *router* de 4 portas, permitindo assim a interligação entre os vários dispositivos na mesma rede. A figura 63 apresenta um esquema de interligação entre os vários equipamentos. Desta forma, pretende-se demonstrar a exequibilidade do protocolo FINS na comunicação realizada entre os vários equipamentos.

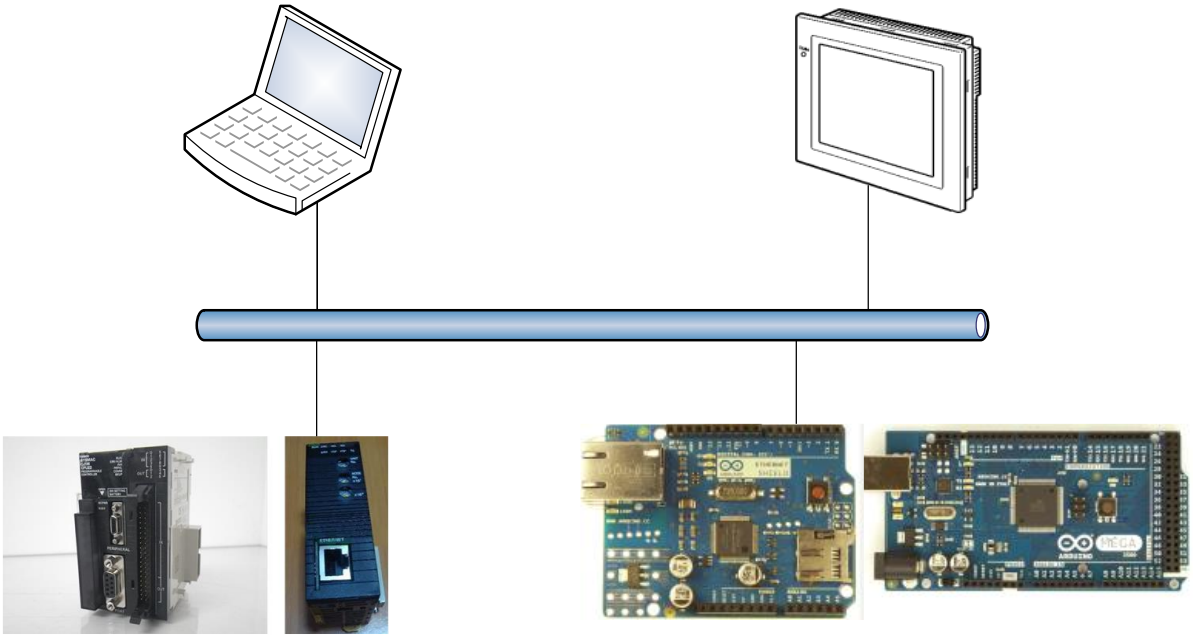


Figura 63 – Esquema de interligação entre os vários equipamentos utilizados.

7.2 Testes Realizados

7.2.1 Implementação de Contadores

A implementação dos contadores consiste na comunicação realizada entre a consola e os restantes equipamentos ligados na rede *Ethernet*, onde foram simulados contadores com as opções de incremento, decremento e *reset* nos respetivos equipamentos, sendo estas operações sinalizadas por iluminação de uma saída sempre que um dos respetivos botões seja premido.

Estes testes foram importantes neste trabalho, visto que ajudaram a compreender os comandos FINS que a consola envia aos equipamentos para leitura e escrita de dados, nomeadamente o *Arduino* e a biblioteca FINS implementada num computador portátil. A troca de comandos FINS foi monitorizada através de *Wireshark*, que é um *software* que permite análise do tráfego de dados na rede entre diversos equipamentos.

A figura 64 a), b) e c) dá uma perspetiva dos ecrãs desenvolvidos para a comunicação com os respetivos equipamentos.

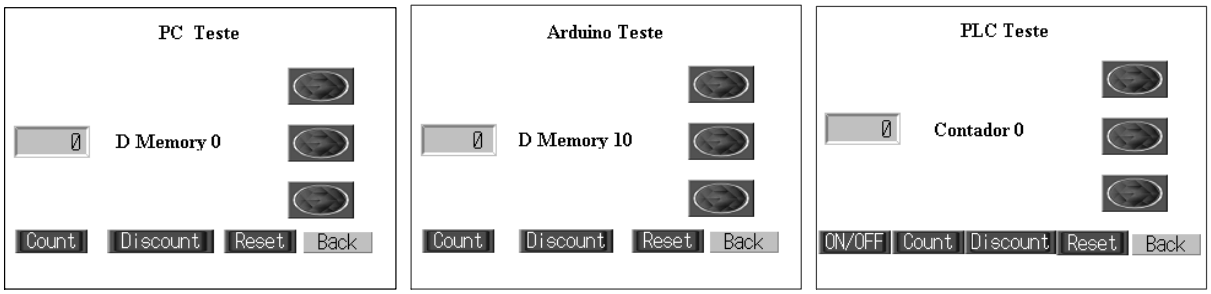


Figura 64 – a) Ecrã para comunicação com biblioteca FINS; b) Ecrã para comunicação com o *Arduino*; c) Ecrã para comunicação com o PLC OMRON.

Todos os ecrãs implementam as mesmas opções, com exceção do ecrã PLC Teste que apresenta um botão ON/OFF que viabiliza ou não viabiliza a realização das ações mencionadas nos respetivos contadores. Para retornar ao painel principal da consola interativa, foi criado o botão *back* que permite voltar ao ecrã principal.

7.2.2 Implementação de processo com motores DC

Na realização deste teste foi utilizada a biblioteca FINS como ponte de ligação entre a consola interativa e o *Arduino*, tendo como objetivo o comando de direção de dois motores DC. A biblioteca FINS é previamente configurada com quatro etapas para envio de comandos FINS para escrita nos pinos de saída do *Arduino* (ver Figura 65). A primeira e terceira etapa são responsáveis pela alteração do estado lógico dos motores DC de modo a definir a direção de rotação dos mesmos e sinaliza igualmente as etapas através de LED (*light-emitting diode*).

A segunda e quarta etapa são responsáveis pela paragem dos motores DC no *Arduino*.

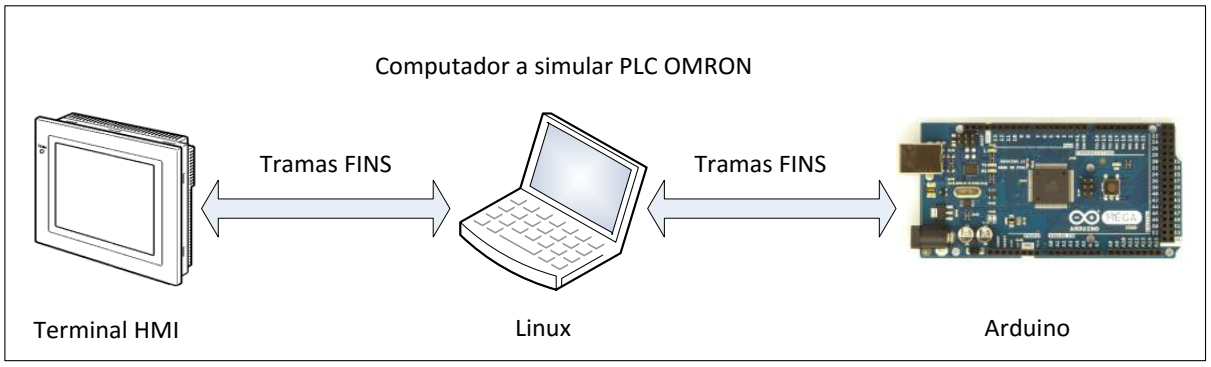
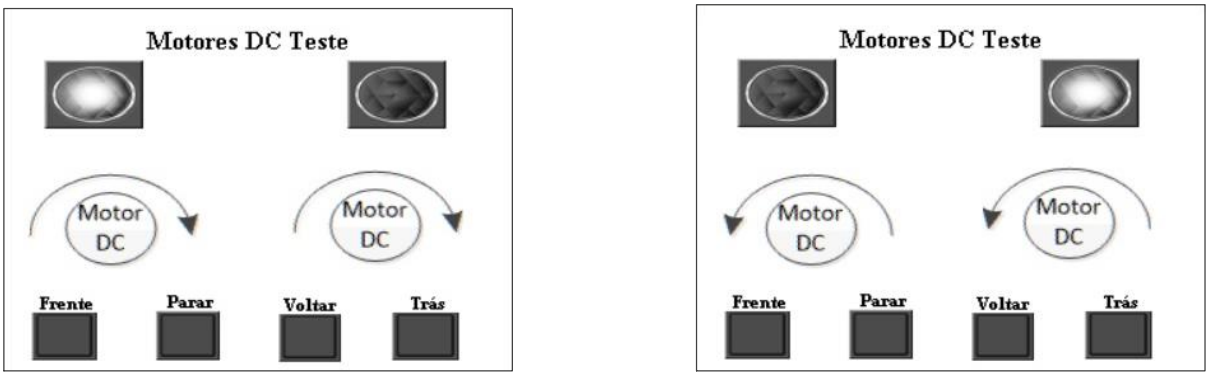


Figura 65 – Ligação entre Consola e *Arduino* através de biblioteca FINS.

Na figura 66 a) e b) são apresentados os ecrãs de funcionamento na consola para o controlo do processo descrito. Sempre que o utilizador premir um dos botões, a consola envia um comando para a biblioteca FINS, de modo a realizar a respetiva etapa. Caso a etapa seja realizada, a biblioteca FINS envia um comando de escrita para o *Arduino*, de forma a estipular o estado das saídas do *Arduino*. O estado das saídas do *Arduino* é monitorizado pela consola, através das lâmpadas que sinalizam a ordem de marcha dos motores DC e, pelas imagens, os motores DC que aparecem no ecrã, caso os motores estejam ligados.



a) b)

Figura 66 – a) Marcha em frente dos motores; b) Marcha para trás dos motores.

A figura 67 apresenta a montagem utilizada para o teste com o *Arduino* e os respetivos motores DC, sendo utilizado um circuito integrado L293D para o controlo destes motores.

