UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
BACHARELADO EM ENGENHARIA ELETRÔNICA

BRUNO LUCAS MUNARINI

DESENVOLVIMENTO DE UM KIT DIDÁTICO PARA ELETRÔNICA DIGITAL

TRABALHO DE CONCLUSÃO DE CURSO

CAMPO MOURÃO
2016

BRUNO LUCAS MUNARINI

DESENVOLVIMENTO DE UM KIT DIDÁTICO PARA ELETRÔNICA DIGITAL

Trabalho de Conclusão de Curso de Graduação do curso de Bacharelado em Engenharia Eletrônica do Departamento Acadêmico de Eletrônica (DAELN) da Universidade Tecnológica Federal do Paraná, como requisito final para obtenção do título de Engenheiro em Eletrônica.

Orientador: Prof. Dr. Marcio Rodrigues da Cunha

CAMPO MOURÃO
2016

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Campo Mourão
Coordenação do Curso de Engenharia Eletrônica

TERMO DE APROVAÇÃO
DO TRABALHO DE CONCLUSÃO DE CURSO INTITULADO

Desenvolvimento de um kit didático para eletrônica digital

por

Bruno Lucas Munarini

Trabalho de Conclusão de Curso apresentado no dia 22 de Novembro de 2016 ao Curso Superior de Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná, Campus Campo Mourão. O Candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____

Prof. Me. Flávio Luiz Rossini
(UTFPR)

_____

Prof. Fábio Pereira de Lima
(UTFPR)

_____

Prof. Dr. Marcio Rodrigues da Cunha
(UTFPR)
Orientador

A Folha de Aprovação assinada encontra-se na Coordenação do Curso

RESUMO

BRUNO, Lucas Munarini. **Desenvolvimento de um kit didático para eletrônica digital**. 2016. 92 f. Trabalho de Conclusão de Curso - Engenharia Eletrônica. Universidade Tecnológica Federal do Paraná. Campo Mourão, 2016.

Este trabalho tem como objetivo o desenvolvimento de um kit didático para Eletrônica Digital, com finalidade de auxiliar desde conceitos iniciais básicos como Tabela Verdade, Mapa de Karnaugh e Expressão Booleana, até a implementação física de um circuito digital e seus testes. É apresentado um referencial teórico dos conteúdos de Eletrônica Digital e de como o uso da tecnologia pode influenciar na absorção do conteúdo ministrado aos discentes. São apresentadas as principais dificuldades dos alunos, assim como a eficiência dos métodos de ensino utilizados atualmente. Após a definição dos recursos necessários para o desenvolvimento do projeto, são apresentados todos os itens referentes à sua implementação, tais como: componentes utilizados na composição do *hardware* e justificativa, programa de controle desenvolvido para o microcontrolador, simulações, *software* de comunicação via *bluetooth* e resultados encontrados. A conclusão do trabalho apresenta uma síntese dos resultados alcançados por meio da aplicação de dois roteiros de laboratório, onde se verifica a eficácia tanto na elaboração de um projeto de um circuito lógico a partir dos dados iniciais e básicos, como na extração dos mesmos de forma automatizada por meio de um circuito previamente implementado. Em geral os resultados encontrados foram satisfatórios, o que justifica a utilização do kit em sala de aula.

Palavras-chave: Tabela Verdade, Mapa de Karnaugh, Expressão Booleana, Kit Didático, Eletrônica Digital.

ABSTRACT

BRUNO, Lucas Munarini. **Development of an educational kit for digital electronics**. 2016. 92 f. Trabalho de Conclusão de Curso - Bacharelado em Engenharia Eletrônica. Universidade Tecnológica Federal do Paraná. Campo Mourão, 2016.

This work has as the objective to develop a teaching kit for Digital Electronics, with the aim of aiding on its basic concepts from the Truth Table, Karnaugh Map and Boolean Expression, to the physical implementation of a digital circuit and it's tests. It presents the theoretical reference of the contents of Digital Electronics and how the use of the technology can influence in the absorption of the content taught to the students. Here are presented the main challenges faced by the students, as well as the efficiency of the teaching methods used nowadays. After the definition of the necessary resources for the development of the project, all of the items that refer to its implementation are then presented, such as: components utilized in the composition of hardware and justification, control program developed for the microcontroller, simulations, communication software via Bluetooth and the results achieved. The conclusion of this work presents a synthesis of the results met through means of application of two lab guides, aiming to verify the efficiency in the elaboration of a project of an logical circuit starting off from initial data and basics, and also the extraction of the same in an automated form through a circuit previously implemented. In general, the results found were satisfactory, justifying the utilization of the kit in the classroom.

Keywords: Truth Table, Karnaugh Map, Boolean Expression, Teaching Kit, Digital Electronics.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CIs	Circuitos Integrados
PDIP	Encapsulamento Plástico em Linha Dupla (*Plastic Dual Inline Package*).
GND	Terra (*Ground*)
ROM	Memória apenas de leitura (*Read-Only Memory*)
SRAM	Memória de acesso estático aleatório (*Static Random Access Memory*)
CC	Corrente Contínua
PCI	Placa de Circuito Impresso
IDE	Ambiente de Desenvolvimento Integrado *Integrated Development Environment*

SUMÁRIO

1 INTRODUÇÃO

Durante muitos anos as aplicações do âmbito da eletrônica digital ficaram restritas a sistemas puramente computacionais. Atualmente essa tecnologia encontra-se difundida em diversas áreas, onde evolui a cada dia e torna-se algo fundamental no mundo moderno (FLOYD, 2007).

Essa difusão se deve ao fato dos sistemas digitais apresentarem vantagens como maior confiabilidade e precisão, quando comparados aos sistemas analógicos. Justamente por isso, diversas tarefas antes feitas de forma analógica passaram a ser digitalizadas (ROTH JR; KINNEY, 2010).

Segundo Braga e Paiotti (2012) a principal diferença entre a eletrônica analógica e digital é a forma como tratam os sinais envolvidos em seu funcionamento. Na eletrônica analógica os sinais variam de forma contínua em uma escala, enquanto na eletrônica digital assumem somente valores discretos. Normalmente um sinal digital varia entre dois níveis pré-definidos, geralmente representados por dois níveis de tensão distintos.

Devido a essa característica, todos os equipamentos que possuem esse tipo de implementação obedecem um certo tipo de lógica, denominada Lógica Digital. Um dos objetivos básicos na área da eletrônica digital é o conhecimento acerca dessa lógica e dos conceitos intrínsecos à mesma. Parte desses conceitos são explorados na Álgebra Booleana, que dá a possibilidade de descrever as relações entre as entradas e saídas dos Circuitos Digitais (ou Circuitos Lógicos) como uma equação algébrica, dita Expressão Booleana (TOCCI; WIDMER; MOSS, 2011).

Muitas vezes é necessário reduzir uma Expressão Booleana para uma forma mais simples, ou manipula-la para um formato mais conveniente a fim de obter uma implementação eficiente da mesma em forma de circuito. Uma das ferramentas de simplificação utilizada é o Mapa de Karnaugh, que consiste num arranjo de células, onde cada uma representa um valor binário das variáveis de entrada, e a forma mais simples das expressões lógicas é obtida pelo agrupamento visual dessas células (FLOYD, 2007).

No que se diz respeito ao ensino destes conceitos em sala de aula, geralmente na parte inicial do conteúdo que os discentes possuem uma maior dificuldade: obtenção da tabela da verdade, transcrição da tabela da verdade para o mapa de Karnaugh e simplificação do mesmo.

De acordo com Mlsna e Liszewski (2005), os alunos tendem a cometer erros como: entrada incorreta da tabela da verdade no mapa de Karnaugh, agrupamentos ilegais ou redundantes, e até mesmo, a obtenção de uma expressão Booleana incorreta a partir de um mapa correto.

As dificuldades citadas ocorrem pelo fato dos conceitos trabalhados serem por muitas vezes abstratos, sendo de difícil absorção por parte dos alunos, o que acaba dificulta a aprendizagem dos conteúdos subsequentes, que possuem como base toda essa parte introdutória. Um recurso educacional digital que permite a interação dos discentes com esses conceitos introdutórios tem grande apelo no processo de aprendizagem, tornando o conteúdo apresentado mais

dinâmico (MLSNA; LISZEWSKI, 2005).

Segundo Sedra e Smith (2007), o uso de práticas embasadas na simulação de circuitos eletrônicos vem sendo cada vez mais difundido com o passar dos anos. Além da parte prática realizada em computadores, o uso de um kit didático com integração entre *hardware* e *software* é uma alternativa para o envolvimento do aluno diretamente com o problema que está sendo proposto, além da versatilidade de permitir diferentes possibilidades de funcionamento de acordo com a implementação do mesmo.

Consideradas então as abordagens mencionadas com relação ao uso da tecnologia no processo de ensino e aprendizagem, desenvolveu-se com este trabalho um equipamento que visa a junção de *hardware* e *software* em um único sistema, com o objetivo de facilitar e tornar mais versátil o desenvolvimento de um projeto envolvendo eletrônica digital. O intuito da ferramenta é auxiliar o aluno desde a parte inicial do projeto, que contempla a obtenção da tabela da verdade, a simplificação da expressão Booleana e a resolução do mapa de Karnaugh, até a parte de montagem do circuito, onde seu teste de funcionamento é efetuado de maneira automática ao mesmo tempo em que é possível identificar todas as etapas envolvidas em sua implementação.

2 JUSTIFICATIVA E OBJETIVOS

A justificativa e os objetivos deste trabalho são apresentados neste capítulo. Os objetivos estão divididos em Objetivo Geral e Objetivos Específicos, onde contempla-se desde a bibliografia necessária para consolidação de uma base para o desenvolvimento do projeto, até a parte prática a ser desenvolvida para alcance dos resultados desejados.

2.1 JUSTIFICATIVA

Nas aulas práticas da disciplina de Circuitos Digitais, Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná - Campus Campo Mourão, é utilizado o Módulo Universal 2000, da Datapool Eletrônica Ltda (DATAPOOL, 2015). Neste recurso educacional, o aluno realiza a montagem do circuito lógico após o desenvolvimento do projeto de forma manual, sendo as interconexões entre os CI's referentes as portas lógicas realizadas por meio de fios utilizados como *jumpers* em uma protoboard embutida. Além disso, os testes de funcionamento também são feitos de forma manual, a partir de chaves seletoras contidas no próprio equipamento.

Todo esse processo, em conjunto com a dificuldade dos alunos nos preceitos básicos, torna a complexidade dos passos envolvidos na realização de um projeto envolvendo a Eletrônica Digital mais elevada do que realmente se espera. Por esta razão, um recurso didático que atue na facilitação do ensino e aprendizagem dessa parte, ao passo que também auxilie em sua implementação prática, é de interesse para a comunidade envolvida, docentes e discentes.

Desenvolveu-se um *software* que auxilia na obtenção dos itens relacionados aos conceitos iniciais de lógica digital, tais como a obtenção da tabela da verdade, sintetização da mesma em um mapa de Karnaugh e a simplificação de expressões lógicas (MUNARINI *et al.*, 2014). O aplicativo apresenta de forma interativa todos os itens supracitados, como por exemplo, a disposição dos agrupamentos celulares do mapa de Karnaugh de forma destacada, e simultaneamente, os mintermos correspondentes aos mesmos, conforme Figura 1.

A dinâmica fornecida pelo aplicativo possibilita um grande diferencial nas disciplinas que abordam os conceitos de Circuitos Digitais, utilizada como complemento ao conteúdo, tornando o desenvolvimento de projetos mais simples, além de reduzir as chances de erros no seu decorrer.

Tendo isso como base, a implementação de um novo kit possuindo interfaceamento com um programa se torna muito vantajoso, pois abre a possibilidade do processo ser feito via *software* (MUNARINI *et al.*, 2014), e posteriormente, implementado no próprio *hardware* disponível, onde testa-se o mesmo afim de verificar seu funcionamento. Além disso, também é proporcionado ao discente a possibilidade de verificar a influência da modificação de um parâmetro original do projeto diretamente no circuito prático obtido.

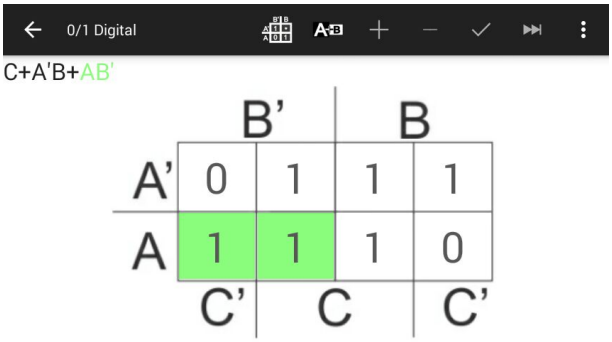

Figura 1 – Exemplo: mapa de Karnaugh
Fonte: (MUNARINI *et al.*, 2014).

Com o desenvolvimento do recurso educacional proposto, sua utilização durante as aulas proporcionará além do ganho de desempenho na aprendizagem dos alunos, uma maior produtividade na resolução dos projetos, pois converterá o processo até então manual para um processo semi-automatizado, com ferramentas úteis e não acessíveis aos alunos atualmente.

2.2 OBJETIVO GERAL

Desenvolver uma placa para auxiliar no processo ensino-aprendizagem de Eletrônica Digital, onde a partir da integração da mesma com um *software* via comunicação *bluetooth* ou Serial, obtém-se a possibilidade de realizar todo o projeto do circuito e posteriores testes em uma única plataforma. O conjunto geral ainda visa possibilitar o caminho inverso, ou seja, retirar os dados básicos do projeto com o circuito previamente implementado.

2.3 OBJETIVOS ESPECÍFICOS

O desenvolvimento do projeto contempla alguns objetivos específicos, onde cada um deles representa uma parcela para alcançar o resultado desejado citado Objetivo Geral. Os objetivos específicos são:

- Pesquisar artigos e trabalhos correlatos ao tema proposto;
- Realizar um estudo sobre o funcionamento e implementação da comunicação Serial e *bluetooth*;
- Estabelecer os materiais e métodos necessários;
- Desenvolver o projeto do *hardware*;
- Realizar simulações para constatar o funcionamento do *hardware*;

- Desenvolver o código do microcontrolador para o controle do módulo;
- Implementar e otimizar o *software*;
- Desenvolver um protótipo do *hardware*;
- Realizar o interfaceamento entre *software* e *hardware*;
- Redigir testes no protótipo e realizar possíveis otimizações de código;
- Comparar a eficácia do projeto desenvolvido em relação aos kits convencionais utilizados em sala de aula, através de roteiros de laboratório.

3 FUNDAMENTAÇÃO TEÓRICA

Para obter embasamento suficiente para o desenvolvimento do kit didático para ensino de Eletrônica Digital, necessita-se compreender os elementos básicos acerca da Álgebra Booleana e os conceitos intrínsecos a mesma. Esse processo é essencial para a integração entre o protótipo e o *software* desenvolvido em Munarini *et al.* (2014), que utiliza como base o algoritmo de Quine-McCluskey.

Além disso, é de suma importância a compreensão dos padrões de comunicação Serial e *bluetooth*, tal como seu funcionamento, leitura de dados via *software* e sua implementação em *hardware*. Em contrapartida, também é necessário o estudo de microcontroladores que atendam o propósito do trabalho.

Todos os itens necessários citados acima são tratados no decorrer deste capítulo.

3.1 ÁLGEBRA BOOLEANA E SEUS CONCEITOS

Os conceitos básicos de Eletrônica Digital são os circuitos e sistemas que operam em apenas dois níveis distintos, e por esta razão, podem ser representados com a utilização do sistema numérico binário. Floyd (2007, p. 22) confirma o que foi dito anteriormente no seguinte trecho:

> A eletrônica digital envolve circuitos e sistemas nos quais existem apenas dois estados possíveis. Esses estados são representados por dois níveis de tensão diferentes: um ALTO e um BAIXO. Os dois estados também podem ser representados por níveis de corrente, bits e ressaltos num CD ou DVD, etc. Em sistemas digitais tais como computadores, as combinações de dois estados, denominadas códigos, são usadas para representar números, símbolos, caracteres alfabéticos e outros tipos de informações. O sistema de numeração de dois estados é denominado de binário e os seus dois dígitos são 0 e 1. Um dígito binário é denominado de bit.

Cada circuito digital obedece a um determinado conjunto de regras lógicas, sendo por esta razão também chamados de circuitos lógicos. Devido a característica de possuírem intervalos de operação pré-definidos, tais circuitos podem ser descritos e analisados com o uso da álgebra booleana, que é utilizada desde a simplificação do mesmo, até a obtenção de uma relação entrada/saída na forma de uma expressão (TOCCI; WIDMER; MOSS, 2011).

Segundo Idoeta e Capuano (2012), é na álgebra booleana que se encontram todos os fundamentos da Eletrônica Digital. Ela é composta por variáveis representadas como letras, assumindo valores distintos (0 ou 1) de acordo com a definição do sistema binário. A combinação dessas variáveis gera uma sentença matemática, denominada expressão booleana, onde o circuito é sintetizado somente pelo arranjo das mesmas.

A construção das expressões geradas pela álgebra booleana é realizada pela disposição das variáveis em conjunto com circuitos padronizados, denominados portas lógicas. Idoeta e Capuano (2012, p. 41) confirmam este fato no trecho: “Através da utilização conveniente dessas portas, pode-se ‘implementar’ todas as expressões geradas pela álgebra de Boole [...]”. Na Figura 2 é feita a relação entre as portas lógicas básicas, com seu nome, simbologia, função lógica e sua expressão correspondente. Também é mostrado na Figura a existência dos CIs de portas lógicas da família 74XX e seus correspondentes, utilizados na montagem prática de circuitos digitais.

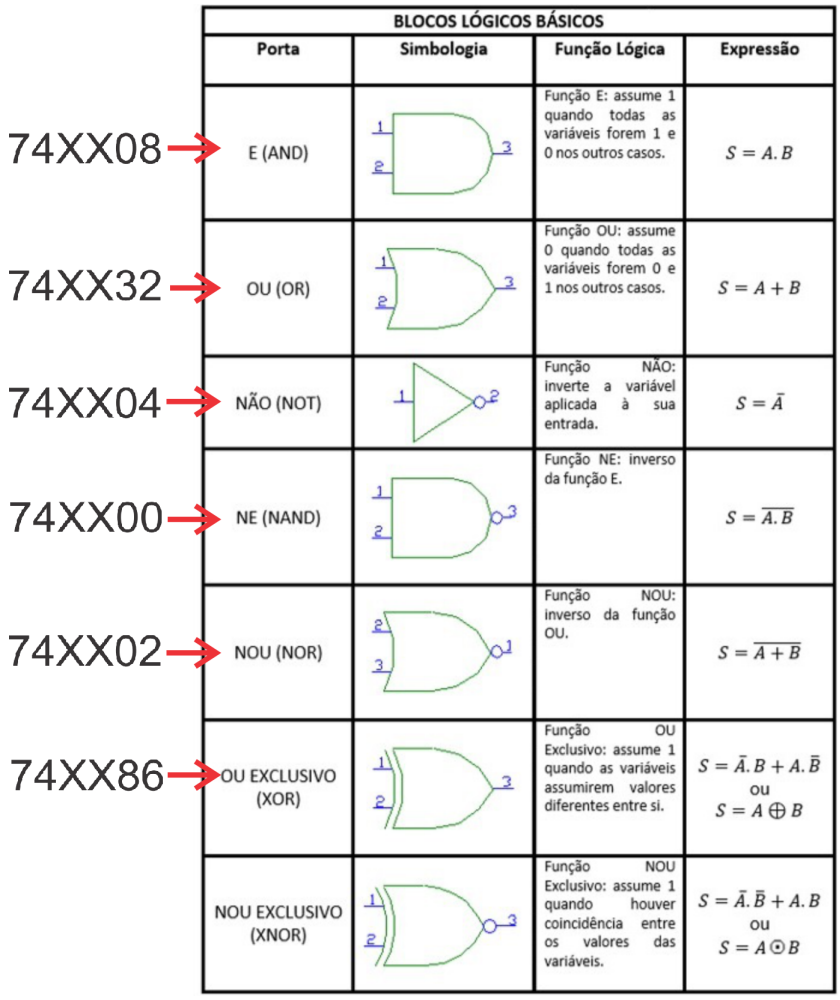

Figura 2 – Portas Lógicas básicas
Fonte: adaptado de (CUNHA, 2014a, p. 15).

Uma das maneiras de descrever a dependência dos níveis lógicos contidos na entrada de um circuito com sua saída é a tabela da verdade, e essa relação é feita através de todas as combinações possíveis entre os valores assumidos por ela. A partir da tabela da verdade é possível a sintetização da expressão booleana, seja por mintermos (utilizando saídas referentes a nível lógico alto) ou maxtermos (referentes a nível lógico baixo) (TOCCI; WIDMER; MOSS, 2011).

De acordo com Floyd (2007), por muitas vezes é necessário a simplificação da expressão obtida através da tabela da verdade. Pode-se utilizar para isso as relações fornecidas pela

própria álgebra booleana, ou de forma gráfica, uma técnica denominada mapa de Karnaugh, constituído por um arranjo da tabela em células, cujo agrupamento das mesmas (de forma eficiente) resulta numa expressão lógica simplificada. A título de exemplo, é mostrado na Figura 3 como é feito a transcrição da tabela da verdade no mapa de Karnaugh, para o caso de três variáveis.

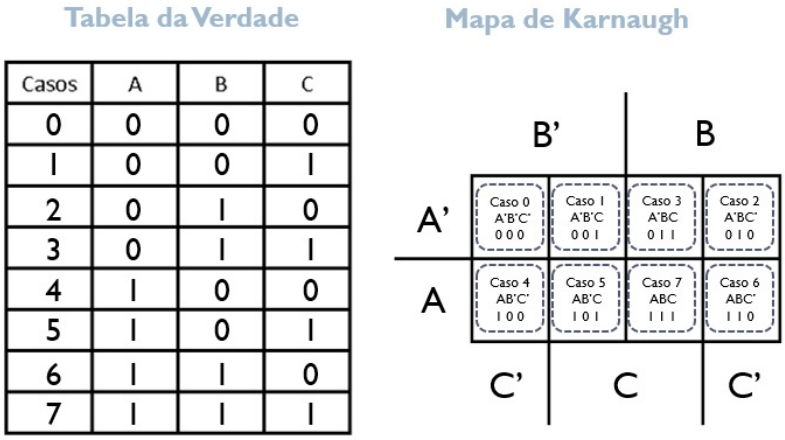

Figura 3 – Tabela da verdade x mapa de Karnaugh (3 variáveis)
Fonte: (CUNHA, 2014b, p. 23).

3.2 ALGORITMO DE QUINE-MCCLUSKEY

O Algoritmo de Quine-McCluskey, também conhecido como Método Tabular ou Método dos Implicantes Primos, foi desenvolvido por Willard Van Orman Quine (QUINE, 1952, 1955) e posteriormente aprimorado por Edward Joseph McCluskey (MCCLUSKEY, 1956).

Tal método consiste numa forma gráfica de simplificar Expressões Booleanas, porém, ao contrário do mapa de Karnaugh, sua abordagem é feita na forma tabular. No contexto geral, esse meio possui duas grandes vantagens em relação ao mapa: a primeira é o fato de uma menor dependência da habilidade do indivíduo para reconhecer os padrões necessários na simplificação, e a segunda, é que ao contrário do Karnaugh que é limitado praticamente a seis variáveis pela sua abordagem visual, pode-se trabalhar com um número muito maior das mesmas (NELSON *et al.*, 1995).

Esse método permite sua implementação em um meio computacional de forma muito eficiente, o que não seria possível com o desenvolvimento de um algoritmo que segue a lógica utilizada no mapa de Karnaugh, que é visual (ROTH JR; KINNEY, 2010).

Ainda conforme Roth Jr e Kinney (2010), o algoritmo consiste de duas tarefas básicas: a geração dos implicantes primos (mintermos que não podem ser combinados à outros) que podem vir a constituir a expressão simplificada, e a escolha do menor subconjunto dos mesmos.

De acordo com o projeto (MUNARINI *et al.*, 2014), a entrada do usuário no aplicativo

era traduzida para um binário correspondente a saída da tabela da verdade, seja ela qual fosse, e então era aplicado o algoritmo de Quine-McCluskey, seguindo os passos:

1. Geração dos implicantes primos através de um método interno à implementação do algoritmo;

2. Arranjo dos implicantes primos encontrados em uma forma tabular, conforme exemplo da Figura 4 (que será utilizada como exemplo para todos passos do algorítimo), onde são marcadas com um "X" as colunas correspondentes aos agrupamentos utilizados para concepção dos mesmos;

Implicantes Primos	Mintermos							
	0	4	8	10	11	12	13	15
AB'D'			X	X				
AB'C				X	X			
ABC'						X	X	
ACD					X			X
ABD							X	X
C'D'	X	X	X			X		

Figura 4 – Tabela dos implicantes primos
Fonte: Autoria própria.

3. Redução da tabela obtida através das consecutivas iterações, até a mesma ser zerada:

    a) Remoção dos implicantes primos essenciais e das colunas envolvidas por eles, incluídos posteriormente na expressão simplificada. Esses implicantes são os que possuem uma marcação exclusiva na coluna da tabela, o que pode ser observado no termo *C'D'* da Figura 4, e seu processo de eliminação é mostrado na Figura 5;

Implicantes Primos	Mintermos							
	0	4	8	10	11	12	13	15
AB'D'			X	X				
AB'C				X	X			
ABC'						X	X	
ACD					X			X
ABD							X	X
C'D'	X	X	X			X		

Implicantes Primos	Mintermos			
	10	11	13	15
AB'D'	X			
AB'C	X	X		
ABC'			X	
ACD		X		X
ABD			X	X

Figura 5 – Remoção dos implicantes primos essenciais
Fonte: Autoria própria.

    b) Exclusão das linhas dominadas, ou seja, existem outras na tabela com marcações suficientes para torná-las redundantes na resposta, conforme visto na Figura 6;

    c) Remoção de colunas dominantes, que análogo as linhas, tem marcações suficientes para tornar outras redundantes. No exemplo adotado, essa

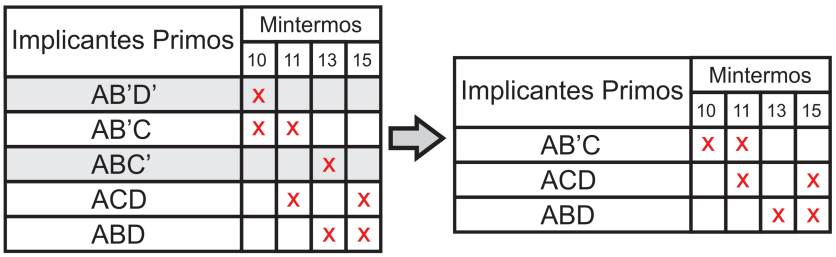

Figura 6 – Remoção das linhas dominadas
Fonte: Autoria própria.

iteração não foi necessária, o que mostra que as iterações podem variar de acordo com cada caso de entrada;

Observa-se, que no exemplo da Figura 4, a tabela não foi zerada com apenas uma iteração dos itens (a), (b) e (c), sendo necessário uma nova recursão para zerá-la e obter a resposta. A nova iteração do item (a) é mostrada na Figura 7,

Implicantes Primos	Mintermos			
	10	11	13	15
AB'C	X	X		
ACD		X		X
ABD			X	X

Figura 7 – Iteração final
Fonte: Autoria própria.

