UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE SÃO JOÃO DA BOA VISTA

GABRIEL GOMES NOGUEIRA

Utilização de circuitos caóticos aplicados em meios de comunicação para maior confidencialidade entre transmissor-receptor: estudo teórico e prático na área de Telecomunicações

São João da Boa Vista
2019

Nogueira, Gabriel Gomes

Utilização de circuitos caóticos aplicados em meios de comunicação para maior confidencialidade entre transmissor-receptor: estudo teórico e prático na área de Telecomunicações / Gabriel Gomes Nogueira. -- São João da Boa Vista, 2019.
72 p. : il. color.

Trabalho de Conclusão de Curso – Câmpus Experimental de São João da Boa Vista – Universidade Estadual Paulista "Júlio de Mesquita Filho".
Orientador: Prof. Dr. André Alves Ferreira
Bibliografia
1. Arduino (Controlador programável) 2. Circuitos elétricos 3. Circuitos elétricos não-lineares 4. Circuitos eletrônicos 5. Codificação 6. Comportamento caótico nos sistemas 7. Eletrônica 8. Telecomunicações

CDD 23. ed. – 621.382

Ficha catalográfica elaborada pela Biblioteca-BJB
Bibliotecário responsável: João Pedro Alves Cardoso – CRB-8/9717

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CÂMPUS EXPERIMENTAL DE SÃO JOÃO DA BOA VISTA
GRADUAÇÃO EM ENGENHARIA ELETRÔNICA E DE TELECOMUNICAÇÕES

TRABALHO DE CONCLUSÃO DE CURSO

UTILIZAÇÃO DE CIRCUITOS CAÓTICOS APLICADOS EM MEIOS DE COMUNICAÇÃO PARA MAIOR CONFIDENCIALIDADE ENTRE TRANSMISSOR-RECEPTOR: ESTUDO TEÓRICO E PRÁTICO NA ÁREA DE TELECOMUNICAÇÕES

Aluno: Gabriel Gomes Nogueira

Orientador: Prof. Dr. André Alves Ferreira

Banca Examinadora:
- André Alves Ferreira (Orientador)
- Edgar Eduardo Benitez Olivo (Examinador)
- Paula Ghedini Der Agopian (Examinador)

A ata da defesa com as respectivas assinaturas dos membros encontra-se no prontuário do aluno (Expediente nº 46/2018)

São João da Boa Vista, 20 de agosto de 2019

DADOS CURRICULARES

GABRIEL GOMES NOGUEIRA

NASCIMENTO	05/04/1995 - Franca / SP
FILIAÇÃO	Ricardo Padovan Nogueira Ana Elisa Gomes Nogueira
2013 / 2019	Graduação em Engenharia Eletrônica e de Telecomunicações UNESP-SJBV

Dedico este trabalho a minha família.
Sem o apoio deles provavelmente eu não chegaria tão longe.

AGRADECIMENTOS

Primeiramente a Deus por permitir que eu esteja aqui. Seja em vida, seja na universidade.

Ao meu orientador prof. dr. André Alves Ferreira pela dedicação e paciência durante todo o desenvolvimento deste trabalho.

A Universidade Estadual Paulista Júlio de Mesquita Filho por me acolher durante meus anos de graduação.

A unidade da UNESP de São João da Boa Vista e todos os colaboradores por todo o apoio e dedicação em proporcionar um ambiente agradável e propício ao aprendizado e desenvolvimento de tecnologia.

A minha família por me dar todo apoio necessário para percorrer este caminho até hoje e sempre me encorajar a continuar por mais árduo que tenha sido em alguns momentos.

Por fim, gostaria de agradecer a todos que fizeram parte de minha vida acadêmica, dentro ou fora de sala de aula, pois todos foram importantes na minha formação e por quem eu sou hoje. Sou eternamente grato a todos que participaram da realização deste sonho.

O maior bem do homem é uma mente inquieta.

- Isaac Asimov

RESUMO

Por meio do estudo de circuitos não-lineares, desde sua concepção teórica, simulação em ambiente computacional e desenvolvimento prático, este projeto tem como objetivo principal adicionar maior confidencialidade em um canal de comunicação entre um transmissor e um receptor, utilizando um sinal considerado caótico como base para codificação de informações. Para isso, inicialmente foi desenvolvida uma pesquisa teórica a fim de identificar o estado da arte a respeito da criptografia em camada física, sincronização e aplicações práticas utilizando circuitos caóticos. Validadas cada uma das teorias, o Circuito de Chua foi escolhido e simulado em ambiente computacional por apresentar um sinal com comportamento não previsível e altamente sensível às condições iniciais, em diferentes situações, utilizando diferentes componentes eletrônicos. Em seguida, as simulações feitas foram validadas por montagens em laboratório utilizando *protoboards* e componentes discretos tais como capacitores, amplificadores operacionais, potenciômetros e resistores. Os resultados então foram comparados com os observados em simulação e verificou-se a possibilidade de uso e limitações no cenário atual de telecomunicações. Considerando transmissão em meios digitais, obteve-se resultados satisfatórios com a codificação de uma mensagem em caracteres digitais através de um microcontrolador, que serão mais detalhadamente abordados ao longo deste trabalho.

PALAVRAS-CHAVE: CAOS. CIRCUITOS NÃO-LINEARES. CAMADA FÍSICA. SINCRONIZAÇÃO. CRIPTOGRAFIA. CODIFICAÇÃO. ALEATORIEDADE.

ABSTRACT

Through the study of nonlinear circuits, from its theoretical conception, simulation in computational environment and practical development, this project has as main objective to add greater confidentiality in a channel of communication between a transmitter and a receiver using a signal considered chaotic as basis for information encoding. For this, a theoretical research was initially developed to identify the state of the art regarding physical layer encryption, synchronization and practical applications using chaotic circuits. Chua Circuit was chosen and simulated in a computational environment because it can present a signal with behavior not predictable and highly sensitive to the initial conditions, in different situations using different electronic components. Then, the simulations were validated with laboratory prototyping using protoboards and discrete components such as capacitors, operational amplifiers, potentiometers and resistors. The results were then compared with those observed in simulation, and it was verified the possibility of use and its limitations in the current telecommunications scenario, considering transmission in digital media. Satisfactory results were obtained, sending a digital coded message through a microcontroller, which will be covered throughout this work.

KEYWORDS: CHAOS. NONLINEAR CIRCUITS. PHYSICAL LAYER. SYNCHRONIZATION. CRYPTOGRAPHY. CODIFICATION. RANDOMNESS.

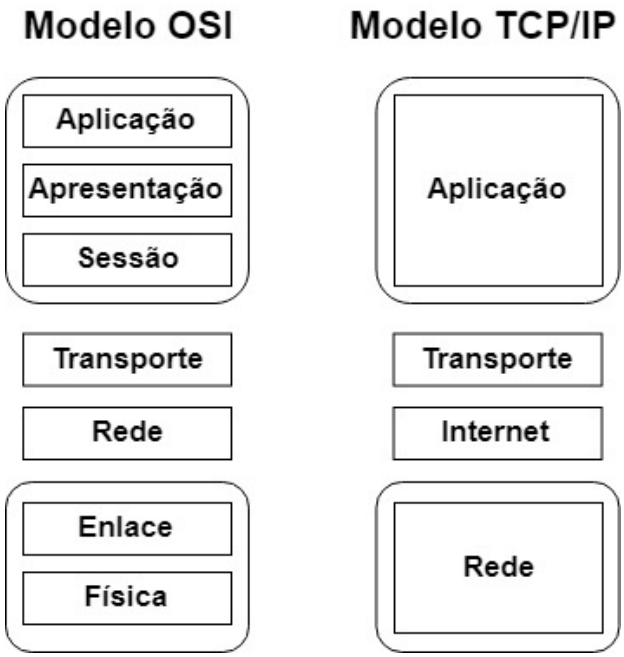
SUMÁRIO

1 INTRODUÇÃO

Com o crescente avanço e uso da tecnologia no dia-a-dia e a forma como cada vez mais dados importantes são transmitidos a todo momento, surge também a necessidade de protegê-los de potenciais bisbilhoteiros (do termo inglês "*eavesdropper*") que podem obter acesso à essas informações de maneira ilegal e causar grandes danos tanto a pessoas, como a grandes empresas e corporações (KONG; KADDOUM; TAHA, 2015). A maioria das pesquisas e desenvolvimentos realizados em segurança da transmissão de dados, seguindo o Modelo OSI (Figura 1), estão concentrados nas seguintes camadas: Rede, Sessão, Apresentação e Aplicação.

O modelo OSI vem sendo cada vez mais difundido por estratificar mais especificamente cada uma das diferentes funções desempenhadas pelo processo de transmissão de informações de redes em relação ao tradicional modelo TCP/IP (MARMITT, 2012; MAURITY, 2016).

Figura 1 – Modelo OSI (Open Systems Interconnect) e TCP/IP de redes



Fonte: Autoria própria.

Neste trabalho, o foco será nas camadas **Física**, **Enlace** e **Apresentação** do modelo OSI, as quais não são tratadas separadamente no modelo TCP/IP.

Sucintamente, cada camada do modelo OSI possui sua devida importância:

- **Camada Física:** Diz respeito aos meios de conexão pelos quais os dados são gerados (hardware) e irão trafegar (via cabos coaxiais, fibra ou pelo ar). (MARMITT, 2012; MAURITY, 2016)
- **Camada de Enlace:** ou Camada de Ligação de Dados, é responsável pela codificação (modulação) dos dados para que sejam transportados via camada física, opcionalmente detecta e corrige erros que possam vir a ocorrer no nível físico. (MARMITT, 2012; MAURITY, 2016)

- **Camada de Rede:** É responsável pelo endereçamento, transporte e roteamento dos pacotes enviados. (MARMITT, 2012; MAURITY, 2016)
- **Camada de Transporte:** Responsável pela ligação total entre o bloco de Rede e o bloco de Sessão. Cuida da fragmentação dos dados em pequenos pacotes e verifica se foram transmitidos corretamente, garantindo um serviço com custo-benefício dentro do esperado. (MARMITT, 2012; MAURITY, 2016)
- **Camada de Sessão:** Permite que duas aplicações em hardwares diferentes estabeleçam e mantenham uma sessão, (iniciar um diálogo entre duas máquinas: verificar se o outro lado está pronto pra se comunicar, identificar-se, etc.) ou reestabelecer a comunicação caso esta seja interrompida com o mínimo de perda de informações possível. (MARMITT, 2012; MAURITY, 2016)
- **Camada de Apresentação:** ou Camada de Tradução, é responsável pela representação dos dados, convertendo o formato da informação recebida em um formato que será entendido pelo protocolo utilizado. (MARMITT, 2012; MAURITY, 2016)
- **Camada de Aplicação:** Corresponde às aplicações (programas, softwares), no topo da camada OSI que serão utilizados para promover uma interação entre a máquina destinatária e o usuário da aplicação. (MARMITT, 2012; MAURITY, 2016)

Com a necessidade de constantemente aprimorar a segurança em cada uma dessas camadas, novos meios de cifragem representam opções extremamente vantajosas para o sistema como um todo. A criptografia na camada física, apesar de não ser um campo novo, ainda é extremamente promissor por propiciar uma forma de codificação de mensagem baseado em condições não-digitais (JIANG et al., 2013).

2 MOTIVAÇÃO E JUSTIFICATIVA

A criptografia pode ser entendida como um conjunto de métodos e técnicas para codificar informações legíveis através de um algoritmo, convertendo um texto original em um texto ilegível, sendo possível através do processo inverso recuperar as informações originais (MORENO; PEREIRA; BARROS, 2006).

A criptologia é tão antiga quanto a própria escrita, pois sempre houve fórmulas e informações confidenciais que não deveriam cair no domínio público ou em mãos erradas. Segundo Kahn (1967), o primeiro exemplo de escrita cifrada aconteceu aproximadamente em 1900a.C, quando o escriba Khnumhotep II teve a idéia de substituir algumas palavras ou trecho de texto(MORENO; PEREIRA; BARROS, 2006). Em 50 a.C, Júlio César utilizou-se de sua famosa cifra (conhecida como Cifra de César ou Cifra de Troca) para cifrar informações governamentais. Na composição de César, cada letra era desviada em 3 posições, "A"se tornava "D", "B"se tornava "E"e assim por diante, este método de cifragem é utilizado até hoje (MORENO; PEREIRA; BARROS, 2006).

Cada vez mais a população utiliza-se de aparelhos eletrônicos, tais como computadores, *smartphones* e até relógios inteligentes que facilitam ou até mesmo executam tarefas que, manualmente, levariam muito mais tempo para serem cumpridas. A maioria absoluta de todos os dados e informações, produto da análise computacional (GÖHRING; SCHMITZ, 2015), quando criptografados são cifrados em meio virtual, seja pelo sistema operacional que proporciona esta facilidade, por um aplicativo ou software terceirizado que exerce essa função ou por um simples algorítimo para “embaralhar” a informação original.

O interesse em desenvolver novos métodos de cifrar informações está no crescente número de conteúdo gerado, de pesquisas sendo feitas, e de tecnologia sendo desenvolvida a um passo cada vez mais rápido. Paralelamente, a tecnologia criada para acessar estas informações sem permissão, para uso indevido (roubo de informações confidenciais, códigos de acesso restrito, número e senhas de contas bancárias, entre outros.), também cresce, com constantes tentativas de invasões a bancos de dados de grandes multinacionais, como a Adobe e Sony, que tiveram dados de usuários expostos nos últimos anos.

Outro exemplo é a vulnerabilidade no código de criptografia Open SSL, um código open-source de implementação dos protocolos SSL (Secure Sockets Layer ou Camada) e TLS (Transport Layer Security), chamada de Heartbleed (US-CERT, 2014; TEAM, 2014), que deixou vulneráveis aproximadamente 17% de todos os servidores virtuais conectados à internet (Incluindo grandes corporações como o Facebook, Google e Yahoo), além de softwares e jogos.

Em 2018 os assuntos referentes a segurança de comunicação silenciosamente se moldam em grandes preocupações governamentais acerca da privacidade de dados sigilosos em uma era totalmente digital. Acessar esse tipo de informação se tornou uma arma poderosa, e *hackers* não são mais apenas curiosos na área da informática, são pessoas ligadas à área de inteligência ou militar. Algumas fontes sugerem que uma Terceira Guerra Mundial irá acontecer no mundo cibernético, sem o uso de armas de fogo, sendo a posse de informações confidenciais de um Estado o maior material de poder em uma

situação de tensão (LEE, 2018).

Apesar de haver algum tipo de segurança em cada uma das camadas do modelo OSI de redes, a concentração maior de protocolos de segurança e medidas "*anti-eavesdroppers*" se encontram nas camadas de Rede (transporte e endereçamento dos pacotes), Sessão (estabelece diálogo entre duas máquinas) e Aplicação (softwares, browsers, e-mails, trocas de mensagens, aplicativos, etc.).

Graças à essa concentração de medidas de segurança em certas camadas, o campo da criptologia nos níveis mais baixos do sistema de transmissão de dados, aliado à osciladores caóticos, ao mesmo tempo que pouco explorado, é extremamente promissor (GÖHRING; SCHMITZ, 2015; KONG; KADDOUM; TAHA, 2015) devido ao baixo nível de pesquisa realizada neste campo (relativo e proporcional à pesquisas realizadas na área de segurança em camadas superiores do modelo OSI).

Durante as últimas duas décadas, a teoria do caos no campo da segurança de comunicação vem ganhando atenção em vários ramos da engenharia devido a sua não-usualidade nas tradicionais aplicação de criptografia. A possibilidade de sincronização do caos também foi decisivo, abrindo novas possibilidades nessa área (BANERJEE; KURTHS, 2014).

A ideia de utilizar um circuito com comportamento caótico como encriptador se torna cada vez mais atraente uma vez que, por possuir uma semente dependente de condições inciais físicas e não digitais, possibilita a criação de um gerador realmente aleatório de sinais, bits ou códigos (KONG; KADDOUM; TAHA, 2015).

Em tese, sem acesso ao circuito original seria impossível gerar um sinal exatamente igual ao do oscilador caótico, entretanto por possuir um padrão e ser possível a sincronização de dois circuitos (GÁMEZ-GUZMÁN et al., 2009), existem inúmeras possíveis aplicações, em especial na área de comunicação de maneira extremamente segura, podendo ser implementada para garantir transmissão de dados sigilosa.

Neste trabalho Caos será definido como o comportamento de um dado modelo matemático caracterizado por: Imprevisibilidade de estados posteriores do sistema após intervalos de tempo relativamente longos, dependência das condições iniciais, o fato do conjunto ser topologicamente transitivo e existência densa de pontos periódicos no domínio do conjunto (padrão). Estas condições já foram enfatizadas no estudo matemático de funções e sistemas caóticos, tais como o Atrator de Rössler ou o Atrator de Lorenz, exemplos mais comuns e conhecidos quando se refere à modelagem matemática de sistemas caóticos (SANTANA; TERRA, 2005).

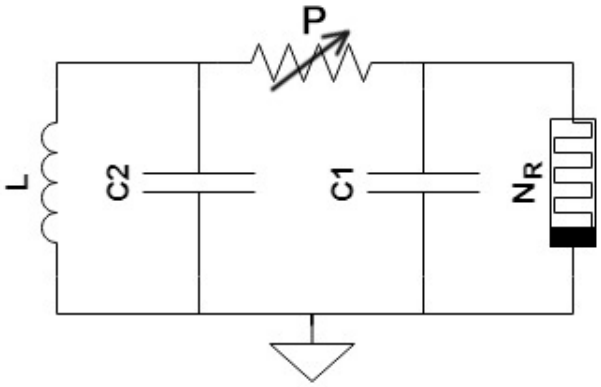
Circuitos compostos por componentes eletro-eletrônicos padrão devem satisfazer três condições para exibição de comportamento caótico (KENNEDY, 1993):

- Um ou mais elementos não lineares;
- Um ou mais resistores;
- Três ou mais elementos armazenadores de energia.

O circuito mais simples desenvolvido até os dias de hoje, que satisfazem os critérios anteriores é chamado de Circuito de Chua (CHUA, 1992), composto essencialmente de um Diodo de Chua (Elemento não-linear, N_R), um potenciômetro (P), dois capacitores(C_1 e C_2) e um indutor (L),

totalizando três elementos armazenadores de energia para satisfazer os critérios mínimos definidos para exibição do comportamento caótico (Figura 2).

