



INSTITUTO FEDERAL
Goiás

Câmpus
Goiânia

ENGENHARIA DE CONTROLE E AUTOMAÇÃO

MATEUS MORAIS DE AGUIRRE

GERADOR DE SINAIS COM RASPBERRY PI PICO

Goiânia – GO
2024



INSTITUTO FEDERAL
Goiás

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
SISTEMA INTEGRADO DE BIBLIOTECAS

TERMO DE AUTORIZAÇÃO PARA DISPONIBILIZAÇÃO NO REPOSITÓRIO DIGITAL DO IFG - ReDi IFG

Com base no disposto na Lei Federal nº 9.610/98, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia de Goiás, a disponibilizar gratuitamente o documento no Repositório Digital (ReDi IFG), sem ressarcimento de direitos autorais, conforme permissão assinada abaixo, em formato digital para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IFG.

Identificação da Produção Técnico-Científica

- | | |
|--|---|
| ☐ Tese | ☐ Artigo Científico |
| ☐ Dissertação | ☐ Capítulo de Livro |
| ☐ Monografia – Especialização | ☐ Livro |
| ☒ TCC - Graduação | ☐ Trabalho Apresentado em Evento |
| ☐ Produto Técnico e Educacional - Tipo: _____ | |

Nome Completo do Autor: Mateus Morais de Aguirre

Matrícula: 20132010700149

Título do Trabalho: GERADOR DE SINAIS COM RASPBERRY PI PICO

Autorização - Marque uma das opções

1. (X) Autorizo disponibilizar meu trabalho no Repositório Digital do IFG (acesso aberto);
2. () Autorizo disponibilizar meu trabalho no Repositório Digital do IFG somente após a data ____/____/____ (Embargo);
3. () Não autorizo disponibilizar meu trabalho no Repositório Digital do IFG (acesso restrito).

Ao indicar a opção **2 ou 3**, marque a justificativa:

- () O documento está sujeito a registro de patente.
() O documento pode vir a ser publicado como livro, capítulo de livro ou artigo.
() Outra justificativa: _____

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O/A referido/a autor/a declara que:

- i. o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;
- ii. obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autor/a, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia de Goiás os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;
- iii. cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia de Goiás.



Documento assinado digitalmente
MATEUS MORAIS DE AGUIRRE
Data: 19/09/2024 10:47:37-0300
Verifique em <https://validar.iti.gov.br>

Goiânia - GO, 19/09/2024.
Local Data

Assinatura do Autor e/ou Detentor dos Direitos Autorais



INSTITUTO FEDERAL
Goiás

Câmpus
Goiânia

ENGENHARIA DE CONTROLE E AUTOMAÇÃO

MATEUS MORAIS DE AGUIRRE

GERADOR DE SINAIS COM RASPBERRY PI PICO

Trabalho de conclusão de curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Goiás, Campus Goiânia, como parte dos requisitos para a obtenção do Título de Engenheiro de Controle e Automação.

Orientador: Prof. Dr. Cláudio Afonso Fleury

Goiânia – GO
2024

A284g Aguirre, Mateus Morais de.

Gerador de sinais com Raspberry Pi Pico / Mateus Morais de Aguirre. –
Goiânia: Instituto Federal de Educação, Ciência e Tecnologia de Goiás, 2024.
93f.

Orientação: Prof. Dr. Cláudio Afonso Fleury.

TCC (Trabalho de Conclusão de Curso) – Curso de Bacharelado em
Engenharia de Controle e Automação, Instituto Federal de Educação, Ciência e
Tecnologia de Goiás.

1. Automação. 2. Raspberry Pi (computador). 3. Processamento de sinais. I.
Fleury, Cláudio Afonso (orientação). II. Instituto Federal de Educação, Ciência e
Tecnologia de Goiás. III. Título.

CDD 005.1



INSTITUTO FEDERAL
Goiás

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS
CÂMPUS GOIÂNIA

ATA DE DEFESA DE TCC 2

ATA DA SESSÃO PÚBLICA ONLINE DE APRESENTAÇÃO E DEFESA DO TRABALHO DE CONCLUSÃO DE CURSO DE BACHARELADO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO DAS ÁREAS ACADÊMICAS IV DO INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS, CAMPUS GOIÂNIA.

Aos **10** dias do mês de **setembro** de **dois mil e vinte e quatro**, às **08:30** na sala **S-310** do IFG Campus Goiânia, foi realizada a sessão pública de apresentação e defesa do Trabalho de Conclusão de Curso intitulado **"Gerador de Sinais com Raspberry Pi Pico"**, de autoria do graduando **Mateus Moraes Aguirre** (matricula: [20132010700149](#)), acadêmico do curso de Bacharelado em Engenharia de Controle e Automação, como requisito final para obtenção do título de Engenheiro de Controle e Automação. A banca examinadora foi composta pelos seguintes membros: **Prof. Dr. Cláudio Afonso Fleury** (Orientador/Presidente), do IFG Campus Goiânia, **Prof. Me. Eliézer Alves Teixeira**, do IFG Campus Goiânia e **Prof. Dr. Charles dos Santos Costa**, do IFG Campus Goiânia.

Após a apresentação, o autor foi arguido pela banca examinadora cujo parecer final foi:

() Aprovado (x) Aprovado com correções () Aprovado com restrições () Reprovado

A presente ata foi lavrada após encerrar-se a sessão, sendo assinada pela banca examinadora.

Observações: O aluno deverá fazer as correções sugeridas pela Banca Examinadora no texto da monografia e apresentá-las ao orientador para providências legais.

O trabalho corrigido deverá ser apresentado ao Professor Orientador no prazo máximo de 30 dias corridos a partir da data da defesa.

FECHAMENTO: Nada mais havendo a tratar, o presidente da banca avaliadora, Prof. Dr. Cláudio Afonso Fleury, agradece a participação de todos e encerra a sessão pública de apresentação e defesa do Trabalho de Conclusão de Curso (TCC) às **10 horas e 15 minutos**. Eu, Leonardo Costa de Paula, Coordenador do curso de Bacharelado em Engenharia de Controle e Automação, sem mais nada a constar neste documento, lavrei a presente ata assinada digitalmente.

Banca Examinadora:

Prof. Dr. Cláudio Afonso Fleury (Orientador/Presidente) - IFG

Prof. Me. Eliézer Alves Teixeira - IFG

Prof. Dr. Charles dos Santos Costa - IFG

Goiânia, 10 de setembro de 2024.

Documento assinado eletronicamente por:

- **Charles dos Santos Costa**, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 17/09/2024 07:07:08.
- **Eliezer Alves Teixeira**, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 16/09/2024 15:30:31.
- **Claudio Afonso Fleury**, PROFESSOR ENS BASICO TECN TECNOLOGICO, em 16/09/2024 15:01:50.

Este documento foi emitido pelo SUAP em 06/09/2024. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse <https://suap.ifg.edu.br/autenticar-documento/> e forneça os dados abaixo:

Código Verificador: 560733

Código de Autenticação: 530837e20f



Instituto Federal de Educação, Ciência e Tecnologia de Goiás
Rua 75, nº 46, Centro, GOIÂNIA / GO, CEP 74055-110
Sem Telefones cadastrados

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS

MATEUS MORAIS DE AGUIRRE

GERADOR DE SINAIS COM RASPBERRY PI PICO

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Goiás, Campus Goiânia, como parte dos requisitos para a obtenção do título de Engenheiro de Controle e Automação.

Aprovado em: ____/____/____

Por:

Banca examinadora:

Prof. Dr. Cláudio Afonso Fleury
Instituto Federal de Educação, Ciência e Tecnologia de Goiás (IFG)

Prof. Me. Eliézer Alves Teixeira
Instituto Federal de Educação, Ciência e Tecnologia de Goiás (IFG)

Prof. Me. Charles dos Santos Costa
Instituto Federal de Educação, Ciência e Tecnologia de Goiás (IFG)

Aos meus pais pelo apoio e paciência durante o curso.

AGRADECIMENTOS

Agradeço aos meus pais e amigos. Principalmente ao meu pai que me ajudou nas correções gramaticais.

A todos os professores que fizeram parte da minha graduação, principalmente ao Prof. Dr. Cláudio Afonso Fleury, meu orientador desta monografia, por sua orientação, dedicação e paciência.

Aos professores, membros da banca avaliadora, Prof. Me. Eliézer e Prof. Me. Charles.

E a todos os profissionais que fazem parte do Instituto Federal de Goiás, Campus Goiânia que, mesmo indiretamente, contribuíram para a minha formação.

Aos professores e amigos de meus pais, Eunice Dias de Paula Luiz Gouvêa de Paula, por apontar os gramaticais de termos científicos.

RESUMO

Este trabalho tem por objetivo o desenvolvimento de um gerador de sinais de baixo custo para aplicação nos cursos técnicos e de engenharias do IFG, utilizando como plataforma de desenvolvimento a placa Raspberry Pi Pico. Fazem parte do gerador de sinais o próprio Raspberry Pi Pico, programado em Micropython, responsável pelo processamento digital dos sinais, e um circuito externo, responsável pelo condicionamento dos sinais. O Raspberry Pi gera e controla os sinais digitais conforme a forma de onda desejada e seus parâmetros (amplitude, frequência e fase). O circuito de condicionamento de sinais é responsável pela conversão dos sinais digitais em sinais analógicos, pela amplificação dos sinais e pela redução de possíveis ruídos. O projeto foi dividido em três partes: I – Obtenção dos parâmetros para a programação das saídas digitais; II – Conversão digital para analógico; III – Circuito para alternar, amplificar e filtrar o sinal. O projeto visou a simplificação em termos de variação de componentes eletrônicos ficando restrito ao microcontrolador, resistores, amplificadores operacionais e apenas um capacitor.

Palavras-chave: Gerador de sinais. Raspberry Pi Pico, Automação, Conversão Digital-Analógico.

ABSTRACT

The objective of this work is to develop a low-cost signal generator for use in technical and engineering courses at IFG, using the Raspberry Pi Pico board as a development platform. The signal generator includes the Raspberry Pi Pico itself, programmed in Micropython, responsible for digital signal processing, and an external circuit, responsible for signal conditioning. The Raspberry Pi generates and controls digital signals according to the desired waveform and its parameters (amplitude, frequency and phase). The signal conditioning circuit is responsible for converting digital signals into analog signals, amplifying the signals and reducing possible noise. The project was divided into three parts: I – Obtaining the parameters for programming the digital outputs; II – Digital to analog conversion; III – Circuit to switch, amplify and filter the signal. The project aimed to simplify the variation of electronic components, being restricted to the microcontroller, resistors, operational amplifiers and only one capacitor.

Key-Words: Signal generator. Raspberry Pi Pico, Automation, Digital-to-Analog Conversion.

LISTA DE ABREVIATURAS E SIGLAS

DAC	<i>Digital to Analog Converter</i> (Conversor Digital para Analógico).
LCD	<i>Liquid Crystal Display</i> (Display de Cristal Líquido).
PWM	<i>Pulse Width Modulation</i> (Modulação por Largura de Pulso).
PCM	<i>Pulse Code Modulation</i> (Modulação por Código de Pulso).
GPIO	<i>General Purpose Input/Output</i> (Entrada/Saída de Propósito Geral).
GND	<i>Ground</i> (Terra).
PIO	<i>Programmable Input/Output</i> (Entrada/Saída Programável).

LISTA DE FIGURAS

FIGURA 1 - ESTRUTURA DO GERADOR DE SINAIS.....	15
FIGURA 2 - FAMÍLIA RASPBERRY PI PICO.....	16
FIGURA 3 - PINAGEM RASPBERRY PI PICO.....	17
FIGURA 4 - FORMAS DE ONDA DE UM DAC.....	19
FIGURA 5 - MODELO DE QUANTIZAÇÃO.....	20
FIGURA 6 - CARACTERÍSTICA DA QUANTIZAÇÃO MID-TREAD.....	21
FIGURA 7 - CARACTERÍSTICA DA QUANTIZAÇÃO MID-RISE.....	22
FIGURA 8 - MALHA DE RESISTORES R2R.....	23
FIGURA 9 - MALHA DE RESISTORES COM PONDERAÇÃO BINÁRIA.....	23
FIGURA 10 - GRÁFICOS: SENOIDAL, QUADRADO, PWM, DENTE DE SERRA E TRIANGULAR.....	25
FIGURA 11 - GRÁFICOS DOS SINAIS: DISCRETO, QUANTIZADO E ERRO DE QUANTIZAÇÃO.....	29
FIGURA 12 - GRÁFICOS DOS ACIONAMENTOS DE CADA <i>BIT</i> EM RELAÇÃO AO TEMPO PARA O SINAL SENOIDAL.....	35
FIGURA 13 - GRÁFICOS DOS ACIONAMENTOS DE CADA <i>BIT</i> EM RELAÇÃO AO TEMPO PARA O SINAL TRIANGULAR.....	37
FIGURA 14 - GRÁFICOS DOS ACIONAMENTOS DE CADA <i>BIT</i> EM RELAÇÃO AO TEMPO PARA O SINAL DENTE DE SERRA.....	41
FIGURA 15 - CIRCUITO DE CONDICIONAMENTO DE SINAL.....	43
FIGURA 16 - CIRCUITO C3.....	45
FIGURA 17 - CIRCUITO AMPLIFICADOR INVERSOR.....	47
FIGURA 18 - CIRCUITO C2, AMPLIFICADOR DIFERENCIAL.....	48
FIGURA 19 - CIRCUITO AMPLIFICADOR NÃO-INVERSOR.....	48
FIGURA 20 - CIRCUITO AMPLIFICADOR NÃO-INVERSOR.....	50
FIGURA 21 - MONTAGEM GERAL DOS CIRCUITOS.....	52
FIGURA 22 - LOCALIZAÇÃO DOS COMPONENTES NA PROTOBOARD.....	53
FIGURA 23 - SINAL SENOIDAL SEM FILTRO PASSA-BAIXA - 1 PERÍODO.....	54
FIGURA 24 - SINAL SENOIDAL SEM FILTRO PASSA-BAIXA - 2 PERÍODOS.....	54
FIGURA 25 - SINAL SENOIDAL FILTRADO - 1 PERÍODO.....	55
FIGURA 26 - SINAL SENOIDAL COM FILTRADO - 2 PERÍODOS.....	55
FIGURA 27 - SINAL SENOIDAL ANALISADO EM V4 DO CIRCUITO 3.....	56
FIGURA 28 - SINAL SENOIDAL DE 58 AMOSTRAS, SEM FILTRO PASSA-BAIXA - 1 PERÍODO.....	58
FIGURA 29 - SINAL SENOIDAL DE 58 AMOSTRAS E FILTRADO.....	58
FIGURA 30 - SINAL TRIANGULAR GERADO A PARTIR DE 62 AMOSTRAS.....	59
FIGURA 31 - SINAL TRIANGULAR GERADO A PARTIR DE 58 AMOSTRAS.....	59
FIGURA 32 - SINAL DENTE DE SERRA DE 32 AMOSTRAS.....	60
FIGURA 33 - SINAL DENTE DE SERRA DE 31 AMOSTRAS.....	61

