UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE ENGENHARIA
CÂMPUS DE ILHA SOLTEIRA

Alan Soares de Sousa

Aplicativo Android para Minimização de Equações Booleanas

Ilha Solteira
2021

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

Alan Soares de Sousa

Aplicativo Android para Minimização de Equações Booleanas

Trabalho de Graduação apresentado à Faculdade de Engenharia de Ilha Solteira – Unesp como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Nome do orientador
Profa. Dra. Suely Cunha Amaro Mantovani

Ilha Solteira
2021

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

S725a

Sousa , Alan Soares De .
Aplicativo android para minimização de equações booleanas / Alan Soares de Sousa . -- Ilha Solteira: [s.n.], 2021
74 f. : il.

Trabalho de conclusão de curso (Graduação em Engenharia Elétrica) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira, 2021

Orientador: Suely Cunha Amaro Mantovani
Inclui bibliografia

1. Aplicativos para smartphone. 2. Método de Quine-McCluskey. 3. Linguagem Python.

João Josué Barbosa
Serviço Técnico de Biblioteca e Documentação
Diretor Técnico
CRB 8-5642

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

ATA DE DEFESA DE TRABALHO DE GRADUAÇÃO

Aos vinte e dois dias do mês de dezembro do ano de dois mil e vinte e um, o discente **Alan Soares de Sousa,** matriculado sob o RA: 141051523, tendo como banca examinadora sua orientadora a Profa. *Dra. Suely Cunha Amaro Mantovani*, o Prof. Dr. *Alexandre Cesar Rodrigues da Silva* e o Prof. Dr Ricardo Tokio Higuti, apresentou o Trabalho de Graduação intitulado "**Aplicativo Android para Minimização de Equações Booleanas**", obtendo a nota 10 (dez) e conceito APROVADO

Alan Soares de Sousa
- discente -

Profa. Dra. Suely Cunha Amaro Mantovani
- orientadora -

Prof. Dr. Alexandre Cesar Rodrigues da Silva
- Membro da Banca -

Prof. Dr. Ricardo Tokio Higuti
- Membro da Banca -

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus pela vida e suas oportunidades.

A meus pais que sempre me apoiaram, me deram forças, estabilidade e um bom exemplo a seguir.

A Amanda de Melo Benedito que sempre está comigo para todos os momentos.

A *Zoran Slavulj* e *Robert Knuth*, com quem pude aprender muito durante o período de estágio.

Ao Prof. Dr. Alexandre César Rodrigues da Silva que me apresentou o método utilizado neste trabalho e desde a primeira aula sempre foi muito inspirador, mostrando que sim, engenheiros “resolvem problemas”.

A minha orientadora Profa. Dra. Suely Cunha Amaro Mantovani que se mostrou extremamente paciente, prestativa e compreensiva, sempre auxiliando em tudo o que foi possível.

A UNESP que me propiciou diversas oportunidades dentro e fora do país.

A todos o meu muito obrigado.

“A tarefa não é tanto ver aquilo que ninguém viu, mas pensar o que ninguém ainda pensou sobre aquilo que todo mundo vê.” Arthur Schopenhauer.

RESUMO

O grande avanço tecnológico em circuitos integrados e também na comunicação sem fio, nas últimas décadas, disponibiliza materiais e softwares que facilitam o controle e a simplificação de tarefas realizadas pelo usuário. A minimização de equações booleanas faz parte deste avanço, permitindo que circuitos digitais sejam cada vez mais otimizados, compactos e de baixo custo. Entre os métodos de minimização clássicos estão o Mapa de *Karnaugh*, e o algoritmo de *Quine-McCluskey*. Nesta pesquisa tem-se como proposta desenvolver um aplicativo para dispositivos móveis capaz de realizar a minimização de equações booleanas, baseado no método de minimização de *Quine-McCluskey.* Utiliza-se na implementação a linguagem de programação *Python* e algumas bibliotecas de seu ecossistema, como *Sympy*, *Kivy* e *KivyMD*. O aplicativo desenvolvido, chamado *Minimizer*, aceita até vinte e uma variáveis com entradas de informação por tabelas, equações, mintermos e *don't care*, além de realizar o desenho de circuitos, e também pode ser visualizada a etapa de Geração de Implicantes. Como resultado tem-se uma ferramenta de caráter didático e lúdico, motivadora para o estudo de minimização de equações booleanas em sistemas digitais.

Palavras-chave: Minimização de equações booleanas; Método de *Quine-McCluskey*; Aplicativos para *smartphone*; linguagem *Python*.

ABSTRACT

The great technological advance in integrated circuits and also in wireless communication, in the last decades, makes available materials and software that facilitate the control and simplification of tasks performed by the user. The minimization of Boolean equations is part of this advance, allowing digital circuits to be increasingly optimized, compact and low cost. Among the classical methods of minimization are the Karnaugh map and the Quine-McCluskey algorithm. This research aims to develop an application for mobile devices capable of performing the minimization of Boolean equations, based on the Quine-McCluskey minimization method. The implementation uses the Python programming language and some libraries from its ecosystem, such as Sympy, Kivy and KivyMD. The developed application, called Minimizer, accepts up to twenty-one variables with information input by tables, equations, minterms and don't care, besides being able to draw circuits, and also to visualize the Implicant Generation step. The result is a didactic and playful tool, motivating the study of minimization in digital systems.

Keywords: Boolean Equation Minimization; Quine-McCluskey Method; smartphone applications; Python language.

LISTA DE FIGURAS

LISTA DE TABELAS

Sumário

1 INTRODUÇÃO

O estudo de técnicas que possibilitem a redução de circuitos lógicos sempre foi de grande interesse dos desenvolvedores de circuitos integrados, possibilitou a evolução dos dispositivos eletrônicos em geral e principalmente os digitais e a sua ampla difusão, gerando dispositivos e equipamentos miniaturizados e com alto desempenho. Portanto, métodos de minimização de funções booleanas se tornam relevantes por possibilitarem a otimização de circuitos e a sua redução em circuitos integrados.

A minimização de funções booleanas possibilita a redução do número de literais (variável complementada ou não) e termos produto, na construção de um circuito elétrico, resultando na menor dissipação de calor e área ocupada, maior velocidade de resposta, colaborando para aumentar o seu desempenho. Um circuito lógico para ser considerado minimizado deve ter um menor número de entradas nas portas lógicas (FRANCISCANI, 2016; DAGHLIAN, 1995; TOCCI; WIDMER; MOSS, 2019).

Muitos métodos foram desenvolvidos para a minimização de funções booleanas, tais como o de Hlavicka e Fiser (2001), e Silva (1993), sendo a maioria deles baseados em métodos clássicos, como o Mapa de *Karnaugh*, desenvolvido por Maurice Karnaugh (1953) e o método de *Quine-McCluskey,* de Edward J. McCluskey (1956) citado por Nelson, Nagle, Irwin e Carroll (1995).

O Mapa de *Karnaugh*, método mais tradicional para minimização de funções booleanas, utiliza-se da ideia de tabelas. Parte do princípio no qual os mintermos (termo-produto, no qual cada variável de uma função aparece exatamente uma vez, complementada ou não) ou maxtermos (termo-soma, no qual cada variável aparece exatamente uma vez, complementada ou não) considerados podem ser representados em células de uma tabela, e neste método, não há necessidade do emprego de manipulações de expressões algébricas, depende unicamente da habilidade do usuário.

Quine (1952), no início da década de 50, propôs um método para encontrar formas normais simples disjuntivas de uma função verdade. Apresentou posteriormente, em outro trabalho, a utilização da operação de consenso entre dois termos adjacentes para gerar todos os implicantes primos

de uma função. Não há consideração de estados irrelevantes (*don't care states*) e foi utilizado apenas para funções simples.

A geração dos implicantes[1] de uma função booleana é uma das etapas do processo de minimização, no qual a função mínima obtida deve ser composta pelo conjunto de implicantes primos que possua o menor custo.

McCluskey (1956) se baseou no processo de minimização proposto por Quine para apresentar um método de geração de implicantes primos e de cobertura dos mintermos para obtenção da função mínima, com as etapas para a geração da tabela de cobertura para obtenção dos implicantes primos e formas alternativas para implementar o procedimento, sendo considerados os estados irrelevantes de uma função. Em 1961, McCluskey desenvolveu um método para minimizar funções booleanas com múltiplas saídas.

As pesquisas na área de minimização de funções booleanas foram realizadas até o final da década de 60. Depois disso, somente voltaram a ter importância na década de 80, com a criação dos dispositivos lógicos programáveis, denominados Matriz de Lógica Programável ou *Program Logic Array* (PLA), empregada na implementação de funções booleanas, na qual cada termo produto da função é implementado como uma linha na matriz e com múltiplas saídas. O PLA impulsionou o desenvolvimento de métodos para minimização de várias funções com múltiplas saídas, cujos implicantes primos compartilhados são gerados uma única vez, reduzindo o número de portas lógicas do circuito e o custo das funções resultantes.

Todo o problema de minimização tem seu custo computacional elevado com o aumento de variáveis de entrada, por isso, é de grande relevância o estudo relacionado às estruturas capazes de otimizar e minimizar o custo computacional dos métodos. Pela possibilidade de redução do circuito lógico, a análise e a otimização no processo de geração dos implicantes primos é um ponto crucial para melhor desempenho do processo.

Circuitos cada vez menores, mais eficientes e que consomem menos energia, são necessários, uma vez que seguem o avanço tecnológico, que

[1] Implicante é um termo produto obtido considerando-se o maior número possível de células adjacentes. Quando não há a possibilidade de reduzir um implicante, este é dito um implicante primo e caso um mintermo seja coberto por um único implicante primo, este é um implicante primo essencial.

utilizam a nanotecnologia no desenvolvimento de equipamentos . Por isso, os métodos de minimização de funções booleanas continuam relevantes por possibilitarem a otimização de circuitos lógicos, com a mesma funcionalidade, porém minimizados e adaptados às novas tecnologias.

Atualmente, encontram-se na *internet*, em lojas de aplicativos para *smartphone*, alguns aplicativos como o BooleanTT e o Karnaugh Kmap Solver, que simplificam e minimizam expressões, resolvem Mapa de *Karnaugh*, simulam e geram circuitos lógicos. Possuem baixos requisitos de *hardware*, mas são poderosos, todavia algumas funções são bloqueadas ou limitadas à versão paga de seus respectivos *softwares.*

Pelo exposto, tem-se como proposta desenvolver um aplicativo para smartphone, utilizando o método de *Quine-McCluskey* e a linguagem *Python*.

1.1 OBJETIVOS

Tem-se como proposta desenvolver um aplicativo para dispositivo móvel do sistema *Android* para minimizar equações booleanas, utilizando o método de Quine-McCluskey, bem como evidenciar os métodos de resolução para uso didático, em cursos de sistemas digitais, utilizando a linguagem de programação *Python* e seus recursos. No seu desenvolvimento têm-se as etapas de programação da minimização de funções, a realização do aplicativo para o dispositivo móvel e a interface, visando torná-lo prático, atraente e lúdico ao usuário. Com esta pesquisa consolidam-se os conhecimentos obtidos nas disciplinas, durante o curso da graduação, e o aprendizado em temas relevantes e atuais visando gerar um produto.

1.2 MOTIVAÇÃO

A motivação para a proposta deste trabalho se originou da experiência vivenciada em sala de aula durante as aulas de Circuitos Digitais, em um curso técnico e no início do curso de Engenharia Elétrica, na graduação. Na época foi notada a dificuldade de diversos colegas de turma, em compreender a

simplificação de equações booleanas utilizando-se do método do Mapa de *Karnaugh*, principalmente, para um grande número de variáveis e alguns questionavam se seria possível a simplificação de circuitos complexos com mais de 10 variáveis, por exemplo. Com esta proposta cria-se a possibilidade de mostrar aos alunos como automatizar a simplificação de funções booleanas, além de servir como um guia de comparação de resultados obtidos empregando outros métodos. Com a realização deste aplicativo para celular tem-se também a oportunidade de obter um maior aprendizado na linguagem *Python*.

1.3 ORGANIZAÇÃO DO TEXTO

Além desta introdução, no capítulo 2 é feita uma revisão de literatura, contendo alguns artigos relacionados a métodos de simplificação de equações booleanas visando fornecer os subsídios para o entendimento desta pesquisa. Para situar o trabalho no mercado de aplicativos e servir de comparação, apresentam-se também alguns aplicativos para dispositivo móveis que simplificam e minimizam equações booleanas disponíveis na internet.

No capítulo 3 como fundamentação teórica são abordados os conceitos básicos da álgebra booleana, como notações e teoremas da área. Em seguida, apresentam-se os métodos de simplificação usando o Mapa de *Karnaugh* e o algoritmo de Quine-McCluskey, com estudos de casos.

Tem-se no capítulo 4 uma introdução à linguagem *Python*, bem como as bibliotecas utilizadas no desenvolvimento deste trabalho, entre outras ferramentas. Finalizando este capítulo, são discutidas algumas características e atributos da linguagem, e apresentados casos de uso da linguagem em aplicativos famosos do cotidiano utilizados para diversas finalidades.

No capítulo 5 descreve-se a implementação do aplicativo usando um diagrama de blocos, a programação e o interfaceamento do algoritmo do aplicativo, e algumas das funções utilizadas em sua concepção.

Apresentam-se no capítulo 6 as telas do aplicativo desenvolvido em figuras explicativas, com os tipos de entradas no modo tabela, por mintermos, equações, ou circuito e os detalhes da simplificação pelo método de Quine-

McCluskey que podem ser utilizadas na minimização. Também neste capítulo têm-se as discussões sobre o desenvolvimento e os resultados em gráficos do desempenho do aplicativo, em situações de stress.

Por fim, têm-se as conclusões abordando a concepção e o desenvolvimento, a análise dos resultados, bem como propostas de aperfeiçoamento para trabalhos futuros, seguido das referências e um Apêndice.

2 REVISÃO DE LITERATURA

Muitos são os trabalhos encontrados na literatura que abordam a minimização de funções booleanas, uns mais complexos como em Silva (1993) e Franciscani (2016) e outros mais simples de interesse deste trabalho que ~~devem~~ fornecem os subsídios necessários para o seu desenvolvimento. Neste tópico são apresentadas algumas dessas referências sobre os temas de minimização de funções booleanas e também sobre alguns aplicativos que existem nas lojas de aplicativos para dispositivos móveis.

Os estudos relacionados à área de minimização de equações booleanas se iniciaram faz muito tempo, por isso existem diversos métodos já consolidados, alguns deles baseando-se em propostas de melhoramento em relação ao método de Quine, e do próprio *Quine-McCluskey* utilizado como base neste projeto.

O método de Quine-McCluskey datado de 1956 tinha como objetivo a minimização para a geração dos implicantes primos de uma função booleana. Consiste de um método tabular e iterativo clássico, muito utilizado na literatura como base ou comparativo a outros procedimentos. Apresenta duas fases, sendo a primeira, a geração de implicantes primos e, a segunda, a cobertura dos mintermos da função. Este método não utiliza nenhuma heurística, somente o teorema $A\,B + \bar{A}\,B = B$ repetidamente, entre os termos da função. Os termos da função são classificados em caixas, de acordo com a quantidade de dígitos “1′s” que possui e inseridos no grupo inicial. Compara todos os termos pertencentes a uma caixa, com todos os termos pertencentes à próxima caixa. Os implicantes gerados são armazenados em um novo grupo e todo o processo é repetido até que não seja gerado mais implicante, ou que em um novo grupo contenha apenas uma caixa. Após a geração dos implicantes primos, a fase de cobertura dos mintermos é realizada e a função mínima é obtida.

