Centro Federal de Educação Tecnológica de Minas Gerais *Campus*
Divinópolis
Graduação em Engenharia Mecatrônica

Cauê Emilio de Moraes

Desenvolvimento de um sistema de múltiplos comandos e protocolo de comunicação industrial

Divinópolis
2021

Cauê Emilio de Moraes

Desenvolvimento de um sistema de múltiplos comandos e protocolo de comunicação industrial

Monografia de Trabalho de Conclusão de Curso apresentada ao Colegiado de Graduação em Engenharia Mecatrônica como parte dos requisitos exigidos para a obtenção do título de Engenheiro Mecatrônico.
Áreas de integração: Computação e Elétrica.

Orientador: Prof. Dr. Alan Mendes Marotta

Banca Examinadora:
Prof. Dr. João Carlos de Oliveira
Prof. Dr. Daniel Morais dos Reis

Divinópolis
2021

Resumo

O projeto visa o desenvolvimento de um sistema de comunicação residencial para um sistema de múltiplos comandos com o uso do protocolo industrial (*ModBus*) e implementação por padrão de comunicação diferencial (RS-485) e microcontroladores (ATMEGA328). A motivação se dá pela popularização da tecnologia necessária para instalação e controle da rede, assim como a ótima resiliência e flexibilidade do sistema em relação às alternativas. Neste trabalho será feito um estudo para o desenvolvimento de um um sistema de comunicação confiável para gerir uma rede de acionamentos em quatro prédios que possuem um painel de acionamentos central, com ênfase na sua aplicabilidade e capacidade de modulação. Como a automação permeia, atualmente, desde as indústrias até residências, pretende-se criar um sistema que seja adaptável para ser utilizado em outras aplicações que necessitem de uma comunicação cabeada segura e confiável.

Palavras-chave: *ModBus*, RS-485, Automação, Comunicação em Rede.

Sumário

Lista de Figuras

Capítulo 1

INTRODUÇÃO

Os protocolos de comunicação industrial estão em um processo de melhoria contínua pelas organizações responsáveis por cada um deles e são divulgados abertamente para todos que se interessarem por utilizá-los. A organização *ModBus* (MODBUS, 2021) é responsável por desenvolver e manter atualizados os protocolos de comunicação *ModBus*, assim como manter a infraestrutura para obter e compartilhar informações sobre os protocolos, suas especificações e certificações. O projeto residencial de uma rede com a utilização deste protocolo possui segurança e confiabilidade certificada pela organização, além de possuir um vasto acervo de referências e materiais disponibilizados pela mesma.

Segundo Thomas e McDonald (THOMAS; MCDONALD, 2015), o processo de automatizar consiste em garantir que o comando dado seja propriamente traduzido e transmitido pela rede e que se converta para a ação correta no campo. Este trabalho busca a utilização de um método de comunicação exaustivamente testado na industria para que se faça uma aplicação de baixo custo, quando comparado à aplicação industrial, em um ambiente residencial onde as demandas são menores, mas a confiabilidade é necessária. Isto significa que esta rede poderia ser utilizada para diversos tipos de aplicações distintas, como a implementação de uma rede para alarmes de incêndio em um prédio, um painel central de campainhas para um condomínio com diversos blocos, uma rede de aviso de chegada de correspondência para um condomínio, entre várias outras aplicações onde se possa utilizar um sistema com um único servidor capaz de acionar múltiplos clientes e comandos distintos.

Atualmente, as campainhas para condomínios se encontram em uma dicotomia de componentes muito sofisticados e caros ou não muito consistentes ou confiáveis, mas de um sistema legado. Os modelos de campainha e interfone para casas e apartamentos mais comuns cabeados no mercado (AMELCO, 2010);(INTELBRAS, 2020a), fazem o cabeamento com uma topologia em estrela, o que aumenta muito o valor e complexidade da instalação, visto que

necessitariam de uma grande quantidade de cabos.

O avanço tecnológico contribui para uma redução cada vez maior do custo de microcontroladores e por este motivo viabiliza a instalação de outras topologias para a comunicação, como a de barramento, que possibilita uma redução no custo de instalação. Para aproveitar esta mudança de paradigma, uma boa alternativa é o padrão RS-485, implementado por meio do protocolo de comunicação *ModBus*. Para Yong-jie e Zong-jin(YONG-JIE; ZONG-JIN, 2014) este padrão é confiável, possui alta escalabilidade e custo baixo.

1.1 Definição do problema

A aplicação proposta para este trabalho é a de um painel de controle para um conjunto de múltiplas campainhas para um condomínio de prédios, onde o projeto poderá ser validado em um sistema com capacidade de expansão e possibilidades de desdobramentos futuros.

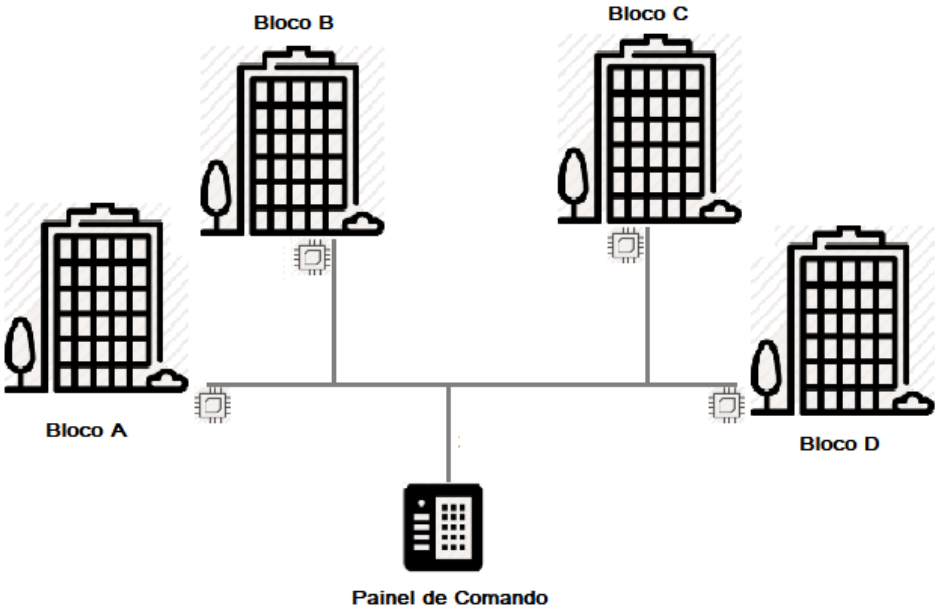

Figura 1.1: Esquemática do condomínio e conexão com a portaria central (*Fonte: Autor*[1])

O projeto será de um condomínio com quatro blocos e uma portaria central e interligadas à todos blocos, onde cada bloco possui oito apartamentos com uma campainha em cada, como mostrado na Figura 1.1. Para comandar estes acionamentos, será utilizada uma rede industrial de topologia em barramento *ModBus*, utilizando microcontroladores de menor custo, porém também com um poder computacional menor, que serão conectados às campainhas por meio da topologia estrela.

[1]A partir desta figura, as demais, cujas referências forem omitidas são de autoria do próprio autor.

1.2 Motivação

A automação se difunde à medida que a tecnologia avança, por ser um facilitador de processos que antes precisavam de um circuito elétrico muito complexo e uma grande quantidade de cabos e componentes. A difusão e o barateamento da produção de circuitos integrados, que antes eram caros e utilizados apenas em processos industriais, possibilitou sua utilização para resolver problemas industriais. Estas tecnologias industriais são muito mais confiáveis em projetos residenciais por possuirem uma demanda muito menor e protocolos robustos mantidos por organizações regulamentadoras globais.

Com a popularização de microcontroladores e de circuitos integrados, está cada vez mais barata e simples a aquisição de componentes que antes eram utilizados apenas na indústria. Estes componentes podem ser utilizados para criar uma rede de comunicação com padrão industrial confiável que não precisa de supervisionamento constante, como o que ocorre com as redes sem fio. Com a utilização do protocolo *ModBus*, é possível se instalar uma rede de múltiplas campainhas com uma topologia que necessita de uma quantidade menor de cabos para interligá-las, gastando menos materiais para a implementação do que as disponíveis no mercado, como pode ser visto em (AMELCO, 2010) e (INTELBRAS, 2020a).

Ao mesmo tempo, há um aumento do uso de IoT (Internet das Coisas) para aplicações residencias e soluções de campainha e monitoramento. Estas soluções são muito sensíveis à ruídos já que os sinais são transmitidos pelo ar, o que reduz a sua consistência em ambientes urbanos muito movimentados. Por este motivo, uma solução cabeada e de fácil manutenção pode funcionar até como uma rede auxiliar, caso ocorra algum problema na rede sem fio.

Este trabalho então é feito para construir uma base robusta em que um sistema cabeado com protocolo de comunicação industrial é utilizado para um sistema de múltiplas campainhas. O sistema é flexível ao ponto de poder ser utilizado para até 32 blocos sem a necessidade de um repetidor de sinal para cada rede e mais de 20 apartamentos por bloco. Embora este trabalho tenha ênfase no sistema cabeado e nas redundâncias para manter a confiabilidade e robustez da comunicação, é preciso destacar que o sistema também permite a expansão para o uso de IoT, já que o protocolo ModBus é feito com a capacidade de comportar o uso do padrão TCP/IP.

1.3 Objetivos

Esta seção informa os objetivos gerais e específicos, a fim de explicar o objetivo almejado ao final do projeto e as etapas nas quais o mesmo é dividido ao longo de seu desenvolvimento.

1.3.1 Objetivos Gerais

O seguinte trabalho têm como objetivo desenvolver um sistema de múltiplos comandos e acionamentos utilizando o protocolo de comunicação *ModBus*, padrão RS-485 e também a programação dos microcontroladores que ditam a dinâmica de comunicação e o gerenciamento da rede. Logo, pretende-se projetar um sistema com um menor custo e confiável que possa ser utilizado comercialmente, sendo desenvolvido para a aplicação de múltiplas campainhas em condomínio.

1.3.2 Objetivos Específicos

Esta sessão divide o projeto nas principais partes necessárias para o desenvolvimento do mesmo, sendo elas:

- Projetar e simular a Camada Física
- Desenvolver a Camada de Dados
- Programar os microcontroladores
- Aplicar e testar a rede e o transporte de dados
- Simular a comunicação entre os microcontroladores
- Desenvolver e testar o protótipo
- Avaliar o custo do projeto

1.4 Estado da Arte

A disseminação cada vez maior de objetos inteligentes, faz com que haja um crescimento na instalação de campainhas inteligentes e que se utilizam da rede sem fio. Ainda que esta seja uma alternativa viável para condomínios de luxo ou de grande porte, a necessidade de um constante gerenciamento de rede, servidores e aplicativos dedicados e constante conexão com a internet eleva muito o custo de manutenção deste serviço. Além disso, como a comunicação ocorre pelo ar e o ambiente residencial não é controlado como o ambiente industrial, há uma grande quantidade de ruído e a possibilidade dos canais disponíveis nas bandas de rede sem fio estarem muito ocupados. A alternativa de menor custo de funcionamento presente no mercado são produtos como a central de portaria *Comunic 48* (INTELBRAS, 2020a) da *Intelbras* e o porteiro predial AM-PPR (AMELCO, 2010) da *Amelco*, que não precisam do

gerenciamento de rede e são seguros e confiáveis. Estes produtos, embora tenham um custo elevado de instalação, por serem compostos por uma rede cabeada, eliminam a necessidade de gerenciamento externo de rede.

Atualmente, os protocolos de comunicação industriais são muito sofisticados e resilientes, com uma alta aplicabilidade o que leva diversas empresas se dedicarem a mantê-los atualizados e com fácil acesso, devido a competição entre os fabricantes.

A organização *ModBus* (MODBUS, 2021) é composta por mais de 40 empresas que regulamentam, padronizam e documentam a aplicação do protocolo, o tornando confiável e com uma fonte segura e centralizada. O uso do protocolo também traz consigo a vantagem de ser um protocolo amplamente estudado e em processo de melhoria contínua, tendo uma bibliografia extensa sobre a rede e sua interação com outros protocolos.

1.5 Organização do Documento

Este trabalho está dividido em 5 capítulos distintos que abordam, respectivamente, a introdução sobre o projeto, os fundamentos necessários para o desenvolvimento deste trabalho, seguido das etapas necessárias para tal desenvolvimento, os resultados obtidos ao longo do projeto e, finalmente, as conclusões obtidas. O presente capítulo introduz, contextualiza e expões as motivações para o desenvolvimento do trabalho. O segundo capítulo fundamenta o trabalho, por meio de referências bibliográficas que embasam o projeto em outras pesquisas, trabalhos, artigos e documentações técnicas. O terceiro capítulo expõe o desenvolvimento do projeto, desde as escolhas de componentes, ambientes de desenvolvimento e bibliotecas de *software* utilizadas para a construção do trabalho, assim como as justificativas para tais escolhas. O quarto capítulo mostra os resultados do projeto. No quinto e último capítulo são descritas as conclusões que podem ser derivadas do projeto e indicados trabalhos futuros que podem ser desenvolvidos a partir do que foi projetado neste trabalho.

Capítulo 2

FUNDAMENTOS

O presente capítulo tem como intuito a descrição dos principais fundamentos utilizados para o desenvolvimento deste trabalho. Inicialmente, será feita uma revisão de literatura sobre o tema de redes computacionais e automação de sistemas e então descrita a fundamentação teórica necessária para o desenvolvimento do projeto.

2.1 Revisão de Literatura

O estudo de telecomunicações se iniciou no século XIX, com avanço do conhecimento sobre eletromagnetismo e o surgimento do telégrafo. Segundo Pires (PIRES, 2006), houve uma grande expansão do uso das telecomunicações ao longo do século XX, impulsionando o surgimento de novas tecnologias como a comunicação por ondas de rádio em 1907, a invenção do PCM (*Pulse Code Modulation*) por Alec Reeves, que tornou possível a digitalização dos sinais em 1936 e finalmente o conceito da comutação de pacotes apresentado por P. Baran na década de 60. Todo este avanço tecnológico demandou a normalização das telecomunicações e diversas organizações possuíram este papel no cenário mundial desde esta época, algumas se renomeando e funcionando até os dias atuais.

A União Internacional de Telecomunicações (UIT) organiza atualmente grupos de estudos para manterem atualizadas estas normas de telecomunicações (UIT, 2021). A área de telecomunicação é ampla e se utiliza de diversas tecnologias que são regulamentadas e normalizadas por outras organizações globais como a ISO (*International Standards Organization*) e a IEEE (*Institute of Electrical and Electronic Engineers*). Segundo Alani (ALANI, 2014), a ISO foi a responsável pela regulamentação do modelo OSI, que é utilizado até hoje para o desenvolvimento de redes de comunicação por causa de sua capacidade de expansão e adaptação.

Segundo o museu da história da computação (LAWS, 2021), a descoberta de Russell Ohl sobre o comportamento do silicone leva à criação do primeiro semicondutor na década de 40, o que dá inicio à trajetória dos microprocessadores. Durante a década de 60, por causa da corrida espacial, houve um alto investimento em microprocessadores, já que os mesmos seriam necessários para guiar as aeronaves. Este investimento levou à primeira aplicação de um circuito integrado à uma aeronave, já que o seu alto custo era justificado por ser mais leve, menor e consumir menos energia.

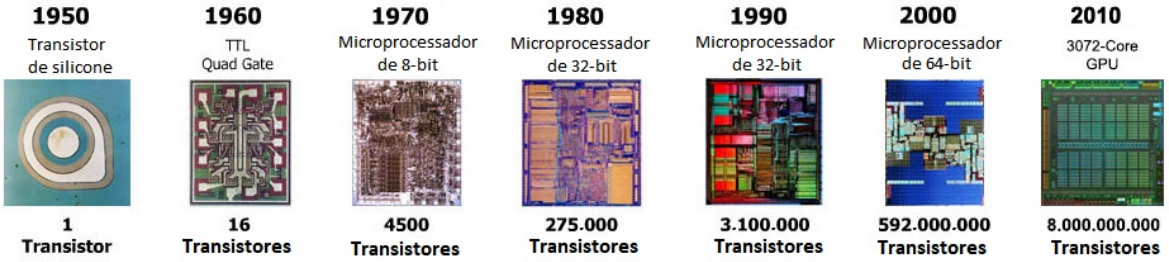

Figura 2.1: Microprocessadores ao longo das décadas (Imagens fora de escala) *Fonte: Computer History Museum* (LAWS, 2021)

Gradativamente, foram sendo descobertos novos materiais e tecnologias para a confecção de semicondutores, assim como arquiteturas para melhor aproveitar o espaço físico dos circuitos. Atualmente, os microprocessadores são muito menores e possuem uma capacidade de processamento altíssima, além de possuírem baixo custo para operações que necessitam de menos processamento. Na Figura 2.1 é possível ver a melhoria do processamento com o passar do tempo de acordo com a quantidade de transistores dentro de cada microprocessador.

2.2 Fundamentação Teórica

Nesta seção, será discutido com mais profundidade os principais conhecimentos necessários para o desenvolvimento do projeto, desde o modelo de rede escolhido, passando pela topologia de rede utilizada para o desenvolvimento e, por último, o protocolo de comunicação adotado.

2.2.1 Modelo OSI

Para o desenvolvimento deste trabalho, será desenvolvida uma rede para a transmissão de dados entre os microcontroladores. Esta rede é dividida em camadas, pois para Alani

(ALANI, 2014), esta divisão funcional da rede serve múltiplos propósitos: simplifica o entendimento, a implementação, a análise de erros, facilita sua construção e ainda garante consistência em suas funções e protocolos. Padronizado no final da década de 70, o modelo OSI continuou sendo constantemente adaptado e melhorado, tendo uma revisão mais intensa em 1995 devido ao avanço da computação.

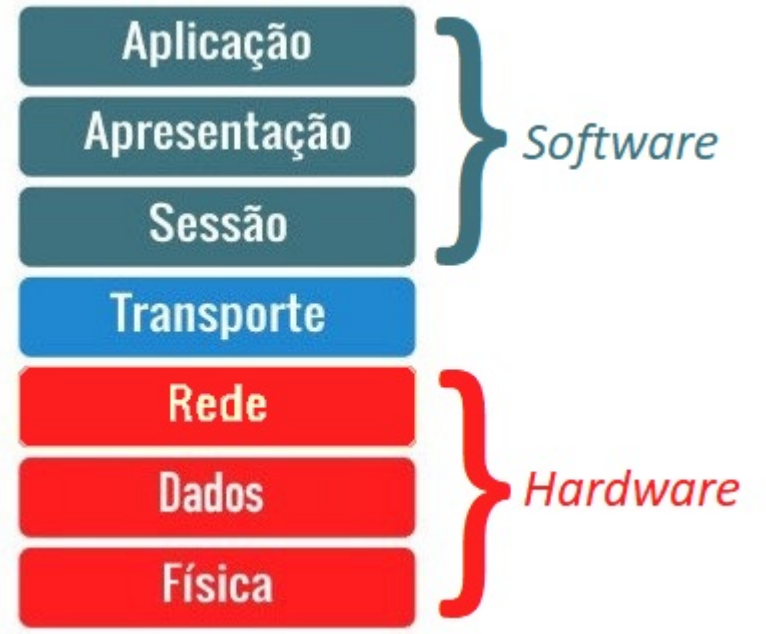

Figura 2.2: Divisão do modelo OSI

O modelo OSI, como mostrado na Figura 2.2, é separado em sete camadas distintas, cada uma aborda uma etapa da movimentação e estruturação dos dados na rede e quanto maior o número de camadas necessárias para descrever a rede, maior a complexidade de implementação (ALANI, 2014). Começando pelo nível mais básico, a primeira camada é a *Física*, o circuito elétrico que compõe o sistema que será interligado a rede. O segundo nível, de *Dados*, é referente à comunicação entre as camadas físicas e a correção de erros vindos desta camada, então cada componente da primeira camada é considerado um nó que é interligado por componentes da segunda camada. O padrão *RS-485* é contido nestas duas primeiras camadas, por se tratar da construção física e do enlace do protocolo *ModBus*, como descrito pelos responsáveis pela padronização da rede (MODBUS, 2021).

A terceira camada é a *Rede* e é relacionada a ligação entre diferentes enlaces e é onde são presentes os roteadores de sinais. A próxima camada, a de *Transporte*, faz a comunicação entre as camadas de *Hardware* e de *Software* e ainda trata da segurança, controle de erros e confirmação e confiabilidade de conexão. Para o protocolo *ModBus*, nenhuma dessas ca-

madas é necessária já que as mesmas são utilizadas quando o protocolo possui uma maior complexidade de rede e de transporte dos dados.