4. Há casos em que se utilizadas somente as iterações acima não é possível a obtenção da expressão simplificada, ou seja, a tabela dos implicantes primos não pôde ser zerada mesmo com todas as iterações. Sendo assim, escolhe-se o implicante que foi mais vezes combinado (com mais marcações na tabela) e o integra a resposta, retornando após isso para o passo 3;

5. Tradução da expressão encontrada para o usuário com a utilização das variáveis booleanas e portas lógicas, conforme Figura 2, *AND*, *OR* e *NOT*. No casp dp exemplo dado na Figura 4, após a realização do passo 3, a expressão final será definida por *AB'C+ABD+C'D'*, que são justamente os implicantes primos essenciais retirados e inseridos na resposta pelas iterações mostradas na Figura 5, 6 e 7.

3.3 MICROCONTROLADORES

O *hardware* objetivado necessita de um controle dos dados enviados e recebidos, além do gerenciamento de todos recursos disponíveis. Para esse controle, segundo Nicolosi (2007),

utiliza-se um microcontrolador, que são dispositivos eletrônicos que tem como propósito a execução de uma tarefa pré-determinada gravada em sua memória de código (*ROM*). A figura 8 mostra o diagrama de blocos de um microcontrolador genérico.

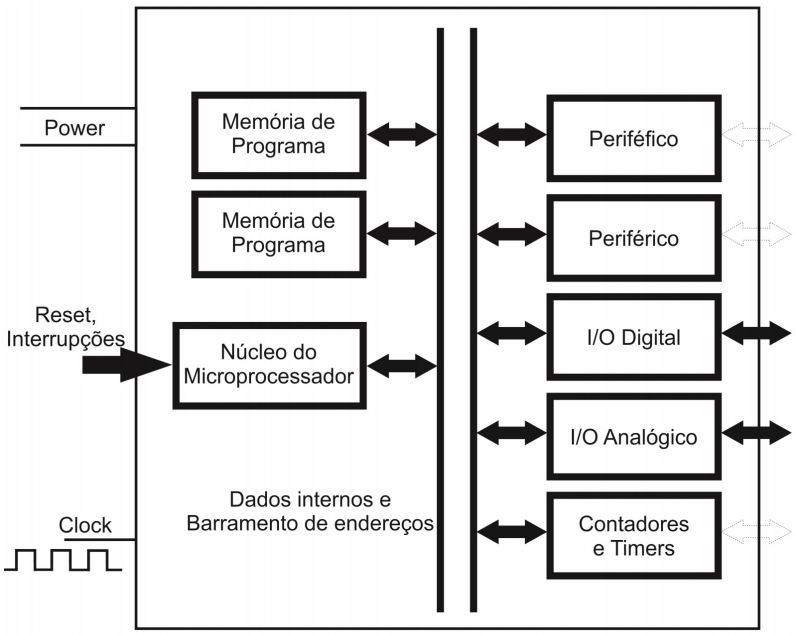

Figura 8 – Diagrama de blocos de um microcontrolador genérico
Fonte: adaptado de (WILMSHURST, 2010).

Atualmente os microcontroladores são encontrados nas mais diversas áreas de atuação, como por exemplo, automação de setores antes manuais, eletrodomésticos e telecomunicações (MICALOSKI, 2012). Devido a esta variedade, são necessários diferentes tipos de microcontroladores para atender à demanda, tendo isso acarretado o surgimento de diferentes fabricantes dos mesmos.

Tem-se então que há milhares de microcontroladores diferentes no mercado, feitos pelos mais diversos fabricantes. Um determinado fabricante desenvolve um núcleo de um microprocessador, observado na Figura 8, e a partir dele cria diferentes famílias de microcontroladores, entre elas variações como combinações de periféricos e tamanhos diferentes de memória (WILMSHURST, 2010).

Devido ao escopo do projeto conter 5 entradas, que representam as variáveis A, B, C, D e E tradadas pelo programa didático (MUNARINI *et al.*, 2014), além de 8 saídas, verificou-se a necessidade da utilização de um microcontrolador com pinos e memoria suficiente para o controle do fluxo de dados constante entre o *software* e o *hardware*.

O microcontrolador escolhido foi o Atmega32A da *Atmel*, Figura 9, que contém 40 pinos em sua versão PDIP, sendo 32 deles disponíveis para programação. Possui também uma Memória Flash de 32 Kbytes, 2 Kbytes de SRAM, e é capaz de executar 16 Milhões de operações por segundo à um *clock* de 16 MHz.

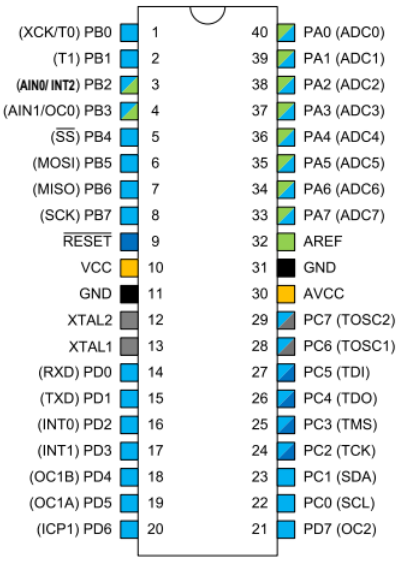

Figura 9 – Diagrama de pinos do Atmega32A PDIP
Fonte: (ATMEL, 2016).

Esse microcontrolador se torna uma escolha ideal, pois além de condizer com o número de entradas/saídas minímas do projeto, ainda pode ser programado com a utilização a IDE do Arduino, bastando incluir em sua biblioteca os arquivos de configuração dos pinos (CONNER, 2016). Isso possibilita o uso de um amplo número de bibliotecas oficiais disponibilizadas no ambiente de desenvolvimento do *Arduino*, o que otimiza o processo de programação do chip utilizado.

4 ESCOPO DO PROJETO

Neste capítulo apresentam-se algumas considerações sobre os recursos disponibilizados pelo *hardware* e *software* após sua finalização, além dos programas utilizados para confecção e realização das simulações dos mesmos.

4.1 ELEMENTOS DO *HARDWARE*

O *hardware* foi composto em sua versão final dos seguintes subelementos:

- *Protoboard* principal: assim como no Módulo Universal 2000 da Datapool Eletrônica Ltda (DATAPOOL, 2015), o *hardware* disponibilizou uma *protoboard* central para conexão dos CIs da família 74XX, porém com o diferencial de já possuir alimentação interna embutida e identificação imediata do componente conectado pelo *software*. Acoplado a *protoboard* foram disponibilizados 14 soquetes ZIF, Figura 10, com o intuito de facilitar a inserção desses CIs para montagem A *protoboard* utilizada no projeto conta com 840 furos, número suficiente para o almejado;

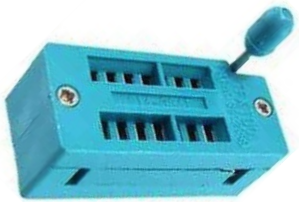

Figura 10 – Soquete ZIF 14 pinos
Fonte: (ELECTRONICS, 2016).

- Soquete de identificação de erros: Foi disponibilizado um soquete ZIF para a verificação de todos os pinos do CI inserido, a fim de identificar possíveis erros e isolá-los, garantindo a reutilização do mesmo caso não esteja totalmente prejudicado para então prevenir falhas provenientes de tal ocorrência;

- 5 entradas Lógicas: Foram disponibilizadas 5 entradas lógicas que representam as variáveis A, B, C, D e E, atualmente tratadas com sucesso pelo programa base ao *software* posteriormente implementado (MUNARINI *et al.*, 2014). Também foram implementadas as versões negadas dessas saídas: A', B', C', D' e E';

- 8 saídas lógicas: O kit oferece 8 saídas lógicas para verificação do comportamento do circuito lógico montado na *protoboard* principal. Esse número foi escolhido pois

garante a implementação de circuitos mais complexos, derivados de projetos mais elaborados;

- Gerador de *Clock*: Um gerador de *clock* com as frequência de 0,1Hz, 1Hz e 10Hz foi implementado para o uso em projetos posteriores que envolvem circuitos síncronos, o que permite a sequência do trabalho aqui desenvolvido;
- Comunicação Serial e *Bluetooth*: Um circuito para comunicação serial e *bluetooth* foi disponibilizado a partir do uso de módulos já fabricados, além da funcionalidade de comutação automáticas entre os dois meios;
- Circuito regulador de tensão para fonte externa com proteções: Juntamente ao *hardware* foi implementado um circuito para o uso de fontes externas de tensão CC, com as devidas proteções contra inversão de polaridade e curto circuito;
- Interface de programação: Acoplado ao *hardware* foi adicionado um montante de pinos para programação direta do microcontrolador, sem a necessidade da retirada do mesmo do módulo.

O programa para elaboração do esquemático do *hardware* foi o *Eagle*. Sua escolha foi feita devido a simplicidade de utilização e o amplo número de bibliotecas disponíveis em sua comunidade ativa de usuários.

Um protótipo foi desenvolvido contendo todos os circuitos mencionados, e seu funcionamento foi avaliado por meio de roteiros de laboratório com o intuito de extrair o máximo de seus recursos.

4.2 ELEMENTOS DO *SOFTWARE*

No presente trabalho, embora o *hardware* desenvolvido aceite tanto comandos via serial quanto *bluetooth*, somente foi implementado a versão do *software* para *Android*. Essa escolha foi feita devido ao aplicativo na plataforma *Android* ser de maior complexidade de desenvolvimento em conjunto com a comunicação *bluetooth*, ao mesmo tempo em que a partir dela é possível ter um parâmetro de como implementar o programa para *Windows*, já que o linguagem base para ambas é o *Java*.

Utilizou-se para o desenvolvimento do *software* a IDE *Android Studio*, da própria detentora do sistema operacional, a *Google*.

4.3 SIMULAÇÃO

Para a simulação do *hardware* obtido utilizou-se o *software Proteus*, pois o mesmo já contém em suas bibliotecas nativas todos os meios necessários para os testes circuito total, além de uma interface de comunicação virtual da serial fiel à real, que garante a verificação de funcionamento da comunicação entre as partes sem a necessidade da montagem física para tal.

5 DESENVOLVIMENTO DO HARDWARE

Apresentam-se neste capítulo, o desenvolvimento dos circuitos eletrônicos necessários para a implementação do *hardware* completo, bem como seus componentes e métodos de proteção, além da exemplificação de seu funcionamento e cálculos necessários para sua confecção.

O esquemático completo do circuito e seus *layouts* para confecção da PCI encontram-se no Apêndice C, e a lista de todo material necessário para confecção do projeto juntamente com sua estimativa de custos no Apêndice B.

5.1 CIRCUITO DO MICROCONTROLADOR

Como mencionado, o microcontrolador escolhido foi o Atmega32A da Atmel. A Figura 11 mostra o esquema utilizado para sua ligação, tal como sua distribuição de pinos entre todos os outros circuitos contidos no projeto.

Foi utilizado um cristal oscilador de 16 MHz para garantir a máxima operação do microcontrolador, e junto a ele foram ligados dois capacitores cerâmicos de C2 e C3 de 22 pF, conforme orientações do *datasheet* (ATMEL, 2016). No pino de *reset* (pino 9) foi inserido um resistor *pull-up* de 10 kΩ, prevenindo-o de *resetar* randomicamente devido a ruídos que venham a ocorrer.

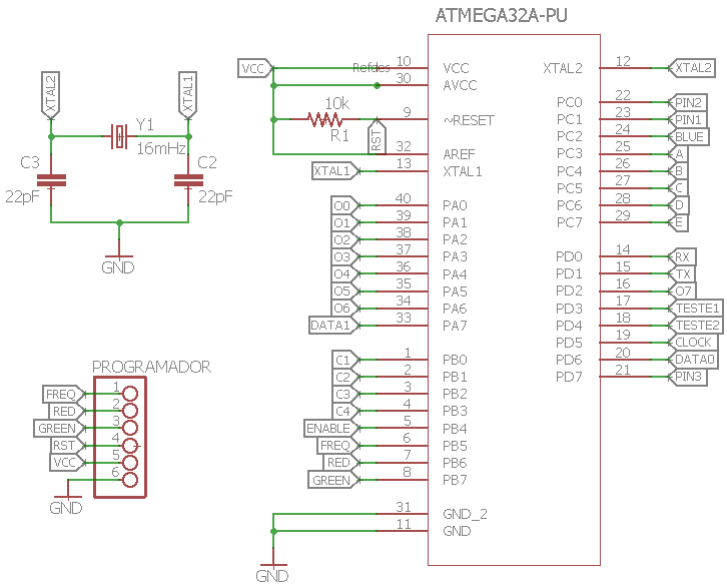

Figura 11 – Circuito do microcontrolador Atmega32A
Fonte: Autoria própria.

Adicionou-se também uma barra de pinos fêmea aos terminais programáveis do chip, para que depois de montado o *hardware* final disponibilize de uma forma fácil de programação, sem a necessidade de retirar o CI da placa para tal.

5.2 CIRCUITO DE ALIMENTAÇÃO

O circuito elaborado para alimentação da placa está representado na Figura 12. Seu componente principal é o CI LM7805, de encapsulamento TO-220, capaz de oferecer uma tensão regulada de 5 V com uma corrente de até 1,5 A, partindo de uma tensão contínua mínima de 7 V à uma máxima de 35 V (TEXAS, 2016). A entrada do circuito pelo DC *power jack* (DCJ0202) representada no esquemático deve ser de no mínimo 7 V, pois o regulador possui um *dropout* de 2 V, valor que ele consome para regular a saída com eficiência.

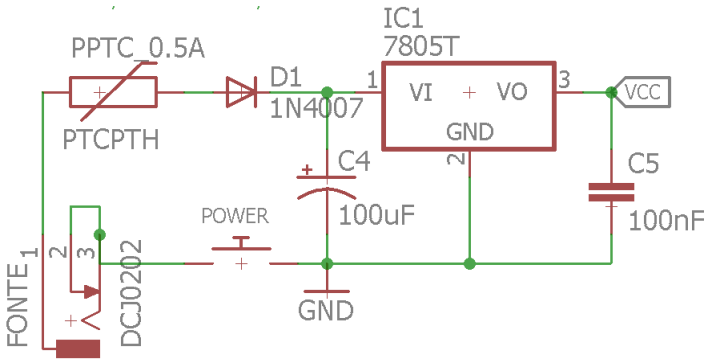

Figura 12 – Circuito de alimentação
Fonte: Autoria própria.

Apesar da tensão máxima de entrada LM7805 ser de 35 V, é interessante utilizá-lo sem um dissipador de calor, a fim de otimizar o tamanho final na placa. O cálculo é feito levando em consideração o gráfico da Figura 13, que indica a não necessidade do uso de um dissipador de calor (*heat sink*) até uma potência de aproximadamente 2 W à uma temperatura ambiente de 75°C. Faz-se então as seguintes considerações:

- A potência dissipada pelo LM7805 é dada por: $P = (V_{in} - V_{out}) * I$, onde V_{in} é a tensão de entrada, V_{out} a de saída (5 V) e I a corrente. Então:

$$V_{in} = \frac{P}{I} + V_{out} \tag{5.1}$$

- Definiu-se uma corrente máxima total de todo o circuito principal em 800 mA, pela soma de todos as correntes calculadas no decorrer do capítulo. Levou-se em consideração uma corrente máxima de 200 mA para o microcontrolador de acordo com o *datasheet* (ATMEL, 2016) e uma corrente máxima de operação de 15 mA para cada CI conetado aos soquetes em pleno funcionamento;

- Considerou-se uma temperatura ambiente de 40°C, então a potência dissipada naturalmente sem um dissipador é de aproximadamente 2,4 W, de acordo com Texas (2016).

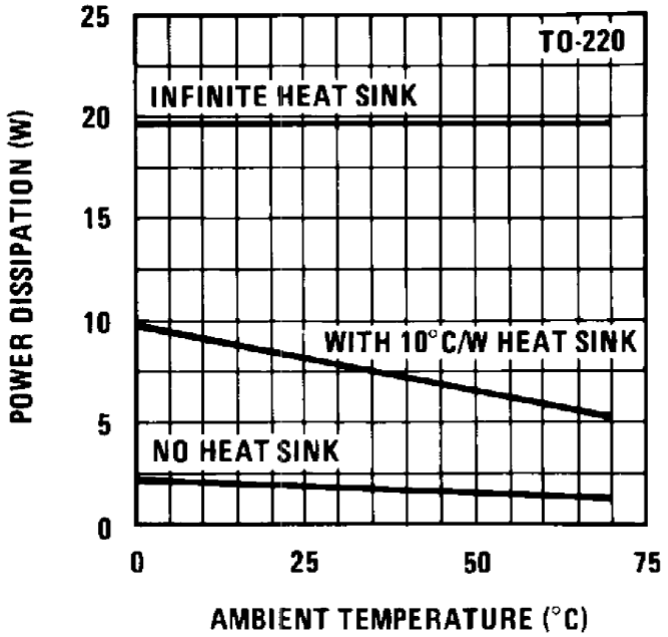

Figura 13 – Potência dissipada vs Temperatura Ambiente no LM7805
Fonte: (TEXAS, 2016).

- Substituindo as informações na Equação 5.1:

$$V_{in} = \frac{2.4W}{800mA} + 5V = 8V \tag{5.2}$$

Visto isso, é possível utilizar uma tensão de entrada de 8 V sem a necessidade de um dissipador, à uma corrente de 800 mA. Utilizou-se então o Plano de Terra (*Ground Plane*) do circuito como dissipador, além de considerar que os cálculos foram feitos com os máximos absolutos, e uma queda de tensão de aproximadamente 0,7 V existente no diodo. Pode-se utilizar então como alimentação de todo *hardware* uma fonte externa de tensão contínua de 7 à 9 V, com corrente recomendada de 1 A.

Os Capacitores C4 e C5 possuem o papel de filtrar os ruídos e as variações da fonte, enquanto o diodo D1 serve de proteção contra inversão de polaridade, o que garante a alimentação de todo circuito principal com a polaridade correta. O circuito disponibiliza também de um Fusível Rearmável PPTC, onde em caso de curto entre +5 V e o GND, uma sobrecorrente é gerada e o mesmo desarma, protegendo o circuito principal contra danos.

Por fim, o circuito de alimentação contém um botão interruptor, representado no esquemático da Figura 12 por POWER. O botão foi inserido entre o terra da fonte de alimentação externa e o de todo circuito principal, desconectando ou conectando o mesmo de acordo com sua posição.

5.3 COMUNICAÇÃO SERIAL/*BLUETOOTH*

Como estipulou-se no escopo que a comunicação será feita no *Windows* via Serial e no *Android* via bluetooth, o circuito da Figura 14 foi desenvolvido.

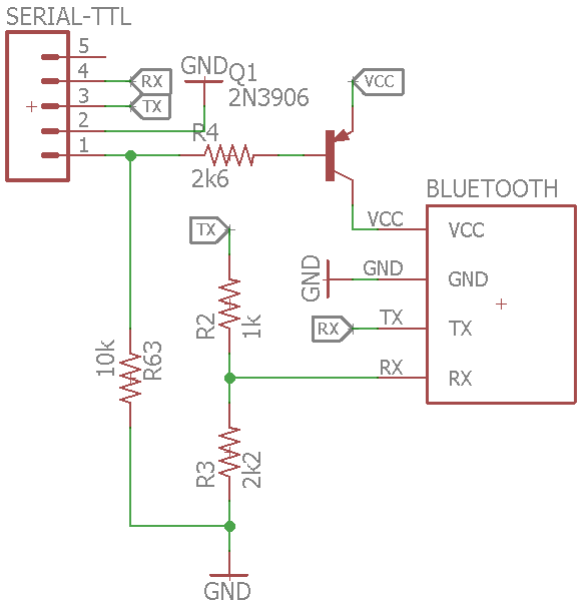

Figura 14 – Circuito comutador Serial/Bluetooth
Fonte: Autoria própria.

5.3.1 Comunicação serial

A comunicação com o computador é feita através do conversor USB para Serial PL-2303HX da *Prolific Technology Inc.* (PROLIFIC, 2013), que converte os sinais da porta USB do computador para TTL Serial, com dois níveis de referência: +5 V para o nível lógico alto e 0 V para o baixo. Esse tipo de sinal é compatível com a comunicação UART (Transmissão/Recepção Universal Assíncrona) utilizada pelo Atmega32A (ATMEL, 2016), que se baseia na transmissão de um *bit* de cada vez à uma taxa de transmissão (*baud rate*) específico.

O conversor encontra-se representado na Figura 14 por SERIAL-TTL, e os sinais são enviados pelo mesmo por meio do pino TX e recebidos pelo RX.

5.3.2 Comunicação Bluetooth

Para a efetivação da comunicação *bluetooth*, utilizou-se o módulo HC-06 da *Guangzhou HC* (Guangzhou HC, 2011). Assim como no conversor USB para Serial, o módulo se comunica com a utilização de sinais TTL, porém o nível lógico recomendado utilizado por seu receptor

RX é de 3,3 V, o que faz necessário o uso de um divisor de tensão para torná-lo compatível com o microcontrolador, que utiliza +5 V como referência superior. O cálculo dos resistores R2 e R3 da Figura 14 é feito da seguinte forma:

$$V_{RX_b} = \frac{R3}{R2 + R3} \cdot V_{TX_m} \tag{5.3}$$

Onde V_{RX_b} é a tensão do pino RX do módulo *bluetooth* HC-06 que deve ser +3,3 V, e V_{TX_m} a tensão do pino TX do Atmega32A que é de +5 V. Levando em consideração estes dados, e a utilização de um resistor R2 fixo de 1 kΩ, a Equação 5.3 se dá por:

$$3,3V = \frac{R3}{1k\Omega + R3} \cdot 5V \tag{5.4}$$

Resultando no valor de R3:

$$R3 = 1.9412k\Omega \tag{5.5}$$

O valor comercial mais próximo para o resistor calculado é de 2 kΩ, porém devido a tolerância citada no *datasheet* (Guangzhou HC, 2011) de até 3,5 V, pode-se utilizar um resistor de até 2,3 kΩ. O valor utilizado no projeto é de 2.2 kΩ.

5.3.3 Comutador Serial/Bluetooth

Definido que a comunicação do projeto com o *software* será feita por *bluetooth*, ou via Serial, implementou-se um circuito para evitar o uso dos dois dispositivos simultaneamente, comutando-os de maneira automática. Para isso utilizou-se um transistor PNP ligado entre o VCC (+5 V) do adaptador serial e do módulo *bluetooth*. Nesse tipo de transistor, necessita-se que a tensão da base seja menor que a do emissor em aproximadamente 0,7 V para seu saturamento.

Assim que o serial é conectado ao computador, uma corrente de aproximadamente 1,7 mA chega ao transistor Q1 por meio de um resistor R4 de 2,6 kΩ, e a condição dita anteriormente não é atendida, o que faz com que somente a comunicação Serial seja feita com o microcontrolador por meio dos pinos RX/TX, desconectando o módulo *bluetooth*.

Ao desconectar o conversor USB-Serial do computador, o resistor R63 com a configuração de *pull-down* é ativado, e a base do transistor Q1 é ligada ao Terra por meio da soma dos resistores R4 e R63, totalizando 12,6 kΩ. Isso faz com que a tensão da base seja menor que a do emissor, como já dito, e o módulo *bluetooth* é ligado à alimentação principal automaticamente, alimentado por aproximadamente 4,3 V devido à queda de 0,7 V da junção, o que fica dentro da sua faixa de tolerância para alimentação.

Visto o funcionamento, o transistor escolhido foi o 2N3906, pois além de ter um baixo consumo (625 mW), suporta um tensão Coletor-Emissor de 40 V e uma corrente de coletor de

200 mA (ON, 2010). Como o módulo consome somente 8 mA em total comunicação e 30 à 40 mA durante o pareamento (Guangzhou HC, 2011), os parâmetros do transistor são satisfatórios.(Guangzhou HC, 2011)

5.4 ENTRADAS LÓGICAS

Para a alimentação das entradas dos circuitos lógicos (A, B, C, D e E) foi utilizado o esquema da Figura 15 e Figura 16. As saídas digitais 25 à 29 do microcontrolador, Figura 11, são conectadas individualmente a uma barra de pinos fêmea por meio de um resistor de 1 kΩ. Esses resistores tem a função de limitar a corrente drenada do Atmega32A em no máximo 5 mA, protegendo-o de um curto entre os terminais e prevenindo possíveis danos, além de garantir um limite de corrente para os circuitos integrados da família 74xx conectados as portas do *chip*.

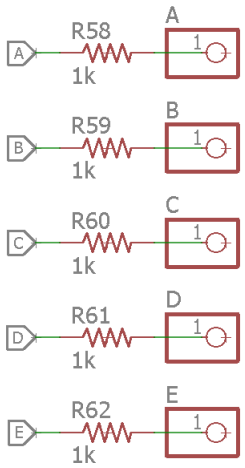

Figura 15 – Circuito das Variáveis de Entrada
Fonte: Autoria própria.

5.4.1 Entradas negadas

Além das entradas normais, também foram implementadas suas versões negadas. A base de funcionamento é uma porta inversora (NOT), Figura 16, feita com um transistor NPN 2N3904 (ON, 2012), que possui as mesmas características elétricas do 2N3904 citado na Subseção 5.3.3, porém funciona de maneira inversa.

Quando as saídas do microcontrolador estão em nível lógico baixo (0 V), não há tensão presente na base dos transistores T1 à T5 e eles não são saturados. As saídas A', B', C', D' e E' são então alimentadas pelos resistores de 1 kΩ, com uma corrente de 5 mA, entrando assim em nível lógico alto.

Porém, quando as saídas do Atmega32A são setadas em nível lógico baixo, a base do

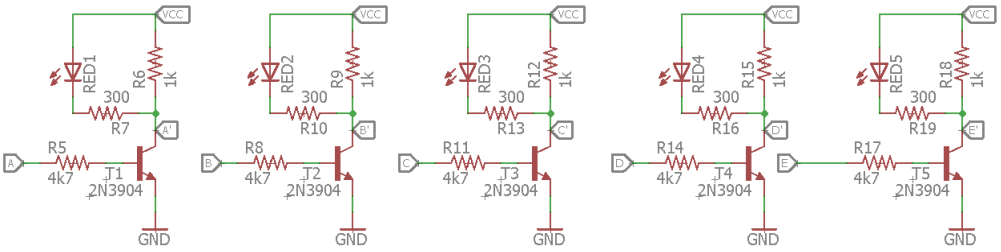

Figura 16 – Circuito das Variáveis de Entrada Negadas
Fonte: Autoria própria.

transistor recebe uma corrente de aproximadamente 1 mA, saturando-o. Desse modo a corrente Coletor-Emissor flui diretamente para o terra colocando-as em nível baixo.

5.4.2 Leds Indicadores

Para a indicação das entradas foram utilizados LEDS vermelho, RED1 a RED5 na Figura 15. O cálculos dos resistores para os LEDS foram baseados na Tabela 1.

Tabela 1 – Cores dos LEDS vs Queda de Tensão

Cor do LED	Queda de Tensão
Vermelho	1.6V
Verde	2.0V
Amarelo	2.0V
Azul	3.2V

Fonte: Autoria própria.

Como o LED utilizado é vermelho, sua queda de tensão V_{LED} é de aproximadamente 1,6 V, e só será acionado quando as entradas negadas estiverem em nível lógico baixo, conectadas diretamente ao terra. Para uma corrente I de 10 mA e considerando a queda de tensão Coletor-Emissor V_{CE} do transistor em 0,3 V, de acordo com o *datasheet* (ON, 2012), o cálculo do resistor é feito da seguinte forma:

$$R_{LED} = \frac{VCC - V_{LED} - V_{CE}}{I} \tag{5.6}$$

Ficando:

$$R_{LED} = \frac{5V - 1,6V - 0,2V}{10mA} = 320\Omega \tag{5.7}$$

No projeto foi utilizado o valor comercial imediatamente abaixo para o resistor, 300Ω, resultando em uma corrente de aproximadamente 10,7mA, dentro dos padrões aceitáveis de até 20 mA para os LEDS de 5 mm.