Em Tison (1967), para encontrar os implicantes primos, propôs estender o processo da operação de consenso ao utilizar um maior número de termos, não somente a dois, este seu método pode ser aplicado em funções de saída simples, múltiplas, com ou sem *don’t care states*.

Slagle, Chang e Lee (1970), propuseram um algoritmo diferente, com o objetivo de não gerar um mesmo implicante primo mais de uma vez, reduzindo o número de operações realizadas e assim, sendo mais eficiente. Neste é gerada uma lista de prioridades, onde se analisa a frequência em que determinada variável é selecionada, e desta forma as mais relevantes sofrem o processo de expansão, a fim de gerar implicantes.

Brayton (1984) apresentou o método Espresso-II, com o objetivo de aperfeiçoar o método de Quine-McCluskey, contornando suas limitações. Este método trabalha com dois conceitos, a expansão e a redução, sendo a primeira o ato de ampliar todos os implicantes gerados, removendo aqueles cobertos pelo implicante expandido. O segundo é a redução de um implicante ao menor número de literais possível, ainda cobrindo a função original. De forma iterativa estes dois conceitos são aplicados até que se encontre a melhor cobertura possível.

Kuo e Chou (1986) introduziram a ideia de consenso assimétrico em seu algoritmo, onde seria possível gerar todos os implicantes primos essenciais (único implicante capaz de cobrir determinado mintermo) sem realizar a cobertura dos mintermos pertencentes à função booleana. Para tal, foi implementado um método de verificação de redundâncias, cujos implicantes, obtidos em uma primeira etapa, são testados, chamados inicialmente de primos parcialmente essenciais, passam então, caso aptos, a serem implicantes primos essenciais.

Em Silva (1993) é desenvolvido uma técnica também utilizando a operação de consenso iterativo entre dois termos de uma função booleana. O método chamado GeraPlex representa os termos em uma estrutura de árvore, sendo cada caminho relacionado a um termo da função. Com ele pode-se gerar todos os implicantes primos de uma função. É possível encontrar três situações na geração de novos termos: a Fusão (termo gerado cobre os dois termos originais, eliminando-os da árvore), o Deslocamento (termo gerado cobre um termo original, sendo este eliminado da árvore) e a Expansão (termo gerado não cobre os termos originais).

Em Franciscani (2016) é realizado um estudo da comparação de eficiência na implementação dos métodos de *Quine McCluskey* para múltiplas saídas, GPMultiplo e MultiGeraPlex (baseados no algoritmo GeraPlex),

comprovando que os métodos baseados no trabalho de Silva são um aperfeiçoamento e podem ser mais eficientes e viáveis, uma vez que demandam um menor tempo de execução e uso de memória ao fazer um menor número de comparações e gerar um número menor de implicantes se comparados a primeira etapa do método de *Quine-McCluskey*.

Como o objetivo do trabalho é implementar a minimização de equações booleanas em um aplicativo para dispositivos móveis, foi realizada uma pesquisa quanto aos aplicativos disponíveis na *PlayStore* com a mesma finalidade, observando-se a funcionalidade, apresentação, gratuitos ou não, o método utilizado para a minimização e a linguagem empregada em seu desenvolvimento. Desta pesquisa foram selecionados dois deles , o BooleanTT e o Karnaugh Kmap Solver.

Criado por Hashan Chamara Rajapaksha, o aplicativo BooleanTT não necessita de *hardware* potente para o seu funcionamento (considerado um aplicativo leve), fornece a plotagem de circuitos e minimizações de funções booleanas, possuindo opções gratuitas e pagas para uso didático. De forma diferenciada permite a resolução de expressões via Mapa de *Karnaugh* e possui calculadora utilizando bases não convencionais, como binários, octais e hexadecimais. O *software* fornece opções exclusivas para uso pago, como um teclado diferenciado com mais opções de entrada (função limitada na versão gratuita) e outros recursos.

Desenvolvido por Adriano Moutinho, o Karnaugh Kmap Solver é um aplicativo capaz de resolver equações booleanas utilizando o Mapa de *Karnaugh* para duas, três, quatro ou cinco variáveis, e todas as possíveis soluções para o mapa de entrada. Além disso, o aplicativo mostra versões diferentes de um circuito lógico otimizado, sendo a versão tradicional, outra com inversores comuns e a versão com um circuito único utilizando portas NAND e NOR.

Pelo exposto, observa-se que os estudos sobre métodos de minimização de funções booleanas continuam a ser realizados na busca de descobrir teorias mais eficazes, sendo motivados pelas novas teorias relacionadas à lógica reversível, a qual promete a construção de circuitos com melhor eficiência térmica, demonstrando sua relevância nos dias atuais.

Enquanto com relação aos aplicativos, é inegável que existe um forte interesse do mercado na criação de aplicativos, procurando atender ao usuário, facilitando os serviços, fidelizando o cliente, e motivando o domínio de linguagens e das técnicas de criação, por parte do desenvolvedor.

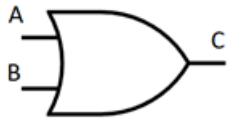
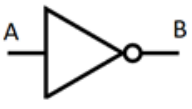
3 FUNDAMENTAÇÃO TEÓRICA

Neste tópico são apresentados alguns dos conceitos que foram utilizados no trabalho que envolve a construção do algoritmo de minimização, como a Álgebra booleana, a representação de funções e os métodos de minimização, tais como, o Mapa de *Karnaugh* e o método de *Quine-McCluskey* este usado no desenvolvimento do aplicativo móvel.

3.1 ÁLGEBRA BOOLEANA

A álgebra booleana consiste de um conjunto finito de termos, sendo dois operadores binários, um operador unário e as constantes ou variáveis que podem ter apenas dois valores possíveis, 0 e 1, conforme ilustrado na Figura 1, e as operações básicas utilizando esses operadores que são apresentadas nas Tabelas 1 e 2.

Figura 1 - Representação dos operadores lógicos básicos

Operadores **BINÁRIOS**	Correspondência	Representação gráfica
+	OR (OU)	
●	AND (E)	
UNÁRIOS		
—	NOT (NÃO)	
VARIÁVEIS E CONSTANTES	0 ou 1 (Falso ou Verdadeiro)	

Fonte: Elaborado pelo autor

Tabela 1 - Operação lógica NOT

A	Ā
0	1
1	0

Fonte: Elaborado pelo autor

Tabela 2 - Operações lógicas OR e AND

Variáveis		**Operação OR**	**Operação AND**
A	B	A+B	A.B
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Fonte: Elaborado pelo autor

Para auxiliar na minimização das equações booleanas, existem, assim como na álgebra tradicional, alguns teoremas booleanos que podem ser facilmente aplicados. São mostrados na Tabela 3 os teoremas que são aplicados quando há uma única variável, A, podendo esta assumir os valores 0 ou 1 (TOCCI; WIDMER; MOSS, 2019).

Tabela 3 - Teoremas booleanos com variável única

A . 0 = 0	**A + 0 = A**
A . 1 = A	**A + 1 = 1**
A . A = A	**A + A = A**
A . Ā = 0	**A + Ā = 1**

Fonte: Elaborado pelo autor

A Álgebra de Boole é regida pelas leis comutativa, associativa e distributiva, resumidas na Tabela 4, como sendo:

- **Lei comutativa:** mostra que a ordem em que as variáveis aparecem não importa;
- **Lei associativa:** mostra que é possível realizar o agrupamento de variáveis;

- **Lei distributiva:** semelhante à álgebra convencional, permite a expansão de uma expressão por meio da multiplicação termo a termo, ou ainda selecionar termos em evidência para possibilitar a fatoração.

Tabela 4 - Leis da Álgebra de Boole

Lei	Exemplo
Comutativa	$\mathrm{A} + \mathrm{B} = \mathrm{B} + \mathrm{A}$
	$\mathrm{A} \,.\, \mathrm{B} = \mathrm{B} \,.\, \mathrm{A}$
Associativa	$(\mathrm{A} + \mathrm{B}) + \mathrm{C} = \mathrm{A} + (\mathrm{B} + \mathrm{C})$
	$(\mathrm{A} \,.\, \mathrm{B}) \,.\, \mathrm{C} = \mathrm{A} \,.\, (\mathrm{B} \,.\, \mathrm{C})$
Distributiva	$\mathrm{A} \,.\, (\mathrm{B} + \mathrm{C}) = \mathrm{A} \,.\, \mathrm{B} + \mathrm{A} \,.\, \mathrm{C}$
	$\mathrm{A} + (\mathrm{B} \,.\, \mathrm{C}) = (\mathrm{A} + \mathrm{B}) \,.\, (\mathrm{A} + \mathrm{C})$

Fonte: Elaborado pelo autor

Os teoremas booleanos em situações que envolvem mais de uma variável são apresentados na Tabela 5. Os teoremas das duas últimas linhas da tabela são contribuições do grande matemático chamado de DeMorgan, por isso, recebem o nome de teoremas de DeMorgan bastante úteis nas simplificações de expressões booleanas.

Tabela 5 - Teoremas booleanos com mais de uma variável

$(A \,.\, B) + (\bar{\mathrm{A}} \,.\, B) = B$
$(A + B) \,.\, (\bar{\mathrm{A}} + B) = B$
$A + A \,.\, B = A$
$A \,.\, (A + B) = A$
$A + (\bar{\mathrm{A}} \,.\, B) = A + B$
$A \,.\, (\bar{\mathrm{A}} + B) = A \,.\, B$
$\overline{\overline{A}} = A$
$(A + B) \,.\, (A + C) = A + B \,.\, C$
$\overline{(A + B)} = \overline{A} \,.\, \overline{B}$
$\overline{(\mathrm{A} \,.\, \mathrm{B})} = \overline{\mathrm{A}} + \overline{\mathrm{B}}$

Fonte: Elaborado pelo autor

Ilustra-se na Figura 2, a utilização de alguns dos teoremas presentes nas Tabelas 3 e 5 na minimização de uma função qualquer.

Figura 2 - Simplificação de uma função usando álgebra booleana

$F(A,B) = (A\,.\,B) + \left(A.\overline{B}\right) + \overline{(A.B)} \rightarrow$	Aplica-se: $(A\,.\,B) + \left(A.\overline{B}\right) = A$
$F(A,B) = A + \overline{(A.B)} \rightarrow$	Segundo DeMorgan: $\overline{(A.B)} = \overline{A} + \overline{B}$
$F(A,B) = A + \overline{A} + \overline{B} \rightarrow$	Pelo complemento: $A + \overline{A} = 1$
$F(A,B) = 1 + \overline{B} \rightarrow$	Por fim: $1 + A = 1$
$F(A,B) = 1$	

Fonte: Elaborado pelo autor

3.2 REPRESENTAÇÃO DE FUNÇÕES BOOLEANAS

Funções booleanas são dependentes da combinação das variáveis de entrada e dos operadores para que seja produzida uma saída. Uma das formas de representação de uma função booleana é a tabela verdade, onde cada linha representa uma das 2^n combinações possível, das variáveis de entrada, por exemplo, considere a Função (1),

$$F(A,B,C) = (A + B).C \qquad (1)$$

Para 3 variáveis de entrada , existirão 2^3 combinações, sendo assim tem-se a tabela verdade apresentada na Tabela 6.

Tabela 6 - Tabela verdade para a função (1)

A	B	C	F(A,B,C)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Fonte: Elaborado pelo autor

Há outras formas de se representar uma equação booleana, como a forma canônica, onde a função $F(x)$ pode ser expressa por uma soma de produtos ou produto de somas de todas as variáveis. Portanto, a função da Tabela 6 pode ser reescrita, utilizando-se das linhas referentes às variáveis que tem saída '1', resultando na Expressão (2),

$$F(A,B,C) = \overline{A}\,.B\,.C + A\,.\overline{\mathrm{B}}\,.C + A\,.B\,.C = \overline{A}BC + A\overline{B}C + ABC \qquad (2)$$

O produto canônico, também chamado de mintermo, é o termo produto resultante da combinação das variáveis, cujo resultado da função é 1. No exemplo dado, têm-se três mintermos, $\overline{A}BC$, $A\overline{\mathrm{B}}C$ e ABC, cuja representação pode ser feita pela somatória dos mintermos em decimal, conforme visto na Expressão (3), onde m é o conjunto dos mintermos (FRANCISCANI, 2016),

$$F(A,B,C) = \sum m(3,5,7) \qquad (3)$$

Existem ainda funções que possuem combinações de entrada irrelevantes ao funcionamento final do projeto, chamadas de *don't care state*, comumente representadas na tabela verdade pela letra X e seu conjunto é representado pela letra D, como no exemplo da Expressão (4), cuja tabela verdade é dada na Tabela 7, onde os termos $\overline{A}\,\overline{B}\,C$ e $A\,B\overline{C}$ são mintermos, cujas saídas são irrelevantes ao projeto.

$$F(A,B,C) = \sum m(3,5,7) + D(1,6) \qquad (4)$$

Tabela 7 - Tabela verdade para a expressão (4)

A	B	C	F(A,B,C)
0	0	0	0
0	0	1	X
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	X
1	1	1	1

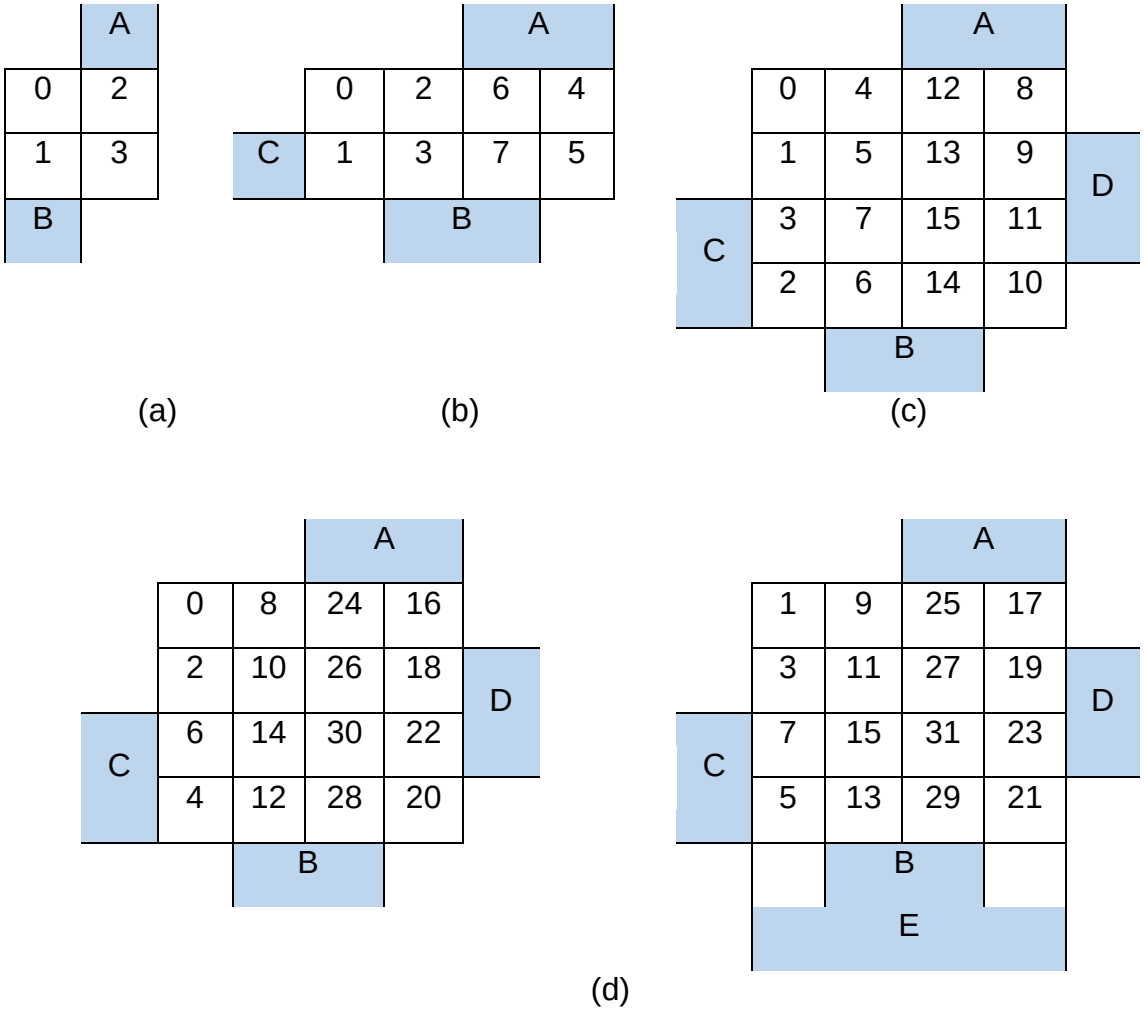
Fonte: Elaborado pelo autor

3.3 MINIMIZAÇÃO DE FUNÇÕES POR MAPA DE KARNAUGH

Criado por Edward Veitch (1952), citado em Silva (1993), e aperfeiçoado posteriormente por Maurice Karnaugh (1953), este é um método gráfico de simplificação de expressões booleanas que se baseia em visualizar grupos de mintermos, em uma representação tabular, para que a expressão possa ser simplificada de maneira relativamente simples (quando com poucas variáveis).