A quinta camada, de *Sessão*, é onde é feita a transmissão entre dois computadores distintos, coordenando a comunicação e delimitando pontos de marcação como referência, caso a comunicação seja interrompida. A camada de *Apresentação* é a sexta camada e é responsável pela compressão, tradução ou criptografia dos dados vindos da camada sete. Tanto a camada cinco como a camada seis devem ser implementadas no protocolo *ModBus* caso haja a utilização do protocolo TCP/IP na rede. Na camada de *Aplicação* os dados são gerados pelos usuários, já que nesta camada é onde há a interação homem-máquina e, por este motivo, que há a modificação dos dados na camada seis, para que a comunicação seja mais rápida e segura.

2.2.2 Topologia de redes

Os processos de comunicação industrial são implementados utilizando uma topologia de rede que mais facilite a comunicação entre os processadores e sensores, pensando na dinâmica de troca de informações requerida da rede.

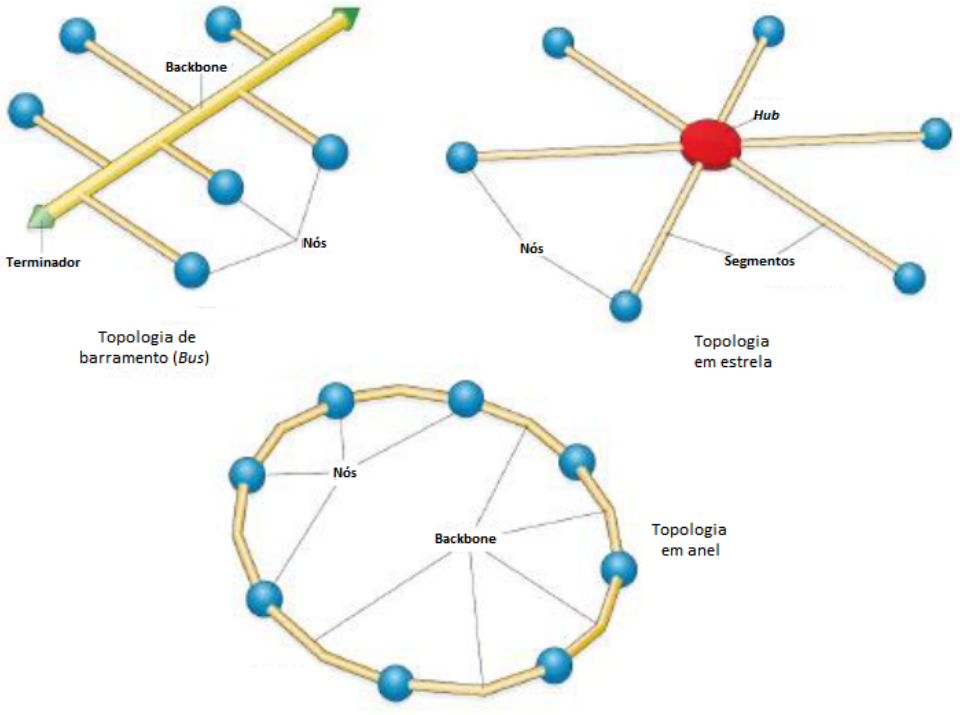

Figura 2.3: Topologias de redes mais comuns

Existem três principais maneiras de se interligar uma rede na industrial: Anel, Estrela e

Barramento que podem ser observadas na Figura 2.3. Cada uma das topologias traz consigo limitações e vantagens que podem ser consideradas ou exploradas pelo projetista da rede.

A topologia em anel conecta a rede de maneira circular, fazendo que seja necessário um *token* de prioridade para que seja definido quem enviará dados na rede. Esta topologia é interessante para redes onde os servidores (nós) tenham todos os mesmos níveis de prioridade na comunicação, podendo assim alternar o acesso ao *token* entre os nós. Portanto, esta topologia não é prática para a aplicação no sistema de campainhas, pois apenas um nó, o painel de botões, que necessita de prioridade, já os nós de cliente, os dispositivos sonoros, apenas precisam trocar informações se requeridos pelo painel de controle.

No mercado, a topologia mais utilizada é em estrela, como pode se observar nos produtos da *Amelco* (AMELCO, 2010) e *Intelbras* (INTELBRAS, 2020b). A vantagem desta topologia é que o porteiro eletrônico funciona como o *Hub* e se conecta simultaneamente à todos os pontos da rede e o rompimento de um cabo afeta apenas um cliente. Em contrapartida, é valido ressaltar que é necessário interligar individualmente a central com cada ponto, necessitando de uma grande quantidade de cabos, o que eleva o custo e a complexidade de implementação para projetos com múltiplos blocos ou os mesmos distantes da portaria central.

Finalmente, existe a topologia de barramento onde são feitas conexões com a rede a partir de um *backbone* e cada transceptor se conecta eletricamente de maneira paralela ao barramento. Neste tipo de topologia é possível levar o barramento aos blocos e depois interligá-los com cada apartamento por meio da topologia em estrela, reduzindo assim a quantidade de cabos necessária. Diferentemente da topologia em estrela, todos os blocos são interligados ao mesmo barramento de comunicação, composto de três fios de cobre de $0,75mm$, sendo dois para a comunicação e um comum para reduzir os ruídos eletromagnéticos, necessitando apenas de uma identificação na camada de aplicação para diferenciar cada apartamento. Como o painel de controle (Servidor) e o microcontrolador em cada bloco (Clientes) estão ligados e se comunicando constantemente, é possível se verificar a integridade dos cabos de comunicação como parte da rotina de comunicação, aumentando a robustez do sistema.

2.2.3 Protocolo *ModBus*

O protocolo de comunicação *ModBus* pode ser aplicado de diferentes maneiras na rede de comunicação, como visto na Figura 2.4. A comunicação pode ser direta entre a camada de aplicação e o cliente/servidor e também pode ser traduzida para outros protocolos como o TCP/IP e transmitida por uma rede Ethernet ou pela camada física como discutido em *ModBus.org* (MODBUS, 2021).

Para a comunicação com outros protocolos, como o TCP/IP, é necessária a utilização de um manual de implementação para o protocolo de mensagem, que padroniza a tradução entre protocolos. Porém, como dito anteriormente, a utilização de TCP/IP aumenta a complexidade de desenvolvimento do projeto por precisar utilizar mais duas camadas do modelo OSI e não será abordado neste trabalho.

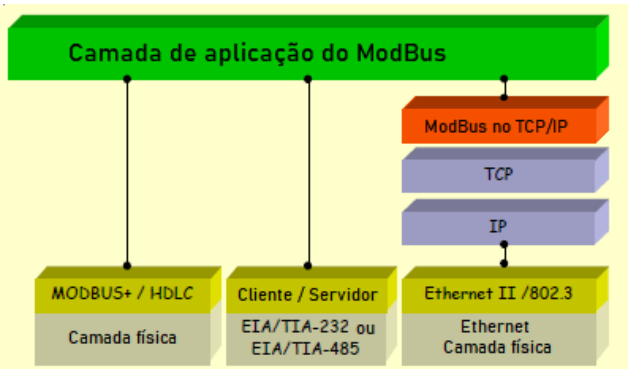

Figura 2.4: Protocolo ModBus utilizando o modelo OSI (*Fonte: ModBus Organization* (MODBUS, 2012))

Na Figura 2.4 é possível se relacionar o protocolo *ModBus* com as camadas nas quais é dividido o modelo OSI. Em verde e no topo da figura, está representada a sétima camada, onde o protocolo é aplicado de acordo com *ModBus.org* (MODBUS, 2012). Em vermelho e azul estão representadas as camadas seis e cinco, respectivamente, onde os dados são traduzidos e então transmitidos para dispositivos que se utilizam do protocolo TCP/IP, como dispositivos sem fio. Na cor amarela está representada a camada de dados, onde a comunicação serial é feita em mais baixo nível e junto com a primeira camada, em cinza, sua implementação é também padronizada (MODBUS, 2012). Como pode ser observado, neste projeto se utilizará apenas três camadas (Um, dois e sete) do modelo OSI, o que reduz a complexidade do projeto e as fragilidades que poderiam surgir no mesmo.

Na comunicação *ModBus*(MODBUS, 2012) apenas um servidor e 247 clientes —que devem possuir um endereçamento individual— podem enviar dados na rede a cada momento e a comunicação deve sempre ser iniciada pelo servidor. Os clientes não se comunicam entre si e não mandam informações à linha sem a requisição do servidor, com o servidor iniciando apenas uma transição de cada vez e possuindo duas maneiras distintas de comunicação com os clientes, uma individual(*Unicast*[1]) e outra coletiva (*Broadcast*[2]). Para a comunicação

[1]Todas as comunicações feitas pelo servidor são enviadas para todos os clientes, mas apenas o cliente com o endereçamento correto na comunicação *unicast* que consegue acessar e decodificar a mensagem, já que cada cliente possui um endereço único.

[2]Para a comunicação *broadcast* é atribuído o endereço zero e todos os clientes tem a permissão de deco-

Unicast se utiliza o endereço individual do cliente e requer uma resposta do mesmo, já a comunicação *Broadcast* não recebe uma resposta dos clientes, mas se envia uma mensagem simultânea para todos clientes.

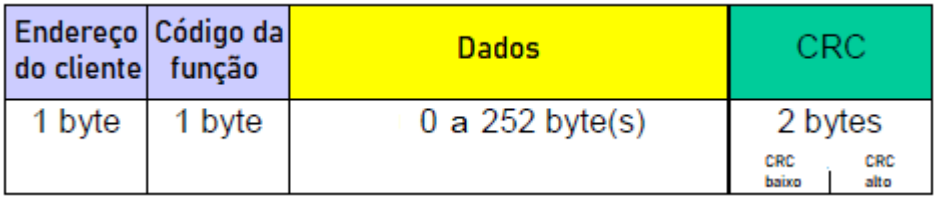

Figura 2.5: Composição do bloco de informações transmitido na rede *ModBus* (*Fonte: ModBus Organization* (MODBUS, 2012))

Como mostrado na Figura 2.5, a parcela de endereçamento do *ModBus* tem o tamanho de um *byte*, com o valor zero utilizado para *Broadcast*, os valores de um a 247 para o endereço de cada cliente, os valores de 248 a 255 são reservados. Não há um endereço para o servidor, pois o mesmo é único na rede e o cliente retorna mensagens enviadas a ele com seu próprio endereço para que o servidor saiba de qual cliente veio a mensagem.

O próximo trecho da série é a função que será aplicada, seguida dos dados enviados, sendo que a função tem o tamanho de um *byte*. Os dados possuem um comprimento de 252 *bytes* no modo de transmissão RTU (Remote Terminal Unit) e de 504 *bytes* no modo de transmissão ASCII(*American Standard Code for Information Interchange*).

Por último é feita a checagem cíclica de redundância (CRC) que possui dois *bytes* de tamanho, sendo que o primeiro *byte* é composto dos algarismos menos significativos e o segundo com os mais significativos. O valor do CRC é calculado com base na mensagem que será enviada e recalculado com a mensagem recebida, fazendo uma comparação entre ambos para detectar erros de transmissão.

2.2.4 Comunicação RS-485

A norma TIA/EIA-485, popularmente conhecido como RS-485, descreve uma interface de comunicação que atua nas duas primeiras camadas do modelo OSI, como relatado pela organização *ModBus* (MODBUS, 2012). Existem diversos modelos de transceptores de baixa potência que podem ser utilizados para as aplicações de RS-485, como os modelos descritos na ficha de dados da empresa *Maxim Integrated* anexado no Apêndice A. Os transceptores

dificar esta mensagem.

da *Maxim* podem se comunicar, em sua maioria, com outros 32 dispositivos na mesma trilha de dados e com uma frequência de transmissão de 250 $Kbps$.

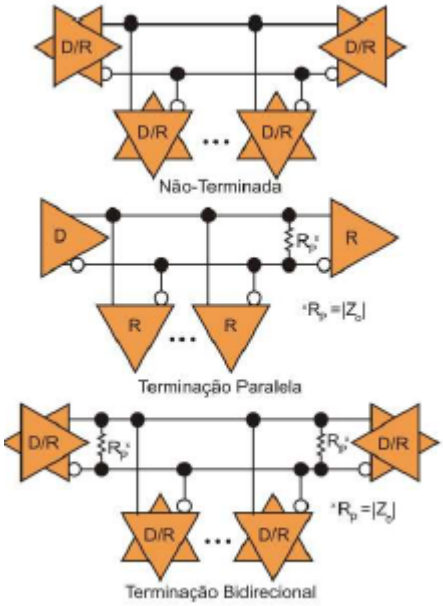

Figura 2.6: Terminações possíveis para o RS-485

Para a comunicação entre drivers (D) e receptores (R) há a necessidade de haver uma impedância entre o par trançado de acordo com a ficha de dados do dispositivo (MAXIM, 2014). Como mostrado na Figura 2.6, há como implementar a comunicação sem a impedância, mas isto limita a velocidade de transmissão e a distância máxima sem ruídos de transmissão. Esta aplicação tem a vantagem de um menor custo, mas o tamanho máximo do cabo sem interferência é de apenas 100 m e há uma sensibilidade maior à ruídos de transmissão.

Com o uso da terminação paralela, o sistema fica limitado à um transmissor, sendo que os outros dispositivos limitam-se a receber dados. Esta ligação *full-duplex* não é aceita por todos os modelos de transceptores (MAXIM, 2014). Com a terminação bidirecional, impedância em ambos extremos do par trançado, é possível que todos os transceptores conetados à rede recebam e transmitam dados com a contrapartida do aumento do consumo de energia pela rede.

A convenção de normas para a implementação do *ModBus* (MODBUS, 2012) possui dois níveis distintos para a conformidade com a norma: incondicionalmente conformado e condicionalmente conformado. O manual de implementação possui algumas demandas que podem ser ignoradas em determinadas aplicações, mas se feito, deve se justificar o porquê desta escolha e isto é chamado no manual de conformidade condicional. A conformidade

incondicional é quando todas as demandas obrigatórias e recomendadas são utilizadas no projeto.

Para a implementação incondicional do protocolo *ModBus*(MODBUS, 2021), o padrão RS-485 é o demandado, podendo ser feita a implementação de outros padrões em conjunto com ele.

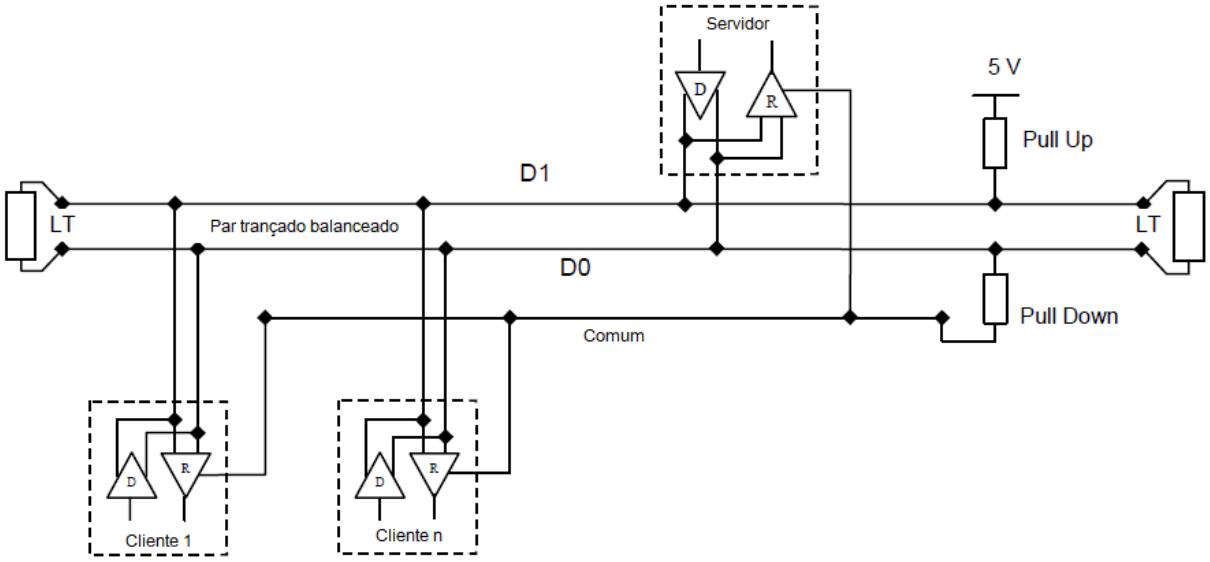

Figura 2.7: Topologia genérica requerida para implementação do protocolo *ModBus* (*Fonte: ModBus Organization* (MODBUS, 2012))

A terminação demandada para o protocolo, mostrada na Figura 2.7, é a bidirecional, gerada por um par trançado balanceado e um fio comum, que possibilita o envio de informações por qualquer transceptor. É requerida a implementação das taxas de transmissão de $9600Bps$ e $19200Bps$, sendo a segunda o valor padrão do protocolo.

ModBus RTU

Conforme determinado pela organização *ModBus* (MODBUS, 2012), o protocolo de mensagem de Unidade Terminal Remota (RTU) é um protocolo obrigatório a ser implementado. A implementação de outros protocolos de comunicação como o ASCII e o TCP(*Transmission Control Protocol*) podem também serem feitas, caso sejam requiridas por outros dispositivos da rede, mas suas implementações não são obrigatórias. O protocolo RTU se baseia na comunicação serial assíncrona e, por este motivo, cada *byte* enviado possui também um *bit* de inicio, um *bit* de paridade(WAKERLY, 2005) e um *bit* de fim, tendo, portanto, um comprimento de 11 *bits* por *byte* enviado, como mostrado na Figura 2.8.

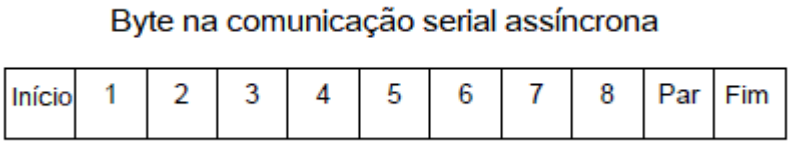

Figura 2.8: Sequência do *Byte* no protocolo RTU (*Fonte: ModBus Organization* (MODBUS, 2012))

Na comunicação serial assíncrona, o inicio da comunicação é sempre em nível baixo, seguido do *byte* de informação enviado, o *bit* de paridade e terminado em nível alto. Para o protocolo RTU, a paridade é obrigatoriamente par, podendo também ser implementado sem paridade e com dois *bits* de fim ou com paridade impar caso necessário e justificado. Cada *byte* transmitido também segue a ordem do algarismo menos significativo para o mais significativo.

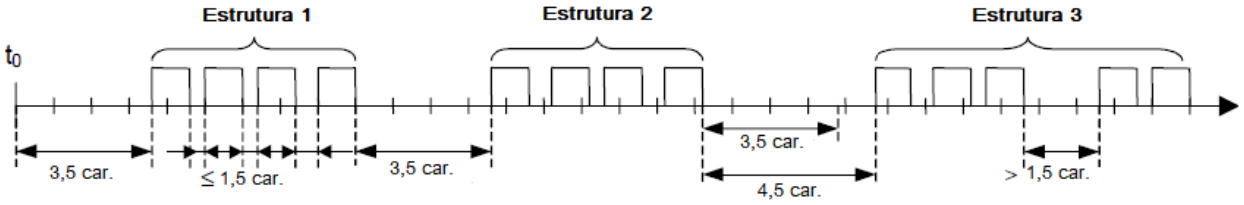

Figura 2.9: Intervalos internos e entre mensagens no protocolo RTU (*Fonte: ModBus Organization* (MODBUS, 2012))

A Figura 2.9 mostra os padrões do protocolo para mensagens enviadas pela rede, que é enviada *byte* a *byte*, em uma sequência constante de caracteres e deve possuir uma distância máxima de 1,5 caractere($t_{1,5}$) entre cada *byte* dentro da estrutura e uma distância mínima de 3,5 caracteres($t_{3,5}$) entre cada estrutura de mensagem. Esta distância em caracteres se dá pelo tempo despendido para a transmissão de cada *byte*, calculado com base na velocidade de transmissão permitida pelo transceptor (MAXIM, 2014), que pode variar entre $1KBps$ e $10MBps$, dependendo do comprimento do fio, características dos dispositivos instalados e resistores terminação. A limitação pelas distâncias $t_{1,5}$ e $t_{3,5}$ podem causar muitas interrupções durante a comunicação e elevar a demanda do microprocessador, por este motivo é recomendado um valor fixo de $t_{1,5} = 750\mu s$ e $t_{3,5} = 1,75ms$ para velocidades de transmissão superiores à $19200Bps$, mas obrigatório o valor de 1,5 e 3,5 caracteres, respectivamente, para velocidades iguais ou inferiores a esta.