SUMÁRIO

1 INTRODUÇÃO	14
2 RASPBERRY PI PICO	16
2.1 FAMÍLIA RASPBERRY PI PICO	16
2.2 CARACTERÍSTICAS	16
2.3 PINAGEM DO RASPBERRY PI PICO	17
3 CONVERSÃO DIGITAL-ANALÓGICO (DAC) - CODIFICAÇÃO	18
3.1 MÉTODO UTILIZADO	18
3.2 QUANTIZAÇÃO LINEAR UNIFORME.....	18
3.2.1 O PROCESSO DE QUANTIZAÇÃO PODE SER DIVIDIDO EM DUAS ATAPAS.....	19
3.2.3 QUANTIZAÇÃO LINEAR E UNIFORME	20
3.2.4 MID-TREAD	20
3.2.5 MID-TREAD	21
3.3 CURVA CARACTERÍSTICA.....	22
4 CONVERSÃO DIGITAL-ANALÓGICO (DAC) – DECODIFICAÇÃO	23
4.1 DECODIFICAÇÃO DOS <i>BITS</i> DO SINAL DIGITAL PARA O SINAL ANALÓGICO.....	23
5 QUANTIFICANDO E CODIFICANDO UM SINAL	24
5.1 FORMAS DE ONDA	24
5.2 SAÍDAS GPIO DO RASPBERRY PI PICO	25
5.3 EQUAÇÃO DE QUANTIZAÇÃO	25
5.4 DEFININDO O NÚMERO DE <i>BITS</i> CODIFICADOS	26
5.5 SINAL SENOIDAL QUANTIZADO EM MID-RISE	27
5.5.1 DEFININDO O NÚMERO DE <i>BITS</i> E OS PARÂMETROS DO CONVERSOR DAC.....	28
5.5.2 QUANTIZANDO O SINAL SENOIDAL PELO PROCESSO MID-RISE	28
5.5.3 SEPARANDO TODOS OS VALORES DE AMPLITUDES QUANTIZADAS	30
5.5.4 CONVERTENDO OS VALORES DECIMAIS EM BINÁRIOS	32
5.5.5 COMPLETANDO OS <i>BITS</i> , COM ZEROS	33
5.5.6 VALORES DE AMPLITUDES E FRAÇÕES DO PERÍODO DO SINAL SENOIDAL	33
5.5.7 FORMATO GRÁFICO DOS ACIONAMENTOS - SINAL SENOIDAL.....	34
6 SINAL DE ONDA TRIANGULAR.....	36
6.1 COPIANDO AS AMPLITUDES CODIFICADAS	36
6.2 CALCULANDO O INTERVALO DE TEMPO DO SINAL TRIANGULAR	36
6.3 VALORES DE AMPLITUDES E FRAÇÕES DO PERÍODO DO SINAL TRIANGULAR	36
6.4 FORMATO GRÁFICO DOS ACIONAMENTOS - SINAL TRIANGULAR	37
7 SINAL DE ONDA DENTE DE SERRA	38
7.1 LISTA DE AMPLITUDES DO SINAL DENTE DE SERRA	38
7.2 CONVERTENDO OS VALORES DECIMAIS EM VALORES BINÁRIOS	38
7.3 COMPLETANDO COM ZEROS, OS <i>BITS</i> RESTANTES DO CÓDIGO BINÁRIO	38
7.4 O INTERVALO DE TEMPO DE AMOSTRAGEM	39
7.5 VALORES DE AMPLITUDES E FRAÇÕES DO PERÍODO DO SINAL DENTE DE SERRA	39
7.6 FORMATO GRÁFICO DOS ACIONAMENTOS - SINAL DENTE DE SERRA.....	39
8 CONDICIONAMENTO DE SINAL	41
8.1 CIRCUITO DE CONDICIONAMENTO DO SINAL	41
8.2 DESCRIÇÃO DO CIRCUITO.....	42
8.3 DIMENSIONAMENTO DO CIRCUITO C3.....	42

8.3.1 MALHA DE RESISTORES	42
8.3.2 AMPLIFICADOR SOMADOR	44
8.4 DIMENSIONAMENTO DO CIRCUITO C1	44
8.5 DIMENSIONAMENTO DO CIRCUITO C2	45
8.6 DIMENSIONAMENTO DO CIRCUITO C4	46
8.7 CIRCUITO C5 - FILTRO PASSA-BAIXA	47
8.8 CONCLUSÃO DO CIRCUITO DE CONDICIONAMENTO DE SINAL	48
9 ANÁLISE DOS SINAIS.....	50
9.1 MONTAGEM DO CIRCUITO.....	50
9.1.1 LOCALIZAÇÃO DOS COMPONENTES	51
9.2 GERANDO O SINAL SENOIDAL.....	51
9.2.1 SINAL SENOIDAL SEM FILTRO PASSA-BAIXA	52
9.2.2 SINAL SENOIDAL COM FILTRO PASSA-BAIXA.....	53
9.2.3 SINAL SENOIDAL EM V4 DO CIRCUITO C3.....	54
9.2.4 MODIFICANDO AS AMOSTRAGENS DO SINAL	54
9.2.5 RESULTADO DO NOVO SINAL SENOIDAL	55
9.3 SINAL TRIANGULAR.....	56
9.4 SINAL DENTE DE SERRA	57
10 CÓDIGO EM MICROPYTHON.....	60
11 CONCLUSÃO.....	62
REFERÊNCIAS	64
APÊNDICE	66
APÊNDICE A	67

1 INTRODUÇÃO

O trabalho de conclusão do curso (TCC) do curso de engenharia de controle e automação no Instituto Federal de Goiás é dividido em duas etapas, TCC I e TCC II.

O Trabalho de Conclusão de Curso I (TCC I) teve por objetivo utilizar uma TV-BOX RK3229 descaracterizada para identificar, na placa, as conexões GPIO e apresentar um exemplo de aplicação prática destas conexões. Todo o trabalho de investigação e exemplo desta aplicação foi relatado no relatório disponibilizado no Apêndice A.

O TCC II teve por objetivo o desenvolvimento de um gerador de sinais de baixo custo, utilizando como plataforma de desenvolvimento a placa Raspberry Pi Pico.

Na natureza, a maioria das grandezas físicas são analógicas, tais como temperatura, pressão etc. Essas grandezas analógicas (sinais contínuos) podem ser convertidas em sinais digitais (sinais amostrados, quantizados e codificados). Um bom exemplo neste caso é o sinal sonoro, que é um sinal mecânico propagado pelo ar e gerado pela vibração de moléculas. As vibrações das moléculas são captadas pelo microfone (transdutor) que as converte em sinais elétricos. Estes sinais elétricos possuem várias formas de ondas, frequências e amplitudes (TOCCI, 2007).

Com o som convertido em sinais elétricos, podemos manipular estes sinais de modo a isolar frequências, eliminar ou inserir ruídos, amplificar ou reduzir a amplitude, aumentar ou diminuir a frequência. Todas estas manipulações podem ser feitas com equipamentos elétricos analógicos. Mas isto demanda uma grande variedade e complexidade de equipamentos analógicos. Uma solução de menor custo é converter estes sinais analógicos em sinais digitais.

Uma vez convertidos em sinais digitais, eles podem ser manipulados e visualizados digitalmente no computador. Mas, esses sinais digitais necessitam ser convertidos novamente em sinais elétricos analógicos para novamente serem convertidos em sinais sonoros, para que possamos ouvir o resultado do processamento. Para isto, utilizamos um conversor digital para analógico (DAC).

O projeto proposto é o de um gerador de sinais baseado na conversão digital para analógico. Com o gerador de sinais, o usuário pode definir a forma de onda, a frequência e a amplitude do sinal.

O projeto é baseado no desenvolvimento de um gerador de sinais controlado pela plataforma Raspberry Pi Pico, na qual os sinais gerados podem ter as seguintes

formas de onda: senoidal, triangular, dente de serra e quadrado. Todos os sinais são de tensão alternada, ou seja, alternam entre amplitude positiva e negativa, onde a tensão máxima de saída é de 5 V pico-a-pico.

O Raspberry Pi Pico, além de ser responsável pela geração das formas de onda, também fornece opções para o usuário alterar a frequência e a amplitude.

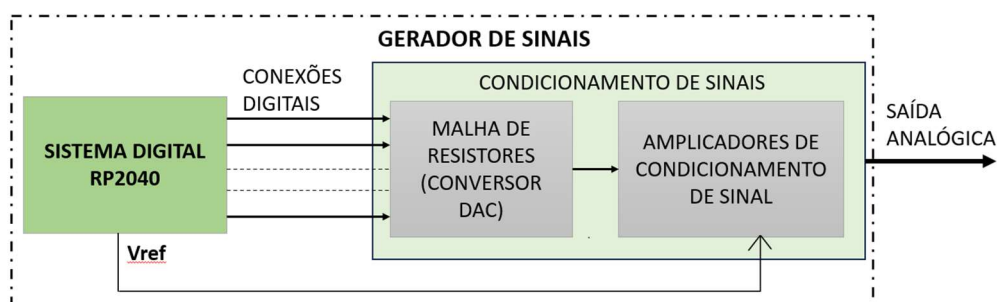
Como este projeto é experimental, o programa em MicroPython foi armazenado no computador que está conectado diretamente ao Raspberry Pi Pico. Os demais ajustes do sinal ficam a cargo de potenciômetros instalados no circuito de condicionamento de sinais.

O Raspberry fornece uma tensão positiva de saída, de 3,3 V. Esta tensão é convertida para uma tensão alternada e amplificada para ± 5 V. Um circuito formado por amplificadores operacionais (LM741) e resistores foi projetado para executar esta operação.

Todas as programações deste projeto foram feitas nas linguagens de programação Python e MicroPython, nas quais os parâmetros de execução do programa no microcontrolador foram calculados e estabelecidos com o uso do Python. E o programa que executa diretamente no microcontrolador foi escrito em MicroPython.

A figura 01 mostra o diagrama da estrutura do conversor DAC, no qual temos o sistema digital que é o Raspberry Pi Pico responsável por fornecer a tensão de referência para o sistema. Este também controla os sinais digitais enviados para a malha de resistores, através das conexões digitais. A malha de resistores com um amplificador somador faz a decodificação dos sinais digitais para um sinal analógico quantizado. O circuito de condicionamento de sinal também converte o sinal analógico somente negativo para um sinal analógico alternado e amplificado.

Figura 01 - Estrutura do gerador de sinais.



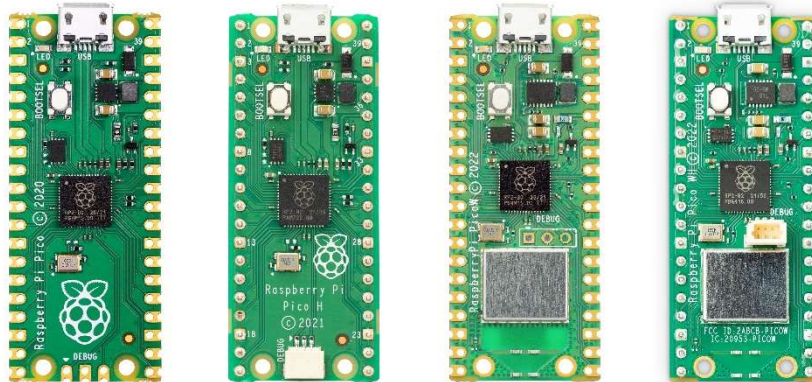
Fonte: autoria própria, 2024.

2 RASPBERRY PI PICO

2.1 Família Raspberry Pi Pico

O Raspberry Pi Pico é uma placa de desenvolvimento com um microcontrolador *Dual-Core* desenvolvido pela Fundação Raspberry Pi. Ele faz parte de uma família de 4 modelos de placas, mostradas na figura 02, da esquerda para direita: Raspberry Pi Pico, Pico H, Pico W e Pico WH. Neste projeto o modelo utilizado é a placa Raspberry Pi Pico (RASPBERRY, 2024).

Figura 02- Família Raspberry Pi Pico.



Fonte: <https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html>, 2024.

2.2 Características

De acordo com a própria fabricante, a família Raspberry Pi Pico possui as seguintes características:

- Chip microcontrolador RP2040, projetado pela Raspberry Pi (Reino Unido);
- Processador Arm Cortex M0+ *dual-core*, *clock* de até 133 MHz;
- 264 kB de SRAM e 2 MB de memória *flash on-board*;
- Interface USB 1.1, com suporte a dispositivo e *host*;
- Modos de suspensão e dormiente de baixo consumo de energia;
- Programação de arrastar e soltar, com armazenamento em massa via USB;
- 26 × pinos GPIO multifuncionais;
- 2 × SPI, 2 × I2C, 2 × UART, 3 × ADC de 12 *bits*, 16 × canais PWM controláveis;
- Relógio e temporizador precisos no *chip*;
- Sensor de temperatura;

3 CONVERSÃO DIGITAL-ANALÓGICO (DAC) - CODIFICAÇÃO

3.1 Método utilizado

A conversão digital-analógica pode ser obtida a partir de vários métodos e várias configurações de circuitos. O método utilizado neste projeto é o da Modulação por Código de Pulso, PCM (*Pulse Code Modulation*) na qual, a partir de amostras quantizadas de níveis de tensão, convertemos estas amostras em valores codificados em sequência de *bits* (OPPENHEIM, 2010, p. 364).