Para a construção do mapa se considera inicialmente o número de variáveis, n, então, haverá 2^n células, com um padrão de significância. É possível deduzir que há um aumento exponencial no número de células, e consequentemente, na complexidade da aplicação do método, conforme se aumenta o número de variáveis. Nos exemplos a seguir, toma-se cinco variáveis (A, B, C, D), sendo a variável A considerada a mais significativa do termo, desta forma, na Figura 3, representa-se o Mapa de *Karnaugh* (MK) para duas, três, quatro e cinco variáveis.

Figura 3 – MK para: a) Duas variáveis. b) Três variáveis. c) Quatro variáveis. d) Cinco variáveis



Fonte: Elaborado pelo autor

O Mapa de *Karnaugh* é montado ao se inserir o valor de 1 ou 0 nas células que representam os termos desejados (mintermo ou maxtermo, respectivamente), pode-se ainda atribuir X a uma célula, em caso de *don't care*.

A simplificação é realizada quando se agrupa células adjacentes, sendo estas divergentes em apenas um bit. É importante ressaltar que o mapa deve ser interpretado de maneira cilíndrica, ou seja, células da última linha são adjacentes às da primeira linha, enquanto a coluna mais à esquerda é adjacente da coluna mais à direita. Além disso, deve-se sempre buscar o maior agrupamento possível de células, até que todos os termos sejam cobertos,

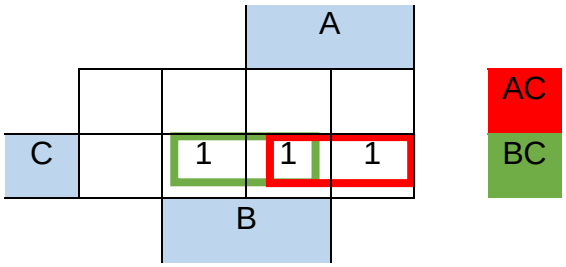
caso haja *don't cares*, estes podem ser úteis para possibilitar agrupamentos maiores, minimizando ainda mais o circuito lógico.

Para ilustrar o método de minimização por MK são apresentadas nas Figuras 4 e 5, dois exemplos de simplificação, Expressões (3) e (4),

$$F(A,B,C) = \sum m(3,5,7) \tag{3}$$

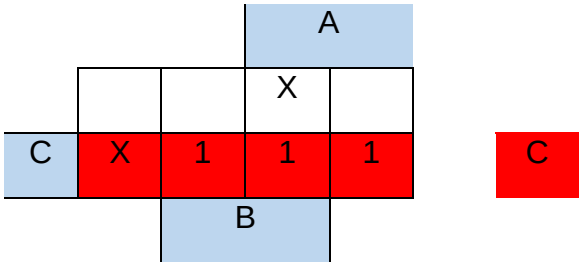
$$F(A,B,C) = \sum m(3,5,7) + D(1,6) \tag{4}$$

Figura 4 - Mapa de *Karnaugh* da função (3)



Fonte: Elaborado pelo autor

Figura 5 - Mapa de *Karnaugh* da função (4)



Fonte: Elaborado pelo autor

Na Figura 4, observa-se a seleção de dois agrupamentos, representados por AC *e* BC, estes cobrem todos os mintermos desejados, sendo, portanto, os implicantes primos a ser selecionados para a função de custo mínimo, que se dará por $AC + BC$.

Na Figura 5, observa-se a presença de *don't care states* que facilitam na obtenção de um agrupamento maior, gerando neste caso o implicante primo C, que sozinho cobre todos os mintermos desejados, representando o mínimo custo da função.

Os agrupamentos gerados a partir da combinação das células são chamados implicantes, e a composição destes fundamentará a resolução de custo mínimo, onde se considerou neste trabalho como custo mínimo a somatória ponderada de entradas e portas lógicas da função, seguindo os seguintes critérios de seleção:

- Em caso de agrupamento formado por apenas um elemento (termo isolado), este é um implicante primo e deverá obrigatoriamente pertencer a função mínima, pois somente ele pode cobrir determinado mintermo;
- Agrupamentos que não estão contidos em nenhum outro fazem parte da função mínima;
- Em caso de diferentes agrupamentos que possam cobrir os mesmos termos sem necessariamente estarem contidos um no outro, deve-se escolher de acordo com o número de variáveis, e o agrupamento com menos variáveis deve ser incluído na função mínima. Caso sejam agrupamentos de mesmo tamanho, a escolha pode ser aleatória.

O Mapa de *Karnaugh* cresce exponencialmente com o número de variáveis, por isso, a partir de seis variáveis, métodos iterativos como o de *Quine-McCluskey* a ser apresentado na seção 3.4, pode se tornar uma alternativa mais interessante.

3.4 MÉTODO DE MINIMIZAÇÃO POR QUINE-MCCLUSKEY

O método de *Quine-McCluskey* é recomendado para minimização de equações booleanas, principalmente quando é aplicado à programação, de forma automatizada, como é o caso desta pesquisa, desta forma, buscando fornecer os subsídios necessários para o entendimento do algoritmo de minimização implementado no aplicativo, faz-se uma breve introdução teórica baseada nos trabalhos de Nelson, Nagle, Irwin e Carroll (1995).

Desenvolvido em 1956, pelo trabalho conjunto, inicialmente de Willard V. Quine (professor da Universidade de *Harvard*) e estendido por Edward J.

McCluskey (professor da Universidade de *Stanford*), com o método de *Quine-McCluskey* buscavam realizar a minimização de funções booleanas de um modo mais simples de se implementar computacionalmente, do que o método formalizado até então, o chamado Mapa de *Karnaugh*.

O método de *Quine-McCluskey* é considerado simples, porém trabalhoso de se realizar quando o número de variáveis é muito grande, devido ao seu caráter iterativo. Consiste nos seguintes passos:

a) Agrupar todos os mintermos e *don't care* em “caixas”, onde o índice da caixa é o número de “1’s” na notação binária do mintermo:
   - Por exemplo: mintermo 0 $(0000)_b$ pertence a caixa 0, enquanto o mintermo 3 $(0011)_b$, pertence a caixa 2.
b) Comparar cada mintermo da caixa de índice i com cada mintermo da caixa $i+1$, caso diferirem em apenas um *bit*, ambos os mintermos devem ser “marcados” como já combinados, e essa combinação resultará em um implicante, dada pela operação lógica *AND* entre os mintermos, com o único valor diferente denotado como *don’t care* (no caso, usando a notação “ X”):
   - Por exemplo: a combinação entre o mintermo 3 $(0011)_b$ e 7 $(0111)_b$ resulta no implicante 0X11, no entanto o mintermo 3 não pode ser combinado ao mintermo 14 $(1110)_b$ porque há três *bits* diferentes entre eles, não apenas um.
c) Caso algum item não tenha sido marcado, este será considerado um implicante primo, pois ao não ser combinado com nenhum outro, subentende-se que este é o único implicante capaz de suprir à determinado mintermo ou conjunto de mintermos;
d) Há a separação em caixas do item “a” realizada novamente, utilizando os implicantes gerados e marcados na etapa “b”;
e) Os passos anteriores devem ser repetidos até que não seja mais possível realizar a combinação de nenhum implicante.

Esta etapa, chamada de etapa de “Geração de Implicantes” não fornece o custo mínimo da expressão, apenas realiza a combinação de mintermos

visando obter uma expressão equivalente, de menor custo do que a inicial, porém ainda minimizável.

Para reduzir ainda mais, é utilizada a “Carta de cobertura de implicantes”, que consiste, basicamente, em uma tabela (ou matriz), onde as linhas são caracterizadas pelos implicantes primos gerados, e as colunas pelos mintermos que se deseja cobrir. Mapeiam-se, por este formato, quais mintermos são cobertos por cada implicante primo, então, por tentativa e erro, é possível determinar a seleção de menor número de implicantes primos possível, que cubram todos os mintermos necessários. Preferencialmente são selecionados na carta inicialmente, os implicantes que cobrem o maior número de implicantes possível.

Como exemplo, tem-se o agrupamento em caixas na Tabela 8, com o desenvolvimento da minimização ilustrado nas Tabelas 9, 10 e 11, para a Expressão (5),

$$F(A,B,C,D) = \sum m(0,2,4,8,10,12) \tag{5}$$

Tabela 8 - Grupo 0 do exemplo

Caixa	Mintermo	Marca
0	0000	X
1	0010	X
	0100	X
	1000	X
2	1010	X
	1100	X

Fonte: Elaborado pelo autor

Tabela 9 - Grupo 1 do exemplo

Caixa	Implicante	Marca
0	00X0	X
	0X00	X
	X000	X
1	X010	X
	X100	X
	10X0	X
	1X00	X

Fonte: Elaborado pelo autor

Tabela 10 - Grupo 2 do exemplo

Caixa	Implicante	Marca
0	X0X0	
	XX00	

Fonte: Elaborado pelo autor

Como não é possível fazer mais combinação alguma, estes são os implicantes primos gerados pela etapa de geração, podendo-se então construir a carta de cobertura, Tabela 11.

Tabela 11 - Carta de cobertura de mintermos do exemplo

	0	**2**	**4**	**8**	**10**	**12**
X0X0	X	X		X	X	
XX00	X		X	X		X

Fonte: Elaborado pelo autor

Pela Tabela 11 se observa que ambos são implicantes primos essenciais, pois X0X0 é o único capaz de cobrir os mintermos 2 e 10, enquanto que $XX00$ é o único capaz de cobrir os mintermos 4 e 12, ambos cobrem os mintermos 0 e 8. Assim sendo, selecionando os dois implicantes, são cobertos todos os mintermos e é tido por expressão de mínimo custo, na notação de soma de produtos, $X0X0 + XX00$, também representada pela Expressão (6),

$$F(A,B,C,D) = \sum m(0,2,4,8,10,12) = \overline{B}\,\overline{D} + \overline{C}\,\overline{D} \quad (6)$$

No Apêndice A podem ser encontrados outros exemplos de expressões booleanas minimizadas com resultados, que foram testados visando validar o aplicativo para *smartphone*.

Deve-se ressaltar que, devido a sua característica iterativa, com alto potencial de implementação em *loops*, o método de *Quine-McCluskey* é uma alternativa importante para implementação computacional de sistemas de minimização de equações booleanas (BOOLE, 1854).

4 A LINGUAGEM PYTHON E SUAS BIBLIOTECAS

Para o desenvolvimento do aplicativo de minimização por *Quine-McCluskey*, para dispositivo móvel no sistema *Android*, foi utilizada a linguagem *Python*[2], e algumas bibliotecas de seu ecossistema, como a *Sympy*[3], *Kivy*[4], *KivyMD*[5] e *Schemdraw*[6], além disso, também foi utilizada a *Buildozer*[7] que é uma ferramenta de empacotamento de aplicações móveis as quais são apresentadas a seguir .

4.1 LINGUAGEM DE PROGRAMAÇÃO *PYTHON*

Python é uma linguagem de programação orientada a objetos de alto nível, seu lançamento ocorreu em 1991 em uma versão 0.9.0 (a versão 1.0 foi lançada posteriormente em 1994). Foi criada por Guido van Rossum no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação (CWI) com o objetivo de suceder a ABC, que se trata de uma linguagem básica de propósitos gerais, sem muitos recursos para aplicações em *softwares* sofisticados, porém de fácil leitura para propósitos didáticos. Assim sendo, *Python* conseguiu agregar a possibilidade de sofisticação a um ambiente de fácil leitura (SHEROMOVA, 2020).

Inicialmente a proposta era a concepção de uma linguagem com código de alta legibilidade, colocando em segundo plano a velocidade e o esforço computacional. E de fato, desde suas versões iniciais, *Python* se destacou pela simplicidade de interpretação do código, com uma sintaxe muito intuitiva, posteriormente, com o desenvolvimento e amadurecimento da linguagem, outra vantagem é a sua escalabilidade, ou seja, fornece flexibilidade para resolver

[2] Informações e *download* em: https://www.python.org
[3] Informações em: https://www.sympy.org/pt/index.html
[4] Informações em: https://kivy.org/#home
[5] Informações em: https://kivymd.readthedocs.io/en/latest/
[6] Informações em: https://schemdraw.readthedocs.io/en/stable/
[7] Informações em: https://buildozer.readthedocs.io/en/latest/

problemas que não podem ser resolvidos usando outras linguagens de programação (SANTANA, 2019).

4.1.1 Características básicas da linguagem

Atualmente, a linguagem é desenvolvida em formato comunitário, de código-fonte aberto e gerenciado pela organização sem fins lucrativos, *Python Software Foundation*, um de seus pontos fortes. Conta com uma comunidade muito expressiva em fóruns da internet, fazendo com que haja a criação de bibliotecas que solucionam os mais diversos problemas por vários programadores, amadores ou não, ao redor do mundo.

Python é uma linguagem de programação interpretada, e disponível para vários sistemas operacionais. Ser uma linguagem interpretada significa dizer que ao se escrever um programa, este não será compilado (traduzido para uma linguagem de máquina), existe um interpretador para a máquina que traduz o que o programa quer dizer. O interpretador para *Python* é interativo, ou seja, é possível executá-lo sem fornecer um *script* (programa) para ele. Ao invés disso, o interpretador disponibilizará uma interface interativa onde é possível inserir os comandos desejados, um por um, e ver o efeito de cada um deles.

Caso o usuário esteja utilizando um sistema Linux ou OS X (*Apple*), o interpretador para *Python* já vem instalado por padrão, sendo apenas necessário escrever o comando "*python*" no seu programa de terminal favorito. Para usuários do sistema operacional *Windows*, o interpretador para *Python* deve ser baixado e instalado. Neste último sistema o usuário encontra um utilitário para fazer o papel de terminal (e editor de *python*) no *Windows*, denominado IDLE (GRUPO PET-TELE, 2009; MENEZES,2019).