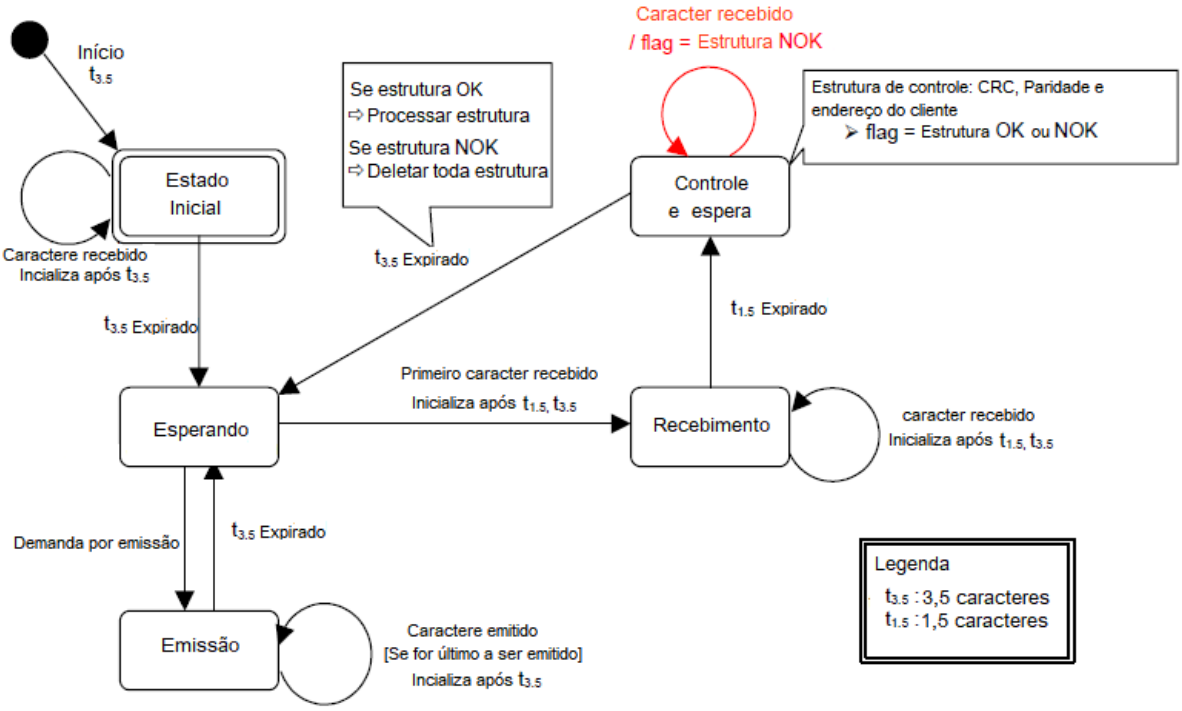

Figura 2.10: Diagrama de comunicação no protocolo RTU (*Fonte: ModBus Organization* (MODBUS, 2012))

Como visto na Figura 2.10, o processo de comunicação tem um fluxo bem claro de informação baseado no tempo de espera visto anteriormente. Com base nestes períodos de espera é possível identificar o início da estrutura da mensagem, falhas na mensagem e detectar o fim de cada mensagem, possibilitando a mudança de estado apenas quando todo conteúdo foi transmitido e aferido. Sempre que o sistema não está recebendo ou emitindo, o mesmo se encontra no estado "Esperando"e o calculo do CRC é feito apenas quando é detectado o fim da mensagem (após um tempo de $t_{3,5}$).

Calculo da checagem cíclica de redundância

Ao final de cada pacote de dados enviado pelo protocolo RTU, há uma checagem de erro feita com base nas informações da mensagem (MODBUS, 2012). Esta checagem é feita independentemente da conferência da paridade que ocorre em cada *byte* de dado da mensagem, mas o CRC se utiliza apenas dos 8 *bits* da mensagem e ignora os *bits* de inicio, paridade e fim. Há um procedimento de oito passos para a construção do CRC, como mostrado abaixo.

1. Carregar todos os *bits* registrador de 16 *bits* com nível alto (1)

2. Realizar um ou exclusivo (*XOR*) entre os oito primeiros *bits* da mensagem e os bits de baixa ordem do CRC

3. Deslocamento em um *bit* para a direita no registrador CRC, preenchendo o *bit* mais significativo com zero e extraindo e analisando o *bit* menos significativo

4. Caso o bit menos significativo seja zero, o passo 3 é repetido. Caso seja um, é feito um *XOR* entre o CRC e o valor 1010 0000 0000 0001

5. Repetir os passos 3 e 4 até oito deslocamentos serem feitos

6. Repetir dos passos 2 ao 5 para os próximos 8 *bits* da mensagem até que todos os *bytes* tenham sido processados

7. O conteúdo final no registrador é o valor do CRC

8. O CRC deve ser anexado à mensagem com o último *byte* da mensagem sendo o *byte* de ordem mais alta, invertendo assim o primeiro e o segundo *byte* do CRC

Como dito anteriormente, o valor do CRC é calculado em duas etapas do processo, antes da mensagem ser enviada pelo transmissor e logo após a mensagem ser recebida pelo receptor. A comparação feita após a transmissão protege o processo de falhas na transmissão de uma maneira mais robusta do que apenas a checagem de paridade de cada *byte* e que a checagem de tempo entre as mensagens. Para a redução de processamento, é possível o receptor aferir o *byte* de endereço antes do cálculo do CRC, para que só seja feita a averiguação caso a mensagem seja do endereço esperado, no caso do servidor, ou seja endereçada para o próprio, no caso do cliente.

2.2.5 Camada de Aplicação no *ModBus*

A camada de Aplicação do protocolo é implementada com base nas escolhas de projeto feitos para as outras camada do modelo OSI. O protocolo divide os dados como mostrado na Figura 2.5, em que a unidade de dados do protocolo (PDU) é composta pelo Código da função e de Dados, já a unidade de dados da aplicação (ADU) também inclui Endereços do cliente e o CRC. A organização *ModBus* especifica os comandos que devem ser implementados e o método de codificação em sua documentação oficial sobre a aplicação do protocolo *ModBus* (MODBUS, 2012).

A ADU possui tamanho de *bytes* limitado de acordo com o protocolo implementado na camada e, como mostrado anteriormente, a comunicação RS485 é padronizada em 256 *bytes*.

Já o PDU não inclui o *byte* de endereçamento e o *byte* da checagem de erro e por isso tem um tamanho padronizado de até 253 *bytes* para este tipo de comunicação serial. O *ModBus* define a implementação de três unidades de protocolo de dados: Requisição de dados, Resposta dos dados e Exceção de resposta dos dados requeridos.

A requisição de dados e a resposta de dados, segundo a organização *ModBus* (MODBUS, 2012), seguem o modelo exposto na Figura 2.5, mas contendo apenas as informações do PDU. Os códigos de função possuem espaços reservados e valores pré-definidos que serão mostrados mais à frente e possuem o tamanho máximo de um *byte* de dados alocado no PDU. Já a exceção de resposta possui uma tabela pré-definida de valores e quando as exceções devem ser evidenciadas, além de necessitarem de apenas dois *bytes*, um para o código da função e um para armazenar a exceção levantada.

O protocolo (MODBUS, 2012) distingue quatro modelos de dados que podem existir na sua implementação. A interação das entradas, saídas, endereçamento de *bits* e endereçamento de palavras com a aplicação é definida a partir da maneira mais natural de interpretação da máquina na qual a aplicação é implementada. A Tabela 2.1 descreve os modelos de dados existentes no protocolo, assim como o seu tipo de objeto, operações aceitas e descrição.

Tabela 2.1: Tabelas primárias das quatro características de dados distintas utilizadas pelo protocolo *ModBus*(MODBUS, 2012)

Tabela primária	Tipo do objeto	Tipo de operação
Entrada discreta	Booleano	Apenas leitura
Coils	Booleano	Leitura e escrita
Registrador de entrada	Palavra de 16 *bits*	Apenas leitura
Registrador de Holding	Palavra de 16 *bits*	Leitura e escrita

Para este projeto, foi utilizada a aplicação desenvolvida para a plataforma *Arduino* (M., 2019) que possui uma biblioteca onde as tabelas primárias descritas na Tabela 2.1 são implementadas como funções para duas classes distintas, uma para o cliente e outra para o servidor. A biblioteca é de licença aberta, se encontra na plataforma *GitHub* (M., 2019) e implementa funções para a aplicação do protocolo RTU utilizado pelo projeto e ainda oferece a alternativa de visualização serial similar à ferramenta *ModBus poll*, para a averiguação do estado da porta serial. Esta biblioteca foi escolhida por ser uma biblioteca que contempla os recursos necessários para a concretização do projeto e possui um forte embasamento no protocolo de comunicação escolhido para o projeto.

Capítulo 3

DESENVOLVIMENTO

Neste capítulo será documentado o desenvolvimento do projeto em seções que abrangem os passos tomados para o desenvolvimento do projeto, descrevendo cada etapa necessária para a finalização do mesmo.

3.1 Metodologia

Para o desenvolvimento do projeto é utilizado o modelo OSI, que segundo Pires (PIRES, 2006) é um modelo de sete camadas que define a comunicação de uma rede. Primeiro é desenvolvida a camada física, o circuito elétrico, onde é realizado o dimensionamento dos componentes da rede, assim como a simulação do sistema. Após o dimensionamento do circuito, o próximo passo é a implementação da camada de rede e a programação dos microcontroladores, a programação do servidor e do cliente e a construção da dinâmica de comunicação da rede entre o servidor e os clientes.

Feitas estas definições, é simulada a rede que conecta tais transceptores e feitos os testes de comunicação e checagem de funcionamento do protocolo RTU. Após aferido o funcionamento da conexão entre os microcontroladores, são testadas as funções e modos de comunicação entre o servidor e os clientes, assim como a resposta aos diferentes endereçamentos de mensagens.

Finalmente, é feito o orçamento do protótipo baseado nos componentes definidos ao longo do desenvolvimento. Este protótipo será feito a fim de validar o funcionamento do projeto durante a próxima etapa do trabalho. Na próxima etapa também será concluído o projeto para as definições feitas na seção 1.1 e desenhado o *layout* da placa de circuito impresso para que seja possível orçar o custo da implementação do projeto.

3.2 Escolhas de projeto

Nesta seção estão evidenciadas todas as escolhas e definições feitas ao longo do projeto. Inicialmente são especificadas a camada física e de dados, então é simulada a conexão dos controladores com a rede e escolhida como será feita a programação de tais microcontroladores. Por fim, é implementada a camada de Aplicação, com o desenvolvimento do painel de controle e também especificada a biblioteca que será utilizada para auxiliar a implementar o protocolo *ModBus* na sétima camada do modelo OSI.

3.2.1 Especificação da camada física e de dados

O desenvolvimento da camada física do projeto é feito a partir dos requerimentos de implementação da Organização ModBus (MODBUS, 2021) e das características do projeto. O sistema de múltiplas campainhas é feito com base em um painel central de acionamento localizado na entrada do condomínio e *buzzers* que estão em cada apartamento de cada um dos quatro blocos. As campainhas são acionadas pelo painel e então é tocada no apartamento identificado, do bloco requisitado, se utilizando de uma topologia de barramento, com uma ligação comum entre todos os blocos e uma topologia de estrela para a ligação entre o cliente e os *buzzers*.

Com isso, é possível definir que são necessários cinco microcontroladores e transceptores, um conjunto no painel (servidor) e um conjunto por bloco (clientes), para que cada cliente possa se ligar com todos apartamentos de seu bloco através de um barramento secundário. Esta distribuição se dá ao fato de apenas o painel central ter a necessidade de iniciar o contato com a rede e a necessidade de cada bloco possuir seu microprocessador para identificar e acionar a campainha de cada apartamento. Esta configuração foi escolhida, pois garante que haja apenas um par trançado conectando painel a cada um dos blocos, diferentemente de alternativas como a central de portaria Comunic 48 (INTELBRAS, 2020a), que se utiliza da topologia em estrela. A Figura 3.1 mostra a esquemática da topologia que será utilizada para a rede.

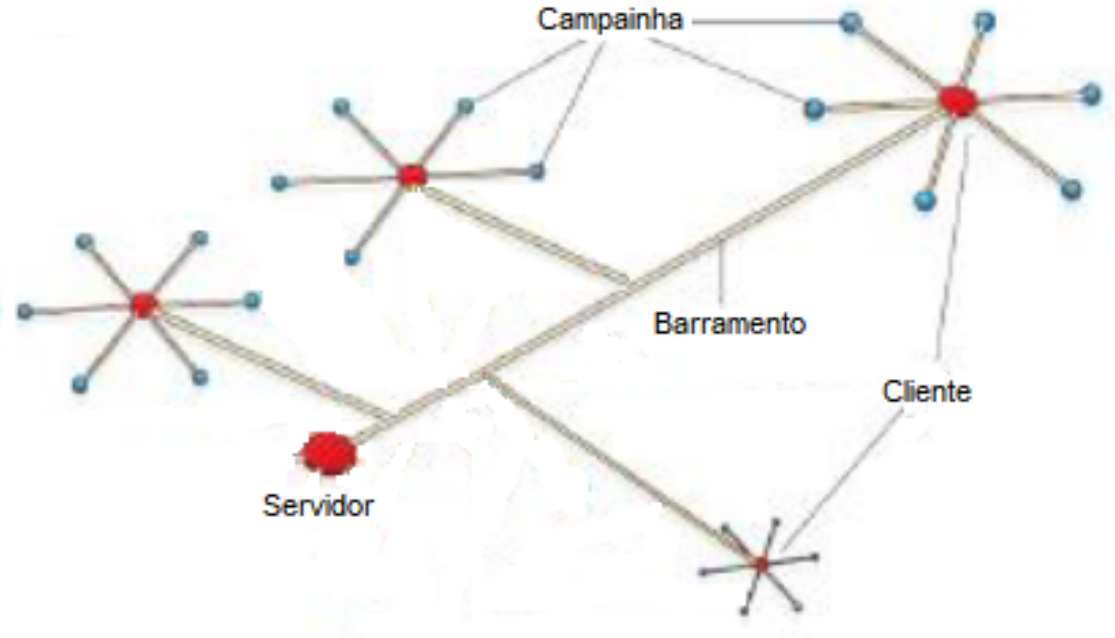

Figura 3.1: Representação da topologia proposta para a rede.

Como será necessário o uso de apenas quatro clientes, não é necessária a implementação de um replicador de sinal, já que o protocolo *MobBus* (MODBUS, 2012) só o exige caso haja mais de 32 dispositivos na mesma trilha de dados. Entre os transceptores de baixa potência da empresa *Maxim Integrated* (MAXIM, 2014), foi escolhido o MAX487, por possuir uma melhor resposta a cabeamentos com problemas de terminação e menor consumo corrente, mas maiores atrasos de propagação. As características de propagação do transceptor escolhido não é a melhor entre os dispositivos da mesma família, mas a aplicação residencial de campainhas se beneficia mais de um menor consumo e maior confiabilidade contra problemas na terminação do que de uma propagação de sinal veloz.

3.2.2 Conexão dos microcontroladores com a rede

Inicialmente, é feita uma simulação do circuito, no *software Proteus 8*, com apenas dois microcontroladores, um representando o painel central e um representando o bloco *A* do condomínio. Esta escolha de simulação é feita a fim de simplificar a rede inicial e incluir e validar os protocolos de comunicação entre o servidor e um cliente antes de aumentar a complexidade da rede, reduzindo a quantidade de clientes para facilitar a resolução de problemas de implementação e entender melhor as limitações e particularidades do *software*. Esta simplificação também será implementada para o teste do protótipo, para serem feitos outros testes e validações no sistema real. Na Figura 3.2 é representada a ligação inicial feita

para o teste de comunicação na camada de rede, onde são incluídas as devidas terminações de rede e resistores de balanceamento da impedância do par trançado.

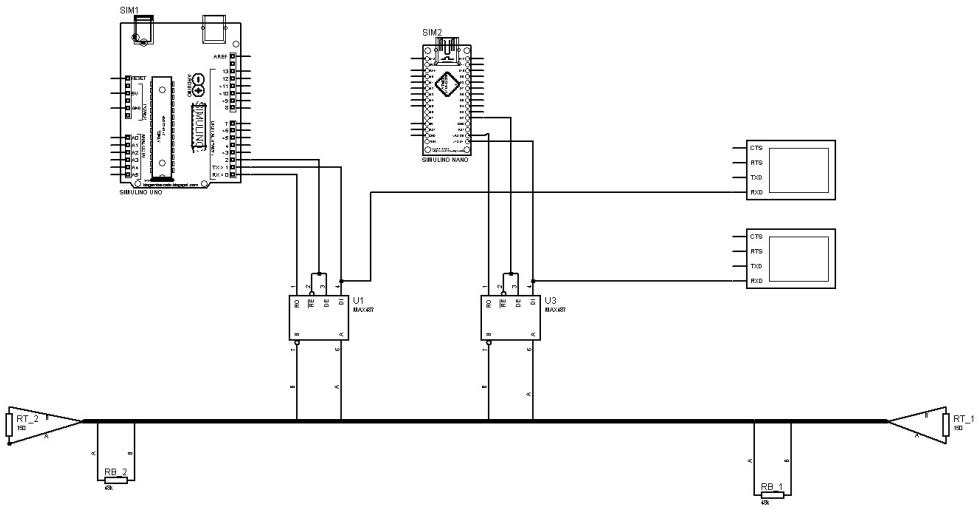

Figura 3.2: Circuito simplificado da conexão dos microcontroladores à da camada de rede utilizando do protocolo RS-485.

No circuito mostrado na Figura 3.2, há dois microprocessadores ligados ao barramento de rede, cada um utilizando de um transceptor *MAX487* para se ligarem à rede, cujas informações gerais podem ser vistas no Apêndice A. Os terminais *RX* e *TX* dos *ATMega328p*, implementados na placa da plataforma *Arduino*, são ligados aos terminais do receptor (RO) e do transmissor (DI) do transceptor, respectivamente (MAXIM, 2014). O terminal *2* controla se o transceptor está recebendo ou transmitindo informações na rede através da interligação das portas $\bar{RE}$ e DE, sendo o sinal baixo para o modo de leitura ($\bar{RE}$) e o sinal alto para o modo de escrita (DE) (MAXIM, 2014).

Segundo a ficha de dados do componente (MAXIM, 2014), para o balanceamento do barramento — o protocolo *ModBus* exige a utilização de uma terminação para comunicação bidirecional (Figura 2.6) — é necessária uma impedância de $R_p = 48K\Omega$ entre os cabos do barramento. Além disso, também é inserida a terminação de linha não polarizada de $R_T = 150\Omega$ em cada extremo do barramento, como sugerido pelo protocolo (MODBUS, 2012). Nesta configuração podem ser adicionados ao barramento até 128 transceptores *MAX487/ MAX1487* ou uma combinação de até 32 diferentes transceptores da família RS-485, como *MAX481/ MAX483/ MAX485/ MAX487/ MAX1487* da marca *Maxim Integrated* e utilizada uma impedância entre os cabos de $R_p = 12K\Omega$ para estes.

A comunicação entre os clientes e o servidor ocorre na segunda camada do modelo OSI, conhecida como camada de dados. Esta camada também é descrita no manual de implementação do *ModBus* (MODBUS, 2012) e nela onde os comandos são codificados, decodificados e aferidos. O protocolo *ModBus* exige a utilização de algumas estruturas comunicação e de averiguação de dados nesta camada, como o protocolo RTU e a checagem cíclica de redundância. O uso da plataforma *Arduino* foi escolhido por possuir uma biblioteca que já possui estas implementações com baixo custo de processamento e também por possuir módulos de microcontroladores de baixo custo e uma interface de programação gratuita. Para a implementação comercial é mais recomendada a utilização do próprio microcontrolador *ATMEGA328*, já que a plataforma *Arduino* é educacional e possui um foco em prototipagem e não em desempenho.