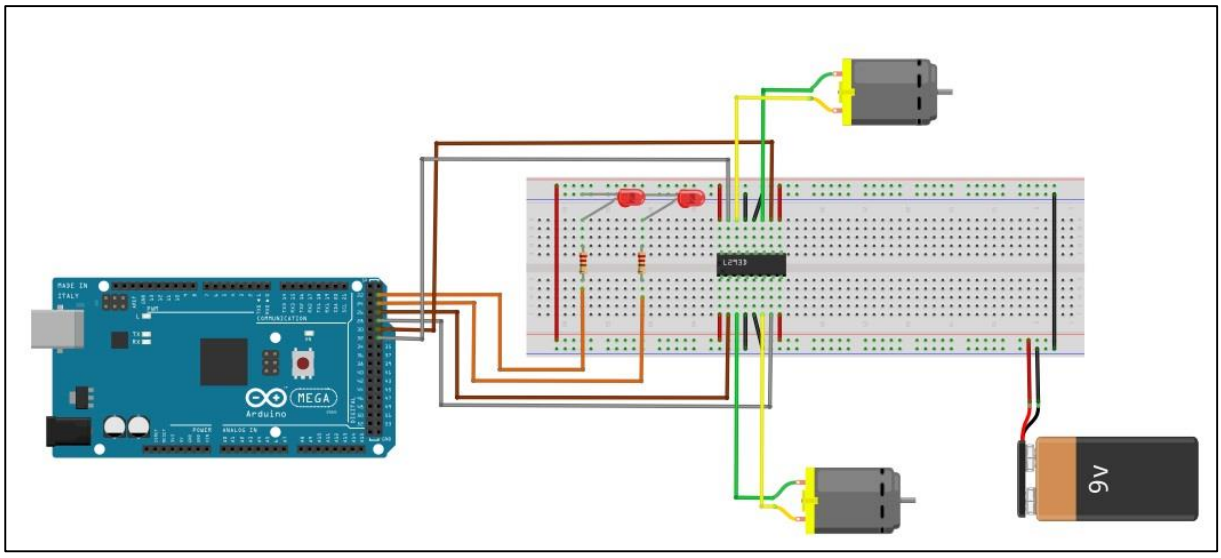


Figura 67 – Montagem para teste de motores DC com *Arduino*.

A figura 68 apresenta o diagrama de funcionamento do teste de motores DC.

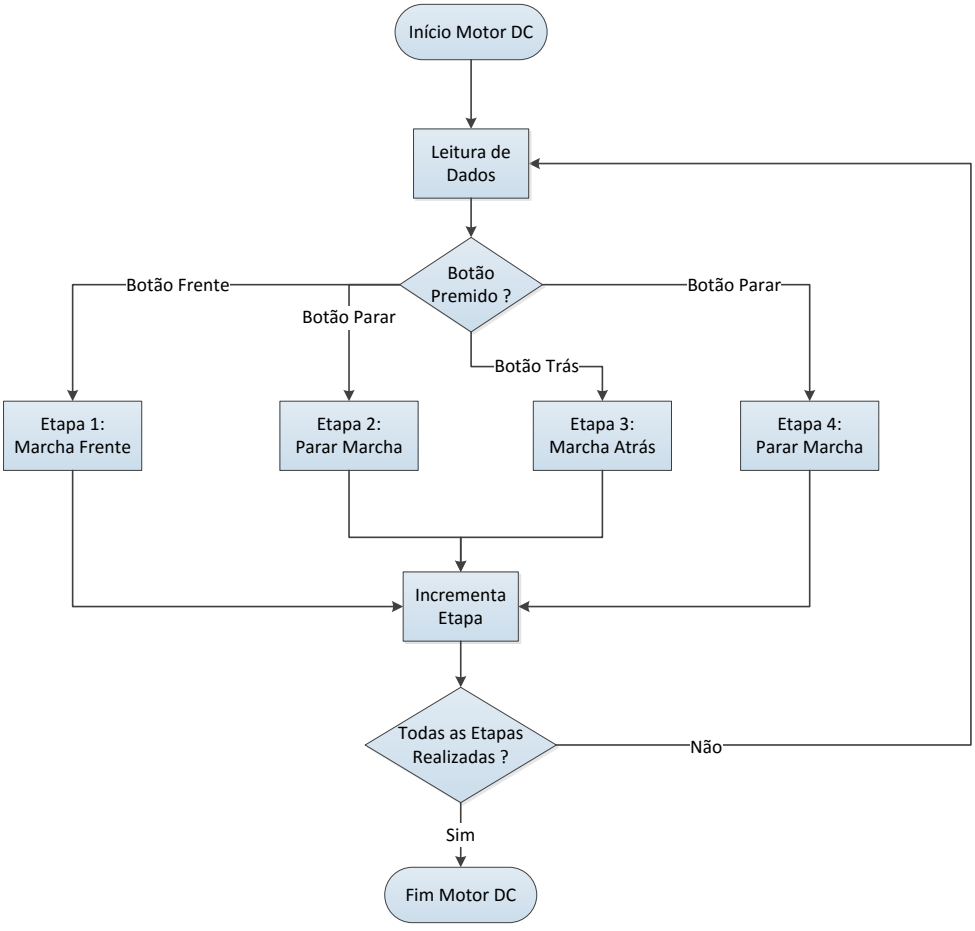


Figura 68 – Diagrama de funcionamento para teste de motores DC.

7.2.3 Implementação de processo com motor de passo

Neste teste foi desenvolvida uma atividade semelhante à descrita no subcapítulo 7.2.2, sendo, neste exemplo, feito o controlo de um motor de passo através do *Arduino*.

Neste caso, a consola controla o motor de passo através de quatro etapas que são ativadas por esta através da biblioteca FINS que de seguida envia os respetivos comandos para o *Arduino*.

O motor executa marcha em frente, marcha atrás ou pára, mediante a etapa em que este se encontre. Estas etapas são sinalizadas através de leds em circuito e em lâmpadas na consola.

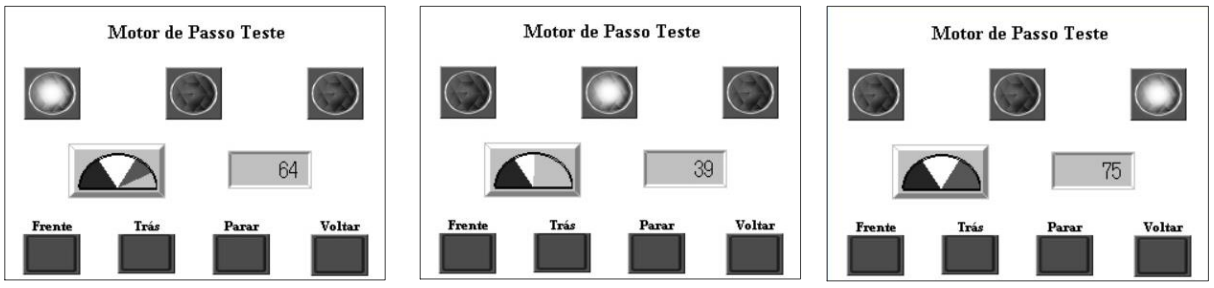
A etapa um e dois são responsáveis pela marcha em frente e atrás respetivamente, a etapa três é responsável pela paragem do motor de passo e a etapa quatro é utilizada para a limpeza do estado dos leds.

A consola permite igualmente monitorizar a velocidade de rotação do motor de passo, podendo esta ir variar entre as 0 e as 75 rotações por minuto.

A figura 69 a), b) e c) representa o formato do painel para o controlo do motor de passo.

Na figura 69 a) é indicada a marcha em frente do motor de passo através do estado ON da primeira lâmpada, e na figura 69 b) é sinalizada a marcha atrás através do acendimento da segunda lâmpada.

A paragem do motor é assinalada na figura 69 c) através da terceira lâmpada.



a) b) c)

Figura 69 – Ecrã de funcionamento do motor de passo.

Na figura 70 é possível observar o diagrama de funcionamento do motor de passo usado como exemplo.

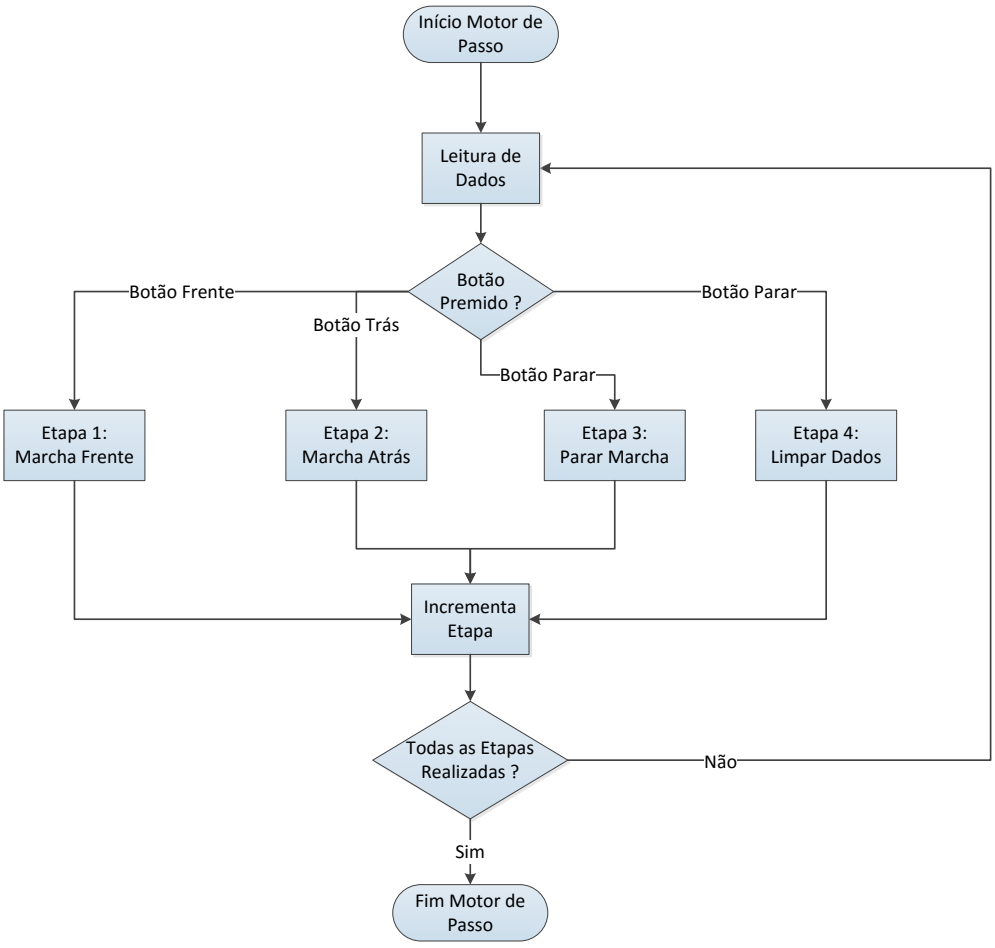


Figura 70 - Diagrama de funcionamento para teste de motor de passo.