É possível utilizar *Python* para muitos propósitos, como por exemplo, criar jogos, construir aplicações web, resolver problemas de negócios e desenvolver ferramentas internas em todo tipo de empresas, além disso, pode ser usada em áreas científicas como pesquisa acadêmica, por exemplo, os estudos sobre ondas gravitacionais da*Laser Interferometer Gravitational - Wave Observatory* (LIGO) ligada ao MIT, onde *Python* é utilizado no tratamento

de dados como processamento de sinais e análises de dados (ROMANO, 2019; MATTHES,2016).

A linguagem é conhecida como um canivete suíço do mundo da programação porque suporta programação estruturada, codificação orientada a objetos, além de ser uma linguagem de programação funcional e ter outras funções. De acordo com a pesquisa do *StackOverflow*, de 2018, *Python* é a linguagem de programação mais popular do mundo e também é a mais adequada para ferramentas e aplicativos de *Data Science*.

Para compreender a implementação do aplicativo desenvolvido é imprescindível que se tenha uma noção, principalmente em relação a uma das estruturas de dados básica, a chamada lista.

Listas são uma das quatro estruturas de dados padrão da linguagem para armazenamento de uma coleção de dados (também existem tuplas, *sets* e dicionários), ela se destaca principalmente por ser ao mesmo tempo ordenada e manipulável e pode ser criada utilizando colchetes.

4.1.2 Visualização e gráficos

Python fornece várias opções de visualização e gráficos, por exemplo, a biblioteca *Matplotlib* possui uma base sólida em torno da qual outras bibliotecas como *Plotly*, *Seaborn* e outras são construídas. Esses pacotes auxiliam na criação de tabelas, gráficos prontos para a *web*, *layouts* gráficos, entre outros tipos de visualização (SANTANA, 2019).

4.1.3 Aplicativos desenvolvidos utilizando *Python*

A linguagem de programação *Python* tem crescido muito em uso, sendo “uma das melhores linguagens de programação que todo desenvolvedor deveria aprender” (“[...] *one of the best programming languages every developer should learn* [...]”, BELANI, 2020). E alguns dos mais conhecidos,

utilizados e mesmo inovadores aplicativos do mundo são construídos parcialmente nesta linguagem.

O *Instagram*, por exemplo, “apresenta atualmente o maior desenvolvimento mundial da estrutura web Django[8], que está escrito inteiramente em *Python*” (“*Instagram currently features the world’s largest deployment of the Django web framework, which is written entirely in Python*.”, NI, 2016). Ainda segundo Min Ni, a linguagem foi escolhida pela sua simplicidade e praticidade e seu projeto foi mantido pela escalabilidade oferecida.

O *Reddit*, um dos sites com mais visitantes dos EUA, também possui seus serviços codificados em *Python*, o principal motivo para a escolha desta linguagem neste caso é a versatilidade que a linguagem entrega com suas bibliotecas que abrangem as mais diversas tarefas.

Outros aplicativos com diversas propostas totalmente diferentes como o *Dropbox* e a *Netflix* também possuem codificação parcial em *Python,* o que reforça a versatilidade e usabilidade desta linguagem para aplicações de diversos espectros.

4.2 BIBLIOTECA *SYMPY*

Trata-se de uma biblioteca *Python* para computação simbólica, oferecendo ferramentas algébricas computacionais, sendo iniciada em 2006, por Ondrej Certik e desde então muitas pessoas contribuíram para o projeto – em 2011 mais de 150 pessoas já haviam contribuído para o código do projeto. Esta biblioteca possui, atualmente, mais de 200 mil linhas de código divididas em 26 módulos, oferecendo recursos como:

- Ferramentas básicas:
    - Operadores de aritmética básica (+, -, *, /, **);
    - Simplificação (Trigonometria e polinômios);

[8] Django é um *framework* que permite o desenvolvimento rápido de aplicações para a web, escrito em *Python*, possui como principais características a simplicidade e escalabilidade.

    - Expansão de polinômios;
    - Funções (trigonométricas, hiperbólicas, exponenciais, raízes, logaritmos, valores absolutos, harmônicas esféricas, fatoriais e funções gama, funções zeta, entre outros).
- Polinômios:
    - Aritmética básica (divisão, mdc); Fatoração; Decomposição de raíz; Bases de *Gröbner.*
- Cálculo:
    - Limites; Diferenciação; Integração; Séries de *Taylor.*
- Resolução de equações:
    - Polinomiais; Algébricas; Diferenciais; Relações de recorrência; Sistemas.
- Matemática discreta:
    - Coeficientes; binomiais; Somatórios; Produtos; Expressões lógicas e Teoria dos números (geração de números primos, fatoração de inteiros, entre outros).
- Matrizes
    - Aritmética básica; Autovalores; Determinantes; Inversão e Resolução.
- Desenhar gráficos
    - Modos de coordenadas; Entidades geométricas; 2D e 3D.
- Física
    - Mecânica; Quântica; Ótica Gaussiana; Álgebra de Pauli.

Muitos outros recursos são oferecidos por esta biblioteca, como a possibilidade de fazer a manipulação e tratamento de expressões lógicas, usando simbolismo e valores booleanos. Este grupo de ferramentas é especialmente importante para garantir a entrega da expressão de mínimo custo na forma de soma de produtos para determinados tipos de entradas no desenvolvimento de aplicativos, após o devido tratamento dessas para se adequar à sintaxe exigida.

4.3 BIBLIOTECAS *KIVY* E *KIVYMD*

A biblioteca *Kivy* é um projeto comunitário, seu lançamento inicial ocorreu em 2011, porém sua versão realmente estável, com compatibilidades e recursos melhor consolidados, ocorreu somente em 2020. Seu uso é gratuito, sendo distribuído sobre os termos da licença MIT[9].

Trata-se de uma ferramenta para desenvolvimento de aplicativos móveis com uma interface natural para o usuário, contendo recursos já popularizados como o *multi-touch*, além disso é multiplataforma, sendo muito atraída por desenvolvedores que buscam respostas a problemática de “escreva uma vez, execute em qualquer lugar” (“*write once, run anywhere*”). Segundo Vasilkov (2015), alguns dos principais recursos do *Kivy* são:

- Compatibilidade: aplicações baseadas em *Kivy* são compatíveis para trabalhar em *Linux*, Mac OS X, *Windows*, *Android* e iOS, todas utilizando uma única base de código;
- Interface natural de usuário: a *Kivy* possibilita a inserção de diferentes métodos de entrada, permitindo ao desenvolvedor trabalhar com inúmeras possibilidades de interação do usuário, usando código similar é possível manipular eventos de mouse, gestos *multi-touch* e semelhantes;
- Gráficos rápidos acelerados por *hardware*: *Kivy* utiliza *OpenGL* para a renderização, permitindo que aplicações com gráficos relativamente pesados, como jogos, possam ser criados ou ter a experiência do usuário melhorada com transações mais suáveis.
- Os aplicativos que usam *Kivy* são escritos em *Python*, assim sendo, além de muito portável e legível, os recursos oferecidos pela linguagem e seu ecossistema tornam as possibilidades muito amplas.

A biblioteca *KivyMD* funciona como uma espécie de extensão para a *Kivy*, trazendo recursos de *material design*, tornando a interface mais elegante e atrativa para o usuário, utilizando uma sintaxe muito similar à do *Kivy*, esta

[9] Licença MIT (MIT License) é uma permissão para distribuição e uso de softwares e recursos totalmente gratuitos originária do Instituto de Tecnologia de Massachusetts, o MIT.

biblioteca ainda se encontra em estágios iniciais de teste, porém seus recursos já são bem amplos e oferecem uma boa compatibilidade e aceitação por parte da comunidade.

Um recurso interessante possibilitado pela biblioteca *Kivy* é o reconhecimento da linguagem KV, recurso de facilitação ao desenvolvedor. À medida que a construção de um aplicativo se torna mais complexa, com mais *widgets*, sendo atribuídos respeitando uma hierarquia, o *script* pode se tornar confuso ou de difícil entendimento, para solucionar este problema se utiliza a KV, uma linguagem declarativa que torna a construção do aplicativo muito mais natural, além de facilitar a separação da lógica em relação à interface.

4.4 BIBLIOTECA *SCHEMDRAW*

Schemdraw é um pacote *Python* que pode ser utilizado para a produção e plotagem de esquemas de circuitos elétricos. Trata-se de uma biblioteca muito recente, encontra-se atualmente em sua versão 0.11, mas já oferece recursos interessantes e amplos, sendo um projeto muito promissor ao que se propõe e útil para desenvolvedores relacionados a área de Engenharia Elétrica. Alguns dos elementos disponíveis para a plotagem são:

- Elementos de circuitos básicos:
    - Chaves; Elementos de terminal duplo (resistores, diodos, capacitores e muitos outros); Elementos de áudio; Transistores; Cabos; Transformadores; Amplificadores operacionais; Circuitos integrados DIP: Multiplexadores, Display de sete segmentos, etc.
- Elementos compostos:
    - Opto acopladores; Relês;
- Portas lógicas.

4.5 *BUILDOZER*

Buildozer é uma ferramenta que tem por objetivo fazer o empacotamento de aplicações móveis de forma fácil. Permite a automação do processo de compilação de aplicativos em um executável, fazendo, por exemplo, a aquisição dos pré-requisitos. Em outras palavras, segundo Gad (2019), usar *Buildozer* para a compilação de aplicações *Android* é similar a ter uma ponte entre *Python* e *Android* (ou *Java*), onde o desenvolvedor cria um projeto *Python* e *Buildozer* faz a conversão para um projeto *Android Java* seguindo as especificações definidas pelo desenvolvedor.

A ferramenta suporta o empacotamento para:

- *Android*: via *Python-for-Android*[10], requer *Linux* ou computador OSX para realizar a compilação;
- iOS: via *Kivy* iOS, requer computador OSX para realizar a compilação;
- Outras plataformas (como .exe para *Windows*, .dmg para OSX e outros).

[10] *Python-for-Android* é uma ferramenta de compilação capaz de transformar um pacote de códigos *Python* em códigos que seguem o padrão para .apk do *Android*.

5 IMPLEMENTAÇÃO DO APLICATIVO MINIMIZER

Neste tópico descrevem-se o desenvolvimento do aplicativo e os recursos em termos de *software* e *hardware* utilizados. Um diagrama de blocos resume os *scripts* construídos que são detalhados, de forma a dar os subsídios para o entendimento de sua implementação.

5.1 RECURSOS DE HARDWARE E SOFTWARE

O aplicativo desenvolvido foi denominado de Minimizer e construído utilizando linhas de código em *Python* versão 3.9 e suas ferramentas. Foram utilizadas as bibliotecas Sympy, versão v1.8; as versões 2.0.0 e 0.104.2 para *Kivy* e *KivyMD* respectivamente. Para o recurso de plotagem de circuitos lógicos na produção do aplicativo, foi utilizada a biblioteca *Schemdraw* em sua versão 0.11 e se utilizou a *Buildozer* 0.11 para compilação de aplicações de *Python* para *Android*.

O programa foi desenvolvido em um computador (PC) com processador Ryzen 7 1800x, com 16GB de memória RAM e placa de vídeo dedicada RX 570 com 8GB de VRAM rodando o sistema operacional *Windows* 10, no qual se executava uma máquina virtual *Ubuntu*, baseada no sistema *Linux*, para se obter compatibilidade ao *Buildozer*.

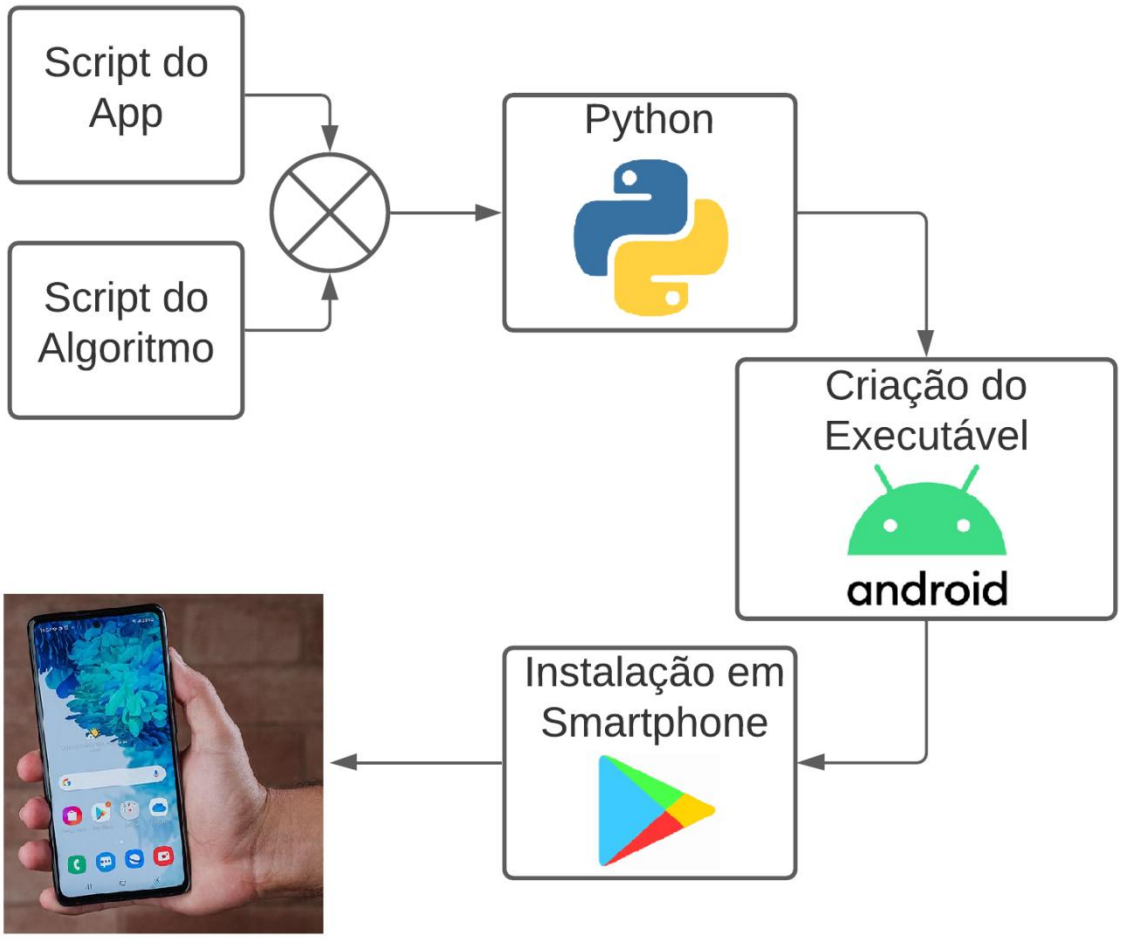
5.2 DIAGRAMA DE BLOCOS DO APLICATIVO

Na Figura 6 tem-se o diagrama de blocos mostrando que basicamente existem dois *scripts* (roteiros de programas) principais, um é responsável pela construção do processo de simplificação - a implementação do algoritmo de *Quine-McCluskey* (*script* do algoritmo) e outros processos. E um segundo *script* contendo a descrição da interface do aplicativo (*script* do aplicativo), neste se utiliza em partes da leitura de um arquivo KV, discutido na seção 5.5, para a descrição dos *widgets* (botões, barras de ações, *cards*, entre outros).