5.5 SAÍDAS LÓGICAS

O escopo do projeto especifica 8 saídas lógicas para a conexão dos circuitos integrados da família 78XX. São representadas na Figura 17 pelos *labels* O0 à O7 e ligadas diretamente ao microcontrolador Atmega32A, Figura 11.

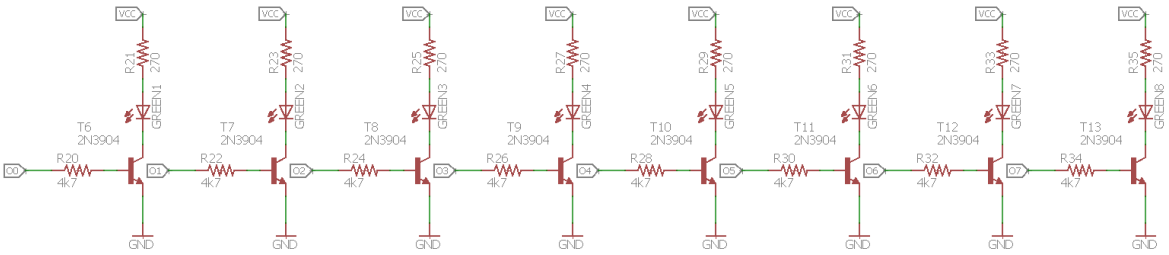

Figura 17 – Circuito das Variáveis de Saída
Fonte: Autoria própria.

Paralelo às conexões ao microcontrolador, foram inseridos LEDS indicadores verdes controlados por transistores NPN 2N3904, assim como referido na subseção 5.4. Os transistores são acionados quando um nível lógico alto é conectado a qualquer uma das saídas, saturando-o com uma corrente de base de aproximadamente 1 mA e efetuando a conexão do terra ao cátodo do LED.

Por ser verde, o LED possui uma queda de tensão aproximada de 2 V segundo a Tabela 1. Utilizou-se as mesmas considerações feitas para a elaboração do circuito de entrada, então:

$$R_{LED} = \frac{5V - 2V - 0.2V}{10mA} = 280\Omega \tag{5.8}$$

Para o projeto foi utilizado um resistor de 270 Ω, que é o valor comercial imediatamente abaixo do R_{LED} calculado na Equação 5.8. Com esse valor obtêm-se uma corrente de aproximadamente 10,4 mA, ficando dentro dos limites aceitáveis do LED como já citado na subseção anterior.

5.6 CIRCUITO DETECTOR DE CIS

Para a implementação do circuito detector dos CIs inseridos, utilizou-se o princípio mostrado na Figura 18, onde observa-se que em todos os circuitos integrados, a porta lógica intrínseca pode ser obtida pela leitura dos pinos 8, 9 e 10.

O projeto em seu escopo disponibiliza 14 soquetes ZIF de 14 pinos para leitura dos circuitos lógicos, e por esse motivo torna-se inviável a conexão de cada um individualmente ao microcontrolador. Como alternativa a tal, foi implementado um circuito de comutação composto por 3 multiplexadores/demultiplexadores 74HC4067 de 16 canais, Figura 19, ligados aos

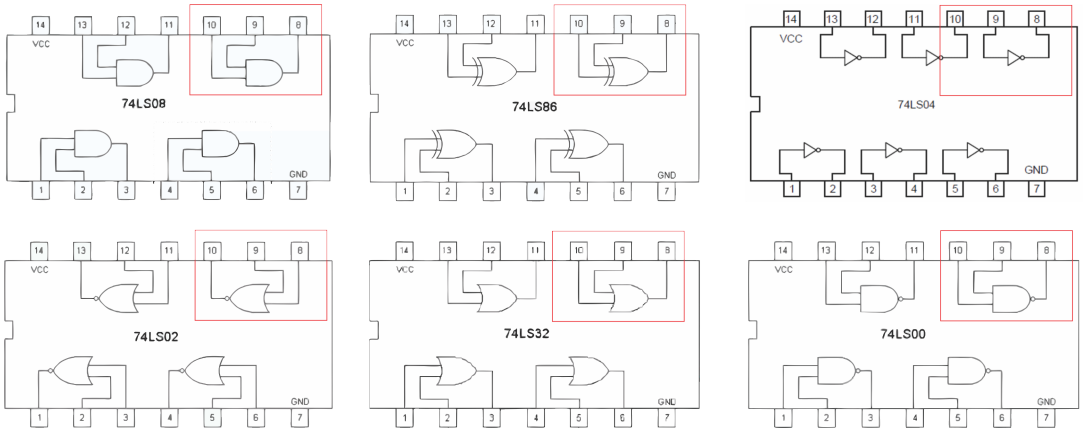

Figura 18 – Princípio utilizado para detecção dos CIs
Fonte: Adaptado de (CUNHA, 2014b).

três últimos pinos de todos os soquetes. A escolha desse componente se deu pelo falo de sua comunicação ser bidirecional, permitindo assim setar as entradas e ler sua saída, conforme desejado (NXP, 2015).

A conexão é feita da seguinte forma: os pinos 10 são representados pelos *labels* A0 à A13, os pinos 9 por B0 à B13 e os pinos 8 por S0 à S13. A Figura traz também o exemplo de ligação de um CI 7408 ao primeiro soquete (A0, B0 e S0), tal como seus pinos correspondentes.

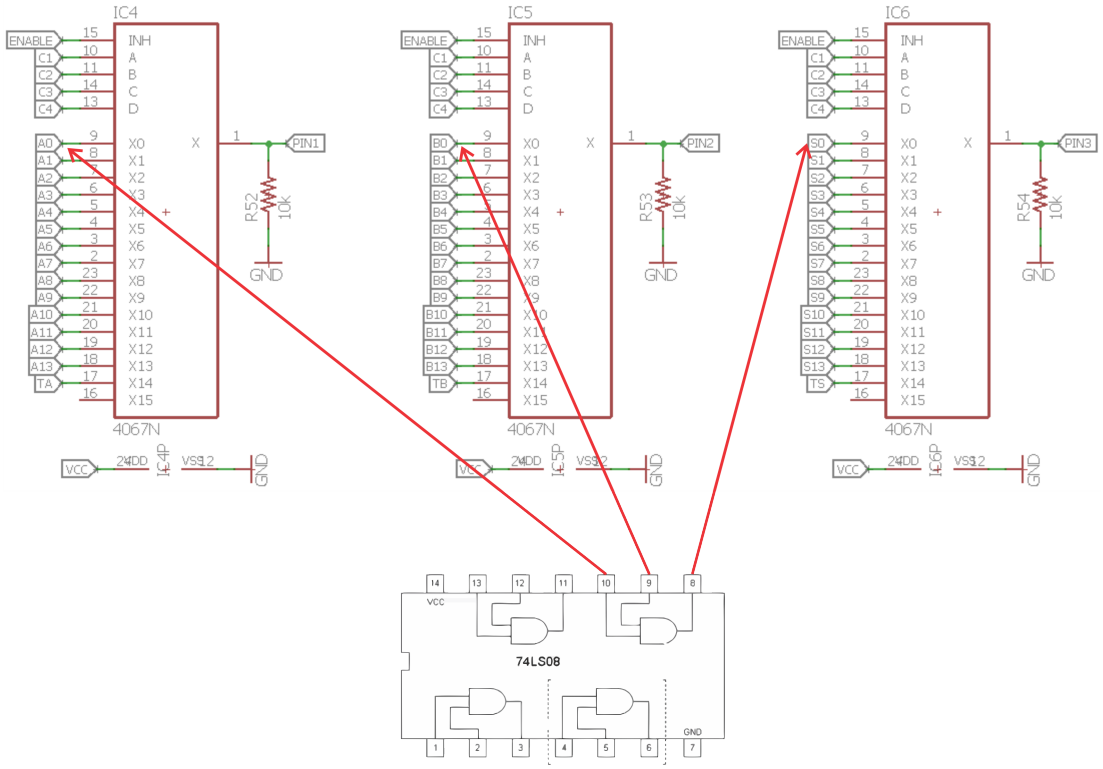

Figura 19 – Circuito identificador dos CIs 74XX
Fonte: Adaptado de (CUNHA, 2014b).

5.6.1 Identificação e certificação contra erros

Para identificação do CI conectado à um dos soquetes, o microcontrolador faz uso de uma Tabela Verdade de duas variáveis enviada e lida por seus pinos 21, 22 e 23, Figura 11. No exemplo da Figura 19, a tabela é enviada para o CI 74LS08 através dos pino 22 pelo IC4 e do pino 23 pelo IC5, lida pelo pino 21 do microcontrolador através do IC6. Com essa operação, o Atmega32A consegue pela variação dos níveis lógicos nas entradas da porta lógica contida no chip conectado so soquete, obter uma saída correspondente e identificá-lo.

Porém, como a operação é feita para cada um dos 14 soquetes individualmente, é necessário desconectar os não utilizados no momento a fim de prevenir erros. Tal erro pode ocorrer quando uma porta com nível lógico alto, 5 V, é conectada à outra que está sendo testada no momento com um nível lógico baixo, 0 V, já que a montagem dos circuitos lógicos se dá pela combinação de um ou mais CIS 74XX conectados ao módulo principal do projeto.

Com o fim de contornar os erros mencionados, foi desenvolvido o circuito da Figura 20. Composto de dois *shift-register* 74HC164 de 8 bits, com entrada serial e saída paralela. O intuito desse esquema é controlar cada soquete individualmente por meio de sua alimentação VCC, através do chaveamento do transistor 2N3906 já mencionado na subseção 5.3.3.

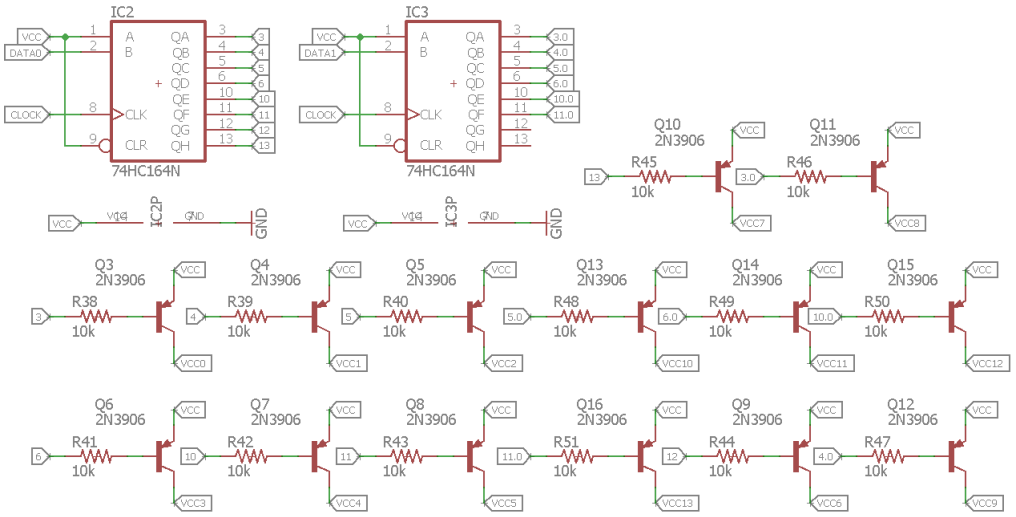

Figura 20 – Circuito comutador de alimentação dos soquetes
Fonte: Autoria Própria.

O circuito comuta a alimentação de cada *slot* através de duas entradas serial binária de 8 bits enviada pelo microcontrolador nas portas 2 dos registradores de deslocamento. Exemplo: caso o Atmega32A envie o *byte* 0111111 para DATA0 e 11111111 para DATA1, somente o soquete de número 0 conectado ao registrador IC2 por meio do transistor Q3 da Figura 20 é ligado à alimentação diretamente, mantendo os outros desligados. O componente IC2 fica responsável pelos primeiro 8 soquetes, enquanto o IC3 pelos 6 restantes.

Isso só é possível pois cada *slot* disponível é conectado intrinsecamente ao GND por

seu pino 7, enquanto o VCC se conecta ao circuito comutador, como é mostrado no exemplo para a primeira conexão, Figura 21.

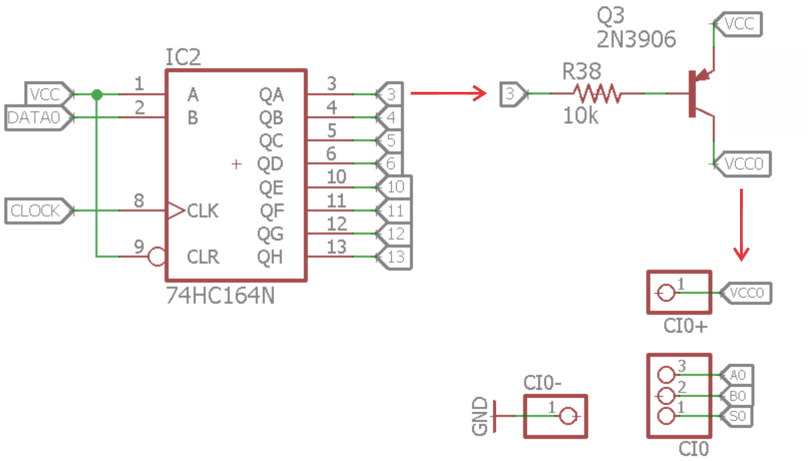

Figura 21 – Exemplo de ligação de um soquete ao *shift-register*
Fonte: Autoria Própria.

Os resistores de 10 kΩ conectados aos pinos 1 dos *mux/demux* IC4, IC5 e IC6, e os pinos 21, 22 e 23 do microcontrolador tem o efeito de *pull-down*, onde liga-se à saída do Atmega diretamente ao GND quando não há CIs conectados aos *sockets* ou o multiplexador é desligado, o que evita a flutuação da entrada e previne possíveis erros.

5.7 SOQUETE PARA DETECÇÃO DE ERROS PORTA A PORTA

Além do módulo principal, o projeto conta também um soquete capaz de identificar erros em todas as portas lógicas contidas dentro de um único CI da família 74. Para isso, utilizou-se de multiplexadores/demultiplexadores analogamente ao circuito de detecção, porém o 74HC4052, que dois canais de 4 bits dentro de um único componente.

O princípio de funcionamento é o mesmo do módulo principal com 14 variantes, porém em vez de conectar somente os 3 últimos, conecta-se todos os pinos de acesso às portas lógicas ao multiplexador bidirecional. Cada entrada e saída é conectada ao microcontrolador por meio dos pinos 17 dos componentes IC4, IC5 e IC6, Figura 19, e a identificação de erros é feita pelo mesmo princípio citado na subseção 5.6.1, porém expandida para todas as portas lógicas presentes.

O circuito, tal como o soquete e suas ligações são identificados na Figura 22.

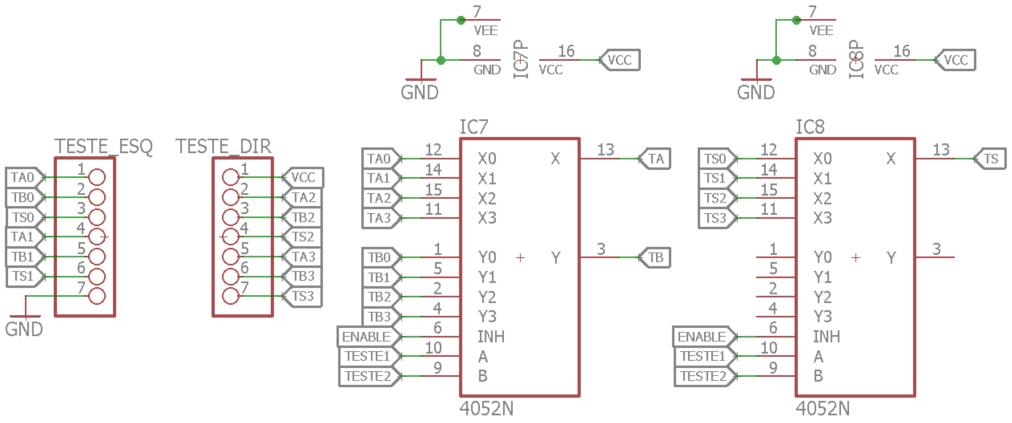

Figura 22 – Circuito do soquete de verificação de Erros porta a porta
Fonte: Autoria Própria.

5.8 GERADOR DE *CLOCK*

Com a finalidade de permitir o uso de circuitos digitais síncronos no projeto desenvolvido, foi implementado um circuito Gerador de *Clock*, mostrado na Figura 23. A saída 6, denominada FREQ no Atmega32A é conectada à base de um transistor 2N3906, e quando é saturado por um nível lógico baixo vindo do mesmo, VCC é conectado diretamente ao pino CLK da placa. O efeito de comutação entre nível baixo e alto gerado pelo microcontrolador em intervalos específicos gera o *clock* com a frequência desejada. .

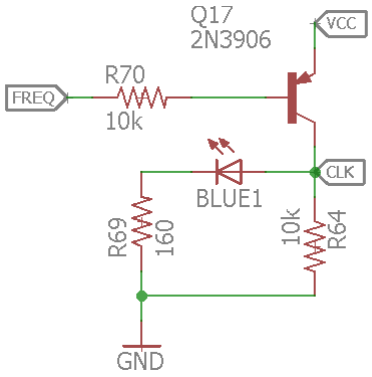

Figura 23 – Circuito gerador de *clock*
Fonte: Autoria Própria.

O resistor para o LED indicador azul acoplado ao circuito é calculado análogo à Equação 5.6, seguindo a queda de tensão 3,2 V descrita na Tabela 1, ficando:

$$R_{LED} = \frac{5V - 3,2V - 0,2V}{10mA} = 160\Omega \tag{5.9}$$

5.9 INDICADOR DE ESTADO

Para a identificação visual do estado das operações feitas pelo *hardware* montado foi implementado o circuito da Figura 24. Ele é composto de um LED 5MM RGB, conetado ao microcontrolador por meio de três transistores 2N3906, e possui o mesmo comportando ja mencionado nos tópicos anteriores. O LED RGB é composto das cores vermelho, verde e azul, e como sua configuração de alimentação é exatamente a mesma das utilizadas nos outros circuitos que compõem o módulo, não houve a necessidade de novos cálculos para os resistores.

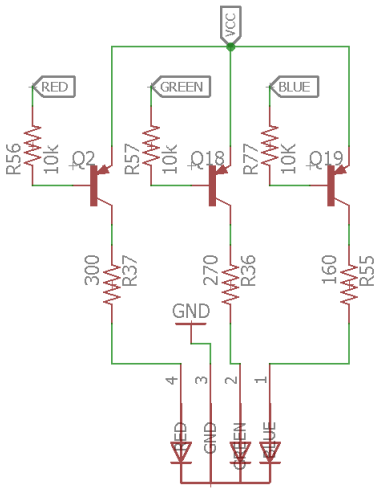

Figura 24 – Circuito indicador de estado
Fonte: Autoria Própria.

O circuito é utilizado da seguinte forma:

- Cor vermelha: quando não há conexão entre o *hardware* e o *software*, seja ela *bluetooth* ou serial, ou há erros de funcionamento por algum motivo.
- Cor verde: O *hardware* está conectado com o *software* e funciona corretamente.
- Cor azul: O *hardware* está ocupado ou processando alguma informação. A cor volta para verde assim que a operação acaba.

6 CONFIGURAÇÃO DO MICROCONTROLADOR ATMEGA32A

Apresentam-se neste capítulo todos os métodos utilizados pelo microcontrolador para o controle dos circuitos implementados, tal como sua forma de comunicação com o *software* via tratamento dos comando enviados e recebidos pelo formato de comunicação UART.

6.1 CONFIGURAÇÃO INICIAL

O primeiro passo seguido para a configuração do microcontrolador conforme o circuito desenvolvido no Capitulo 5, Figura11, é a definição de seus pinos. Para isso necessita-se a desabilitação do circuito interno JTAG, que quando habilitado não permite a utilização dos pinos 24, 25, 26 e 27 como entradas ou saídas digitais.

Segundo o *datasheet*, o microcontrolador Atmega32A possui no registrador MCUCSR um *bit* específico para a desabilitação do JTAG, o JTD, que quando setado com o valor 1, causa o efeito desejado (ATMEL, 2016). Para tal, adiciona-se à parte de configuração inicial o comando visto na Figura 25, que é duplicado pois toma 4 ciclos de *clock* para sua efetivação.

```
MCUCSR = (1<<JTD);
MCUCSR = (1<<JTD);
```

Figura 25 – Trecho de código para desabilitação do JTAG
Fonte: Autoria Própria.

Adiciona-se também à configuração inicial o *baud rate* de operação da Serial utilizada pelo microcontrolador como 9600 bps, como forma de padronizar com o módulo *bluetooth* e a comunicação do conversor USB-Serial TTL utilizados no projeto.

Todas as configurações aqui mencionadas foram feitas na rotina *setup* dentro do código do microcontrolador, e são executadas uma única vez assim que o mesmo é ligado.

6.2 TRATAMENTO SERIAL

O tratamento dos dados recebidos pela serial do microcontrolador foi feito dentro da rotina *loop*, que ao contrário da *setup*, é executada constantemente, repetindo seus métodos conforme padrões pré-estabelecidos no código. A função interna à IDE do *Arduino*, *Serial.available*, retorna o número de *bytes* recebidos pelo Atmega em sua interface serial, utilizada como um indicador de que algum comando chegou e precisa ser tratado.

A leitura desses dados é feita pelo trecho de código da Figura 26, onde verifica-se o retorno constante da função mencionada anteriormente. Utiliza-se uma variável do tipo *String*

denominada *aux* para armazenamento temporário, e cada caractere recebido é concatenado até a função indicar que não há mais *bytes* para leitura.

```
String leSerial(){
  String aux = "";
  char caractere;

  while(Serial.available() > 0){//le enquanto receber algo pela serial
    caractere = Serial.read();
    if(caractere != '\n'){//se não acabou concatena
      aux.concat(caractere);
    }
    delay(10);//delay para dar tempo de ler
  }
  return aux;
}
```

Figura 26 – Trecho de código da leitura serial
Fonte: Autoria Própria.

Após isso, o comando em forma de *String* recebido é comparado internamente e a partir dele é executada uma determinada função. A Tabela 2 contém todos os comandos possíveis, tal como um exemplo de como ele é enviado do *software* para o microcontrolador e seu retorno depois do tratamento interno.

Tabela 2 – Tabela de comandos implementados aceitos microcontrolador

Número	Comando	Complemento	Valor do complemento	Exemplo de Envio	Exemplo de Retorno
1	TESTE	-	-	TESTE	OK
2	ERROS	-	-	ERROS	#+AND+ERRO+VAZIO+OR+∼
3	CONECTADO	-	-	CONECTADO	-
4	DESCONECTADO	-	-	DESCONECTADO	-
5	R	-	-	R	-
6	G	-	-	G	-
7	B	-	-	B	-
8	C	X	0 à 13.	C1	#0AND∼
9	ALL	-	-	ALL	#+AND+NAND+NOT+AND+XOR+NOR+OR+ NAND+AND+NOT+XOR+ERRO+VAZIO+NOT+∼
10	S	SABCDE	S: 0 à 7; A, B, C, D e E: 0 ou 1.	S011001	#0∼
11	E	NS	N: 2 à 5; S: 0 à 7.	E21	#11110000∼
12	01HZ	-	-	01HZ	-
13	1HZ	-	-	1HZ	-
14	10HZ	-	-	10HZ	-
15	OFF	-	-	OFF	-

Fonte: Autoria própria.

Como foi utilizado no projeto um método de transmissão serial assíncrona pelo microcontrolador, adiciona-se um bit de partida (*start bit*) ao início da transmissão e um bit de parada (*stop bit*) à seu fim. Essa precaução é tomada para verificação da informação recebida pelo *software*, tal como evitar possíveis erros ocasionados durante a transmissão. Como *start bit* foi utilizado o caractere '#' e como *stop bit* o '∼', o que é verificado claramente nos exemplos de retorno exibidos pela Tabela 2.

Os tempos programados para resposta do microcontrolador ao *software* foram em seu máximo absoluto 200ms, sendo extremamente rápido mesmo em seu pior caso.

6.3 COMANDOS ACEITOS E TRATADOS PELO *HARDWARE*

Nesta seção apresentam-se os comandos aceitos pelo *hardware*, Tabela 2, além de sua função e funcionamento detalhado.

6.3.1 Comando TESTE

O *software* envia a *String TESTE* para o microcontrolador a fim de verificar a conexão. Caso esteja estabelecida é retornada a palavra *OK*, tratada posteriormente pelo programa em uma rotina para indicar estabilidade.

O trecho de código responsável pelo comando se encontra na Figura 27, onde a informação recebida pelo microcontrolador, depois de tratada conforme dito na Seção 6.2, é comparada à *String TESTE*. Havendo correspondência, a função *Serial.println* retorna *OK* para o sistema solicitante.

```
if(recebido.equals("TESTE")){
  Serial.println("OK");
}
```

Figura 27 – Trecho de código do comando *TESTE*
Fonte: Autoria Própria.

Vê-se que verificação é feita de forma simples, porém eficiente, uma vez que haja conexão entre o *hardware* e o *software*.

6.3.2 Comando ERROS

Ao receber o comando *ERROS* o microcontrolador analisa o soquete de verificação de erros implementado na Seção 5.7, por meio da conexão em cascata mostrada na Figura 28.

Primeiramente a combinação binária C1 = 0, C2 = 1, C3 = 1 e C4 = 1 é feita nos seletores dos três multiplexadores 74HC4067, conectando TA, TB e TS dos 74HC4052 aos pinos digitais PIN1, PIN2 e PIN3 do Atmega32A.

Faz-se então a combinação da Tabela 3 nos seletores TESTE1 e TESTE2 dos dois multiplexadores bidirecionais 4052, conectando 3 pinos a cada momento no microcontrolador.

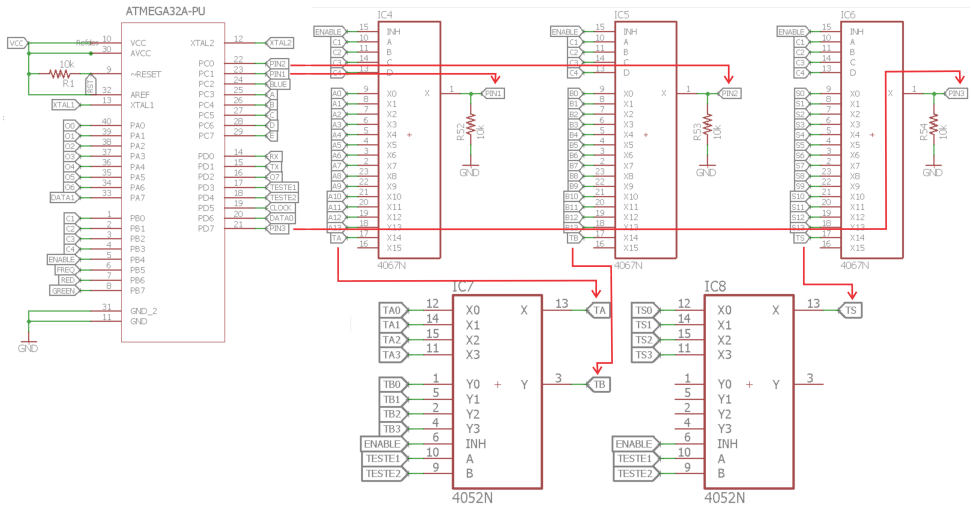

Figura 28 – Conexão do soquete de verificação de erros ao microcontrolador
Fonte: Autoria Própria.