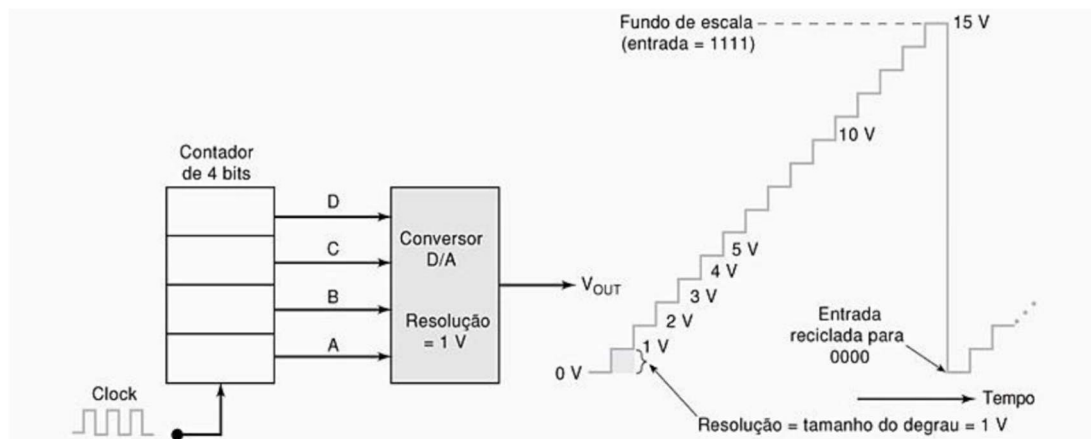
Para cada nível de tensão temos o mesmo código binário. Mas, a sequência deste código e os intervalos de amostragem é que definem a forma da onda do sinal analógico. Por exemplo, em um sinal triangular, os intervalos de amostragem possuem um valor constante. Já em um sinal senoidal, os intervalos são variáveis.

A figura 04 mostra um exemplo de como deve ser a codificação binária de um sinal digital, para converter este sinal em um sinal analógico quantizado. É quantizado porque os valores de tensão não mudam continuamente e, sim, em intervalos de tempo. O sinal gerado se aproxima mais de um sinal analógico à medida que a resolução aumenta. Em Tocci (2007, p. 605), o sinal gerado a partir de um DAC é chamado de “pseudo-analógico” e pode ser chamado de sinal analógico devido às aproximações de um sinal analógico puro.

O sinal digital codificado deve ser enviado para a malha de resistores para que seja feita a decodificação e soma dos sinais, via amplificador operacional somador (TOCCI, 2007, p. 610).

Para obtermos uma codificação digital da forma de onda, utilizamos o método de **quantização linear uniforme**, que faz o mapeamento contínuo das amplitudes de uma forma de onda, em intervalos de tempo.

Figura 03 - Forma de onda de um DAC



Fonte: TOCCI, 2007, p. 606.

3.2 Quantização linear uniforme

A quantização linear uniforme parte do modelo de quantização:

$$x_Q[k] = g(\lfloor f(x[k]) \rfloor) \quad \text{Equação 1}$$

Nela, $g(\cdot)$ e $f(\cdot)$ denotam funções de mapeamento de valor real e $\lfloor \cdot \rfloor$ denota uma operação de arredondamento. E $x[k]$ é um sinal discreto de valor real. Sendo que $f(\cdot)$ pode ser uma função da forma da onda. Como por exemplo, $f(\cdot) = \text{sen}(\cdot)$.

3.2.1 O processo de quantização pode ser dividido em duas etapas

3.2.1.1 Quantização para frente

O mapeamento $f(x[k])$ mapeia o sinal $x[k]$ de modo que ele fique adequado à operação de arredondamento. Isso pode ser um escalonamento do sinal ou um mapeamento não linear. O resultado da operação de arredondamento é um número inteiro $\lfloor f(x[k]) \rfloor \in \mathbb{Z}$ que é denominado *índice de quantização*.

3.2.1.2 Quantização inversa

O mapeamento $g(\cdot)$ mapeia o *índice de quantização* para o valor quantizado $x_Q[k]$, tal que $x_Q[k]$ seja uma aproximação de $x[k]$. Isso pode ser uma operação de escalonamento ou não linear (FLEURY, 2024).

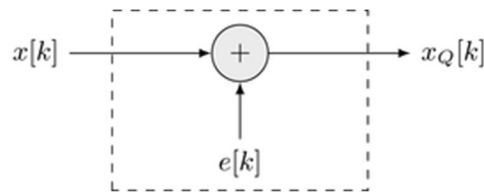
3.2.2 Erro de quantização

O erro de quantização (ou ruído de quantização) $e[k]$ é definido como:

$$e[k] = x_Q[k] - x[k] \quad \text{Equação 2}$$

O processo de quantização pode ser modelado (figura 5) adicionando-se o erro de quantização ao sinal discreto:

Figura 04 - Modelo de quantização.



Fonte: FLEURY, 2024.

3.2.3 Quantização linear e uniforme

Diz-se quantização linear quando as funções de mapeamento $g(\cdot)$ e $f(\cdot)$, são lineares. E quantização uniforme quando L ($L = 2^n$), divide a Faixa de Valores de Amplitude (A_{fv}) do sinal de entrada, em $2^n - 1$ passos de quantização (q_k), ou resolução da quantização (TOCCI, 2007, p. 606).

$$q_k = \frac{A_{fv}}{2^n - 1} \quad \text{Equação 3}$$

Os quantificadores podem ser descritos por sua característica de entrada/saída não linear:

$$x_Q[k] = Q\{x[k]\} \quad \text{Equação 4}$$

Nessa equação, $Q\{\cdot\}$ indica o processo de quantização. Na quantização linear uniforme é comum diferenciar entre duas curvas características, as chamadas *mid-tread* (meio passo) e *mid-rise* (meia subida) (FLEURY, 2024).

3.2.4 Mid-Tread

O termo *mid-tread* se deve ao fato de que valores menores $|x[k]| < \frac{Q}{2}$ são mapeados para zero. Tem relação entrada/saída do tipo:

$$x_Q[k] = q_k \cdot \left\lfloor \frac{x[k]}{q_k} + \frac{1}{2} \right\rfloor \quad \text{Equação 5}$$

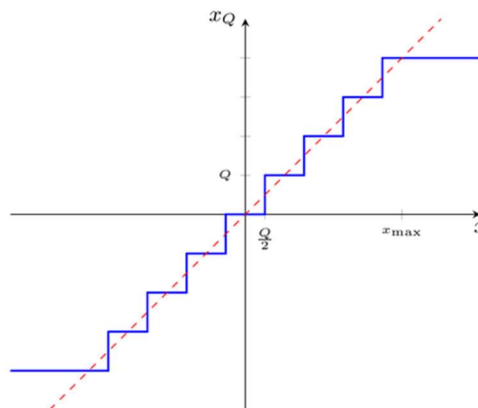
Nessa equação:

- q_k é o *quantum* (tamanho do passo de quantização ou tensão por degrau).
- $\lfloor \cdot \rfloor$ é a função *floor* (retorna o maior número inteiro menor ou igual a um valor dado) que mapeia um número real para o maior número inteiro não maior que seu argumento.

Sem restringir $x[k]$ em amplitude, os índices de quantização resultantes formam um conjunto infinito contável de valores. Para um número finito de índices de quantização, o sinal de entrada deve ser limitado a uma faixa de amplitudes mínima e máxima: $x_{min} < x[k] < x_{max}$ antes da quantização (FLEURY, 2024).

A característica de quantização resultante de um quantizador uniforme linear *mid-tread* é mostrada a seguir.

Figura 05 - Característica da quantização Mid-Tread.



Fonte: (FLEURY, 2024).

3.2.5 Mid-Rise

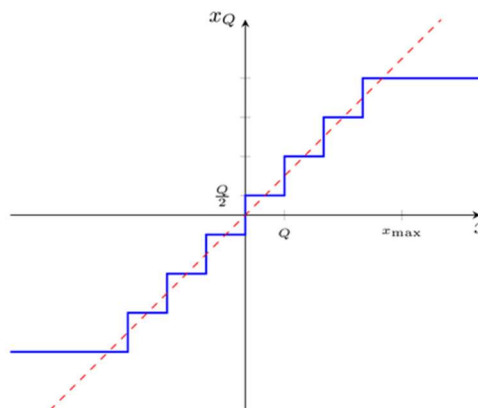
O termo *mid-rise* se deve ao fato de que $x[k] = 0$ não é mapeado para zero. Pequenos valores positivos/negativos em torno de zero são mapeados para $\pm \frac{Q}{2}$ (FLEURY, 2024).

Mid-rise tem relação entrada/saída do tipo:

$$x_Q[k] = q_k \cdot \left(\left\lfloor \frac{x[k]}{q_k} \right\rfloor + \frac{1}{2} \right) \quad \text{Equação 6 (FLEURY, 2024)}$$

A característica de quantização resultante de um quantizador uniforme linear *mid-rise* é mostrada a seguir.

Figura 06 - Característica da quantização Mid-Rise.



Fonte: FLEURY, 2024.

3.3 Curva Característica

Ao observarmos os gráficos das curvas características de *Mid-Tread* e *Mid-Rise*, o gráfico da figura 04 e sabendo que a amplitude A_{fv} , deve ser codificada em 2^n bits e sabendo que o gerador de sinal deve gerar um sinal alternado com amplitudes simétricas, podemos concluir que, para codificarmos os níveis de tensão em termos de códigos binários, o melhor é utilizarmos o método de quantização por curva característica *Mid-Rise*.

4 CONVERSÃO DIGITAL-ANALÓGICO (DAC) – DECODIFICAÇÃO

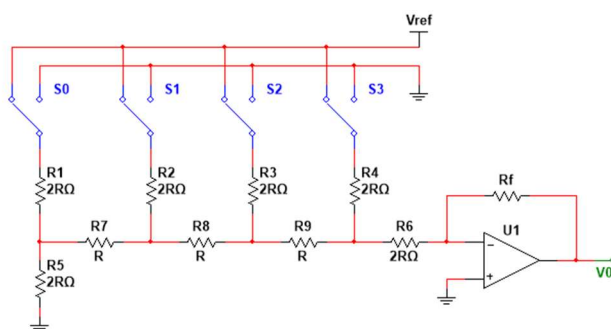
4.1 Decodificação dos *bits* do sinal digital para o sinal analógico

O Raspberry Pi Pico é responsável por controlar e enviar os códigos binários para a malha de resistores decodificadores.

Existem várias configurações de resistores junto ao amplificador somador, para que se possa fazer a decodificação binária. Abaixo temos dois modelos de configuração de 4 *bits*. Na figura 08, temos a configuração R2R e, na figura 9, a configuração de resistores com ponderação binária. Para ambos os casos, temos apenas uma tensão de referência (TOCCI, 2007).

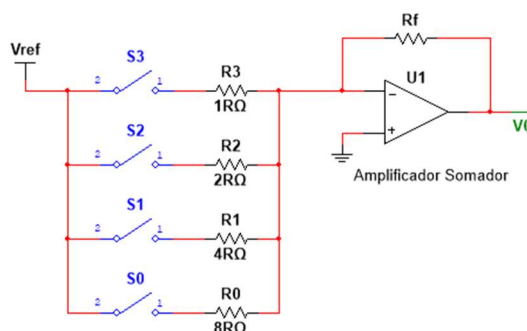
Apenas observando os diagramas elétricos, vemos que o diagrama R2R possui mais que o dobro de resistores que o segundo diagrama. Levando apenas isto em conta, podemos concluir que o segundo diagrama possui menos resistores para terem suas precisões ajustadas. Por isso, a configuração de resistores com ponderação binária, vai ser utilizada para a decodificação dos sinais digitais.

Figura 07 - Malha de resistores R2R.



Fonte: autoria própria, 2024.

Figura 08 - Malha de resistores com ponderação binária.



Fonte: autoria própria, 2024.

5 QUANTIFICANDO E CODIFICANDO UM SINAL

5.1 Formas de onda

Um gerador de sinais pode gerar uma ou mais formas de onda. Com base no código escrito em Python por FLEURY (2024), fiz algumas modificações para mostrar, graficamente, com a mesma frequência e a mesma amplitude, cinco formas de onda mais conhecidas. São elas:

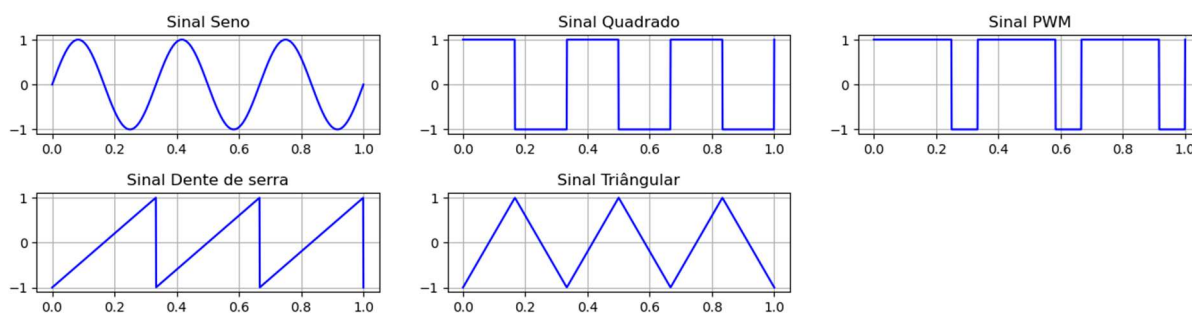
- **Senoidal:** Gerada a partir de uma função Seno;
- **Quadrado:** Gerada a partir da função *square*;
- **PWM:** *Duty cycle* variável (tempo de sinal ligado em máximo volt);
- **Dente de Serra:** Possui rampa ascendente durante todo o período. É gerada pela função *sawtooth*.
- **Triangular:** Também é gerada pela função *sawtooth*. Possui rampa ascendente e descendente, cada uma durando meio período.