O circuito utilizado na realização deste teste é apresentado na figura 71, tendo sido utilizado para o controlo do motor um circuito integrado ULN2003A.

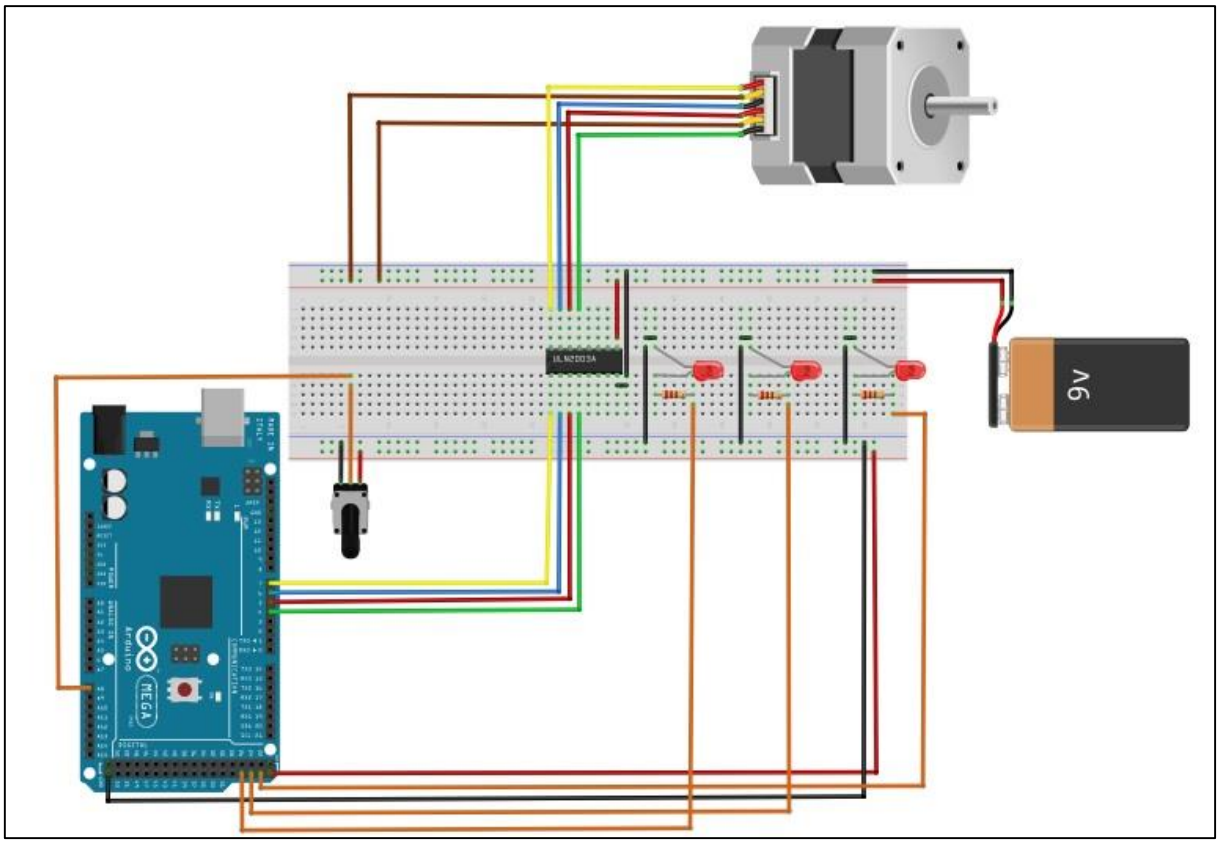


Figura 71 – Montagem para teste de motor de passo com *Arduino*.

7.3 Sumário

Neste capítulo foi demonstrada a interligação dos equipamentos utilizados neste trabalho, sendo apresentado alguns testes que foram realizados com os respetivos dispositivos e a possibilidade de algumas implementações com os mesmos.

8 Conclusões e Trabalhos Futuros

A presente monografia teve como objetivo a descrição do desenvolvimento de uma ferramenta de interligação de dispositivos através do protocolo industrial FINS. Nessa medida, o presente trabalho de projeto reúne um levantamento dos estudos realizados e trabalho desenvolvido para o efeito, como, por exemplo a descrição de uma arquitetura de rede industrial, ou estudo dos protocolos *Ethernet* Industrial mais utilizados no mercado, um estudo aprofundado sobre protocolo FINS da OMRON, entre outros temas.

A ferramenta desenvolvida vai de encontro aos objetivos propostos inicialmente, visto que a plataforma FINS criada permite a interação com equipamentos que implementem o protocolo FINS e estejam ligados à rede *Ethernet*, possibilita a execução desta plataforma como servidor para armazenar dados dos vários equipamentos e permite a simulação do comportamento de um PLC, ou seja, implementar comandos e instruções por etapas de funcionamento.

Esta ferramenta permite a comunicação com todos os equipamentos que utilizem a comunicação FINS, como se pôde comprovar com os dispositivos utilizados neste trabalho da OMRON (Consola HMI e PLC), e com a plataforma eletrónica utilizada (*Arduino*).

Desta forma, a plataforma FINS tem potencialidades para efetuar o controlo de processos, funciona como ponte de ligação entre os diversos equipamentos que se encontrem ligados em rede *Ethernet*, e acima de tudo implementa um protocolo de comunicações *Open Source* que pode ser adaptado a vários tipos de comunicação. Algumas das dificuldades encontradas neste projeto foi a necessidade de conseguir responder a todas as mensagens que a plataforma FINS recebia, e a manipulação dos valores nas estruturas de memória de modo a conseguir alterar o valor binário de uma memória *bi*t a *bit*. A primeira foi resolvida com o recurso ao mecanismo *Thread*, de modo a conseguir dar resposta aos vários pacotes de dados que a plataforma FINS recebe. A questão da manipulação do valor das memórias foi resolvido com recurso ao conceito *Bitwise Operators*, um conjunto de operadores que permitem a manipulação dos *bits* no valor armazenado em memória. Através deste mecanismo é possível fazer a divisão de uma *word* em dois valores de oito *bits* cada e a operação inversa inclusive, assim como, permite manipular o estado de cada *bit* na memória em formato *word*.

De uma forma geral, este trabalho foi uma mais-valia na conclusão deste percurso académico, devido aos conhecimentos adquiridos no que diz respeito à programação de diferentes tipos de

equipamentos, na interação com diferentes ferramentas de desenvolvimento de programação, e nos protocolos de comunicação utilizados em ambiente industrial.

Relativamente a trabalhos futuros, é expetável que exista um contínuo desenvolvimento desta ferramenta e aplicação da mesma em atividades voltadas para a produção e manutenção industrial.

Um dos próximos trabalhos a serem desenvolvidos é a implementação de outros protocolos industriais, como, por exemplo o FINS/TCP, o MODBUS/TCP, o Ethernet/IP, entre outros protocolos de comunicação industrial que possam eventualmente ser estudados. Desta forma aumenta-se o leque de equipamentos com os quais a plataforma desenvolvida possa comunicar.

Outro trabalho que pode vir a ser implementado é a comunicação entre diferentes dispositivos através do protocolo FINS que estejam localizados em redes *Ethernet* diferentes, e comunicação entre os equipamentos através de redes sem fios, visto que já existem tecnologias que permitem a realização deste tipo de teste, como é o caso da plataforma eletrónica *Arduino*.

Como trabalho futuro é igualmente interessante o desenvolvimento de uma ferramenta que permita a ligação dos dados da plataforma desenvolvida à *Internet* e desenvolver uma aplicação *Android* para a monitorização dos dados recebidos e atuação sobre a instalação mediante a informação recebida. Desta forma seria possível verificar uma atividade industrial e conceber a sua manutenção caso fosse necessário.

Referências Bibliográficas

[1] "Citisystems," [Online]. Available: http://www.citisystems.com.br/sete-beneficios-automacao-industrial/. [Acedido em 30 Junho 2015].

[2] B. Galloway e G. P.Hancke, "Introduction to Industrial Control Networks," *Communications Surveys & Tutorials, IEEE,* vol. 15, pp. 860 - 880, 2012.

[3] S.Djiev, " Industrial Networks for Communication and Control," [Online]. Available: http://anp.tu-sofia.bg/djiev/PDF%20files/Industrial%20Networks.pdf. [Acedido em 18 Junho 2015].

[4] J. Powell e H. Vandeline, Catching the Process Fieldbus: An Introduction to Profibus for Process Automation, New York: Momentum Press, 2013.

[5] D. N. Mahalik, Fieldbus Technology: Industrial Network Standards for Real-Time Distributed Control, Berlin: Springer, 2003.

[6] R. Zurawski, Industrial Communication Technology Handbook, Boca Raton: CRC Press, 2015.

[7] S. K. Sen, Fieldbus and Networking in Process Automation, Boca Raton: CRC Press, 2014.

[8] J. Park, S. Mackay e E. Wright, Practical Data Communications for Instrumentation and Control, Amsterdam: Newnes, 2003.

[9] T. Sauter, "The Three Generations of Field-Level Networks - Evolution and Compability Issues," *IEEE Transactions on Industrial Electronics,* vol. 57, pp. 3585 - 3595, 2010.

[10] B. Lipták, Instrument Engineers Handbook: Process Software and Digital Networks, Boca Raton: CRC Press, 2002.

[11] J.-P. Thomesse, "Fieldbus Technology in Industrial Automation," *Proceedings of the IEEE* , vol. 93, pp. 1073 - 1101, 2005.

[12] Turck , "AC Controls," 28 Agosto 2013. [Online]. Available: http://www.accontrols.com/assets/docs/eng-resources/Industrial%20Ethernet%20Field%20Guide.pdf. [Acedido em 14 Junho 2015].

[13] E. Monteiro e F. Boavida, Engenharia de Redes Informáticas, Lisboa: FCA - Editora de Informática, 2000.

[14] Z. Lin e S. Pearson, "Texas Instruments," Novembro 2013. [Online]. Available: http://www.ti.com/lit/wp/spry254/spry254.pdf. [Acedido em 16 Janeiro 2015].

[15] "Schneider Electric," Schneider Electric, [Online]. Available: http://www.schneider-electric.co.uk/sites/uk/en/products-services/automation-control/products-offer/human-machine-interface/human-machine-interface.page. [Acedido em 18 Junho 2015].