A criação do executável ocorre em seguida, para isto se utilizou do *Buildozer* funcionando em um sistema virtual *Linux Ubuntu*. O arquivo gerado

após este empacotamento possui extensão .apk e pode ser instalado em *smartphones* com sistema operacional *Android*. Descrevem-se a seguir, seções 5.3 e 5.4, algumas das funções utilizadas na concepção do aplicativo.

Figura 6 - Diagrama de blocos do aplicativo



Fonte: Elaborado pelo autor

5.3 *SCRIPT* DO ALGORITMO

No *script* do algoritmo é implementado o método de *Quine-McCluskey* utilizando recursos básicos do *Python*, sendo construído de forma quase pura, exceto pela função *product*, descrita mais adiante, disponibilizada pelo módulo padrão *itertools*.

Todas as etapas se baseiam na utilização e manipulação da estrutura de listas do *Python*, capaz de armazenar diversos itens em uma única variável, ordenadas, manipuláveis, indexadas e permitindo duplicidade de valores.

O *script* recebe como entrada duas sequências de valores, que são armazenadas em duas listas diferentes, relativas aos mintermos e aos *don't care*, caso haja. Ambas as listas são agrupadas em uma terceira lista, contendo mintermos e *don't care, e*stas listas passam pela função Binário():

- Binário(): esta função recebe como argumentos uma lista contendo números inteiros (representando mintermos) e o número de variáveis, ela converte todos os itens da lista para binários. Exemplo: Binário ([0,1,2,6],3) retorna ['000','001','010','110'].

A lista contendo mintermos e *don't care* em binário é então enviada para função NumCaixas() e seu retorno é então enviado para a função Combinação():

- NumCaixas(): esta função recebe como argumento uma lista contendo *strings* que representam números binários relativos aos mintermos e *don't care* da entrada, faz a contagem do número de 1's da *string* e a agrupa, dentro de listas, com outras *strings* que possuam a mesma quantidade de 1's, fazendo assim a representação do sistema de "caixas" do algoritmo. Exemplo: NumCaixas(['000','001','010','011','100']) retorna [['000'],['001','010','100'],['011]].
- Combinação(): esta função recebe como argumento uma lista, contendo listas que agrupam os binários (representando os implicantes) em "caixas", dividindo-os conforme sua quantidade de 1's. Com esta entrada, é feita a comparação de todos os itens de cada caixa, com todos os itens da caixa subsequente, caso haja apenas um bit de diferença entre os itens comparados, estes dois itens serão copiados para uma lista de itens marcados, e o bit diferenciado é substituído por um 'X'. O retorno desta função será uma lista contendo os implicantes que foram comparados com sucesso, já com o bit modificado, caso nenhum implicante seja combinado, a função retornará uma lista vazia. Exemplo: Combinação([['0X','X0'],['X1','1X']]) retorna ['XX'].

Os termos ficam em *loop* passando pelas funções NumCaixas() e Combinação() até que nenhum termo possa ser combinado (função Combinação() retornará uma lista vazia). Os implicantes primos gerados pela combinação são aqueles que não foram marcados, são então agrupados e vão para a segunda etapa do método, a Cobertura dos Mintermos.

Inicialmente na etapa de cobertura, os implicantes primos gerados na primeira etapa passam pela função ValPrimos():

- ValPrimos(): esta função recebe como argumento uma lista de *strings* representando implicantes primos, esta função utiliza o método de permutação da biblioteca *itertools* com sua função *product* para realizar uma expansão do implicante primo mostrando todos os mintermos que este pode cobrir. Exemplo: ValPrimos([‘10X’,’XX0’]) retorna [[‘100’,’101],[‘000’,’010’,’100’,’110’]].

O próximo passo é fazer a seleção dos implicantes primos essenciais, para isto, cada mintermo é comparado com todos os valores cobertos pelos implicantes gerados anteriormente (lista retornada pela função ValPrimos()), devido a característica de ordenamento das listas é possível utilizá-las semelhante a uma matriz linha, uma vez que os índices dos itens que as compõe são respeitados. Assim sendo, cada vez que um mintermo é coberto por um implicante, uma lista de checagem com o mesmo tamanho da lista de implicantes atribui o valor 1 ao índice relativo deste, caso, ao fim da comparação só haja um único 1 na lista de checagem, significa que o implicante com aquele índice é essencial, pois é o único que cobre determinado mintermo, sendo este automaticamente atribuído ao resultado final e tendo seus mintermos cobertos retirados da próxima checagem.

Após terem sido encontrados todos os implicantes essenciais, caso existam, serão selecionados prioritariamente os implicantes que cobrem a maior quantidade de mintermos até que todos estejam cobertos, para isto a mesma propriedade matricial das listas será utilizada. Dentro de um *loop* serão comparados todos os mintermos restantes com os implicantes restantes ainda não selecionados, cada vez que a cobertura for atingida, uma lista de contagem adicionará 1 ao respectivo índice. Ao fim do *loop*, o implicante com o maior índice será selecionado ao resultado final, pois cobre o maior número de

mintermos sendo mais eficaz na diminuição dos custos. Os mintermos cobertos pelos implicantes são retirados da próxima etapa de contagem, e o *loop* se repete até que todos os mintermos sejam cobertos.

Embora os implicantes da seleção de custo mínimo já estejam selecionados, este *script* ainda realiza algumas tarefas, inicialmente os dados são tratados para a criação das *strings* que serão utilizadas na demonstração ao usuário da Etapa de Geração de Implicantes (função discutida mais adiante).

Após, os resultados obtidos são convertidos para o formato de soma de produtos pela função Expressão():

- Expressão(): esta função recebe como argumento uma lista que contém *strings* contendo ‘0’, ‘1’ e ‘X’ representando implicantes e as transforma em uma soma de produtos mais legível ao usuário. Exemplo: Expressão([‘000’,’010’,’X00’]) retorna [“A’B’C’ + A’BC’ + B’C’”].

Por fim, os custos da expressão inicial e final são calculados, considerando a quantidade de entradas das portas lógicas *AND* e *OR*. Na Figura 7 mostra-se uma parte deste *script*, contendo o loop que utiliza algumas das funções descritas.

Figura 7 - Trecho do *script* do algoritmo

```
#Etapa 1: Combinação de mintermos e g
#Decisão se a entrada será via teclad

while 1:
    mini1=mini.copy()
    if dont:
        mini1=mini1+dont
#Os mintermos inseridos são agrupados
    mintermos1=Binário(mini1,var)
    mintermos=Binário(mini,var)
    Caixas=NumCaixas(mintermos1)
    Lista2=Combinação_1(Caixas)
    Grupos.append(Caixas)
    while 1:
        Lista0=NumCaixas(Lista2)
        Lista2=Combinação(Lista0)
        Grupos.append(Lista0)
        if Lista2==[]:
            break
    Primos=ImplicantesPrimos(Grupos)
    break
```

Fonte: Elaborado pelo autor

5.4 *SCRIPT* DA INTERFACE DO APLICATIVO

O *script* da interface contém as funções que recebem e respondem às entradas do usuário, ou seja, existe uma função para cada botão, em cada página presente no aplicativo, esses *widgets* são descritos dentro do arquivo KV, que por sua vez é lido por este *script*.

As funções principais são button_press(), button_press_tabvdd(), button_press_exp() e button_press_draft(), sendo as três primeiras responsáveis por coletar a entrada do usuário, verificar sua validade e enviá-las ao algoritmo de minimização e em seguida receber o resultado e o demonstrar pela criação de caixas de diálogo (MDDialog); a quarta função manipula a entrada para o desenho do circuito com auxílio dos métodos da biblioteca *Schemdraw.*

A verificação de validade das entradas considera o valor máximo dos mintermos (só serão aceitos mintermos menores do que 2097151) e na devida sintaxe de entrada indicada para o método escolhido (critério explicitado mais adiante), caso os critérios não sejam respeitados, o *script* gerará uma exceção, indicando ao usuário a fonte do provável erro, por meio de uma caixa de diálogo.

Outras funções disponíveis nesse script são create_table() e clear_table(), responsáveis pela criação e limpeza da tabela verdade disponibilizada. Além disso, cada caixa de diálogo, seja de ajuda ou amostra de resultado, possui uma função própria, todas semelhantes, diferindo apenas nas mensagens apresentadas em sua comunicação.

Na Figura 8 é possível observar parte da função buttonpress(), cobrindo sua recepção de mintermos e ferramenta de proteção contra entradas indesejadas.

Figura 8 - Trecho do *script* do aplicativo

```
def build(self):
    self.title = 'MiniMizer'
    self.theme_cls.primary_palette = "BlueGray"

def button_press(self):
    self.root.ids.detalhes.clear_widgets()
    MainApp.description_result=''
    mintermos_str = self.root.ids.minter.text
    mintermos_list = mintermos_str.split(',')
    donts_str = self.root.ids.dontcares.text
    donts_list = donts_str.split(',')
    if mintermos_str:
        if ',' in mintermos_str:
            for i in mintermos_list:
                try:
                    int(i)
                    if int(i)>2097151:
                        raise Exception
                except:
                    self.entrada_erro()
                    self.root.ids.minter.text=''
                    self.root.ids.dontcares.text=''
                    return
        else:
```

Fonte: Elaborado pelo autor

5.5 ARQUIVO KV

O arquivo KV é construído na linguagem descritiva de mesmo nome, ele contém a árvore de hierarquia de todos os *widgets* do aplicativo, bem como faz a ligação de cada botão com sua respectiva função no *script* da interface que o lê.

A descrição apresentada no KV para o Minimizer possui a camada mais externa da hierarquia contendo as telas, acessadas pelo botão inferior de navegação, dentro destas telas os *widgets* estão dispostos em formato de caixas, cada uma com suas peculiaridades e funcionalidades.

Para ilustrar mostra-se na Figura 9, um trecho do arquivo KV, contendo a descrição dos *widgets* da página de entrada via Tabela Verdade. Pode-se observar que critérios como posicionamento e cores devem ser considerados neste arquivo, que possui caráter mais visual.

Figura 9 - Trecho do arquivo KV

```
MDLabel:
    text: 'Tabela Verdade'
    halign: 'center'
    font_size: 75
    color: (101/255,120/255,149/255,1)
    bold: True
    italic: True
    outline_color: (0,0,0,1)
    outline_width: 5
MDBoxLayout:
    orientation:"horizontal"
    padding:dp(15)
    MDTextField:
        id: num_var
        hint_text: "Número de variáveis"
        size_hint_x: 0.65
        pos_hint: {'center_x': 0.15,'center_y': 0.8}
    MDIconButton:
        id: var_question
        icon: "file-question-outline"
        pos_hint: {'center_x': 0.8,'center_y': 0.8}
        on release: app.var help()
```

Fonte: Elaborado pelo autor

6 RESULTADOS E DISCUSSÃO

O aplicativo desenvolvido foi denominado Minimizer, cujo logotipo é apresentado na Figura 10 e deve ser executado em dispositivos *Android.* Foi testado em um *smartphone Xiaomi Mi Note* 10 utilizando a MIUI 12, baseado em *Android* 12, porém, em teoria, qualquer dispositivo que utilize *Android* acima da versão 2.2 poderá executá-lo. Neste item apresentam-se todas as funcionalidades (telas) implementadas e ilustradas por meio de exemplos.

Figura 10 - Logotipo do aplicativo desenvolvido

Fonte: Elaborado pelo autor

Minimizer oferece de forma simples e intuitiva, cinco opções diferentes de telas selecionáveis por meio de uma barra de abas na parte inferior .

Os resultados minimizados são entregues na forma de soma de produtos em seu mínimo custo utilizando portas AND e OR (exceto pela entrada via equações). De forma geral, foi atribuída a limitação de 21 variáveis (mintermo máximo 2097151), mas obviamente, o uso de valores excessivos, dependendo do caso, irá tornar lento o sistema, como mostrado nas Figuras 19, 20 e 21.

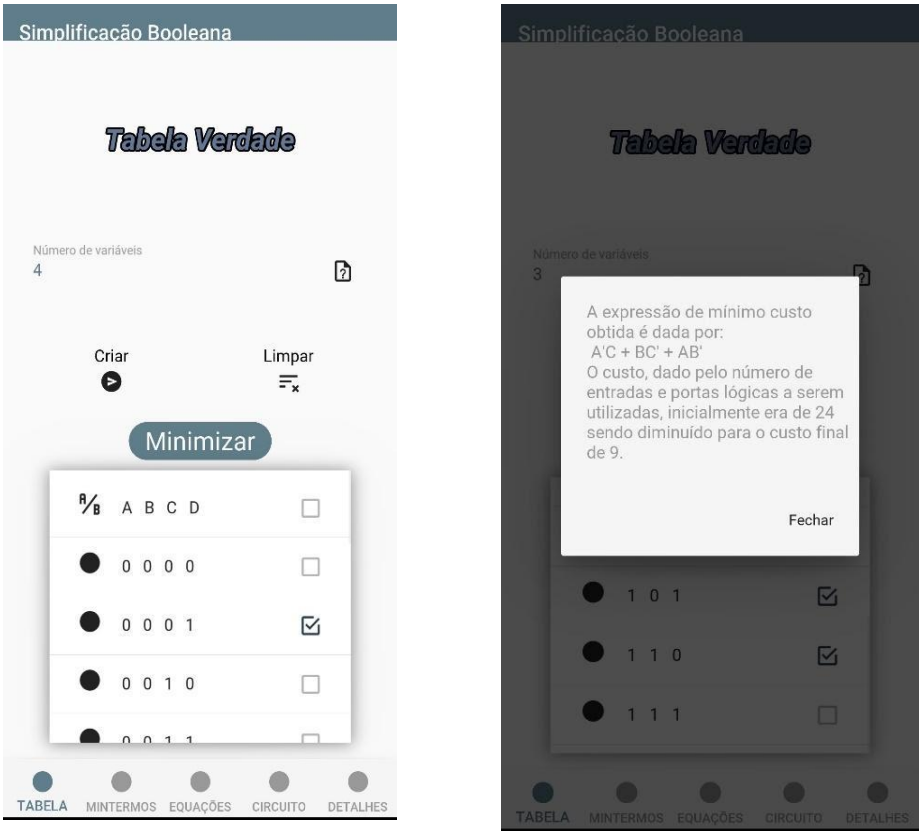
A primeira tela é para entrada via preenchimento de tabela verdade, mostrada na Figura 11.

Para realizar a minimização utilizando a entrada por tabela verdade o usuário deve preencher o campo “Número de variáveis” com o valor desejado, em seguida tocar em “Criar”, uma tabela surgirá na tela, bastando ao usuário marcar os mintermos desejados, como pode ser visualizado na Figura 12(a) em seguida tocar em “Minimizar”. O resultado surgirá na tela em formato de caixa de diálogo, como exemplificado na Figura 12(b).

Figura 11 - Tela para entrada via preenchimento de tabela verdade

Fonte: Elaborado pelo autor

Figura 12 - a) Criação de Tabela Verdade no aplicativo. b) Resultado da minimização por entrada via Tabela Verdade



(a) (b)

Fonte: Elaborado pelo autor

O resultado do teste mostrado na Figura 12(b) apresenta a minimização para a Expressão (7), resultando na somatória de produtos de mínimo custo dado pela Expressão (8), sendo o custo, dado pela quantidade de entradas das portas AND e OR, diminuído de 24 para 9, conforme observado,

$$F(A,B,C) = \sum m(1,2,3,4,5,6) \tag{7}$$

$$F'(A,B,C) = \overline{A}C + B\overline{C} + A\overline{B} \tag{8}$$

Em caso de dúvidas o usuário poderá tocar no ícone de questão (localizado ao lado direito das barras de entrada de dados via teclado e simbolizado por um arquivo com um ponto de interrogação) e uma dica flutuante surgirá para orientação, também é possível recriar a tabela clicando

no ícone “Limpar”. Como mostrado na Figura 13, em caso de entrada irregular, uma mensagem de aviso surgirá a título de informação.