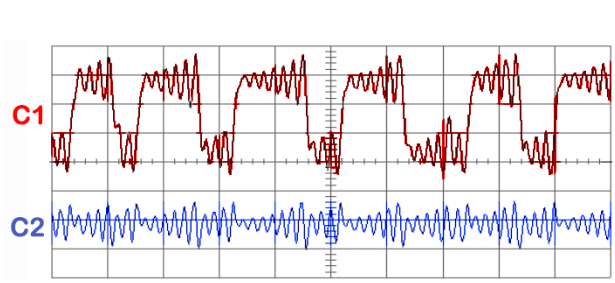
Figura 2 – Circuito de Chua tradicional



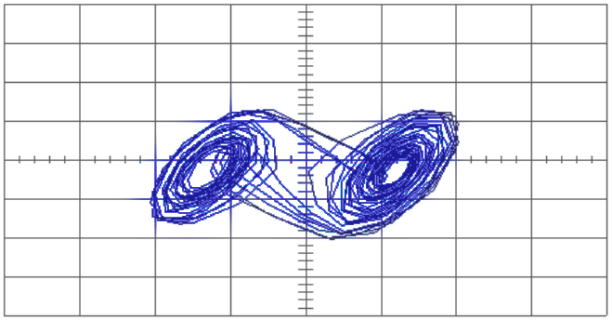
Fonte: Autoria própria, baseado em (CHUA, 1992)

Os modelos mais conhecidos do Circuito de Chua para geração de um atrator duplo (KILIÇ, 2010) (obtido pela representação da tensão sobre o primeiro capacitor no eixo X e a tensão sobre o segundo capacitor no eixo Y) (Figura 3), ou "double scroll" em tradução livre do inglês, possuem frequência de oscilação na faixa dos kHz (KILIÇ, 2010), o que é inviável para transmissão de informações em tempo real seguindo os padrões atuais de tecnologia comercial. Entretanto, estudos sugerem a possibilidade de frequências quase na ordem dos Gigahertz utilizando circuitos caóticos e parâmetros ideais(JOTHIMURUGAN et al., 2014; DEMIRKOL et al., 2007).

Figura 3 – Tensão observada sobre cada capacitor do circuito de Chua



(a) Tensão observada no domínio do tempo



(b) Tensão observada em XY

Fonte: (KILIÇ, 2010), modificado

Os modelos mais comuns do Circuito de Chua, utilizam amplificadores operacionais de baixo custo (como o LM741) e consequentemente baixa frequência de operação, inviabilizando a codificação por chaveamento caótico (Chaos Shift Keying – CSK) (DEDIEU; KENNEDY; HASLER, 1993).

Estes valores de frequência, e até maior tolerância a valores mais altos de tensão podem ser melhorados com amplificadores operacionais mais rápidos e configurações diferentes, resultando em uma saída com uma maior gama de valores aleatórios em função da tensão aplicada (SIDERSKIY; KAPILA, 2014; LÓPEZ, 2012). Com isto, o presente trabalho vem colaborar no campo da segurança da informação, com o intuito de se tornar ponto de partida para posteriores estudos realizados na área.

2.1 METODOLOGIA DE PESQUISA

O trabalho de conclusão de curso em questão utiliza-se das metodologias exploratória e descritiva na quase totalidade de seus tópicos. Isto é, os assuntos são tratados de acordo com a seguinte sequência:

- O assunto é investigado e levantadas informações relevantes sobre o estado da arte a respeito de determinado tema, a fim de entender como funciona;
- O assunto é então descrito baseado em referências utilizadas e ligado ao tema do trabalho de acordo com sua aplicação.

Para a realização do trabalho foram executadas as seguintes etapas:

- Levantamento de dados a respeito do estado da arte da aplicação de circuitos caóticos em criptografia;
- Escolha do Circuito de Chua para o projeto, levando em conta a disponibilidade de componentes, aplicação e viabilidade;
- Simulação, prototipagem e avaliação dos resultados do Circuito de Chua;
- Simulação, prototipagem e avaliação do gerador de bits para codificação;
- Aplicação de codificação e decodificação com o sistema transmissor-receptor e avaliação de resultados.

3 O CAOS APLICADO EM CIRCUITOS ELETRÔNICOS

Historicamente, o estudo do caos teve início nas áreas da matemática e física, posteriormente sendo estendido para áreas como engenharia, informática, ciências sociais, entre outras (FIEDLER-FERRARA; PRADO, 1994).

Como abordado em (STACEY, 2016), o Caos não significa desordem absoluta, significa que sistemas guiados por certos tipos de leis perfeitamente ordenadas são capazes de se comportar de uma maneira aleatória e, desta forma, completamente imprevisível no longo prazo, em um nível específico. Portanto, o caos pode ser entendido como uma "aleatoriedade com padrão de similaridade"(PAIVA, 2010).

Inúmeros sistemas apresentam comportamentos variáveis, irregulares ou imprevisíveis seja a longo ou curto prazo desde a determinação do início do funcionamento do mesmo. Efeitos não-lineares e sensibilidade às condições iniciais são elementos presentes em quase todo sistema real e considerar tais efeitos em avaliações sejam reais ou virtuais é vital para o entendimento da dinâmica de um conjunto de fenômenos.

Assim, mesmo que sistemas possam ser modelados satisfatoriamente por equações deterministas (sem ruídos ou incertezas explícitas), estes podem a longo prazo evoluir de maneira não previsível por não considerar efeitos não lineares e/ou sensibilidade às condições iniciais (LAM, 1998).

Na teoria de sistemas dinâmicos, a presença de caos é caracterizada pela forte dependência em relação às condições iniciais de um sistema. Entretanto, a compreensão de fenômenos caóticos tem em sua motivação a necessidade do entendimento do papel das instabilidades responsáveis por gerar comportamentos não-periódicos e/ou instáveis em sistemas dinâmicos não lineares (KELLERT, 1994). Neste contexto, é possível considerar o comportamento de alguns materiais semicondutores como responsáveis pelas instabilidades em sistemas de natureza elétrica, pela variação de componentes de campo elétrico externo, irradiação ou injeção de corrente, entre outros fatores.

A presença de fortes estímulos colocam o sistema fora de equilíbrio, ocasionando o processo de geração e recombinação de elétrons e lacunas-livres, fazendo com que o sistema apresente características espaciais não-uniformes em termos de distribuição de campo e cargas livres.

3.1 TIPOS DE CIRCUITOS CAÓTICOS

Circuitos eletrônicos podem ser lineares ou não-lineares. A vasta variedade de circuitos não-lineares compreende uma quantidade de circuitos com diferentes comportamentos. Circuitos autônomos não-lineares podem ser classificados por um conjunto de equações que descrevem seu comportamento (OGORZAŁEK, 1997). As soluções podem ser:

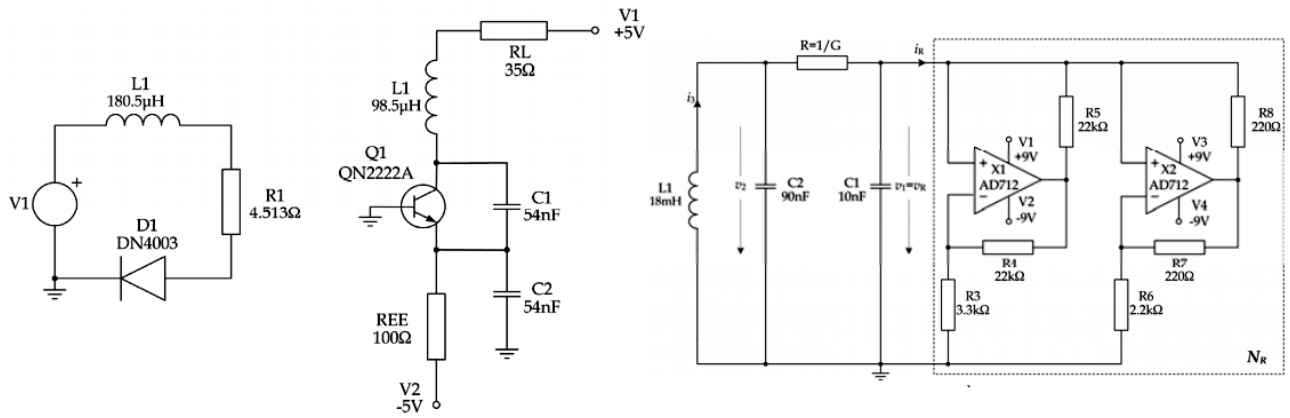
- Convergir para um único ponto de equilíbrio (filtros RLC, amplificadores, etc.);
- Convergir para um dos vários pontos de equilíbrio (circuitos bi-estáveis, memórias, schimitt-trigger, etc.);

- Ser periódico ou quase periódico (osciladores, geradores, sinais periódicos, etc).

As soluções acima podem ser descritas como um comportamento "normal"de circuito (OGORZAŁEK, 1997).

Nas últimas décadas, o comportamento observado e descrito como caótico vem sido mencionados em diversos artigos científicos(ŠALAMON, 2012), utilizando-se de circuitos RLC (resistores, indutores e capacitores), osciladores, acionamento capacitivo, filtros digitais, flip-flops, fontes, conversores entre outros, como abordado em (ŠALAMON, 2012). Este tipo de comportamento não pode ser modelado apenas por um conjunto de equações, pois são altamente sensíveis a condições iniciais e parâmetros individuais.

Figura 4 – Exemplos de circuitos com comportamento caótico. Em ordem da esquerda para a direita: ressonador com diodo, oscilador de Colpitts e oscilador de Chua

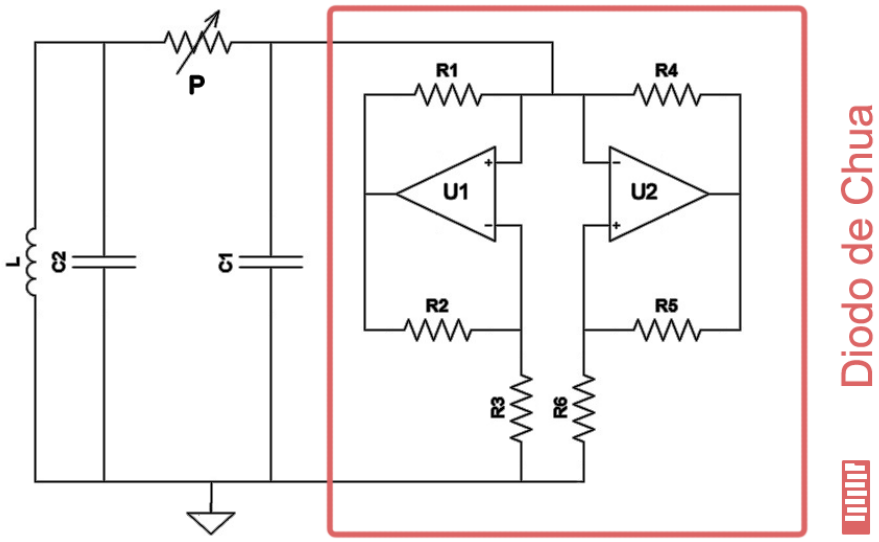


Fonte: (ŠALAMON, 2012)

3.2 O CIRCUITO DE CHUA

O circuito desenvolvido por Leon O. Chua em 1983 (KILIÇ, 2010) (Figura 5) consiste de componentes eletrônicos simples: resistores, um indutor, capacitores e amplificadores operacionais. O indutor L e o capacitor C2 formam um circuito ressonante, em que seus valores determinam a frequência básica de oscilação.

Figura 5 – Exemplo de Circuito de Chua

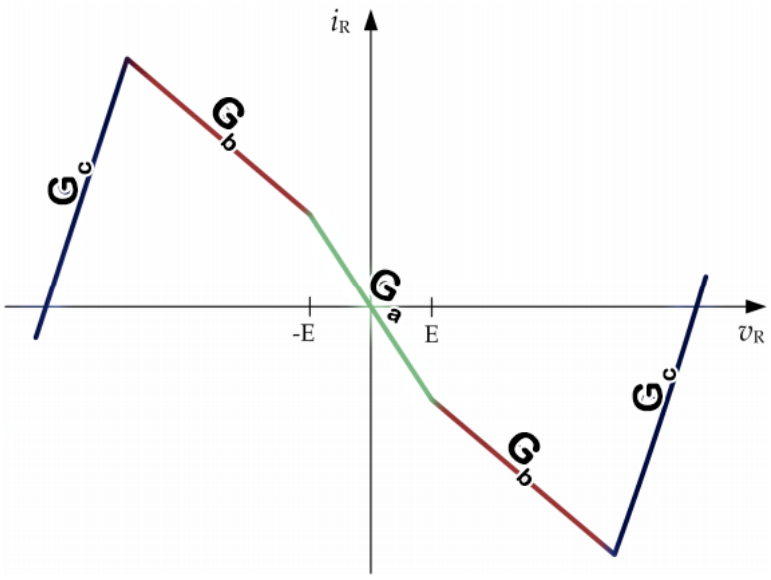


Fonte: Autoria própria, baseado em (KILIÇ, 2010)

Os amplificadores operacionais U1 e U2 em conjunto com os resistores R1 ao R6 formam um resistor negativo controlado por tensão (Negative Resistor, Figura 5), também chamado de Diodo de Chua (Chua's Diode) (CHUA, 1992) que sustenta o comportamento oscilatório.

Basicamente a curva de tensão do Diodo de Chua (vR-iR), que é o componente não-linear do sistema, consiste basicamente em duas inclinações negativas (em G_a e G_b) e duas inclinações positivas em G_c, como observado na figura Figura 6 (ŠALAMON, 2012).

Figura 6 – Curva de caracterização do Diodo de Chua



Fonte:(ŠALAMON, 2012), modificado

Caso $R_1 = R_2$ e $R_4 = R_5$, baseando-se na Figura 5, G_a e G_b podem ser dados por:

$$G_a = \frac{-1}{R_3} - \frac{-1}{R_6} \tag{3.1}$$

$$G_b = \frac{1}{R_5} - \frac{1}{R_6} \tag{3.2}$$

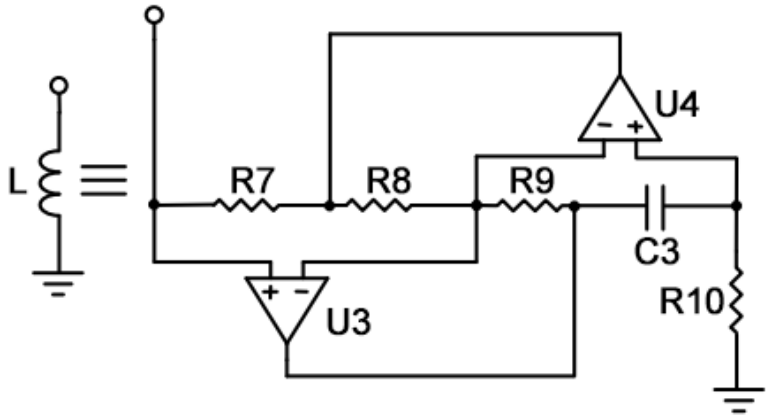
C_1 pode ser representado como uma capacitância parasita no circuito, a qual sem ela o circuito de Chua não se comportaria de maneira caótica.

O resultado após ajuste do circuito é apresentado na Figura 3, onde são observados os sinais sobre os dois terminais do resistor variável P em relação ao terra do circuito. Quando os sinais são representados em um gráfico XY, é possível ver a semelhança com a forma do Atrator de Lorenz, já rigorosamente estudado e provado seu comportamento caótico (SPARROW, 2012).

3.3 O CIRCUITO DE CHUA COM INDUTOR ELETRÔNICO

O Circuito de Chua pode ser desenvolvido sem a necessidade de um indutor real (KUMAR; SHUKLA et al., 1989), mas sim de uma associação de componentes que se comportem como um indutor. Com esta abordagem, o ressonador antes LC se transforma em uma associação RC de componentes com amplificadores operacionais, como observado na Figura 7.

Figura 7 – Circuito de indutância eletrônico



Fonte:(KILIÇ, 2010), modificado

A configuração apresentada na Figura 7 tem como denominação *gyrator* (KUMAR; SHUKLA et al., 1989), e o valor da indutância equivalente em Henry é dado por

$$L_{eq} = \frac{R7 + R9 + R10 + C3}{R8} \tag{3.3}$$

A utilização do *gyrator* tem as seguintes vantagens vantagens em relação à utilização de indutores tradicionais:

- relativo melhor controle do valor equivalente;
- menor custo, dependendo do valor necessário;
- menor volume, dependendo do valor necessário;
- melhor controle de resistência e capacitância parasitas.

Entretanto apresenta as seguintes desvantagens:

- adiciona maior complexidade ao circuito;
- maior custo dependendo do valor necessário;
- limitação em frequência devido à não-idealidade dos amplificadores operacionais.

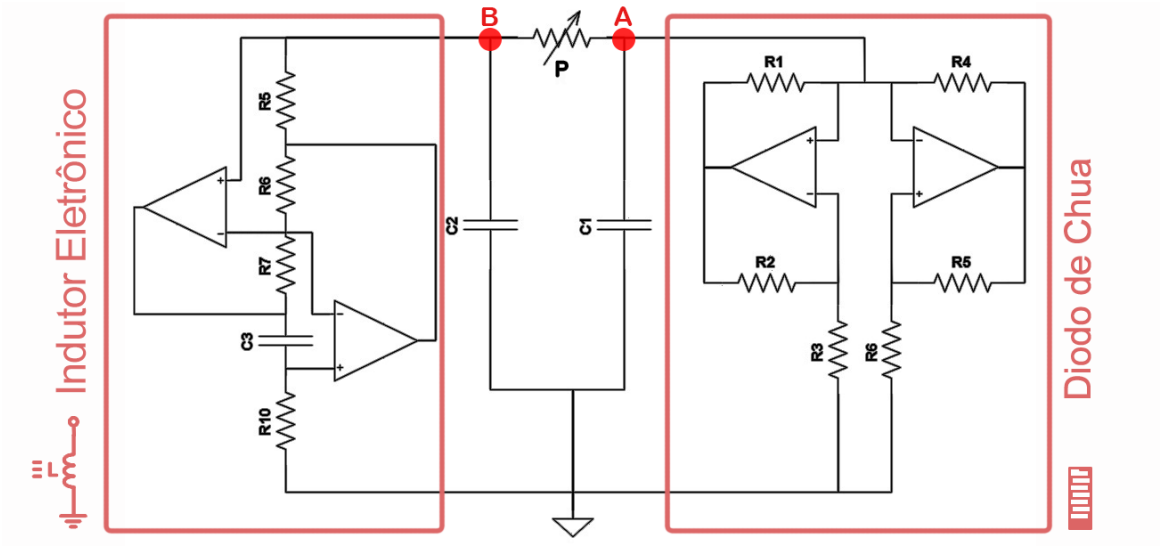
A utilização do *gyrator* também é limitada à necessidade de alimentação e aterramento, por se tratar de um indutor eletrônico à base de amplificadores operacionais (KILIÇ, 2010), porém funciona perfeitamente para o Circuito de Chua. Para a utilização de indutores com indutância flutuante, outras configurações e disposição de componentes devem ser utilizadas (KUMAR; SHUKLA et al., 1989).

4 DESENVOLVIMENTO COM O CIRCUITO DE CHUA

4.1 SIMULAÇÃO DO CIRCUITO DE CHUA

Utilizando o circuito mostrado na Figura 5 em conjunto com o circuito apresentado na Figura 7, o circuito resulta e pode ser observado na Figura 8.

Figura 8 – Circuito de Chua base para o projeto



Fonte: Autoria própria

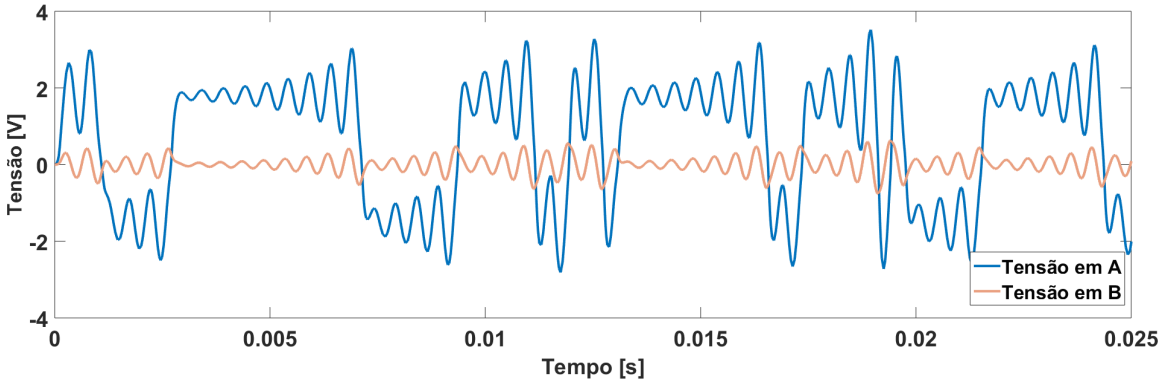
Como desenvolvido em (SIDERSKIY; KAPILA, 2014) e apresentado em (SIDERSKIY, 2019), foi desenvolvido o circuito da Figura 8 com os componentes e valores da Tabela 1 em ambiente computacional e observados os valores de tensão sobre os dois terminais da resistência variável R.