O código Python gera os gráficos da figura 10.

```
1 from numpy import linspace, pi, sin
2 from scipy.signal import sawtooth, square
3 from matplotlib.pyplot import plot, figure, grid, subplots, subplot, title, plt
4 from pylab import figure, plot, xlabel, axis, legend, grid
5 # -----
6 t = linspace(0,1,1001); freq = 3.                                # Amostragem e frequência
7 # -----
8 seno = sin(2*pi*freq*t)                                          # Seno
9 # -----
10 quad = square(2*pi*freq*t)                                      # Quadrado
11 # -----
12 dutycycle = 0.75; pwm = square(2*pi*freq*t, dutycycle) # PWM
13 # -----
14 ds = sawtooth(2*pi*freq*t, 1)                                    # Dente de serra
15 # -----
16 tr = sawtooth(2*pi*freq*t,0.5)                                  # Triângular
17 # -----
18 figure(); subplots(figsize=(14,3))
19 subplot(231); plot(t,seno, color='blue'); title('Sinal Seno');   grid()
20 subplot(232); plot(t,quad, color='blue'); title('Sinal Quadrado'); grid()
21 subplot(233); plot(t,pwm, color='blue'); title('Sinal PWM');    grid()
22 subplot(234); plot(t,ds, color='blue'); title('Sinal Dente de serra'); grid()
23 subplot(235); plot(t,tr, color='blue'); title('Sinal Triângular'); grid()
24 plt.subplots_adjust(left=0.125, bottom=0.1, right=1.0, top=1.0, wspace=0.2, hspace=0.6)
```

Fonte: FLEURY, 2024.

Figura 9 - Gráficos: senoidal, quadrado, PWM, dente de serra e triangular.



Fonte: autoria própria, 2024.

5.2 Saídas GPIO do Raspberry Pi Pico

O Raspberry Pi Pico possui saídas de sinais PWM (*Pulse Width Modulation*, ou Modulação por Largura de Pulso) com a possibilidade de ajustes de frequência e ciclo de trabalho (*duty-cycle*). A tensão máxima (tempo ligado) dessas saídas é de 3,3 V. E a tensão mínima (tempo desligado) é de 0 V. Ou seja, uma tensão de pico a pico positiva (RASPBerry, 2024).

Ao fazermos um ajuste de *duty-cycle* de 50% no sinal PWM, obteremos um sinal de onda quadrada. Também podemos obter um sinal de onda quadrada, ligando e desligando uma saída GPIO, em que o tempo ligado e o tempo desligado equivalem, respectivamente, à metade do período do sinal.

Para o sinal PWM e de onda quadrada, podemos utilizar apenas uma saída GPIO. Mas, para as demais formas de onda, precisamos utilizar mais saídas GPIO, para codificar o sinal com mais de um *bit*, através das ações de liga/desliga de cada GPIO. Para determinar o momento em que cada saída GPIO deve ser ligada ou desligada foi desenvolvido um programa em MicroPython.

O método de quantização *Mid-Rise* gera combinações nas saídas GPIO, representando um número binário que corresponde a um quantum da amplitude do sinal. O valor de cada quantum permanece constante por um intervalo de tempo que é uma fração do período do sinal. Para sinais como dente de serra e triangular, este intervalo é fixo, enquanto para sinais senoidais ele varia ao longo do tempo. Essa variação no tempo de cada quantum é fundamental para representar a forma de onda senoidal de forma precisa. A Figura 11 ilustra esse conceito de forma mais detalhada, mostrando como as combinações GPIO evoluem ao longo do tempo para representar um sinal senoidal quantizado.

5.3 Equação de quantização

Como vimos anteriormente, o sinal quantizado é obtido através da Equação 6:

$$x_Q[k] = q \cdot \left(\left\lfloor \frac{x[k]}{q} \right\rfloor + \frac{1}{2} \right) \quad \text{Equação 6}$$

Desta forma obtém-se quantização das amplitudes positivas e negativas. Mas, o Raspberry Pi Pico fornece apenas uma amplitude variável positiva. A mudança para um sinal de saída alternado, do gerador de sinais fica a cargo do circuito de amplificadores de condicionamento de sinal.

Portanto, devemos codificar as amplitudes em termos de sinais positivos. Para isto, apenas adicionamos um deslocamento D vertical positivo do sinal, igual à amplitude do sinal, resultando na equação:

$$x_Q[k] = q \cdot \left(\left\lfloor \frac{x[k]}{q} \right\rfloor + \frac{1}{2} + D \right) \quad \text{Equação 7}$$

5.4 Definindo o número de *bits* codificados

Como vimos nos gráficos de formas de onda, os formatos quadrados e PWM, necessitam de apenas um *bit* para serem representados, sendo que este *bit* assume dois estados $0 = \text{duty cycle Off}$ e $1 = \text{duty cycle On}$. Assim, para um período completo, basta 1 *bit*. Sendo a quantidade de passos igual a amplitude do sinal:

$$q_k = \frac{A_{fv}}{2^{n-1}} \Rightarrow \frac{A_{fv}}{2^{1-1}} = A_{fv} \quad \text{Equação 8}$$

É importante lembrar que a amplitude do sinal é igual à tensão de pico do sinal $A_{fv} = V_p$.

O sinal dente de serra assume gradualmente todos os *bits* disponíveis durante a subida do sinal e depois reinicia a partir do ponto inicial. Sendo assim, para um período completo, repetem-se todos os *bits* apenas uma vez. A quantidade de passos calculada pela equação 3 (TOCCI, 2007, p. 606):

$$q_k = \frac{A_{fv}}{2^{n-1}} \quad \text{Equação 3}$$

Para as formas de onda senoidal e triangular, um período completo do sinal é necessário para que os *bits* sejam repetidos duas vezes, uma na subida e outra na descida. Assim, a quantidade de passos é:

$$q_k = \frac{A_{fv}}{2(2^n-1)} \quad \text{Equação 9}$$

Por sua vez, o número total de degraus para um período da onda senoidal e da onda triangular é:

$$D_{qtd} = 2(2^n - 1) \quad \text{Equação 10.}$$

5.5 Sinal senoidal quantizado em *Mid-Rise*

Neste trabalho, a primeira forma de onda a ter as amplitudes quantizadas codificadas para os valores binários, será a forma de onda senoidal. Para este processo, iremos seguir os seguintes passos:

1. Definir o número de *bits* do conversor DAC e definir os parâmetros do sinal senoidal, além da própria geração do sinal.
 - 1.1. Determinar a amplitude de pico, do sinal: A.
 - 1.2. Determinar o número de *bits* para a amplitude de pico a pico: n.
 - 1.3. Determinar o quantum (passo da quantização): q.
 - 1.4. Determinar a quantidade de amostras: N.
2. Quantizar uma onda senoidal através do processo *Mid-Rise*, deslocando todas as amplitudes do sinal, para valores positivos.
3. Separar todos os valores de amplitudes quantizadas, resultando em uma lista de tamanho conforme o número de *bits* definido para o conversor digital-analógico.
4. Converter os valores decimais fracionados em números inteiros, para depois converter em números binários.
5. Converter os valores decimais em binários.
6. Completar com zeros, os *bits* restantes para completar o número binário.

Exemplo:

Número máximo de *bits*: $4 = 0000$.

Número decimal representado em *bits*: $3_{10} = 0011_2$.

Como deve ficar: $2_{10} = 0010_2$.

7. Valores de amplitudes e frações do período.
8. Formato gráfico do acionamento de *cada bit do* conversor DAC.

5.5.1 Definindo o número de *bits* e os parâmetros do conversor DAC

O número de *bits* do conversor DAC será de **$n = 5$ bits**. Logo, a quantidade de degraus é definida pela equação 10:

$$D_{qtd} = 2(2^5 - 1) \Rightarrow D_{qtd} = 62$$

Sendo a amplitude de pico a pico igual a $2^5 - 1 = 31$ degraus, podemos calcular a amplitude de pico **A** como:

$$A = \frac{2^n - 1}{2} \quad \text{Equação 11}$$

Calculando a amplitude de pico pela equação 11:

$$A = \frac{2^5 - 1}{2} = 15,5$$

Calculando a quantização de pico a pico:

$$q = \frac{2A + 1}{2^n} = \frac{2 \cdot 15,5 + 1}{2^5} = 1$$

A quantidade de amostras (**N**) definida para o período do sinal é de 2000 amostras por ser um valor inteiro e por nos fornecer uma boa resolução para quantização.

5.5.2 Quantização do sinal senoidal pelo processo *Mid-Rise*

Após a definição dos parâmetros para a geração do sinal e para a quantização, executamos o processo de quantização pelo processo *Mid-Rise*, através do programa abaixo. Este mesmo programa também gera os gráficos de um sinal analógico ($x[k]$), de erro de quantização ($e[k]$) e o sinal quantizado ($x_Q[k]$), mostrados na figura 11 (FLEURY, 2024).

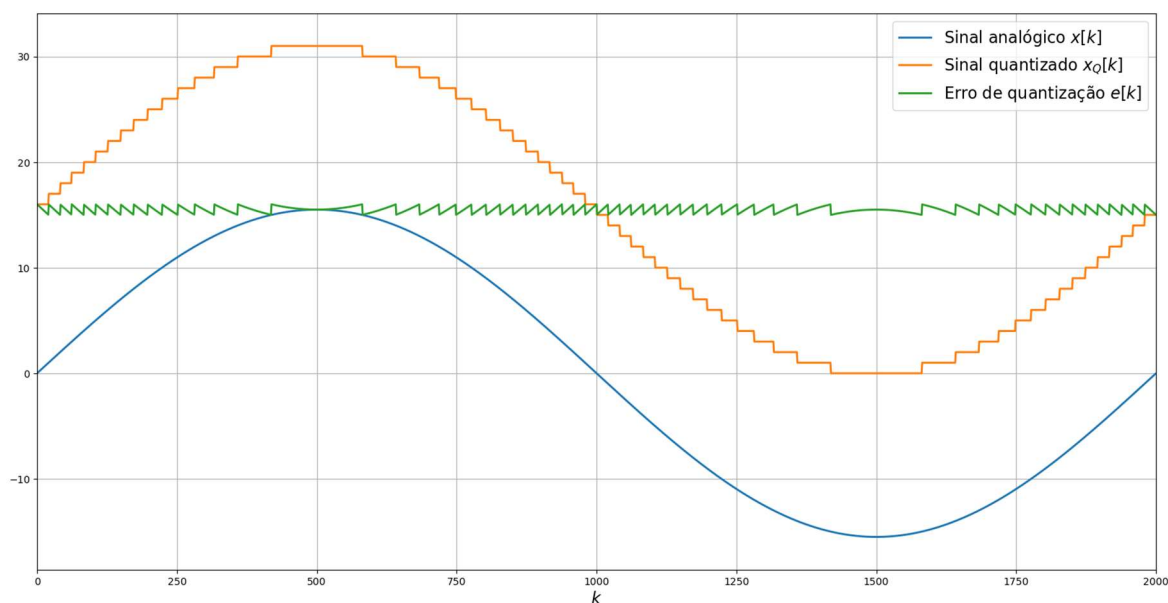
```

1 import matplotlib.pyplot as plt
2 from numpy import arange, sin, pi, copy, where, sign, floor
3 from pylab import figure, plot, xlabel, axis, legend, grid, show, subplots, subplot
4
5 def quantizacao_uniforme_midtrise(x,q):
6     x = copy(x) # Vetor x aloca novas posições de memória
7     indices = where(abs(x) >= 35) # Limita amplitudes (valores de 'x' entre -35 e 35)
8     x[indices] = sign(x[indices]) # Atribui aos valores localizados amplitudes -1 ou 1
9     xQ = q*(floor(x/q)+0.5)+15.5 # Quantização linear com deslocamento vertical inferior
10    return xQ
11
12 def plota_sinais(x,amplit,qtam,xQ):
13     e = xQ - x # Sinal erro de quantização
14     figure(figsize=(20,10))
15     plot(x, label=r'sinal discreto $x[k]$', linewidth=2)
16     plot(xQ, label=r'sinal quantizado $x_Q[k]$', linewidth=2)
17     plot(e, label=u'erro de quantização $e[k]$', linewidth=2)
18     xlabel(r'$k$', fontsize=16); axis([0, qtam, -0.6*amplit, 1.1*amplit]);
19     legend(fontsize=16); grid()
20     return
21
22 A = 15.5 # Amplitude de pico do sinal
23 nBit = 5 # Número de bits para a amplitude
24 q = 1 # Quantum (passo da quantização)
25 N = 2000 # Quantidade de amostras
26 x = A * sin(2*pi/N * arange(N)) # Gerando o sinal senoidal discreto
27 xQ = quantizacao_uniforme_midtrise(x,q) # Quantiza o sinal 'x'
28 plota_sinais(x,2*A,N,xQ) # Amplitude multiplicada por 2 apenas para plotagem

```

Fonte: FLEURY, 2024.

Figura 10 - Gráficos dos sinais: analógico, quantizado e erro de quantização.



Fonte: autoria própria, 2024.

5.5.2.1 Amplitudes máxima e mínima calculadas

Após a quantização, temos as amplitudes máximas e mínimas:

```
print(f'Amplitude máxima: {max(xQ)}')
print(f'Amplitude mínima: {min(xQ)}')
>> Amplitude máxima: 31.0
>> Amplitude mínima: 0.0
```

Fonte: autoria própria, 2024.

5.5.3 Separando todos os valores de amplitudes quantizadas

No programa anterior, os valores de amplitudes foram calculados e armazenados no objeto **xQ**. A quantidade total de amostras ficou armazenada em **N**.

Este programa deve armazenar em **ampNum** os valores das amplitudes não repetidas e armazenar em **tempSeno**, em termos de porcentagem da amostragem total, a posição da amostragem atual relativa anterior.

$$temSeno = \frac{t_{atual}}{TotalAmostra} - \frac{t_{anterior}}{TotalAmostra} = \frac{\Delta t}{N_{amostra}} \quad \text{Equação 12}$$

No Raspberry Pi Pico os valores de **tempSeno** devem multiplicar o período (**T**) do sinal, para indicar quanto tempo o sinal deve ficar ativado. Ou seja, quanto tempo cada saída GPIO deve ficar em nível lógico 1, enviando sinal para o conversor digital-analógico.

5.5.3.1 Calculando os valores de **ampNum** e **tempSeno**

Em **ampNum** e **tempSeno**, são armazenadas 2000 amostras em cada variável. Há amostras repetidas dos valores de amplitudes. E assim deve ser quando se analisa estas amostras ao longo de todo o período. Mas, quando se analisa a rampa de subida, os valores não podem se repetir. E assim também é para a rampa de descida. Portanto, devemos isolar os valores repetidos em cada rampa do sinal. E, assim, temos o programa abaixo que isola os valores repetidos a cada $1/4$ do sinal.