As limitações para este tipo de entrada é que não são aceitos *don’t care*, além disso, há um considerável uso de memória na criação de tabelas escalonando com o número de variáveis solicitadas (o tempo médio de execução pode ser visto na Figura 19).

Figura 13 - Entrada irregular para a criação de Tabela Verdade no aplicativo

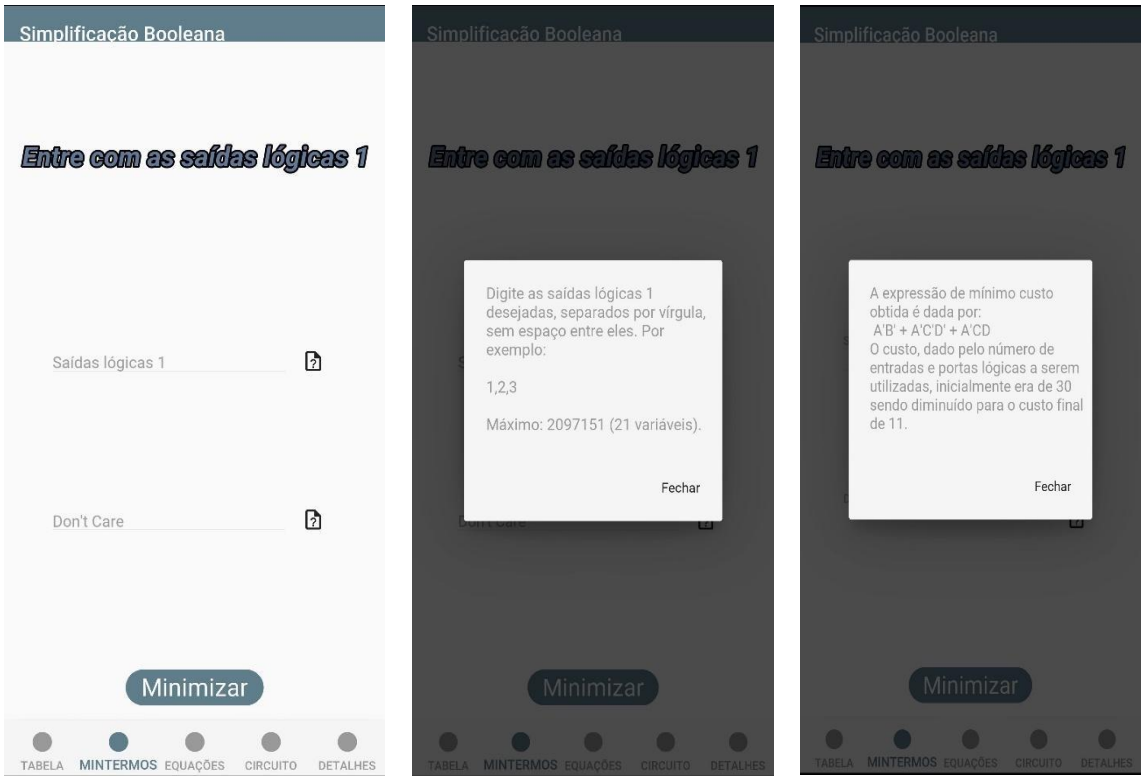
Fonte: Elaborado pelo autor

A segunda tela disponível, apresentada na Figura 14(a), possibilita a entrada via mintermos e *don’t cares*, o usuário deve inserir os valores desejados em números inteiros separados por vírgula. Após isto, basta clicar em “Minimizar”. Este método de entrada tem por vantagens a possibilidade de inserir *don’t care* e não necessita de processamento adicional para criação de uma tabela verdade, no entanto a inserção dos valores via teclado é mais trabalhosa. No caso de dúvidas, o usuário pode ainda tocar no ícone de ajuda, como mostrado na Figura 14(b). Na Figura 14(c) é mostrado o resultado da minimização da Expressão (9), dado na forma de soma de produtos na Função booleana (10) e tendo seu custo diminuído de 30 para 11,

$$F(A,B,C,D) = \sum m(0,1,2,3,4,7) + d(8,9) \tag{9}$$

$$F(A,B,C,D) = \overline{A}\,\overline{B} + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}CD \tag{10}$$

Figura 14 – a) Tela de entrada via mintermos, b) Tela de ajuda. c) Resultado da minimização da função booleana (9) via mintermos



(a) (b) (c)

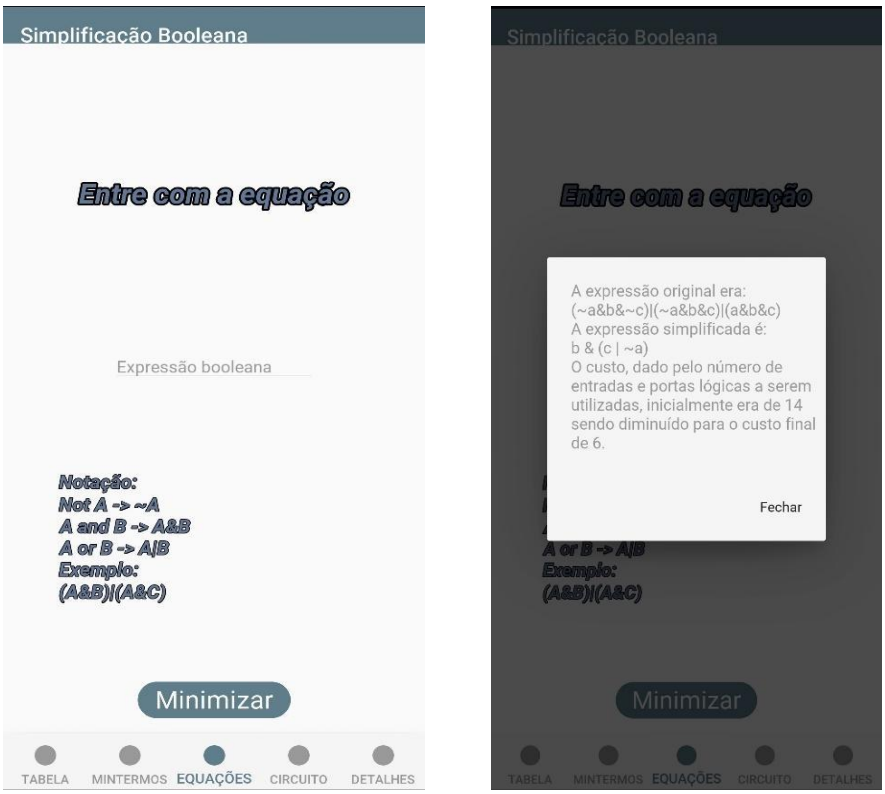
Fonte: Elaborado pelo autor

A terceira tela disponível permite a inserção de uma equação em soma de produtos, a ser simplificada, cuja notação exigida consta na tela e pode ser visualizada na Figura 15(a). O resultado simplificado para este tipo de entrada não é obtido diretamente pelo algoritmo implementado de *Quine-McCluskey*, mas sim por meio de recursos da biblioteca *Sympy*, a fim de demonstrar outras opções disponíveis. Poderá ser verificada neste caso uma notação de resultado ligeiramente diferente em relação aos outros, devido à diferença do método utilizado, mas em todos os casos são apresentados a opção de mínimo custo. Na Figura 15(b) mostra-se a resolução obtida para a Expressão (11), tendo como resultado minimizado a Função (12),

$$F(A,B,C) = \sum m(2,3,7) = \overline{A}B\overline{C} + \overline{A}BC + ABC \quad (11)$$

$$F(A,B,C) = B.(\overline{A} + C) \quad (12)$$

Figura 15 - a) Tela de entrada via equações, b) Resultado da minimização



(a) (b)

Fonte: Elaborado pelo autor

A quarta tela permite ao usuário inserir uma equação, seguindo a notação exigida, e receber o desenho do circuito equivalente ao clicar em “Desenhar”. Na Figura 16 mostra-se esta tela, enquanto nas Figuras 17(a) e 17(b) pode ser visualizado o circuito equivalente para a Expressão (13),

$$F(A,B,C,D) = \left(\overline{A}B + \overline{A}BC\right) \oplus D \quad (13)$$

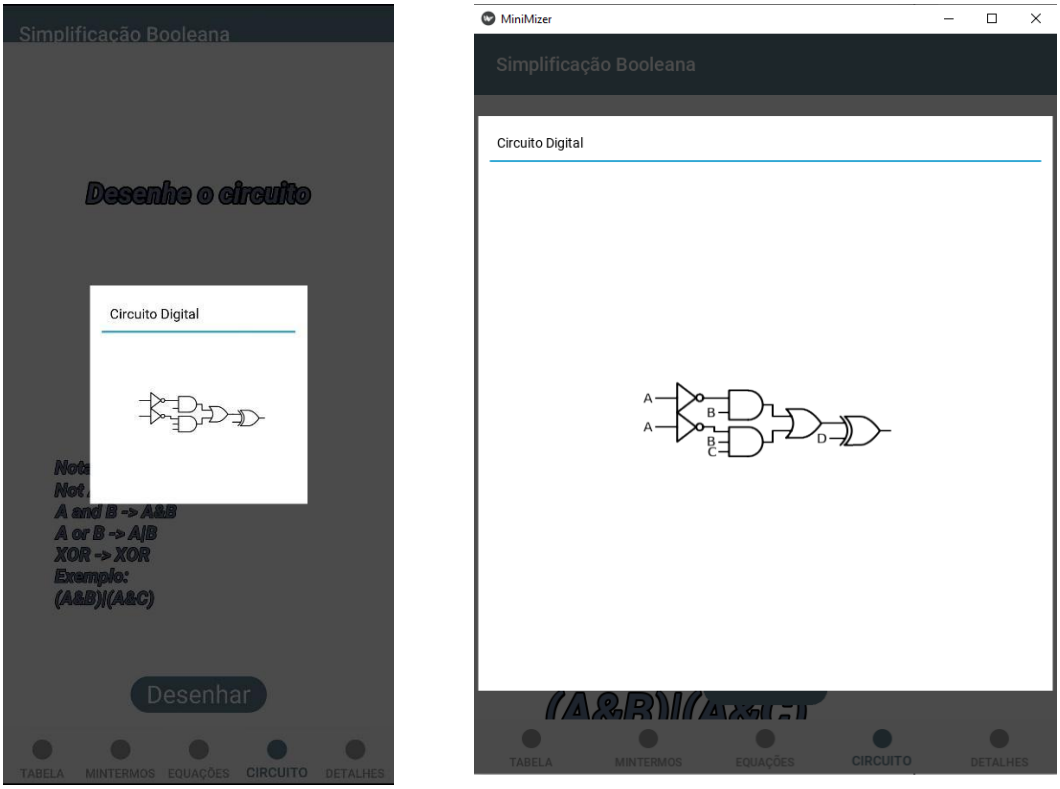
Como pode ser observado na Figura 17(a), as variáveis relativas às entradas não ficam visíveis, a biblioteca *Schemdraw,* responsável pela plotagem do circuito, ainda está em fase de desenvolvimento e nem todos os seus recursos estão completamente compatíveis para adaptação e

empacotamento em um aplicativo *Android*. Espera-se que em futuras atualizações esse recurso esteja disponível e ofereça melhor compatibilidade, evitando estas limitações. No entanto, apresenta-se na Figura 17(b) a execução do aplicativo no computador, antes da conversão para *Android*, onde é possível visualizar as variáveis de entrada, uma vez que não passa pelo processo de empacotamento.

Figura 16 - Tela de desenho de circuito

Fonte: Elaborado pelo autor

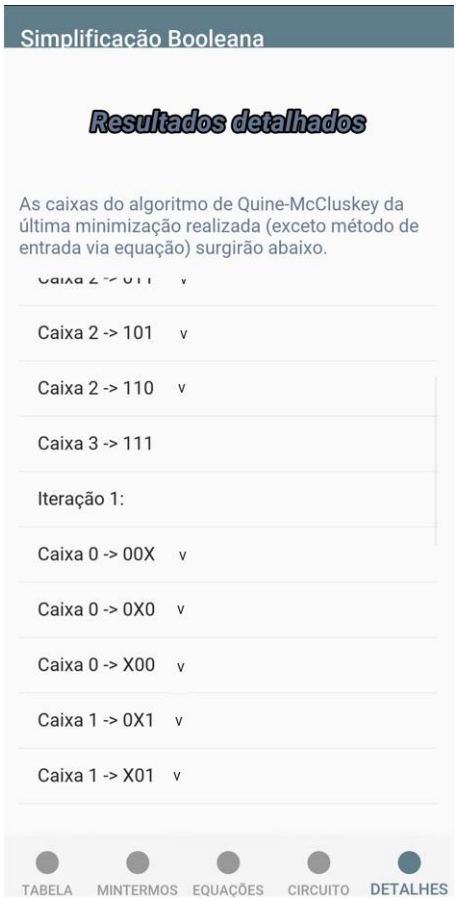
Figura 17 - Resultado de circuito plotado: a) Pós-empacotamento, b) Pré-empacotamento



(a) (b)

A quinta e última tela possui caráter didático, nela é possível visualizar todo o processo da Etapa de Geração de Implicantes da última minimização realizada com entrada via tabela verdade ou via mintermos e *don't cares*, deste modo o usuário pode ver a separação em caixas e combinação de implicantes em cada uma das iterações, conforme mostrado na Figura 18, e também é possível visualizar a notação utilizada na execução de um exemplo.

Figura 18 - Exemplo da tela de Detalhes



Fonte: Elaborado pelo autor

De maneira geral, Minimizer é um aplicativo compacto, ocupando cerca de 83MB de espaço (considerado pequeno), não necessitando de um *hardware* potente para ser executado em uso casual, para atestar isso foram coletados os tempos de execução da criação da tabela verdade e da execução do algoritmo no dispositivo utilizado para teste, para o caso mais trabalhoso, chamado de **tudo um**, caso onde há a combinação de todos os implicantes, tendo como resultado minimizado um ou zero.

Em todos os casos é possível observar que o tempo de execução aumenta exponencialmente para o aumento do número de variáveis, n utilizadas, isso ocorre devido ao aumento exponencial do número de termos a ser manipulado por causa da base binária, 2^n, conforme mostrado na Tabela 12. Analisou-se apenas os casos **tudo um** até oito variáveis, a partir disso foram notados travamentos e quedas de performance significativos.