Tabela 3 – Combinação dos seletores do 74HC4052 vs Pinos do Atmega32A

TESTE1	TESTE2	PIN1	PIN2	PIN3
0	0	TA0	TB0	TS0
0	1	TA1	TB1	TS1
1	0	TA2	TB2	TS2
1	1	TA3	TB3	TS3

Fonte: Autoria própria.

Isso permite a leitura individual de cada porta lógica presente no CI conectado ao soquete, conforme exemplo da Figura 29, sem o uso excessivo de entradas e saída disponíveis no Atmega.

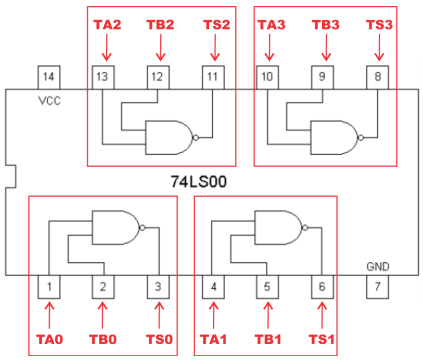

Figura 29 – Exemplo da conexão de um CI ao soquete de verificação de erros
Fonte: Autoria Própria.

A identificação da porta lógica pelo microcontrolador é feita com base na Figura 30, em que a cada combinação binária setada nos pinos PIN1 e PIN2, é gerada uma determinada saída em seu PIN3. Para o CI da porta lógica XOR (74LS86) a leitura é feita de forma inversa devido a sua configuração vista na Figura 18, com PIN2 e PIN3 entradas, enquanto PIN1 se torna saída.

O conjunto dessas combinações gera um vetor binário de saída equivalente ao comportamento único de uma determinada porta lógica, permitindo assim a distinção entre elas.

PIN1	PIN2	PIN3						
		NOT	AND	OR	NAND	XOR	NOR	VAZIO
0	0	1	0	0	1	0	1	0
0	1	0	0	1	1	1	0	0
1	0	1	0	1	1	1	0	0
1	1	0	1	1	0	0	0	0

Figura 30 – Identificação da porta lógica pelo microcontrolador
Fonte: Autoria Própria.

Caso o vetor de saída lido seja composto apenas de zeros, significa que nenhum CI está conectado no soquete e a leitura se deu pelo nível lógico setado pelos resistores *pull-down* conectados à entrada do Atmega32, onde neste caso retorna-se a *String* VAZIO. Caso não haja correspondência do que foi lido com todas as varições contidas na Figura 30, é retornada a palavra ERRO para o requisitante do comando, indicando desde uma porta com defeito até um CI erroneamente conectado.

A partir dos conceitos citados, o trecho de código da Figura 31 realiza o processo de identificação, cujo retorna pela serial uma *String* no formato: '$\# + P_0 + P_1 + P_2 + P_3 + \sim$', onde cada P_x representa uma porta lógica lida conforme Figura 30.

```
for(int j = 0; j < 4; j++){//Tabela Verdade nos ´pinos 9 e 10, saída pino 8
  digitalWrite(pino10, bitRead(j, 0));
  digitalWrite(pino9, bitRead(j, 1));
  delay(10);//espera um tempo para ler a saida
  tabela[j] = digitalRead(pino8);//lê a saída
  delay(10);
}
if(tabela[0] == 1 && tabela[1] == 1 && tabela[2] == 0 && tabela[3] == 0){//PORTA NOT (74LS04)
  enviar += "NOT";
}
else if(tabela[0] == 0 && tabela[1] == 0 && tabela[2] == 0 && tabela[3] == 1){//PORTA AND (74LS08)
  enviar += "AND";
}
else if(tabela[0] == 0 && tabela[1] == 1 && tabela[2] == 1 && tabela[3] == 1){//PORTA OR (74LS32)
  enviar += "OR";
}
else if(tabela[0] == 1 && tabela[1] == 1 && tabela[2] == 1 && tabela[3] == 0){//PORTA NAND (74LS00)
  enviar += "NAND";
}
else if(tabela[0] == 0 && tabela[1] == 1 && tabela[2] == 1 && tabela[3] == 0){//PORTA XOR (74LS86)
  enviar += "XOR";
}
else if(tabela[0] == 1 && tabela[1] == 0 && tabela[2] == 0 && tabela[3] == 0){//PORTA XOR (74LS86)
  enviar += "NOR";
}
else if(tabela[0] == 0 && tabela[1] == 0 && tabela[2] == 0 && tabela[3] == 0){//Não há CI
  enviar += "VAZIO";
}
else{//CI DESCONHECIDO
  enviar += "ERRO";
}
}
```

Figura 31 – Trecho de código responsável pela identificação de erros
Fonte: Autoria Própria.

Verifica-se que ao *software* requisitante do comando analisar às informações retornadas à ele como resposta, pode-se concluir que um CI conectado ao soquete de verificação de erros está ou não funcionando corretamente, tal como identificar e descartar a utilização de um

conjunto de pinos atrelado a uma porta lógica que foi identificada como não funcional. Para isso, basta-se analisar se as quatros variantes P_x mencionadas diferem-se entre si.

6.3.3 Comandos CONECTADO e DESCONECTADO

Os comandos *CONECTADO* e *DESCONECTADO* são utilizados pelo requisitante para setar um indicador visual no *hardware* de que a conexão foi bem ou mal sucedida. Tais comandos podem ser utilizados em conjunto com a resposta do comando *TESTE* anteriormente mencionado.

O trecho do código responsável pela tarefa é o da Figura 32, simplesmente setando nível lógico alto (*HIGH*) ou baixo (*LOW*) nos terminais RED, GREEN e BLUE da Figura 24. A função utilizada para tal é a *digitalWrite*, também interna a IDE do *Arduino*.

```
else if(recebido.equals("CONECTADO")){
  digitalWrite(RED, HIGH);
  digitalWrite(GREEN, LOW);
  digitalWrite(BLUE, HIGH);
}
else if(recebido.equals("DESCONECTADO")){
  digitalWrite(RED, LOW);
  digitalWrite(GREEN, HIGH);
  digitalWrite(BLUE, HIGH);
}
```

Figura 32 – Trecho de código do circuito indicador
Fonte: Autoria Própria.

Nota-se que o nível lógico da cor verde correspondente ao estado conectado, e a vermelha ao desconectado, são definidas com nível lógico baixo. Isso se deve pela lógica de acionamento do transistor PNP utilizado, com seu funcionamento descrito na Subseção 5.3.3.

6.3.4 Comandos R, G e B

Assim como para conexão e desconexão, esse comando diz respeito ao indicador visual de estado da Figura 24. É utilizado para outras indicações visuais, como diferenciação entre comunicação *bluetooth* ou serial.

Sua implementação no microcontrolador é feita pelo código da Figura 33, em que o LED da cor desejada tem seu estado atual invertido, ou seja, desligando caso esteja ligado e ligando caso desligado. Os outros LEDs não correspondentes a cor requerida são desligados com nível lógico baixo na base do transistor PNP responsável por seu controle.

```
else if(recebido.equals("R")){
  digitalWrite(RED, !digitalRead(RED));
  digitalWrite(GREEN, HIGH);
  digitalWrite(BLUE, HIGH);
}
else if(recebido.equals("G")){
  digitalWrite(RED, HIGH);
  digitalWrite(GREEN, !digitalRead(GREEN));
  digitalWrite(BLUE, HIGH);
}
else if(recebido.equals("B")){
  digitalWrite(RED, HIGH);
  digitalWrite(GREEN, HIGH);
  digitalWrite(BLUE, !digitalRead(BLUE));
}
```

Figura 33 – Trecho de código do LED RGB
Fonte: Autoria Própria.

6.3.5 Comando C e ALL

O comando *C* é utilizado para identificar individualmente algum CI conectado à algum dos 14 soquetes contidos na *protoboard*. Assim como o mostrado na Tabela 2, possui um complemento que deve ser enviado junto ao mesmo pelo *software*, com variação de C0 à C13.

A identificação ocorre de maneira análoga a dita na Subseção 6.3.2, porém só são conectados os 3 últimos pinos de cada soquete diretamente aos 74HC4067, como mostra a Figura 19. O código base para detecção dos CIs é o mesmo apresentado na subseção anterior, Figura 31, diferindo apenas nos seletores C1, C2, C3 e C4 dos multiplexadores bidirecionais e na utilização dos *shift-registers*.

Durante o processo realizado sobre um *slot*, o circuito implementado na Subseção 5.6.1 é acionado para chavear a alimentação dos CIs conectados. O chaveamento é feito pelo trecho de código da Figura 34, em que com a variação da variável de controle *i* conforme o número de soquetes, 0 à 13, a função *shiftOut* automaticamente gera o *clock* necessário para o deslocamento dos *bits*, o que permite a conexão de somente uma alimentação de cada vez conforme exemplificado durante a implementação do circuito.

```
if(i < 8){
  shiftOut(data0, CLOCK, MSBFIRST, ~(1 << (i%8)));
}
else{
  shiftOut(data1, CLOCK, MSBFIRST, ~(1 << (i%8)));
}
```

Figura 34 – Trecho de Código do comutador de alimentação
Fonte: Autoria Própria.

Com base no comando recebido pelo microcontrolador é feita a combinação binária dos seletores dos multiplexadores 74HC4067 como mostrado na Tabela 4. A partir dessa seleção, um soquete correspondente é lido e seu valor é retornado ao *software* requisitante como indicado, onde P_x representa uma *String* referente à porta lógica e seu valor se dá, como na seção anterior, de acordo com a Figura 30.

O comando *ALL* tem o funcionamento básico idêntico ao do *C*, porém são testados e identificados todos os soquetes de uma só vez, pela variação automática dos seletores apre-

Tabela 4 – Combinação dos seletores dos 74HC4067 para cada soquete.

Comando	C1	C2	C3	C4	Soquete Lido	Retorno
C0	0	0	0	0	Soquete 0	#0Px∼
C1	0	0	0	1	Soquete 1	#1Px∼
C2	0	0	1	0	Soquete 2	#2Px∼
C3	0	0	1	1	Soquete 3	#3Px∼
C4	0	1	0	0	Soquete 4	#4Px∼
C5	0	1	0	1	Soquete 5	#4Px∼
C6	0	1	1	0	Soquete 6	#6Px∼
C7	0	1	1	1	Soquete 7	#7Px∼
C8	1	0	0	0	Soquete 8	#8Px∼
C9	1	0	0	1	Soquete 9	#9Px∼
C10	1	0	1	0	Soquete 10	#10Px∼
C11	1	0	1	1	Soquete 11	#11Px∼
C12	1	1	0	0	Soquete 12	#12Px∼
C13	1	1	0	1	Soquete 13	#13Px∼
ALL	0000 à 1101				Soquete 0 à 13	#+P0+P1+P2+P3+P4+P5+P6+ P7+P8+P9+P10+P11+P12+P13+∼

Fonte: Autoria própria.

sentados na Tabela 4. Seu retorno é uma concatenação de todas as portas lógicas identificadas, também de acordo com a Figura 30.

Ao retornar uma mensagem no formato indicado é possível tratar uma rotina externa implementada no *software* para que depois de lido, seja setada convenientemente no soquete correspondente a indicação da porta lógica, se o mesmo encontra-se vazio ou no estado de erro já mencionado.

6.3.6 Comando S

Como forma de ler uma saída resultante de uma combinação de entradas, o comando *S* foi implementado. Possui um valor de complemento correspondente ao número da saída em seu segundo dígito e a combinação das variáveis booleanas de entrada A, B, C, D e E em seus últimos cinco caracteres.

O trecho de código referente à leitura do comando e tratamento da sequência de caracteres recebidos é o da Figura 35. O primeiro caractere da *String* recebida é comparado ao indicador de comando, e em caso de correspondência, são setadas as entradas requeridas pela função *digitalWrite*, convertendo-as para um número inteiro representando os binários 0 e 1.

```
else if(recebido.charAt(0) == 'S'){//caracter 0 é o comando
  //último caracteres são as variáveis booleanas de entrada
  digitalWrite(A, recebido.charAt(2)-'0');
  digitalWrite(B, recebido.charAt(3)-'0');
  digitalWrite(C, recebido.charAt(4)-'0');
  digitalWrite(D, recebido.charAt(5)-'0');
  digitalWrite(E, recebido.charAt(6)-'0');
  getSaida(recebido.charAt(1)-'0');//caracter 1 é o número da saída
}
```

Figura 35 – Trecho de código responsável pelo comando 'S'
Fonte: Autoria Própria.

Após setadas, a função *getSaida* é executada, Figura 36, consistindo simplesmente na leitura da saída recebida intrinsecamente ao comando enviado, através da função interna à IDE *digitalWrite*. É retornado ao *software* requisitante uma *String* no formato '#0∼' ou '#1∼', representando os níveis lógicos baixo e alto obtidos da saída.

```
void getSaida(int saida){
  int saidas[] = {S0, S1, S2, S3, S4, S5, S6, S7};//vetor das entradas
  enviar = "#"+(String)digitalRead(saidas[saida])+"~";//concatena no forma #0~ ou #1~
  delay(5);//delay antes de enviar
  Serial.println(enviar);//envia os dados
}
```

Figura 36 – Trecho de código da função getSaida()
Fonte: Autoria Própria.

Desse modo é possível gerar a resposta do circuito lógico implementado na área de trabalho do *hardware* a uma combinação binária individual, obtendo seu comportamento para aquele instante analisado.

6.3.7 Comando E

Diferentemente do último comando apresentado, este testa o circuito lógico implementado no *hardware* com um conjunto de combinações binárias, obtendo seu comportamento para todo o leque de possibilidades disponíveis.

A mensagem recebida possui como complemento ao comando *E* o número de variáveis booleanas de entrada, indo de 2 (A e B) até 5 (A, B, C, D e E), e a saída requisitada no momento, 0 à 7. É tratada inicialmente da mesma forma que o código da Figura 35, onde é separado o comando referente à seu complemento para a execução da função.

Encontra-se no trecho de código da Figura 37 a função responsável pelo comando recebido, em que é feito automaticamente, a partir do número de variáveis de entrada e da saída atual, o teste do circuito lógico implementado.

```
void testaCircuito(int numeroVariaveis, int saida){
  int saidas[] = {S0, S1, S2, S3, S4, S5, S6, S7};
  enviar = "#";//flag de início de transmissão

   for(int j = 0; j < pow(2, numeroVariaveis); j++){
     digitalWrite(A, bitRead(j, numeroVariaveis-1 >= 0 ? numeroVariaveis-1 : 6));
     digitalWrite(B, bitRead(j, numeroVariaveis-2 >= 0 ? numeroVariaveis-2 : 6));
     digitalWrite(C, bitRead(j, numeroVariaveis-3 >= 0 ? numeroVariaveis-3 : 6));
     digitalWrite(D, bitRead(j, numeroVariaveis-4 >= 0 ? numeroVariaveis-4 : 6));
     digitalWrite(E, bitRead(j, numeroVariaveis-5 >= 0 ? numeroVariaveis-5 : 6));
     delay(5);
     enviar += digitalRead(saidas[saida]);
    }
  enviar += "~";//flag de fim da transmissão
  delay(5);//delay antes de enviar
  Serial.println(enviar);//envia os dados
  reset();//reseta as configurações inicias
}
```

Figura 37 – Trecho de código executado pelo comando 'E'
Fonte: Autoria Própria.

Ao fim é retornada uma *String* no formato da saída de uma Tabela Verdade, pois a mesma contém todas as variações binárias de entrada. Na Figura 38 é representado um exemplo do retorno para um conjunto de testes de 3 variáveis: A, B e C, porém o mesmo esquema de funcionamento é expandido para todo o conjunto de variáveis booleanas trabalhado.

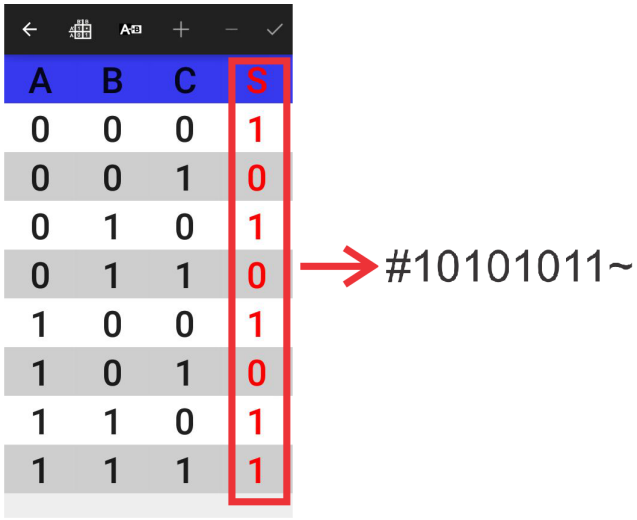

Figura 38 – Exemplo do formato da *String* trabalhada
Fonte: Autoria Própria.

6.3.8 Comandos 01HZ, 1HZ, 10HZ e OFF

Os comandos aqui apresentados dizem respeito à geração do *clock* através do circuito implementado na Seção 5.8. O trecho de código feito para o controle desse circuito encontra-se na Figura 39.

Durante o início da execução da rotina do microcontrolador a variável *freqTimer* é inicializada com a função *millis* contida na biblioteca da IDE do *Arduino*, que retorna o tempo

```
double freq = 0;//frequência em HZ (0.1, 1 ou 10), 0 = desligado
double freqInterval = 0;//intevalo da frequência, inicializado com 0
unsigned long freqTimer; //timer da frequência

void geraFrequencia(){
  if(freq != 0){
    freqInterval = ((1/(freq*0.001))/2);//atualiza intervalo da frequência com o cálculo
    digitalWrite(pinFreq, !digitalRead(pinFreq));
    freqTimer = millis();//atualiza o timer
  }
  else{//frequencia = 0, seta pino em HIGH e desliga
    digitalWrite(pinFreq, HIGH);
  }
}
```

Figura 39 – Trecho de código do gerador de *clock*
Fonte: Autoria Própria.

em milissegundos contado a partir de sua inicialização. Ao receber um comando, a variável *freq* é atualizada com os valores: 0, 0.1, 1 ou 10 de acordo com a frequência do *clock* desejada.

A cada ciclo do microcontrolador a comparação da Figura 40 é executada, acionando a função *geraFrequencia* quando o tempo definido por *freqInterval* corresponder ao tempo percorrido pelo microcontrolador. O valor da variável *freqInterval* é calculado de acordo com a Equação 6.1, onde T_{ms} é o tempo em milissegundos de metade de um período, já que o *duty cycle* utilizado é de 50%.

$$T_{ms} = \frac{\frac{1}{freq \cdot 0.001}}{2} \tag{6.1}$$

```
if((millis() - freqTimer) >= freqInterval){//gera a frequência desejada
  geraFrequencia();
}
```

Figura 40 – Comparação responsável pela geração do *clock*
Fonte: Autoria Própria.

Uma vez inicializada, a função *geraFrequencia* define o intervalo de tempo em que o *clock* ficará em nível lógico baixo ou alto com base no valor definido por *freqInterval*, além de atualizar o *timer* da frequência para a geração de um novo contador para o próximo ciclo.

Todo esse processo permite gerar um *clock* sem a necessidade da paralisação do programa principal do microcontrolador. Isso permite que as operações sejam realizadas em paralelo, o que otimiza a utilização de todos os recursos.

7 SIMULAÇÃO DO *HARDWARE*

A simulação do circuito foi realizada no *software Proteus*. Os principais circuitos do *hardware* foram implementados no mesmo para obter um parâmetro de funcionamento fiel ao comportamento real do projeto finalizado.

7.1 MICROCONTROLADOR

Como o microcontrolador utilizado, Atmega32A, já se encontra nativamente na biblioteca do *Proteus*, basta configurá-lo com um *clock* de 16MHz de acordo com as especificações do projeto, não sendo necessário adicionar um cristal oscilador para sua simulação. O microcontrolador configurado e com todas suas ligações encontra-se na Figura 41.

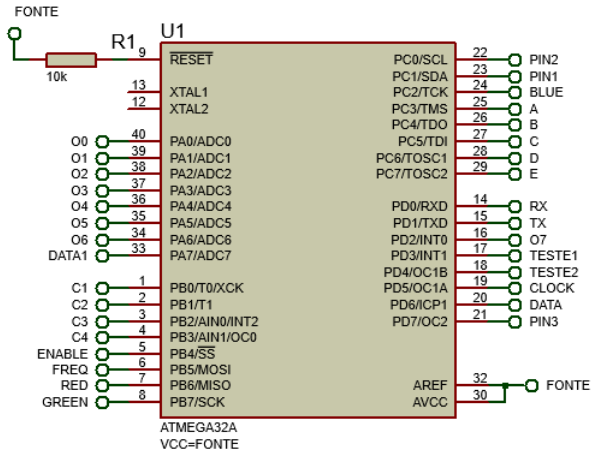

Figura 41 – Atmega32A no Proteus
Fonte: Autoria Própria.

Após sua pinagem, o código hexadecimal gerado para o microcontrolador através da IDE do *Arduino* é inserido no campo *Program File* do *software* de simulação, constituindo assim a base do funcionamento do circuito.

7.2 SOQUETES

Cada um dos soquetes 14 para conexão dos CIs foi implementado conforme a Figura 42, onde como já dito no Capitulo 5, o pino VCC é conectado a um registrador de deslocamento e os pinos 8, 9 e 10 são conectados à três multiplexadores bidirecionais, que por sua vez os conectam ao microcontrolador.

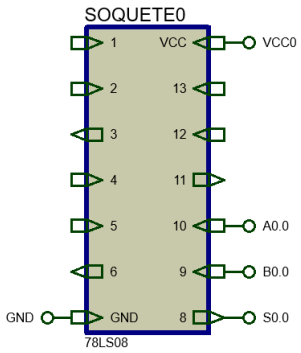

Figura 42 – Soquete no Proteus
Fonte: Autoria Própria.

O *software* de simulação de circuitos eletrônicos utilizado já contém em sua biblioteca todos os circuitos integrados da família 74XX implementados. Os CIs foram conectados ao soquete mencionado anteriormente inserido-os dentro do bloco da Figura 42, e seus pinos conectados conforme a Figura 43, para o caso de um 74LS08. Essa configuração é expandida para toda a família de CIs lógicos definida no escopo.

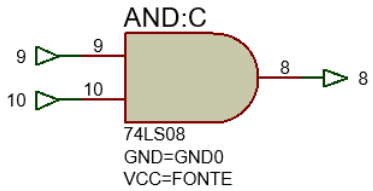

Figura 43 – Família 74XX no Proteus
Fonte: Autoria Própria.

7.3 COMUNICAÇÃO SERIAL

Para a comunicação Serial foi utilizado o bloco *COMPIM* do *Proteus*, Figura 44, que simula uma porta serial física conectada ao computador fielmente. Esse bloco foi conectado aos pinos RX e TX do microcontrolador Atmega32A implementado no programa.

7.4 SIMULAÇÃO

A simulação da comunicação foi realizada por meio de *Strings* enviadas e recebidas pelo *Serial Monitor* da IDE do *Arduino*, correspondentes ao código gravado no microcontrolador. Pode-se usar alternativamente qualquer *software* que possua o mesmo papel de comunicação pela porta Serial do computador.

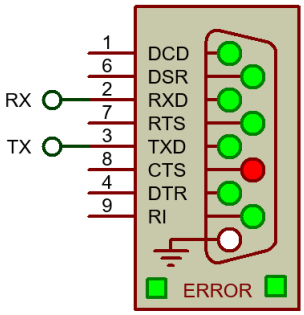

Figura 44 – BLOCO COMPIM
Fonte: Autoria Própria.

7.4.1 Circuito indicador

Após a implementação de todos os circuitos do Capítulo 5 e feitas as configurações iniciais ditas anteriormente, iniciou-se a simulação no *Proteus*. Logo de início, o circuito indicador de estado demonstrado na Seção 5.9 chaveia a cor vermelha do LED RGB, representada na Figura 45 por D1. Como ainda não foi enviado o comando de conexão para serial, o circuito demonstra comportamento esperado.

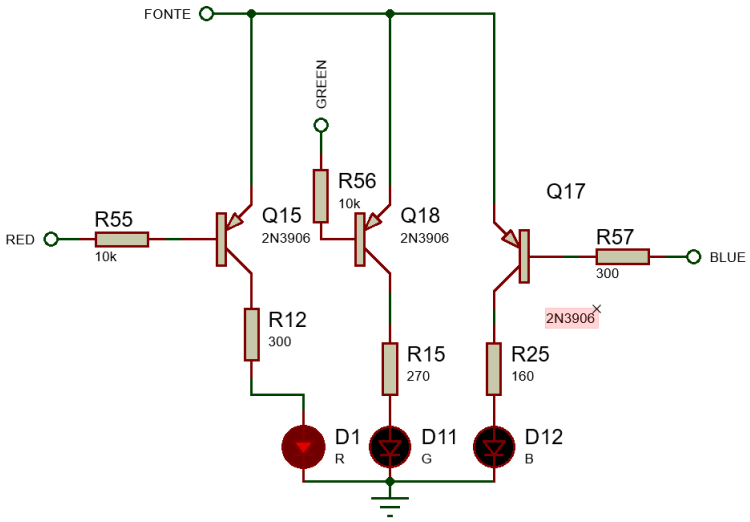

Figura 45 – Circuito indicador com status Desconectado
Fonte: Autoria Própria.

Ao envio do comando *CONECTADO*, o circuito indicador comporta-se como na Figura 46, onde a conexão serial bem sucedida com o microcontrolador é realçada.

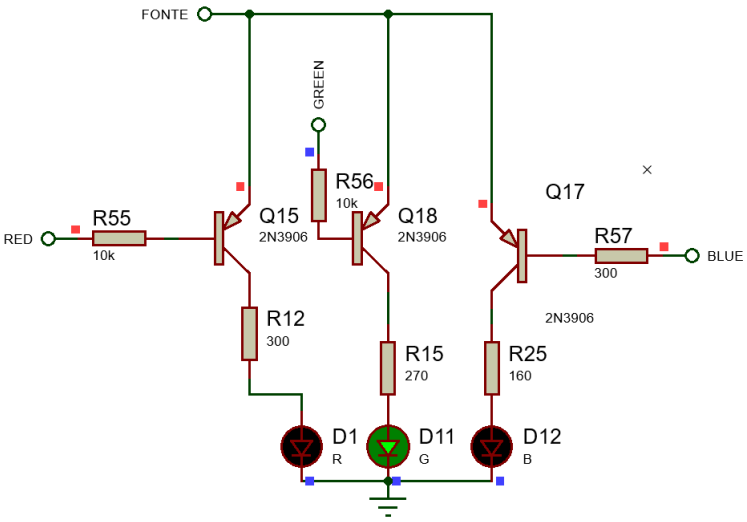

Figura 46 – Circuito indicador com status Conectado
Fonte: Autoria Própria.

7.4.2 Identificador de CIs

A simulação do circuito identificador foi feita preenchendo todos os 14 soquetes disponíveis na seguinte ordem: 74LS08, 74LS32, 74LS04, 74LS00, 74LS02, 74LS86, vazio, 74LS08, 74LS32, 74LS04, 74LS00, 74LS86, vazio, 74LS08. Foram deixados 2 *slots* vazios, e o segundo ligado inversamente para efeito sobre os testes. Após isso a *String ALL* foi enviada ao microcontrolador pelo *Serial Monitor*.

Assim que o comando é enviado, o circuito da Figura 45 adquire padrão azul, indicando que o *hardware* está ocupado processando as informações para efetuar um retorno. Após o circuito indicador voltar para verde, a mensagem da Figura 47 é exibida no *prompt*. Se comparado o que foi recebido com os CIs inseridos, observa-se que o *hardware* conseguiu não só identificá-los um a um, como também o *slot* vazio e o que foi inserido inversamente, apresentando a mensagem VAZIO e ERRO para tais casos.