3.2.3 Programação dos microcontroladores

A programação dos microcontroladores se dá na camada de aplicação do sistema, já que o código desenvolvido será traduzido pelas bibliotecas instaladas que são feitas com base na documentação da camada de aplicação(MODBUS, 2012). Como explicado anteriormente, há certas aplicações padrões do protocolo *ModBus*, (Tabela 2.1), mas há aplicações específicas que podem ser inseridas pelo desenvolvedor. Para este projeto, todas as funções necessárias são contempladas pela biblioteca escolhida.As nuances do projeto são programadas em *C++* no próprio Ambiente de Desenvolvimento Integrado (IDE) da plataforma *Arduino*, para que os códigos possam ser testados tanto no protótipo, quanto no ambiente de simulação.

O *software* escolhido para a simulação, o *Proteus 8.7*, traz a possibilidade da criação de um *firmware* em *C++* com a integração com a plataforma *Arduino*, que, em contrapartida, exige uma grande demanda computacional quando implementada. Outro aspecto importante a ser considerado é a facilidade da instalação das bibliotecas utilizando a IDE da plataforma *Arduino*, pela possibilidade de poderem ser instaladas pelo próprio ambiente. O *Proteus 8.7* não disponibiliza a opção para inclusão da biblioteca *ArduinoModBus* (M., 2014) e faria necessária a implantação da camada de aplicação por completo.

Na biblioteca é considerado que os terminais de habilitação de leitura e escrita do circuito integrado *MAX485* estão interligados, já que a organização *ModBus* (MODBUS, 2012) define que apenas um dispositivo pode estar transmitindo a cada momento. O terminal de comando é definido pelo programador e, como no microcontrolador *ATMEGA328p* os terminais digitais *0* e *1* são utilizados para a transmissão serial, estes não podem ser escolhidos para o controle de transmissão. Esta escolha só precisa ser feita caso esteja sendo utilizado o padrão *RS-485*,

já que não é necessário este controle no padrão *RS-232*. Porém, como dito anteriormente, foi feita a escolha do terminal digital *2* tanto para o servidor, quanto para os clientes.

A plataforma *Arduino* disponibiliza as mensagens de compilação, que informa a pasta temporária onde o código hexadecimal, que é aplicado no microcontrolador é armazenada. Com isso, é possível incluir este caminho para o código hexadecimal no microcontrolador simulado no *Proteus*, possibilitando a utilização das bibliotecas que são apenas compatíveis com a IDE do *Arduino*, já que a compilação é feita na mesma. Isto possibilita testes no simulador compilados pelo mesmo programa que os códigos que serão carregados no microcontrolador do protótipo, ocorrendo uma constância entre o projeto simulado e prototipado.

3.2.4 Desenvolvimento do painel de interface

Como o condomínio possui quatro blocos, bloco A, B, C e D, e são oito apartamentos por bloco, foi feita esta escolha do uso de um teclado matricial, já que o mesmo utiliza apenas oito terminais digitais do microcontrolador. Esta abordagem matricial foi pensada para ser lido caractere a caractere, pois uma abordagem para 32 botões precisaria de um teclado matricial 6×6, que se utilizaria de 12 saídas do microcontrolador. Botões individuais, caso não fosse implementada esta abordagem matricial, necessitariam de outros componentes como de-multiplexadores, conversor serial ou ou um decodificador em alta velocidade, como o *74LS139* para a conversão de informações entre os 32 botões e as 11 entradas digitais restantes do microcontrolador.

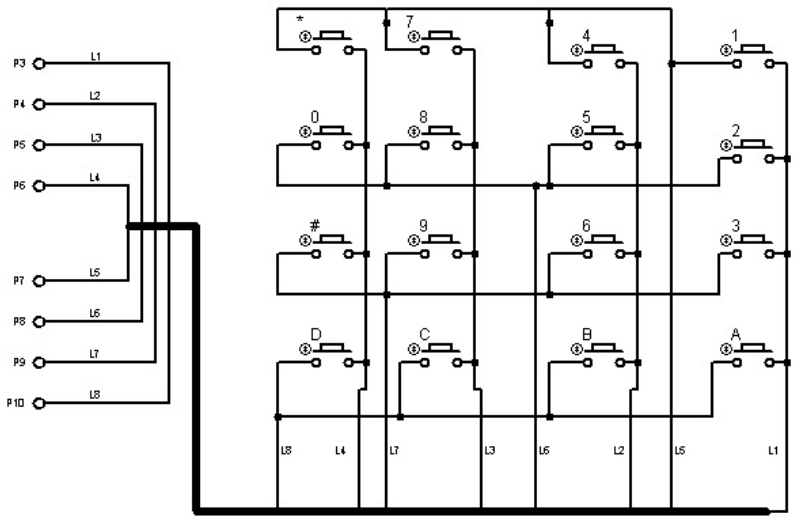

Figura 3.3: Ligação interna do teclado matricial com ligação de interrupção

Para o painel de botões de comando será utilizado um teclado de membrana matricial 4x4. Este teclado possui baixo custo, é de simples implementação, durável e à prova d'água (PROTOSUPPLIES, 2021). Este teclado se utiliza de uma interligação entre linhas e colunas, com os botões servindo como a interligação entre cada linha e cada coluna, conforme mostrado na Figura 3.3. A identificação da tecla pressionado, as definições da função dos botões e outras funcionalidades para este painel são feitos através da programação. Foi definido que a chamada para cada apartamento será feita por *Bloco + Nº do apartamento + #*, onde *Bloco* é a letra do bloco em que se encontra o apartamento, *Nº do apartamento* é o valor de três dígitos do apartamento e o símbolo da cerquilha (#) é um caractere de fim de mensagem, para o microcontrolador reconhecer que todos os dígitos foram pressionador e a campainha pode ser tocada.

Pensando na experiência do usuário durante a utilização do painel de acionamento, a tecla asterisco (*) é utilizada para limpar da memória o que acabou de ser digitado. Isto é, caso o usuário tenha errado algum dos números do apartamento ou o bloco, apertando asterisco é possível redigitar a mensagem do início. No caso da cerquilha ser pressionada antes do bloco e do apartamento serem pressionados na íntegra, ocorre o mesmo que no pressionamento do asterisco.

Feitas estas configurações, é programado o teclado matricial em *C++* com o uso da biblioteca *Keypad.h* (ANDREWS, 2015). Esta biblioteca possui funções para facilitar a implementação do teclado matricial na plataforma *Arduino*. Suas funções são a atribuição de uma variável para cada tecla, interrupção relacionada ao pressionamento de tecla, laço que aguarda tecla ser pressionada, diferenciação entre tecla pressionada, não pressionada e segurada e verificação do estado de cada tecla.

Em suma, o teclado é desenvolvido para aceitar apenas valores que sigam as regras pré-estabelecidas. O teclado também é programado de maneira a somente salvar na variável global a mensagem que será enviada para o cliente. Além disso, o envio somente é habilitado se a mensagem for escrita por completo, restringindo os erros de mensagem incompletas apenas para erros derivados da camada de dados ou física.

3.2.5 Especificação da camada de Aplicação

Para a camada de Aplicação, definida pelo protocolo (MODBUS, 2012), é utilizada uma biblioteca existente (M., 2019) que possui as funcionalidades necessárias para o projeto. Então é necessário conhecer cada uma das funções existentes e erros que podem ser retornados pela mesma. As tabelas 3.1 e 3.2 apresentam, respectivamente, as funções implementadas e os

erros e exceções que a biblioteca pode retornar. As funções, erros e exceções implementados são baseados na Tabela 2.1 e na documentação do protocolo *ModBus* (MODBUS, 2012)

Tabela 3.1: Tabela de funções implementadas pela biblioteca (M., 2019)

Nome da função	Número da função	Descrição
MB_FC_NONE	0	Operador nulo
MB_FC_READ_COILS	1	Lê Coils ou saídas digitais
MB_FC_READ_DISCRETE_INPUT	2	Lê entradas digitais
MB_FC_READ_REGISTERS	3	Lê saídas analógicas
MB_FC_READ_INPUT_REGISTER	4	Lê entradas analógicas
MB_FC_WRITE_COIL	5	Escreve um único *coil* ou saída
MB_FC_WRITE_REGISTER	6	Escreve em um único registrador
MB_FC_WRITE_MULTIPLE_COILS	15	Escreve em múltiplos *coils* ou saídas
MB_FC_WRITE_MULTIPLE_REGISTERS	16	Escreve em múltiplos registradores

Na Tabela 3.1 há todas as funções básicas de cliente e de servidor geralmente utilizadas em operações com o protocolo, mas como a aplicação exige o recebimento de uma informação no cliente enviada pelo servidor, é implementada a função de número *6* para o servidor e lida esta informação pelo cliente. Esta escolha é feita com o intuito de direcionar a complexidade computacional para o cliente, reduzindo assim a quantidade de informações enviadas pela rede e chances de ocorrerem falhas na transmissão. Assim, o cliente pode ler a informação digital enviada pelo servidor e identificar e acionar o comando correto com base na mensagem.

Tabela 3.2: Tabela de erros e exceções levantados pela biblioteca (M., 2019)

Nome do erro/exceção	Número	Descrição
ERR_NOT_MASTER	-1	Erro levantado por utilizar comando de servidor em um cliente
ERR_POLLING	-2	Erro levantado pela requisição enquanto dados ainda então sendo aferidos
ERR_BUFF_OVERFLOW	-3	Erro levantado por extrapolar o tamanho máximo do *buffer*
ERR_BAD_CRC	-4	Erro levantado por haver um CRC inválido
ERR_EXCEPTION	-5	Erro levantado por ter ocorrido alguma exceção
NO_REPLY	255	Exceção: Sem resposta
EXC_FUNC_CODE	1	Exceção: Código de função
EXC_ADDR_RANGE	2	Exceção: Escopo de endereçamento
EXC_REGS_QUANT	3	Exceção: Quantidade de registradores
EXC_EXECUTE	4	Exceção: Execução

Na Tabela 3.2 há todos os erros e exceções que podem ser sinalizados pela biblioteca utilizada. O registrador para estes erros possui um tamanho de *3 bytes*, ou seja, mais exceções e erros podem ser implementados caso haja necessidade. Neste projeto, o erro que vale ser ressaltado é o *ERR_BAD_CRC*, já que o erro de checagem sinaliza um problema de comunicação que impossibilita a transmissão que deve ser feita.

Nesta biblioteca, o tamanho do *buffer* definido é de 64 *bytes* e a organização *ModBus* padroniza um tamanho de *256 bytes* e esta diferença pode acarretar no erro de número

-3 caso não seja considerada. Como a aplicação feita utilizará um baixo número de *bytes* transportados e se utiliza de microcontroladores com uma memória interna baixa, foi decidido manter este tamanho de 25% do total, já que isto não afetará o produto final. Caso sejam necessárias serem feitas operações que utilizem a capacidade total definida pelo protocolo, é preciso redefinir, dentro da biblioteca, a variável *MAX_BUFFER* para o valor de 256 *bytes* e, caso isso não seja feito, uma exceção será levantada.

Como se faz necessária a diferenciação entre cada um dos clientes na etapa final, foi feita a definição do endereço de cada um dos blocos fisicamente. Como há quatro blocos, dois terminais serão utilizados para diferenciar entre cada um deles, utilizando a contagem em binário de 0 a 3 para representar os blocos de A a D. Os terminais 5 e 6 dos microcontroladores ditarão o seu endereço na camada de aplicação, com o terminal 5 representando o algarismo menos significativo e o terminal 6 o mais significativo.

3.3 Teste do protótipo

Para o desenvolvimento do protótipo, foi utilizado o módulo *MAX485*, diferente do utilizado na simulação e da escolha do projeto, pois este já possui módulos no mercado de mais fácil prototipagem e aplicação. Como mostrado na Tabela 3.3, há algumas diferenças entre os transceptores, como a velocidade de transmissão, a limitação do *slew-rate* e a corrente de repouso, além do número máximo de transceptores na linha.

Tabela 3.3: Características dos transceptores MAXIM Integrated utilizados para comunicação por padrão RS-485 (Fonte: Datasheet MAXIM Integrated (MAXIM, 2014))

Nº do componente	**Taxa de transmissão (Mbps)**	***Slew-rate limitado***	**Desligamento à baixa energia**	**Corrente de repouso (μA)**	**Nº máximo de transceptores na linha**
MAX481	2,5	Não	Sim	300	32
MAX483	0,25	Sim	Sim	120	32
MAX485	2,5	Não	Não	300	32
MAX487	0,25	Sim	Sim	120	128
MAX488	0,25	Sim	Não	120	32
MAX489	0,25	Sim	Não	120	32
MAX490	2,5	Não	Não	300	32
MAX491	2,5	Não	Não	300	32
MAX1487	2,5	Não	Não	230	128

Estas diferenças mostradas na Tabela 3.3 não afetam o funcionamento do protótipo em relação à simulação, já que a frequência dos dados definida anteriormente ($19,2Kbps$) é muito inferior ao limite máximo do transmissor *MAX487*. A limitação de *slew-rate* é importante

para o produto final, já que problemas nos resistores de terminação ou em outros pontos da rede que podem causar ruídos são mitigados pelo *Max487*, mas como o protótipo é testado em uma rede propriamente terminada e monitorada, não há necessidade desta função. A função de desligamento à baixa energia só é iniciada caso as portas de escrita e leitura do circuito integrado estejam em um nível de sinal baixo, mas isto não é uma função necessária e nem implementada no projeto.

Como o protótipo tem por finalidade a validação do funcionamento do sistema a corrente de repouso não influencia nos testes de validação planejados, já que o intuito é validar o sistema com a comunicação ativa. Como o projeto prevê a inclusão de apenas cinco transceptores na linha e o protótipo possui apenas dois, não há a limitações advindas do número máximo permitido de transceptores. Feitas estas considerações, é então implementado o protótipo com o uso do transceptor *MAX485*.

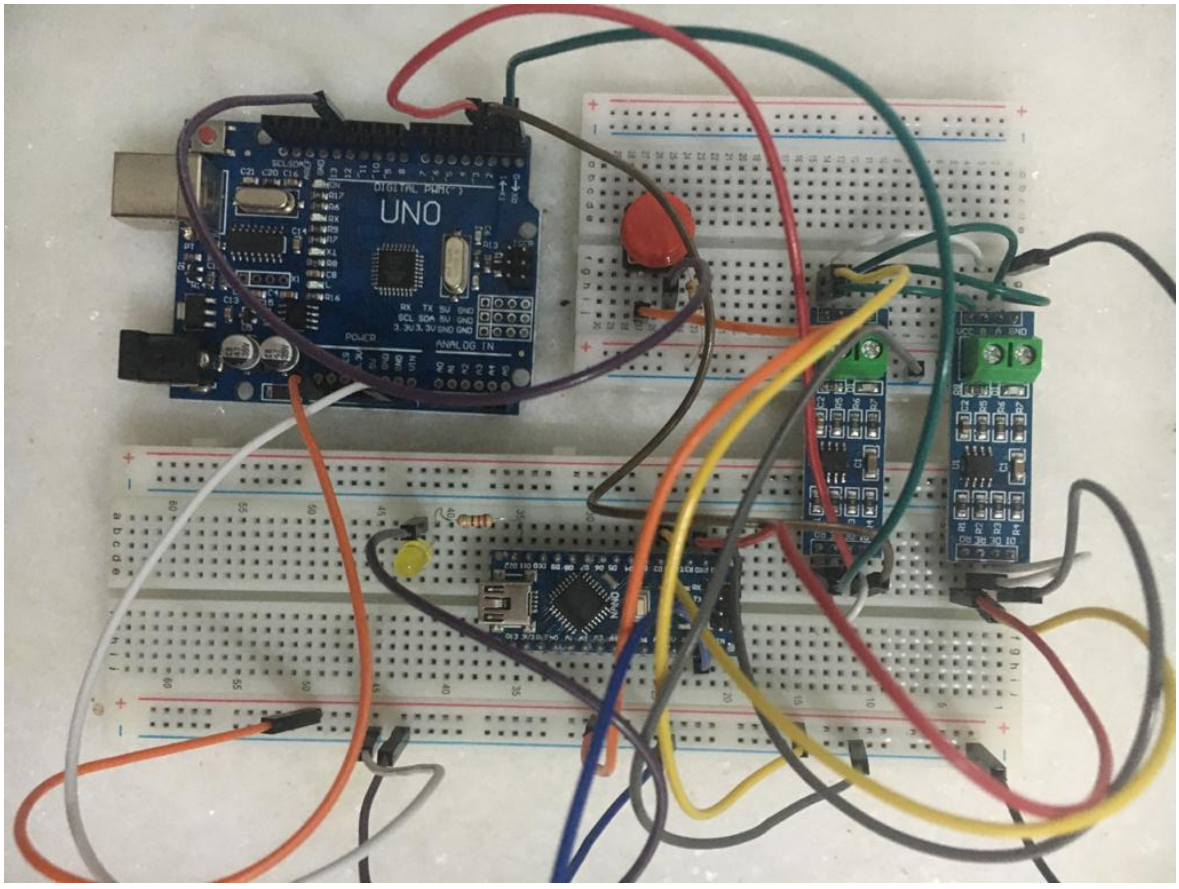

Figura 3.4: Protótipo montado e desenergizado

A Figura 3.4 mostra o protótipo montado com base no circuito simplificado que foi apresentado na Figura 3.2. Como dito anteriormente, é utilizado um módulo *MAX485* e nele já

são inclusos os resistores de terminação de linha não polarizada (R_T) mencionada anteriormente. Além disso, os *LEDs* dos *Arduino* são utilizados para aferir a comunicação serial e a tecla vermelho e o *LED* amarelo são utilizados para a validação da comunicação que será explicada em maiores detalhes no próximo capítulo.

3.3.1 Materiais utilizados

Nesta seção estão descritos os materiais necessários para o desenvolvimento do protótipo, tendo em vista que neste primeiro momento, o projeto será apenas simulado. Na segunda etapa do trabalho será desenvolvido um protótipo para teste da comunicação física do sistema e orçados todos os componentes necessários descritos neste capítulo. Como no desenvolvimento da segunda etapa do projeto, também será feito um protótipo para serem testadas algumas funções da simulação, os componentes já foram comprados e suas informações, quantidades e valores estão descritas na Tabela 3.4

Tabela 3.4: Tabela de materiais necessários para implementação do protótipo

Componente	Quantidade	Custo total (R$)
Arduino Uno	1	45,00
Arduino Nano	1	38,90
Protoboard 830 pontos	1	14,90
Módulo RS485	3	23,00
Kit básico Arduino	1	52,22
Total		174,02

O custo total mostrado na tabela é referente a quantidade comprada, tendo em vista que certos componentes eletrônicos, como o módulo de RS485, possuem um desconto incremental em relação ao número de unidades compradas. Ambos preços dos *Arduinos* referenciados na Tabela 3.4 incluem o microcontrolador e o cabo de dados e alimentação para os mesmos. O Kit básico Arduino inclui 20 cabos *jumpers*, 40 resistores —Sendo dez resistores de 220Ω, dez resistores de $1k\Omega$, dez resistores de $10k\Omega$ e dez resistores de $100k\Omega$—, dez botoeiras de acionamento, 15 LEDs de variadas cores, uma *protoboard* de 400 pontos e um conector para baterias de $9V$. Todos os itens citados acima, com exceção do conector para baterias, serão utilizados na montagem do protótipo na segunda etapa deste projeto.

Além disso, são feitos dois orçamentos para o protótipo em duas lojas *online* distintas, *Baú da Eletrônica* e *Mercado Livre*. Estes orçamentos se encontram no Apêndice B, sendo que seus valores no dia 17/08/2021 foram de $R\$240,76$ e $R\$282,89$, respectivamente. Observa-se que o valor é mais elevado por serem compradas as peças em separado, não aproveitando

promoções, conjuntos pré-montados e considerações de produtos em relação ao frete.

3.4 Definições de instalação

Nesta seção são descritas as especificações de cada ligação e consideração que precisa ser feita para a instalação no condomínio. É descrita como são ligadas as campainhas aos clientes, assim como os limites e o porquê das escolhas feitas. Então, são colocadas algumas considerações levantadas para o painel de controle e as soluções escolhidas. Por último, são feitas as escolhas e definições necessárias para a placa de circuito impresso, como a definição dos componentes que serão acoplados, como será a alimentação do circuito e o acoplamento da mesma na prática.