[16] F. Lamb, Industrial Automation Hands-On, Mc Graw Hill, 2013.

[17] "IDC Technlogies," [Online]. Available: http://www.idc-online.com/technical_references/pdfs/data_communications/tutorial_3.pdf. [Acedido em 10 Junho 2015].

[18] P. Zhang, Advanced Industrial Control Technology, Oxford: Elsevier, 2010.

[19] Schneider Electric, "Schneider Electric," 2008. [Online]. Available: http://www2.schneider-electric.com/documents/automation-control/pdf/Automation_Solution_guide_2008-EN_web.pdf. [Acedido em 29 Julho 2015].

[20] J. Love, Process Automation Handbook, London: Spinger, 2007.

[21] D. Reynders, S. Mackay e E. Wright, Practical Industrial Data Communications, Oxford: Newnes, 2005.

[22] J. C. Warren, "Industrial Networking," *IEEE Industry Applications Magazine*, vol. 17, pp. 20 - 24, 2011.

[23] P. S. Marshall e J. S. Rinaldi, Industrial Ethernet, ISA - The Instrumentation, Systems, and Automation Society, 2004.

[24] T. Leyrer, "Texas Industruments," Abril 2013. [Online]. Available: http://www.ti.com/lit/wp/sszy007/sszy007.pdf. [Acedido em 13 Julho 2015].

[25] "EtherCAT Technology Group," EtherCAT Technology Group, [Online]. Available: https://www.ethercat.org/default.htm. [Acedido em 13 Julho 2015].

[26] ODVA, "ODVA," 2008. [Online]. Available: http://www.odva.org/Portals/0/Library/Publications_Numbered/PUB00213R0_EtherNetIP_Developers_Guide.pdf. [Acedido em 14 Julho 2015].

[27] Allen-Bradley; Automation, Rockwell;, "Rockwell Automation," [Online]. Available: http://literature.rockwellautomation.com/idc/groups/literature/documents/rm/enet-rm002_-en-p.pdf. [Acedido em 13 Julho 2015].

[28] "Open Safety," Open Safety, 2011. [Online]. Available: http://www.open-safety.org/. [Acedido em 29 Julho 2015].

[29] "Modbus Organization," Modbus Organization, [Online]. Available: http://www.modbus.org/specs.php. [Acedido em 27 Julho 2015].

[30] Kepware Technologies, "Kepware Technologies," 2014. [Online]. Available: http://www.kepware.com/Support_Center/SupportDocuments/Help/omron_fins_ethernet.pdf. [Acedido em 27 Novembro 2014].

[31] OMRON, "OMRON," Julho 2009. [Online]. Available: http://www.fa.omron.com.cn/data_pdf/mnu/w343-e1-07_cs1w_cj1w-etn.pdf?id=31. [Acedido em 16 Novembro 2014].

[32] OMRON, "OMRON," Abril 2009. [Online]. Available: http://www.fa.omron.com.cn/data_pdf/mnu/w421-e1-04_cs1w_cj1w-etn21.pdf?id=1338. [Acedido em 16 Janeiro 2015].

[33] OMRON, "OMRON," 2003. [Online]. Available: http://www.fa.omron.com.cn/data_pdf/mnu/v083-e1-26_ns5_8_10_12_15.pdf?id=155. [Acedido em 22 Janeiro 2015].

[34] Prof2000, "Prof2000," [Online]. Available: http://www.prof2000.pt/users/lpa/. [Acedido em 6 Outubro 2014].

[35] OMRON, "OMRON," OMRON, [Online]. Available: https://www.ia.omron.com/products/family/1569/. [Acedido em 27 Julho 2015].

[36] C. Silva, "Clube de Eletrónica," 11 Junho 2011. [Online]. Available: http://www.clubedaeletronica.com.br/Automacao/PDF/Capitulo%20003%20-%20Normalizacao%20IEC61131.pdf. [Acedido em 22 Julho 2015].

[37] "Beckhoff New Automation Technology," Beckhoff New Automation Technology, [Online]. Available: http://infosys.beckhoff.com/english.php?content=../content/1033/tcplccontrol/html/tcplcctrl_languages%20il.htm&id=. [Acedido em 22 Julho 2015].

[38] "GCC, the GNU Compiler Collection," GCC, the GNU Compiler Collection, [Online]. Available: https://gcc.gnu.org/. [Acedido em 07 Agosto 2015].

[39] "GNU Operating System," GNU Operating System, [Online]. Available: https://www.gnu.org/. [Acedido em 07 Agosto 2015].

[40] M. Mitchell, J. Oldham e A. Samuel, Advanced Linux Programming, New Riders, 2001.

[41] M. Pereira Ponti Jr., "Universidade de São Paulo," [Online]. Available: http://wiki.icmc.usp.br/images/0/0a/ApostilaMakefiles2011.pdf. [Acedido em 15 Outubro 2014].

[42] N. Malheiro, M. J. Viamonte, B. Batista e L. L. Ferreira, "Sistemas Operativos I- ISEP," Abril 2006. [Online]. Available: http://www.dei.isep.ipp.pt/~bbatista/Sop1/ParteIII.pdf. [Acedido em 7 Agosto 2015].

[43] M. J. Donahoo e K. L. Calvert, TCP/IP Sockets in C : Practical Guide for Programmers, San Francisco: Morgan Kaufmann Publishers, 2001.

[44] "Tutorialspoint Simply Easy Learning," Tutorialspoint Simply Easy Learning, [Online]. Available: http://www.tutorialspoint.com. [Acedido em 11 Agosto 2015].

[45] "Github," Github, 19 Setembro 2010. [Online]. Available: https://github.com/practicingruby/guides/wiki/Socket. [Acedido em 11 Agosto 2015].

[46] P. Krzyzanowski, "Distributed Systems - Programming with UDP sockets," Rutgers School of Arts and Sciences, [Online]. Available: https://www.cs.rutgers.edu/~pxk/417/notes/sockets/udp.html. [Acedido em 13 Outubro 2014].

[47] "Ubuntu Manuals," Ubuntu Manuals, [Online]. Available: http://manpages.ubuntu.com/manpages/quantal/pt/man7/ip.7.html. [Acedido em 27 Novembro 2014].

[48] A. Moreira, "Documentação de apoio às aulas - ISEP," [Online]. Available: http://www.dei.isep.ipp.pt/~andre/documentos. [Acedido em 11 Agosto 2015].

[49] OMRON, "Manualslib," Julho 2009. [Online]. Available: http://www.manualslib.com/manual/346443/Omron-Cj-Reference-Manual-07-2009.html. [Acedido em 27 Novembro 2014].

[50] M. Kerrisk, The Linux Programming Interface: A Linux and UNIX System Programming Handbook, San Francisco: No Starch Press, 2010.

[51] "Arduino," Arduino, [Online]. Available: https://www.arduino.cc/. [Acedido em 14 Maio 2015].

[52] Atmel, "Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V," Fevereiro 2014. [Online]. Available: http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf. [Acedido em 20 Agosto 2015].

Anexos

Anexo A Conceitos de Rede e Protocolos

Anexo A.1 Redes *Ethernet*

A tecnologia *Ethernet* foi desenvolvida na década de 70 pela *Xerox*, *Intel* e *DEC*, no seguimento da invenção, por Bob Metcalfe e David Boggs, da *Xerox*, em 1973 – tendo sido posteriormente normalizadas pelo IEEE (norma IEEE 802.3) e pela ISO (ISO 8802-3). É considerada uma tecnologia de grande aceitação e divulgação, alcançando a esmagadora maioria do parque generalizado de redes locais.

A enorme divulgação desta tecnologia originou um baixo custo e uma grande maturidade, tendo sido estes os fatores responsáveis pela manutenção do domínio do mercado.

A tecnologia *Ethernet* usa a técnica CSMA/CD (*Carrier Multiple Access with Collision Detection*) para controlo do acesso ao meio. Inicialmente foi desenvolvida para redes com topologia em *bus* físico recorrendo à utilização de cabo coaxial mas, com o avançar do tempo foi evoluindo. Atualmente suporta uma grande variedade de meios físicos. Da mesma forma, a topologia utilizada evoluiu, passando de um *bus* físico para um *bus* lógico. A este *bus*, normalmente é atribuído uma topologia em estrela ou árvore. Em paralelo com estas mudanças, o mecanismo CSMA/CD tem vindo a sofrer alterações, de maneira a proporcionar uma rede eficiente a débitos elevados (com valores de débito de 100Mbps e 1Gbps).

O suporte de diferentes meios físicos e diferentes velocidades levou ao aparecimento de diversas variantes de *Ethernet*, geralmente designadas por *x-Base-y*, em que *x* é o número que identifica o débito binário em Mbps, *Base* significa que a transmissão é efetuada em banda base, e *y* é o número ou letras que identifica o género de meio físico utilizado.

Endereçamento e formato de pacotes:

O endereçamento numa rede *Ethernet* é feito com base numa sequência de 6 *bytes* (48 *bits*) que identifica o equipamento ou nó na rede. Esta sequência é, geralmente conhecida por endereço físico ou endereço MAC, sendo representado por numeração hexadecimal, separado por dois pontos. Ex: 12 : DE : FF : 3C : DB : 01.

A comunicação entre equipamentos numa rede *Ethernet* tem por norma a utilização de pacotes de dados, sendo estes designados por tramas, *frames* ou quadros. Na mesma rede *Ethernet*

podem coexistir vários formatos de tramas sem grandes problemas, sendo o "*Ethernet II*" o formato mais divulgado.

Na imagem seguinte é apresentado o formato de uma trama "*Ethernet II*":

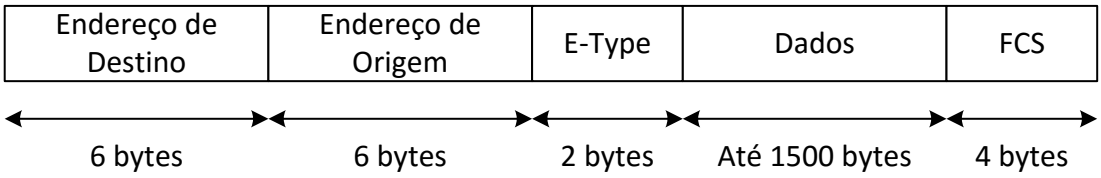


Figura 72 - Formato de trama *Ethernet II*; adaptado de [48].

O *E-Type* (*Ethernet-Type*) é o identificador para multiplexagem, sendo atribuído o identificador de cada protocolo. Ex: protocolo IP = 800.