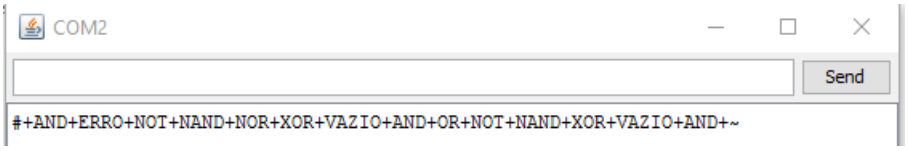

Figura 47 – Resposta ao comando *ALL*
Fonte: Autoria Própria.

Além do teste simultâneo, efetuou-se o teste dos CIs individualmente, enviando pelo mesmo processo que o anterior o comando *CX*, onde *X* é substituído pelo número do soquete a ser testado (0 à 13). Dessa operação, a mensagem da Figura 48 foi exibida, e como é equivalente à Figura 47, chega-se a mesma conclusão.

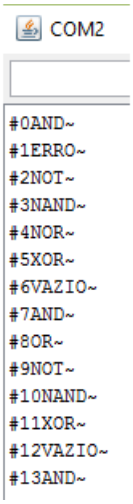

Figura 48 – Resposta ao comando *C*
Fonte: Autoria Própria.

7.4.3 Identificação de erros

Como forma de simular o soquete reservado para identificação de erros, utilizou-se o mesmo esquema da Seção 7.2, porém em seu interior foram conectados três circuitos lógicos 74LS00 (NAND), deixando os pinos 4, 5 e 6 desconectados para a identificação ser possível, como mostra a Figura 49.

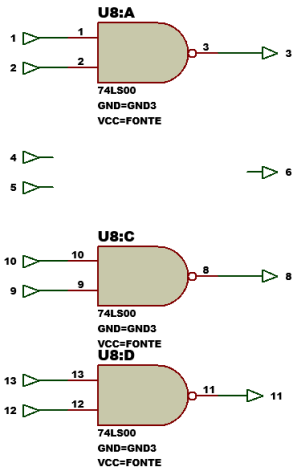

Figura 49 – Soquete utilizado para identificação de erros
Fonte: Autoria Própria.

Enviou-se então o comando *ERROS* pela serial, exibindo como resultado a resposta da Figura 50, onde nota-se equivalência ao circuito desenvolvido da Figura 49. Observa-se que onde os pinos foram corretamente conectados, o microcontrolador foi capaz de identificar corretamente o circuito lógico presente. Já no caso do pino desconectado, retornou VAZIO, indicando um eventual erro que pode ser tratado posteriormente no interior do *software*.

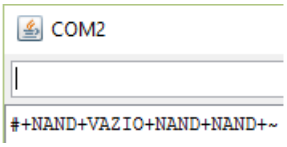

Figura 50 – Resposta ao comando *ERROS*
Fonte: Autoria Própria.

7.4.4 Leitura do circuito lógico

Uma das partes mais importantes simuladas é a extração dos dados do projeto (Tabela Verdade, Mapa de Karnaugh e Expressão Lógica) de um circuito previamente montado no módulo principal. O circuito escolhido para testes é o circuito da Figura 51, de complexidade relativamente alta. A expressão lógica extraída do mesmo, se feita de forma manual se resume a:

$$S = A' + B' + D.E + C \tag{7.1}$$

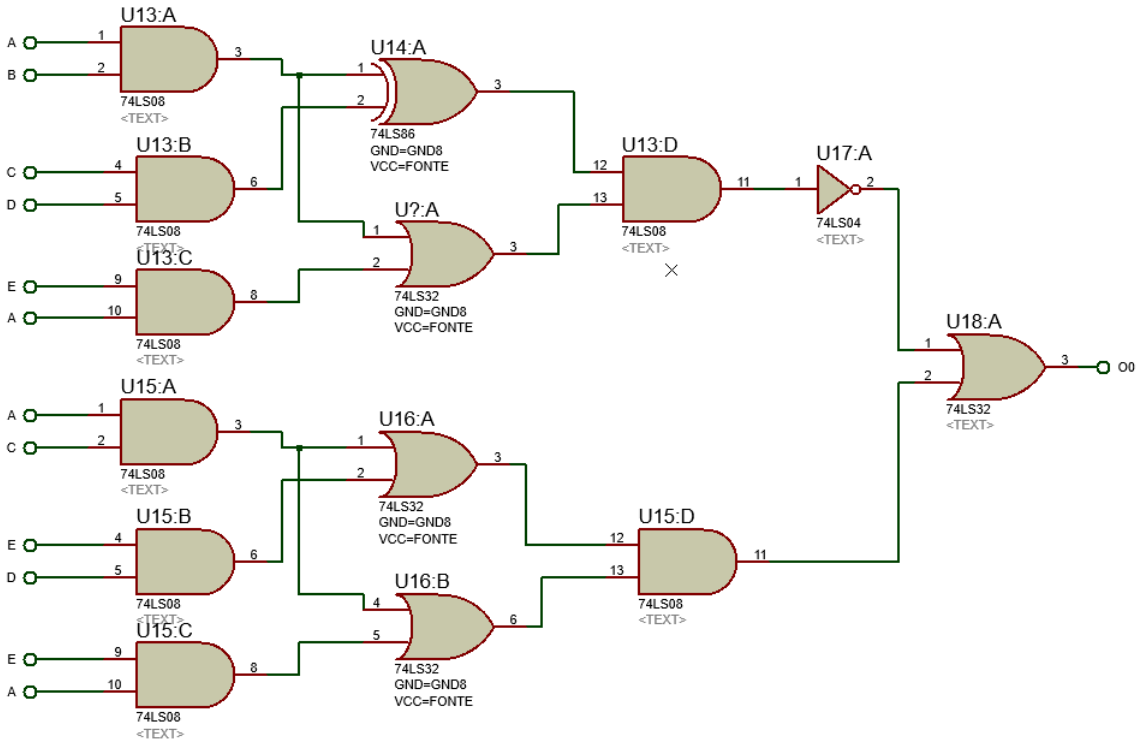

Figura 51 – Circuito lógico utilizado para simulação
Fonte: Autoria Própria.

Para o teste do circuito utilizou-se a *String ENS*, onde *E* é o comando de teste, *N* o número de variáveis e *S* o número da saída. Observa-se na Figura 51 que *N* é 5 pois estão contidas na entrada as variáveis lógicas A, B, C, D e E, e *S* a saída 0, pois é representada por O0 na mesma. Baseado nisso, envia-se *E50* pelo *Serial Monitor*, resultando no retorno da Figura 52. O que foi recebido é o formato de saída de uma Tabela Verdade, base de funcionamento de

todos os componentes do aplicativo em que o *software* acopla-se no final (MUNARINI *et al.*, 2014).

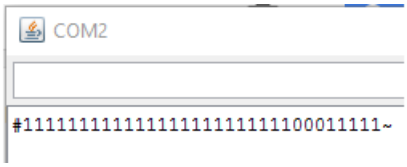

Figura 52 – Retorno para o comando *E*
Fonte: Autoria Própria.

Coloca-se o que foi recebido pela serial, Figura 52, no aplicativo citado, o que gera a expressão destacada pela Figura 53. Verifica-se que essa expressão é idêntica a da Equação Booleana 7.1, simplificada manualmente do circuito. Pode-se concluir então que a parte simulada funciona perfeitamente e responde fielmente ao esperado.

Figura 53 – Expressão para o circuito apresentado
Fonte: Autoria Própria.

7.4.5 Entradas e Saídas Lógicas

Outra funcionalidade simulada é a setagem das saídas de forma manual através da sequência *SXABCDE*, em que *S* é o comando de teste, *X* a saída setada e *ABCDE* as cinco variáveis booleanas de trabalho.

Para a realização da simulação foi utilizado o mesmo circuito da subseção anterior, Figura 51. Como exemplo utilizou-se a combinação de entradas A'.B.C.D'.E', bastando-a converter para o binário 01100 com a utilização da lógica da Tabela Verdade, onde uma variável negada é a representante do nível lógico baixo para aquela entrada, e inverso também se aplica. Visto que o circuito exemplo está conectado a saída O0, enviou-se a *String S001100* pela serial.

O resultado do exemplo anterior acarretou no comportamento do circuito de entradas e saídas da Figura 54. Nota-se que a combinação de entradas enviadas pela serial A'.B.C.D.'E' pôde ser vista claramente na parte inferior da figura, tal como sua versão negada A.B'.C'.D.E. Já na parte superior pôde ser visto o comportamento do circuito de saídas para aquele determinado instante testado.

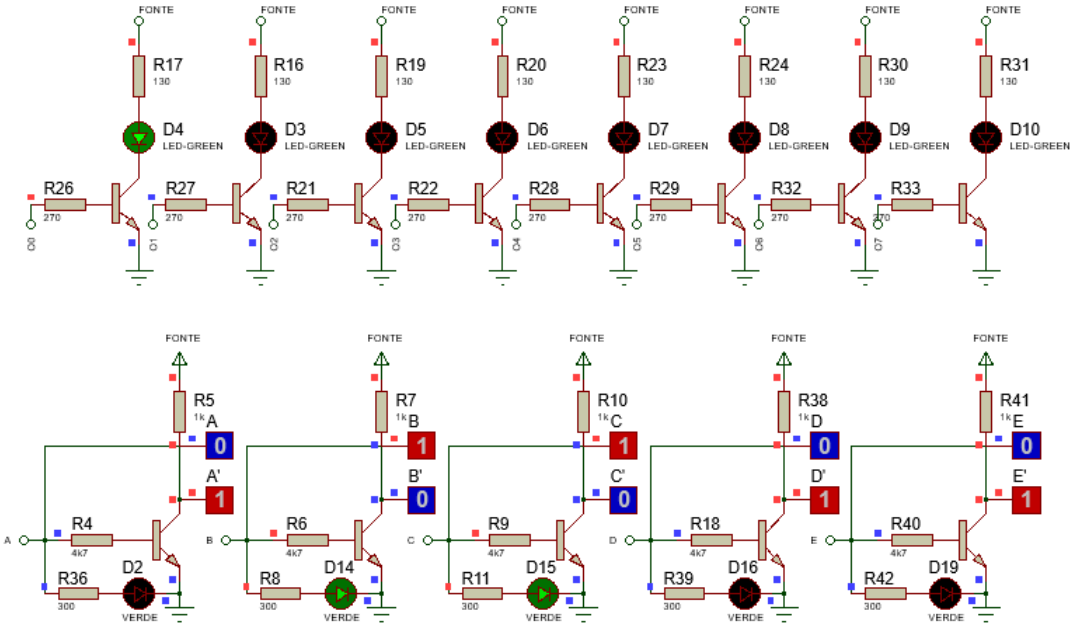

Figura 54 – Simulação das entradas e saídas
Fonte: Autoria Própria.

Ao prosseguir com a combinação analisada, a mensagem da Figura 55 foi retornada pelo microcontrolador, indicando que a saída para aquele instante é um nível lógico alto, ou simplesmente 1, o que pode ser visto também na saída O0 da Figura 54.

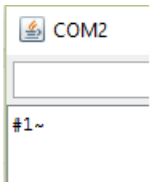

Figura 55 – Retorno do obtido comando *S*
Fonte: Autoria Própria.

7.4.6 Gerador de *Clock*

A última parte verificada foi o gerador de *clock*. Foram programadas três frequências no microcontrolador: 0,1Hz, 1Hz e 10Hz, sendo acionadas pelos comandos *01HZ*, *1HZ* e *10HZ*, respectivamente. A Figura 56 mostra como foi efetuado o teste para uma frequência de 10Hz, através de um leitor disponível no próprio *Proteus*.

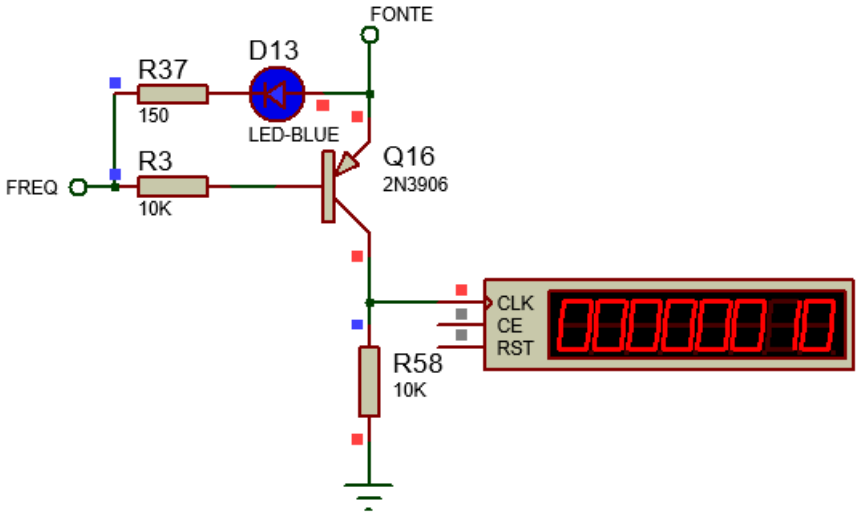

Figura 56 – Simulação para uma frequência de 10Hz
Fonte: Autoria Própria.

Analisando a figura, vê-se que o *clock* gerado pelo circuito através do comando *10HZ* foi exatamente o esperado. O comportamento se repetiu para os testes feitos nas frequências 0,1Hz e 1Hz, o que mostra a plena funcionalidade do circuito.

8 SOFTWARE

Para o presente trabalho foi implementada apenas a versão *Android*, utilizando a comunicação *bluetooth*. Porém, como o *hardware* já é preparado nativamente para comunicação serial, Seção 5.3, o mesmo procedimento abordado neste capítulo pode ser utilizado.

8.1 COMUNICAÇÃO *BLUETOOTH*

O processo de comunicação *bluetooth* no *Android* foi feito via uso de APIs, disponíveis no pacote interno de bibliotecas do sistema. A primeira classe utilizada para conexão é a *BluetoothAdapter*, que permite através de um endereço MAC a criação de uma instância denominada *BluetoothDevice*. Essa última é base para a criação de *sockets*, responsáveis pelo envio e recebimento de mensagens entre os aparelhos conectados.

A Figura 57 apresenta a interface utilizada para a obtenção do endereço MAC, no exemplo '98:D3:31:80:81:E6'. Isso foi feito utilizando a função *getBondedDevices* da própria classe *BluetoothAdapter*, que tem a função de receber todos os dispositivos pareados pelo aparelho. Nota-se que antes da execução do aplicativo deve-se parear o módulo *bluetooth* do *hardware* nas configurações do sistema *Android*.

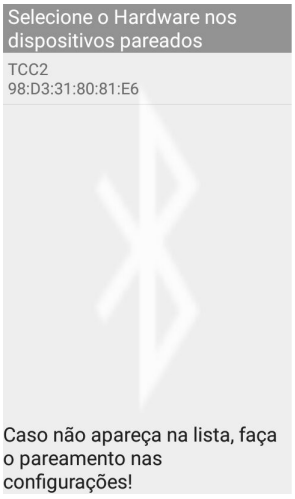

Figura 57 – Tela de conexão *bluetooth*
Fonte: Autoria Própria.

Após a conexão com a placa desenvolvida ser bem sucedida, é criado um objeto *BluetoothDevice* com o endereço MAC obtido do dispositivo pareado. Finalmente, com os dados requisitados, cria-se um *BluetoothSocket* para sustentar todo o fluxo de comunicação. O *socket* criado é atrelado então à um *Thread* denominado *ThreadConectado*, Figura 58, que monitora a todo momento o fluxo de dados entre as partes, além de ser utilizado para enviar e receber mensagens com o apoio de um *Handler* personalizado.

```
//classe para conectar o thread
private class ThreadConectado extends Thread {
    private final InputStream streamEntrada;
    private final OutputStream streamSaida;

    public ThreadConectado(BluetoothSocket socketBluetooth) {///criação do thread
        //variaveis temporárias
        InputStream entrada = null;
        OutputStream saida = null;
        try {//fonte de dados de entreda e saída do bluetooth
            entrada = socketBluetooth.getInputStream();
            saida = socketBluetooth.getOutputStream();
        } catch (IOException e) {//erro
            Toast.makeText(Bluetooth.this, "Falha na conexão!", Toast.LENGTH_LONG).show();
        }
        streamEntrada = entrada;
        streamSaida = saida;
    }

    public void run() {
        byte[] buffer = new byte[256];
        //Thread fica esperando a chegada de mensagens via bluetooth
        while (true) {
            try {
                String readMessage = new String(buffer, 0, streamEntrada.read(buffer));
                //envia os bytes recebidos para o handler
                h.obtainMessage(estadoHandler, streamEntrada.read(buffer), -1, readMessage).sendToTarget();
            } catch (IOException e) {
                break;
            }
        }
    }

    public void escreve(String entrada) {//envia String pelo bluetooth
        lastCommand = entrada;
        byte[] bufferMensagem = entrada.getBytes();//converte a String de parâmetro para bytes
        try {
            streamSaida.write(bufferMensagem);//envia os dados via bluetooth
        } catch (IOException e) {//se não conseguiu enviar via bluetooth
            Toast.makeText(Bluetooth.this, "Falha na conexão!", Toast.LENGTH_LONG).show();
            finish();
        }
    }
}
```

Figura 58 – Trecho de código do *Thread bluetooth*
Fonte: Autoria Própria.

Um exemplo de *Handler* é o da Figura 59, utilizado na Tela Principal do aplicativo. O *Thread* criado monitora a todo momento as mensagens recebidas, e caso haja correspondência, ela é enviada ao *Handler*, que por sua vez faz a verificação do *start bit* e *stop bit* mencionados na Seção 6.2. A mensagem desejada é extraída da *String* recebida através da remoção dos bits de detecção de erros, e a função correspondente a ser executa é feita baseando-se no último comando enviado pelo *bluetooth*, obtido através da variável *lastCommand* da Figura 58.

Cada tela disponível no aplicativo dispões de seu próprio *Handler*, porém a base de funcionamento segue a mesma da Figura 59, diferindo apenas no tratamento da mensagem recebida depois de extraída dos bits de verificação. Os últimos comandos enviados, comparados internamente à função analisada, são os implementados no microcontrolador, seguindo o padrão da Tabela 2.

```
Handler h = new Handler() {
    public void handleMessage(android.os.Message msg) {
        if (msg.what == bluetooth.getHandlerState()) {
            aux.append((String) msg.obj);//concatena mensagem recebida ao aux
            int index = aux.indexOf("~");//detecta o StopBit
            if (index > 0) {//se acabou a transmissao, detectado pelo StopBit
                String recebido = aux.toString();
                if (recebido.charAt(0) == '#') {//detecta o Start Bit
                    recebido = recebido.substring(1, index);//retira a informação da String recebida
                    if (bluetooth.getLastCommand().contains("C")) {//caso comando conter letra C
                        setCI(recebido);
                    } else if (bluetooth.getLastCommand().contains("E")) {//caso comando conter letra E
                        if (operacao == 0) {//expressao
                            expressao(recebido);
                        } else if (operacao == 1) {//tabela
                            tabela(recebido);
                        } else if (operacao == 2) {//mapa
                            mapa(recebido);
                        }
                    }
                }
                //limpa para proxima transmissão
                aux.delete(0, aux.length());
                recebido = "";
            }
        }
    }
};
```

Figura 59 – *Handler* atrelado ao *Thread*
Fonte: Autoria Própria.

8.2 TELA PRINCIPAL

A tela principal do aplicativo desenvolvido é mostrada na Figura 60. Nela encontra-se distribuída em uma representação de uma *protoboard* os 14 soquetes (C0 à C13) referentes aos CIs. Cada soquete foi feito com base de funcionamento semelhante a um botão, sendo à seu clique acionado um determinado evento.

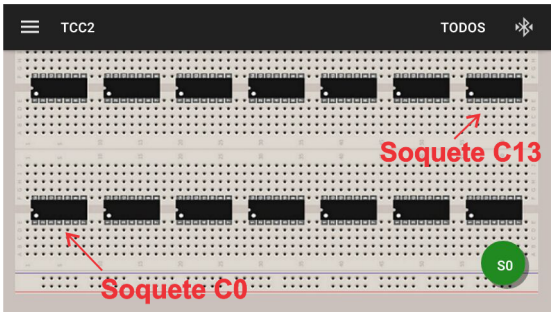

Figura 60 – Tela principal do Aplicativo
Fonte: Autoria Própria.

Ao soquete C0 ser pressionado envia-se via *bluetooth*, pela função *escreve* vista na Figura 58, a *String C0*. Já no pressionamento de C1 é enviado *C1*, e assim por diante conforme o número do CI requisitado. São identificados individualmente todos os CIs clicados pela resposta do microcontrolador aos comandos, conforme Subseção 6.3.5, e de acordo com ela são setados seus respectivos *labels*, seguindo a Tabela 5.

Tabela 5 – *Labels* setados nos CIs vs Código recebido

Código Recebido	Texto do Soquete
AND	7408
OR	7432
NOT	7404
NAND	7400
NOR	7402
XOR	7486

Fonte: Autoria própria.

O trecho de código responsável pela conversão dos dados recebidos para forma visual é apresentado na Figura 61.

```
private boolean setCI(String recebido) {//seta a porta lógica lida para o respectivo soquete
    String numeroCI = String.valueOf(recebido.charAt(0));//numero do CI a ser setado
    recebido = recebido.replace(numeroCI, "");//refaz a String sem o número do CI

    if (recebido.equals("AND")) {
        recebido = "7408";
    } else if (recebido.equals("OR")) {
        recebido = "7432";
    } else if (recebido.equals("NOT")) {
        recebido = "7404";
    } else if (recebido.equals("NAND")) {
        recebido = "7400";
    } else if (recebido.equals("NOR")) {
        recebido = "7402";
    } else if (recebido.equals("XOR")) {
        recebido = "7486";
    }

    //Seta o label no respectivo CI pelo index obtido
    int index = getResources().getIdentifier("ic" + numeroCI, "id", getPackageName());
    Button botao = ((Button) findViewById(index));
    botao.setText(recebido);
    return true;
}
```

Figura 61 – Trecho de código utilizado na conversão dos *labels*
Fonte: Autoria Própria.

O botão *S0* visto na Figura 60 é responsável pela definição da saída atual a ser lida ou setada. Ao clicá-lo, como o exemplo da Figura 62, é expandido uma lista com todas as saídas disponíveis no *hardware*. Uma variável global, *saidaAtual*, representando um número inteiro, é então alimentada de acordo com o botão (0 à 7) pressionado, e é utilizada posteriormente como base para as operações selecionadas pelo usuário.

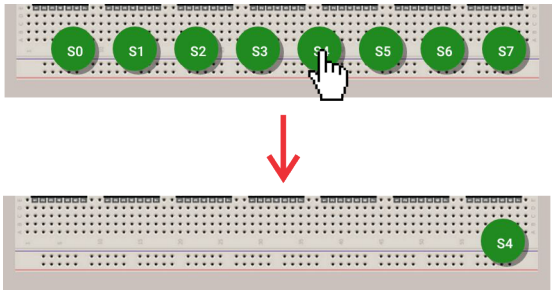

Figura 62 – Exemplo dos botões de saída
Fonte: Autoria Própria.

8.3 MENUS

Para acesso as outras ferramentas do aplicativo, além de manter a interface limpa e apresentável, criou-se um menu lateral deslizante, Figura 63. Apresentam-se no decorrer deste tópico os recursos derivados do menu e suas respectivas funções.

Figura 63 – Menu do aplicativo
Fonte: Autoria Própria.

8.3.1 Tabela Verdade

Ao clicar-se no item Tabela Verdade mostrado no menu da Figura 63, é enviado via *bluetooth* o comando *E5S*, onde *S* representa a saída atual a ser lida, substituída pela variável global *saidaAtual* mencionada na seção anterior.

Figura 64 – Item: Tabela Verdade
Fonte: Autoria Própria.

Visto as possíveis variações na Tabela 2, o comando *E* é enviado com o complemento 5, pois este é o número máximo de variáveis booleanas de entrada. Isso permite, através do trecho de código da Figura 65, a identificação do número de variáveis utilizadas pelo circuito lógico montado de forma automática.

Essa identificação automática do número de variáveis é feita com a utilização do método de Quine-McCluskey, já implementado no aplicativo a qual o *software* desenvolvido nesse projeto integra-se posteriormente (MUNARINI *et al.*, 2014). A *String* é recebida do *hardware*

```
public void tabela(String expressao) {
    expressao = makeExpressao(expressao);//gera expressão pelo método de Quine-McCluskey
    chooseNumeroVariaveis(expressao);//retira o número de variaveis da expressão gerada
    Express inicial = new Express(expressao);//método conversor
    expressao = inicial.getStringSolucao();//converte no formato de Tabela Verdade
    chooseTabelaMapa(0, expressao);//seta a tabela verdade na interface
}
```

Figura 65 – Trecho de código da Tabela Verdade
Fonte: Autoria Própria.

segue o formato indicado pela Subseção 6.3.7, e através dela é retirada a expressão booleana simplificada por este método. A partir da expressão lógica, é possível contar o número de variáveis envolvidas simplesmente analisando os caracteres *A*, *B*, *C*, *D* e *E*, o que é feito pela função *chooseNumeroVariaveis* na Figura 66.

```
private void chooseNumeroVariaveis(String finalExpressao) {
    numeroVariaveis = 0;
    if (finalExpressao.contains("A")) {
        numeroVariaveis++;
    }
    if (finalExpressao.contains("B")) {
        numeroVariaveis++;
    }
    if (finalExpressao.contains("C")) {
        numeroVariaveis++;
    }
    if (finalExpressao.contains("D")) {
        numeroVariaveis++;
    }
    if (finalExpressao.contains("E")) {
        numeroVariaveis++;
    }
}
```

Figura 66 – Código seletor de variáveis
Fonte: Autoria Própria.

Após a contagem do número de varáveis, de acordo com a Figura 64, gera-se uma nova *String* no mesmo formato da recebida inicialmente pelo *bluetooth*, setando-a na tabela verdade, o que gera o resultado da Figura 64. É possível também setar manualmente cada linha da Tabela Verdade no circuito de entradas implementado na Seção 5.4, simplesmente clicando individualmente na linha desejada, assim a coluna de resposta S_n é atualizada em tempo real.

Caso deseje-se alterar o número da saída a ser lida, clica-se na coluna S_n da Figura 64 ou no campo presente no menu, também representado por S_n, onde *n* é o número da saída escolhida atualmente. No momento de escolha, a interface mostrada na Figura 67 é exibida e a *flag* global responsável pelo controle da saída atual é atualizada.

Como resultado ao clique de uma linha, o comando *S* exemplificado na Subseção 6.3.6 é enviado via *bluetooth* para o *hardware*. O formato de envio depende do número de variáveis e saída utilizadas, e imediatamente após a chegada da resposta ao comando, a coluna S_n é setada com o valor obtido.

No exemplo da Figura 64, a linha clicada foi a terceira, com o valor de A = 1 e B = 0. Isso gera o envio do comando no formato *S010000*, de acordo com as especificações contidas na Tabela 2. Nota-se que a resposta obtida da placa foi 0, pois esse valor binário está realçado na cor verde.

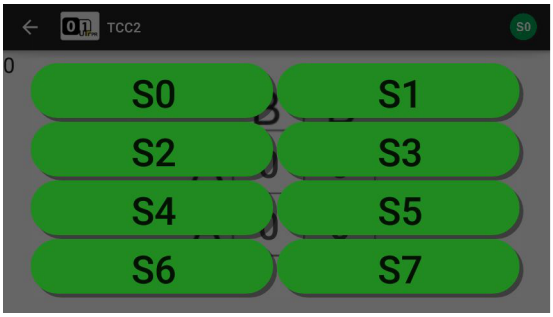

Figura 67 – Tela seletora de variáveis
Fonte: Autoria Própria.