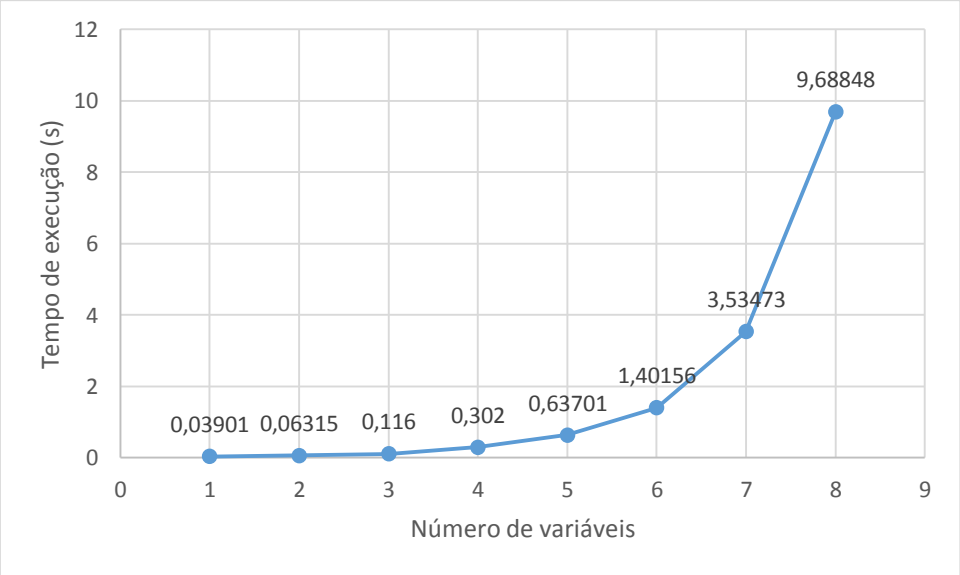
Tabela 12 - Relação entre número de variáveis e mintermos no caso **tudo um**

Número de variáveis (n)	**Número de termos** (2^n)
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256

Fonte: Elaborado pelo autor

O gráfico do tempo de execução na criação da tabela verdade é apresentado na Figura 19, em que se verifica que, a partir de sete variáveis o aumento do tempo de execução torna-se considerável.

Figura 19 - Tempo de execução x Nº de variáveis na criação da Tabela Verdade

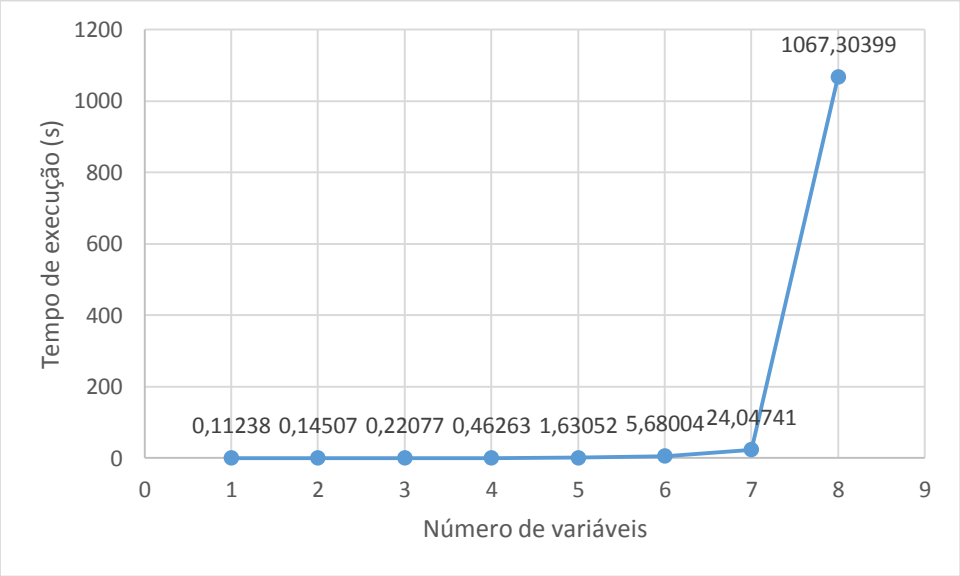


Fonte: Elaborado pelo autor

Os gráficos de tempo de execução do algoritmo, desde o momento em que o usuário solicita a minimização até a entrega dos resultados para entrada

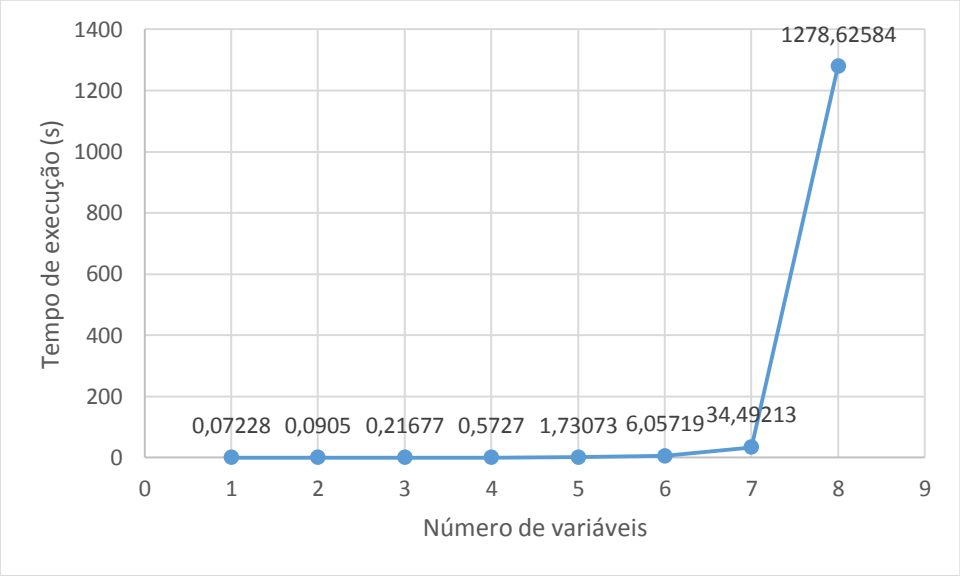
de dados via inserção de mintermos e via preenchimento da tabela verdade, respectivamente, são apresentados nas Figuras 20 e 21.

Figura 20 - Tempo de execução x Nº de variáveis para execução da minimização com entrada via mintermos



Fonte: Elaborado pelo autor

Figura 21 - Tempo de execução x Nº de variáveis para execução da minimização com entrada via preenchimento da Tabela Verdade



Fonte: Elaborado pelo autor

Em ambos os casos o teste foi feito na situação de maior *stress* possível, o caso **tudo um**, onde todos os termos são iterados para apresentar

o resultado “1”, ou “Verdade”. Como para ambas as entradas o processo de minimização ocorre por meio do mesmo algoritmo, não se verifica muita diferença de tempo de execução entre elas, no entanto a entrada via mintermos é ligeiramente mais rápida, pois exige uma alocação de dados um pouco menor, devido a não necessidade da construção da Tabela Verdade.

Nos testes realizados foram considerados até oito variáveis, acima disso verificou-se lentidão e instabilidade do aplicativo devido a quantidade de termos iterados. Vale ressaltar que foi analisada a situação de *stress* máximo, em que para casos mais simples, com menos termos, é possível a execução rápida com até 21 variáveis, mostrando que a lentidão é devida ao número de termos iterados e não à quantidade de variáveis ou tamanhos dos mintermos desejados.

Melhores resultados com menor custo, inclusive, podem ser obtidos com a minimização via Método de *Quine-McCluskey* estendido (SANCHES, 2017).

7 CONCLUSÕES

Neste trabalho foi desenvolvido um aplicativo para dispositivo *Android* denominado Minimizer visando a minimização de funções booleanas, utilizando o algoritmo clássico de *Quine-McCluskey* e a linguagem *Python*.

A implementação de funções booleanas com o menor custo possível é historicamente muito importante para o desenvolvimento da eletrônica, como discutido no trabalho e diferentes métodos continuam a ser estudados, de forma que o desenvolvimento de ferramentas como o Minimizer podem auxiliar não apenas na minimização rápida e prática, mas no aprendizado de disciplinas de circuitos digitais, de uma maneira didática e lúdica.

O desenvolvimento da minimização usando o algoritmo clássico de *Quine-McCluskey* em aplicativo apresenta, em todos os casos testados, a equação de mínimo custo, demonstrando ser eficaz nas implementações computacionais.

A linguagem de programação *Python* juntamente com suas bibliotecas, usadas no desenvolvimento de aplicativos móveis, permitiu uma maior liberdade na construção do *layout*, embora ainda existam limitações de compatibilidade entre as bibliotecas ofertadas pela linguagem e seu método de empacotamento, o que se espera que seja resolvido em futuras versões.

O aplicativo desenvolvido fornece cinco funções de entradas e uma visualização gráfica didática que comparativamente se assemelham com os aplicativos encontrados atualmente, em lojas de aplicativos.

Com este trabalho obteve-se uma publicação no XXXIII Congresso de Iniciação Científica – CIC UNESP 2021, cujo resumo consta do Apêndice B.

Por fim, considera-se que este trabalho de graduação alcançou os seus objetivos com a implementação de um aplicativo que está disponível para download no *link*[11] indicado.

Como possibilidades de trabalhos futuros e atualizações, o aplicativo poderia também implementar métodos diferenciados, trazendo opções como a minimização via Método de *Quine-McCluskey* estendido, possibilitando

[11] Download do aplicativo MiniMizer:
https://drive.google.com/file/d/1n3Bh8Rz8SjKEOYIvffc505agsw39F3CJ/view?usp=sharing

minimizações com menor custo utilizando portas *XOR* e *XNOR*, e com a opção de múltiplas saídas. Além disso, será disponibilizado o aplicativo Minimizer para *download,* na *PlayStore*.

8 REFERÊNCIAS

BELANI, G. **Programming languages you should learn in 2020**. IEEEE Computer Society, 2020. Disponível em: <https://www.computer.org/publications/tech-news/trends/programming-languages-you-should-learn-in-2020>. Acesso em: 09 jun. 2021.

BOOLE, G. **An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities**. Dover: Broadway, 1854.424 p.

BRAYTON, R. K.; HATCHTEL, G. D.; MCMULLEN, C. T.; SANGIOVANNI-VINCENTELLI, A. L**. Logic minimization algorithms for VLSI synthesis**. United States: Springer Science & Business Media, 1984. v. 2. 54–138 p.

DAGHLIAN, J. **Lógica e álgebra de Boole**. 4. ed. S.P.: Atlas, 1995. 168 p.

FRANCISCANI, J. F. **Consenso Iterativo: geração de implicantes primos para minimização de funções Booleanas com múltiplas saídas**. 2016. Dissertação (Mestrado em Engenharia Elétrica) - Faculdade de Engenharia, Universidade Estadual Paulista "Júlio de Mesquita Filho", Ilha Solteira, 2016.

GAD, A. F. M. **Building Android Apps in Python Using Kivy With Android Studio: With Pyjnius, Plyer and Buildozer**. 1. ed. Berkeley: Apress, 2019. 422 p.

GRUPO PET-TELE. **Tutorial de Introdução ao Python (Versão: 2k9)**. 2009. Escola de Engenharia - Universidade Federal Fluminense, Niterói, RJ, 2009.

HLAVICKA, J.; FISER, P. B. **A Heuristic Boolean minimizer**. In: IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281), San Jose: IEEE, 2001.

KARNAUGH, M. **The map method for synthesis of combinational logic circuits**. Transactions of the American Institute of Electrial Engineers, [S.1.], v.72, part I, 1953.

KUO, Y.-S.; CHOU, W. **Generating essential primes for a boolean function with multiple-valued inputs**. In: DESIGN AUTOMATION CONFERENCE, 23., 1986, Las Vegas. Conference... United States: IEEE Computer Society, 1986. p. 193–199.

MATTHES, E. **Curso intensivo de python**. São Paulo: Novatec, 2016. 656 p.

MCCLUSKEY, E. J. **Minimization of boolean function**. The Bell System Technical Journal, Kansas, v. 35, p. 1417-1444, 1956.

MENEZES, N. N. C. **Introdução à programação com Python:** algoritmos e lógica de programação para iniciantes. 3. ed. São Paulo: Novatec, 2019. 328 p.

NELSON, V. P.; NAGLE, H. T.; IRWIN, J. D.; CARROLL, B. D. **Digital Logic Circuit Analysis and Design**. 2. ed. Prentice Hall, 1995. 896 p.

NI, M. **Web Service Efficiency at Instagram with Python**. Instagram Engineering, 2016. Disponível em: <https://instagram-engineering.com/web-service-efficiency-at-instagram-with-python-4976d078e366>. Acesso em: 10 nov. 2021.

QUINE, W. V. **The problem of simplifying truth functions**. The American Mathematical Monthly, Mathematical Association of America, United States, v. 59, n. 8, p. 521–531, 1952.

ROMANO, J. **The Complementarity of Multi-wavelength and Multi-messenger Observations**. Texas Tech University, 2019. Disponível em: <https://github.com/jkanner/aapt>. Acesso em: 06 dec. 2021.

SANCHES, A.D.P. **Emprego do método de Quine-McCluskey estendido para gerar circuito mínmo com estruturas ESOP (XOR-XNOR)**. 2017. Dissertação (Mestrado em Engenharia Elétrica) - Faculdade de Engenharia, Universidade Estadual Paulista "Júlio de Mesquita Filho", Ilha Solteira, 2017.

SANTANA, F. **Por que o Python é a Linguagem mais adotada na área de Data Science?**. INSIGHT, 2019. Disponível em: <https://insightlab.ufc.br/por-que-o-python-e-a-linguagem-mais-adotada-na-area-de-data-science/>. Acesso em: 15 nov. 2021.

SHEROMOVA, V. **A brief history of Python**. Exyte, 2020. Disponível em: <https://exyte.com/blog/a-brief-history-of-python>. Acesso em: 06 dec. 2021.

SILVA, A. C. R. da. **Contribuição à síntese de circuitos digitais utilizando programação linear inteira 0 e 1. 1993**. 119 f. Tese (Doutorado em Engenharia Elétrica) -Universidade Estadual de Campinar, Campinas, 1993.

SLAGLE, J. R.; CHANG, C.-L.; LEE, R. C. **A new algorithm for generating prime implicants**. IEEE Transactions on Computers, IEEE, United States, v. 100, n. 4, p.304–310, 1970.

TISON, P. **Generalization of consensus theory and application to the minimization of boolean functions**. IEEE Transactions on Electronic Computers, IEEE, United States, EC–16, n. 4, p. 446–456, 1967.

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. **Sistemas Digitais: Princípios e** Aplicações**.** 12. ed. Pearson Prentice Hall, 2019. 1056 p.

VASILKOV, M. **Kivy Blueprints:** Build your very own app-store-ready, multi-touch games and applications with Kivy!. 1. ed. Birmingham: Packt Publishing Ltd., 2015. 282 p.