O FCS (*Frame Check Sequence*) é o campo utilizado na deteção de erros na trama [13], [44].

Anexo A.2 Protocolo IP

O protocolo IP (*Internet Protocol*) foi implementado em 1970 no desenvolvimento da ARPANET, tendo sido esta rede interligada a outras. Em 1980 o vasto conjunto destas redes ficaria conhecido como *Internet* [48].

Este protocolo é a arquitetura responsável pela circulação dos pacotes de dados, sendo designados também por datagramas, executando o seu encaminhamento com base nos endereços de destino. Este protocolo executa ações de fragmentação e de reconstituição de pacotes, permitindo nestas operações o ajuste do tamanho dos pacotes de dados ao tamanho máximo dos quadros suportados pela tecnologia da rede subjacente. Este protocolo funciona em modo ausência de ligação, ou seja, não garante a transferência fiável de informação, não executa mecanismos de deteção ou recuperação de erros, e estas operações são entregues a protocolos de níveis superiores para a sua realização.

Na imagem seguinte é possível visualizar o formato do datagrama IP.

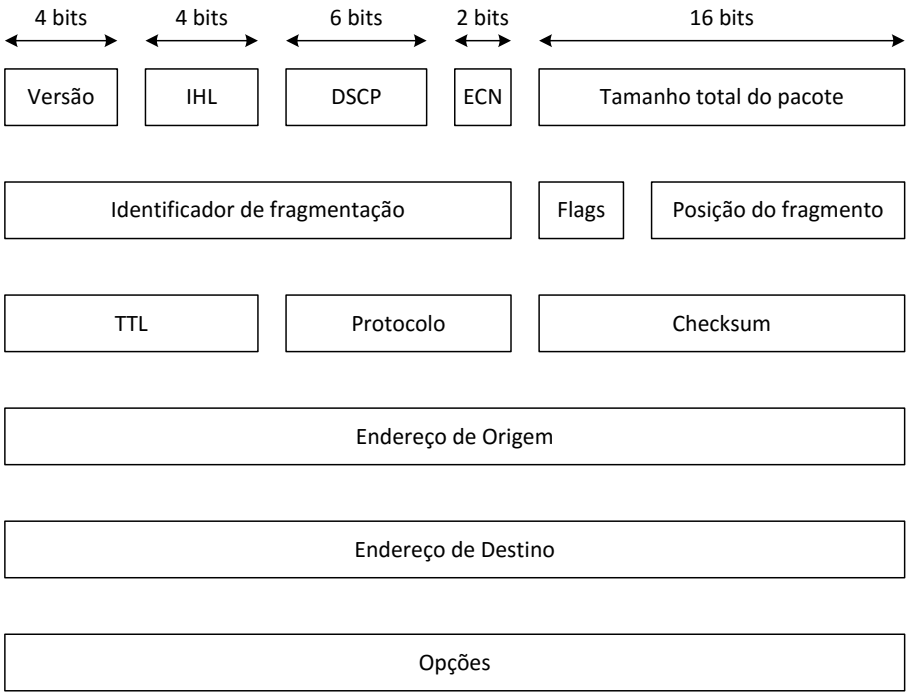


Figura 73 – Datagrama IP; adaptado de [48].

A constituição do datagrama IP é a seguinte:

- Versão: É o número da versão do protocolo IP. Ex: IPv4;
- IHL (*Internet Header Length*): Identifica o comprimento do datagrama IP;
- DSCP (*Differentiated Services Code Point*): Identifica o tipo de serviço;
- ECN (*Explicit Congestion Notification*): É o campo responsável pela informação do congestionamento visto na rota;
- Tamanho total do pacote: Identifica o tamanho inteiro do pacote IP (Datagrama IP +Dados);
- Identificação de fragmentação: É o campo responsável pela identificação dos fragmentos de pacote IP, isto é, caso um pacote IP seja fragmentado durante a sua transmissão, todos os fragmentos tem um número de identificação que permite identificar o pacote original aos quais pertencem;
- *Flags*: É o campo requerido pelos recursos de rede, isto é, caso o pacote IP seja bastante grande este campo assinala se o pacote pode ser fragmentado ou não;
- Posição do fragmento: Identifica a posição exata do fragmento no pacote IP original;
- TTL (*Time to Live*): Este campo estabelece um valor ao pacote IP a sinalizar quantos pontos de ligação ele pode atravessar (Ex:*Router*). Em cada ponto que o pacote

atravesse o valor TTL é decrementado e, no momento em que este campo chegar a zero o pacote é descartado;

- Protocolo: Este campo é responsável por informar a camada de rede no operador destino, qual é o protocolo a que o pacote pertence, ou seja, o próximo nível de protocolo. Ex: UDP=17, TCP = 6;
- *Checksum*: Este campo é utilizado para verificar se o pacote é recebido sem erros;
- Endereço de Origem: Este endereço identifica o endereço de origem do remetente do pacote IP;
- Endereço de Destino: Este endereço identifica o endereço de destino do destinatário do pacote IP;
- Opções: Este campo é opcional, sendo utilizado caso o valor de IHL seja superior a 5. As opções utilizadas são de transporte de valores de segurança, registo de rota, entre outros.

Endereço IP e Máscara de Rede:

O mecanismo de endereçamento do protocolo IP identifica o utilizador ligado na rede através de um endereço do nível de rede – um endereço IP. O endereço IP é constituído por 32 *bits* na versão 4 do protocolo IP, onde os *bits* mais significativos identificam a rede ao qual o equipamento pertence e os *bits* menos significativos identificam o equipamento dentro da rede. Estes endereços são representados por notação decimal, sendo separados por um ponto. Ex: 192.168.20.40.

A máscara de rede é uma sequência de 32 *bits* que possibilita a identificação da rede e do utilizador no endereço IP. A utilização da máscara de rede permite assim uma utilização mais eficiente do espaço de endereçamento, simplificando o encaminhamento e desperdiçando menos os endereços. Ex: 255.255.255.0. [13], [44], [48].

Anexo A.3 Protocolo UDP

O protocolo UDP é um protocolo de troca de mensagens, mais conhecido por modo de comunicação datagrama, utilizado para a comunicação entre equipamentos através de comutação de pacotes de dados dentro de uma rede. Este protocolo fornece um processo para o envio de mensagens entre programas de aplicação com um mecanismo mínimo de protocolo.

O protocolo UDP adiciona ao protocolo IP, o número de porta e o *checksum*, sendo um protocolo de transação orientada. Esta particularidade acaba por ser uma desvantagem devido à ausência de garantia na entrega dos datagramas ou na ordem de entrega dos datagramas.

Em termos de vantagens, trata-se de um bom protocolo para fluxo de dados numa única direção, e é adequado a comunicações baseadas em perguntas "*query*".

A constituição do cabeçalho UDP é formada por 4 campos de 16 bits cada, como se pode observar na imagem seguinte:

Porta de Origem	Porta de Destino
Comprimento	Checksum

Figura 74 – Cabeçalho UDP.

- Porta de Origem: Este campo é utilizado para identificação do número de porta de origem do pacote de dados;
- Porta de Destino: Este campo é utilizado para identificação do número de porta da aplicação da máquina destino;
- Comprimento: Este campo específica o comprimento do pacote de dados UDP, incluindo o cabeçalho;
- *Checksum*: Este parâmetro armazena o valor *checksum* criado pelo remetente antes de enviar o pacote de dados. Caso o valor seja zero ou venha em branco, o pacote de dados é eliminado [44], [48].

Anexo A.4 Protocolo TCP

O protocolo TCP é um protocolo mais amigável para o transporte de pacotes de dados na rede. Este protocolo tem como objetivo fornecer uma comunicação fiável entre aplicações, baseado no conceito de ligação e garantindo desta forma a transmissão dos dados livres de erros de qualquer fluxo de *bytes* entre o emissor e o recetor.

Este protocolo apresenta assim as seguintes caraterísticas:

- O protocolo TCP envia e recebe os dados em fluxo "*stream*", não existindo o conceito de mensagem;
- Eventuais erros de transmissão são corrigidos de forma transparente para as aplicações;
- Os dados chegam ao destino no mesmo formato em que foram emitidos;
- Define os números de porta para multiplexar o acesso ao TCP por vários programas residentes na mesma máquina.

Na imagem seguinte é apresentado o cabeçalho TCP.

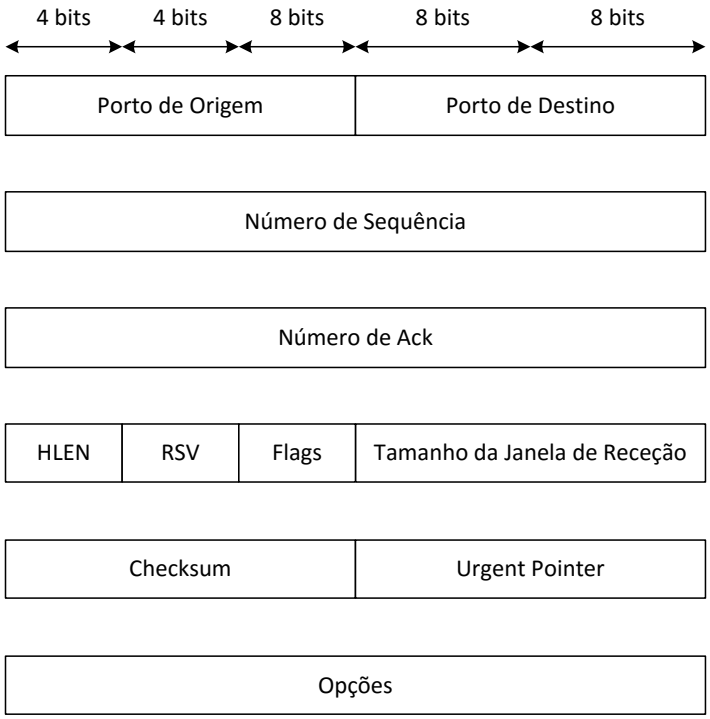


Figura 75 - Cabeçalho do protocolo TCP.