Tabela 1 – Componentes utilizados na simulação do circuito

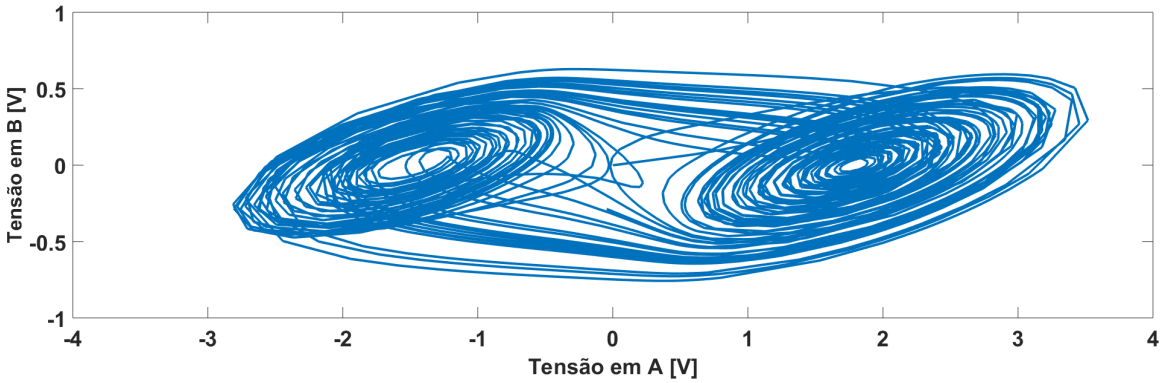
Componente	Modelo	Valor
R1	1/4 W	22 kΩ
R2	1/4 W	22 kΩ
R3	1/4 W	3.3 kΩ
R4	1/4 W	220 Ω
R5	1/4 W	220 Ω
R6	1/4 W	2.2 kΩ
R7	1/4 W	330 Ω
R8	1/4 W	270 Ω
R9	1/4 W	1 kΩ
R10	1/4 W	240 Ω
P	1/4 W	5 kΩ
C1	Poliester	10 nF
C2	Poliester	100 nF
C3	Poliester	100 nF
U1	LM741	-
U2	LM741	-
U3	LM741	-
U4	LM741	-

Os resultados da simulação medindo-se as tensões sobre os pontos A e B da Figura 8 (sobre os capacitores C1 e C2 respectivamente), podem ser avaliados na Figura 9.

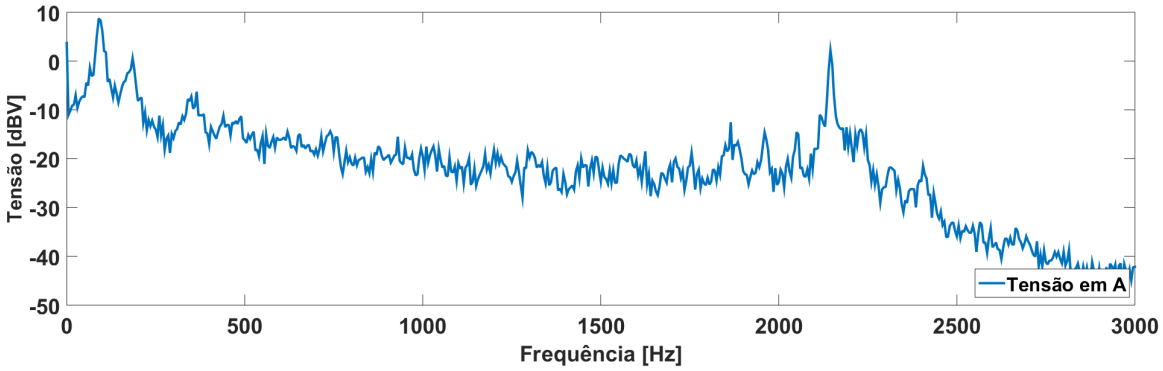
Figura 9 – Simulação de Circuito de Chua em ambiente computacional



(a) Tensão observada no domínio do tempo (simulação)



(b) Sinais A e B graficados no modo XY (simulação)



(c) Observação do sinal no domínio da frequência (FFT) (DUHAMEL; VETTERLI, 1990) (simulação)

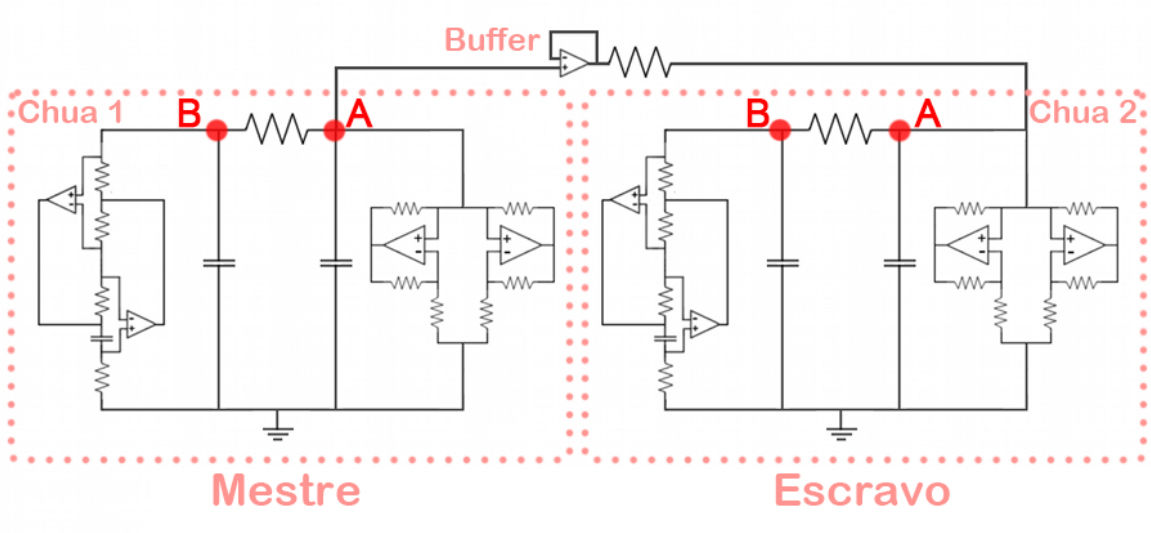
Fonte: Autoria própria

Observou-se o comportamento esperado de acordo com a literatura, portanto será levado adiante o projeto baseando-se nestas simulações.

4.2 SINCRONIZAÇÃO DE DOIS CIRCUITOS INDEPENDENTES

Nos últimos anos, variações do Oscilador de Chua foram desenvolvidas (KILIÇ, 2010), porém a sincronização entre circuitos autônomos, sendo composta por dois circuitos, é obtida com a ligação entre dois terminais ou com um resistor e amplificador operacional, no caso de uma composição de Mestre-Escravo (SIDERSKIY; KAPILA, 2014).

Figura 10 – Dois circuitos de Chua sincronizados no modo Mestre-Escravo



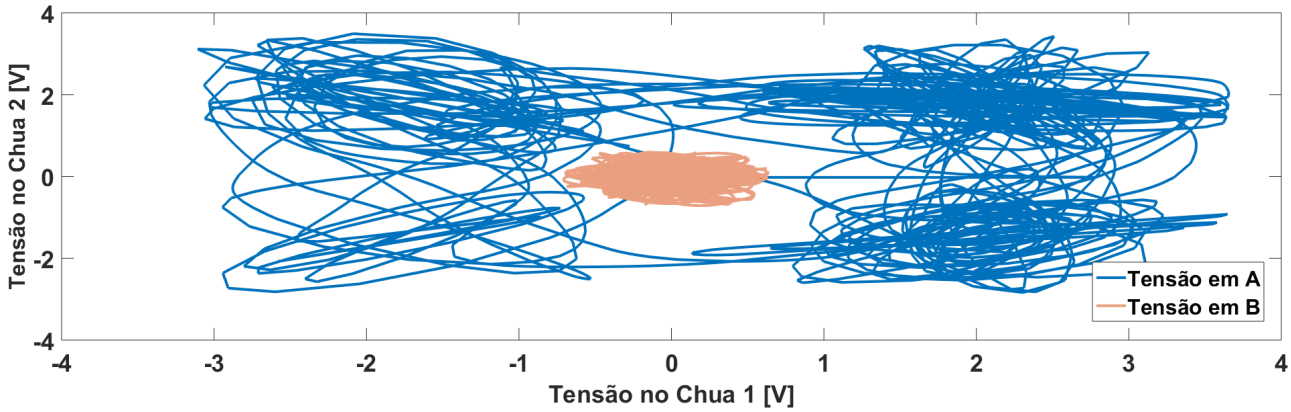
Fonte: (SIDERSKIY, 2019), modificado

No caso da Figura 10, considerando dois circuitos idênticos, a topologia utilizada é de Mestre-Escravo, pois apenas o sinal enviado pelo circuito *Chua 1* sinconizará o sinal do circuito *Chua 2*. Neste caso, o circuito *Chua 2* não interfere no funcionamento do primeiro (*Chua 1*), porém a variação de parâmetros pelo potenciômetro P no circuito do primeiro para o segundo pode realizar a sincronização.

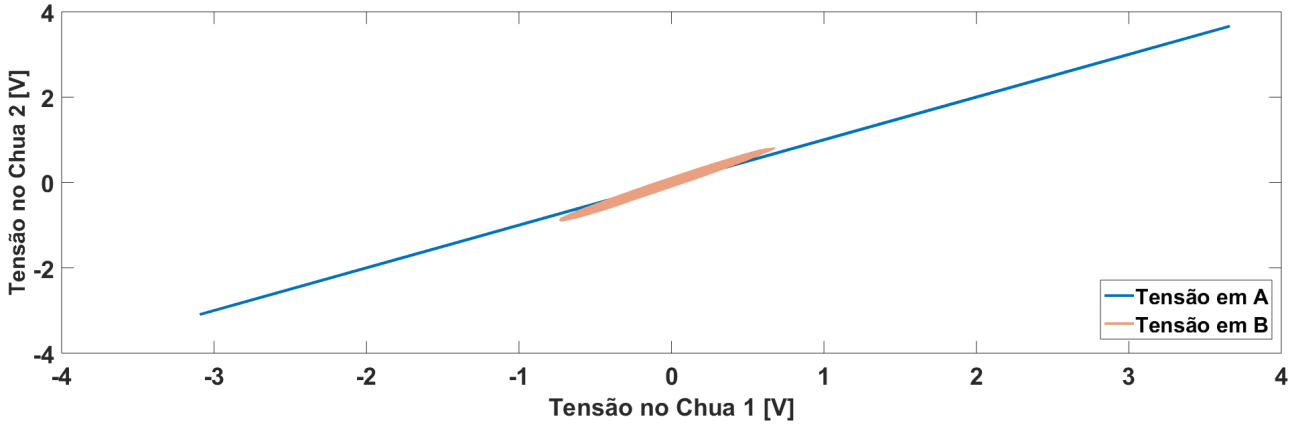
Isso ocorre devido a alta impedância no terminal de saída do amplificador operacional *Buffer*, isolando qualquer tentativa de transmissão do circuito 2 para o circuito 1. Caso ocorresse a inversão do sentido do amplificador operacional, o circuito *Chua 2* se comportaria como sincronizador e o circuito *Chua 1* como o sincronizado, até que ambos possuam o mesmo comportamento.

Uma simulação computacional utilizando o circuito da Figura 10 e os componentes descritos na Tabela 1, referente a dois circuitos não sincronizados podem ser observados na Figura 11a, e o sinal para os mesmos circuitos sincronizados são apresentados na Figura 11b.

Figura 11 – Simulação de circuito de Chua em ambiente computacional, experimento de sincronização



(a) Sinais A e B em modo XY, não sincronizados (simulação)



(b) Sinais A e B em XY, sincronizados (simulação)

Fonte: Autoria própria.

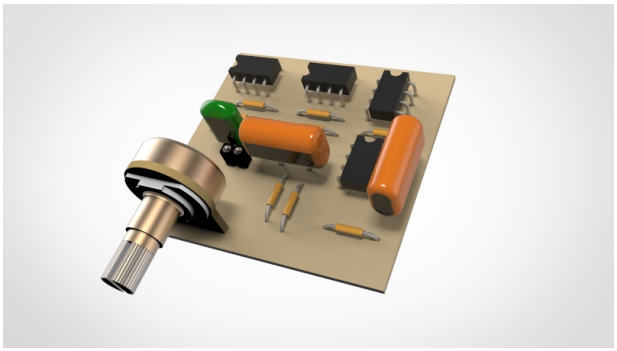
Apesar de não ser utilizado a característica de sincronização neste projeto, foi verificada esta possibilidade para que seja uma opção a ser considerada em melhorias futuras.

4.3 DESENVOLVIMENTO DO CIRCUITO EM LABORATÓRIO

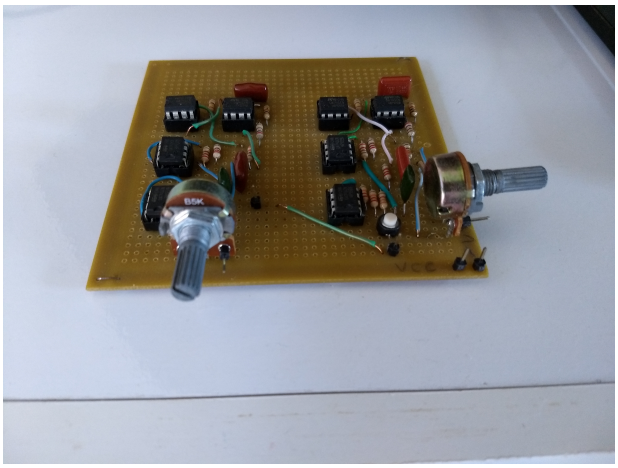
Para avaliar o comportamento descrito na literatura e simulação realizadas, o circuito da Figura 10 com os componentes da Tabela 1 foram levados a laboratório e soldados em placa padrão.

A simulação de disposição dos componentes pode ser observada na Figura 12a e o circuito real montado na Figura 12b. Foram montados dois circuitos em uma mesma placa para utilizar a mesma alimentação e referência ("terra"), uma vez que para promover a sincronização de circuitos é necessário que os mesmos estejam no mesmo plano terra.

Figura 12 – Circuito de Chua físico



(a) Simulação de circuito a ser montado



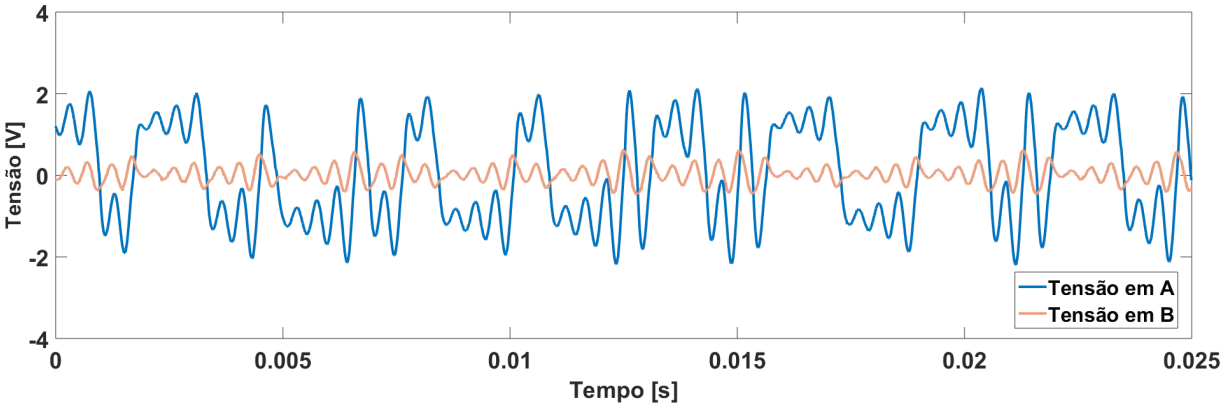
(b) Circuito soldado em placa padrão

Fonte: Autoria própria

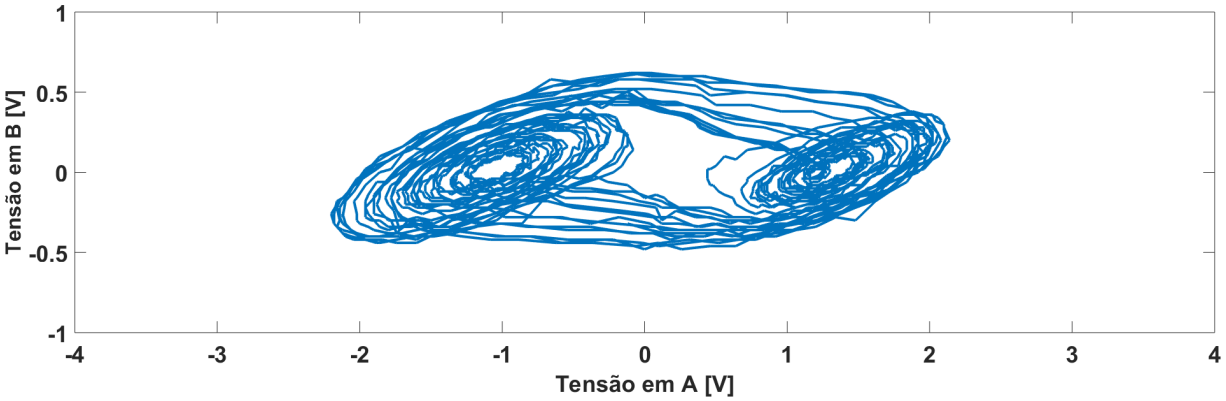
4.4 AVALIAÇÃO DE RESULTADOS

Alimentando o circuito devidamente com tensões de $-5V$ e $5V$ para os respectivos terminais de acordo com o proposto, as medições práticas do circuito podem ser observadas na Figura 13.