3.4.1 Ligação das campainhas

Como dito anteriormente, para cada bloco há um cliente no qual estão ligadas as campainhas com utilização da topologia estrela. A campainha é um *buzzer* ativo de $4 - 8V$ e Frequência de ressonância de $2300 \pm 300\,Hz$ da marca *Pro-Signal* (PRO-SIGNAL, 2016). Foi escolhido um *Buzzer* ativo, pois como não é necessária a modulação da frequência da campainha, é mais vantajoso utilizar um dispositivo de mais fácil implementação e que utiliza menos memória do microcontrolador.

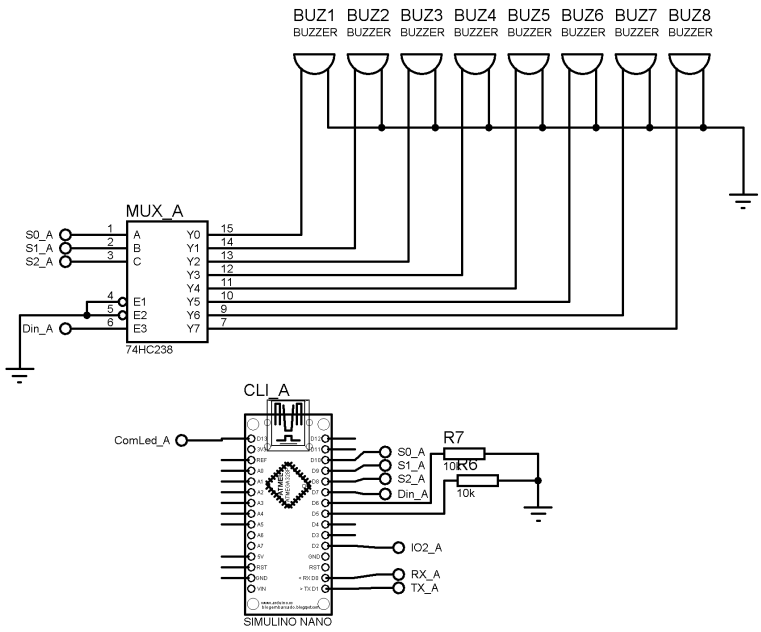

Figura 3.5: Ligação das campainhas para o bloco A

A Figura 3.5 revela a ligação das campainhas com o cliente do bloco A e como pode ser visto, é utilizado um de-multiplexador *74HC238* (PHILLIPS, 1990) para o acionamento das campainhas. Esta escolha foi feita para reduzir o número de portas utilizadas pelo cliente, permitindo a implementação de campainhas para pelo menos outros 16 apartamentos[1] por bloco. O uso do de-multiplexador também gera uma maior liberdade para utilização de outros periféricos em trabalhos futuros, como a implementação de botoeiras para a abertura dos portões ou a instalação de cartões SD para a gravação de erros e exceções do sistema.

[1]Há 10 portas não utilizadas no microcontrolador e se usados outros dois decodificadores *74HC238* apenas oito destas portas seriam utilizadas. Outros decodificadores ou métodos poderiam ser aplicados para expandir a quantidade de apartamentos contemplados para além de 24.

Tabela 3.5: Tabela verdade do decodificador 74HC238

Entradas						Saídas							
$\bar{E}_1$	$\bar{E}_2$	E_3	C	B	A	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
1	X	X	X	X	X	0	0	0	0	0	0	0	0
X	1	X	X	X	X	0	0	0	0	0	0	0	0
X	X	0	X	X	X	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	1	0	1	0	0	0	0	0	0
0	0	1	0	1	0	0	0	1	0	0	0	0	0
0	0	1	0	1	1	0	0	0	1	0	0	0	0
0	0	1	1	0	0	0	0	0	0	1	0	0	0
0	0	1	1	0	1	0	0	0	0	0	1	0	0
0	0	1	1	1	0	0	0	0	0	0	0	1	0
0	0	1	1	1	1	0	0	0	0	0	0	0	1

O acionamento das campainhas foi embasado na Tabela 3.5, já que o de-multiplexador está conectado com os *Buzzers*. Quando está sendo utilizado apenas um decodificador, não é necessário o uso das portas $\bar{E}_1$ e $\bar{E}_1$, já que a porta E_3 é suficiente para o controle das saídas. Caso sejam utilizados três de-multiplexadores, cada uma das portas E pode fazer o acionamento de cada um dos dispositivos, já o sinal nestas portas é que define se as saídas podem ou não ser acionadas.

Como mostrado na Figura 3.5, as entradas digitais 10, 9 e 8 do microcontrolador controlam as entradas A, B e C, respectivamente, e a entrada digital 7 controla a entrada E_3 do decodificador. Então, foram decididos os valores a serem digitados para cada um dos *buzzers*, enumerados de *BUZ1* a *BUZ8*. Seguindo a ordem crescente, os valores escolhidos para cada apartamento são escolhidos de acordo com a tabela 3.6.

Tabela 3.6: Relação entre saída do de-multiplexador e número do apartamento

Saída decodificador	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
Num. Apartamento	101	102	201	202	301	302	401	402

3.4.2 Interface Homem-Máquina

Segundo Alani, a interface Homem-Máquina (IHM) é parte da camada de aplicação e recebe comandos de um operador humano (ALANI, 2014). No projeto, a interface se dá pelo painel de controle onde é feito o acionamento das campainhas. A interface necessita de uma proteção que facilita o acesso aos botões, porém protege os outros componentes.

Para que haja uma comunicação com menos chances de interferência, menos exposição ao clima e menor risco à integridade dos componentes é necessário uma caixa de proteção para o painel de controle. A utilização de placas de circuito impresso e soldas para assegurar o contato elétrico entre os dispositivos também diminui o risco de erros e melhora a capacidade de replicação do circuito.

3.4.3 Escolhas de componentes e placa de circuito impresso

Para este sistema, serão necessárias duas placas de circuito impresso distintas, uma para o servidor e uma para os clientes. Tendo em mente este aspecto, é necessário considerar as diferentes ligações nos terminais 5 e 6 do microcontrolador, já que os mesmos ditam o endereço do cliente, que é único. Para a implementação do projeto, é escolhido o *Arduino Nano* por causa de seu tamanho reduzido, mas que necessita de uma alimentação através do terminal 27[1] ou 30[2], já que este modelo possui apenas a entrada de alimentação através da porta micro-USB.

Primeiro é feito o cálculo das correntes atuantes no sistema para se encontrar a fonte necessária para suprir a potência demandada pelo sistema. Para os multiplexadores da família 74HC (PHILLIPS, 1988) o máximo demandado é de $90\,mA$ de corrente enquanto o mesmo faz a mudança de estado. Já o *buzzer* ativo (PRO-SIGNAL, 2016) possui uma frequência fixa e, por isso, um consumo constante de $30\,mA$, quando acionado. Já o microcontrolador (ARDUINO, 2021) consome em média uma corrente de $19\,mA$ para a sua operação e até $40\,mA$ para cada porta de saída acionada e outros $33\,mA$ para as portas Tx e Rx. Como o cliente usa 5 terminais digitais, seu consumo máximo esperado é de $252\,mA$ e o consumo máximo total esperado do circuito é de $372\,mA$.

Considerando a demanda máxima do cliente e a tensão de trabalho do microcontrolador, foi escolhida uma fonte de pequeno porte que supra as necessidades do projeto. A fonte HLKPM01 da marca *Hi-Link* (HI-LINK, 2018) têm uma entrada de $100 - 270V$ AC e saída

[1] O terminal 27 aceita 5V DC regulado e tensões continuas sem regulação podem danificar o microcontrolador

[2] O terminal 30 aceita tensões de 7V à 12V contínua e não há necessidade de regulação, já que o microcontrolador faz a regulação desta tensão internamente

de $5 \pm 0,1V$ DC e potência de $3\,W$. Com os $600\,mA$ de corrente fornecidos pela fonte é possível alimentar o circuito até nos períodos de maior demanda e as dimensões da fonte são ideias para colocá-la em uma placa de circuito impresso.

As mesmas considerações são feitas para o servidor, que também se utilizará de um *Arduino Nano* como microcontrolador, tendo ligado a ele somente o teclado de membrana 4×4 e o transceptor Maxim487. O teclado de membrana (PROTOSUPPLIES, 2021) utiliza oito portas do microcontrolador, sendo quatro como terminais de entrada e quatro como terminais de saída e, portanto consumindo até $160\,mA$. Então, o total esperado de corrente máxima consumida pelo circuito é de $212\,mA$, sendo possível a utilização da mesma fonte neste circuito e deixando margem para a implementação de outros periféricos em projetos futuros.

A norma IPC-2221 regulamenta a menor largura de trilha segura de acordo com o comprimento, corrente, espessura do cobre e variação de temperatura da placa. Para trilhas externas, considerando o peso do cobre como $0,5oz/ft^2$, uma variação de temperatura de $10°C$, uma temperatura ambiente de $25°C$ e um comprimento de condutor máximo de $150\,mm$, a espessura da trilha dependerá apenas da corrente máxima e da menor espessura de trilha fornecida pelo fabricante. Para a alimentação do circuito, a corrente máxima suportada é de $600\,mA$, para o CI 74HC238 e para o *Maxim487* a corrente máxima é de $90\,mA$ e o *Arduino* suporta o máximo de $40\,mA$, em seus terminais digitais. Portanto, a alimentação necessitaria de uma largura mínima de trilha de $0,3\,mm$, os CIs de uma trilha de $0,02\,mm$ e as portas digitais do *Arduino Nano* $0,01\,mm$. Entretanto, segundo Williams (WILLIAMS, 2005), a menor espessura preferível para uma trilha é de $0,15\,mm$, por causa do processo de fabricação utilizado para corroer as trilhas.

A literatura e os simuladores geralmente utilizam o milésimo de polegada como a unidade para a espessura da trilha antecedido pela letra T. Então, a trilha de alimentação, por exemplo, necessitaria uma espessura $T12$ e os outros componentes o mínimo recomendado de $T6$, mas este tamanho mínimo pode variar dependendo do fabricante. As trilhas escolhidas para o projeto são todas de espessura T25 ($0,635\,mm$) dada a possibilidade de fazer a corrosão de placas de circuito impresso manualmente e não apenas industrialmente. Também é projetado um painel de circuitos para a fabricação industrial com as espessuras de trilha calculadas.

Segundo Williams (WILLIAMS, 2005), é preferível evitar ângulos retos e agudos quando se traça a trilha para evitar riscos na corrosão da trilha. Williams também diz para evitar trilhas a menos de $0,5\,mm$ da borda da placa e um mínimo de $0,15\,mm$ de espaçamento

entre as trilhas. Para este projeto será feito um circuito impresso simples[1], pois há poucos componentes que são possíveis de se ligar, sem haver cruzamento de trilhas.

Como será utilizado o circuito impresso simples, é possível a implementação tanto com Dispositivos de Montagem Superficial (SMD), quanto por Pino Pelo Furo (PTH). De acordo com Williams (WILLIAMS, 2005), a implementação de SMD reduz o tamanho da placa, melhora a performance elétrica e facilita a automação de montagem da placa. Entretanto, esta montagem dificulta na manutenção do circuito, possui uma dificuldade altíssima de fabricação, é muito mais caro de se produzir e complexo de se testar, modificar e consertar em comparação com o PTH. Feitas estas considerações, é escolhido o modelo PTH por sua facilidade de troca e manutenção de componentes, assim como a facilidade e custo de montagem.

[1]Um circuito impresso simples possui trilhas em apenas um dos lados da placa.

Capítulo 4

RESULTADOS E DISCUSSÕES

Neste capítulo são expostos os resultados obtidos ao longo do desenvolvimento do trabalho. Primeiro são mostrados os resultados obtidos na simulação do teste de transporte de dados com o uso da camada de aplicação especificada no capítulo anterior. Então é exposta a integração do painel de botões desenvolvido e o processo de comunicação. Por último, é mostrado um possível *layout* da placa de circuito impresso para a instalação deste circuito em um condomínio, assim como o orçamento de tal projeto.

4.1 Resultados

4.1.1 Teste do transporte de dados

Para o teste de transporte de dados são feitos dois códigos distintos, um para o Servidor e outro para o cliente e ambos são testados tanto na simulação, quanto no protótipo. É definido no servidor o bloco com o qual se deve iniciar o contato e então um sinal é enviado, sendo em baixa, caso a tecla não esteja pressionada, e em alta, caso a tecla esteja pressionada (MORAES, 2021a). Já o cliente, recebe o pacote de dados do servidor, lê se o sinal é alto ou baixo e reflete esta leitura no *LED* azul (D2) (MORAES, 2021b).

Simulação do transporte de dados

Primeiro é feita uma simulação de um teste simples para aferir a comunicação entre ambos microcontroladores. Há três camadas do modelo OSI usadas para a implementação do protocolo *ModBus* e este teste engloba todas elas. A comunicação só funcionará se a camada física estiver ligada corretamente, a mensagem só chegará se a camada de rede estiver correta e a mensagem só será interpretada caso a camada de aplicação esteja programada da maneira certa.

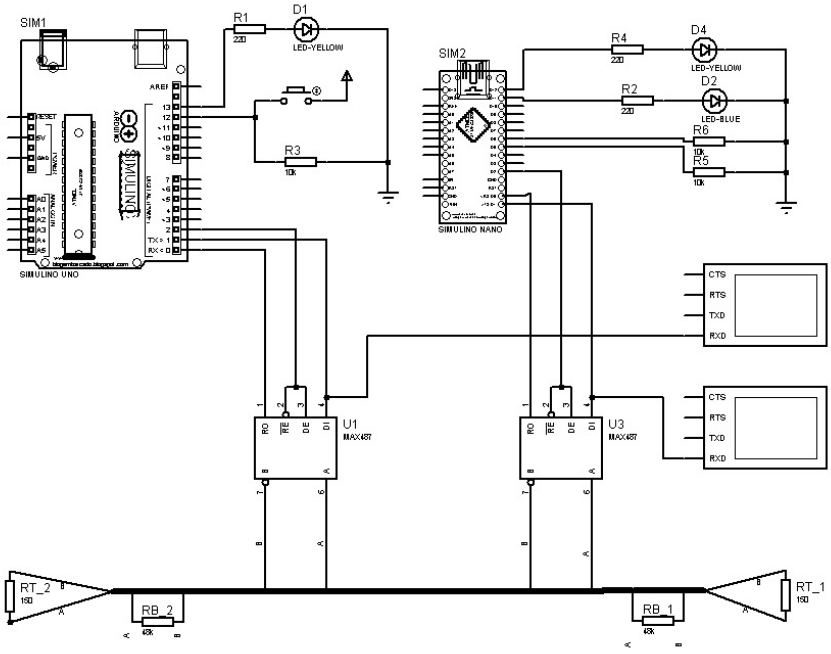

Figura 4.1: Circuito simulado que será reproduzido no protótipo

Como pode se observar na Figura 4.1, há um *LED* amarelo conectado a cada um dos microcontroladores. Esses *LEDs* são ligados quando é feita a ligação serial com a rede e serão necessários para os testes no protótipo, já que no mesmo não haverá o monitor serial. Os resistores de *pull-down* R5 e R6 são utilizados para manter os pinos 5 e 6 do cliente em baixo nível de sinal, caracterizando-o como o bloco *A* (MORAES, 2021a);(MORAES, 2021b).

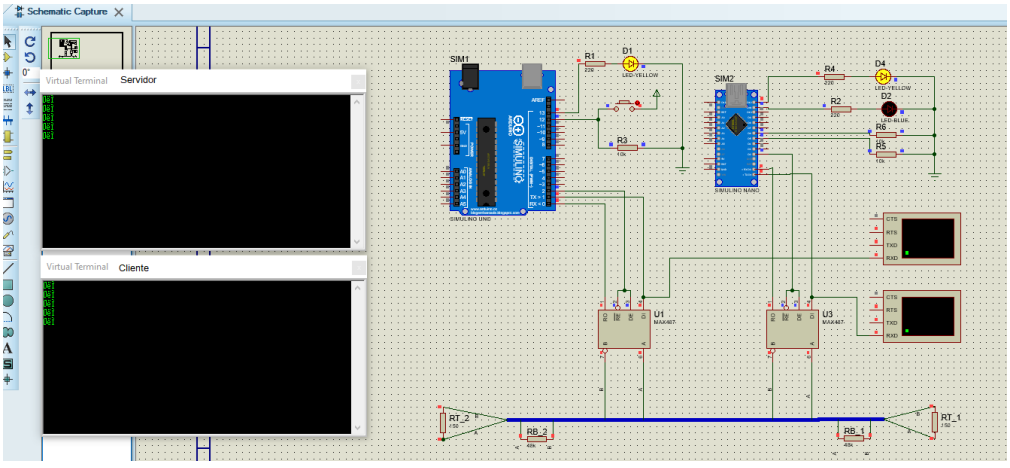

Figura 4.2: Resposta simulada sem pressionar a tecla para o bloco A

Na Figura 4.2 é possível observar a resposta da comunicação entre o cliente e o servidor com a tecla não pressionado. Os monitores seriais mostram a mensagem enviada seguido do

CRC calculado pelo programa e é possível notar que a mesma mensagem está sendo observada no cliente e no servidor. A mensagem é *0*, já que a tecla não está pressionada e por este motivo o *LED* azul *D2* está apagado.

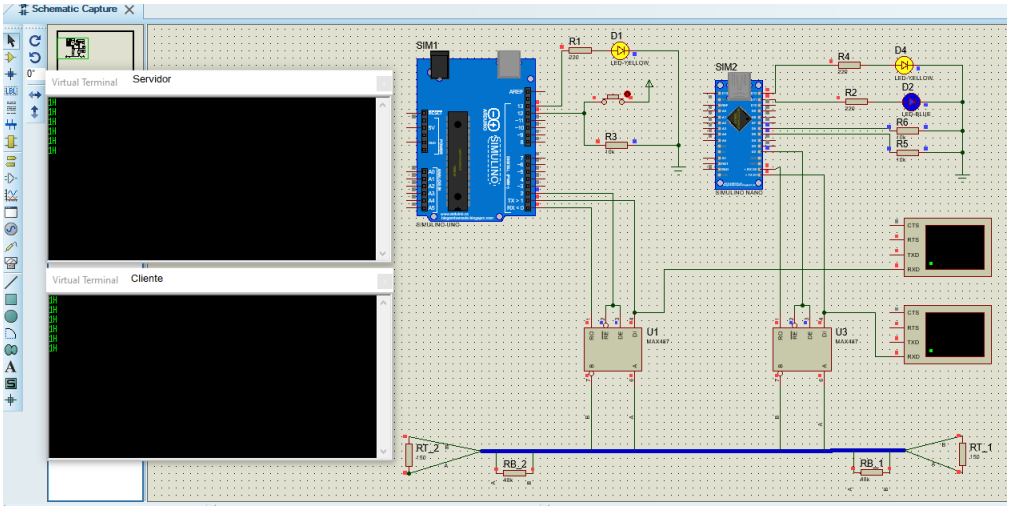

Figura 4.3: Resposta simulada pressionando a tecla para o bloco A

A figura 4.3 mostra a comunicação entre os microcontroladores com a tecla pressionada e validada a leitura da mensagem pelo cliente. É fácil notar que o datagrama reproduzido nos monitores seriais mudou, já que a mensagem é baseada no sinal da tecla e o CRC é baseado na mensagem. A mensagem lida pelo cliente é interpretada pelo cliente e então é acionado o *LED D2*

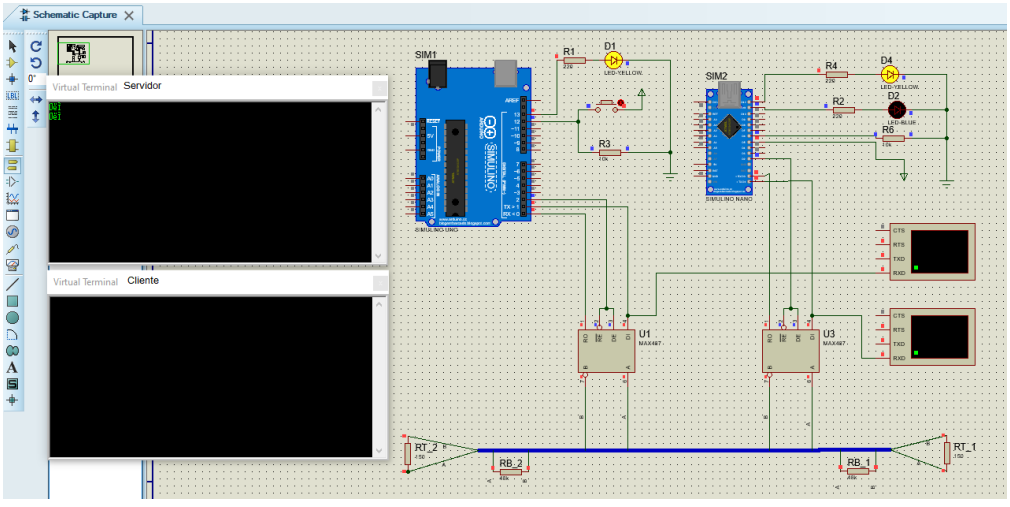

Figura 4.4: Resposta simulada para o bloco B pressionando a tecla

O último teste de comunicação foi feito para checar se a comunicação era exclusiva para o Bloco *A*. Foi alterado o sinal no pino *5* do cliente, mudando assim na camada de aplicação

a sua designação para bloco *B*. Na Figura 4.4 é mostrado o sinal alto na entrada *5* do cliente e é visto que o monitor serial mostra que não há sinal sendo transmitido serialmente para o cliente.