Desse modo, as variáveis **ampNum** e **tempSeno** que possuíam 2000 amostras cada, agora passam a ter as 62 amostras necessárias para reproduzir o sinal.

```

1 '''
2     Obs: As linhas que contêm o comando print(f'{cont}: {c} => {xQ[c]}'),
3     imprimem a amplitude e a posição da amostragem
4 '''
5 Dv = 0.0          # Deslocamento vertical
6 ampNum = []       # Armazena os valores de amplitudes
7 tempSeno = []     # Armazena os intervalos de tempo atual, relativos anterior
8 t1 = 0           # Variável temporária
9 t2 = []          # Armazena valores fracionados do eixo X
10 cont = 0         # Variável temporária
11 #print('Indice: Amostragem => Amplitude no ponto de amostragem:\n')
12 # -----
13 for c in range(len(xQ)):
14     if c < int((N/4)+1):
15         if xQ[c] > Dv and c != int(N/4) and xQ[c] > xQ[c-1]:
16             ampNum.append(xQ[c])
17             if c/N > 0:
18                 tempSeno.append(round(c/N - t1, 6))
19                 t2.append(round(c/N, 6))
20                 t1 = c/N
21             cont = cont + 1
22             #print(f'{cont}: {c} => {xQ[c]}')
23             Dv = xQ[c]
24     if c > int((N/4)+1) and c < int((3*N/4)+1):
25         if xQ[c] < Dv:
26             ampNum.append(xQ[c])
27             tempSeno.append(round(c/N - t1, 6))
28             t2.append(round(c/N, 6))
29             t1 = c/N
30             cont = cont + 1
31             #print(f'{cont}: {c} => {xQ[c]}')
32             Dv = xQ[c]
33     if c > int((3*N/4)+1) and c < int(N):
34         if xQ[c] > Dv:
35             ampNum.append(xQ[c])
36             tempSeno.append(round(c/N - t1, 6))
37             t2.append(round(c/N, 6))
38             t1 = c/N
39             cont = cont + 1
40             #print(f'{cont}: {c} => {xQ[c]}')
41             Dv = xQ[c]
42     if c == int(N-1):
43         tempSeno.append(round(1 - t1, 6))
44         t2.append(round(c/N, 6))

```

Fonte: autoria própria, 2024.

5.5.3.2 Convertendo as amostras de amplitudes para valores inteiros

As amostras obtidas dos valores de amplitudes são amostras do tipo *float* (flutuante). Portanto, devemos converter estas amostras para o tipo *int* (inteiro). Assim, o programa abaixo executa esta operação. Estas amostras são armazenadas na variável **ampNum**.

```

1 for c in range(len(ampNum)):
2     ampNum[c] = int(round(ampNum[c], 0))
3
4 print(f'Quantidade de amostras armazenadas em ampNum: {len(ampNum)}\n')
5 print(ampNum[:10])                # Os 10 primeiros valores de ampNum
6
7 >> Quantidade de amostras armazenadas em ampNum: 62
8 >> [16, 17, 18, 19, 20, 21, 22, 23, 24, 25]

```

Fonte: autoria própria, 2024.

5.5.3.3 Amostras do período

Já para o período, temos as amostras relativas a cada amplitude do sinal, armazenadas em termos de fração do período, na variável **tempSeno**.

```

1 print(f'Quantidade de amostras armazenadas em tempSeno: {len(tempSeno)}\n')
2 print(tempSeno[:10])                # Os 10 primeiros valores de tempNum
3
4 >> Quantidade de amostras armazenadas em tempSeno: 62
5 >> [0.0105, 0.0105, 0.01, 0.011, 0.0105, 0.011, 0.0115, 0.0115, 0.0125, 0.013]

```

Fonte: autoria própria, 2024.

5.5.3.4 Máxima frequência do sinal senoidal

O intervalo mínimo de tempo, entre uma amostra e outra deve ser de $100\mu s$. Para um período com 62 amostras, temos um período: $T = 100 \cdot 62 = 6,2ms$. Como a frequência é o inverso do período ($f = \frac{1}{T}$), então, temos a frequência máxima ($f_{m\acute{a}x}$) para 62 amostras iguais: $f = \frac{1}{6,2 \times 10^{-3}} = 161Hz$.

Como visto em 5.5.3.3, o sinal senoidal possui valores diferentes para intervalos de tempo entre uma amostra e outra. Sendo assim, podemos determinar a máxima frequência para este sinal, a partir da menor amostra de intervalo de tempo. Podemos verificar isto a partir da função *min*.

```

1 min(tempSeno)
2 >> 0.01

```

Fonte: autoria própria, 2024.

Vemos que a menor fração do período é de: $0,01T = 100 \times 10^{-6}s$. Logo, para frequência do sinal senoidal temos: $f_{m\acute{a}x} = \frac{0,01}{100 \times 10^{-6}} = 100Hz$.

Esta é a maior frequência que o sinal senoidal emitido a partir de um DAC de 5 *bits*, pode gerar sem que haja distorção significativa na amostragem, na qual não se

faça uso de um filtro passa-baixo.

5.5.4 Convertendo os valores decimais em binários

Este programa converte os valores decimais das amplitudes (*ampNum*), em valores binários (*nBinSeno*).

```

1  nBinSeno = []                                # Valores binários
2  for c in range(len(ampNum)):
3      if ampNum[c] == 0.0:
4          t = '0'
5      else:
6          # Para os demais valores, deve ser removido o termo "-0b"
7          t = bin(int(ampNum[c])).lstrip('-0b') # Converte em binário e remove "-0b"
8      nBinSeno.append(t)
9  # Amostra dos 8 primeiros valores binários
10 print(nBinSeno[:8])
10 >> ['10000', '10001', '10010', '10011', '10100', '10101', '10110', '10111']

```

Fonte: autoria própria, 2024.

5.5.5 Completando os *bits*, com zeros

Ao convertermos os valores decimais em binários, valores decimais menores que 16 são representados por menos de 5 *bits*. Portanto, devemos completar os demais dígitos binários que faltam à esquerda do valor.

Assim, este programa converte cada *string* gerada anteriormente, em uma lista de dimensão de 5 *bits*, onde cada *string* passa a ser um número inteiro (*int*). Depois completa com zeros à esquerda cada lista que possuir dimensão menor que 5 *bits*.

```

1  for c in range(len(nBinSeno)):
2      while len(nBinSeno[c]) < nBit:
3          nBinSeno[c] = '0' + nBinSeno[c]
4          nBinSeno[c] = list(nBinSeno[c])
5          for i in range(len(nBinSeno[c])):
6              nBinSeno[c][i] = int(nBinSeno[c][i])
7
8  # Amostra das listas com os valores binários inteiro
9  print(nBinSeno[:4])
10 >> [[1, 0, 0, 0, 0], [1, 0, 0, 0, 1], [1, 0, 0, 1, 0], [1, 0, 0, 1, 1]]

```

Fonte: autoria própria, 2024.

5.5.6 Valores de amplitudes e frações do período do sinal senoidal

A lista *nBinSeno* é a lista que contém sublistas com os valores 0 e 1 que irão indicar o acionamento (*HIGH/LOW*) de cada saída GPIO do conversor DAC.

A lista **tempSeno** representa as frações do período do sinal, no qual cada valor deve multiplicar o período total (**T**) para indicar tempos de valores inteiros, quanto tempo a saída GPIO deve ficar em *HIGH* ou *LOW*, sendo que o tempo de acionamento é medido em microsegundos (μs).

Lista de amplitudes **nBinSeno**:

```
[1,0,0,0,0], [1,0,0,0,1], [1,0,0,1,0], [1,0,0,1,1], [1,0,1,0,0], [1,0,1,0,1], [1,0,1,1,0],
[1,0,1,1,1], [1,1,0,0,0], [1,1,0,0,1], [1,1,0,1,0], [1,1,0,1,1], [1,1,1,0,0], [1,1,1,0,1],
[1,1,1,1,0], [1,1,1,1,1], [1,1,1,1,0], [1,1,1,0,1], [1,1,1,0,0], [1,1,0,1,1], [1,1,0,1,0],
[1,1,0,0,1], [1,1,0,0,0], [1,0,1,1,1], [1,0,1,1,0], [1,0,1,0,1], [1,0,1,0,0], [1,0,0,1,1],
[1,0,0,1,0], [1,0,0,0,1], [1,0,0,0,0], [0,1,1,1,1], [0,1,1,1,0], [0,1,1,0,1], [0,1,1,0,0],
[0,1,0,1,1], [0,1,0,1,0], [0,1,0,0,1], [0,1,0,0,0], [0,0,1,1,1], [0,0,1,1,0], [0,0,1,0,1],
[0,0,1,0,0], [0,0,0,1,1], [0,0,0,1,0], [0,0,0,0,1], [0,0,0,0,0], [0,0,0,0,1], [0,0,0,1,0],
[0,0,0,1,1], [0,0,1,0,0], [0,0,1,0,1], [0,0,1,1,0], [0,0,1,1,1], [0,1,0,0,0], [0,1,0,0,1],
[0,1,0,1,0], [0,1,0,1,1], [0,1,1,0,0], [0,1,1,0,1], [0,1,1,1,0], [0,1,1,1,1]]
```

Fonte: autoria própria, 2024.

Lista de frações do período **tempSeno**:

```
[0.0105, 0.0105, 0.01, 0.011, 0.0105, 0.011, 0.0115, 0.0115, 0.0125, 0.013, 0.014, 0.015,
0.0175, 0.021, 0.03, 0.0815, 0.03, 0.021, 0.0175, 0.015, 0.014, 0.013, 0.0125, 0.0115, 0.0115,
0.011, 0.0105, 0.011, 0.01, 0.0105, 0.0105, 0.01, 0.0105, 0.01, 0.011, 0.0105, 0.011, 0.0115,
0.0115, 0.0125, 0.013, 0.014, 0.015, 0.0175, 0.021, 0.03, 0.0815, 0.03, 0.021, 0.0175, 0.015,
0.014, 0.013, 0.0125, 0.0115, 0.0115, 0.011, 0.0105, 0.011, 0.01, 0.0105, 0.01]
```

Fonte: autoria própria, 2024.

5.5.7 Formato gráfico dos acionamentos - Sinal Senoidal

Se analisarmos graficamente os acionamentos (*HIGH/LOW*) de cada *bit* ao gerar um sinal senoidal, temos o resultado gerado e mostrado pelo programa abaixo.

```
1 xn = [] # Lista com as listas de estados para cada bit
2 N = 2000 # Número de amostras
3 for c in range(len(nBinSeno[0])):
4     xn.append([])
5 # -----
6 for c in range(len(nBinSeno)):
7     n = int(tempSeno[c] * N)
8     for i in range(len(nBinSeno[c])):
9         if nBinSeno[c][i] == 0:
10             for j in range(n):
11                 xn[i].append(0.)
12         else:
13             for k in range(n):
14                 xn[i].append(1.)
15 # -----
16 for c in range(len(xn)):
17     for i in range(len(xn[c])):
18         xn[c][i] = xn[c][i] + c + 0.05
```

```

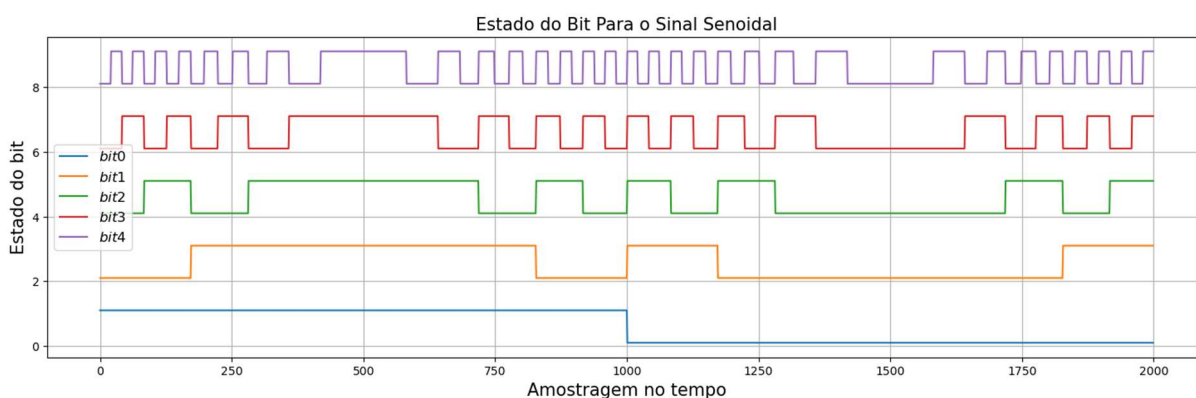
19     for i in range(len(xn[c])):
20         xn[c][i] = xn[c][i] + c + 0.05
21 # -----
22 z1 = arange(len(xn[0]))
23 len(z1)
24 # -----
25 subplots(figsize=(18,5))
26 plot(z1, xn[0], z1, xn[1], z1, xn[2], z1, xn[3], z1, xn[4])
27 plt.title('Estado do Bit Para o Sinal Senoidal', fontsize = 15)
28 plt.xlabel('Amostragem no tempo', fontsize = 15);
29 plt.ylabel('Estado do bit', fontsize = 15)
30 legend(['$bit 0$', '$bit 1$', '$bit 2$', '$bit 3$', '$bit 4$'], fontsize = 12)
31 grid(True); show()

```

Fonte: autoria própria, 2024.

A figura 12 mostra os gráficos de tempo *HIGH* e *LOW* das GPIOs que formam os números binários correspondentes a cada salto na amplitude do sinal senoidal. Sendo *bit0* o *bit* menos significativo de peso 1 e o *bit4* o *bit* mais significativo de peso 16.

Figura 11 - Gráficos dos acionamentos de cada *bit* em relação ao tempo para o sinal senoidal.



Fonte: autoria própria, 2024.

6 SINAL DE ONDA TRIANGULAR

Vimos no capítulo 5 as séries binárias e temporais para gerarmos o sinal senoidal. Também vimos que a quantidade de amostras é a mesma que a quantidade de amostras para gerarmos o sinal triangular. A diferença está nos valores das frações do período, sendo que para o sinal triangular, estas amostras possuem um valor constante. É este valor que vamos calcular e armazenar na variável **tempTr**.

Os valores das amplitudes armazenados em **nBinSeno** são copiados diretamente para **nBinTr**.

6.1 Copiando as amplitudes codificadas

```
# Amplitudes do sinal triangular assumem valores do sinal seno
nBinTr = nBinSeno
```

Fonte: autoria própria, 2024.

6.2 Calculando o intervalo de tempo do sinal triangular

O intervalo de tempo é:

$$Intervalo = \frac{1}{2(2^n - 1)} = \frac{1}{2(2^5 - 1)} = 0,0161$$

Equação 13.