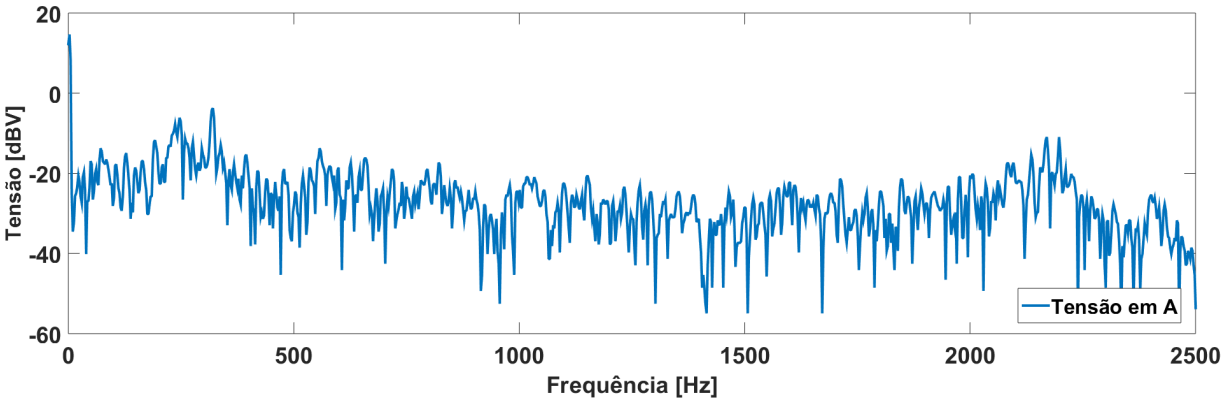
Figura 13 – Sinais observados no circuito físico montado em laboratório



(a) Sinais A e B ao longo do tempo (laboratório)



(b) Sinais A e B em XY (laboratório)



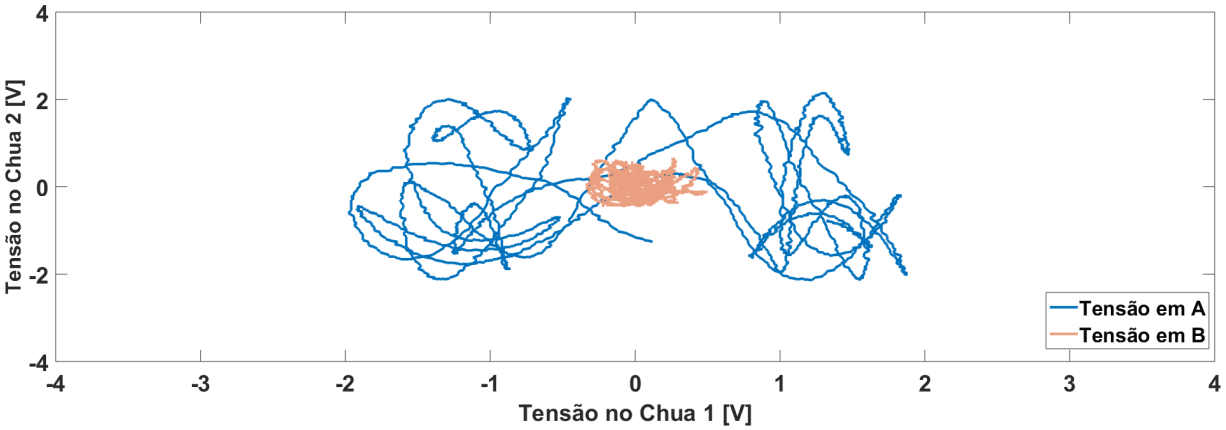
(c) Sinais A e B em frequência (laboratório)

Fonte: Autoria própria

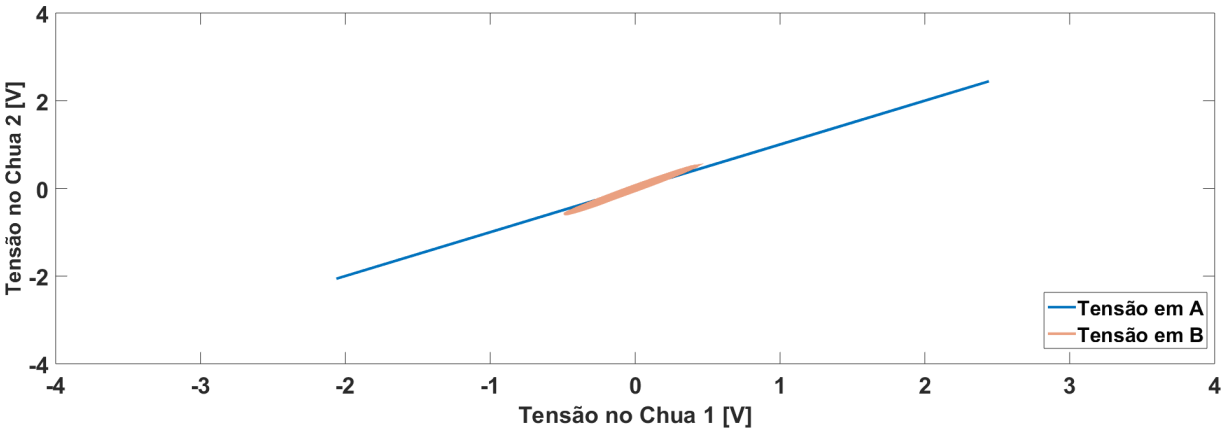
Como observado, apesar da medição não ser igual à simulação devido a condições iniciais, o circuito mostrou comportamento estável e a forma de onda gerada corresponde com o comportamento caótico esperado em teoria, quando se sobrepõe os dois sinais em modo XY com o potenciômetro em valores entre 2.9kΩe 3.1kΩ(Figura 8).

Ligando-se os pontos A dos dois circuitos desenvolvidos (Figura 10), a fim de obter a sincronização de sinais, obteve-se o resultado apresentado na Figura 14.

Figura 14 – Sinais observados no circuito físico montado em laboratório sem e com sincronização



(a) Sinais A e B em modo XY, sem sincronização (laboratório)



(b) Sinais A e B em modo XY, com sincronização (laboratório)

Fonte: Autoria própria.

A sincronização de circuitos também provou-se possível, e, apesar de não ser utilizada neste projeto como parte do trabalho de codificação, é um forte ponto extra em melhorias futuras utilizando-se disto para incrementar o nível de segurança.

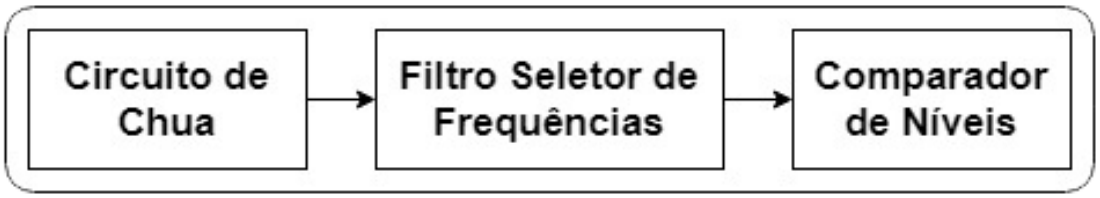
5 UTILIZAÇÃO DO CIRCUITO DE CHUA COMO UM GERADOR ALEATÓRIO DE BITS

Como a utilização do circuito de Chua provou-se viável na prática, seu comportamento será utilizado a favor da cifragem de informações antes de serem transmitidas. Para isso, antes será necessário transformar o sinal gerado, que é analógico, em informação digital, ou seja, utilizá-lo como um gerador de bits aleatórios (TRNG - do inglês True Random Number Generator) (JITEURTRAGOOL; WANNABOON; MASAYOSHI, 2015).

5.1 DESENVOLVIMENTO TEÓRICO E SIMULAÇÃO

Para atingir o objetivo de codificar uma mensagem antes de ser transmitida valendo-se do comportamento do sinal gerado pelo circuito de Chua, propõe-se filtrar as componentes de frequência mais elevada e discretizar o sinal resultante com um comparador de tensões, de acordo com a Figura 15.

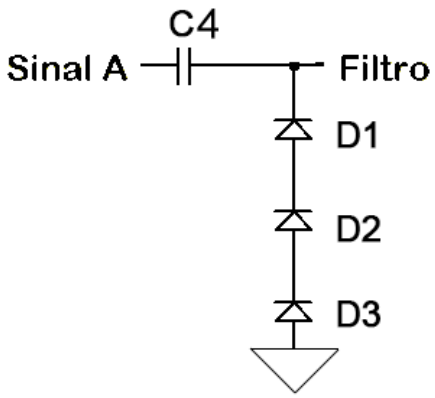
Figura 15 – Esquema para se obter o sinal gerado em níveis discretos



Fonte: Autoria própria

Devido ao fato de que o sinal proveniente do Circuito de Chua possui componentes positivas e negativas de tensão, e também devido às características da entrada do comparador de níveis é necessário elevar para que apenas componentes positivas entrem. Para isso foi utilizado um circuito composto por diodos em série associados a um capacitor para o nível de corrente contínua ser mantido apenas em tensões positivas, mostrado na Figura 16.

Figura 16 – Elevador de tensão



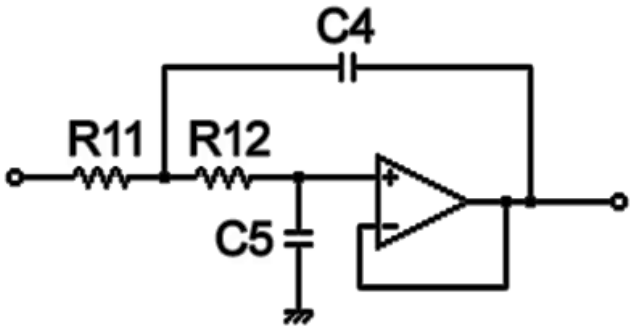
Fonte: Autoria própria

Os diodos $D1, D2$ e $D3$ são do modelo 1N4148 e o capacitor $C4$ de $1nF$.

Como observado na Figura 13c, as componentes com amplitudes mais elevadas encontram-se abaixo de 500Hz, portanto frequências acima disso serão rejeitadas pelo filtro passa-baixa para que seja possível criar um sinal com níveis discretos de tensão.

Para o desenvolvimento do filtro, foi utilizado o circuito mostrado da Figura 17, escolhido levando em consideração os recursos disponíveis e praticidade de implementação. A frequência de corte para se calcular o valor dos componentes é descrita na Equação 5.1(DESIGN, 2019).

Figura 17 – Filtro passa-baixa Sallen Key de primeira ordem (DESIGN, 2019)



Fonte: (DESIGN, 2019), modificado

$$f_c = \frac{1}{2\pi\sqrt{R11 * R12 * C4 * C5}} \tag{5.1}$$

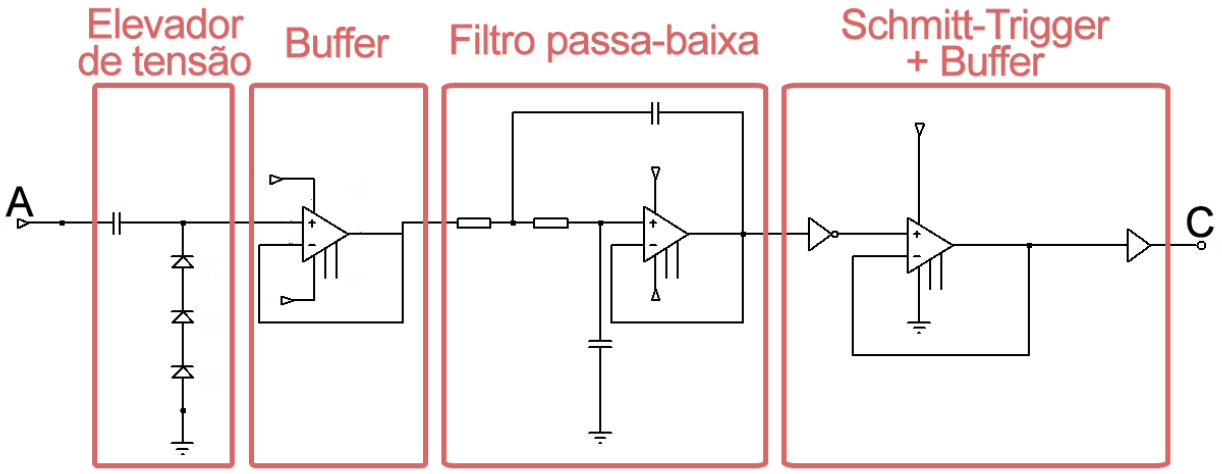
Os valores mais próximos a se obter com valores comerciais de componentes foram de $R11 = R12 = 33k\Omega$ e $C4 = C5 = 0.01uF$. A frequência de corte então será de 482.28Hz.

Com isso, o sinal resultando será passado por um circuito Schmitt-Trigger (74LS14), o qual possibilita gerar uma saída com níveis discretos dependendo do sinal de entrada: quando o nível de tensão de entrada é maior que um limiar escolhido, a saída está em nível alto; quando a entrada está abaixo de outro limiar, a saída está em nível baixo(Wikipedia contributors, 2019).

Afim de isolar os estágios de processamento do sinal, foram adicionados circuitos acopladores de tensão (Wikipedia contributors, 2017) após cada estágio, para que a interligação com o estágio seguinte não influenciasse no comportamento do estágio anterior, que foram construídos com uma ligação em um amplificador operacional LM741.

O circuito digitalizador pode ser observado em Figura 18.

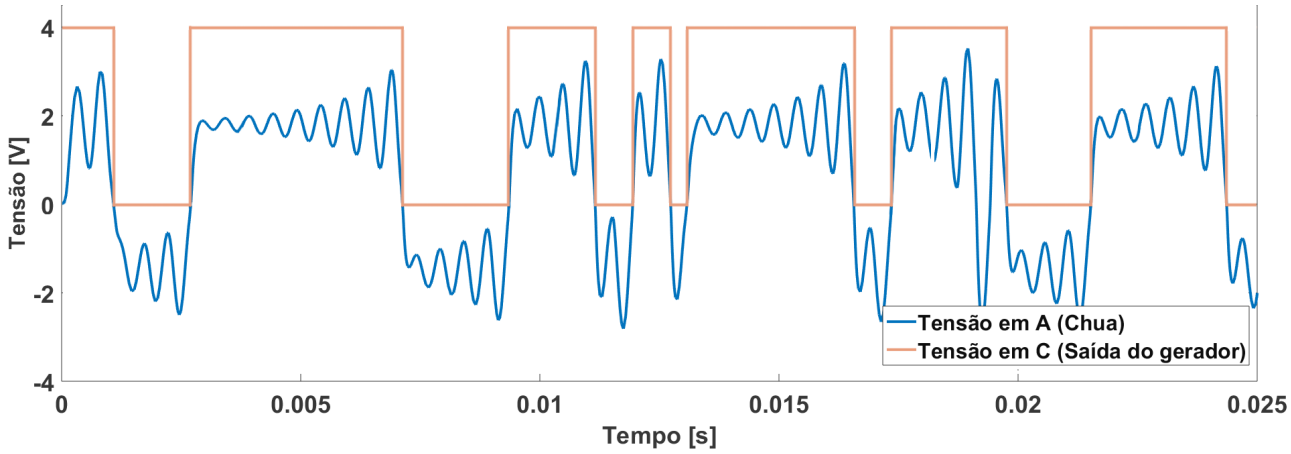
Figura 18 – Circuito final para digitalização do sinal proposto



Fonte: Autoria própria

Simulado computacionalmente, o resultado da saída do circuito proposto em Figura 18 pode ser visto na Figura 19.

Figura 19 – Simulação de sinal gerado por circuito discretizador de níveis (simulação)



Fonte: Autoria própria

5.2 AVALIAÇÃO DO SINAL DIGITAL GERADO

A fim de garantir a aleatoriedade dos bits gerados pelo circuito projetado, será necessário abordar algumas técnicas para garantir que os bits não possuem um padrão de geração.

Serão usadas algumas técnicas descritas em (BASSHAM et al., 2010).

5.2.1 Teste de frequência monobit

A função teste de frequência monobit é a da proporção entre valores 1 e 0 em uma sequência binária. Em uma sequência aleatória, o esperado é que a proporção entre os valores seja o mais próximo de 1:1. Todos os testes subsequentes dependem de passar neste teste.

A sequência é dada como aleatória caso $P_{value} \geq 0.01$,

$$S_{obs} = \frac{|S_n|}{\sqrt{n}} \tag{5.2}$$

e

$$P_{value} = \text{erfc}\left(\frac{S_{obs}}{\sqrt{2}}\right) \tag{5.3}$$

sendo S_n a soma de todos os bits 1s e 0s considerando o valor dos bits 1 = 1 e dos bits 0 = -1.

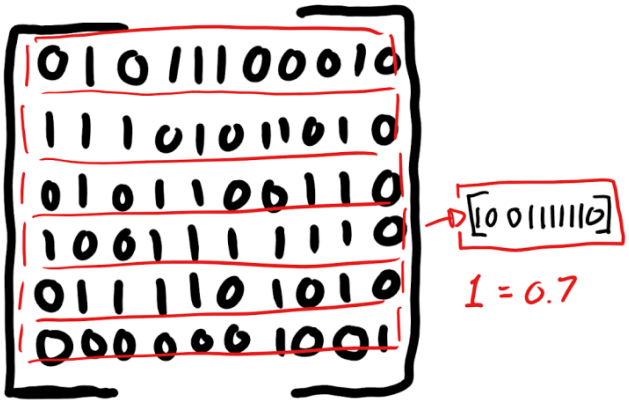
5.2.2 Teste de frequência em blocos

O propósito deste teste é parecido com o primeiro, entretanto avaliar a frequência de 1s e 0s em certos blocos de bits.

O teste é feito particionando a quantidade n de bits em N blocos de M bits cada. Caso a divisão não seja exata, os bits restantes são descartados do teste. Para $M = 1$, o teste torna-se um teste de frequência monobit.

Em um gerador aleatório alguns blocos não serão dados como aleatórios e se espera que alguns reprovem no quesito aleatoriedade, entretanto na avaliação geral deve-se observar um valor muito próximo de bits 1s e 0s, como observado na Figura 20.

Figura 20 – Exemplo de avaliação do teste de bloco para $M = 10$, com o bloco destacado contendo 7 1s



Fonte: (ROGERS, 2019), modificado

Considerando uma quantidade n de bits em N blocos de M bits cada é feita a proporção de 1s no bloco de acordo com

$$\pi_i = \frac{\sum_{j=1}^{M} \epsilon(i-1)M + j}{M} \tag{5.4}$$

Sendo ϵ a sequência de bits avaliada.

Em seguida é calculada a estatística χ^2 de acordo com a equação 5.5.

$$\chi^2(obs) = 4M \sum_{i=1}^{N} (\pi_i - \frac{1}{2})^2 \tag{5.5}$$

O teste é dado como satisfatório quando $P_{value} \geq 0.01$, dado pela equação 5.6.

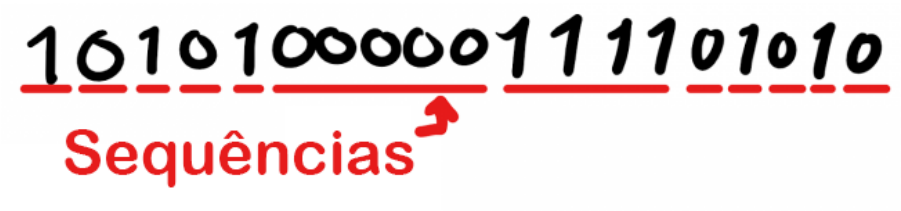
$$P_{value} = \Gamma(\frac{N}{2}, \frac{\chi^2(obs)}{2}) \tag{5.6}$$

Sendo Γ a função gama incompleta superior(BASSHAM et al., 2010).

5.2.3 Teste de sequência

O teste de sequência foca em determinar uma sequência ininterrupta de 0s ou 1s, ou seja, bits iguais. Uma sequência de comprimento k consiste de exatamente k bits idênticos e é contada a partir de um bit contrário e termina antes de um bit contrário. O propósito deste teste é verificar se a frequência de oscilação está muito lenta ou muito rápida de acordo com a frequência de leitura desses bits.