Teste no protótipo

Após aferido o funcionamento na simulação, é feito o mesmo teste no protótipo. Isto é feito para validar o funcionamento do projeto em um sistema real, que pode estar sujeito à interferências externas. Os testes foram feitos utilizando a montagem exposta na Figura 3.4, energizando ambos microcontroladores por suas entradas USB.

Diferentemente do simulador, não há um monitor serial conectado a cada microcontrolador e, por este motivo, a comunicação é confirmada apenas com o uso dos *LEDs*. Apesar disso, os testes de comunicação são similares aos feitos na simulação, usando uma tecla no servidor para enviar uma mensagem para o cliente. São também testadas as situações onde é mudado o endereço do cliente para testar a comunicação com o servidor.

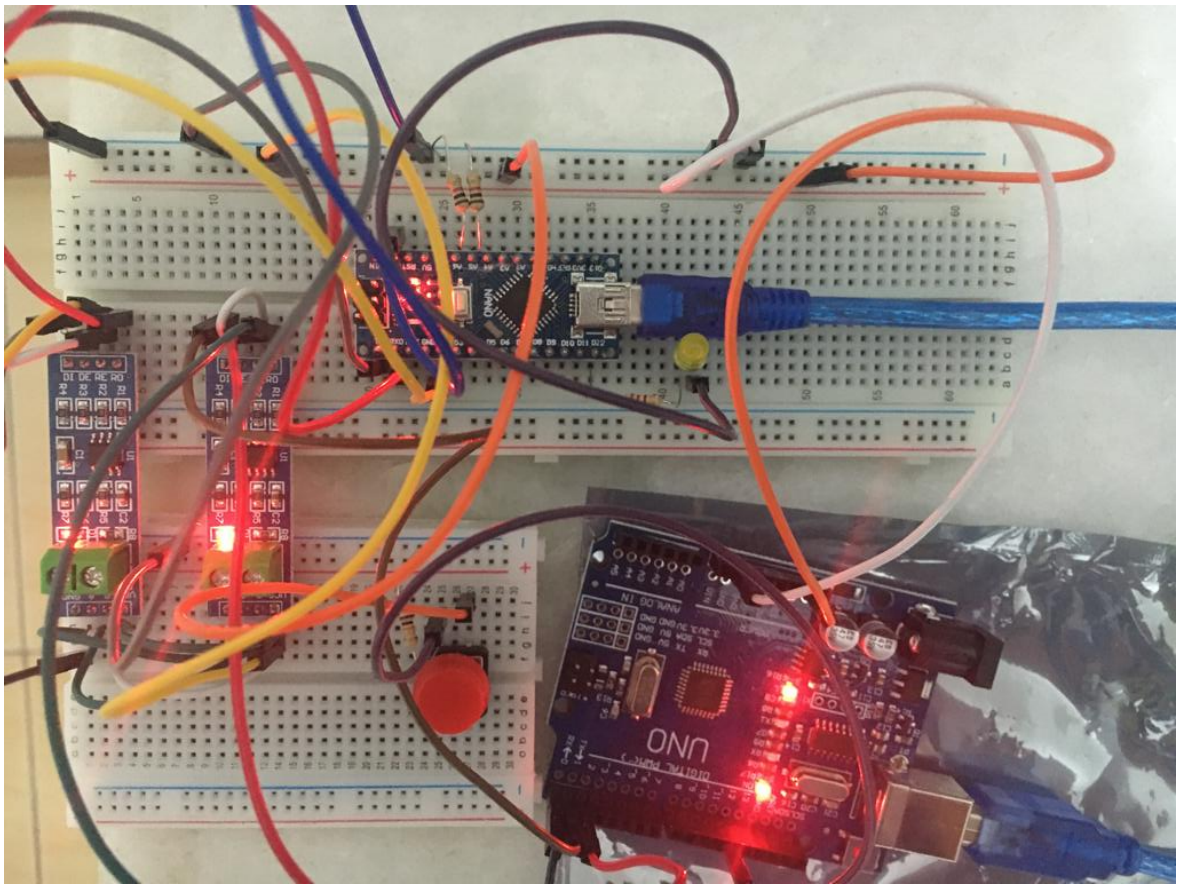

Figura 4.5: Teste como cliente A sem pressionar tecla no protótipo

A Figura 4.5 mostra a ligação feita no protótipo de maneira similar à feita na simulação vista na Figura 4.2. É importante notar que as saídas 5 e 6 do *Arduino* estão ligadas em nível baixo e, portanto, este microcontrolador está representando o bloco A. Enquanto a tecla não é pressionada, o *LED* amarelo se mantém apagado, mesmo com o circuito energizado e conectado à rede de comunicação.

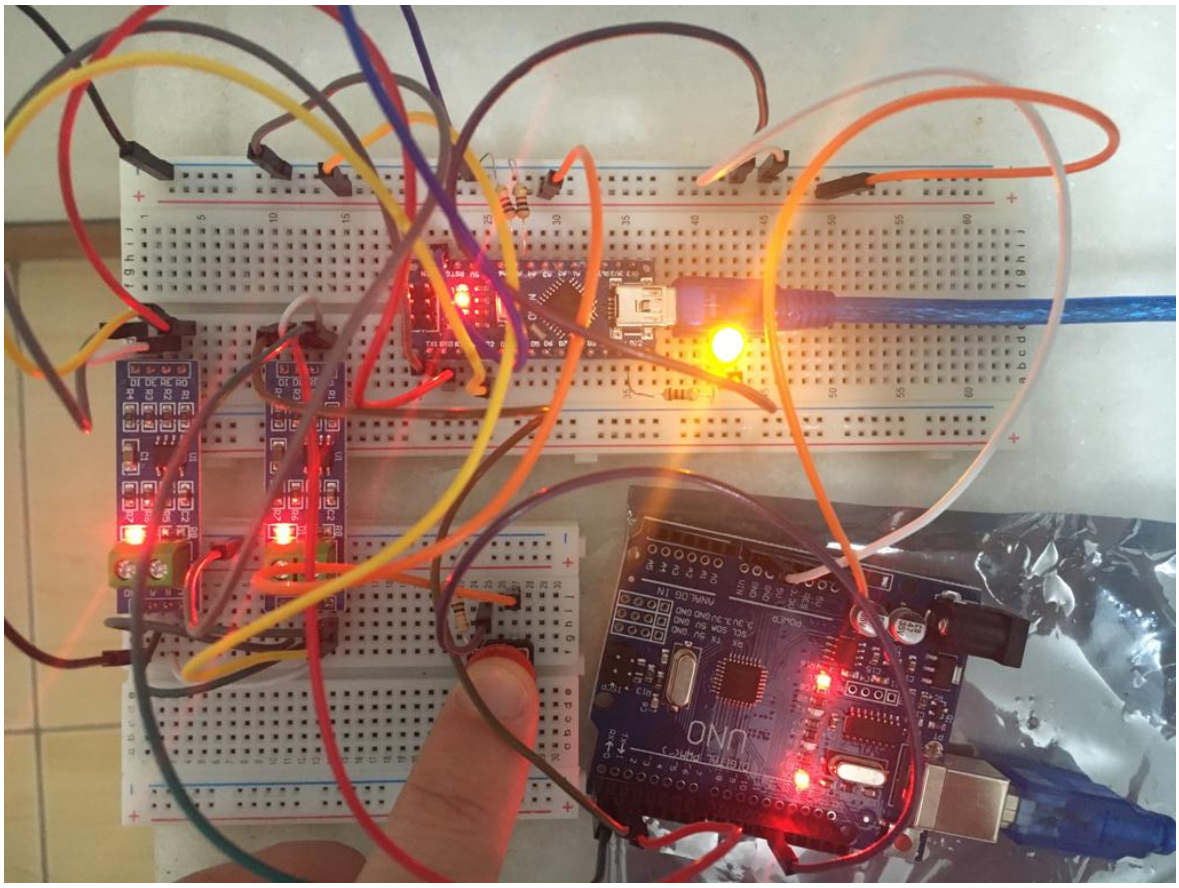

Figura 4.6: Teste como cliente A com tecla pressionada no protótipo

A montagem exposta na Figura 4.6 é a implementação do circuito simulado na Figura 4.3 Quando a tecla vermelha é pressionada, o *LED* amarelo é acionado, como explicado anteriormente. Como mostrado no código (MORAES, 2021a), o servidor está mandando a mensagem apenas para o cliente A, então é necessário o teste da mudança física do cliente para averiguar se só o cliente escolhido está recebendo a mensagem.

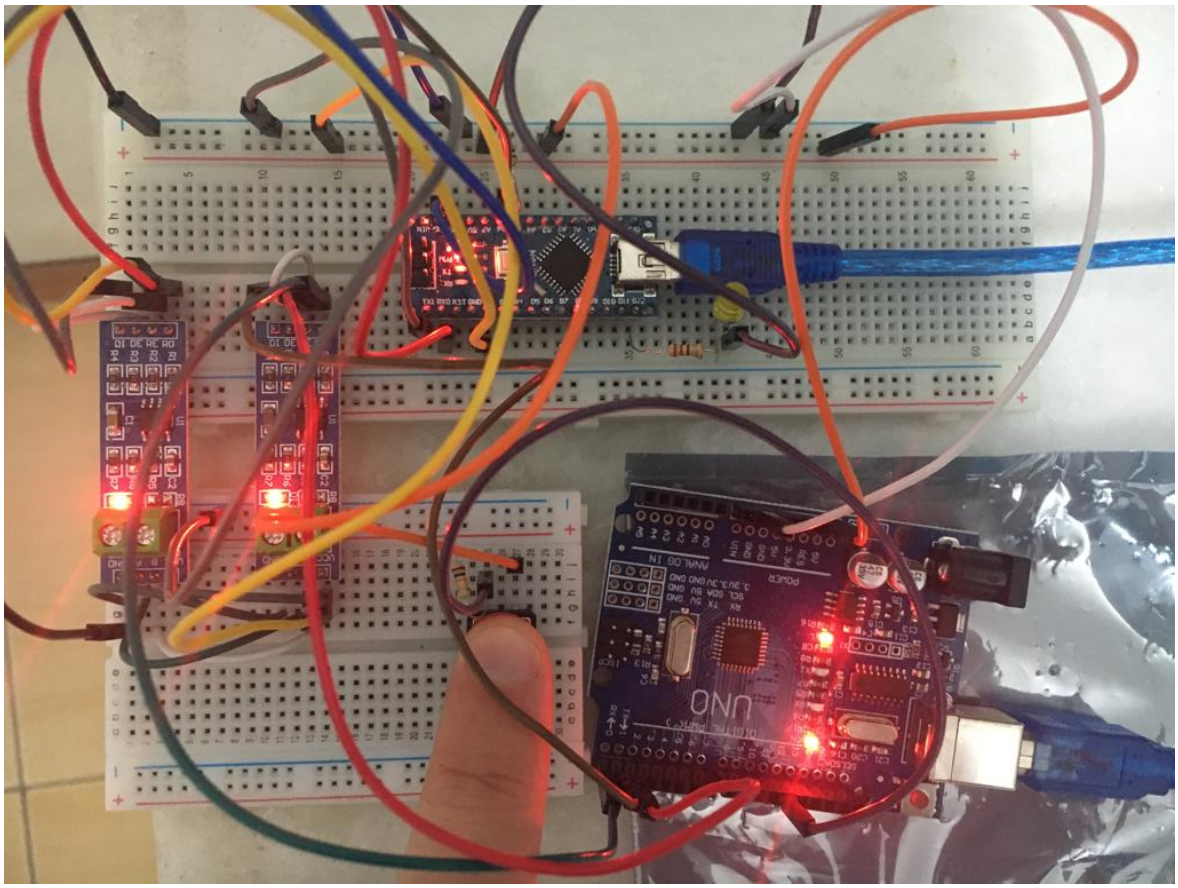

Figura 4.7: Teste como cliente C com tecla pressionada no protótipo

Finalmente, é feita a mudança da entrada 6 do microcontrolador para o sinal alto, definindo este cliente como bloco C. Como feito na Figura 4.4, a Figura 4.7 exibe o teste no circuito físico. Como visto, o *LED* amarelo não acende ao se pressionar a tecla vermelho mesmo com o circuito energizado.

4.1.2 Painel de controle

Após ser testado o transporte de dados com uma rede mais simples, é implementado o painel de controle que é utilizado como a interface homem-máquina para a camada de aplicação do sistema. Primeiro é testado o redirecionamento para os clientes na rede de acordo com o comando dado no painel de controle. Feito isto, é possível então testar o envio da mensagem e a interpretação da mensagem por parte do cliente.

O código do teclado desenvolvido para o painel de controle (MORAES, 2021c) considerou todas as restrições e requerimentos detalhados na subseção 3.2.4. Além disso, também considera a ordem dos botões pressionada, aceitando apenas o bloco, seguido do número do apartamento. Sendo que, a mensagem somente é enviada se o cerquilha for a quinta tecla a

ser pressionada. Isto é feito para garantir a integridade e uniformidade do pacote de dados transmitido pela saída serial do microcontrolador e evitar envios parciais de dados. Como a mensagem é escrita tecla a tecla, é utilizada a variável do tipo *char* (ARDUINO, S., 2019), que possui um tamanho de pelo menos 8 *bits*, a qual é preferencial para caracteres que não são exclusivamente numéricos.

A comunicação é feita com base no endereço de cada cliente, por este motivo, o bloco recebido no datagrama vindo do painel será comparado pelo servidor e redirecionado para o cliente. A figura 4.8 mostra o trecho do código do servidor que faz esta comparação. É importante ressaltar, que apenas um registrador foi utilizado nesta etapa pois apenas o dado de uma tecla estava sendo enviado.

```
case 1:
  if (digitalRead(buttonPin)){au16data[0] = 49;} //Muda a mensagem dependendo se o botão é pressionado
  else{au16data[0] = 48;}
  if (bl == 'A') telegram.u8id = 1; // Endereço cliente 1
  if (bl == 'B') telegram.u8id = 2; // Endereço cliente 2
  if (bl == 'C') telegram.u8id = 3; // Endereço cliente 3
  if (bl == 'D') telegram.u8id = 4; // Endereço cliente 4
  else {u8state=0; break;} //Retorna ao estado inicial para fazer outra requisição
  telegram.u8fct = 6; // código da função (6 é escrita de registrador)
  telegram.u16RegAdd = 0; // Endereço que começa a ser lido o registrador do cliente
  telegram.u16CoilsNo = regSize; // Número de registradores a serem lidos (função 6 aceita apenas 1 e
  telegram.au16reg = au16data; // aponta para o local da memória que foi definido a cima

  master.query( telegram ); // Faz requisição para o cliente
  u8state++;
  break;
```

Figura 4.8: Trecho do código onde os blocos são atribuídos a cada cliente distinto

Como pode ser visto na Figura 4.8(MORAES, 2021a), também é feita uma exceção para caso haja algum erro na variável *Bl*, responsável pela aquisição do bloco. Esta exceção faz com que seja iniciado novamente o processo de comunicação, reduzindo assim a chance do comando ser direcionado ao cliente errado. Esta escolha foi feita por se optar por não enviar comandos para nenhum cliente no caso de endereços desconhecidos, mas pode também ser feita a escolha de redirecionar estas exceções para um cliente específico, como um porteiro ou zelador.

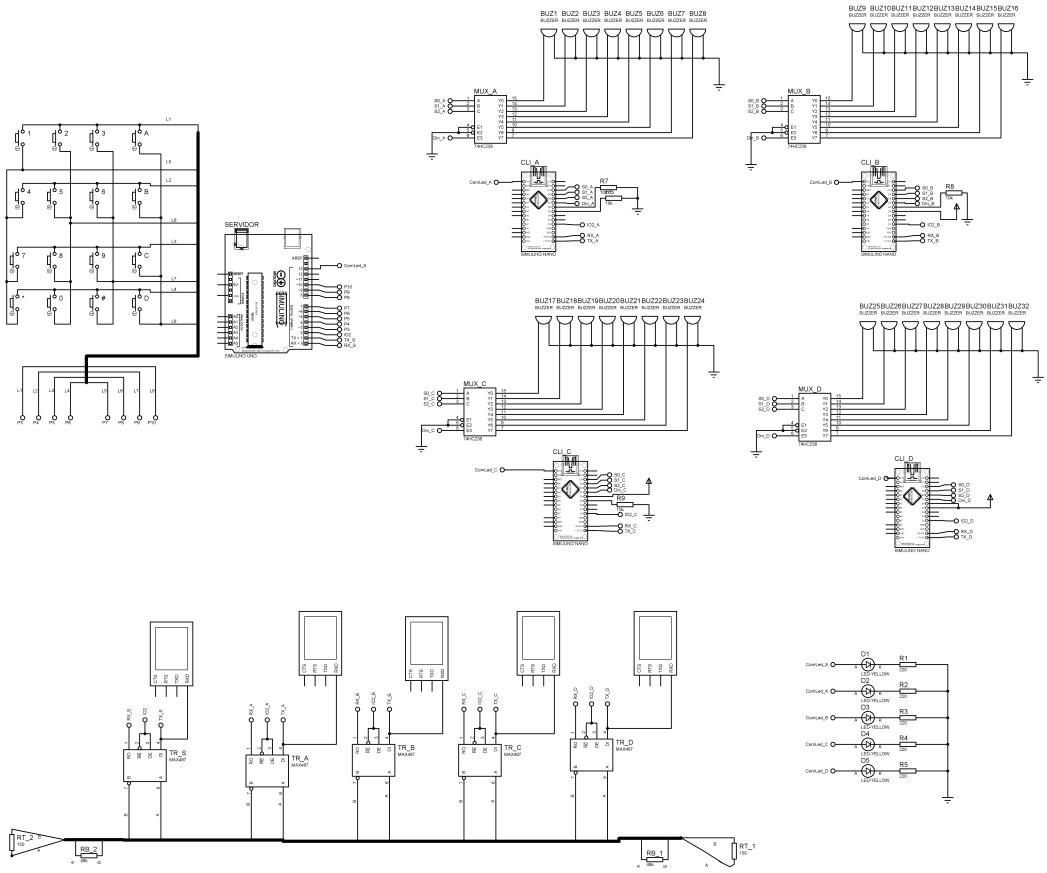

Figura 4.9: Circuito com quatro clientes, servidor e painel de controle

Finalmente, é mostrada na Figura 4.9 a camada física que será utilizada para a próxima etapa do projeto. Todos os microcontroladores já se encontram conectados à rede e o painel de controle conectado ao servidor, mas ainda são necessários adicionar os oito comandos a serem acionados por cada cliente. Este circuito será a base para as implementações seguintes.

4.1.3 Teste da implementação do protocolo *ModBus*

Após o teste de comunicação e a implementação do painel de controle, a última etapa para a validação do projeto é a ativação de uma campainha por meio de um comando no painel de controle. Neste teste, diferentemente do teste feito anteriormente, são utilizadas as campainhas para se aferir o envio da mensagem. Isso implica que o cliente agora deve interpretar a mensagem enviada pelo servidor e acionar o *buzzer* correto.