8.3.2 Mapa de Karnaugh

O Mapa de Karnaugh, de acordo com a Figura 3 da Seção 3.1, consiste no arranjo da tabela da verdade em um formato de células. Por esse motivo, a ferramenta Mapa de Karnaugh disponível no menu tem seu funcionamento idêntico ao da Tabela Verdade tratada na seção anterior, tanto nos comandos enviados e recebidos quanto na escolha de variáveis de saída. Difere-se apenas na seleção da tela subsequente, a qual é gerada a interface mostrada na Figura 68.

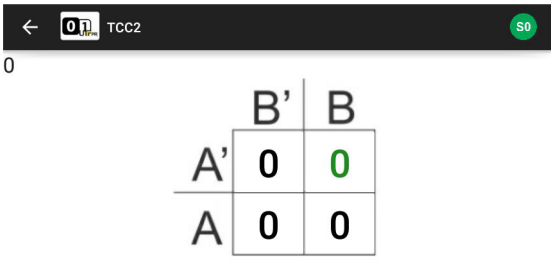

Figura 68 – Item: Mapa de Karnaugh
Fonte: Autoria Própria.

O exemplo da Figura 68 mostra uma combinação de A = 0 e B = 1, onde o circuito responde com o valor binário 0, constatado devido a cor setada para o padrão de resposta.

8.3.3 Expressão Lógica

Um dos principais recursos implementados pelo aplicativo é a extração da expressão lógica de um circuito digital montado na placa, além da verificação de equivalência de uma expressão digitada com o mesmo. Ao selecionar o item Expressão Lógica no menu apresentado na Figura 63, exibi-se o submenu da Figura 69, possibilitando a escolha entre as duas funções mencionadas.

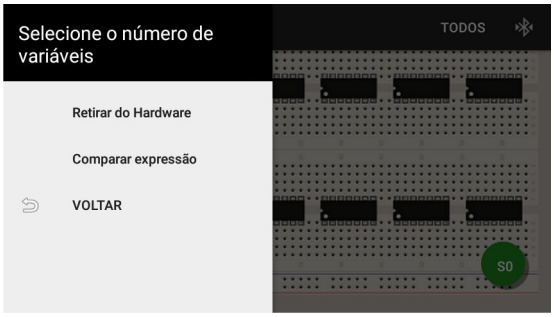

Figura 69 – Submenu da Expressão Lógica
Fonte: Autoria Própria.

Caso deseje-se retirar a expressão de um circuito implementado no *hardware*, é utilizado o mesmo procedimento dos recursos anteriores, com o envio do comando *E5S* ao microcontrolador, onde *S* corresponde a saída atual escolhida pelo usuário. Executa-se então o trecho de código da Figura 70, com algoritmo de Quine-McCluskey para retirada da expressão, por meio da classe *GeraImplicantes* do aplicativo base. O método *executa* da classe mencionada, com um parâmetro de uma *String* no formato da saída de uma tabela verdade, como exemplificado na Figura 38, retorna a expressão simplificada no formato convencional.

```
public void expressao(String expressao) {//simplifica a expressao
    GeraImplicantes resposta = new GeraImplicantes();
    expressao = resposta.executa(expressao);
    Intent i = new Intent();
    i.setClass(this, Expressao.class);
    i.putExtra("Expressao", expressao);
    startActivity(i);
    onStop();
}
```

Figura 70 – Trecho de código da Expressão
Fonte: Autoria Própria.

Após a retirada da expressão pelo método de Quine-McCluskey, inicia-se uma nova atividade dando origem a Interface da Figura 71, exibindo-a no campo disponível para digitação. Essa tela permite a edição da expressão lógica, além retirada da mesma do *hardware* de forma dinâmica com a modificação da saída apresentada como *S0* no exemplo.

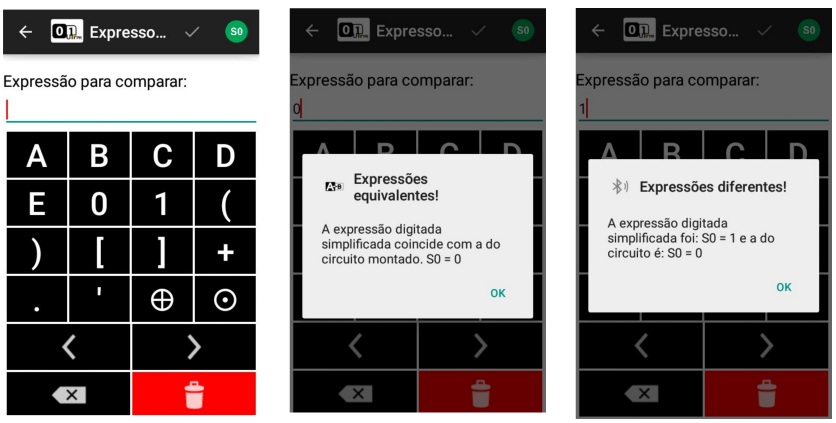

Figura 71 – Funcionalidades do item Expressão Lógica
Fonte: Autoria Própria.

Em contrapartida, caso escolha-se o recurso *Comparar Expressão* no submenu, a tela é exibida automaticamente, sem a execução do método dos implicantes primos anteriormente. Após a digitação da expressão lógica desejada no campo *Expressão para comparar*, e a escolha da saída na barra superior, executa-se o código da Figura 70.

Faz-se então uma comparação da variante digitada com a resposta da placa por meio do código da Figura 72, dando origem à duas telas de dialogo distintas, também mostradas na Figura 71.

```
public void expressao(String expressao) {
    GeraImplicantes resposta = new GeraImplicantes();
    expressao = resposta.executa(expressao);
    if (expressao.equals(this.expressao)) {
        new AlertDialog.Builder(this)
                .setTitle("Expressões equivalentes!")
                .setMessage("A expressão digitada simplificada coincide com a do circuito montado. S" + saidaAtual + " = " + expressao)
                .setPositiveButton("Ok", new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int which) {

                    }
                })
                .setIcon(R.mipmap.expressao_black)
                .show();
    } else {
        new AlertDialog.Builder(this)
                .setTitle("Expressões diferentes!")
                .setMessage("A expressão digitada simplificada foi: S"+saidaAtual+" = "+this.expressao+" e a do circuito é: S"+
                        saidaAtual+" = "+expressao)
                .setPositiveButton("Ok", new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int which) {

                    }
                })
                .setIcon(R.mipmap.bluetooth_procurando)
                .show();
    }
}
```

Figura 72 – Trecho de Código dos alertas exibidos
Fonte: Autoria Própria.

É possível à qualquer momento transitar entre as interfaces de forma dinâmica, e rapidamente obter a resposta do *hardware* para um determinado circuito lógico montado.

8.3.4 Testar CI

A tela do aplicativo responsável pelo controle do circuito da Seção 5.7 é apresentada na Figura 73. Isso é feito com o envio do comando *ERROS* ao microcontrolador, tratado na Subseção 6.3.2.

O código responsável pela lógica da operação simplesmente recebe a *String* do Atmega32A no formato '#+P0+P1+P2+P3+∼', e a trata da forma adequada para a identificação de erros. Para tal, basta-se comparar os termos P_x referentes às portas lógicas, que em caso de igualdade entre si, representam um bom funcionamento em todas as composições do CI. Caso um termo difira, significa que um mal funcionamento é apresentado, identificado no aplicativo de acordo com o trecho de código da Figura 74, que basicamente seta a cor vermelha no conjunto de pinos referente à porta lógica defeituosa.

Figura 73 – Tela do soquete de verificação
Fonte: Autoria Própria.

```
public void setColor(int i) {
    switch (i) {
        case 1:
            pin1.setBackgroundColor(getResources().getColor(R.color.vermelho_transparente));
            pin2.setBackgroundColor(getResources().getColor(R.color.vermelho_transparente));
            pin3.setBackgroundColor(getResources().getColor(R.color.vermelho_transparente));
            break;
        case 2:
            pin4.setBackgroundColor(getResources().getColor(R.color.vermelho_transparente));
            pin5.setBackgroundColor(getResources().getColor(R.color.vermelho_transparente));
            pin6.setBackgroundColor(getResources().getColor(R.color.vermelho_transparente));
            break;
        case 3:
            pin8.setBackgroundColor(getResources().getColor(R.color.vermelho_transparente));
            pin9.setBackgroundColor(getResources().getColor(R.color.vermelho_transparente));
            pin10.setBackgroundColor(getResources().getColor(R.color.vermelho_transparente));
            break;
        case 4:
            pin11.setBackgroundColor(getResources().getColor(R.color.vermelho_transparente));
            pin12.setBackgroundColor(getResources().getColor(R.color.vermelho_transparente));
            pin13.setBackgroundColor(getResources().getColor(R.color.vermelho_transparente));
            break;
    }
}
```

Figura 74 – Trecho de Código indicador de erros
Fonte: Autoria Própria.

Nessa etapa também é feito o processo de identificação do CI, assim como na Tela Principal, onde o caso apresentado na Figura 73 mostra um soquete vazio.

8.3.5 Gerar Frequência

O último recurso disponível no menu da Tela Principal é o responsável pela geração do Clock. Para tal, o aplicativo envia os comandos *01HZ*, *1HZ*, *10HZ* ou *OFF* de acordo com a Tabela 2, pela função *escreve* do *Thread* implementado para o *bluetooth*, Figura 58. Envia-se o comando de acordo com o clique em seu respectivo item, mostrado nos submenus da Figura 75.

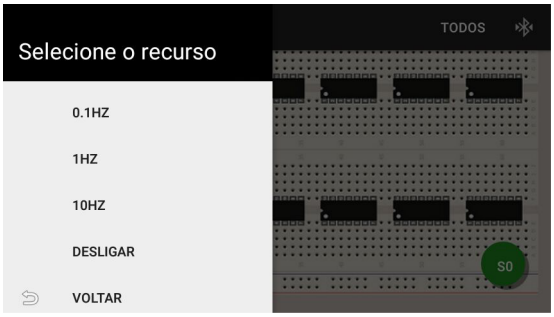

Figura 75 – Tela com opções de frequência
Fonte: Autoria Própria.

9 RESULTADOS E DISCUSSÕES

Para a verificação dos resultados foram utilizados dois roteiros de laboratório diferentes, Apêndice A, de forma que fosse possibilitado o pleno uso dos recursos disponibilizados, tanto pelo *hardware* quanto pelo *software*, visando a demonstração dos benefícios do uso dos mesmos na disciplina de Circuitos Digitais.

Os roteiros foram elaborados com possibilidade de dois caminhos: a extração de todos os dados de um projeto (Tabela Verdade, Mapa de Karnaugh e Expressão Lógica) a partir de um circuito previamente implementado na placa, e a elaboração de um circuito lógico montado no *hardware* após seus dados tratados pelo programa, assim como seu teste de funcionamento.

Para o primeiro caminho escolheu-se um circuito somador de dois *bits*, e para o segundo um subtrator de dois *bits*. A razão da escolha desses circuitos é que, apesar de desempenharem funções diferentes, possuem uma base de implementação equivalente, o que permite a análise do comportamento do conjunto *hardware*/*software* por meio de dois prismas distintos.

Os dois roteiros seguidos, tal como seus resultados são descritos a seguir.

9.1 DO *HARDWARE* PARA O *SOFTWARE*

O primeiro roteiro consiste no caminho inverso ao comum, ou seja, a montagem de um circuito somador de dois bits, como o mostrado na Figura 76, diretamente no *kit*, com o intuito de retirar seus dados de projeto automaticamente via integração do *hardware* com o *software* desenvolvido.

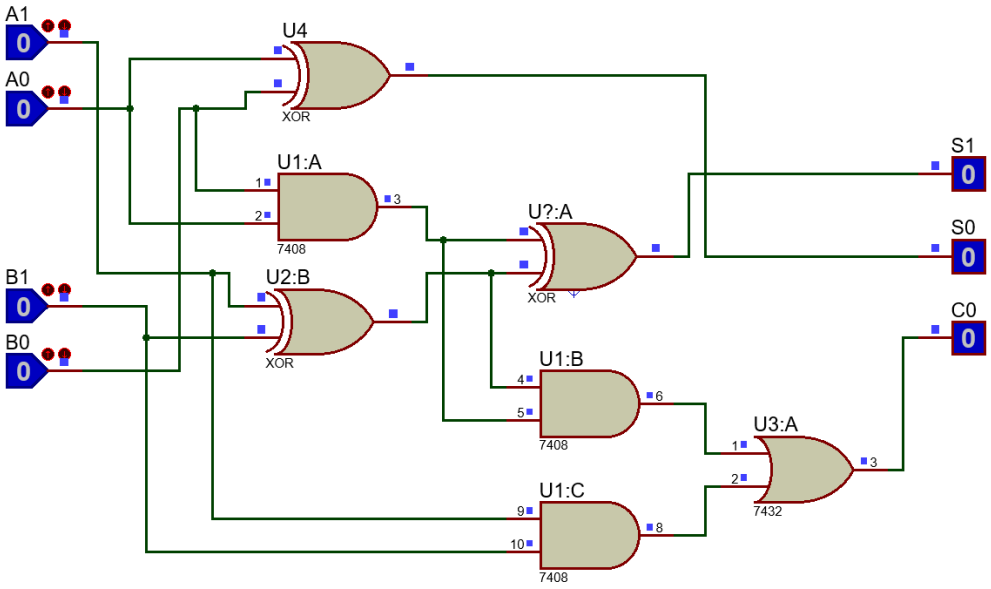

Figura 76 – Circuito somador de dois bits
Fonte: Autoria própria.

Verifica-se que são necessários três CIs ao todo para a composição do circuito apresentado: 7486 (XOR), 7408 (AND) e 7432 (OR). Os três componentes foram conectados ao soquete de verificação de erros do protótipo e testados um a um, dando origem as telas mostradas na Figura 77. Pode-se observar que o kit não só identifica prontamente, como também exibe a mensagem esperada à um CI em perfeito estado de funcionamento. Esse recurso disponibilizado pela placa previne o uso de componentes danificados, o qual ocasionaria um grande atraso na montagem prática, tal como a dificuldade do diagnóstico caso haja algum erro por parte da implementação física.

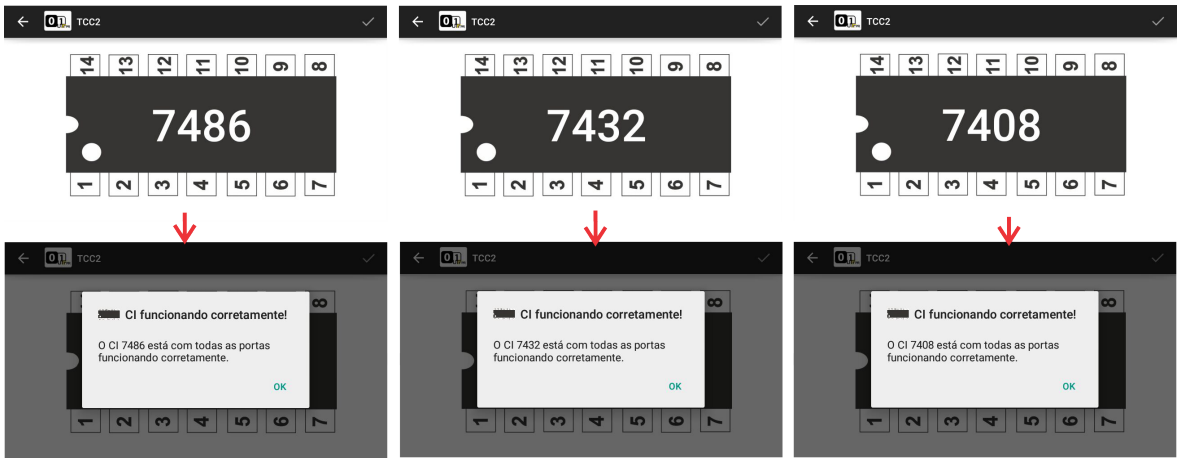

Figura 77 – 7486, 7432 e 7408 inseridos no soquete de verificação de erros
Fonte: Autoria própria.

Posteriormente os CIs foram inseridos na *protoboard* principal, onde efetuam-se todas as ligações físicas necessárias para a reprodução do circuito somador. As entradas e saídas lógicas do circuito foram feitas no protótipo conforme as adequações mostradas na Tabela 6.

Tabela 6 – Distribuição das entradas e saídas do Somador no protótipo

Entradas do Protótipo	Entradas do Somador	Saídas do protótipo	Saídas do Somador
A	A1	S0	S1
B	A0	S1	S0
C	B1	S2	C0
D	B0		

Fonte: Autoria própria.

A operação de soma binária é exemplificada na Figura 78, onde vê-se a soma do número binário 11 (decimal 3) e 10 (decimal 2), resultando em 011 (decimal 5). Também mostra-se as adequações apresentadas na Tabela 6, além de como é feita a soma dos números a partir da combinação binária das entradas lógicas.

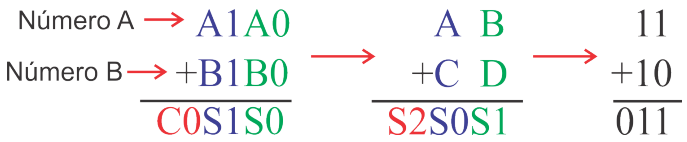

Figura 78 – Exemplo da operação de soma binária
Fonte: Autoria própria.

Após a montagem do somador no protótipo, depois de feitas as adequações mencionadas anteriormente, obteve-se a identificação dos CIs inseridos na *protoboard* ao pressionar suas respectivas posições de soquete, conforme mostrado na Figura 79. Rapidamente, os *labels* foram setados, constatando o funcionamento não só do circuito de identificação da Seção 5.6, como também do circuito comutador de alimentação utilizado para prevenção de erros na leitura, Seção 5.6.1.

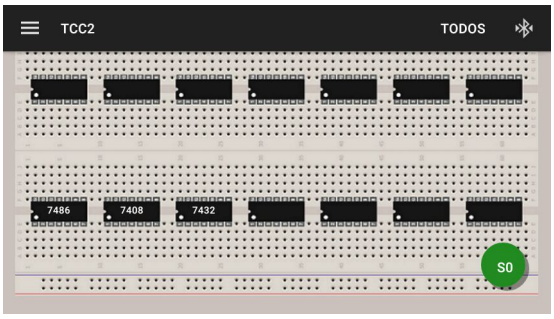

Figura 79 – CIs identificados no Aplicativo
Fonte: Autoria própria.

Para a extração dos dados do projeto, utilizou-se os recursos do menu apresentados na Subseção 8.3. O primeiro item obtido é a Tabela Verdade para cada uma das três saídas do somador de 2 *bits* (S1, S0 e o *carry* C0). Observando a Figura 80 e a Tabela 6, concluiu-se que o circuito foi sintetizado com sucesso, tanto nos dois *bits* de resposta da soma representados por S0 e S1, quanto no *carry out*, representado por S2.

Tabela 4 variáv... S0

A	B	C	D	S0
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Tabela 4 variáv... S0

A	B	C	D	S1
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Tabela 4 variáv... S0

A	B	C	D	S2
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Figura 80 – Tabelas Verdade do Somador
Fonte: Autoria própria.

Alternativamente, o Mapa de Karnaugh obtido do circuito encontra-se representado na Figura 81, juntamente com a expressão lógica simplificada de cada uma das saídas.

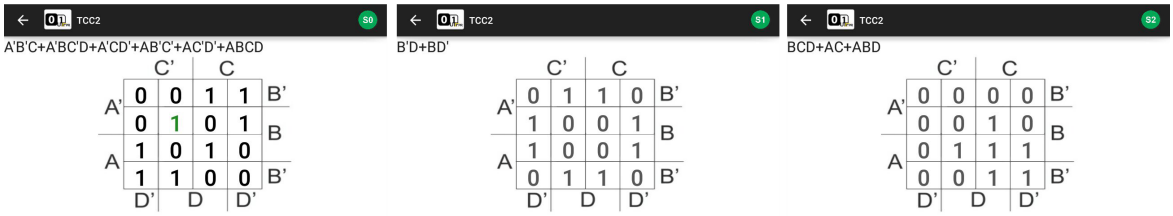

Figura 81 – Mapas de Karnaugh do Somador
Fonte: Autoria própria.

Outra forma de verificar a exatidão de um circuito implementado, é a comparação do mesmo com uma expressão lógica. É mostrado na Figura 82 a comparação feita entre as três saídas e suas respectivas expressões no formato XOR, onde é possível observar que o conjunto *hardware* e *software* identifica fielmente as expressões, mesmo não estando no formato de trabalho do programa principal, o qual simplifica as expressões e as retorna somente no formato de soma de produtos, como visto nos três Mapas de Karnaugh extraídos.

A fim de verificar o último conjunto de instruções relacionadas aos circuitos lógicos, escolheu-se para o teste linha à linha, uma combinação aleatória de variáveis na Tabela Verdade. A Figura 83 mostra essa escolha, onde verifica-se que para uma combinação, seguindo a Tabela 6, de A1, A0, B1 e B0 igual à 0011, a saída S0 do protótipo que corresponde ao S1 do somador é lida como nível lógico alto, o que condiz com a Tabela da Figura 80. Nota-se que as entradas em vermelho foram setadas exatamente igual à combinação escolhida, gerando um conjunto de saídas, em verde, lidas em tempo real pelo *software*.

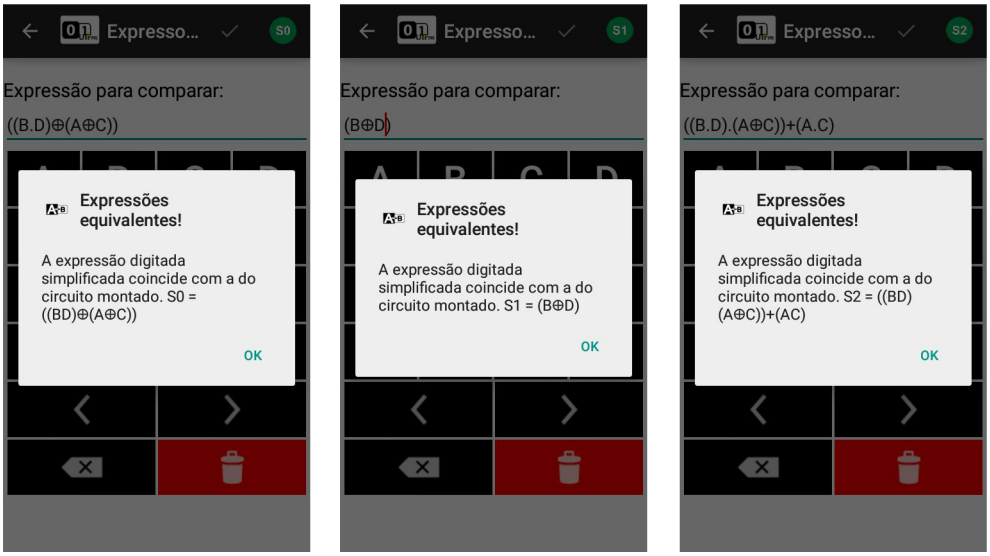

Figura 82 – Expressões retiradas do Circuito
Fonte: Autoria própria.

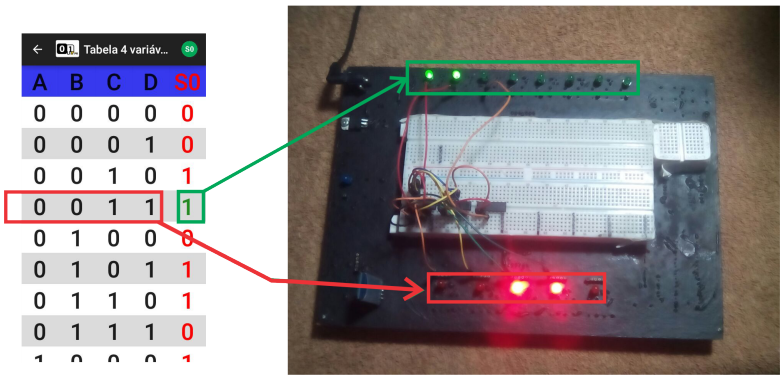

Figura 83 – Disposição da Tabela Verdade no protótipo
Fonte: Autoria própria.

Essa operação pode ser feita com todas a linhas da Tabela Verdade ou Mapa de Karnaugh, um recurso importante na verificação prática de como um circuito se comporta a variação de níveis lógicos, o que permite um ensino menos abstrato dos conceitos envolvidos, ao mesmo tempo em que possibilita a observação da real aplicação dos mesmos na prática.

9.2 DO *SOFTWARE* PARA O *HARDWARE*

Como já mencionado, o circuito de um subtrator de 2 bits foi escolhido para o caminho direto, ou seja, do *software* para o *hardware*. Como primeiro passo, foram montadas as Tabelas Verdade das saídas do circuito em relação à sua entrada, através do conceito de subtração binária apresentado na Tabela 7.

A Figura 84 traz um exemplo de subtração de dois números binários de dois *bits*, juntamente com as adequações necessárias a serem feitas para a montagem do circuito no protótipo,

Tabela 7 – Subtração Binária

A	B	Subtração	Resto
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Fonte: Autoria própria.

análogo ao somador implementado na seção anterior.

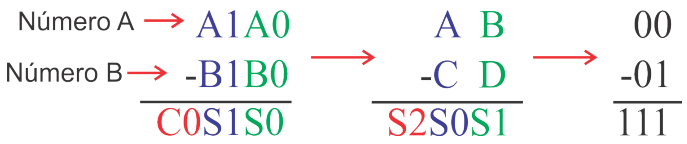

Figura 84 – Exemplo da operação de subtração binária
Fonte: Autoria própria.

Feita a subtração para todas as combinações binárias possíveis para dois números de dois *bits*, gerou-se três Tabelas Verdade, uma para S0, primeiro *bit* do resultado, para S1, segundo *bit* e uma última para S2, o *carry out* C0. As três tabelas foram setadas diretamente no aplicativo e encontram-se disponíveis na Figura 85.

A	B	C	D	S0
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

A	B	C	D	S1
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

A	B	C	D	S2
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Figura 85 – Tabelas Verdade do Subtrator
Fonte: Autoria própria.

A partir das tabelas retira-se os Mapas de Karnaugh, Figura 86, com o intuito de obter uma expressão simplificada que simbolize o circuito. O aplicativo faz esse processo de forma automatizada, basta-se clicar no menu superior com a simbologia do Mapa, conforme indicado pelo círculo vermelho na barra superior da Figura 85. Nota-se que o processo inverso também pode ser feito no *software*, iniciando-se pelo Mapa de Karnaugh e posteriormente extraindo-se a Tabela Verdade.

De acordo com Munarini *et al.* (2014) a expressão simplificada é retirada automaticamente pelo método de Quine-McCluskey, e pode ser acessada em todas as telas disponíveis no programa, Expressão Lógica, Tabela Verdade e Mapa de Karnaugh. Observa-se nos mapas extraídos da tabela que a expressão simplificada se dá, como já mencionado, por soma de produtos, porém pode facilmente ser adaptada para o formato XOR ou outro desejado, somente efetuando o agrupamento dos termos semelhantes.