Figura 21 – Exemplo de avaliação do teste de sequência, destacando a separação de sequências em uma sequência aleatória de bits



Fonte: (ROGERS, 2019), modificado

Inicialmente é computada a proporção de bits 1s em uma amostra de n bits, de acordo com a equação 5.7.

$$\pi = \frac{\sum_j \epsilon_j}{n} \tag{5.7}$$

Então é realizado o teste estatístico V_n, de acordo com a equação 5.8

$$V_n(obs) = \sum_{k=1}^{n-1} r(k) + 1 \begin{cases} k = 0 & \epsilon_k = \epsilon_{k+1} \\ k = 1 & \epsilon_k \neq \epsilon_{k+1} \end{cases} \tag{5.8}$$

Sendo ϵ a sequência de bits. Novamente o teste é dado como satisfatório para uma sequência aleatória caso $P_{value} \geq 0.01$, dado pela equação 5.9

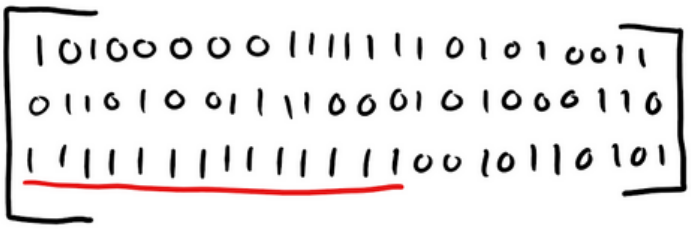
$$P_{value} = \text{erfc}\left(\frac{|V_n(obs) - 2n\pi(1-\pi)|}{2\sqrt{2n\pi}(1-\pi)}\right) \tag{5.9}$$

5.2.4 Teste de maior sequência

O propósito deste teste é determinar se o comprimento da maior sequência de 1s dentro de uma sequência é consistente com o esperado com a maior sequência de 1s que seria esperado em

uma sequência aleatória. O teste é feito apenas para valor de 1 pelo motivo de que se houver uma irregularidade na maior sequência de 1s, consequentemente haverá irregularidades na sequência de 0s também.

Figura 22 – Exemplo de avaliação do teste de maior sequência



Fonte: (ROGERS, 2019), modificado

É necessário dividir a amostra total de n bits em blocos de M bits, respeitando a quantidade mínima de bits n para a avaliação de bloco de comprimento M de acordo com a Tabela 2.

Tabela 2 – Valor de n mínimo para blocos de M bits

n **mínimo**	M
128	8
6272	128
750000	10^4

Então é computada a quantidade de bits 1s que aparece em cada bloco de acordo com a Tabela 3.

Tabela 3 – Valores a avaliar caso haja uma sequência correspondente

v_i	$M = 8$	$M = 128$	$M = 10^4$
v_0	≤ 1	4	≤ 10
v_1	2	5	11
v_2	3	6	12
v_3	≥ 4	7	13
v_4		8	14
v_5		≥ 9	15
v_6			≥ 16

A seguir é calculado o $\chi(obs)^2$ de acordo com a equação 5.10.

$$\chi^2(obs) = \sum_{i=0}^{K} \frac{(v_i - N\pi_i)^2}{N\pi_i} \tag{5.10}$$

Nos quais K e π_i são tabelados de acordo com as tabelas apresentadas na Figura 23.

Figura 23 – Valor de π_i para a equação 5.10

K=3, M=8

classes	probabilities
{ν≤1}	$\pi_0 = 0.2148$
{ν=2}	$\pi_1 = 0.3672$
{ν=3}	$\pi_2 = 0.2305$
{ν≥4}	$\pi_3 = 0.1875$

K=5, M=128

classes	probabilities
{ν≤4}	$\pi_0 = 0.1174$
{ν=5}	$\pi_1 = 0.2430$
{ν=6}	$\pi_2 = 0.2493$
{ν=7}	$\pi_3 = 0.1752$
{ν=8}	$\pi_4 = 0.1027$
{ν≥9}	$\pi_5 = 0.1124$

K=6, M=10000

classes	probabilities
{ν≤10}	$\pi_0 = 0.0882$
{ν=11}	$\pi_1 = 0.2092$
{ν=12}	$\pi_2 = 0.2483$
{ν=13}	$\pi_3 = 0.1933$
{ν=14}	$\pi_4 = 0.1208$
{ν=15}	$\pi_5 = 0.0675$
{ν≥16}	$\pi_6 = 0.0727$

Fonte: (BASSHAM et al., 2010)

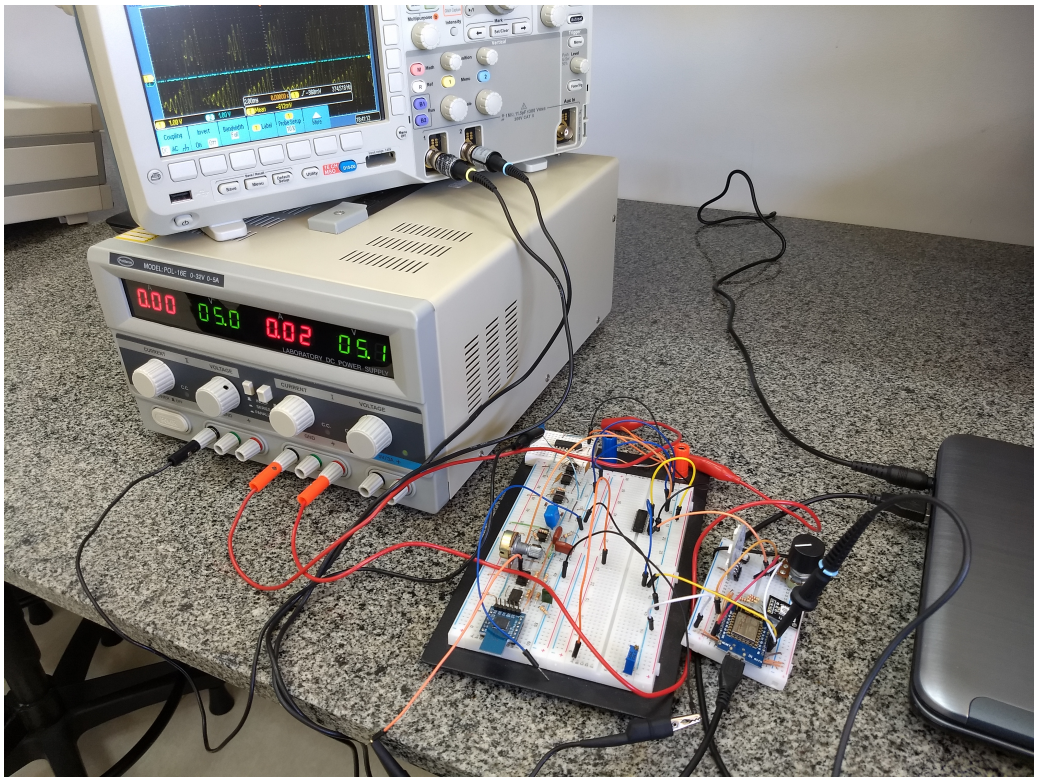
Finalmente P_{value} é dado pela equação 5.11. Se $P_{value} \geq 0.01$, o teste é dado satisfatório para uma sequência aleatória.

$$P_{value} = \Gamma\left(\frac{K}{2}, \frac{\chi^2}{2}\right) \tag{5.11}$$

5.3 DESENVOLVIMENTO EM LABORATÓRIO

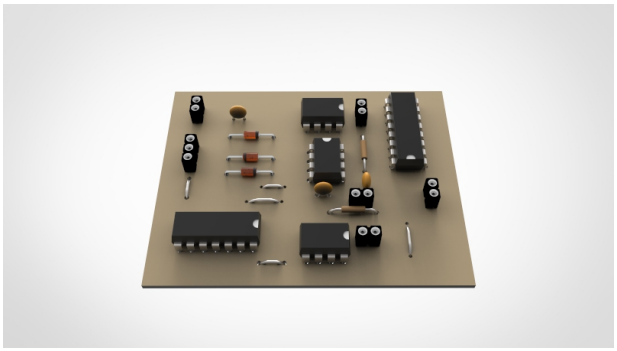
A fim de validar os dados simulados e apresentados na Figura 19, o circuito foi inicialmente montado e testado em *protoboard* (Figura 24) e posteriormente, após validado, soldado em uma placa de circuito impresso (PCI), como apresentado na Figura 25.

Figura 24 – Circuito digitalizador e oscilador de Chua em protoboard para avaliação

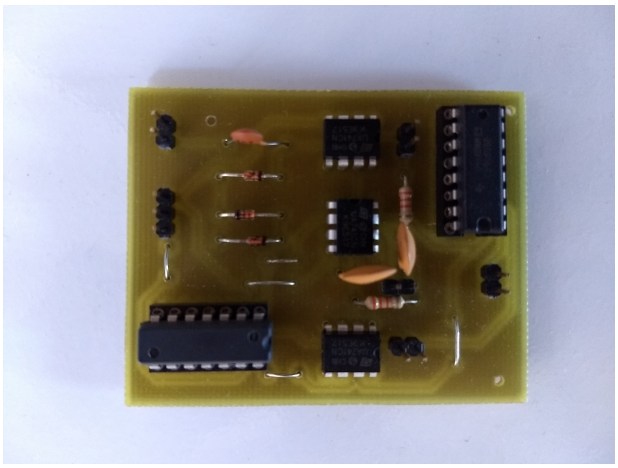


Fonte: Autoria própria

Figura 25 – Circuito de Chua físico



(a) Simulação de circuito a ser montado

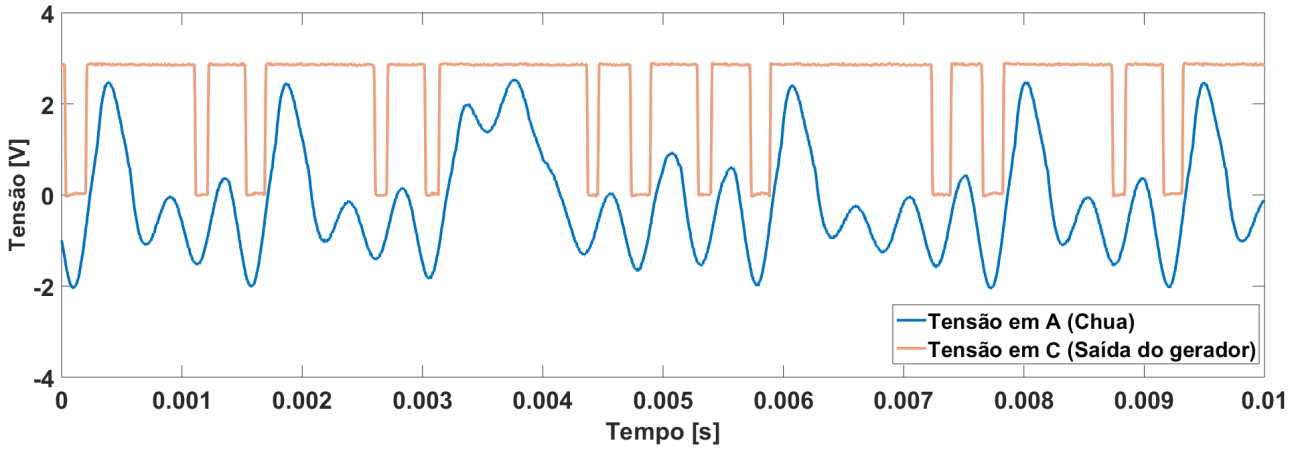


(b) Circuito soldado em placa fenolite-cobre

Fonte: Autoria própria

Os sinais foram adquiridos em um osciloscópio modelo Tektronix TBS-1072B para avaliação, demonstrado na figura Figura 26.

Figura 26 – Sinal gerado por circuito discretizador de níveis (laboratório)



Fonte: Autoria própria

5.4 AVALIAÇÃO DE RESULTADOS

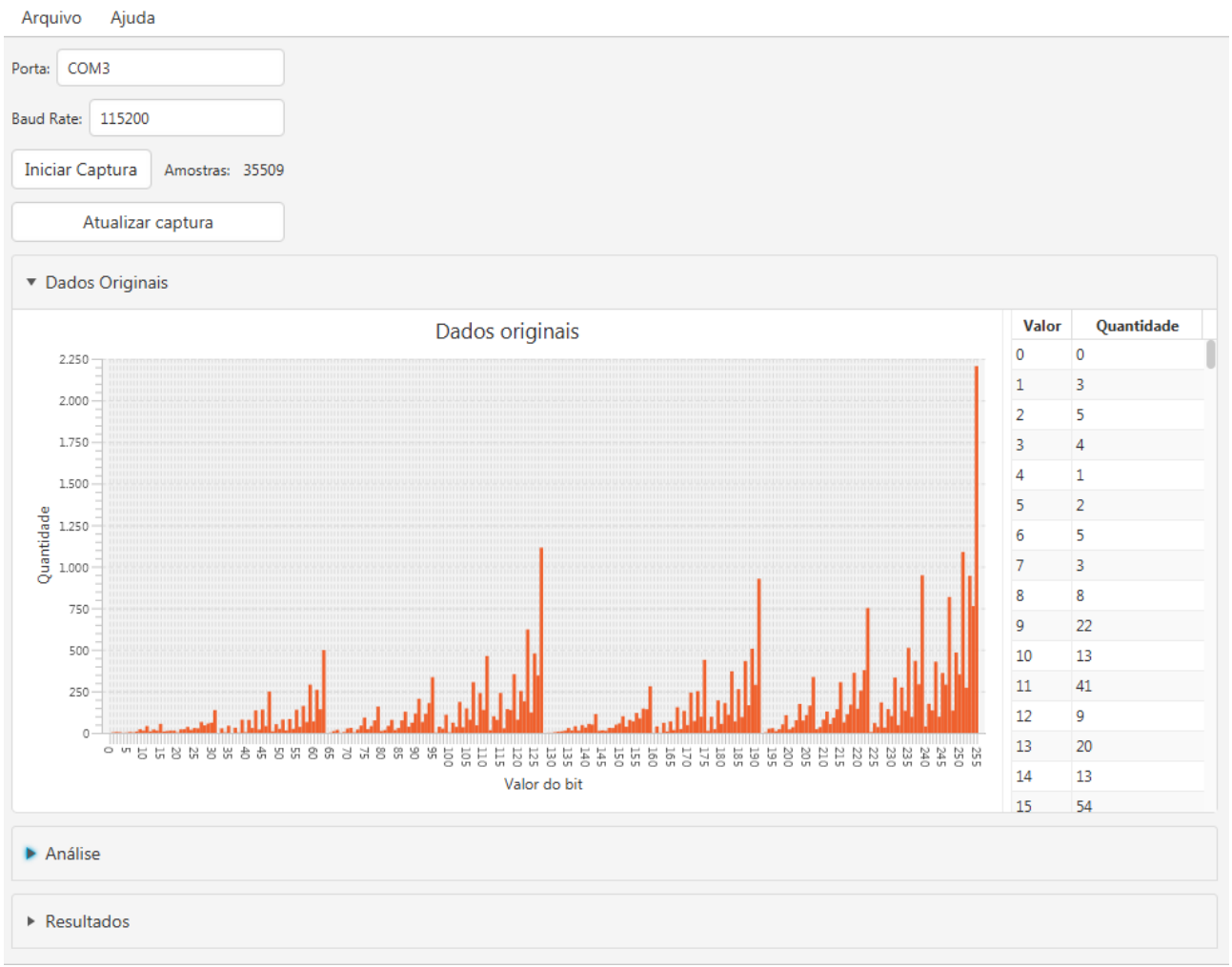
Para avaliar a aleatoriedade dos bits gerados pelo circuito da Figura 25b, foi necessário obtê-los através de um microcontrolador Arduino Uno, que em blocos de 8 bits (1 byte), apresentava o valor recebido em uma taxa de leitura de 100 Baud do sinal proveniente do circuito anterior.

Na transmissão do microcontrolador para o computador, foi utilizada uma taxa de escrita de 115200 Baud.

Foi desenvolvido um software em linguagem Java (Apêndice A.1) utilizando a biblioteca matemática Apache Commons Math3 (APACHE, 2019) para realizar os cálculos matemáticos das avaliações descritas na seção 5.2, e capturando via porta serial do computador os dados recebidos pelo microcon-

trolador, os bytes recebidos são então transformados em uma única sequência de 0s e 1s para serem avaliados (Figura 27).

Figura 27 – Interface de software desenvolvido para avaliar aleatoriedade de bits



Fonte: Autoria própria

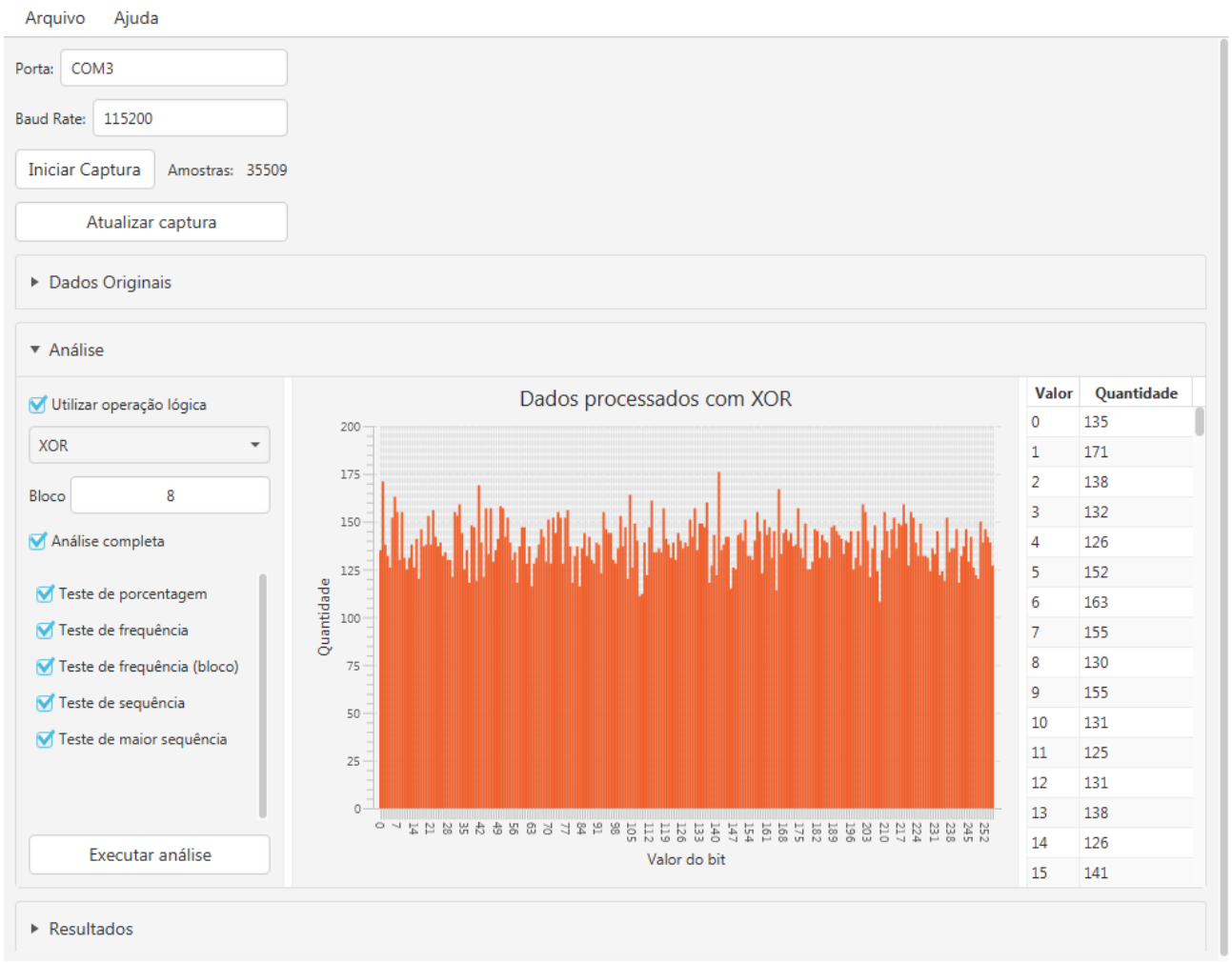
Foram obtidos 35509 bytes de amostra gerados pelo sistema para análise, correspondente a 284072 bits, ou 277kb.