A mensagem que é recebida pelo teclado e que é interpretada pelo cliente se utiliza da Tabela 3.6, definida no desenvolvimento do projeto, para relacionar a mensagem com a

campainha correta. Então, é importante que a informação digitada no painel de controle seja corretamente transmitida para o cliente. Para que a mensagem seja clara e o projeto mais facilmente escalável, são utilizados quatro registradores para guardarem os quatro dígitos da mensagem registrada pelo painel de controle.

O registrador de 16 bits implementado na biblioteca é do tipo inteiro sem sinal (M., 2019), então os caracteres são enviados separadamente nos registradores para que não haja erros em função da mudança de tipo de *string* para *uint16_t* e de *uint16_t* para um conjunto de *char*. A decisão de utilizar quatro registradores também está relacionada com a legibilidade do código, já que a mensagem escrita no painel, interpretada pelo servidor, transmitida pela rede e lida pelo cliente estão todas no mesmo formato. Como a comunicação na simulação é monitorada serialmente, é também mais fácil ler e testar o tipo *char*.

No código feito para o servidor (MORAES, 2021d), o servidor tenta comunicação até que seja inserida uma mensagem válida no painel de controle. Então, o servidor estabelece a conexão com cliente explicitado na mensagem e o datagrama é enviado para o cliente. É importante ressaltar que a mensagem é primeiramente aferida e validada de acordo com padrões estabelecidos na seção 3.2.4 e então o bloco é aferido uma segunda vez antes de ser enviado para o cliente, similar ao mostrado na Figura 4.8.

Já no código implementado no cliente (MORAES, 2021e) é feita a leitura do primeiro registrador para confirmar o bloco para o qual a mensagem deveria ser enviada. A biblioteca já faz a verificação da mensagem enviada, mas isso é feito como uma medida extra de segurança, pois caso o primeiro registrador não seja o bloco do cliente, há algum problema grave com a comunicação e a mensagem deve ser ignorada. Feito isto, o número do apartamento é guardado em uma variável local e então comparado para encontrar a campainha do apartamento escrito na mensagem.

```
48    if (state>4){ // Verifica se a comunicação foi feita com sucesso
49      digitalWrite(comPin, HIGH);
50      achaBloco = au16data[0];
51    if (achaBloco == B1){ //confere bloco do cliente e caso esteja errado, há um problema com a mensagem
52      achaApto[0]=  au16data[1];
53      achaApto[1]=  au16data[2];
54      achaApto[2]=  au16data[3];
55        if (achaApto[1] == '0'){ //Confere que o terceiro digito do registrador é zero antes de verificar o andar e o apartamento
56          switch (achaApto[0]) { //Lê o segundo digito do registrador e define que campainha tocar de acordo com o segundo e último digitos do registrador
57            case '1':
58              if (achaApto[2] == '1'){
59                digitalWrite(S0, LOW);
60                digitalWrite(S1, LOW);
61                digitalWrite(S2, LOW);
62                inicioBuzzer = millis();
63                digitalWrite(Din, HIGH);
64                } //Liga o primeiro buzzer
65              if (achaApto[2] == '2'){
66                digitalWrite(S0, HIGH);
67                digitalWrite(S1, LOW);
68                digitalWrite(S2, LOW);
69                inicioBuzzer = millis();
70                digitalWrite(Din, HIGH);
71                }//Liga o segundo buzzer
72              break;
```

Figura 4.10: Trecho de código onde são feitos os endereçamentos das campainhas

A Figura 4.10 traz o trecho de código onde o cliente verifica a mensagem, o bloco, o número do apartamento e decide que campainha será acionada. A seta vermelha mostra onde é feita a verificação do bloco e em azul é demarcado onde o vetor do tipo *char* recebe o número do apartamento. Demarcado em verde é onde se define o *buzzer* que será acionado e isto é feito por meio de *switch/case* e *ifs*, mas ha várias maneiras de fazer esta verificação, contanto que se respeite a tabela verdade do decodificador (Tabela 3.5) e as definições de campainha/apartamento (Tabela 3.6).

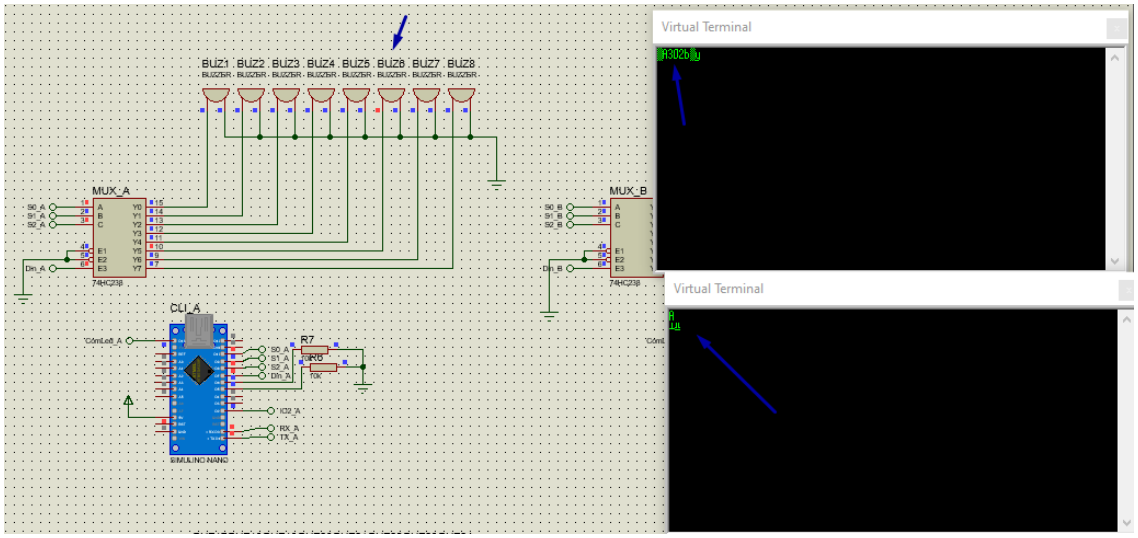

Figura 4.11: Simulação ligando para o apartamento 302

Na Figura 4.11 é possível ver o funcionamento do programa quando é digitada a sequência *A302#*. O monitor serial superior é referente ao servidor e o inferior ao cliente e as setas

azuis sinalizam as mensagens enviadas e recebidas, assim como a campainha acionada. No servidor é mostrada a mensagem enviada pelo cliente, no monitor do cliente é mostrada uma sinalização de mensagem recebida e é possível notar que o *Buzzer 6*, controlado pela saída Y_5 do decodificador que foi acionado.

Com este teste é verificado que a mensagem está sendo enviada para o bloco correto e que a campainha certa está sendo acionada. Agora é feita a verificação para todos os blocos, averiguando se a mensagem enviada para o bloco A é apenas lida pelo cliente correto. A Figura 4.12 faz esta verificação e mostra o monitor serial de cada cliente identificados pela letra do bloco que representam no topo do monitor.

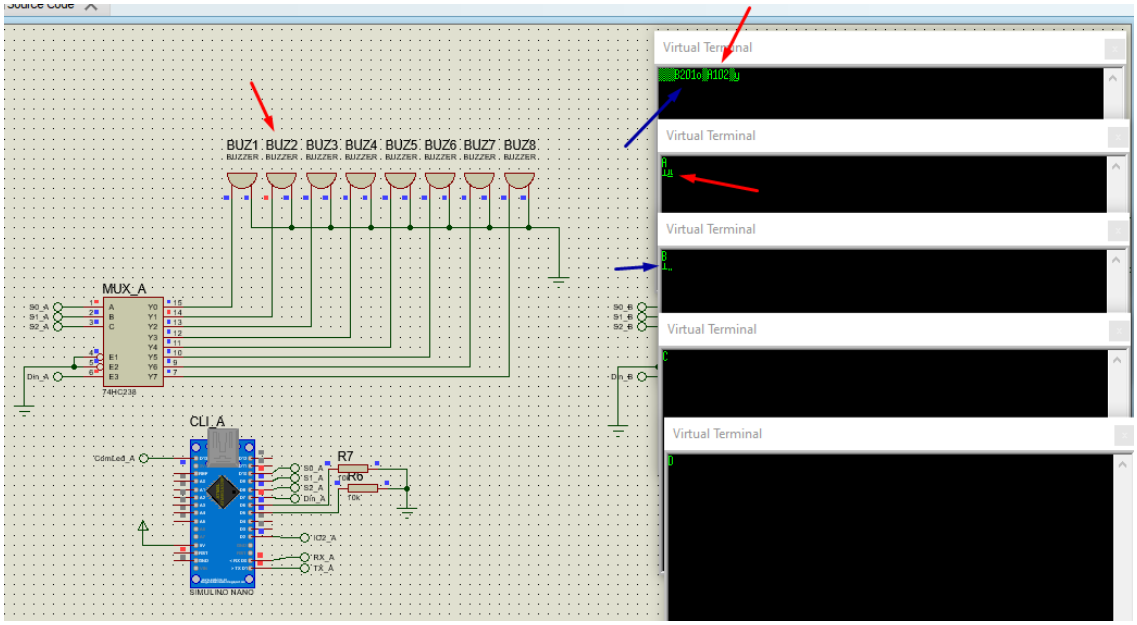

Figura 4.12: Simulação ligando para o apartamento B201 e A102

Na simulação exibida na Figura 4.12, primeiro é digitado *B201#* e então *A102#*. Como esperado e apontado pelas setas azuis, a primeira mensagem foi recebida apenas pelo cliente 2. As setas vermelhas mostram que a segunda mensagem também chegou como esperado e acionou a campainha ligada à saída Y_1 do de-multiplexador.

4.2 Placas de circuito impresso

O projeto tem como base duas ligações distintas de microcontroladores: o Servidor e o Cliente. Para a ligação do Servidor é necessária ligação do teclado matricial 4×4 e do transceptor *Maxim487*, além da alimentação do microcontrolador. Já para a ligação do cliente

não é necessária a ligação do teclado matricial, mas é preciso se considerar a ligação com cada uma das campainhas.

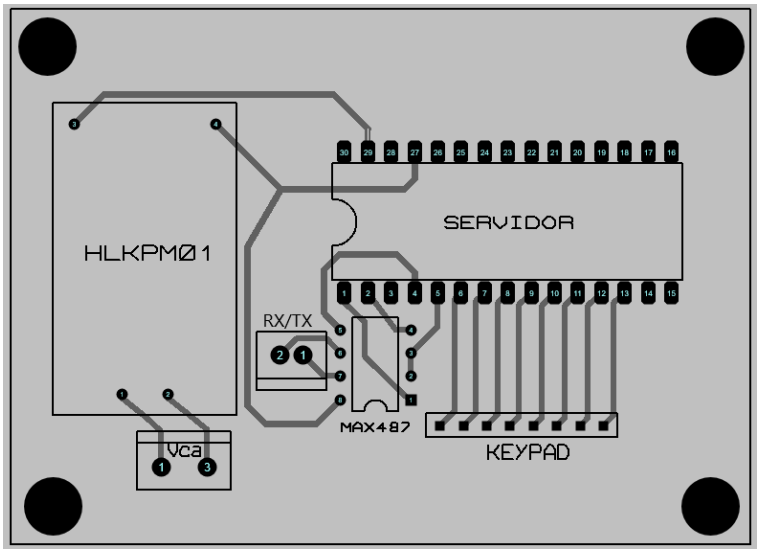

Figura 4.13: Placa de circuito impresso para o servidor

Como discutido na subseção 3.4.3, o projeto para a placa de circuito impresso o *Arduino Nano* é utilizado como microcontrolador e é desenvolvido para dispositivos PTH. A Figura 4.13 mostra o *layout* da placa de circuito impresso (PCI) desenvolvida para o Servidor. Além dos componentes citados anteriormente, foram adicionados um conector borne KRE 2 vias de $3,5\,mm(V_{ca})$, um conector borne KRE 2 vias de $1,5\,mm$(TX/RX), e uma barra de 8 machos à 90°(KEYPAD). O conector V_{ca} é utilizado para a alimentação $100 - 270V$ AC, o conector *TX/RX* para a ligação do servidor à rede *ModBus* e o conector *KEYPAD* para a ligação do teclado de membrana matricial.

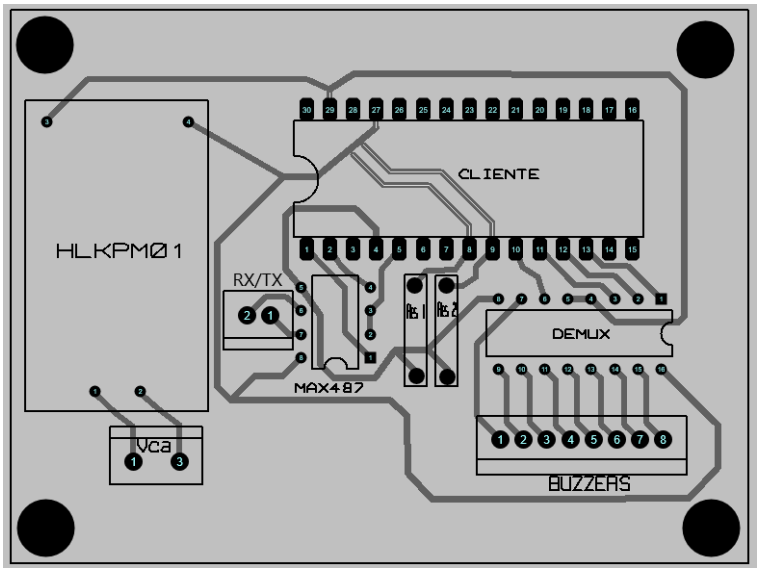

Figura 4.14: Placa de circuito impresso para qualquer cliente

Para ambas as PCI, são incluídos quatro furos M3 para a fixação da placa na caixa de proteção e estes são representados pelos círculos pretos nos quatro cantos das placas. O *layout* da PCI do cliente, mostrado na Figura 4.14, possui os mesmos conectores que o Servidor para a ligação da alimentação e conexão à rede, além de um conector borne KRE 8 vias de $1,5\,mm$(BUZZERS) para a conexão das campainhas à placa. Como o endereçamento do cliente é feito de forma física, há locais para acoplamento de até dois resistores, um conectado à porta digital cinco (pino 8) e outro à porta digital seis (pino 9). Os resistores são conectados para o caso de uma entrada em baixa e há trilhas onde se pode fazer uma ponte entre a porta e o V_{cc}, para o caso de uma entrada em alta.

As placas nas Figuras 4.13 e 4.14 foram dimensionadas com trilhas $T25$ para poderem ser corroídas sem a necessidade de equipamento industrial. Esta escolha reduz muito o custo de produção e a quantidade de placas necessárias para serem produzidas, já que cada placa pode ser corroída individualmente. Para a produção industrial, há tamanhos mínimos padronizados de painéis, com o menor usualmente sendo de 9” $\times$ 12”.

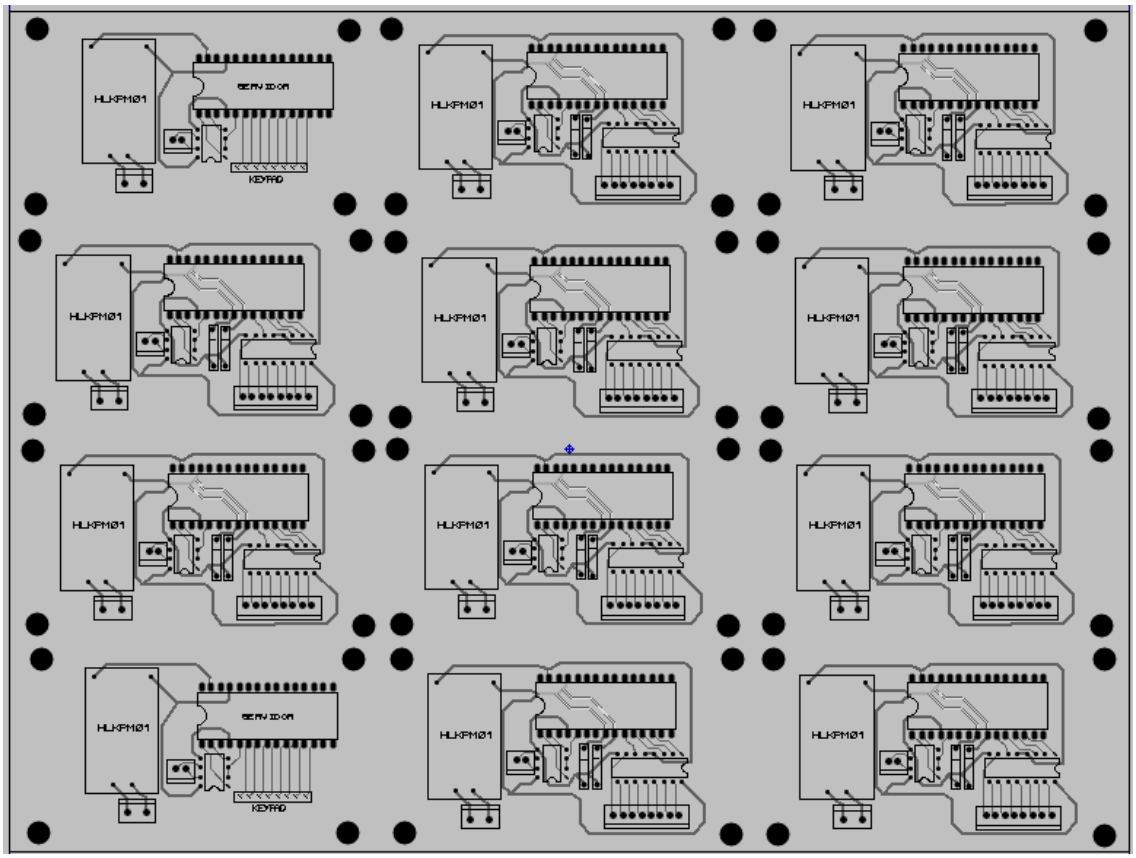

Figura 4.15: Placa de circuito impresso para produção industrial

A figura 4.15 demonstra como seria um painel de 9" × 12" com as PCI para o cliente e servidor. No painel exibido há dois servidores e dez clientes, sendo possível a aplicação em pelo menos dois condomínios de até quatro blocos e pelo menos uma placa de cliente reserva para cada um deles. Outra possibilidade seria fazer um painel apenas de servidores e um apenas de clientes, mas utilizar painéis de dimensões distintas ou produzi-los em quantidades diferentes.

4.3 Orçamento do projeto

Nesta seção é feito um orçamento para a instalação do projeto por completo, com a inclusão dos quatro clientes, o servidor, circuitos integrados necessários, placas de circuito impresso, entre outros. Na Tabela 4.1 é identificado o custo de cada produto, assim como o custo total obtido no orçamento dos componentes necessários e no Apêndice C está o orçamento completo com os valores de entrega e outras informações do fornecedor. Neste orçamento estão inclusos 50m de fio para a alimentação dos microcontroladores.

Tabela 4.1: Custo total para a implementação de um mínimo produto viável

Componente	Quantidade	Custo total (R$)
Arduino Nano	5	169,80
De-multiplexador 74HC238	4	24,00
MAX487	5	50,00
Mini fonte Hi-Link PM01	5	224,50
Teclado matricial 4x4	1	11,99
Buzzer ativo 5V	50	99,00
Borne KRE 2 vias (3,5mm)	5	12,00
Borne KRE 2 vias (1,5mm)	5	9,99
Borne KRE 8 vias (1,5mm)	5	30,00
Resistor 10K Ohms	100	12,49
Tubo de estanho para solda (25g)	1	13,54
Placa fenolite (10x15cm)	5	62,45
Fio flexível 0,75mm (100m)	3	235,44
Fio flexível 2mm (50m)	1	79,00
Total		1034,20

Também estão incluídos 100m de fio para a comunicação entre o servidor e os clientes, mas este valor varia de acordo com a distância entre os blocos e o painel de controle. Conforme requerido pelo protocolo (MODBUS, 2012) são utilizados três cabos de 0,75mm para a comunicação entre os componentes ligados à rede, sendo dois cabos para a comunicação e um cabo comum para regular a impedância e proteger de ruídos da rede, já que os cabos não são blindados.

As placas de fenolite são suficientes para montar dez placas de circuito impresso, considerando que as placas desenhadas nas Figuras 4.13 e 4.14 possuem dimensão de $6x8cm$. As placas de fenolite não se fariam necessárias caso seja escolhida a alternativa da montagem industrial da placa de circuito impresso. A escolha da alternativa industrial necessita de um investimento mais alto, mas produz uma quantidade maior de PCIs de maior qualidade e custo-benefício.