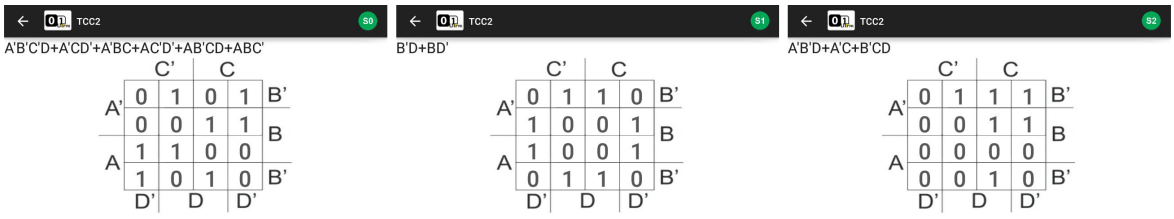

Figura 86 – Mapas de Karnaugh do Subtrator
Fonte: Autoria própria.

A fim de diminuir os CIs de portas lógicas a serem utilizados, fez-se as seguintes simplificações, com a utilização do próprio aplicativo para auxílio a cada passo efetuado:

$$\begin{aligned} S0 &= D.B'(A'C' + AC) + A'C &&= (DB')\oplus(A\oplus C) \\ S1 &= B\oplus D \\ S2 &= DB'(A'C' + AC) + A'C &&= ((DB')(A\oplus C)') + (A'C) \end{aligned} \tag{9.1}$$

Com isso, obtêm-se uma expressão lógica no formato de XOR, mas nada impede que o usuário opte pela utilização das expressões no formato de mintermos, utilizando mais CIs para montagem física. O circuito lógico da Figura 87 é obtido a partir das expressões da Equação 9.1.

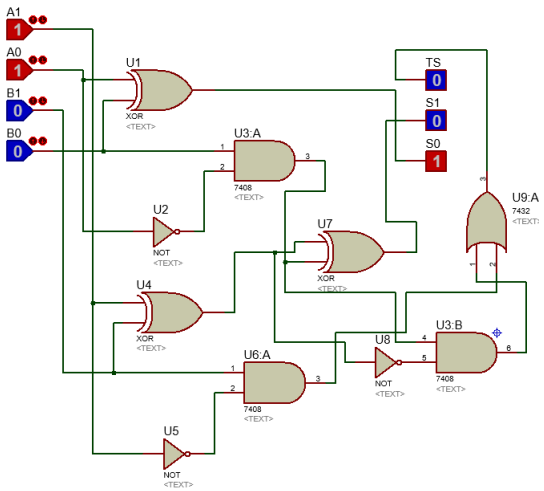

Figura 87 – Circuito lógico do subtrator
Fonte: Autoria própria.

O circuito obtido foi montado na *protoboard* principal do protótipo, Figura 88. Verificou-se que a presença da versão negada das entradas lógicas, recurso desenvolvido na Subseção 5.4.1,

foi de suma importância para a facilitação da montagem do circuito, reduzindo o número de portas inversoras de três para apenas uma. Isso possibilitou a adição de somente um CI extra, o 7404, em relação ao circuito do somador.

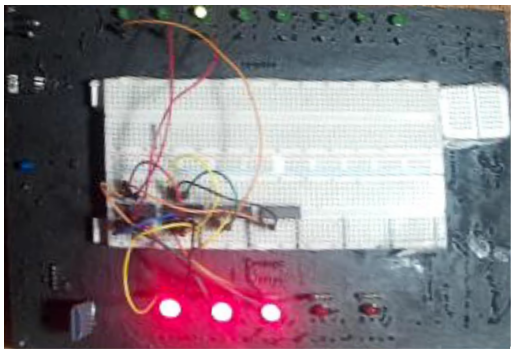

Figura 88 – Circuito no protótipo
Fonte: Autoria própria.

Como forma de verificar o funcionamento correto do circuito implementado a partir dos dados inciais do problema, foram comparadas as expressões lógicas encontradas na Equação 9.1, do mesmo modo que o feito na Figura 82. As comparações feitas são exibidas na Figura 89, onde pode-se observar o sucesso da implementação do circuito lógico de um subtrator de 2 *bits*, o qual todo projeto foi feito junto ao *software* ou com seu auxílio.

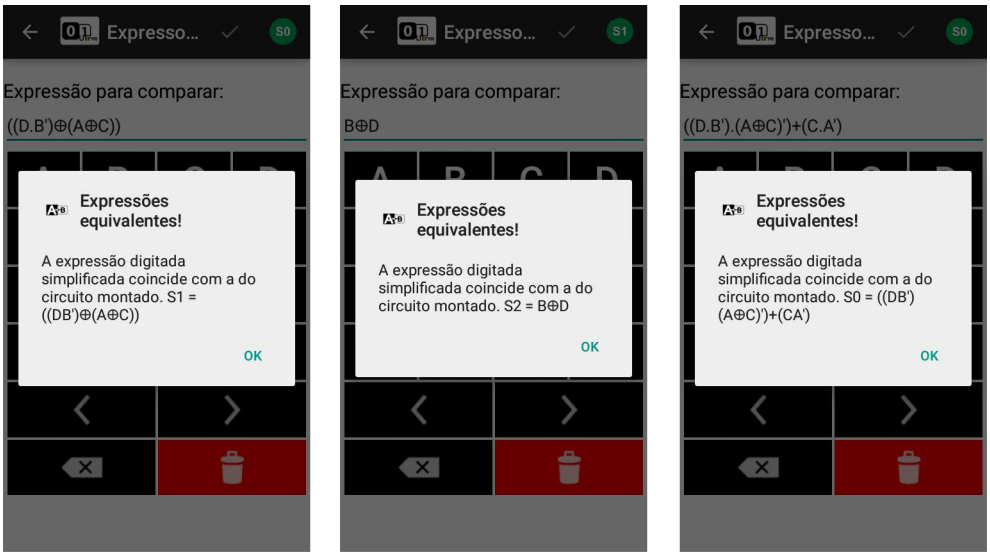

Figura 89 – Expressões retiradas do Subtrator
Fonte: Autoria própria.

O processo de verificação pode ser feito também a partir da extração das Tabela Verdade ou Mapa de Karnaugh do circuito, análogo ao feito pelo primeiro roteiro apresentado, tal como a parte de verificação inicial de erros dos circuitos integrados utilizados e sua identificação junto aos soquetes de montagem.

10 CONCUSÃO

Após a elaboração dos dois roteiros de laboratório apresentados no Apêndice A, tanto pela extração dos dados do projeto a partir do somador de dois *bits* quanto pela elaboração total do projeto do subtrator de dois *bits* no aplicativo, verificou-se uma enorme vantagem no uso do kit didático desenvolvido.

A primeira vantagem se deu pela prevenção inicial de erros devido à um mal funcionamento em alguma porta lógica dos CIs, o que evita o desmanche dos circuitos implementados, caso seja constatado tal problema em meio ao processo de montagem. Além disso, os 14 soquetes disponíveis já com a alimentação empregadas, não só facilitam o processo de ligação, como auxiliam na prevenção de falhas pelo seu recurso de identificação imediato presente no *software*.

Como um dos recursos mais importantes desenvolvidos no kit está a extração dos dados do projeto de um circuito implementado na *protoboard* central e seu teste de funcionamento. Como isso é feito de forma automatizada, o tempo de resposta no pior caso, para o teste do circuito com 5 variáveis, é de 200ms. Se comparado aos kits convencionais utilizados onde tudo é feito de forma manual através de chaves seletoras, facilmente nota-se uma grande vantagem em relação à seu uso.

A integração do *hardware* desenvolvido neste projeto com o *software* já implementado em Munarini *et al.* (2014), permite através do uso do algoritmo de Quine-McCluskey a extração quase imediata dos dados de um circuito lógico, tanto fisicamente implementado quanto desenvolvido no aplicativo. Tendo isso em mente, e levando em consideração que as dificuldades dos discentes encontram-se principalmente nos conceitos iniciais, os resultados aqui obtidos são de grande valia, justificando o uso do kit em todas disciplinas que possuem os conceitos de lógica digital em sua ementa, sendo eles práticos ou teóricos.

Os resultados avaliados foram totalmente satisfatórios, cumprindo todos os requisitos especificados no escopo do projeto e impactando visivelmente na facilidade de absorção do conteúdo e ensino de Eletrônica Digital.

Por fim, além da já mencionada manipulação da Tabela Verdade, Mapa de Karnaugh ou Expressão Lógica pelas duas vias de comunicação, pode-se futuramente implementar no *software* funções de sintetização de circuitos síncronos, já que o *hardware* contém um gerador de *clock* e as leituras das saídas feitas de forma dinâmica.

REFERÊNCIAS

ATMEL, C. **8-Bit AVR Microcontroller, Atmega32 - DATASHEET COMPLETE**. [S.l.]: Atmel Corporation, 2016. Disponível em: <http://www.atmel.com/Images/Atmel-8155-8-bit-Microcontroller-AVR-ATmega32A_Datasheet.pdf>. Acesso em: 3 mar. 2016.

BRAGA, N. C.; PAIOTTI, R. **Eletrônica Digital - I**. São Paulo - Brasil: Editora Newton C. Braga, 2012. v. 3. ISBN 9788565050135. Disponível em: <https://books.google.com.br/books?id=ebfiBgAAQBAJ>. Acesso em: 5 set. 2015.

CONNER, E. **Atmega32-Arduino**. [S.l.]: Microchip Technology Inc, 2016. Disponível em: <https://github.com/eaconner/ATmega32-Arduino>. Acesso em: 20 mar. 2016.

CUNHA, M. R. **Circuitos Digitais**: Aula 03 - circuitos lógicos e suas representações. Universidade Tecnológica Federal do Paraná - Campus Campo Mourão - PR: [s.n.], 2014. 53 p. Disponível em: <http://paginapessoal.utfpr.edu.br/marciocunha/ensino/engenharia-eletronica/circuitos-digitais/Aula03_CD_LT35C.pptx/at_download/file>, note = Notas de Aula. Acesso em: 5 set. 2015.

____. **Circuitos Digitais**: Aula 05 - mapas de karnaugh. Universidade Tecnológica Federal do Paraná - Campus Campo Mourão - PR: [s.n.], 2014. 53 p. Disponível em: <http://paginapessoal.utfpr.edu.br/marciocunha/ensino/engenharia-eletronica/circuitos-digitais/Aula05_CD_LT35C.pptx/at_download/file>. Notas de Aula. Acesso em: 5 set. 2015.

DATAPOOL. **Módulo Universal 2000**. 2015. Disponível em: <http://eletronica.datapool.com.br/produtos/modulo-universal-2000-e-cartoes-microcontroladores-microprocessadores/modulo-universal-2000/>. Acesso em: 09 ago. 2015.

ELECTRONICS, F. **ZIF Socket 14 pin LE 10.00**. [S.l.]: Electronics, Future, 2016. Disponível em: <http://store.fut-electronics.com/products/zif-socket-14-pin>. Acesso em: 3 mar. 2016.

FLOYD, T. L. **Sistemas digitais**: fundamentos e aplicações. 9. ed. Porto Alegre: Bookman, 2007. 888 p. ISBN 9788560031931.

Guangzhou HC, I. T. C. L. **HC-06 Produtuct Data Sheet**. [S.l.]: Guangzhou HC, Information Technology Co. Ltd, 2011. Disponível em: <https://www.olimex.com/Products/Components/RF/BLUETOOTH-SERIAL-HC-06/resources/hc06.pdf>. Acesso em: 10 ago. 2016.

IDOETA, I. V.; CAPUANO, F. G. **Elementos da eletrônica digital**. 41. ed. São Paulo, SP: Érica, 2012. 544 p. ISBN 9788571940192.

MCCLUSKEY, E. J. Minimization of boolean functions*c. **Bell system technical Journal**, v. 35, n. 6, p. 1417–1444, 1956.

MICALOSKI, O. **Mini controlador lógico programável**. 55 p. Monografia (Trabalho de Conclusão de Curso) — Universidade Tecnológica Federal do Paraná, Curitiba, PR, 2012.

MLSNA, P. A.; LISZEWSKI, E. Effectiveness of karnaugh mapplet use in student learning of digital logic skills. In: **2005 Annual Conference**. Portland, Oregon: ASEE Conferences, 2005. Disponível em: <https://peer.asee.org/14557>. Acesso em: 7 set. 2015.

MUNARINI, B. L. *et al.* **Aplicativo para o ensino de eletrônica digital**. Universidade Tecnológica Federal do Paraná - Campus Campo Mourão - PR: [s.n.], 2014. Relatório de projeto.

NELSON, V. P. *et al.* **Digital logic circuit analysis and design**. 1. ed. Eaglewood Cliffs: Prentice Hall, 1995. ISBN 978-0134638942.

NICOLOSI, D. E. C. **Microcontrolador 8051 detalhado**. 8. ed. São Paulo: Érica, 2007. 227 p. ISBN 9788571947214.

NXP, S. **74HC4067; 74HCT4067 - 16-channel analog multiplexer/demultiplexer**. [S.l.]: NXP, Semiconductors, 2015. Disponível em: <http://www.nxp.com/documents/data_sheet/74HC_HCT4067.pdf>. Acessado em: 3 ago. 2016.

ON, S. **2N3906 - General Porpuse Transistor (PNP Silicon)**. [S.l.]: ON, Semiconductor, 2010. Disponível em: <http://www.onsemi.com/pub_link/Collateral/2N3906-D.PDF>. Acesso em: 15 jan. 2016.

____. **2N3903, 2N3904 - General Porpuse Transistor (PNP Silicon)**. [S.l.]: ON, Semiconductor, 2012. Disponível em: <http://www.onsemi.com/pub_link/Collateral/2N3903-D.PDF>. Acesso em: 15 jan. 2016.

PROLIFIC, T. I. **Pl-2303HX Edition (Chip Rev D) USB to Serial Bridge Controller Product Datasheet**. [S.l.]: Prolific, Technology Inc., 2013. Disponível em: <http://www.prolific.com.tw/userfiles/files/ds_pl2303hxd_v1_4_4.pdf>. Acesso em: 10 ago. 2016.

ROTH JR, C. H.; KINNEY, L. L. **Fundamentals of logic design**. 6. ed. Stanford, CT: Cengage Learning, 2010. 756 p. ISBN 9780495668046.

SEDRA, A. S.; SMITH, K. C. **Microeletrônica**. 5. ed. São Paulo, SP: Pearson Prentice Hall, 2007. 848 p. ISBN 9788576050223.

TEXAS, I. **LM340, LM340A and LM7805 Family Wide Vin 1.5-A Fixed Voltage Regulators**. [S.l.]: Texas Instruments, 2016. DS39617A. Disponível em: <http://www.ti.com/lit/ds/symlink/lm7805.pdf>. Acesso em: 2 ago. 2016.

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. **Sistemas digitais**: princípios e aplicações. 11. ed. São Paulo: Pearson Prentice Hall, 2011. 817 p. ISBN 9788576050957.

WILMSHURST, T. **Designing embedded systems with PIC microcontrollers**: principles and applications. 2. ed. Oxford, UK: Elsevier, 2010. 661 p. ISBN 9781856177504.

A ROTEIROS DE LABORATÓRIO

Roteiro 1

1. Para o circuito somador de 2 bits implementado no kit, Figura 90, obtenha:

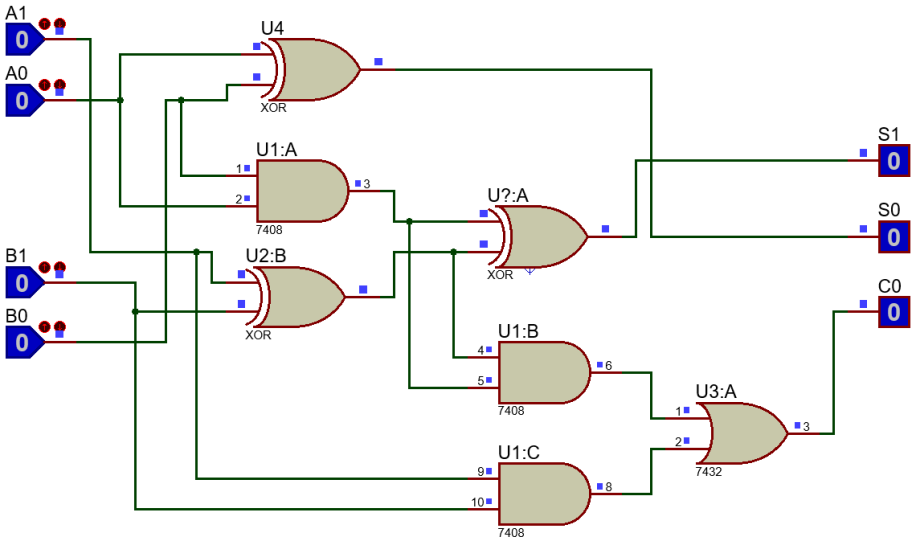

Figura 90 – Circuito somador de dois bits
Fonte: Autoria própria.

   a) Tabela Verdade para cada uma das saídas;
   b) Mapa de Karnaugh para cada uma das saídas;
   c) Expressão lógica para cada uma das saídas.

2. Compare a expressão retirada do circuito no formato de mintermos com o formato XOR.

3. Faça o teste com todas as combinações binárias de entrada do circuito (A1, A0, B1 e B0) e anote as respectivas saídas.

Roteiro 2

1. Obtenha a Tabela Verdade, Mapa de Karnaugh e Expressão lógica para um circuito subtrator de 2 bits.

2. Implemente o circuito obtido no kit utilizando os CIs da família 74XX, testando-os antes.

3. Compare a expressão extraída do circuito implementado com as seguintes expressões, sendo S0 o *bit* menos significativo do resultado, S1 o mais significativo e S2 o *carry out*:

$$\begin{aligned} S0 &= (DB')\oplus(A\oplus C) \\ S1 &= B\oplus D \\ S2 &= ((DB')(A\oplus C)') + (A'C) \end{aligned}$$

4. Verifique o funcionamento do circuito implementado comparando-o com todos os dados de projeto obtidos no Item 1.

B LISTA DE MATERIAL E ESTIMATIVA DOS CUSTO

Tabela 8 – Lista de material e estimativa de custos do kit

Componente	Preço por unidade	Nº de unidades	Valor Total
Resistores 10kΩ 0.25W	R$ 0,12	22	R$ 2,64
Resistores 2k6Ω 0.25W	R$ 0,12	1	R$ 0,12
Resistores 1kΩ 0.25W	R$ 0,12	11	R$ 1,32
Resistores 2k2Ω 0.25W	R$ 0,12	1	R$ 0,12
Resistores 300Ω 0.25W	R$ 0,12	6	R$ 0,72
Resistores 270Ω 0.25W	R$ 0,12	9	R$ 1,08
Resistores 160Ω 0.25W	R$ 0,12	2	R$ 0,24
Resistores 4k7Ω 0.25W	R$ 0,12	13	R$ 3,12
Regulador LM7805	R$ 1,20	1	R$ 0,12
Diodo 1N4007	R$ 0,05	1	R$ 0,05
DC Power Jack	R$ 0,50	1	R$ 0,50
Chave Push-Button	R$ 2,00	1	R$ 2,00
Capacitor cerâmico 100nF 50V	R$ 0,05	1	R$ 0,05
Capacitor cerâmico 22nF 50V	R$ 0,05	2	R$ 0,10
Capacitor Eletrolítico 100uF 35V	R$ 0,14	1	R$ 0,14
PPTC 800mA	R$ 3,50	1	R$ 3,50
Transistor PNP 2N3906	R$ 0,22	19	R$ 4,18
Transistor NPN 2N3904	R$ 0,22	13	R$ 2,86
Conversor USB - TTL RS232	R$ 12,50	1	R$ 12,50
Módulo Bluetooth HC-06	R$ 23,00	1	R$ 23,00
LED Vermelho Difuso 5mm	R$ 0,15	5	R$ 0,75
LED Verde Difuso 5mm	R$ 0,15	8	R$ 1,20
LED Azul Difuso 5mm	R$ 0,50	1	R$ 0,50
LED RGB 5mm	R$ 2,00	1	R$ 2,00
MUX/DEMUX 74HC4052	R$ 4,00	2	R$ 8,00
Shift-Register 74HC164	R$ 0,95	2	R$ 1,90
MUX/DEMUX 74HC4067	R$ 6,00	3	R$ 18,00
Soquete ZIF 14 pinos	R$ 3,00	14	R$ 42,00
Barra de pinos Fêmea	R$ 3,00	5	R$ 15,00
Protoboard 840 furos	R$ 20,00	1	R$ 20,00
Soquete 24 pinos largo	R$ 0,41	3	R$ 1,23
Soquete 14 pinos	R$ 0,28	2	R$ 0,56
Soquete 16 pinos	R$ 0,30	2	R$ 0,60
Soquete 40 pinos	R$ 0,65	1	R$ 0,65
Microcontrolador Atmega32A	R$ 24,00	1	R$ 24,00
Cristal oscilador de 16MHz	R$ 1,00	1	R$ 1,00
PCB dupla face pré-fabricada	R$ 150,00	1	R$ 150,00
		Total	R$ 345,75

Fonte: Autoria própria.

C ESQUEMÁTICO DO *HARDWARE*

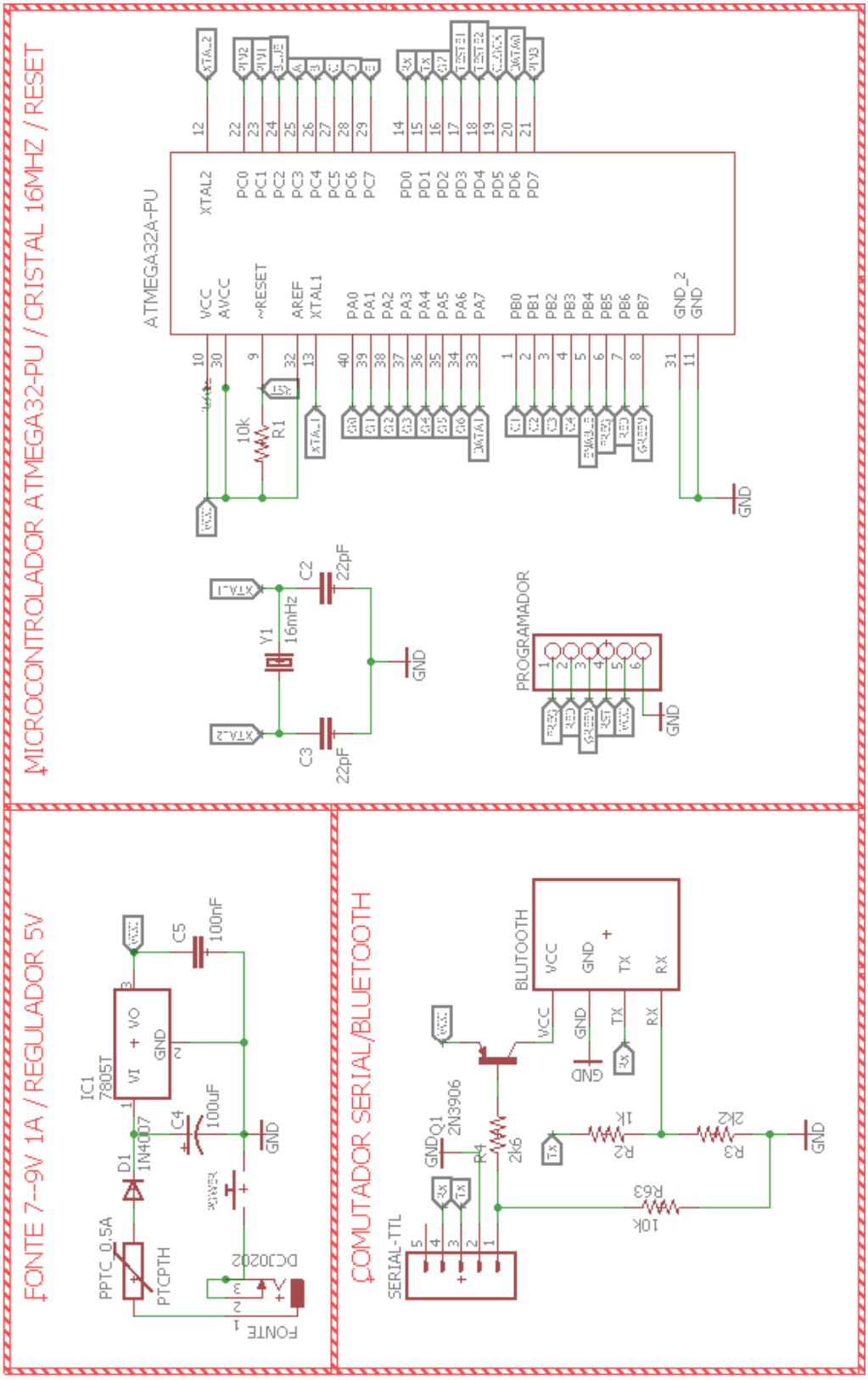

Figura 91 – Esquemático: Parte 1
Fonte: Autoria própria.

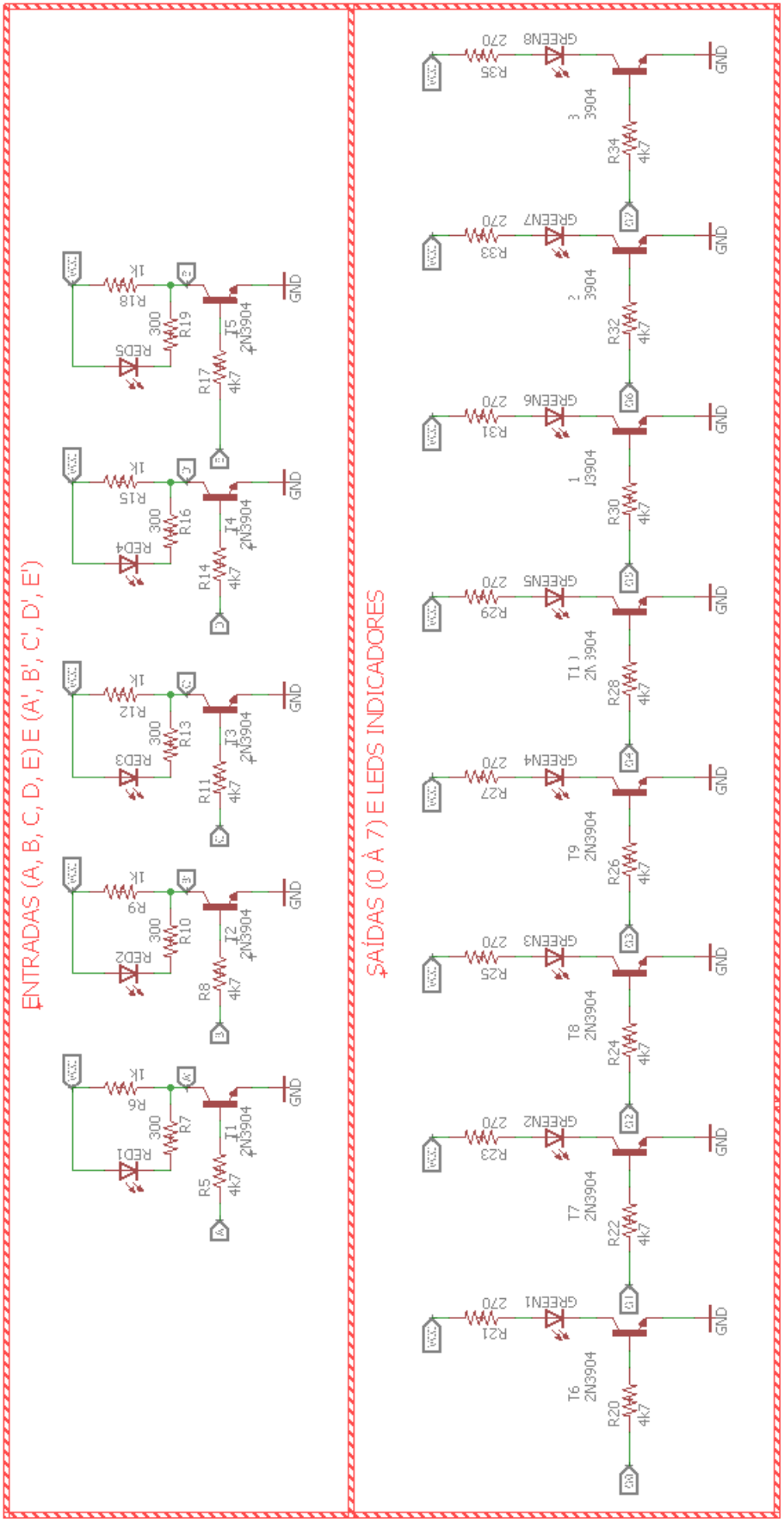

Figura 92 – Esquemático: Parte 2
Fonte: Autoria própria.