Observamos que a quantidade de amostras em **nBinTr** é a mesma que em **tempTr**.

```
1 tempTr = []
2 for c in range(len(nBinTr)):
3     tempTr.append(round(1/len(nBinTr), 4))
4 print(tempTr[:10])           # Os 10 primeiros intervalos
5
6 >> [0.0161, 0.0161, 0.0161, 0.0161, 0.0161, 0.0161, 0.0161, 0.0161, 0.0161, 0.0161]
```

Fonte: autoria própria, 2024.

6.3 Valores de amplitudes e frações do período do sinal triangular

Lista dos 6 primeiros valores das amplitudes **nBinTr** e dos 6 primeiros intervalos tempo **tempTr**:

```
1 print(f'Amplitudes do sinal triangular: \n{nBinTr[:6]}')
2 print('\n')
```

```

3 print(f'Tempo em porcentagem entre o um sinal de amplitude e outro: \n{tempTr[:6]}')
4
5 >> Amplitudes do sinal triangular:
6 >> [[1,0,0,0,0], [1,0,0,0,1], [1,0,0,1,0], [1,0,0,1,1], [1,0,1,0,0], [1,0,1,0,1]]
7 >>
8 >> Tempo em porcentagem entre o um sinal de amplitude e outro:
9 >> [0.0161, 0.0161, 0.0161, 0.0161, 0.0161, 0.0161]

```

Fonte: autoria própria, 2024.

6.4 Formato gráfico dos acionamentos - Sinal triangular

Formato gráfico dos acionamentos de cada *bit* do conversor DAC, para o sinal triangular.

```

1 xn = [] # Lista com as listas de estados para cada bit
2 N = 2000 # Número de amostras
3 for c in range(len(nBinTr[0])):
4     xn.append([])
5 # -----
6 for c in range(len(nBinTr)):
7     n = int(tempTr[c] * N)
8     for i in range(len(nBinTr[c])):
9         if nBinTr[c][i] == 0:
10             for j in range(n):
11                 xn[i].append(0.)
12         else:
13             for k in range(n):
14                 xn[i].append(1.)
15 # -----
16 for c in range(len(xn)):
17     for i in range(len(xn[c])):
18         xn[c][i] = xn[c][i] + c + 0.05
19     for i in range(len(xn[c])):
20         xn[c][i] = xn[c][i] + c + 0.05
21 # -----
22 z1 = arange(len(xn[0]))
23 len(z1)
24 # -----
25 subplots(figsize=(18,5))
26 plot(z1, xn[0], z1, xn[1], z1, xn[2], z1, xn[3], z1, xn[4])
27 plt.title('Estado do Bit Para o Sinal Triangular', fontsize = 15)
28 plt.xlabel('Amostragem no tempo', fontsize = 15);
29 plt.ylabel('Estado do bit', fontsize = 15)
30 legend(['$bit 0$', '$bit 1$', '$bit 2$', '$bit 3$', '$bit 4$'], fontsize = 12)
31 grid(True); show()

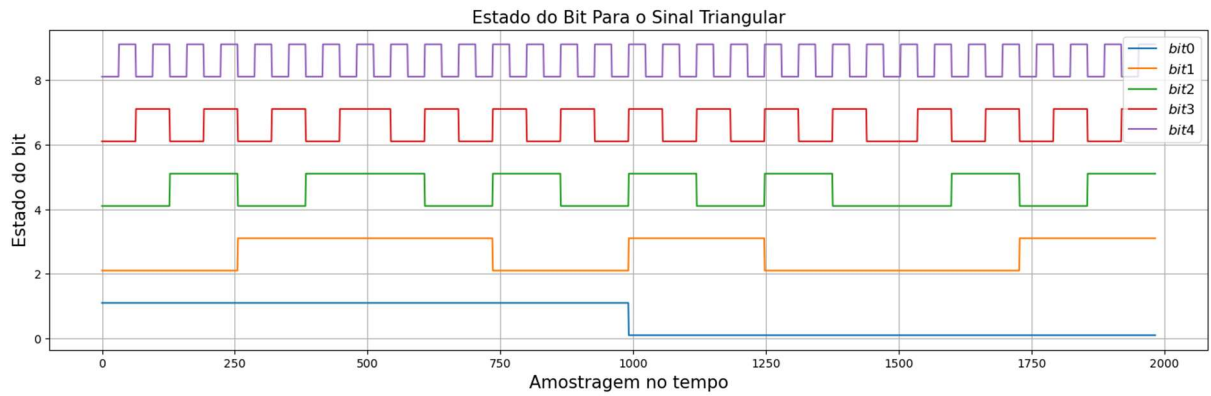
```

Fonte: autoria própria, 2024.

Assim como os gráficos da figura 12, a figura 13 mostra os tempos de acionamentos de cada GPIO correspondente aos *bits* do número binário

correspondente ao salto da amplitude do sinal triangular, sendo *bit0* o *bit* menos significativo (peso 1) e *bit4* o *bit* mais significativo (peso 16).

Figura 12 - Gráficos dos acionamentos de cada *bit* em relação ao tempo para o sinal triangular.



Fonte: autoria própria, 2024.

7 SINAL DE ONDA DENTE DE SERRA

O sinal dente de serra possui apenas uma rampa de subida, na qual a amplitude é quantizada na ordem $2^n = 2^5 = 32$. O intervalo tempo possui a mesma ordem. E assim como o sinal triangular, o sinal dente de serra também possui tempos iguais para cada amostragem.

7.1 Lista de amplitudes do sinal dente de serra

Como o sinal dente de serra possui apenas uma rampa, sendo esta uma rampa de subida, então, podemos gerar a lista (***nBinDs***) de amplitudes do sinal, com o programa abaixo:

```
1 nBinDs = []
2 for c in range(32):
3     nBinDs.append(c)
```

Fonte: autoria própria, 2024.

7.2 Convertendo os valores decimais em valores binários

Assim como os sinais anteriores, neste também devemos converter os valores decimais em binários. Para isto, utilizamos o programa abaixo:

```
1 x = [] # Lista temporária
2 # Substituindo os valores decimais por binários
3 for i in range(len(nBinDs)):
4     if i == 0:
5         t = '0'
6     else:
7         t = bin(int(nBinDs[i])).lstrip('-0b')
8     nBinDs[i] = t
9
10 print(nBinDs[:10]) # Imprime os 10 primeiros valores
11
12 >> ['0', '1', '10', '11', '100', '101', '110', '111', '1000', '1001']
```

Fonte: autoria própria, 2024.

7.3 Completando com zeros os *bits* restantes do código binário

Preenchendo com zeros à esquerda os valores binários menores que 5 *bits* e convertendo a lista de valores binários do tipo *string*, para binários do tipo inteiro.

```

for c in range(len(nBinDs)):
1   while len(nBinDs[c]) < nBit:
2       nBinDs[c] = '0' + nBinDs[c]
3   nBinDs[c] = list(nBinDs[c])
4   for i in range(len(nBinDs[c])):
5       nBinDs[c][i] = int(nBinDs[c][i])
6
7
8 print(nBinDs[:6]) # Os 6 primeiros valores
>> [[0,0,0,0,0], [0,0,0,0,1], [0,0,0,1,0], [0,0,0,1,1], [0,0,1,0,0], [0,0,1,0,1]]

```

Fonte: autoria própria, 2024.

7.4 O intervalo de tempo de amostragem

O intervalo de tempo de amostragem é determinado pela equação 14:

$$Intervalo = \frac{1}{2^n} = \frac{1}{2^5} = 0,03125 \quad \text{Equação 14}$$

Nesta equação, $2^n = 2^5 = 32$, tem o mesmo valor do tamanho da lista **nBinDs**. Portanto, através dela, pode ser gerada uma lista de intervalos a partir do programa abaixo:

```

1 tempDs = []
2 for c in range(len(nBinDs)):
3     tempDs.append(round(1/len(nBinDs), 4))
4 print(tempDs[:6]) # Os 6 primeiros intervalos
5
6 >> [0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312]

```

Fonte: autoria própria, 2024.

7.5 Valores de amplitudes e frações do período do sinal dente de serra

Lista de amplitudes **nBinDs**:

```

[[0,0,0,0,0], [0,0,0,0,1], [0,0,0,1,0], [0,0,0,1,1], [0,0,1,0,0], [0,0,1,0,1], [0,0,1,1,0],
[0,0,1,1,1], [0,1,0,0,0], [0,1,0,0,1], [0,1,0,1,0], [0,1,0,1,1], [0,1,1,0,0], [0,1,1,0,1],
[0,1,1,1,0], [0,1,1,1,1], [1,0,0,0,0], [1,0,0,0,1], [1,0,0,1,0], [1,0,0,1,1], [1,0,1,0,0],
[1,0,1,0,1], [1,0,1,1,0], [1,0,1,1,1], [1,1,0,0,0], [1,1,0,0,1], [1,1,0,1,0], [1,1,0,1,1],
[1,1,1,0,0], [1,1,1,0,1], [1,1,1,1,0], [1,1,1,1,1]]

```

Fonte: autoria própria, 2024.

Por sua vez, a lista **tempDs**, possui 32 índices com o valor 0,0312.

```

[0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312,
0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312,
0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312, 0.0312]

```

Fonte: autoria própria, 2024.

7.6 Formato gráfico dos acionamentos - Sinal dente de serra

O formato gráfico dos acionamentos de cada *bit* do conversor DAC, para o sinal dente de serra, pode ser obtido através do programa abaixo:

```

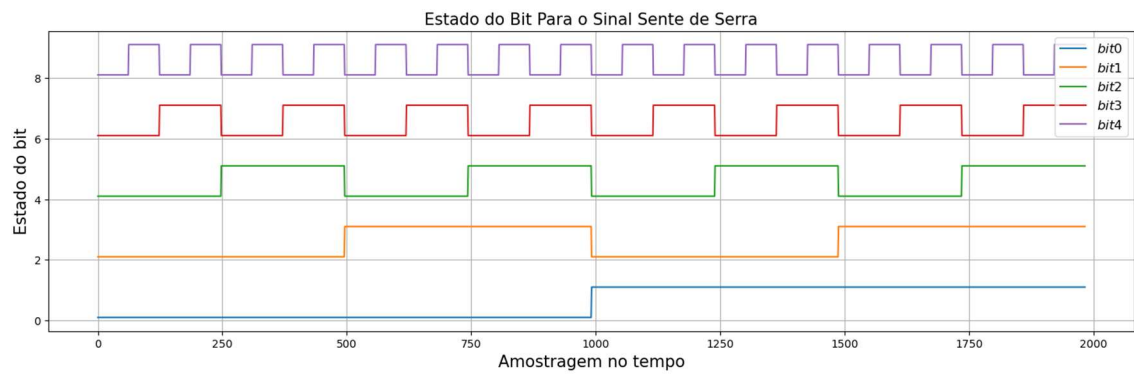
1  xn = []                      # Lista com as listas de estados para cada bit
2  N = 2000                     # Número de amostras
3  for c in range(len(nBinDs[0])):
4      xn.append([])
5  # -----
6  for c in range(len(nBinDs)):
7      n = int(tempDs[c] * N)
8      for i in range(len(nBinDs[c])):
9          if nBinDs[c][i] == 0:
10             for j in range(n):
11                 xn[i].append(0.)
12             else:
13                 for k in range(n):
14                     xn[i].append(1.)
15 # -----
16 for c in range(len(xn)):
17     for i in range(len(xn[c])):
18         xn[c][i] = xn[c][i] + c + 0.05
19     for i in range(len(xn[c])):
20         xn[c][i] = xn[c][i] + c + 0.05
21 # -----
22 z1 = arange(len(xn[0]))
23 len(z1)
24 # -----
25 subplots(figsize=(18,5))
26 plot(z1, xn[0], z1, xn[1], z1, xn[2], z1, xn[3])
27 plt.title('Estado do Bit Para o Sinal Sente de Serra', fontsize = 15)
28 plt.xlabel('Amostragem no tempo', fontsize = 15);
29 plt.ylabel('Estado do bit', fontsize = 15)
30 legend(['$bit 0$', '$bit 1$', '$bit 2$', '$bit 3$'], fontsize = 12)
31 grid(True); show()

```

Fonte: autoria própria, 2024.

Assim como as figuras 12 e 13, a figura 14 também mostra os gráficos de acionamentos das GPIOs do sinal dente de serra, sendo que, *bit0* é o *bit* menos significativo (peso 1) e *bit4* o *bit* mais significativo (peso 16).

Figura 13 - Gráficos dos acionamentos de cada *bit* em relação ao tempo para o sinal dente de serra.



Fonte: autoria própria, 2024.

8 CONDICIONAMENTO DE SINAL

8.1 Circuito de condicionamento do sinal

Como visto no item 4.1, o circuito decodificador definido foi o modelo com a configuração de resistores com ponderação binária.

Após a decodificação do sinal digital para analógico, o sinal precisa ser condicionado para alternado e amplificado para 5 VAC. Para isto, temos o circuito condicionador baseado no amplificador operacional LM741 alimentado com uma tensão simétrica de 12 V.

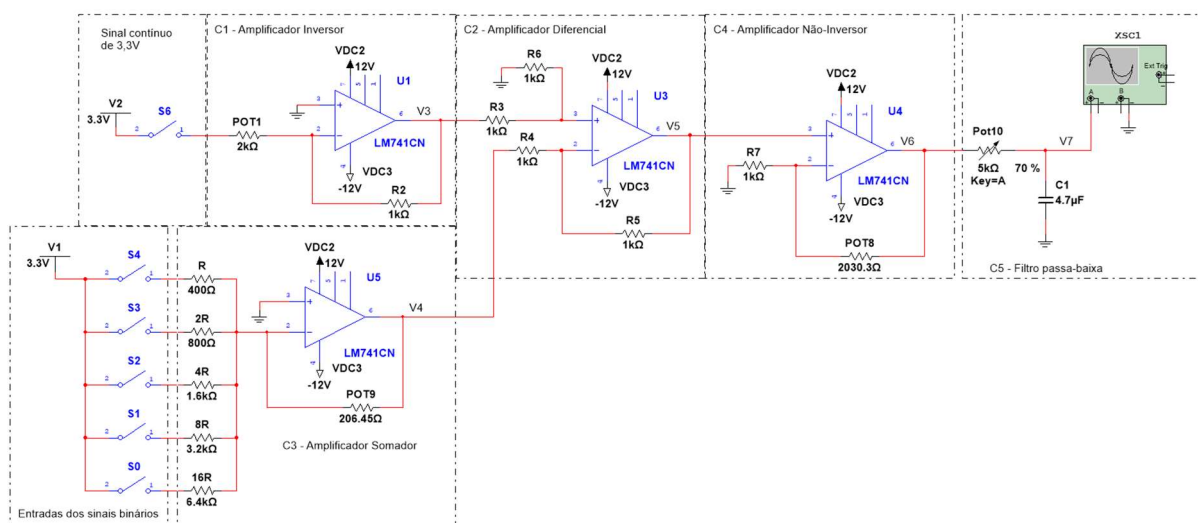
O amplificador operacional é bastante difundido no mercado. E suas configurações atendem ao propósito deste gerador de sinal.