Para complementar a avaliação, foi também adicionado um opcional pós processamento de bits, onde é possível realizar operações lógicas "OU", "E", e "OU EXCLUSIVO"("XOR"), como observado na Figura 28.

A utilização das operações lógicas "OU"e "E"são feitas comparando um byte gerado com seu anterior, entretanto não cumulativo pois tanto a operação "OU"quanto a "E"tendem a produzir bytes de valor "11111111"a longo prazo quando são acumulados valores não-aleatórios.

O mesmo não acontece com a operação "XOR". Esta não tende a gerar bytes "11111111"pois como uma das entradas é originada do gerador de bits, valores "1"podem ser sobrescritos como "0".

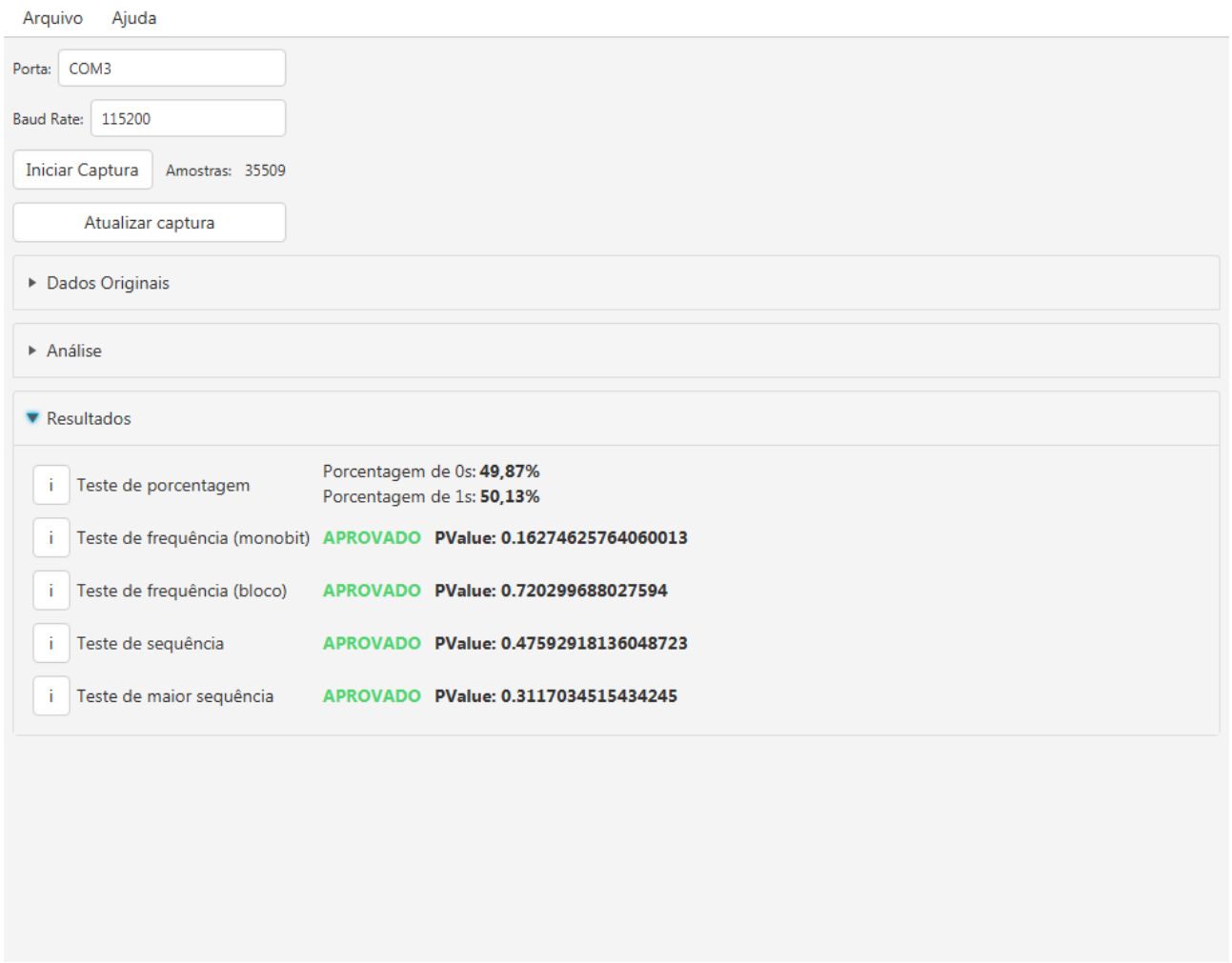
Figura 28 – Opções de pré-processamento antes de análise



Fonte: Autoria própria

Após definir os parâmetros de análise (testes a serem realizados, tamanho de bloco e se utilizará operação lógica) o resultado é exibido, como observado em Figura 29.

Figura 29 – Resultado de testes de aleatoriedade de bits



Fonte: Autoria própria

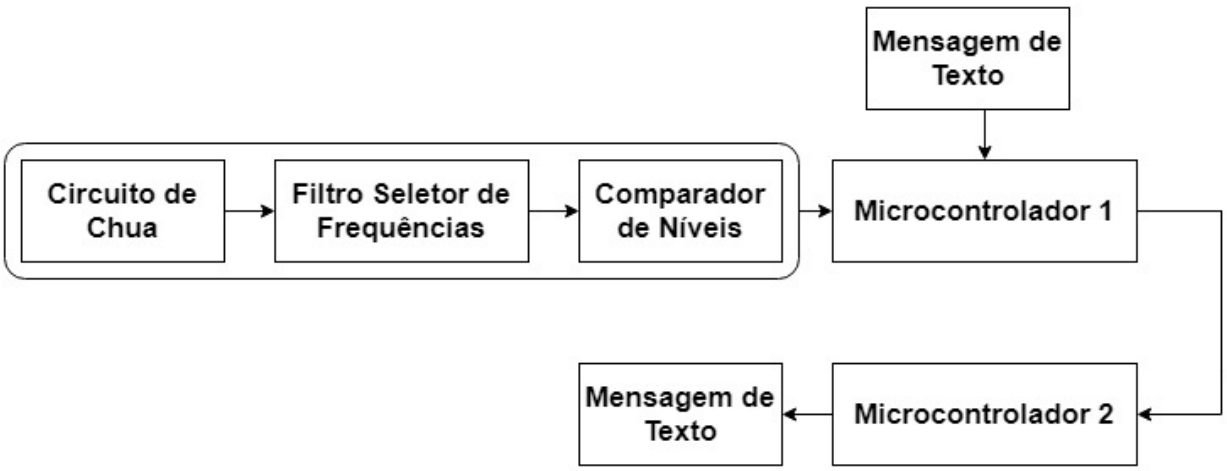
Como descrito na seção 5.2, um $P_{value} \geq 0.01$ garante aprovação no teste de aleatoriedade, assim então é retornado ao programa o resultado dos cálculos realizados em ambiente computacional.

Como avaliado pelo algorítimo, é necessário uma operação de "OU EXCLUSIVO"para que o gerador de bits seja realmente aleatório. Neste projeto isto será compensado no microcontrolador de transmissão, porém estima-se que seja possível eliminar este procedimento com a adição circuitos integrado que realizem esta operação.

6 UTILIZAÇÃO DO GERADOR DE BITS ALEATÓRIOS PARA TRANSMISSÃO DE INFORMAÇÃO CODIFICADA

Já verificado a não previsibilidade do gerador de bits aleatórios, este será utilizado de forma a codificar uma mensagem antes de ser transmitida. Para isso, o modelo de sistema que será utilizado pode ser visto na Figura 30

Figura 30 – Proposta de sistema para codificação de mensagem de texto



Fonte: Autoria própria

6.1 DESENVOLVIMENTO TEÓRICO

Para a codificação da mensagem com os bits aleatórios, esta deve ser feita de modo que seja possível recuperá-la no receptor por um processo inverso.

Para isso, será aplicado um algoritmo relacionando os bits gerados pelo circuito digitalizador e a mensagem a ser transmitida. O processo será guiado da seguinte forma:

1. É introduzida no microcontrolador uma mensagem de entrada de comprimento n.
2. É gerada uma sequência de bytes aleatórios lida do gerador de comprimento n.
3. É realizada uma operação lógica "OU EXCLUSIVO"(XOR) entre cada elemento da mensagem original e o correspondente na sequência aleatória, o resultado é alocado como uma terceira sequência.
4. A mensagem a ser transmitida será o resultado de um embaralhamento de bytes da mensagem codificada e da sequência aleatória de bytes seguindo o padrão da sequência de Fibonacci, por escolha do autor.
5. A mensagem recebida é então decodificada reordenando os bytes aleatórios e os bytes codificados e reaplicando a operação lógica XOR.

A transformação de caractere em valor numérico é dado pela conversão do caractere com seu valor decimal na tabela ASCII de caracteres, então convertida para binário e aplicada a operação.

Neste trabalho será utilizado e considerado o mapa de caracteres *Code page 437 (CP437)*, nos quais os 32 primeiros caracteres não são digitáveis, sendo reservados para operações do sistema como apagar, quebra de linha, espaçamento, escape, entre outros (Figura 31), porém como as operações serão executadas com valores numéricos em bytes, a utilização de caracteres fica limitada a representação visual da criptografia.

Vale apontar que um caractere possui 8 bits, portanto um byte é equivalente a um caractere.

Figura 31 – Mapa de caracteres CP437 em hexadecimal

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	`	p	Ç	É	á	░	└	╨	α	≡
1	SOH	DC1	!	1	A	Q	a	q	ü	æ	í	▒	┴	╤	ß	±
2	STX	DC2	"	2	B	R	b	r	é	Æ	ó	▓	┬	╥	Γ	≥
3	ETX	DC3	#	3	C	S	c	s	â	ô	ú	│	├	╙	π	≤
4	EOT	DC4	$	4	D	T	d	t	ä	ö	ñ	┤	─	╘	Σ	⌠
5	ENQ	NAK	%	5	E	U	e	u	à	ò	Ñ	╡	┼	╒	σ	⌡
6	ACK	SYN	&	6	F	V	f	v	å	û	ª	╢	╞	╓	µ	÷
7	BEL	ETB	'	7	G	W	g	w	ç	ù	º	╖	╟	╫	τ	≈
8	BS	CAN	(	8	H	X	h	x	ê	ÿ	¿	╕	╚	╪	Φ	°
9	HT	EM	)	9	I	Y	i	y	ë	Ö	⌐	╣	╔	┘	Θ	·
A	LF	SUB	*	:	J	Z	j	z	è	Ü	¬	║	╩	┌	Ω	·
B	VT	ESC	+	;	K	[	k	{	ï	¢	½	╗	╦	█	δ	√
C	FF	FS	,	<	L	\	l	|	î	£	¼	╝	╠	▄	∞	n
D	CR	GS	-	=	M	]	m	}	ì	¥	¡	╜	═	▌	φ	²
E	SO	RS	.	>	N	^	n	~	Ä	₧	«	╛	╬	▐	ε	■
F	SI	US	/	?	O	_	o	DEL	Å	ƒ	»	┐	╧	▀	∩	

Fonte: http://www.asahi-net.or.jp/ yw3t-trns/code/cp437.html

A demonstração de uma codificação do caractere "A"com operação lógica XOR pode ser observado na Figura 32.

Figura 32 – Exemplo de codificação de caractere

	Byte Original	Byte Aleatório	Byte Codificado
Char	a	5	T
Decimal ASCII	97	53	84
Binário	01100001	00110101	01010100

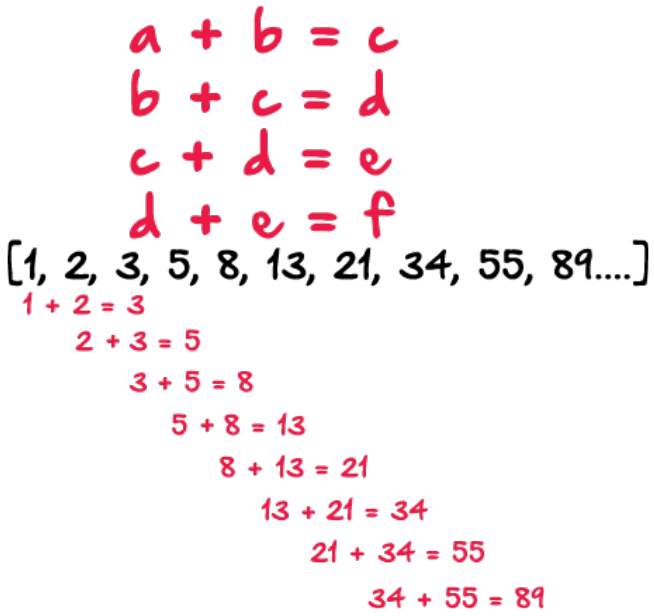
Fonte: Autoria própria

Para receber e decodificar a mensagem, é necessário transmitir o caractere aleatório e o caractere codificado, porém apenas enviando as duas sequências em seguida é uma forma pouco segura de se

transmitir a mensagem uma vez que por uma simples combinação a mensagem original é recuperada.

A fim de dificultar a recuperação por meios indesejados, as sequência de caracteres aleatórios e caracteres codificados serão embaralhados seguindo um algoritmo matemático conhecido como sequência de Fibonacci, como exemplificado na Figura 33.

Figura 33 – Exemplo de sequência de Fibonacci



Fonte: (JARREL, 2019), modificado

De acordo com a sequência, a mensagem final será composta por $2n$ caracteres, sendo eles dispostos da seguinte forma:

- 1 caractere codificado e 1 caractere originário do gerador aleatório;
- 2 caracteres codificados e 2 caracteres originários do gerador aleatório;
- 3 caracteres codificados e 3 caracteres originários do gerador aleatório;
- 5 caracteres codificados e 5 caracteres originários do gerador aleatório;
- 8 caracteres codificados e 8 caracteres originários do gerador aleatório;
- 13 caracteres codificados e 13 caracteres originários do gerador aleatório;
- ...
- x caracteres codificados e x caracteres originários do gerador aleatório caso os últimos caracteres a serem transmitidos não sejam um número exato da sequência de Fibonacci.

Para uma mensagem de 7 caracteres, a proposta de mensagem a ser transmitida pode ser observada na Figura 34.

Figura 34 – Codificação da mensagem "UNESP13"com 7 bytes pelo sistema proposto em teoria

	Sequência em caracteres
Sequência de bytes originais	UNESP13
Sequência de bytes aleatórios	2╣ 3±`ô{
Sequência de bytes codificados	g≈vó0óH
Sequência a ser transmitida	g2≈v╣ 3ó0ó±`ôH{

Fonte: Autoria própria

6.2 DESENVOLVIMENTO EM LABORATÓRIO

Para comprovar a viabilidade de comunicação com o padrão de codificação proposto, foi desenvolvido um código para o microcontrolador Arduino UNO (Apêndice A.2) para ler uma mensagem, gerar uma sequência aleatória baseada no gerador de bits aleatórios para codificar a mensagem original e então embaralhar os bytes e transmiti-los a outro microcontrolador Arduino UNO que receberá o que foi enviado e decodificará a mensagem original (Apêndice A.3).

O algorítimo de codificação foi desenvolvida em linguagem de programação Arduino, padrão do microcontrolador, e a comunicação entre dispositivos realizada por meio da biblioteca padrão *Software Serial* fornecida na instalação da interface de desenvolvimento Arduino.

O resultado da transmissão para a mensagem "UNESP13"pode ser observada na Figura 35a. O recebimento da transmissão realizada pode ser observada na Figura 35b.

Figura 35 – Codificação e decodificação da mensagem "UNESP13"

```
Serial iniciada

#INICIANDO CODIFICAÇÃO

Tamanho da mensagem: 7
Mensagem: UNESP13
Chave: 2§3§`§{
Array chave: 50, 185, 51, 241, 96, 147, 123,
Iterações: 3
Resto: 1
Array de valores: 1, 2, 3, 1,
Array de soma: 1, 3, 6, 11,
Array codificado: 103, 247, 118, 162, 48, 162, 72,
Array final: 103, 50, 247, 118, 185, 51, 162, 48, 162, 241, 96, 147, 72, 123,
Mensagem final: g2§v§3§
Tamanho da mensagem final: 14
```

(a) Codificação da mensagem "UNESP13"com 7 bytes pelo sistema proposto em laboratório

```
#INICIANDO DECODIFICAÇÃO

Array da mensagem recebida: 103, 50, 247, 118, 185, 51, 162, 48, 162, 241, 96, 147, 72, 123,
Tamanho da mensagem recebida: 14
Array da mensagem codificada: 103, 247, 118, 162, 48, 162, 72,
Array da chave: 50, 185, 51, 241, 96, 147, 123,
Array da mensagem decodificada: 85, 78, 69, 83, 80, 49, 51,
Mensagem decodificada: UNESP13
```

(b) Decodificação da mensagem "UNESP13"com 7 bytes pelo sistema proposto em laboratório

Fonte: Autoria própria.

6.3 AVALIAÇÃO DE RESULTADOS

Conclui-se que a transmissão e recepção foram bem sucedidas pois os valores de entrada foram recebidos e decodificados no receptor como esperado. A mensagem no meio de transmissão não poderia ser recuperada por um acesso não autorizado sem conhecimento do algoritmo de decodificação, e este por sua vez é inacessível sem acesso ao código fonte do desenvolvedor.

7 CONCLUSÃO

O desenvolvimento deste trabalho de pesquisa e desenvolvimento foi de extrema importância para ampliar e atualizar o autor sobre as possibilidades de integração e aplicação de diferentes elementos dentro da área eletrônica e possibilitar uma forma de proteger dados durante o desenvolvimento, validação ou aplicação de um projeto, visto que a certeza na confidencialidade de informações prova-se muito relevante no cenário atual de telecomunicações.

Desenvolver um método de codificação baseado em elementos físicos e não em operações puramente virtuais apesar de parecer andar na contramão dos métodos de codificação de informações é uma forma de utilizar-se do não-óbvio para integrar diferentes áreas e assim cada vez mais aprimorar métodos e ferramentas para atingir a finalidade almejada. Afinal, novos métodos de codificação não apenas visam aumentar a eficiência dos atuais, mas também incrementar a eficácia e adicionar novos fatores de imprevisibilidade.