O cabeçalho TCP apresenta os seguintes campos:

- Porta de Origem: Identifica a porta de origem da aplicação no equipamento emissor;
- Porta de Destino: Identifica a porta de destino da aplicação no equipamento recetor;
- Número de Sequência: Este campo é o número de sequência de dados de um segmento na sessão;
- Número de ACK (*Acknowledgement*): Este campo é responsável por identificar o número da próxima sequência de dados a receber e funciona como confirmação dos dados recebidos;

- HLEN (*Header Length*): Este campo é responsável pela identificação do comprimento do encapsulamento TCP (Cabeçalho TCP + Dados);
- RSV (Reserved): Este campo é reservado para utilização futura e deve conter zeros;
- *Flags*: Este campo é constituído por 6 *flags* que possibilitam o controlo de diversos aspetos de implementação;
- Tamanho da Janela de Receção: Este parâmetro é utilizado para controlo de fluxo entre dois equipamentos e indica a quantidade de dados que o recetor aloca para o segmento, ou seja, o tamanho que espera receber;
- *Checksum*: Este parâmetro permite a deteção de erros e é aplicado ao cabeçalho TCP e ao cabeçalho IP;
- *Urgent Pointer*: Identifica os dados urgentes, caso esteja definido a 1;
- Opções: Este campo permite a adição de opções adicionais ao cabeçalho TCP, que não são abrangidas pelo cabeçalho base deste protocolo [13], [44], [48].

Anexo A.5 Protocolo HTTP

O HTTP (*Hypertext Transfer Protocol*) é um protocolo ao nível de aplicação, sendo utilizado para distribuição de sistemas de informação. Este protocolo é o alicerce para a comunicação de dados via WWW (*World Wide Web*), conhecida também como *internet*. O protocolo HTTP baseia-se no protocolo de comunicação TCP para o transporte de diversos dados, como, por exemplo: documentos HTML, ficheiros de imagens, entre outros; a porta TCP padrão deste protocolo é a número 80, mas pode ser utilizada outra para o mesmo efeito.

O HTTP utiliza um modelo de pedido/resposta baseado no modelo cliente/servidor, onde o cliente, normalmente é um *web browser*, e o servidor um *Web server* [44].

Anexo B Tabela de Comandos FINS

A tabela 4 contém todos os comandos que a unidade CPU do PLC OMRON interpreta através do *firmware* programado pelo fabricante.

Tabela 4 – Tabela de comandos FINS; adaptado de [31].

Tipo	Código de Comando		Nome	Função
	MR	SR		
I/O memory area access	01	01	*Memory Area Read*	Lê os conteúdos nas *words* de memória de forma consecutiva.
	01	02	*Memory Area Write*	Escreve os conteúdos nas *words* de memória de forma consecutiva.
	01	03	*Memory Area Fill*	Escreve os mesmos dados numa gama de *words* numa memória.
	01	04	*Multiple Memory Area Read*	Lê os conteúdos de *words* de memória escolhidas.
	01	05	*Memory Area Transfer*	Copia os conteúdos de *words* de uma memória de forma consecutiva para outras *words* de memória.
Parameter area access	02	01	*Parameter Area Read*	Lê os conteúdos das *words* de parâmetros de forma consecutiva.
	02	02	*Parameter Area Write*	Escreve os conteúdos nas *words* de parâmetros de forma consecutiva.
	02	03	*Parameter Area Fill*	Escreve os mesmos dados numa gama de *words* de parâmetros.
Program area access	03	06	*Program Area Read*	Lê a área de memória do utilizador.
	03	07	*Program Area Write*	Escreve na área de memória do utilizador.
	03	08	*Program Area Clear*	Limpa a área de memória do utilizador.
Operating mode changes	04	01	*Run*	Muda o modo de operação do CPU para RUN ou *Monitor*.
	04	02	*Stop*	Muda o modo de operação do CPU para Programa.
Machine configuration reading	05	01	*CPU Unit Data Read*	Lê os dados da unidade CPU.
	05	02	*Connection Data Read*	Lê os números do modelo do equipamento em formato endereço.
Status reading	06	01	*CPU Unit Status Read*	Lê o *status* da unidade CPU.
	06	20	*Cycle Time Read*	Lê o máximo, mínimo, e média de um ciclo de tempo.
Time data access	07	01	*Clock Read*	Lê o presente ano, mês, data, minuto, segundo e dia da semana.
	07	02	*Clock Write*	Muda o presente ano, mês, data, minuto, segundo e dia da semana.
Message display	09	20	*Message Read/Clear*	Lê e limpa mensagens

Access rights	**0C**	**01**	***Access Right Acquire***	**Adquire o direito ao acesso, desde que não haja outro dispositivo ligado.**
	0C	**02**	***Access Right Forced Acquire***	**Adquire o direito ao acesso, mesmo com outro dispositivo ligado.**
	0C	**03**	***Access Right Release***	**Liberta o acesso que tinha sido adquirido.**
Error log	**21**	**01**	***Error Clear***	**Limpa erros ou mensagens de erro.**
	21	**02**	***Error Log Read***	**Lê registos de erros.**
	21	**03**	***Error Log Clear***	**Limpa o ponteiro dos registos de erros.**

Anexo C Tabela de memórias OMRON

A tabela 5 apresenta as memórias OMRON que foram utilizadas no projeto.

Tabela 5 – Tabela memórias OMRON; adaptado de [31].

Área		Tipo de Dados	Modo CS/CJ			Modo CV			Comprimento por elemento
			Código Área Memória (hex)	Endereço Área Memória	Endereço Memória	Código Área Memória (hex)	Endereço Área Memória	Endereço Memória	
CIO Area	CIO	Bit	30	CIO 000000 to CIO 614315	000000 to 17FF0F	00	CIO 000000 to CIO 255515	000000 to 09FB0F	1
Work Area	WR		31	W00000 to W51115	000000 to 01FF0F	---	---	---	
Holding Bit Area	HR		32	H00000 to H51115	000000 to 01FF0	---	---	---	
Auxiliary Bit Area	AR		33	A00000 to A44715 (read only)	000000 to 01BF0F	00	A00000 to A44715 (read only)	0B0000 to 0CBF0F	
				A44800 to A95915 (read/ write)	01C000 to 03BF0F		A44800 to A95915 (read/ write)	0CC000 to 0EBF0F	
CIO Area	CIO	Word	B0	CIO 0000 to CIO 6143	000000 to 17FF00	80	CIO 0000 to CIO 2555	000000 to 09FB00	2
Work Area	WR		B1	W000 to W511	000000 to 01FF00	---	---	---	
Holding Bit Area	HR		B2	H000 to H511	000000 to 01FF00	---	---	---	
Auxiliary Bit Area	AR		B3	A000 to A447 (read only)	000000 to 01BF00	80	A000 to A447 (read only)	0B0000 to 0CBF00	
				A448 to A959 (read/write)	01C000 to 03BF00		A448 to A959 (read/write)	0CC000 to 0EBF00	

Timer Area	**TIM**	**PV**	**89**	**T0000 to T4095**	**000000 to 0FFF00**	**81**	**T0000 to T2047 (T0000 to T1023)**	**000000 to 07FF00 (000000 to 03FF00)**	**2**
Counter Area	**CNT**			**C0000 to C4095**	**800000 to 8FFF00**		**C0000 to C2047 (C0000 to C1023)**	**080000 to 0FFF00 (080000 to 0BFF00)**	
DM Area	**DM**	**Word**	**82**	**D00000 to D32767**	**000000 to 7FFF00**	**82**	**D00000 to D32767**	**000000 to 7FFF00**	**2**

Anexo D Principais Funções Desenvolvidas para a Biblioteca FINS

É pretendido com este anexo descrever as funções desenvolvidas dentro dos diferentes modos de funcionamento na biblioteca FINS.

Anexo D.1 Funções Desenvolvidas no Modo Interativo

Como foi referido anteriormente, é pretendido que este modo de funcionamento permita ao utilizador a consulta e escrita de dados nos diversos equipamentos que permitam comunicação FINS. Este modo de funcionamento comunica com os equipamentos através do protocolo FINS mediante o registo prévio dos dados em ficheiro.

Função *create_File*

Esta função permite criar um ficheiro para efetuar o registo dos respetivos dados de identificação do equipamento que esteja ligado na rede *Ethernet*. Este registo é gerado através de uma estrutura de dados que identifica os diferentes campos do dispositivo.

Esta estrutura identifica os seguintes campos no registo:

- Número de Identificação;
- Tipo de Dispositivo (*Arduino*, PLC, entre outros);
- Módulo *Ethernet*;
- Endereço IP;
- Número de porta FINS;
- Endereço de nó FINS;
- Endereço de rede FINS.

Função insert_Device

A função *insert_Device()* permite ao utilizador a inserção de dados de identificação relativos a um equipamento com o qual se pretenda comunicar futuramente, sendo os dados inseridos escritos no ficheiro que foi criado na função *create_File()*.

Função list_Device

A função *list_Device()* permite ao utilizador a listagem de todos os equipamentos que foram registados previamente em ficheiro. Através desta função é possível identificar o equipamento ao qual se pretender ligar para efetuar as tarefas necessárias.

Função delete_Device

Esta função foi desenvolvida com o intuito de eliminar o registo de um equipamento do ficheiro Equipamentos.txt. Este tipo de ação só é tomada quando já não se pretende executar tarefas sobre o respetivo equipamento ou quando o mesmo foi removido da rede *Ethernet*.

Função cliente_FINS

A função *cliente_FINS()* foi desenvolvida com o intuito de envio e receção de mensagens FINS com outro equipamento, funcionando a biblioteca FINS nesta situação como cliente. Esta troca de mensagens é realizada com recurso ao *socket* UDP que foi mencionado anteriormente no capítulo 5. Esta função é composta da seguinte forma:

int cliente_FINS(unsigned char fins_cmnd[],unsigned char fins_resp[], int sendlen,int num_cmd)

O primeiro parâmetro corresponde ao *array* de dados da trama de comandos FINS que é configurada para enviar de dados.

O segundo parâmetro diz respeito ao *array* de dados da trama de resposta FINS que resulta da receção de dados de outro equipamento que esteja ligado na rede, e que permitirá a verificação da viabilidade da trama de resposta FINS mediante a informação emitida na trama de comando FINS.

O parâmetro *sendlen* corresponde ao comprimento que os dados FINS ocupam na trama de comando FINS.

O parâmetro num_cmd corresponde ao número de vezes que se pretende efetuar a troca de mensagens FINS, tendo um valor superior a um quando se trata de comandos de leitura como é o caso do *Memory Area Read* e o *Multiple Memory Area Read*, e valor igual a um para os comandos de escrita.

De forma a evitar o bloqueio do *socket* UDP na situação de não receber uma trama de resposta FINS, foi implementada a "*system-call*" *ioctl* que permite a ativação da propriedade *FIONBIO* ("*File I/O NonBlocking I/O*"). Esta função é definida da seguinte forma:

*int ioctl(int sock, unsigned long request, char *arg).*

O parâmetro *sock* diz respeito ao *socket* criado para comunicações UDP.