Como se pode ver no diagrama a seguir, este circuito não possui capacitores. O propósito deste circuito é o de ser o mais simples possível em termos de quantidades de tipos de componentes.

Como o formato da onda e a frequência do sinal são gerados no Raspberry Pi Pico, não há a necessidade de capacitores e indutores para executar estas funções.

Após a geração do sinal em V6, o usuário pode escolher ou não uma filtragem do sinal, utilizando um filtro passa-baixa em paralelo com a carga. Este filtro pode ser dimensionado a partir da frequência gerada.

Figura 14 - Circuito de condicionamento de sinal.



Fonte: autoria própria, 2024.

8.2 Descrição do circuito

O circuito é subdividido em 4 outros circuitos C1, C2, C3 e C4, um para cada encapsulamento do LM741. Nesse circuito:

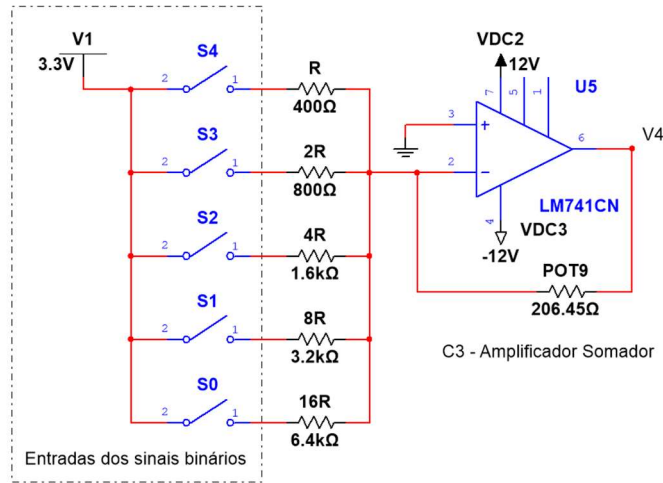
- 1 - O circuito C1, recebe um sinal contínuo de $V_2 = 3,3V$ a partir do Pino GPIO 9 do Raspberry Pi Pico. Depois retorna um sinal também contínuo e invertido em $V_3 = -1,65V$ (PERTENCE, 2003, p. 51).
- 2 - O circuito C3 é composto por uma malha de 6 resistores que compõem o divisor de tensão e um amplificador somador. Os resistores que recebem os sinais de tensão $V_1 = 3,3V$ a partir dos Pino GPIOs de 10 a 14. A sequência de acionamentos representa números binários conforme o formato do sinal que for selecionado. Por sua vez, o amplificador somador faz a soma dos sinais enviados pelos pinos GPIO, resultando em uma tensão de saída invertida de $V_4 = 0V$ a $V_4 = -3,3V$ (PERTENCE, 2003, p. 60).
- 3 - O circuito C2 é um amplificador diferencial. Ele recebe e soma as tensões V_3 e V_4 , resultando em uma saída $V_5 = \pm 1,65V$ (PERTENCE, 2003, p. 62).
- 4 - O circuito C4 é um amplificador não-inversor. Ele tem a função de amplificar a tensão V_5 em uma tensão $V_6 = \pm 5V$ (PERTENCE, 2003, p. 53).
- 5 - Ao final, temos o circuito C5 com um osciloscópio sendo representado para verificação do sinal. O C5 é um circuito RC, tendo a função de filtrar as frequências do sinal senoidal (BOYLESTAD, 2013).

8.3 Dimensionamento do circuito C3

8.3.1 Malha de resistores

O C3 é o circuito conversor digital-analógico (DAC). Nele, as chaves S_0 à S_5 representam as saídas GPIO 10 a 14. O circuito C3 é mostrado na figura 16.

Figura 15 - Circuito C3.



Fonte: autoria própria, 2024.

A tensão de referência para todas as saídas GPIO é $V_1 = 3,3V$. E a tensão na entrada 2 do LM741 é atenuada em cada resistor da entrada, sendo esta atuação em cada resistor (V_{Rn}):

$$V_{Rn} = \frac{V_1}{2^{n-1}} \quad \text{Equação 15}$$

Nesta equação, n_n corresponde ao resistor que verifica a atenuação de tensão.

Na montagem prática do circuito, os resistores R_n foram substituídos por Trimpots, os quais tiveram os valores de resistência medidos e ajustados, antes da montagem.

O resistor equivalente R_{eq} pode ser calculado pela equação 16, na qual n é o número de *bits* do DAC.

$$R_{eq} = \left(\frac{2^{n-1}}{2^n - 1} \right) R \quad \text{Equação 16}$$

Tomando como $R = 400\Omega$ e 5 o número de *bits*, temos a resistência equivalente:

$$R_{eq} = \left(\frac{2^{5-1}}{2^5 - 1} \right) 400 = 206,45\Omega$$

8.3.2 Amplificador somador

Como o ganho no circuito é nulo $A = 1$, a resistência Pot_9 é igual à resistência equivalente: $R_{eq} = Pot_9 = 206,45\Omega$.

Nesse caso, a resistência Pot_9 é um potenciômetro para ajuste do ganho em V_4 . Com $A = 1$, temos a equação 17:

$$\frac{V_1}{R_{eq}} + \frac{V_4}{Pot_9} = 0 \quad \text{Equação 17}$$

Isolando V_4 , onde $V_1 = 3,3V$, temos:

$$V_4 = \left(\frac{-V_1}{R_{eq}} \right) Pot_9 \Rightarrow V_4 = -V_1 \Rightarrow V_4 = -3,3V$$

8.4 Dimensionamento do circuito C1

O circuito C1 é um amplificador inversor que está configurado para se obter um ganho menor que 1.

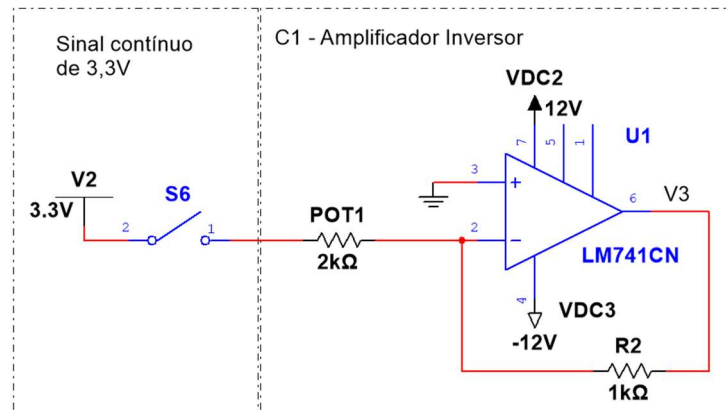
Ele recebe um sinal contínuo de $V_2 = 3,3V$ a partir do pino GPIO 9 do Raspberry Pi Pico, representado por S_6 . Depois retorna um sinal também contínuo e invertido em $V_3 = -1,65V$. Logo, $V_3 = \frac{-V_2}{2}$. Temos então:

$$\frac{V_3}{V_2} = \frac{-R_1}{Pot_1} \quad \text{Equação 18}$$

$$Pot_1 = \frac{-V_2 R_2}{V_3} \Rightarrow Pot_1 = \frac{-3,3 \cdot 1000}{1,65} = 2k\Omega$$

O circuito C1 e a entrada GPIO 9, são mostrados na figura 17.

Figura 16 - Circuito amplificador inversor.



Fonte: autoria própria, 2024.

8.5 Dimensionamento do circuito C2

O circuito C2 é um circuito amplificador diferencial. Ele tem a função de somar as tensões V_3 e V_4 para obtermos uma saída V_5 de tensão alternada.

A tensão V_4 oscila conforme os acionamentos das saídas GPIO, entre $0V$ e $-3,3V$. Por sua vez, a tensão V_3 está constante em $-1,65V$.

A saída V_5 é determinada pela equação 19:

$$V_5 = \frac{R_5}{R_4} (V_3 - V_4) \quad \text{Equação 19}$$

Onde $R_3 = R_4$ e $R_5 = R_6$ e como o ganho é $A = 1$, temos $R_4 = R_5$. Assim:

$$V_5 = V_3 - V_4 \quad \text{Equação 20}$$

Para $V_4 = 0V$, temos:

$$V_5 = -1,65 - 0 \Rightarrow V_5 = -1,65V$$

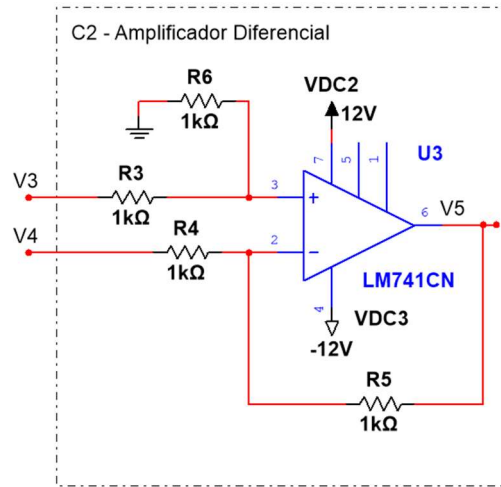
Para $V_4 = -3,3V$, temos:

$$V_5 = -1,65 - (-3,3) \Rightarrow V_5 = 1,65V$$

Logo, podemos ver que V_5 oscila em: $V_5 = \pm 1,65V$.

O circuito C2 pode ser visualizado na figura 18.

Figura 17 - Circuito C2, amplificador diferencial.



Fonte: autoria própria, 2024.

8.6 Dimensionamento do circuito C4

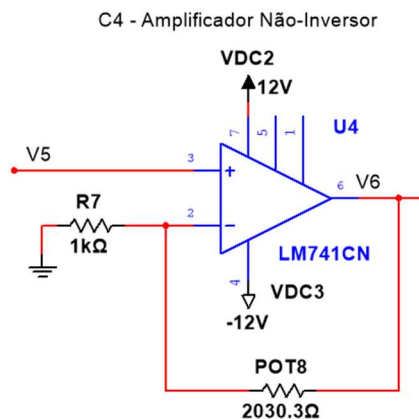
O circuito C4 mostrado na figura 19 é um amplificador não-inversor. Ele tem a função de amplificar o sinal $V_5 = \pm 1,65V$ para $V_6 = \pm 5V$. Temos o ganho A :

$$A = \frac{V_6}{V_5} = 1 + \frac{Pot_8}{R_7} \quad \text{Equação 21}$$

Isolando o potenciômetro Pot_8 , temos:

$$Pot_8 = R_7 \left(\frac{V_6}{V_5} - 1 \right) \Rightarrow Pot_8 = 1000 \left(\frac{5}{1,65} - 1 \right) \Rightarrow Pot_8 = 2,03k\Omega$$

Figura 18 - Circuito amplificador não-inversor.



Fonte: autoria própria, 2024.

8.7 Circuito C5 - Filtro Passa-Baixa

Como pode-se ver graficamente na figura 11, o sinal senoidal quantizado possui degraus e não uma curva suave. Estes degraus podem ser interpretados como ruídos no sinal. Estes ruídos podem ser removidos através de um filtro passa-baixa de primeira ordem formado por um circuito RC, constituído por resistor mais capacitor conectados em paralelo. Este circuito é mostrado na figura 20, juntamente com a representação do osciloscópio conectado na saída V_7 .

A frequência de corte (f_c) para um filtro passa-baixa de primeira ordem pode ser determinada pela equação 22:

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi Pot_{10} C_1} \quad \text{Equação 22}$$

Nesta equação, R é o valor do resistor e C é o capacitor que estão respectivamente representados no circuito Pot_{10} e C_1 .

Como visto no item 5.5.3.4, a maior frequência de um sinal senoidal gerada por este gerador de sinais é de $100Hz$. E a menor frequência é de $8Hz$. Isto porque $8Hz$ também é a menor frequência de um sinal PWM gerado pelo Raspberry Pi Pico.

Como já sabemos, o circuito DAC é de 5 bits , e a quantidade de amostras para um sinal senoidal é de 62 amostras ($N_{amostras} = 2(s^5 - 1) = 62$). Assim, podemos ver que a frequência de amostragem é de 62 vezes a frequência do sinal.

O capacitor C_1 escolhido para este projeto é de $4,7\mu F$. Isto porque o capacitor com este valor é relativamente fácil de se encontrar no mercado ou, até mesmo, removê-lo de outro aparelho eletrônico em desuso.

Já temos a frequência de corte (f_c) que pode variar de $8Hz$ a $100Hz$. Assim, para que possamos ter um melhor ajuste na filtragem do sinal, vamos utilizar um potenciômetro no lugar do valor fixo do resistor R e vamos calcular o valor deste potenciômetro através do rearranjo da equação 22.

$$Pot_{10} = \frac{1}{2\pi f_c C_1} \quad \text{Equação 23}$$

Obs: Os valores de impedância (Z) e de ganho (A_v) não são levados em conta

neste projeto.

Para maior frequência de corte 100Hz , temos o menor valor do potenciômetro:

$$Pot_{10} = \frac{1}{2\pi \cdot 100 \cdot 4,7 \times 10^{-6}} = 389\Omega$$

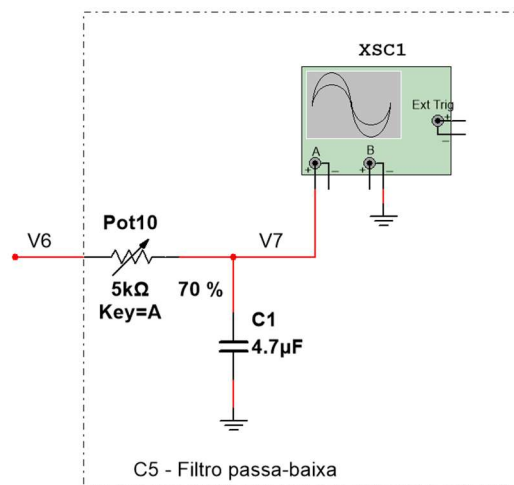
E para menor frequência de corte 8Hz , temos o maior valor do potenciômetro:

$$Pot_{10} = \frac{1}{2\pi \cdot 8 \cdot 4,7 \times 10^{-6}} = 4233\Omega$$

No caso deste projeto, vamos utilizar um potenciômetro linear de $5k\Omega$ de valor máximo de resistência.