Partindo do objetivo de analisar meios de criptografar informações visando aumentar a confidencialidade transmissor-receptor, verificou-se que a tendência é cada vez maior em confiar em métodos virtuais para garantir esta confidencialidade, entretanto, existem constantes tentativas de executar engenharia reversa a fim de acessar informações que não deveriam ser recuperadas indevidamente. Com o integração de osciladores não lineares ao sistema é criada uma forma diferente de codificar informações com possível sincronização física de chaves, e, ao contrário dos tradicionais métodos de codificação, restrito apenas ao transmissor e receptor.

Além de utilizar componentes de baixo custo e relativa baixa complexidade em todo o projeto, o método de codificação desenvolvido também provou-se aleatório na geração de seus bits conforme era esperado do sistema e os resultados como um todo foram satisfatórios.

Por fim, a partir dos estudos realizados conclui-se que a atual aplicação imediata deste projeto é focado na área de Internet das Coisas, onde o comprimento das informações transmitidas é relativamente baixo, porém necessita de alta confidencialidade e confiabilidade e é possível estender a pesquisa visando a sincronização dos sinais digitalizados, aumento de frequência de oscilação e miniaturização de circuitos.

8 REFERÊNCIAS

APACHE. **Commons Math: The Apache Commons Mathematics Library**. 2019. [Online; atualizado em 2016]. Disponível em: <http://commons.apache.org/proper/commons-math/download_math.cgi>.

BANERJEE, S.; KURTHS, J. **Chaos and Cryptography: A new dimension in secure communications**. [S.l.]: Springer, 2014.

BASSHAM, L. E. et al. **A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications| NIST**. [S.l.], 2010.

CHUA, L. O. **The genesis of Chua's circuit**. [S.l.]: Electronics Research Laboratory, College of Engineering, University of California, 1992.

DEDIEU, H.; KENNEDY, M. P.; HASLER, M. Chaos shift keying: modulation and demodulation of a chaotic carrier using self-synchronizing chua's circuits. **Circuits and systems II: Analog and digital signal processing, IEEE Transactions on**, IEEE, v. 40, n. 10, p. 634–642, 1993.

DEMIRKOL, A. et al. High frequency chaos oscillators with applications. **ECCTD 2007**, 2007.

DESIGN, O. E. **Sallen-Key Low-pass Filter Design Tool <"http://sim.okawa-denshi.jp/en/OPseikiLowkeisan.htm»**. 2019. Disponível em: <http://sim.okawa-denshi.jp/en/OPseikiLowkeisan.htm>.

DUHAMEL, P.; VETTERLI, M. Fast fourier transforms: a tutorial review and a state of the art. **Signal processing**, Elsevier, v. 19, n. 4, p. 259–299, 1990.

FIEDLER-FERRARA, N.; PRADO, C. C. do. **Caos: uma introdução**. [S.l.]: Edgar Blucher, 1994.

GÁMEZ-GUZMÁN, L. et al. Synchronization of chua's circuits with multi-scroll attractors: application to communication. **Communications in Nonlinear Science and Numerical Simulation**, Elsevier, v. 14, n. 6, p. 2765–2775, 2009.

GÖHRING, M.; SCHMITZ, R. On randomness testing in physical layer key agreement. In: IEEE. **Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on**. [S.l.], 2015. p. 733–738.

JARREL, E. **Even Fibonacci numbers (Python vs. JavaScript) <"https://hackernoon.com/even-fibonacci-numbers-python-vs-javascript-55590ccb2fd6»**. 2019. Disponível em: <https://hackernoon.com/even-fibonacci-numbers-python-vs-javascript-55590ccb2fd6>.

JIANG, N. et al. Modeling and simulation of chaos-based security-enhanced wdm-pon. **Photonics Technology Letters, IEEE**, IEEE, v. 25, n. 19, p. 1912–1915, 2013.

JITEURTRAGOOL, N.; WANNABOON, C.; MASAYOSHI, T. True random number generator based on compact chaotic oscillator. In: IEEE. **2015 15th international symposium on communications and information technologies (ISCIT)**. [S.l.], 2015. p. 315–318.

JOTHIMURUGAN, R. et al. Improved realization of canonical chua's circuit with synthetic inductor using current feedback operational amplifiers. **AEU-International Journal of Electronics and Communications**, Elsevier, v. 68, n. 5, p. 413–421, 2014.

KELLERT, S. H. **In the wake of chaos: Unpredictable order in dynamical systems**. [S.l.]: University of Chicago press, 1994.

KENNEDY, M. P. Three steps to chaos. ii. a chua's circuit primer. **IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications**, IEEE, v. 40, n. 10, p. 657–674, 1993.

KILIÇ, R. **A practical guide for studying Chua's circuits**. [S.l.]: World Scientific, 2010.

KONG, L.; KADDOUM, G.; TAHA, M. Performance analysis of physical layer security of chaos-based modulation schemes. In: IEEE. **Wireless and Mobile Computing, Networking and Communications (WiMob), 2015 IEEE 11th International Conference on**. [S.l.], 2015. p. 283–288.

KUMAR, U.; SHUKLA, S. K. et al. Analytical study of inductor simulation circuits. **Active and Passive Electronic Components**, Hindawi, v. 13, n. 4, p. 211–227, 1989.

LAM, L. **Nonlinear physics for beginners: fractals, chaos, solitons, pattern formation, cellular automata and complex systems**. [S.l.]: World Scientific, 1998.

LEE, A. **Third World War already well underway in cyberspace**. 2018. <http://www.ejinsight.com/20170804-third-world-war-already-well-underway-in-cyberspace/>. [Online; accessado em 19 de Julho de 2008].

LÓPEZ, C. S. A 1.7 mhz chua's circuit using vms and cf+ s. **Revista mexicana de física**, Sociedad Mexicana de Física, v. 58, n. 1, p. 86–93, 2012.

MARMITT, R. **O Modelo OSI e suas 7 camadas <"http://ctisenac.blogspot.com.br/2012/03/o-modelo-osi-e-suas-7-camadas.html»**. 2012. Disponível em: <http://ctisenac.blogspot.com.br/2012/03/o-modelo-osi-e-suas-7-camadas.html>.

MAURITY, L. **O modelo em camadas, o modelo OSI <"http://br.ccm.net/faq/9415-o-modelo-em-camadas-o-modelo-osi»**. 2016. Disponível em: <http://br.ccm.net/faq/9415-o-modelo-em-camadas-o-modelo-osi>.

MORENO, E. D.; PEREIRA, F. D.; BARROS, R. Criptografia em software e hardware ed. **Novatec Editora Ltda**, 2006.

OGORZAŁEK, M. J. **Chaos and complexity in nonlinear electronic circuits**. [S.l.]: World Scientific, 1997.

PAIVA, W. P. d. A teoria do caos e as organizações. **REGE Revista de Gestão**, v. 8, n. 2, 2010.

ROGERS, G. **Provable Randomness: How to test RNGs <"https://www.unitychain.io/blog/how-to-test-rngs-for-true-randomness/»**. 2019. Disponível em: <https://www.unitychain.io/blog/how-to-test-rngs-for-true-randomness/>.

ŠALAMON, M. Chaotic electronic circuits in cryptography. In: **Applied cryptography and network security**. [S.l.: s.n.], 2012. p. 295.

SANTANA, T. L.; TERRA, M. **Osciladores Não-Lineares e caóticos e aplicações**. [S.l.]: Instituto Tecnológico de Aeronáutica, 2005.

SIDERSKIY, V. **Building Chua's Circuit <"http://www.chuacircuits.com/howtobuild2.php»**. 2019. Disponível em: <http://www.chuacircuits.com/howtobuild2.php>.

SIDERSKIY, V.; KAPILA, V. Parameter matching using adaptive synchronization of two chua's oscillators. **International Journal of Bifurcation and Chaos**, World Scientific, v. 24, n. 11, p. 1430032, 2014.

SPARROW, C. **The Lorenz equations: bifurcations, chaos, and strange attractors**. [S.l.]: Springer Science & Business Media, 2012. v. 41.

STACEY, R. D. **The chaos frontier: creative strategic control for business**. [S.l.]: Butterworth-Heinemann, 2016.

TEAM, M. **The Heartbleed Hit List: The Passwords You Need to Change Right Now <"http://mashable.com/2014/04/09/heartbleed-bug-websites-affected/od2OFVkYdaqV»**. 2014. Disponível em: <http://mashable.com/2014/04/09/heartbleed-bug-websites-affected/#od2OFVkYdaqV>.

US-CERT. **OpenSSL 'Heartbleed' vulnerability (CVE-2014-0160) <"https://www.us-cert.gov/ncas/alerts/TA14-098A»**. 2014. Disponível em: <OpenSSL'Heartbleed' vulnerability(CVE-2014-0160)>.

Wikipedia contributors. **Buffer (eletrônica) — Wikipédia, a enciclopédia livre**. 2017. [Online; atualizado em 16 de Dezembro de 2017]. Disponível em: <https://pt.wikipedia.org/w/index.php?title=Buffer_(eletr%C3%B4nica)&oldid=50771484>.

Wikipedia contributors. **Schmitt trigger — Wikipedia, The Free Encyclopedia**. 2019. [Online; atualizado em 9 de Agosto de 2019]. Disponível em: <https://en.wikipedia.org/w/index.php?title=Schmitt_trigger&oldid=899467663>.

A APÊNDICE

A.1 CLASSE EM JAVA PARA AVALIAÇÃO DE ALEATORIEDADE EM SEQUÊNCIA DE BITS

```
1  package analisador;
2
3  import static org.apache.commons.math3.special.Erf.erfc;
4  import org.apache.commons.math3.special.Gamma;
5
6  /**
7   *
8   * @author Gabriel Nogueira
9   */
10 public class AnalysisUtil {
11
12     public static double percentageOne(String binary) {
13         int zeroes = countNumberOfZeroes(binary);
14         return (((double) zeroes / (binary.length())));
15     }
16
17     public static double percentageZero(String binary) {
18         int ones = countNumberOfOnes(binary);
19         return (((double) ones / (binary.length())));
20     }
21
22     public static int countNumberOfOnes(String str) {
23         return (str.length() - str.replace("1", "").length());
24     }
25
26     public static int countNumberOfZeroes(String str) {
27         return (str.length() - str.replace("0", "").length());
28     }
29
30     /**
31      * O input   uma string binria composta de 0s e 1s. Inicia
           transformando os
32      * 0s em -1 e somando com os 1s. Calcula-se o S_n, s_obs e
           Pvalue em
33      * seguida. Se Pvalue < 0.01, a sequncia no   aleatria.
```

```
34      *
35      * @param binary
36      */
37     public static boolean frequencyTest(String binary) {
38
39         double n = binary.length(); //comprimento do binrio
40         double S_n = 0; //S_n = soma de 1s e -1s (sendo os zeros
               transformados em -1)
41         double s_obs = 0; //s_obs = (abs(S_n)) / (sqrt(n))
42         double Pvalue = 0; //Valor P = erfc(s_obs/sqrt(2))
43
44         for (int i = 0; i < n; i++) {
45             if (binary.charAt(i) == '0') {
46                 S_n += -1;
47             } else {
48                 S_n += 1;
49             }
50         }
51         System.out.println("S_n: " + S_n);
52
53         s_obs = (Math.abs(S_n)) / (Math.sqrt(n));
54         System.out.println("s_obs: " + s_obs);
55
56         Pvalue = erfc(s_obs / Math.sqrt(2));
57         System.out.println("Pvalue: " + Pvalue);
58
59         if (Pvalue >= 0.01) {
60             System.out.println("A sequncia  aleatria para o
                   teste de frequncia");
61             return true;
62         } else {
63             System.out.println("A sequncia NO  aleatria para o
                   teste de frequncia");
64             return false;
65         }
66
67     }
68
69     public static boolean frequencyTestBlock(String binary, int M)
           {
```

```
70          double n = binary.length(); //comprimento da string binria
71          String blocos[] = splitByNumber(binary, M);
72          double N = (n - (n % M)) / M; //Quantidade de blocos
               inteiros. Ex: se tenho 10 bits, e o bloco   de 3, terei
               3 blocos e 1 bit ser descartado
73 //          N = (n % M) == 0 ? blocos.length : blocos.length - 1; //
       Outra forma de calcular N: Caso a diviso seja exata, no
       necessrio descartar o ltimo bloco.
74          double Pvalue = 0;
75
76          double chi2 = 0; //chi quadrado
77          for (int i = 0; i < N; i++) {
78 //             System.out.println("contedo do bloco: " + blocos[i])
       ;
79              double pi = 0;
80              for (int j = 0; j < M; j++) {
81                  pi += Character.getNumericValue(blocos[i].charAt(j)
                       ); //Se for 1, soma 1. Se for 0, soma 0.
82              }
83              pi /= M; //o valor de pi   a quantidade de 1s no bloco
                   dividido pelo comprimento do bloco
84 //             System.out.println("pi: " + pi);
85
86              chi2 += (pi - 0.5) * (pi - 0.5);
87 //             System.out.println("Acumulado do argumento de chi^2:
       " + chi2);
88          }
89          chi2 *= (4 * M);
90 //         System.out.println("chi^2: " + chi2);
91
92          Pvalue = Gamma.regularizedGammaQ((N / 2), (chi2 / 2));
93          System.out.println("Pvalue:␣" + Pvalue);
94
95          if (Pvalue >= 0.01) {
96              System.out.println("A␣sequncia␣ ␣aleatria␣para␣o␣
                   teste␣de␣blocos[" + M + "].");
97              return true;
98          } else {
99              System.out.println("A␣sequncia␣N O␣ ␣aleatria␣para␣o␣
                   teste␣de␣blocos[" + M + "].");
```

```
100             return false;
101         }
102     }
103
104     /**
105      * Transforma string em um string array em que cada elemento
            contm o
106      * tamanho conforme "size". O ltimo bloco de elementos no
            necessariamente
107      * possuir comprimento igual
108      *
109      * @param s
110      * @param size
111      * @return
112      */
113     private static String[] splitByNumber(String s, int size) {
114         if (s == null || size <= 0) {
115             return null;
116         }
117         int chunks = s.length() / size + ((s.length() % size > 0) ?
                1 : 0);
118         String[] arr = new String[chunks];
119         for (int i = 0, j = 0, l = s.length(); i < l; i += size, j
               ++) {
120             arr[j] = s.substring(i, Math.min(l, i + size));
121         }
122         return arr;
123     }
124
125     /**
126      * O input  uma string binria. O teste de sequncia permite
            avaliar
127      * aleatoriedade baseado em sequncias de mesmo bit. Caso falhe
             o teste de
128      * frequncia bsico, o teste de sequncia automaticamente
            falha.
129      *
130      * @param binary
131      */
132     public static boolean runTest(String binary) {
```

```
133         int n = binary.length();
134         double pi = ((double) countNumberOfOnes(binary) / n);
135         double tau = (double) 2 / Math.sqrt(n);
136         double Pvalue = 0;
137 
138         System.out.println("tau: " + tau); //Se tau >= 0.1, o teste
                de sequncia pode ser aplicado.
139         int Vn = 0; //Vn   o nmero de "blocos" de mesmo bit
               seguido. Ex: 1001101011 = 1;00;11;0;1;0;11. Vn = 7.
140         for (int i = 0; i < n - 1; i++) {
141             int ek = Character.getNumericValue(binary.charAt(i));
142             int ekp1 = Character.getNumericValue(binary.charAt(i +
                   1));
143             if (ek != ekp1) {
144                 Vn += 1;
145             }
146         }
147         Vn += 1; //Veio da equa . Aps  a somatria  soma-se 1
               para o valor real de Vn.
148 //          System.out.println("Vn: " + Vn);
149 
150         Pvalue = erfc((Math.abs(Vn - 2 * n * pi * (1 - pi))) / (2 *
                (Math.sqrt(2 * n)) * pi * (1 - pi)));
151 //          System.out.println("Pvalue: " + Pvalue);
152 
153         if (Pvalue >= 0.01) {
154             System.out.println("A sequncia  aleatria para o
                   teste de sequncia");
155             return true;
156         } else {
157             System.out.println("A sequncia NO  aleatria para o
                   teste de sequncia");
158             return false;
159         }
160     }
161 
162     public static boolean runLongestTest(String binary, int M) {
163         boolean pass = false;
164         double n = binary.length(); //comprimento da string binria
165         double blocoN = (n - (n % M)) / M; //Quantidade de blocos
```

```
            inteiros. Ex: se tenho 10 bits, e o bloco   de 3, terei
            3 blocos e 1 bit ser descartado
166         String blocos[] = splitByNumber(binary, M);
167         double chi2 = 0;
168         double Pvalue = 0;
169
170         //inicializa de variveis
171         int K = 3; //tabelado para M = 8
172         int N = 16; //tabelado para M = 8
173         double pi0 = 0.2148; //tabelado para M = 8
174         double pi1 = 0.3672; //tabelado para M = 8
175         double pi2 = 0.2305; //tabelado para M = 8
176         double pi3 = 0.1875; //tabelado para M = 8
177         double pi4 = 0.1027; //tabelado para M = 128
178         double pi5 = 0.1124; //tabelado para M = 128
179         int v0 = 0; //1 ocorrncia de <= 1 no bloco para M = 8
180         int v1 = 0; //1 ocorrncia de 2 no bloco para M = 8
181         int v2 = 0; //1 ocorrncia de 3 no bloco para M = 8
182         int v3 = 0; //1 ocorrncia de >=4 no bloco para M = 8
183         int v4 = 0; //1 ocorrncia de 3 no bloco para M = 128
184         int v5 = 0; //1 ocorrncia de >=4 no bloco para M = 128
185         double a = 0;
186         double x = 0;
187
188         switch (M) {
189             case 8:
190                 System.out.println("CASE block 8");
191                 K = 3; //tabelado para M = 8
192                 N = (int) (n/M); //tabelado para M = 8
193                 pi0 = 0.2148; //tabelado para M = 8
194                 pi1 = 0.3672; //tabelado para M = 8
195                 pi2 = 0.2305; //tabelado para M = 8
196                 pi3 = 0.1875; //tabelado para M = 8
197                 v0 = 0; //1 ocorrncia de <= 1 no bloco
198                 v1 = 0; //1 ocorrncia de 2 no bloco
199                 v2 = 0; //1 ocorrncia de 3 no bloco
200                 v3 = 0; //1 ocorrncia de >=4 no bloco
201
202                 for (int i = 0; i < blocoN; i++) {
203 //                    System.out.println("contedo do bloco: " +
```

```
    blocos[i]);
204
205                 v0 += 1; //se conter um ou nenhum 1.
206
207                 if (blocos[i].contains("11")) {
208                     v0 -= 1;
209                     v1 += 1;
210                 }
211                 if (blocos[i].contains("111")) {
212                     v1 -= 1;
213                     v2 += 1;
214                 }
215                 if (blocos[i].contains("1111")) {
216                     v2 -= 1;
217                     v3 += 1;
218                 }
219             }
220             System.out.println("v_0: " + v0);
221             System.out.println("v_1: " + v1);
222             System.out.println("v_2: " + v2);
223             System.out.println("v_3: " + v3);
224
225             chi2
226                     = (( (v0 - (N * pi0)) * (v0 - (N * pi0)) )
                           / (N * pi0))
227                     + (((v1 - (N * pi1)) * (v1 - (N * pi1))) /
                           (N * pi1))
228                     + (((v2 - (N * pi2)) * (v2 - (N * pi2))) /
                           (N * pi2))
229                     + (((v3 - (N * pi3)) * (v3 - (N * pi3))) /
                           (N * pi3));
230
231             a = (K / 2);
232             x = (chi2 / 2);
233             Pvalue = Gamma.regularizedGammaQ(a, x);
234             System.out.println("Pvalue: " + Pvalue);
235
236             if (Pvalue >= 0.01) {
237                 System.out.println("A sequncia  aleatria
                       para o teste de maior sequncia");
```

```
238                     pass = true;
239                 } else {
240                     System.out.println("A␣sequncia␣N O␣ ␣aleatria
                        ␣para␣o␣teste␣de␣maior␣sequncia");
241                     pass = false;
242                 }
243                 break;
244             case 128:
245                 System.out.println("CASE␣block␣128");
246                 K = 5; //tabelado para M = 128
247                 N = (int) (n/M); //tabelado para M = 128
248                 pi0 = 0.1174; //tabelado para M = 128
249                 pi1 = 0.2430; //tabelado para M = 128
250                 pi2 = 0.2493; //tabelado para M = 128
251                 pi3 = 0.1752; //tabelado para M = 128
252                 pi4 = 0.1027; //tabelado para M = 128
253                 pi5 = 0.1124; //tabelado para M = 128
254                 v0 = 0; //1 ocorrncia de <= 4 no bloco
255                 v1 = 0; //1 ocorrncia de 5 no bloco
256                 v2 = 0; //1 ocorrncia de 6 no bloco
257                 v3 = 0; //1 ocorrncia de 7 no bloco
258                 v4 = 0; //1 ocorrncia de 8 no bloco
259                 v5 = 0; //1 ocorrncia de >=9 no bloco
260
261                 for (int i = 0; i < blocoN; i++) {
262                     System.out.println("contedo␣do␣bloco:␣" +
                        blocos[i]);
263
264                     v0 += 1; //se conter um ou nenhum 1.
265
266                     if (blocos[i].contains("11111")) {
267                         v0 -= 1;
268                         v1 += 1;
269                     }
270                     if (blocos[i].contains("111111")) {
271                         v1 -= 1;
272                         v2 += 1;
273                     }
274                     if (blocos[i].contains("1111111")) {
275                         v2 -= 1;
```

```
276                         v3 += 1;
277                     }
278                     if (blocos[i].contains("11111111")) {
279                         v3 -= 1;
280                         v4 += 1;
281                     }
282                     if (blocos[i].contains("111111111")) {
283                         v4 -= 1;
284                         v5 += 1;
285                     }
286                 }
287 
288                 chi2
289                         = (((v0 - (N * pi0)) * (v0 - (N * pi0))) /
                               (N * pi0))
290                         + (((v1 - (N * pi1)) * (v1 - (N * pi1))) /
                               (N * pi1))
291                         + (((v2 - (N * pi2)) * (v2 - (N * pi2))) /
                               (N * pi2))
292                         + (((v3 - (N * pi3)) * (v3 - (N * pi3))) /
                               (N * pi3))
293                         + (((v4 - (N * pi4)) * (v4 - (N * pi4))) /
                               (N * pi4))
294                         + (((v5 - (N * pi5)) * (v5 - (N * pi5))) /
                               (N * pi5));
295 
296                 a = (K / 2);
297                 x = (chi2 / 2);
298                 Pvalue = Gamma.regularizedGammaQ(a, x);
299                 System.out.println("Pvalue:␣" + Pvalue);
300 
301                 if (Pvalue >= 0.01) {
302                     System.out.println("A␣sequncia␣ ␣aleatria␣
                           para␣o␣teste␣de␣maior␣sequncia");
303                     pass = true;
304                 } else {
305                     System.out.println("A␣sequncia␣N O␣ ␣aleatria
                           ␣para␣o␣teste␣de␣maior␣sequncia");
306                     pass = false;
307                 }
```

```
308                 break;
309             default:
310                 System.out.println("CASE block DEFAULT");
311                 K = 3; //tabelado para M = 8
312                 N = (int) (n/M); //tabelado para M = 8
313                 pi0 = 0.2148; //tabelado para M = 8
314                 pi1 = 0.3672; //tabelado para M = 8
315                 pi2 = 0.2305; //tabelado para M = 8
316                 pi3 = 0.1875; //tabelado para M = 8
317                 v0 = 0; //1 ocorrncia de <= 1 no bloco
318                 v1 = 0; //1 ocorrncia de 2 no bloco
319                 v2 = 0; //1 ocorrncia de 3 no bloco
320                 v3 = 0; //1 ocorrncia de >=4 no bloco
321 
322                 for (int i = 0; i < blocoN; i++) {
323 //                    System.out.println("contedo do bloco: " +
    blocos[i]);
324 
325                     v0 += 1; //se conter um ou nenhum 1.
326 
327                     if (blocos[i].contains("11")) {
328                         v0 -= 1;
329                         v1 += 1;
330                     }
331                     if (blocos[i].contains("111")) {
332                         v1 -= 1;
333                         v2 += 1;
334                     }
335                     if (blocos[i].contains("1111")) {
336                         v2 -= 1;
337                         v3 += 1;
338                     }
339                 }
340 //                System.out.println("v_0: " + v_0);
341 //                System.out.println("v_1: " + v_1);
342 //                System.out.println("v_2: " + v_2);
343 //                System.out.println("v_3: " + v_3);
344 
345                 chi2
346                         = (((v0 - (N * pi0)) * (v0 - (N * pi0))) /
```

```
                                (N * pi0))
347                           + (((v1 - (N * pi1)) * (v1 - (N * pi1))) /
                                (N * pi1))
348                           + (((v2 - (N * pi2)) * (v2 - (N * pi2))) /
                                (N * pi2))
349                           + (((v3 - (N * pi3)) * (v3 - (N * pi3))) /
                                (N * pi3));
350
351                   a = (K / 2);
352                   x = (chi2 / 2);
353                   Pvalue = Gamma.regularizedGammaQ(a, x);
354                   System.out.println("Pvalue: " + Pvalue);
355
356                   if (Pvalue >= 0.01) {
357                       System.out.println("A sequncia  aleatria
                              para o teste de maior sequncia");
358                       pass = true;
359                   } else {
360                       System.out.println("A sequncia N O  aleatria
                              para o teste de maior sequncia");
361                       pass = false;
362                   }
363                   break;
364           }
365           return pass;
366
367       }
368
369
370   }
```