APÊNDICE A

EQUAÇÕES BOOLEANAS TESTADAS

$$F_1(A,B,C,D) = \sum m(0,2,4,8,10,12) = \overline{B}\,\overline{D} + \overline{C}\,\overline{D}$$

$$F_2(A,B,C) = \sum m(1,2,3,4,5,6) = \overline{A}\,C + B\,\overline{C} + A\,\overline{B}$$

$$F_3(A,B,C,D) = \sum m(0,1,2,3,4,7) + d(8,9) = \overline{A}\,\overline{B} + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}\,C\,D$$

$$F_4(A,B,C) = \sum m(2,3,7) = \overline{A}\,B + B\,C$$

$$F_5(A,B,C) = \sum m(1,5,7) = \overline{B}\,C + A\,C$$

$$F_6(A,B,C) = \sum m(1,2,7) = \overline{A}\,\overline{B}\,C + \overline{A}\,B\,\overline{C} + A\,B\,C$$

$$F_7(A,B,C) = \sum m(0,3,7) = \overline{A}\,\overline{B}\,\overline{C} + B\,C$$

$$F_8(A,B,C) = \sum m(0,1,3,7) = \ \overline{A}\,\overline{B} + B\,C$$

$$F_9(A,B,C) = \sum m(0,1,2,3,4,5,6,7) = 1$$

$$F_{10}(A) = \sum m(0,1) = 1$$

$$F_{11}(A,B) = \sum m(0,1,2,3,4) = 1$$

$$F_{12}(A,B,C,D) = \sum m(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15) = 1$$

$$F_{13}(A,B,C,D,E)$$

$$= \sum m(0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31)$$

$$= 1$$

$$F_{14}(A,B,C,D,E,F) =$$

$$\sum m\left(\begin{matrix}0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,\\ 22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,\\ 45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63\end{matrix}\right) = 1$$

$$F_{15}(A,B,C,D,E,F,G)$$

$$= \sum m\left(\begin{matrix}0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,\\ 23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,\\ 45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,\\ 67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,\\ 89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,\\ 111,112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127\end{matrix}\right)$$

$$= 1$$

$$F_{16}(A,B,C,D,E,F,G,H)$$

$$= \sum m\left(\begin{matrix}0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,\\ 23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,\\ 45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,\\ 67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,\\ 89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,\\ 111,112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127,128,\\ 129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,145,146,\\ 147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,164,\\ 165,166,167,168,169,170,171,172,173,174,175,176,177,178,179,180,181,\\ 182,183,184,185,186,187,188,189,190,191,192,193,194,195,196,197,198,\\ 199,200,201,202,203,204,205,206,207,208,209,210,211,212,213,214,215,216,\\ 217,218,219,220,221,222,223,224,225,226,227,228,229,230,231,232,233,234,\\ 235,236,237,238,239,240,241,242,243,244,245,246,247,248,249,250,\\ 251,252,253,254,255\end{matrix}\right)$$

$$= 1$$

$$F_{17}(A,B,C,D) = \sum m(2,3,7,8,10,12) = \overline{A}\,C\,D + A\,\overline{C}\,\overline{D} + \overline{B}\,C\,\overline{D}$$

$$F_{18}(A,B,C,D) = \sum m(2,3,10) = \overline{A}\,\overline{B}\,C + \overline{B}\,C\,\overline{D}$$

$$F_{19}(A,B,C,D) = \sum m(1,3,10,12) = \overline{A}\,\overline{B}\,D + A\,\overline{B}\,C\,\overline{D} + A\,B\,\overline{C}\,\overline{D}$$

$$F_{20}(A,B,C,D) = \sum m(1,2,3,4,7,10) = \overline{\mathrm{A}}\,\overline{\mathrm{B}}\,\mathrm{D} + \overline{\mathrm{A}}\,\mathrm{B}\,\overline{\mathrm{C}}\,\mathrm{D} + \overline{\mathrm{A}}\,\mathrm{C}\,\mathrm{D} + \overline{\mathrm{B}}\,\mathrm{C}\,\overline{\mathrm{D}}$$

$$F_{21}(A,B,C,D) = \sum m(1,2,3,5,11,13) = \overline{A}\,\overline{B}\,C + \overline{B}\,C\,D + B\,\overline{C}\,D + \overline{A}\,\overline{B}\,D$$

$$F_{22}(A,B,C,D,E) = \sum m(0,10,22,25,26,28,30,31)$$
$$= \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,\overline{E} + B\,\overline{C}\,D\,\overline{E} + A\,C\,D\,\overline{E} + A\,B\,\overline{C}\,\overline{D}\,E + A\,B\,C\,\overline{E} + A\,B\,C\,D$$

$$F_{23}(A,B,C,D,E) = \sum m(1,2,3,7,10,12,13,14,15,20,21)$$
$$= \overline{A}\,\overline{B}\,\overline{C}\,E + \overline{A}\,B\,C + A\,\overline{B}\,C\,\overline{D} + \overline{A}\,\overline{C}\,D\,\overline{E} + \overline{A}\,\overline{B}\,D\,E$$

$$F_{24}(A,B,C,D,E,F)$$
$$= \sum m(1,2,3,7,15,21,33,34,35,36,37,40,41,42,44,62,63)$$
$$= \overline{B}\,\overline{C}\,\overline{D}\,F + \overline{B}\,\overline{C}\,\overline{D}\,E + \overline{A}\,\overline{B}\,D\,E\,F + \overline{A}\,B\,\overline{C}\,D\,\overline{E}\,F + A\,B\,C\,D\,E$$
$$+ A\,\overline{B}\,\overline{C}\,D\,\overline{E} + A\,\overline{B}\,C\,\overline{D}\,\overline{E} + A\,\overline{B}\,\overline{D}\,E\,\overline{F} + A\,\overline{B}\,D\,\overline{E}\,\overline{F}$$

$$F_{25}(A,B,C,D,E,F)$$
$$= \sum m(0,1,2,3,4,13,14,15,16,28,29,30,31,60,61,62,63)$$
$$= \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,\overline{E}\,\overline{F} + \overline{A}\,C\,D\,F + \overline{A}\,C\,D\,E + \overline{A}\,\overline{C}\,\overline{D}\,\overline{E}\,\overline{F} + B\,C\,D$$

$$F_{26}(A,B,C) = \sum m(2,3,7) + d(0,1) = \overline{A} + B\,C$$

$$F_{27}(A,B,C) = \sum m(2,3,7) + d(0,5) = \overline{A}\,B + B\,C$$

$$F_{28}(A,B,C) = \sum m(1,5,7) + d\,(6) = \overline{B}\,C + A\,C$$

$$F_{29}(A,B,C) = \sum m(1,2,4) + d(0,3,7) = \overline{A} + \overline{B}\,\overline{C}$$

$$F_{30}(A,B,C) = \sum m(0,1,6) + d(2,3,4,5,7) = 1$$

$$F_{31}(A,B,C,D) = \sum m(1,2,3,4,5,6,15) + d(12,13) = A\,B\,D + \overline{A}\,\overline{B}\,D + \overline{A}\,C\,\overline{D} + B\,\overline{C}$$

$$F_{32}(A,B,C,D) = \sum m(0,1,13,14,15) + d(3,7) = \overline{A}\,\overline{B}\,\overline{C} + A\,B\,D + A\,B\,C$$

$$F_{33}(A,B,C,D) = \sum m(2,3,7,10,11) + d(15) = \overline{B}C + CD$$

$$F_{34}(A,B,C,D) = \sum m(2,3,7,10,11) + d(0,15) = \overline{B}\,C + C\,D$$

$$F_{35}(A,B,C,D) = \sum m(0,1,4,8,10) + d(7,15) = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{C}\,\overline{D} + A\,\overline{B}\,\overline{D}$$

$$F_{36}(A,B,C,D) = \sum m(2,3,7,11) + d(1,10) = \overline{A}\,C\,D + \overline{B}\,C$$

$$F_{37}(A,B,C,D) = \sum m(0,1,6,10) + d(3,4) = \overline{A}\,B\,\overline{D} + A\,\overline{B}\,C\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}$$

$$F_{38}(A,B,C,D) = \sum m(2,9,11,13) + d(0,1) = \overline{A}\,\overline{B}\,\overline{D} + A\,\overline{B}\,D + A\,\overline{C}\,D$$

$$F_{39}(A,B,C,D) = \sum m(10,11) + d(14,15) = A\,C$$

$$F_{40}(A,B,C,D) = \sum m(0,1,8,11) + d(2,3,4,5,6,7,9,10,12,13,14,15) = 1$$

$$F_{41}(A,B,C,D,E) = \sum m(0,1,5,11,16,20) + d(2,15,31)$$
$$= \overline{A}\,\overline{B}\,\overline{D}\,E + A\,\overline{B}\,\overline{D}\,\overline{E} + \overline{A}\,B\,D\,E + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

$$F_{42}(A,B,C,D,E) = \sum m(2,3,7,10,21,22) + d(30)$$
$$= \overline{A}\,\overline{B}\,D\,E + \overline{A}\,\overline{C}\,D\,\overline{E} + A\,\overline{B}\,C\,\overline{D}\,E + A\,C\,D\,\overline{E}$$

$$F_{43}(A,B,C,D,E) = \sum m(2,3,6,10,31) + d(0,7,11) = \overline{A}\,\overline{B}\,D + \overline{A}\,\overline{C}\,D + A\,B\,C\,D\,E$$

$$F_{44}(A,B,C,D,E) = \sum m(1,3,7,10,18) + d(20,21,22,23,25)$$
$$= \overline{A}\,\overline{B}\,\overline{C}\,E + \overline{A}\,B\,\overline{C}\,D\,\overline{E} + A\,\overline{B}\,D\,\overline{E} + \overline{A}\,\overline{B}\,D\,E$$

$$F_{45}(A,B,C,D,E) = \sum m(0,1,7,15,30) + d(8,31)$$
$$= \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,B\,C\,D\,E + A\,B\,C\,DE$$

$$F_{46}(A,B,C,D,E,F) = \sum m(1,2,3,7,16,31,52) + d(44,45,50)$$
$$= \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,F + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,E + \overline{A}\,\overline{B}\,\overline{C}\,E\,F + \overline{A}\,B\,\overline{C}\,\overline{D}\,\overline{E}\,\overline{F} + \overline{A}\,B\,C\,D\,E\,F$$
$$+ A\,B\,\overline{C}\,D\,\overline{E}\,\overline{F}$$

$$F_{47}(A,B,C,D,E,F,) = \sum m(0,1,6,15,24,63) + d(7,11,22)$$
$$= \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}\,\overline{E} + \overline{A}\,B\,C\,\overline{D}\,\overline{E}\,\overline{F} + A\,B\,C\,D\,E\,F + \overline{A}\,\overline{B}\,\overline{C}\,D\,E + \overline{A}\,\overline{B}\,D\,E\,F$$

$$F_{48}(A,B,C,D,E,F) = \sum m(12,16,25,26,31,40,55,56,59) + d(15,63)$$
$$= \overline{A}\,\overline{B}\,C\,D\,\overline{E}\,\overline{F} + \overline{A}\,B\,\overline{C}\,\overline{D}\,\overline{E}\,\overline{F} + \overline{A}\,B\,C\,\overline{D}\,\overline{E}\,F + \overline{A}\,B\,C\,\overline{D}\,E\,\overline{F}$$
$$+ A\,C\,\overline{D}\,\overline{E}\,\overline{F} + A\,B\,D\,E\,F + A\,B\,C\,E\,F + \overline{A}\,C\,D\,E\,F$$

$$F_{49}(A,B,C,D,E,F)$$
$$= \sum m\begin{pmatrix}0,1,2,3,4,5,6,8,9,10,11,12,13,14,15,16,17,18,19,20,21,\\ 22,23,24,25,26,27,28,29,39,40,41,42,43,44,\\ 45,46,50,51,52,53,54,55,56,57,58,59,61,62\end{pmatrix}$$
$$+ d(7,63)$$
$$= \overline{A}\,\overline{B} + \overline{A}\,\overline{E} + \overline{C}\,D\,E\,F + C\,\overline{D} + B\,\overline{C}\,D + B\,\overline{C}\,E + A\,C\,E\,\overline{F} + \overline{B}\,C\,\overline{E}$$
$$+ B\,D\,\overline{E}\,F$$

$$F_{50}(A,B,C,D,E,F)$$
$$= \sum m\begin{pmatrix}0,1,2,3,4,5,6,8,9,10,11,12,13,14,15,16,17,18,19,20,21,\\ 22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,\\ 45,46,50,51,52,53,54,55,56,57,58,59,61,62\end{pmatrix}$$
$$+ d(7,47,48,49,60,63) = 1$$

APÊNDICE B

RESUMO - CIC UNESP 2021

Aplicativo Android para minimização de equações booleanas.

Alan Soares de Sousa, Suely Cunha Amaro Mantovani, UNESP, Campus de Ilha Solteira, Engenharia Elétrica, alaansousa7@gmail.com.

Palavras Chave: *Equações booleanas, Python, smartphone.*

Introdução

A minimização de equações booleanas possibilita a redução do número de literais e portas lógicas na construção de um circuito digital, resultando na menor dissipação de calor e área ocupada, assuntos de grande interesse na busca pelo alto desempenho e eficiência de um circuito integrado. Destacam-se como métodos utilizados na minimização o Mapa de *Karnaugh* e o Algoritmo de *Quine-McCluskey*[1], que é otimizado para aplicações computacionais, trazendo praticidade ao usuário, quando desenvolvidos como aplicativos de smartphone.

Objetivo

Criar um aplicativo (app) para *Android* capaz de realizar a minimização de funções booleanas, usando o algoritmo de *Quine-McCluskey* e o desenho do circuito.

Material e Métodos

O funcionamento do projeto é resumido no diagrama de blocos da Figura 1, neste nota-se a integração entre os dois *scripts* principais, ambos utilizando *Python* e suas bibliotecas. O *script* do algoritmo recebe os dados de entrada do usuário, que podem ser de uma tabela verdade, declaração de saídas lógicas '1' e *don't cares* ou declaração de equação booleana, com limite de 20 variáveis, e com esses dados realiza as iterações do método de minimização de Quine-*McCluskey*, armazenando os resultados recursivos em listas, e detalhes, do processo de minimização ao usuário. O resultado final de saída é a soma de produtos de mínimo custo obtida com o auxílio da biblioteca *SymPy*. O *script* do app faz a conexão entre o usuário e o algoritmo, utilizando recursos fornecidos pelas bibliotecas *Kivy*[2] e *KivyMD*. Também fornece a opção de desenho do circuito, onde o usuário deve entrar com a equação do circuito desejado, para isso utiliza-se a biblioteca *Schemdraw*.

Resultados e Discussão

O app desenvolvido foi testado em *Android* 10 e versões mais atuais, porém possui compatibilidade de recursos com versões de *Android* acima da 2.2. Apresenta, portanto, minimização com entradas por 'Tabela verdade', por 'mintermos', 'equações booleanas'. Também mostra as iterações realizadas seguindo as combinações do algoritmo de *Quine-McCluskey*. Têm-se na Figura 2 algumas opções de tela do app.

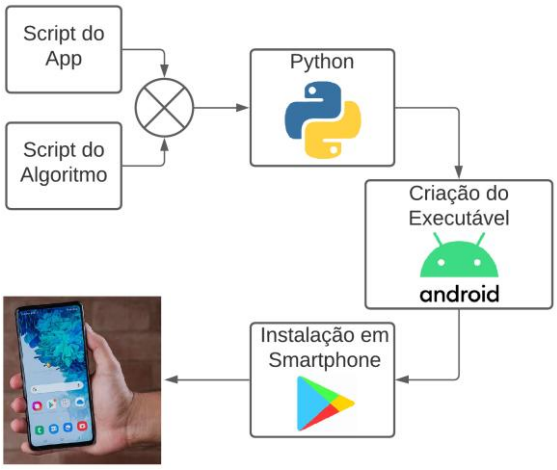


Figura 1. Diagrama de blocos da criação do app.

(a) (b) (c)

Figura 2. Telas do aplicativo: a) Tela inicial. b) Tabela verdade. c) Desenho do circuito: A&B|A&C.

Conclusão

O aplicativo Android desenvolvido, baseado na linguagem *Python*, representa uma ferramenta lúdica e didática para auxiliar e motivar os estudantes de engenharia ou ensino técnico, no aprendizado em disciplinas de circuitos digitais.

[1] McCluskey, Edward J. Minimization of boolean function. The Bell System Technical Journal, Kansas, v. 35, p. 1417-1444, **1956**.

[2] Gad, Ahmed F. M. Building Android Apps in Python Using Kivy With Android Studio: With Pyjnius, Plyer and Buildozer. 1.ed. Berkeley: Apress, **2019**.