O parâmetro *request* diz respeito ao identificador de caraterísticas, que se encontram definidos no ficheiro sys/ioctl.h. Como, neste caso a variável a utilizar é o *FIONBIO,* o parâmetro *arg* deve apontar para um valor inteiro, e que seja diferente de zero [48].

Esta função é utilizada da seguinte forma:

```
int sock, value=1;
sock=socket(AF_INET,SOCK:DGRAM,0);
ioctl(sock,FIONBIO,&value);
```

Função comandos_FINS

A função *comandos_FINS()* é responsável pelo armazenamento de dados de consulta ou execução que a biblioteca FINS venha a tomar sobre um equipamento que esteja ligado na rede *Ethernet*.

A implementação desta função é realizada da seguinte forma:

int comandos_FINS(unsigned char fins_cmnd[],unsigned char fins_resp[],int recvlen).

O parâmetro *fins_cmnd* corresponde à trama de comando FINS enviada pela biblioteca FINS.

O parâmetro *fins_resp* corresponde à trama de resposta FINS recebida pela biblioteca FINS.

O último parâmetro diz respeito ao comprimento de dados do parâmetro *fins_resp*.

Esta função é utilizada para fazer a distinção entre os comandos FINS descritos no capítulo 5, caso a biblioteca esteja a trabalhar no modo interativo, ou seja, como cliente.

Anexo D.2 Funções Desenvolvidas no Modo Servidor

Como referido no capítulo 5, este modo de funcionamento foi desenvolvido de forma a implementar a biblioteca FINS como um servidor, sendo capaz de responder a todos os equipamentos que transfiram dados para a plataforma FINS ou que consultem os dados armazenados na mesma. Este modo de funcionamento implementa o conceito *Thread* para verificação do equipamento que envia dados para a biblioteca FINS e armazena a informação, caso o equipamento esteja registado no ficheiro Equipamentos.txt.

Função Servidor

A função *servidor()* foi desenvolvida com o propósito de implementar na biblioteca FINS o funcionamento como servidor de dados FINS na rede. Esta função trabalha em *loop* e é responsável pela receção de tramas de comandos FINS e cria *Threads* de modo a poder responder a todos os equipamentos que se comunicam com a biblioteca FINS. Esta função está implementada da seguinte forma:

int servidor(void).

Quando a biblioteca FINS recebe dados externos de outro equipamento, os dados de identificação são armazenados numa estrutura desenvolvida para o efeito, de modo a transportar os parâmetros do equipamento externo e do *socket* para a função *Thread* que é gerada para o envio de dados.

Função threadMain

A função *threadMain()* foi implementada com o propósito de aplicar o conceito *Thread* na biblioteca FINS, de forma a conseguir atender a todas as ligações que são realizadas à plataforma FINS pelos equipamentos que se encontram ligados na rede. Esta função é executada através da função *pthread_create()*.

Esta função é implementada da seguinte forma:

*void *threadMain(void *p).*

O parâmetro *void *p* utilizado na função é o argumento utilizado para transferir os dados necessários para dentro da função *threadMain()*. Neste caso será utilizada a estrutura de dados mencionada na descrição da função *servidor()*.

Função serverFINS

Esta função foi desenvolvida com o intuito de verificar as tramas de comandos FINS, para que a biblioteca FINS defina a respetiva trama de resposta FINS a enviar ao equipamento que enviou dados anteriormente.

Esta função está implementada da seguinte forma:

*int serverFINS(unsigned char fins_cmnd[],unsigned char fins_resp[],int *sendlen,int recvlen).*

O primeiro parâmetro diz respeito à trama comando FINS recebida pela biblioteca FINS.

O parâmetro *fins_resp* diz respeito à trama de resposta FINS criada pela biblioteca FINS para responder ao equipamento que enviou dados previamente.

O parâmetro *sendlen* é a variável responsável por definir o comprimento de dados da trama de resposta FINS.

O parâmetro *recvlen* é o comprimento de dados da trama de comandos FINS recebida anteriormente pela biblioteca FINS.

Anexo D.3 Funções desenvolvidas no Modo PLC

Como foi visto no capítulo 5, este modo de funcionamento é implementado como o intuito de representar o comportamento de um PLC na biblioteca FINS. De forma a implementar as etapas de funcionamento através dos vários equipamentos, foi criado o ficheiro conf.txt para fazer a gestão dos equipamentos que podiam executar etapas através da alteração da memória das mesmas. Os dados dos equipamentos e das memórias de entrada e saída da biblioteca FINS são guardados em formato estrutura.

Função create_HostFile

Esta função permite criar um ficheiro para efetuar o registo dos respetivos dados de comunicação dos equipamentos de modo a fazer a distinção dos equipamentos que efetuam a escrita de dados na biblioteca FINS, e os equipamentos para os quais a biblioteca FINS envia dados mediante a execução de etapas.

Esta estrutura é definida da seguinte forma:

- Número de Comunicação;
- Tipo de Dispositivo (*Arduino*, PLC, entre outros);
- Endereço IP de Entrada;
- Memória de Entrada;
- Endereço de Memória de Entrada;
- *Bit* de Memória de Entrada;
- Endereço IP Destino;
- Endereço destino de nó FINS;
- Endereço destino de rede FINS;
- Memória de Saída;

- Endereço de Memória de Saída;
- Bit de Memória de Saída.

Função insert_Host

A função *insert_Host()* permite ao utilizador a inserção de dados relativos às comunicações que a biblioteca FINS poderá executar, fazendo a distinção entre os equipamentos responsáveis pela entrada de dados e o dispositivos para os quais se enviam os dados.

Função list_Host

A função *list_Host()* permite ao utilizador a listagem de todas as ligações entre equipamentos que foram registadas previamente em ficheiro.

Função delete_Host

Esta função foi desenvolvida com o intuito de eliminar o registo de uma ligação do ficheiro conf.txt. Este tipo de ação é tomada quando já não é necessária a utilização das memórias registadas na comunicação.

Função plc_mode

A função *plc_mode()* foi desenvolvida com o intuito de implementar as etapas de tarefas na biblioteca FINS. Caso alguma das etapas seja realizada, a *Thread* de comunicações é criada com o intuito de enviar uma trama de comando FINS para o equipamento destino. Na situação das etapas não serem realizadas, a *Thread* de comunicações é gerada com o intuito de receber tramas de comando FINS do equipamento origem.

Esta função é implementada da seguinte forma:

void plc_mode(void).

Função threadSocket

A função *threadSocket()* foi implementada com o intuito de utilizar o conceito *Thread* para comunicar com os diversos equipamentos na rede *Ethernet*. Esta função implementa o conceito *socket* UDP para envio ou receção de dados UDP, mediante a execução ou não das etapas na função *plc_mode().*

Esta função está implementada da seguinte forma:

*void *threadSocket(void *p).*

O parâmetro *void *p* é o argumento utilizada para definir os dados de identificação do equipamento, assim como os dados das tramas FINS a enviar ou a receber. Este parâmetro é criado em formato de estrutura.

Anexo E Funções utilizadas no *Arduino*

Anexo E.1 Funções utilizadas para comunicação do Servidor FINS Arduino

Na implementação do servidor FINS no *Arduino* foi utilizada a classe *EthernetUDP*, que esta plataforma eletrónica disponibiliza para comunicações com o protocolo UDP. Antes de o utilizador proceder à chamada das funções UDP é necessário declarar a instância UDP que servirá de estrutura para as funções, sendo declarada como: *EthernetUDP* [51].

As funções utilizadas foram as seguintes:

UDP.Begin:

Esta função é responsável pela inicialização da biblioteca *EthernetUDP* e opções de rede, sendo a sua sintaxe a seguinte:

Udp.begin(localPort).

O parâmetro *localPort* define a porta responsável pelas comunicações UDP.

UDP.parsePacket:

Esta função é responsável pela verificação da presença de pacotes UDP no *Arduino*, sendo a sintaxe desta função a seguinte:

Udp.parsePacket().

Esta função é declarada numa variável do tipo inteiro e reporta ao utilizador o tamanho dos dados.

UDP.remoteIP:

Esta função é responsável por registar o endereço IP do equipamento que envia pacotes de dados para o *Arduino*, sendo a sua sintaxe definida da seguinte forma:

Udp.remoteIP().

Esta função é atribuída geralmente a uma variável definida pela classe *IPAddress*.

UDP.remotePort:

Esta função identifica o número da porta UDP do equipamento remoto para o envio da resposta ao pacote de dados recebido, sendo a sua sintaxe a seguinte:

Udp.remotePort().

Esta função deve ser chamada depois da função *Udp.parsePacket*.

UDP.read:

Esta função possibilita ao *Arduino* a leitura do pacote de dados UDP, sendo a sua sintaxe definida da seguinte forma:

Udp.read(packetBuffer,MaxSize).

O parâmetro *packetBuffer* é responsável por manter os pacotes de dados em formato *char*, enquanto a variável *MaxSize* define o comprimento máximo do *packetBuffer*, sendo implementada em formato número inteiro.

UDP.beginPacket:

A função *UDP.beginPacket* é responsável pelo início da ligação para enviar a mensagem UDP para o equipamento remoto. A sua sintaxe é a seguinte:

Udp.beginPacket(Udp.remoteIP(),Udp.remotePort()).

A função *Udp.remoteIP()*, é responsável pela identificação do endereço IP equipamento destino, enquanto a função *Udp.remotePort()* identifica o número da porta do equipamento remoto.

UDP.write:

A função *UDP.Write* é utilizada pela escrita da mensagem UDP para o equipamento remoto, devendo esta função ser inicializada entre a função *UDP.beginPacket()* e a função *UDP.endPacket()*, sendo a sintaxe utilizada a seguinte:

UDP.write(buffer,size).

O parâmetro *buffer* define o conjunto de dados a enviar em séries de *bytes* ou *chars*, enquanto o *size* define o comprimento do conjunto de dados.

UDP.endPacket:

Esta função é responsável pelo envio da mensagem UDP para o equipamento remoto, sendo a sua sintaxe a seguinte:

UDP.endPacket().

Anexo E.2 Funções utilizadas para Servidor *Web Arduino*

Na implementação do servidor *Web* do *Arduino* foram utilizadas um conjunto de funções relativas à classe servidor e cliente da biblioteca *Ethernet*, de modo a implementar protocolo HTTP no *Arduino* para transportar as páginas HTML para o *browser* do utilizador [51].