Capítulo 5

CONCLUSÃO

5.1 Conclusões

Os estudos feitos sobre o modelo OSI (2.2.1) e protocolo industrial *ModBus* (2.2.3) foram cruciais para o desenvolvimento deste projeto. O modelo OSI facilitou o planejamento, divisão e desenvolvimento, por segmentar o trabalho da mesma maneira que o protocolo *ModBus* o faz. A organização *Modbus* possui, muito bem padronizados, os passos necessários para se construir a rede de comunicação.

O desenvolvimento do projeto seguiu o modelo descrito na fundamentação, iniciando-se pela camada física, conectados à camada de dados e então especificada a camada de aplicação. As camadas 1 e 2 foram especificadas em conjunto, com base no protocolo (MODBUS, 2012) e foram contempladas todas as características necessárias para o projeto. A camada de aplicação foi dividida em duas partes: o painel de controle e as funções requeridas pelo protocolo industrial, assim como as verificações de erros necessárias. Além disso, foram feitas todas as considerações práticas necessárias para o projeto e instalação de um Mínimo Produto Viável (MVP) em um condomínio.

Os resultados expostos foram referentes ao teste de comunicação no protótipo, a simulação da rede em conjunto com camada de aplicação e o painel de controle, que foram feitas sob a camada física, e ao desenho do layout da placa de circuito impresso. A comunicação entre o cliente e o servidor com a utilização da camada de aplicação baseada na biblioteca escolhida (M., 2019) trouxe os resultados esperados, demonstrando uma comunicação funcional e consistente da rede. A comunicação no sistema é flexível e robusta o suficiente para, respectivamente, expansão futura e evitar erros.

O layout da placa de circuito impresso é feito de modo a poder se utilizar uma abordagem mais econômica ou em uma escala maior. O orçamento leva em consideração a menor escala,

já que é sabido que há muito potencial de crescimento no projeto e de inclusão de outros periféricos e melhorias. Em suma, foi possível construir uma base sólida de um sistema de múltiplos comandos, com flexibilidade e robustez com o uso de um protocolo industrial bem fundamentado e confiável.

5.2 Trabalhos Futuros

Este projeto pode servir de base para diversas implementações de melhorias futuras, já que utiliza um protocolo bastante popular e com muitas aplicações no mercado. Estas melhorias podem ser outras funções incrementadas ao trabalho atual ou outras aplicações derivadas do acionamento de múltiplos comandos. Primeiro, são colocadas sugestões de outras funções que poderiam ser adicionadas ao sistema de múltiplas campainhas e então são citadas outras alternativas de aplicação para o sistema de múltiplos comandos.

Como o sistema possui comunicação serial, seria possível a implementação de um módulo para a obtenção e armazenamento dos dados em um *chip* de memória SSD. Esta implementação aumentaria o valor de implementação e o consumo de memória e operacional do projeto, já que se faria necessário um módulo de memória SSD e um módulo de relógio para armazenar a data e horário dos dados obtidos. Nesta aplicação, poderiam ser coletados os erros do sistema embarcado e feita a análise de desempenho a longo prazo do sistema, podendo assim desenvolver um projeto com base no funcionamento a longo prazo do sistema de múltiplos comandos.

Outra alternativa é instalar também no projeto uma interface que utiliza a rede sem fio e protocolo TCP/IP em conjunto com o *ModBus*. Desta maneira é possível aumentar as possibilidades do que pode ser feito no projeto, como a implementação de um armazenador dos dados em *PHP* ou a instalação de uma câmera de segurança na entrada. Como o uso do protocolo *ModBus* possui uma maior imunidade à ruídos eletromagnéticos e de interferências causados por sinais de rádio-frequência, o circuito desenvolvido neste trabalho poderia funcionar como uma rede auxiliar de maior robustez, em que o sistema poderia manter suas funções básicas mesmo se houvesse um problema na rede devido à uma grande quantidade de ruídos externos ou da saturação das bandas disponíveis para o sinal de rádio-frequência.

Com a rede de múltiplos comandos instalada, há a possibilidade da instalação de outras funções além da campainha. Uma alternativa que poderia utilizar a mesma rede com apenas alguns acréscimos de módulos é um sistema de alarme de incêndio. A rede de comunicação e as campainhas poderiam ser mantidas, mas precisaria ser implementada uma mudança de tempo nas campainhas para diferenciação para o som do alarme e serem feitas outras

considerações de segurança, assim como a possibilidade de acionamento em paralelo de outras medidas de prevenção e combate ao incêndio.

Referências

ALANI, Mohammed M. *Guide to OSI and TCP/IP Models*. Muscat, Oman: Springer, 2014. ISBN 978-3-319-05151-2.

AMELCO. *Porteiro predial AM-PPR*. [S.l.: s.n.], 2010. Manual de Instalação.

ANDREWS, Christopher. Arduino SA. 2015. Disponível em: <https://playground.arduino.cc/Code/Keypad/#Description>.

ARDUINO. Arduino. 2021. Disponível em: <https://store-usa.arduino.cc/products/arduino-nano/>.

ARDUINO, SA. Arduino SA. 2019. Disponível em: <https://www.arduino.cc/reference/en/language/variables/data-types/char/>.

INTELBRAS. *Central de portaria Comunic 48*. [S.l.: s.n.], 2020. Manual do usuário.

_____. *Módulo de portaria inteligente 1000*. [S.l.: s.n.], 2020. Manual do usuário.

LAWS, David. Computer History Museum. 2021. Disponível em: <https://www.computerhistory.org/siliconengine/timeline/>.

HI-LINK. *3W Ultra-small Power Module PM03/PM01/PM09/PM12 Datasheet*. [S.l.: s.n.], 2018. Folha de dados das fontes Hi-Link de 3W de potência.

M., Samuel. Arduino SA. 2014. Disponível em: <https://forum.arduino.cc/t/new-modbus-master-slave-library/207855/10>.

_____. Arduino SA. 2019. Disponível em: <https://github.com/smarmengol/Modbus-Master-Slave-for-Arduino>.

MAXIM. *Datasheet MAX481-1487*. [S.l.: s.n.], 2014. Manual dos transceptores de baixa potência RS-485/RS-422.

MODBUS, Organization. *MODBUS over Serial Line: Specification and Implementation Guide*. [S.l.: s.n.], 2012. Descreve o protocolo de aplicação do ModBus nas camadas 1 e 2 do modelo OSI.

MODBUS, Organization. Modbus Organization. 2021. Disponível em: <https://modbus.org/>.

_____. *MODBUS Application Protocol Specification*. [S.l.: s.n.], 2012. Descreve o protocolo de aplicação do ModBus.

MORAES, Cauê Emilio de. 2021. Disponível em: <`https://github.com/CaueEmilio/Arduino_ModBus_TCC/blob/main/Servidor_V1.ino`>.

_____. 2021. Disponível em: <`https://github.com/CaueEmilio/Arduino_ModBus_TCC/blob/main/Cliente_V1.ino`>.

_____. 2021. Disponível em: <`https://github.com/CaueEmilio/Arduino_ModBus_TCC/blob/main/Just_Keypad.ino`>.

_____. 2021. Disponível em: <`https://github.com/CaueEmilio/Arduino_ModBus_TCC/blob/main/Servidor_V2.ino`>.

_____. 2021. Disponível em: <`https://github.com/CaueEmilio/Arduino_ModBus_TCC/blob/main/Cliente_V2.ino`>.

PHILLIPS. *74HC family Datasheet*. [S.l.: s.n.], 1988. Folha de dados da família de circuitos integrados 74HC.

_____. *74HC/HCT238 3-to-8 line decoder/demultiplexer Datasheet*. [S.l.: s.n.], 1990. Folha de dados do de-multiplexador.

PIRES, João J. O. *Sistemas e Redes de Telecomunicações*. [S.l.: s.n.], 2006.

PRO-SIGNAL. *Buzzer Datasheet*. [S.l.: s.n.], 2016. Folha de dados do Buzzer.

PROTOSUPPLIES. 2021. Disponível em: <`https://protosupplies.com/product/membrane-keypad-4x4-matrix/`>.

THOMAS, Mini S; MCDONALD, John D. *Power system SCADA and smart grids*. [S.l.]: CRC Press, 2015. ISBN 978-1-4822-2675-1.

UIT. União Internacional de Telecomunicações. 2021. Disponível em: <`https://news.un.org/pt/tags/uniao-internacional-de-telecomunicacoes`>.

WAKERLY, John F. *Digital Design: Principles and Practices*. 4th. [S.l.]: Prentice Hall, 2005. ISBN 978-0-1318-6389-7.

WILLIAMS, Tim. *The Circuit Designer's Companion 2nd Edition*. [S.l.]: Elsevier, 2005. ISBN 0-7506-6370-7.

YONG-JIE, WANG; ZONG-JIN, QIN. Design of lighting circuit system based on RS485 bus, 2014.

Apêndice A

Descrição geral Max487

MAX481/MAX483/MAX485/ MAX487–MAX491/MAX1487

Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers

General Description

The MAX481, MAX483, MAX485, MAX487–MAX491, and MAX1487 are low-power transceivers for RS-485 and RS-422 communication. Each part contains one driver and one receiver. The MAX483, MAX487, MAX488, and MAX489 feature reduced slew-rate drivers that minimize EMI and reduce reflections caused by improperly terminated cables, thus allowing error-free data transmission up to 250kbps. The driver slew rates of the MAX481, MAX485, MAX490, MAX491, and MAX1487 are not limited, allowing them to transmit up to 2.5Mbps.

These transceivers draw between 120µA and 500µA of supply current when unloaded or fully loaded with disabled drivers. Additionally, the MAX481, MAX483, and MAX487 have a low-current shutdown mode in which they consume only 0.1µA. All parts operate from a single 5V supply.

Drivers are short-circuit current limited and are protected against excessive power dissipation by thermal shutdown circuitry that places the driver outputs into a high-impedance state. The receiver input has a fail-safe feature that guarantees a logic-high output if the input is open circuit.

The MAX487 and MAX1487 feature quarter-unit-load receiver input impedance, allowing up to 128 MAX487/MAX1487 transceivers on the bus. Full-duplex communications are obtained using the MAX488–MAX491, while the MAX481, MAX483, MAX485, MAX487, and MAX1487 are designed for half-duplex applications.

Applications

Low-Power RS-485 Transceivers

Low-Power RS-422 Transceivers

Level Translators

Transceivers for EMI-Sensitive Applications

Industrial-Control Local Area Networks

Next Generation Device Features

- **For Fault-Tolerant Applications**
  MAX3430: ±80V Fault-Protected, Fail-Safe, 1/4 Unit Load, +3.3V, RS-485 Transceiver
  MAX3440E–MAX3444E: ±15kV ESD-Protected, ±60V Fault-Protected, 10Mbps, Fail-Safe, RS-485/J1708 Transceivers
- **For Space-Constrained Applications**
  MAX3460–MAX3464: +5V, Fail-Safe, 20Mbps, Profibus RS-485/RS-422 Transceivers
  MAX3362: +3.3V, High-Speed, RS-485/RS-422 Transceiver in a SOT23 Package
  MAX3280E–MAX3284E: ±15kV ESD-Protected, 52Mbps, +3V to +5.5V, SOT23, RS-485/RS-422, True Fail-Safe Receivers
  MAX3293/MAX3294/MAX3295: 20Mbps, +3.3V, SOT23, RS-485/RS-422 Transmitters
- **For Multiple Transceiver Applications**
  MAX3030E–MAX3033E: ±15kV ESD-Protected, +3.3V, Quad RS-422 Transmitters
- **For Fail-Safe Applications**
  MAX3080–MAX3089: Fail-Safe, High-Speed (10Mbps), Slew-Rate-Limited RS-485/RS-422 Transceivers
- **For Low-Voltage Applications**
  MAX3483E/MAX3485E/MAX3486E/MAX3488E/MAX3490E/MAX3491E: +3.3V Powered, ±15kV ESD-Protected, 12Mbps, Slew-Rate-Limited, True RS-485/RS-422 Transceivers

Ordering Information appears at end of data sheet.

Selection Table

PART NUMBER	HALF/FULL DUPLEX	DATA RATE (Mbps)	SLEW-RATE LIMITED	LOW-POWER SHUTDOWN	RECEIVER/ DRIVER ENABLE	QUIESCENT CURRENT (µA)	NUMBER OF RECEIVERS ON BUS	PIN COUNT
MAX481	Half	2.5	No	Yes	Yes	300	32	8
MAX483	Half	0.25	Yes	Yes	Yes	120	32	8
MAX485	Half	2.5	No	No	Yes	300	32	8
MAX487	Half	0.25	Yes	Yes	Yes	120	128	8
MAX488	Full	0.25	Yes	No	No	120	32	8
MAX489	Full	0.25	Yes	No	Yes	120	32	14
MAX490	Full	2.5	No	No	No	300	32	8
MAX491	Full	2.5	No	No	Yes	300	32	14
MAX1487	Half	2.5	No	No	Yes	230	128	8

For pricing, delivery, and ordering information, please contact Maxim Direct at 1-888-629-4642, or visit Maxim Integrated's website at www.maximintegrated.com.

19-0122; Rev 10; 9/14

Apêndice B

Orçamento do protótipo

Baú da Eletrônica
Componentes Eletrônicos

Atendimento:
11 2442-6600

CARRINHO DE COMPRAS

Fechar Pedido

	NOME DO PRODUTO	QUANTIDADE	PREÇO UNITÁRIO	SUBTOTAL	REMOVER
	Arduino Nano R3 + Cabo mini USB	1	R$ 51,17	R$ 51,17	
	Arduino Uno R3 + Cabo USB 2.0 - A-B	1	R$ 78,00	R$ 78,00	
	Protoboard 830 Pontos - MP-830A - Minipa	2	R$ 31,41	R$ 62,82	
	Conversor de Dados TTL Para RS485 - CDT11	2	R$ 9,01	R$ 18,02	
	Resistor 220R 5% (1/4W)	10	R$ 0,05	R$ 0,50	

Ajuda

JA

LIMPAR CARRINHO

	NOME DO PRODUTO	QUANTIDADE	PREÇO UNITÁRIO	SUBTOTAL	REMOVER
	Botão Arduino Pro Mini	2	R$ 0,22	R$ 0,44	
	LED Difuso 10mm Vermelho	4	R$ 0,63	R$ 2,52	
	Resistor 100K 5% (1/4W)	4	R$ 0,05	R$ 0,20	

VOLTAR PARA LOJA

LIMPAR CARRINHO

CALCULAR FRETE

35503-822 OK

Não sei meu CEP

Correios

- PAC - Em média 8 dia(s) **R$ 27,09**
- Sedex - Em média 4 dia(s) **R$ 54,20**

Serviços de Entrega

- Dlog Expresso - Em média 5 dia(s) útil(eis) **R$ 26,56**
- PAC - Em média 9 dia(s) útil(eis) **R$ 32,20**
- SEDEX - Em média 5 dia(s) útil(eis) **R$ 63,38**

Retire em loja

- Retirar na loja: Guarulhos-SP – até 1 dia útil **R$ 0,00**

CUPOM DE DESCONTO

OK

SUBTOTAL

R$ 213,67

FRETE (CORREIOS - PAC - EM MÉDIA 8 DIA(S)) R$ 27,09

VALOR TOTAL **R$ 240,76**

Fechar Pedido

Aproveite e leve também

Fonte bivolt 9V / 1A para Arduino

★★★★★ (42)

R$ 17,92 (5% no boleto bancário)

~~De: R$ 20,95~~

Por: **R$ 18,86**

em **1x** de **R$ 18,86** s/ juros

Adicionar ao carrinho

Buscar produtos, marcas e muito mais...

9+

Enviar para Divinópolis 35503822

Carrinho (11) Salvos (0)

Módulo Conversor Bidirecional Rs485 Ttl Arduino Pic Rasp Pic

Frete grátis FULL

-+2
5 disponíveis

R$ 25 80

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Protoboard 830 Furos Pronta Entrega

Frete grátis FULL

-+2
25 disponíveis

R$ 43 88

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Cabo Wire Jumper 20cm 40 Fios Macho-macho Protoboard Arduino

Frete grátis FULL

-+1
33 disponíveis

R$ 18 30

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Resistor (1/4w) Vários Modelos - 100 Unidades

Tipo: 100 k ohms Alterar

-+1
2 disponíveis

R$ 13 00

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Economize no frete adicionando produtos do mesmo vendedor

- 25pçs -+1
3 disponíveis

R$ 9 45

agoraSalvar para depois

Kit 10 X Botão Chave Microswitch Push Button 4 Pinos Arduino

-+1
951 disponíveis

R$ 9 99

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Resistor De Carbono 220r 5% 1/4w - Cr25 -05pçs

-+1
55 disponíveis

R$ 7 50

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Placa Nano V3 Com Cabo Usb (Compatível Com Arduino)

-+1
238 disponíveis

R$ 37 57

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Placa Compatível Arduino Uno R3 Atmega328 Smd + Cabo E Pinos

-+1
7 disponíveis

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

R$ 46 00

Envio para Divinópolis, CEP 35503822

~~R$ 95 20~~ R$ 71 40

Frete dos produtos Full

~~R$ 23 80~~ Grátis

Frete dos outros produtos

R$ 71 40

Total com frete

R$ 282 89
Em até 6x

Continuar a compra

Apêndice C

Orçamento do projeto

Buscar produtos, marcas e muito mais...

9+

Enviar para Divinópolis 35503822

Carrinho (33) Salvos (0)

Tubo Estanho Para Solda Azul 63x37 25 Gramas 1mm

Frete grátis FULL

-+1
118 disponíveis

R$ 13 54

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Resistor 10k Ohms - 100 Unidades

Frete grátis FULL

-+1
74 disponíveis

R$ 12 49

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Teclado Membrana Matricial 4x4 Teclas Arduino Raspberry Pic

Frete grátis FULL

-+1
2 disponíveis

R$ 11 99

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Mini Fonte Hi-link Hlk-pm01 100~240vac P/ 5v Dc 600ma 3w

Frete grátis FULL

-+5
5 disponíveis

R$ 224 50

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Rolo Fio Cabinho Flexível Cobre 0,75mm Rolos Com 100 Metros

Frete grátis

-+3
3524 disponíveis

R$ 235 44

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Fio Flexível 2mm Rolo Com 50 Metros Fio Energia Promoção

Cores: Vermelho Alterar

Frete grátis

-+1
9997 disponíveis

R$ 79 00

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Placa De Fenolite Cobreada 10x15cm Pci Circuito Impresso Pcb

-+2
3 disponíveis

R$ 24 98

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Placa De Fenolite Cobreada 10x15cm Pci Circuito Impresso Pcb

-+3
3 disponíveis

R$ 37 47

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Borne Kre Cinza 8 Terminais Placa Pcb (5 Peças)

-+1
78 disponíveis

R$ 30 00

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

5 Kre 2 Vias - Kf128 - Borne - 5,08mm

-+1
5 disponíveis

R$ 12 00

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

5 Unidades Conector Borne Kre Kre2 Kf301 2 Vias Pci

-+1
12 disponíveis

R$ 9 99

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

50 Buzzer 5v Ativo Para Arduino Robótica Automação Eletronic

Frete grátis

-+1
2 disponíveis

R$ 99 00

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Componente Eletrônico Max487cpa Dip / Max487

-+5
829 disponíveis

R$ 50 00

ExcluirMais produtos do vendedorComprar agoraSalvar para depois

Sn74hc238 Kit Com 02 Peças -+2

R$ 24 00

Excluir Mais produtos do vendedor Comprar agora Salvar para depois

10 disponíveis

Arduino Nano V3 328p Ch340g - Pronta Entrega

Frete grátis

-+5
800 disponíveis

R$ 169^80

Excluir Mais produtos do vendedor Comprar agora Salvar para depois

Envio para Divinópolis, CEP 35503822

~~R$ 362^50~~ R$ 223^73

Frete dos produtos Full

~~R$ 65^20~~ Grátis

Frete dos outros produtos

R$ 223^73

Total com frete

R$ 1.257^93
Em até 10x

Continuar a compra

Recomendações para você

R$ 12^99
2x R$ 6^50 sem juros
Teclado Membrana Matricial 4x4 16 Teclas P/ Robótica

R$ 14^90
FULL
Modulo Serial I2c / Iic P/ Lcd 16x2 - Arduino / Pic

R$ 26^49
Display Oled 128x64 0.96 I2c Gráfico Arduino Azul E

Produtos que te interessaram