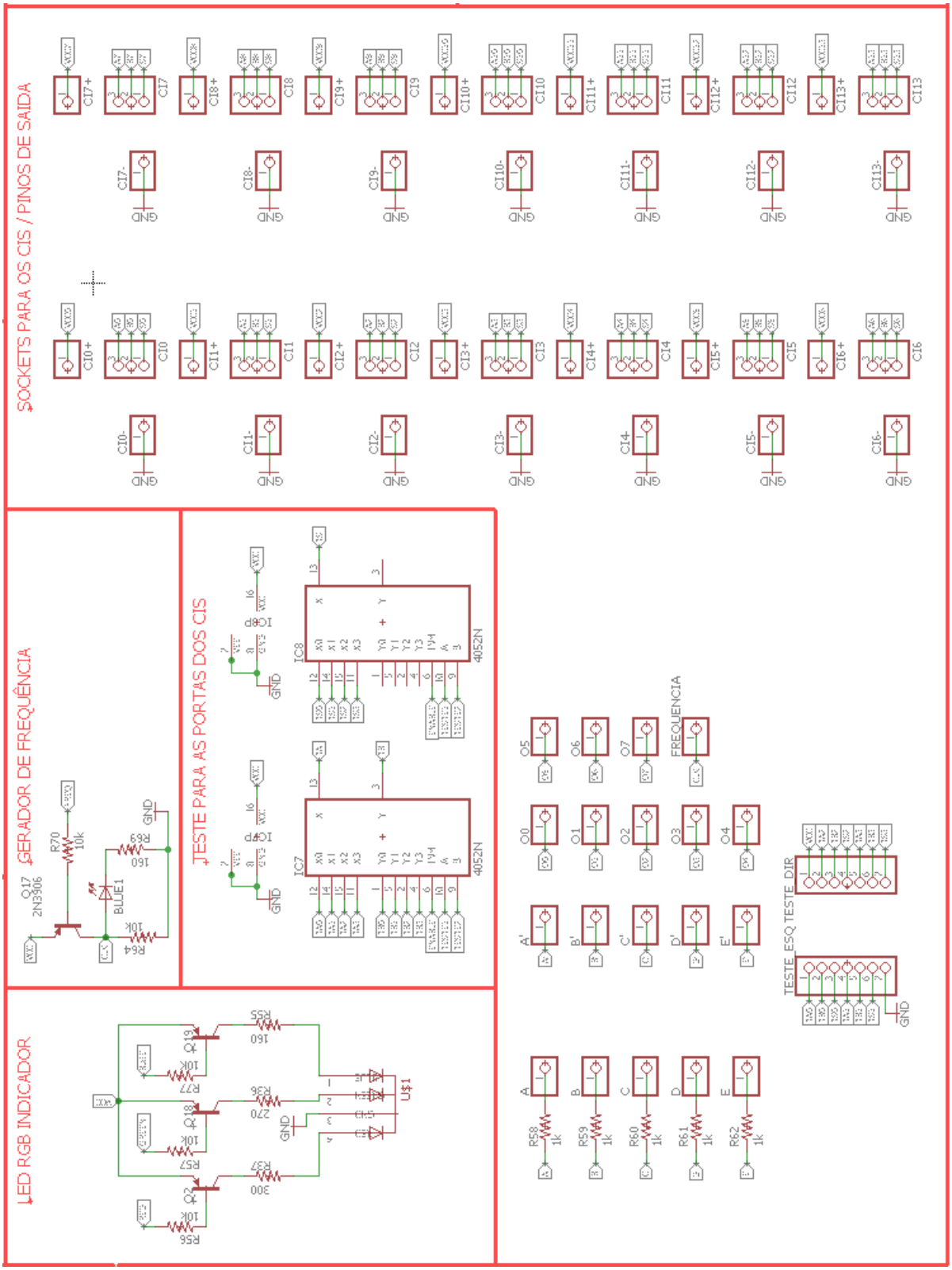

Figura 93 – Esquemático: Parte 3
Fonte: Autoria própria.

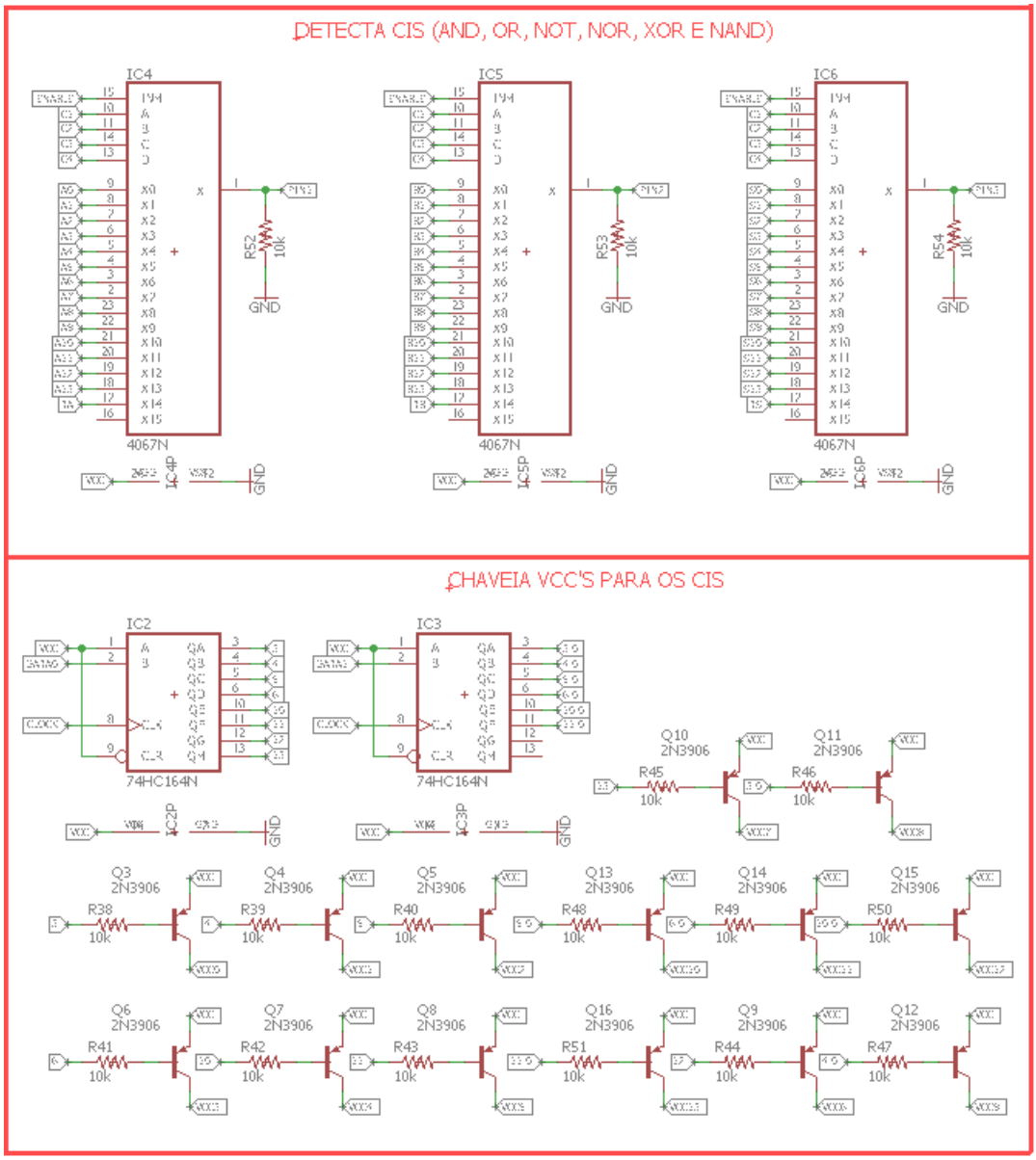

Figura 94 – Esquemático: Parte 4
Fonte: Autoria própria.

D PCB DO PROJETO DO *HARDWARE*

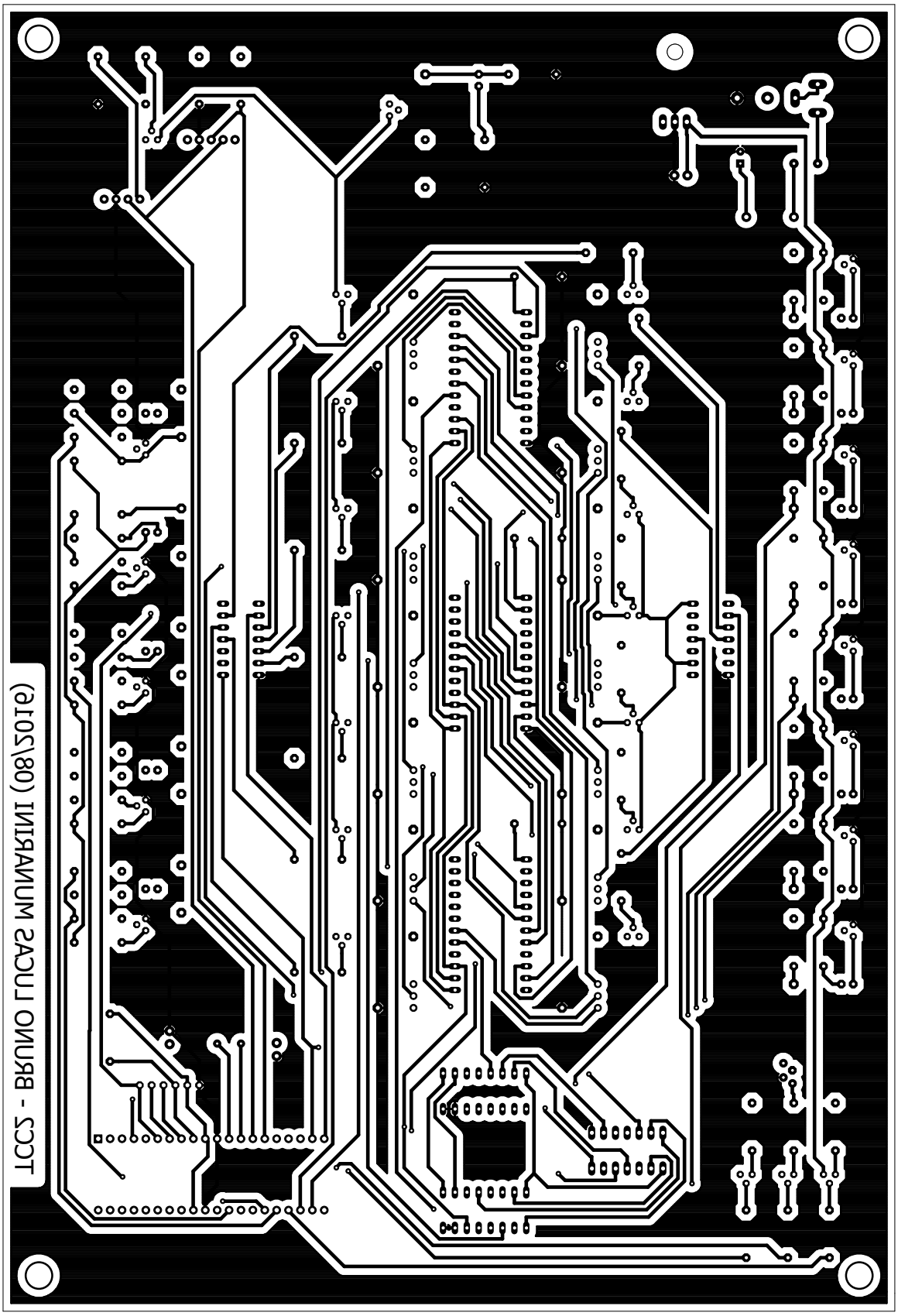

Figura 95 – Face: TOP
Fonte: Autoria própria.

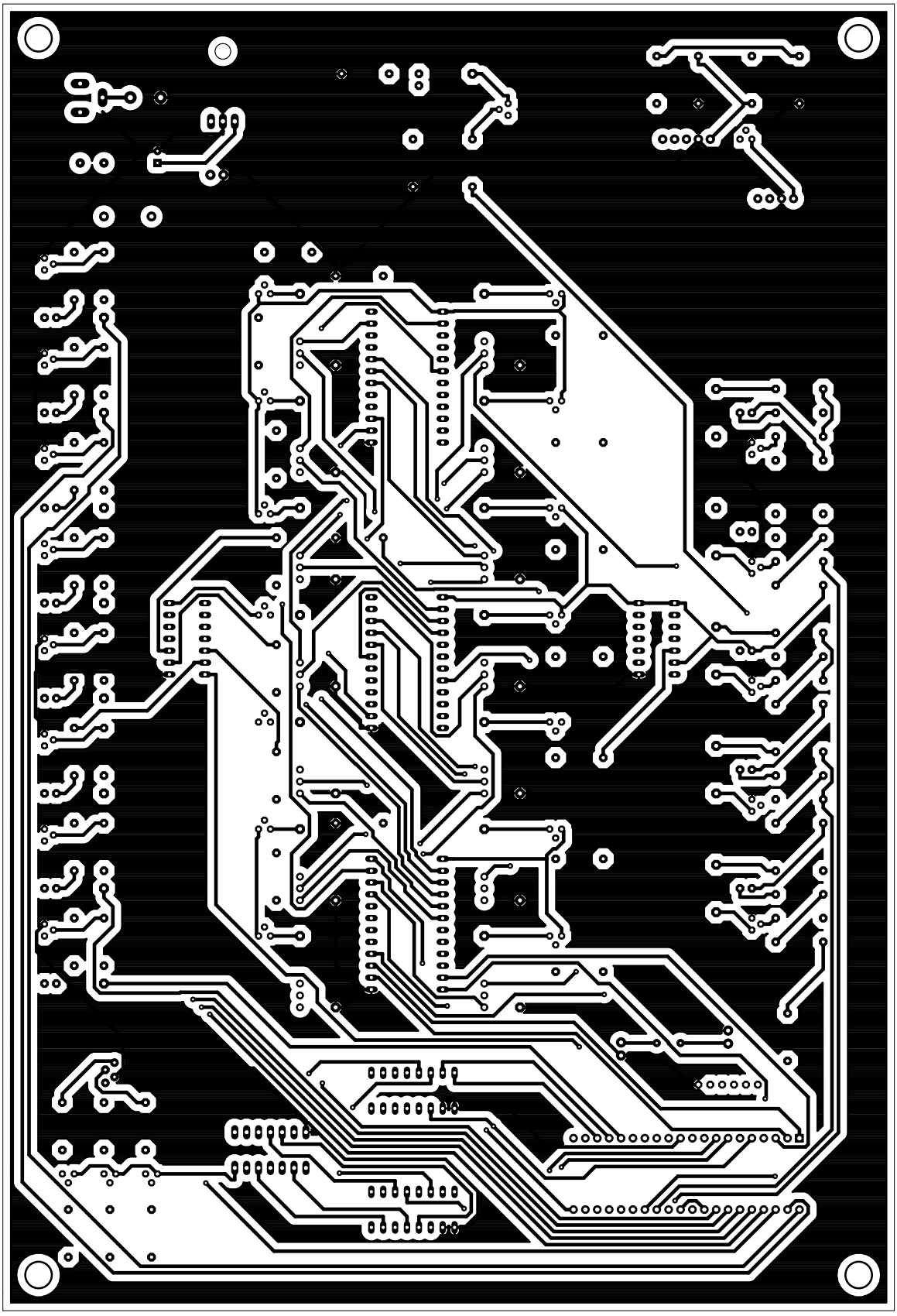

Figura 96 – Face: BOT
Fonte: Autoria própria.

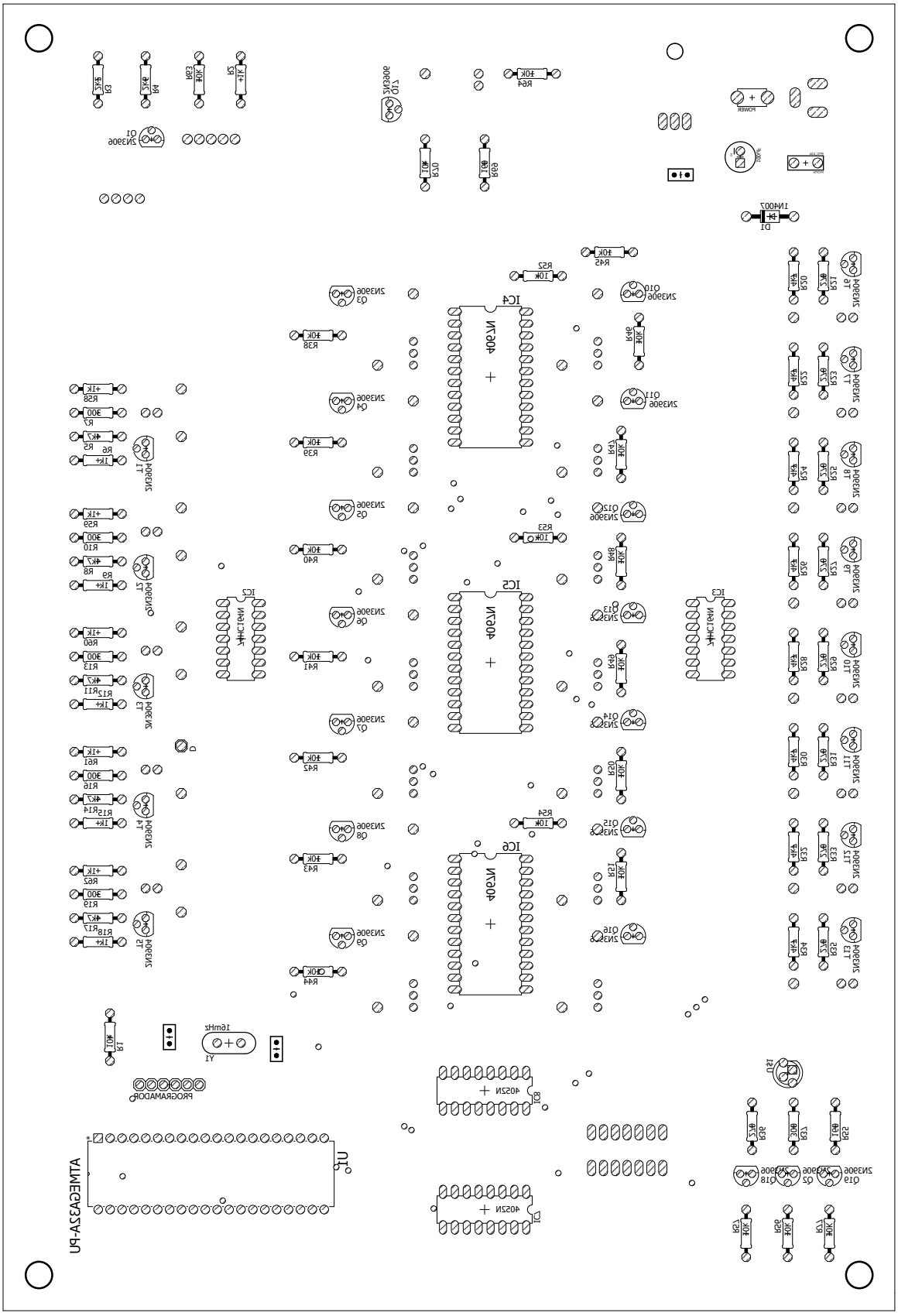

Figura 97 – Serigrafia: TOP
Fonte: Autoria própria.

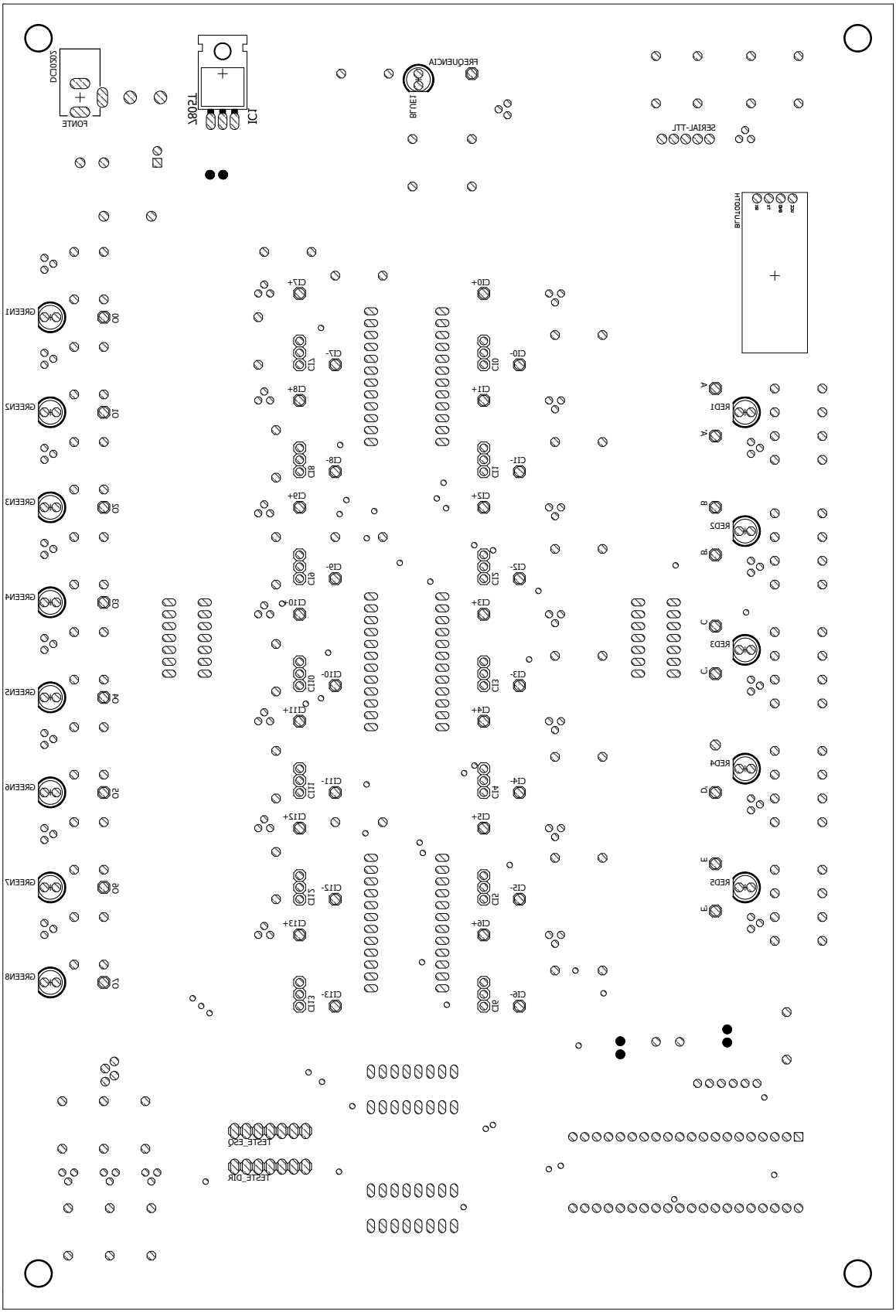

Figura 98 – Serigrafia: BOT
Fonte: Autoria própria.

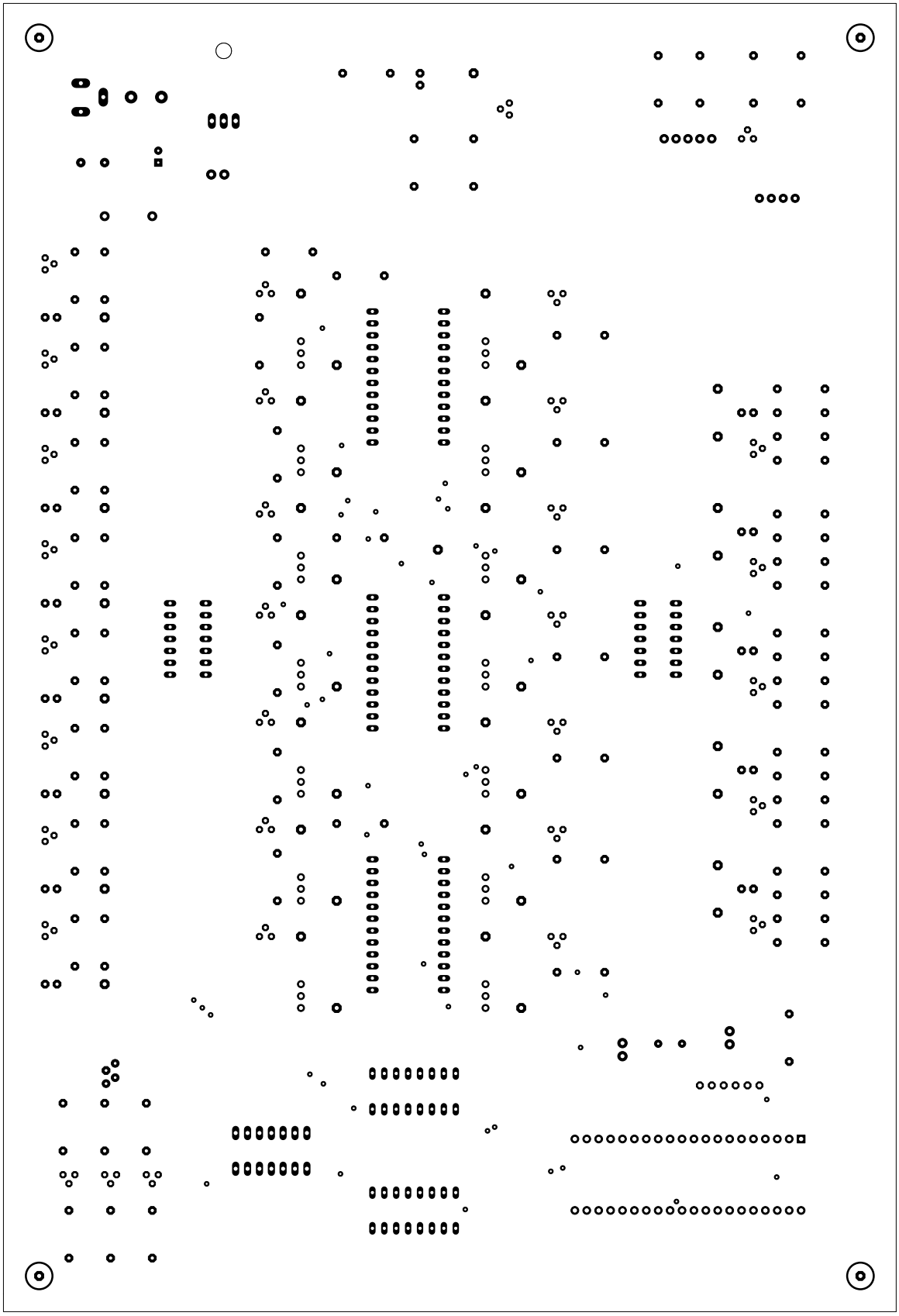

Figura 99 – Diagrama de Furação
Fonte: Autoria própria.