Desta forma existem duas instâncias a declarar primeiro:

- *EthernetServer serverweb* (80): Cria o servidor que escuta a ligações que chegam. Visto que o protocolo usado é o HTTP, o número da porta é a 80.
- *EthernetClient client*: Cria o cliente que pode ligar-se ao *Arduino* através do seu endereço IP e número de porta. Para facilitar a utilização desta estrutura, atribui-se a função *server.available()*, que tem como tarefa o registo dos dados do cliente que liga-se ao *Arduino*, permitindo desta forma o envio das páginas HTML.

As funções utilizadas foram as seguintes:

client.connected:

Esta função verifica se existe um cliente ou não ligado ao *Arduino*. O cliente é considerado ligado caso a ligação seja encerrada, mas existam ainda dados que não foram lidos. A sua sintaxe é a seguinte:

client.connected().

client.available:

Esta função é responsável pelo retorno do número de dados disponíveis para leitura, sendo a sua sintaxe a seguinte:

client.available().

client.read:

Esta função permite a leitura dos dados recebidos do cliente que esteja ligado ao *Arduino*, sendo a sua sintaxe a seguinte:

client.read().

client.println:

A função *client.println* é utilizada para a apresentação das páginas HTML, assim como os dados que são apresentados na mesma página no cliente. A sua sintaxe é a seguinte:

client.println(data).

O parâmetro data é opcional e pode imprimir diversos tipos de dados (*char, byte, int, long* ou *string*).

client.stop:

Esta função é utilizada para encerrar a ligação do *browser* do utilizador com o *Arduino*, sendo a sua sintaxe a seguinte:

client.stop().

Anexo E.3 Armazenamento de dados através da memória EEPROM

A memória EEPROM do microcontrolador do *Arduino* é utilizada para o registo dos dados que o *Arduino* recebe para parametrização das memórias que são mapeadas nos seus pinos, assim como definir os endereços IP de comunicação através da página *Web*. Estes dados ficam registados mesmo com o *Arduino* desligado [50].

As funções utilizadas para interagir com a memória EEPROM foram as seguintes:

EEPROM.get:

Esta função permite a leitura de qualquer tipo de dados da memória EEPROM, sendo a sua sintaxe a seguinte:

EEPROM.get(address,data).

O parâmetro *address* indica a posição onde se inicia a leitura de dados na memória EEPROM, enquanto o campo data transfere para uma variável os dados lidos. Este campo pode ser constituído só por um valor ou por uma estrutura de dados.

EEPROM.put:

Esta função possibilita a escrita de qualquer tipo de dados na memória EEPROM, sendo a sua sintaxe a seguinte:

EEPROM.put(address,data).

O parâmetro *address* indica a posição onde se inicia a escrita de dados na memória EEPROM, enquanto o campo data são os dados que se pretendem escrever na memória EEPROM. Este campo pode ser constituído só por um valor ou por uma estrutura de dados.

Anexo F Esquemas dos Testes de Validação Experimental

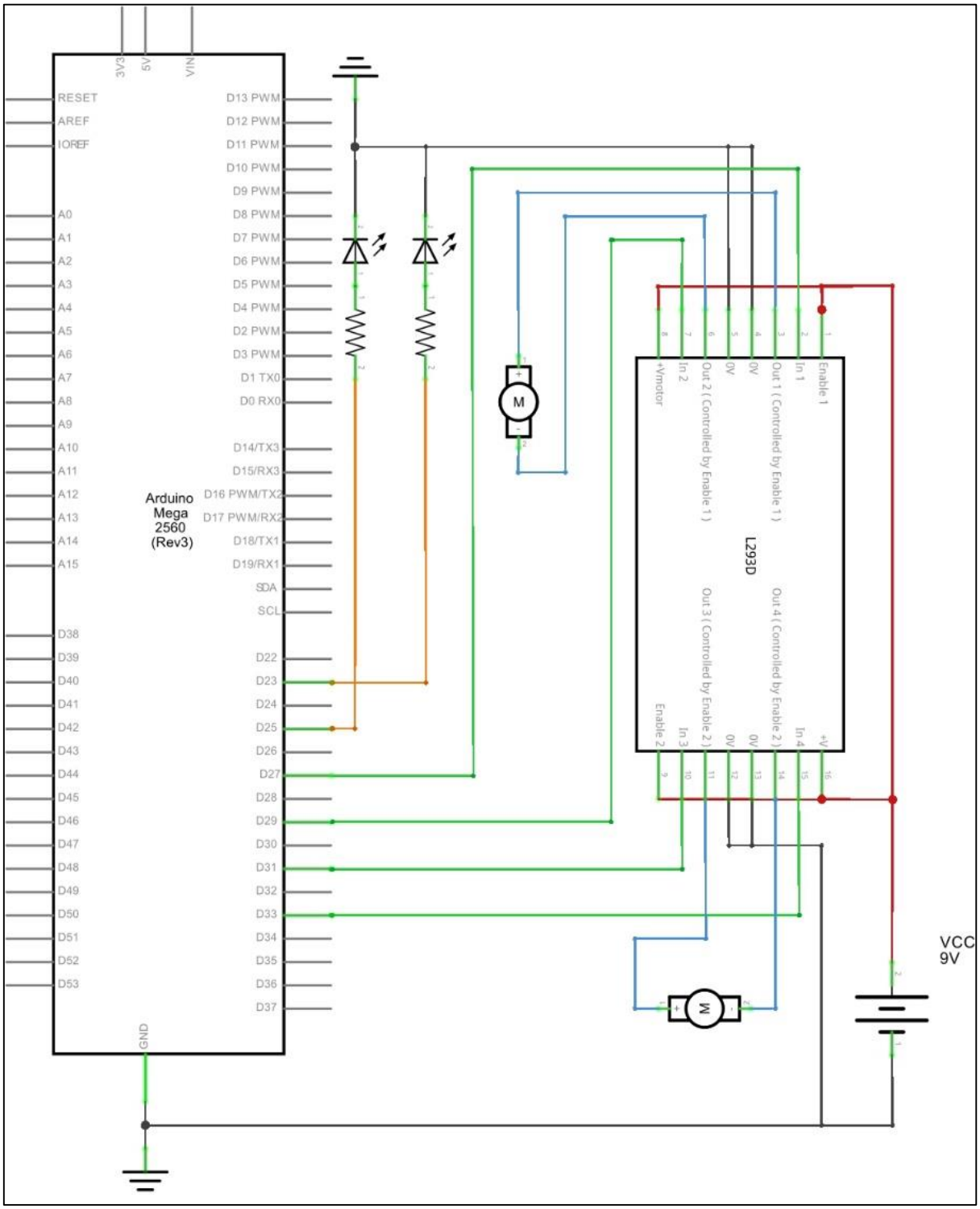


Figura 76 – Esquema de teste dos motores DC.

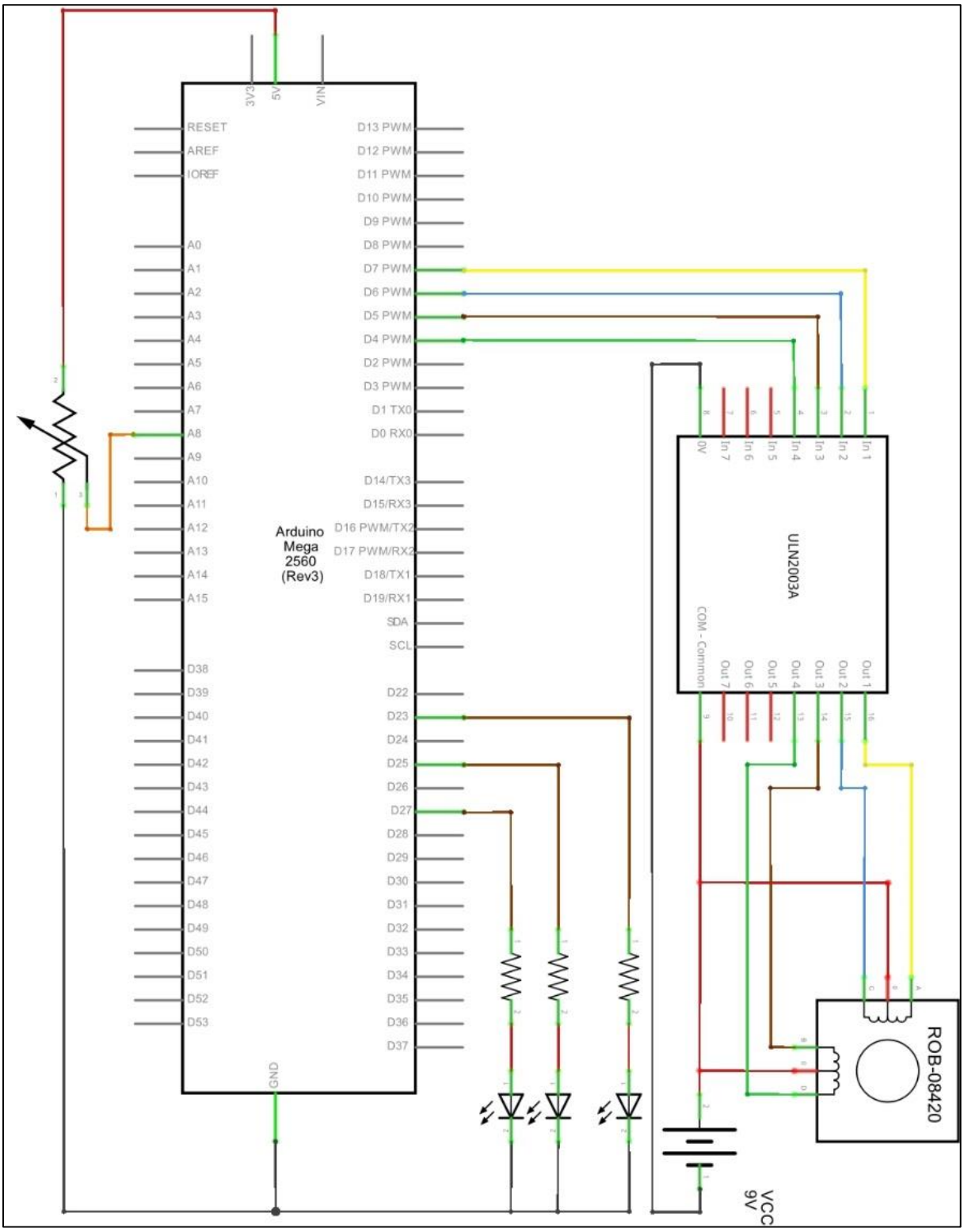


Figura 77 – Esquema de teste do motor de passo.