A.2 CÓDIGO ARDUINO PARA CODIFICAÇÃO DE DADOS

```
1  #include <SoftwareSerial.h>
2  SoftwareSerial chuaSerial(10, 11); // (RX Chua, TX Chua);
3  SoftwareSerial txSerial(5, 6); //utilizado diferente para
       transmitir a um baud rate maior
4
5  void setup() {
6    Serial.begin(115200);
7    chuaSerial.begin(100);
```

```
8   txSerial.begin(9600);
9   Serial.println("Serial iniciada\n");
10  Serial.print("Digite sua mensagem: "); //Mensagem de entrada
11 }
12
13 void loop() {
14  String msgOriginal = inputMessage();
15  codificacao(msgOriginal);
16 }
17
18 String inputMessage() { //para o usuario escrever sua mensagem
19  String msg = "";
20  while (Serial.available() > 0) { //Aguarda o usu rio digitar a
       mensagem
21  }
22  msg = Serial.readString();
23  return msg;
24 }
25
26 void codificacao(String msgOriginal) {
27
28  /*************************************************************
29
30    CODIFICA  O
31
32   *************************************************************/
33
34  int msgSize = msgOriginal.length() + 1;
35  byte msg[msgSize];
36  msgOriginal.getBytes(msg, msgSize);
37
38  if (msgOriginal != "") {
39    Serial.println(msgOriginal);
40    Serial.print("Mensagem a ser enviada(chars): ");
41    for (int i = 0; i < msgSize; i++) {
42      Serial.print((char)msg[i]);
43    }
44    Serial.println("");
45
46    Serial.print("Mensagem a ser enviada(bytes): ");
```

```
47    for (int i = 0; i < msgSize; i++) {
48      Serial.print(msg[i]);
49      Serial.print(",");
50    }
51    Serial.println("\n");
52
53
54
55    Serial.println("#INICIANDO␣CODIFICA  O\n");
56    byte key[msgSize];
57
58    for (int i = 0; i < msgSize; i++) {
59      //      key[i] = (int)random(0, 255);
60      if (chuaSerial.available()) {
61        key[i] = chuaSerial.read();
62        if (i > 0) {
63          key[i] = key[i] ^ key[i - 1];
64        }
65        delay(10);
66      }
67    }
68
69    Serial.print("Tamanho␣da␣mensagem:␣"); Serial.println(msgSize);
70
71    Serial.print("Mensagem:␣");
72    for (int i = 0; i < msgSize; i++) {
73      Serial.print((char)msg[i]);
74    }
75    Serial.println("");
76
77    Serial.print("Chave:␣");
78    for (int i = 0; i < msgSize; i++) {
79      Serial.print((char)key[i]);
80    }
81    Serial.println("");
82
83    Serial.print("Array␣chave:␣");
84    for (int i = 0; i < msgSize; i++) {
85      Serial.print(key[i]); Serial.print(",␣");
86    }
```

```
87      Serial.println("");
88
89      /*
90         GERAR PARMETROS DE FIBONACCI - ITERA  ES
91      */
92
93      int f1 = 0; //fibonacci
94      int f2 = 1;
95      int f3 = 1;
96      int sum = 0; //inicializa o
97      int it = 0; //itera  es
98      int limit = msgSize;
99
100     while (sum < limit) {
101       sum += f3;
102       f3 += f2;
103       f1 = f2;
104       f2 = f3 - f1;
105       if (sum < limit) {
106         it++;
107       }
108     }
109     Serial.print("Itera  es:␣"); Serial.println(it);
110
111     /*
112        GERAR PARMETROS DE FIBONACCI - VALORES
113     */
114
115     int arraySize = it + 1;
116     int valueArray[arraySize];
117     int sumArray[arraySize];
118     int leftover = 0;
119
120     f1 = 0; //inicializa o
121     f2 = 1;
122     f3 = 1;
123     sum = 0;
124
125     for (int i = 0; i <= it; i++) {
126       sum += f3;
```

```
127       f3 += f2;
128       f1 = f2;
129       f2 = f3 - f1;
130       if (sum < limit) {
131         valueArray[i] = f2;
132         sumArray[i] =  sum;
133       } else {
134         leftover = limit - sum + f2;
135         valueArray[i] = leftover; //adiciona o resto no array de
              valores
136         sumArray[i] = sum; //adiciona o resto no array de valores
137       }
138     }
139
140     Serial.print("Resto: "); Serial.println(leftover);
141
142     Serial.print("Array de valores: ");
143     for (int i = 0; i < arraySize; i++) {
144       Serial.print(valueArray[i]); Serial.print(", ");
145     }
146     Serial.println("");
147
148     Serial.print("Array de soma: ");
149     for (int i = 0; i < arraySize; i++) {
150       Serial.print(sumArray[i]); Serial.print(", ");
151     }
152     Serial.println("");
153
154     /*
155        GERAR BYTES CODIFICADOS
156     */
157
158     byte coded[msgSize];
159     int count = 0;
160     for (int i = 0; i < arraySize; i++) { //4 posi  es para l =
          7.
161       for (int j = 0; j < valueArray[i]; j++) { //1,2,3,1
162         coded[count] = (byte)(msg[count] ^ key[count]);
163         count++;
164       }
```

```
165     }
166
167     Serial.print("Array codificado: ");
168     for (int i = 0; i < msgSize; i++) {
169       Serial.print(coded[i]); Serial.print(", ");
170     }
171     Serial.println("");
172
173     /*
174        EMBARALHAR BYTES
175     */
176
177     byte shuffle[(msgSize * 2)];
178     count = 0;
179     int count0 = 0;
180     int count1 = 0;
181     for (int i = 0; i < arraySize; i++) { //[1,2,3,1] para l=7
182       for (int j = 0; j < valueArray[i]; j++) {
183         shuffle[count] = coded[count0];
184         count++;
185         count0++;
186       }
187       for (int j = 0; j < valueArray[i]; j++) {
188         shuffle[count] = key[count1];
189         count++;
190         count1++;
191       }
192     }
193
194     String codedStringByteValues = "";
195     Serial.print("Array final: ");
196     for (int i = 0; i < (msgSize * 2); i++) {
197       Serial.print(shuffle[i]); Serial.print(", ");
198       codedStringByteValues += (int)shuffle[i];
199       codedStringByteValues += ",";
200     }
201     Serial.println("");
202
203     Serial.print("ByteArray em String: "); Serial.println(
        codedStringByteValues);
```

```
204
205     Serial.print("Mensagem final: ");
206     for (int i = 0; i < msgSize; i++) {
207       Serial.print((char)shuffle[i]);
208     }
209     Serial.println("");
210     Serial.print("Tamanho da mensagem final: "); Serial.println(
          sizeof(shuffle));
211     Serial.println("\n");
212     txSerial.print(codedStringByteValues);
213
214     Serial.print("\nDigite sua mensagem: "); //Mensagem de entrada
215   }
216 }
```

A.3 CÓDIGO ARDUINO PARA DECODIFICAÇÃO DE DADOS

```
1  #include <SoftwareSerial.h>
2  SoftwareSerial chuaSerial(10, 11); // (RX Chua, TX Chua);
3  SoftwareSerial serialReceptor(10, 11); // RX, TX
4
5  void setup() {
6    Serial.begin(115200);
7    chuaSerial.begin(100);
8    txSerial.begin(9600);
9    Serial.println("Serial iniciada\n");
10   Serial.print("Digite sua mensagem: "); //Mensagem de entrada
11 }
12
13 void loop() {
14   String mensagemRecebida = receiveMessage();
15   if (mensagemRecebida.length() > 0) {
16     decodificacao(mensagemRecebida);
17   }
18 }
19
20 String receiveMessage() {
21   String mensagem = "";
22   while (serialReceptor.available() > 0) { //Aguarda receber alguma
         mensagem
23     ;
```

```
24    }
25    mensagem = serialReceptor.readString();
26    return mensagem;
27  }
28
29  void decodificacao(String codedStringByteValues) {
30
31    int finalPos = 0;
32    String teste = codedStringByteValues;
33    String testesub = teste;
34
35    int stringByteCount = 0; //quantidade de virgulas = quantidade de
          bytes
36    for (int i = 0; i < teste.length(); i++) {
37      if ((char)teste[i] == ',') {
38        stringByteCount++;
39      }
40    }
41    Serial.print("Quantidade de virgulas: "); Serial.println(
         stringByteCount);
42
43    byte received[stringByteCount]; //Bytes efetivamente recebidos
44
45    //Pega a string e divide ela de acordo com o delimitador ','
46    //  while (teste.indexOf(',') != -1) { //substituir pelo for se
         precisar que seja mais gen rico
47    for (int i = 0; i < stringByteCount; i++) {
48      finalPos = teste.indexOf(',');
49      testesub = teste.substring(0, finalPos);
50      teste = teste.substring(finalPos + 1, teste.length());
51      received[i] = (byte)testesub.toInt();
52    }
53
54    /*************************************************************
55
56      DECODIFICA O
57
58     *************************************************************/
59
60    Serial.println("#INICIANDO DECODIFICA O\n");
```

```
61
62  int receivedMsgSize = sizeof(received);
63  int msgSize = receivedMsgSize / 2;
64
65  Serial.print("Array da mensagem recebida: ");
66  for (int i = 0; i < receivedMsgSize; i++) {
67    Serial.print(received[i]); Serial.print(", ");
68  }
69  Serial.println("");
70  Serial.print("Tamanho da mensagem recebida: "); Serial.println(
       receivedMsgSize);
71
72  /*
73     GERAR PARMETROS DE FIBONACCI - ITERA ES
74  */
75
76  int f1 = 0; //fibonacci
77  int f2 = 1;
78  int f3 = 1;
79  int sum = 0; //inicializa o
80  int it = 0; //itera es
81  int limit = msgSize;
82
83  while (sum < limit) {
84    sum += f3;
85    f3 += f2;
86    f1 = f2;
87    f2 = f3 - f1;
88    if (sum < limit) {
89      it++;
90    }
91  }
92  Serial.print("Itera es: "); Serial.println(it);
93
94  /*
95     GERAR PARMETROS DE FIBONACCI - VALORES
96  */
97
98  int arraySize = it + 1;
99  int valueArray[arraySize];
```

```
100  int sumArray[arraySize];
101  int leftover = 0;
102
103  f1 = 0; //inicializa o
104  f2 = 1;
105  f3 = 1;
106  sum = 0;
107
108  for (int i = 0; i <= it; i++) {
109    sum += f3;
110    f3 += f2;
111    f1 = f2;
112    f2 = f3 - f1;
113    if (sum < limit) {
114      valueArray[i] = f2;
115      sumArray[i] =  sum;
116    } else {
117      leftover = limit - sum + f2;
118      valueArray[i] = leftover; //adiciona o resto no array de
           valores
119      sumArray[i] = sum; //adiciona o resto no array de valores
120    }
121  }
122
123  Serial.print("Resto: "); Serial.println(leftover);
124
125  Serial.print("Array de valores: ");
126  for (int i = 0; i < arraySize; i++) {
127    Serial.print(valueArray[i]); Serial.print(", ");
128  }
129  Serial.println("");
130
131  Serial.print("Array de soma: ");
132  for (int i = 0; i < arraySize; i++) {
133    Serial.print(sumArray[i]); Serial.print(", ");
134  }
135  Serial.println("");
136
137  //realizado acima, separar quando for colocar no decodificador
138
```

```
139  /*
140     SEPARANDO CHAVE DA MENSAGEM
141  */
142
143  byte decodedKey[msgSize];
144  byte codedMessage[msgSize];
145  int count = 0;
146  int count0 = 0;
147  int count1 = 0;
148  for (int i = 0; i < arraySize; i++) { //[1,2,3,1] para l=7, 4
        elementos.
149    for (int j = 0; j < valueArray[i]; j++) {
150      codedMessage[count0] = received[count];
151      count++;
152      count0++;
153    }
154    for (int j = 0; j < valueArray[i]; j++) {
155      decodedKey[count1] = received[count];
156      count++;
157      count1++;
158    }
159  }
160
161  Serial.print("Array␣da␣mensagem␣codificada:␣");
162  for (int i = 0; i < msgSize; i++) {
163    Serial.print(codedMessage[i]); Serial.print(",␣");
164  }
165  Serial.println("");
166
167  Serial.print("Array␣da␣chave:␣");
168  for (int i = 0; i < msgSize; i++) {
169    Serial.print(decodedKey[i]); Serial.print(",␣");
170  }
171  Serial.println("");
172
173  /*
174     DECODIFICAR BYTES CODIFICADOS
175  */
176
177  byte decoded[msgSize];
```

```
178   count = 0;
179   for (int i = 0; i < arraySize; i++) { //4 posi  es para l = 7.
180     for (int j = 0; j < valueArray[i]; j++) { //1,2,3,1
181       decoded[count] = (byte)(codedMessage[count] ^ decodedKey[
              count]);
182       count++;
183     }
184   }
185
186   Serial.print("Array␣da␣mensagem␣decodificada:␣");
187   for (int i = 0; i < msgSize; i++) {
188     Serial.print(decoded[i]); Serial.print(",␣");
189   }
190   Serial.println("");
191
192   Serial.print("Mensagem␣decodificada:␣");
193   for (int i = 0; i < msgSize; i++) {
194     Serial.print((char)decoded[i]);
195   } Serial.println("");
196   Serial.println("
        ----------------------------------------------------------------
        n");
197 }
```