Na figura 20, temos apenas o circuito do filtro passa-baixa.

Figura 19 - Circuito amplificador não-inversor.



Fonte: autoria própria, 2024.

8.8 Conclusão do circuito de condicionamento de sinal

E assim, concluímos o circuito de condicionamento de sinal. Ele é responsável por receber as comutações das saídas GPIO 10, 11, 12, 13 e 14 do Raspberry Pi Pico, representadas no diagrama por S_0 , S_1 , S_2 , S_3 e S_5 . Nele, a sequência de comutações que representa, respectivamente, um número binário e um valor de amplitude do sinal. O conversor DAC é responsável por converter estas sequências de comutações em

um valor de amplitude do sinal analógico.

Todas as saídas GPIO fornecem uma tensão de $3,3V$ positiva, independente do formato do sinal. Após o conversor DAC, o sinal é modificado para amplitudes positivas e negativas, formando, assim, um sinal alternado entre $V_5 = \pm 1,65V$. Depois, este sinal é amplificado para $V_6 = \pm 5V$.

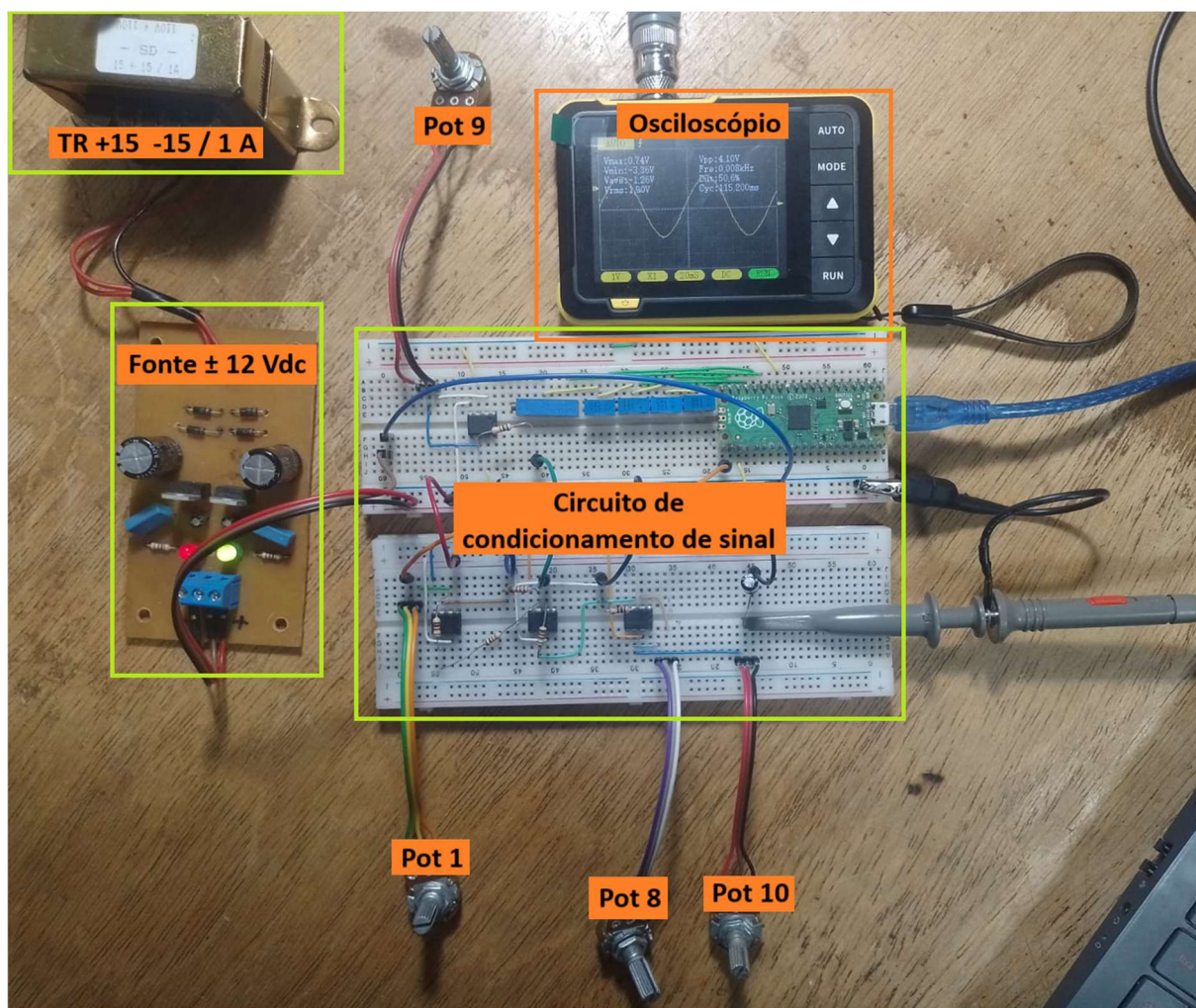
Com o filtro passa-baixa, podemos ter um sinal senoidal mais limpo, com menos ruídos. Como vimos no item 8.7, o filtro foi projetado para isolar as frequências da amostragem do sinal. E também serve para remover os ruídos que são gerados na placa de montagem, como veremos mais à frente nas figuras 25 e 26.

9 ANÁLISE DOS SINAIS

9.1 Montagem do circuito

A montagem do circuito foi feita em duas placas *protoboard* e conectadas entre si através de fios elétricos; uma fonte simétrica de $\pm 12V$ com um transformador $+15V$ e $-15V / 1A$, que alimenta os amplificadores operacionais LM741. As ponteiros do osciloscópio estão conectadas ao GND do circuito e ao ponto V_7 . O Raspberry Pi Pico é alimentado e recebe os comandos diretamente do computador através do cabo micro USB. Todo o arranjo desta montagem pode ser visto na figura 21.

Figura 20 - Montagem geral dos circuitos.

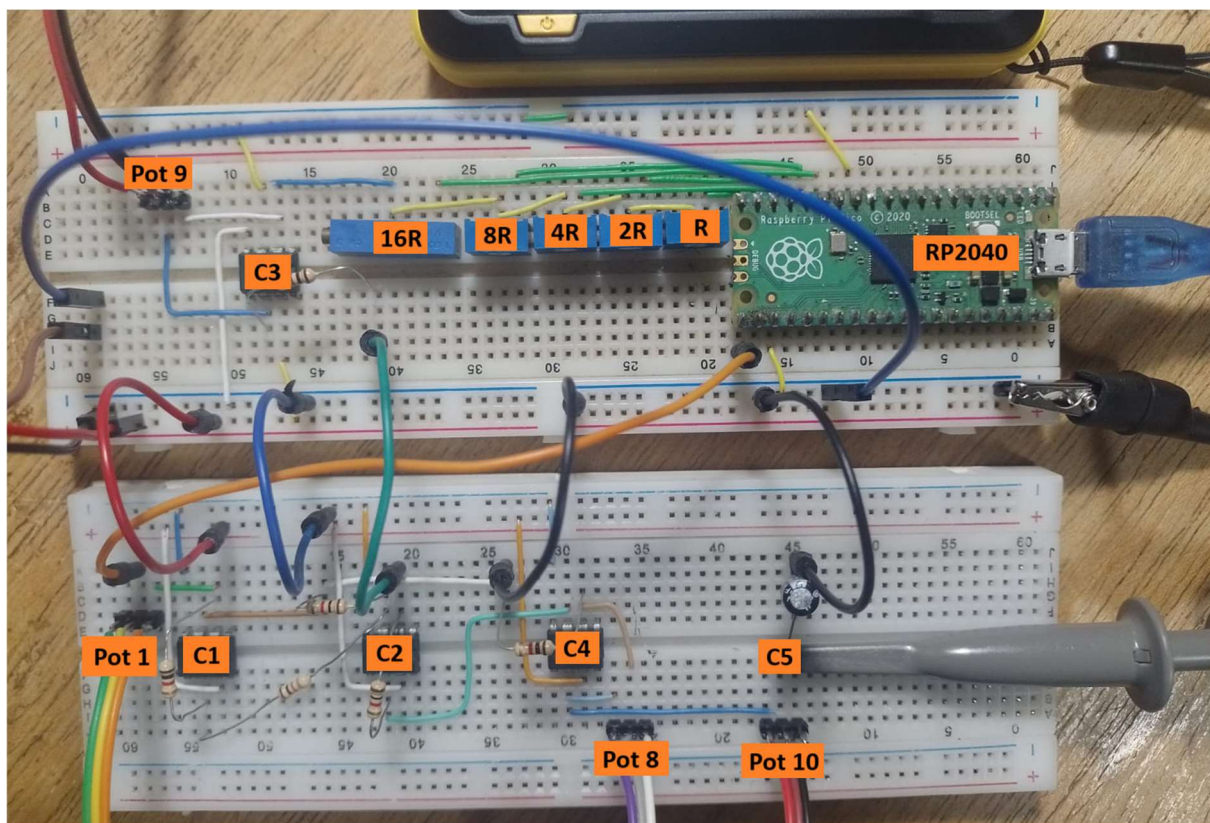


Fonte: autoria própria, 2024.

9.1.1 Localização dos componentes

O circuito de condicionamento de sinal montado nas placas *protoboard*, pode ser verificado na figura 22, onde também pode-se ver as localizações dos principais componentes.

Figura 21 - Localização dos componentes na *protoboard*.



Fonte: autoria própria, 2024.

9.2 Gerando o sinal senoidal

Após as definições dos parâmetros do projeto, da obtenção das amostras do sinais e do desenvolvimento do circuito de condicionamento de sinal. Podemos gerar os sinais e analisar os resultados utilizando o programa em Micropython mostrado no capítulo 10. As análises ficaram restritas à qualidade dos formatos dos sinais, sendo que apenas o sinal senoidal foi filtrado através de um filtro passa-baixa.

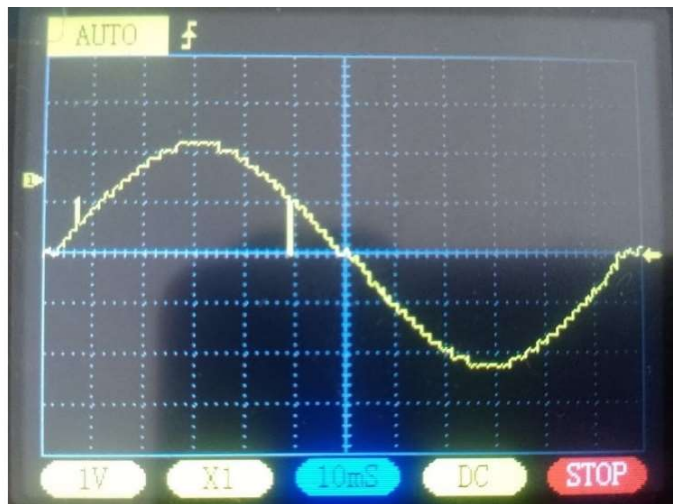
Em todos os casos, os sinais foram verificados através do Osciloscópio FNIRSI DSO152.

9.2.1 Sinal senoidal sem filtro passa-baixa

O primeiro formato de onda a ser analisado foi o sinal senoidal sem filtro no ponto V_7 no circuito C5. O resultado pode ser verificado na figura 23, na qual o sinal é mostrado em apenas um período. E, na figura 24, o sinal é mostrado em dois períodos.

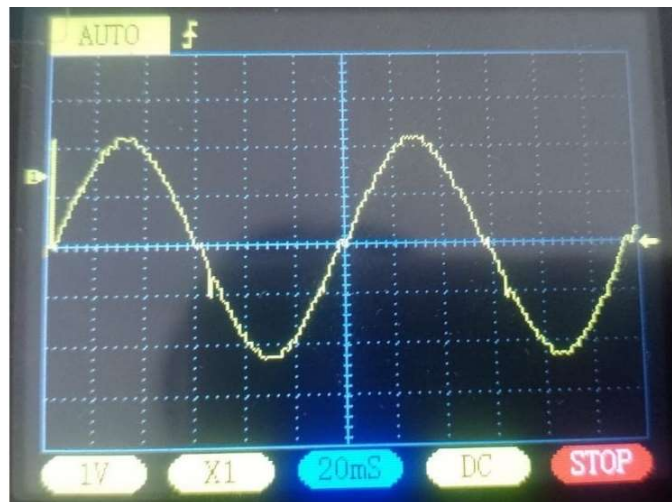
Visualmente podemos notar nas duas imagens, que há um atraso do sinal em meio período ($T/2$) e no final do período (T).

Figura 22 - Sinal senoidal sem filtro passa-baixa - 1 período.



Fonte: autoria própria, 2024.

Figura 23 - Sinal senoidal sem filtro passa-baixa - 2 períodos.

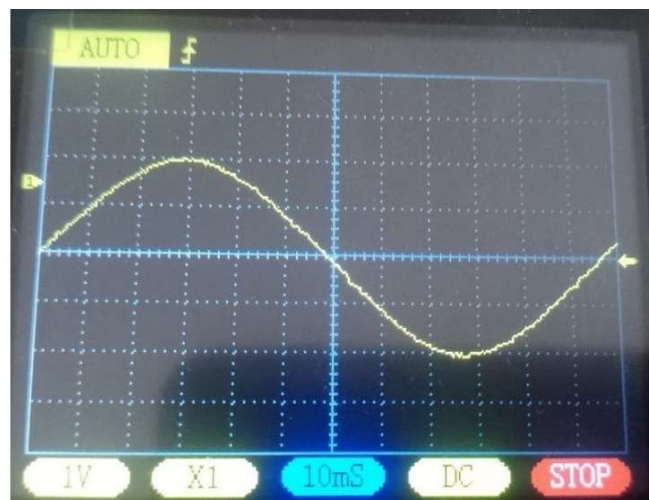


Fonte: autoria própria, 2024.

9.2.2 Sinal senoidal com filtro passa-baixa

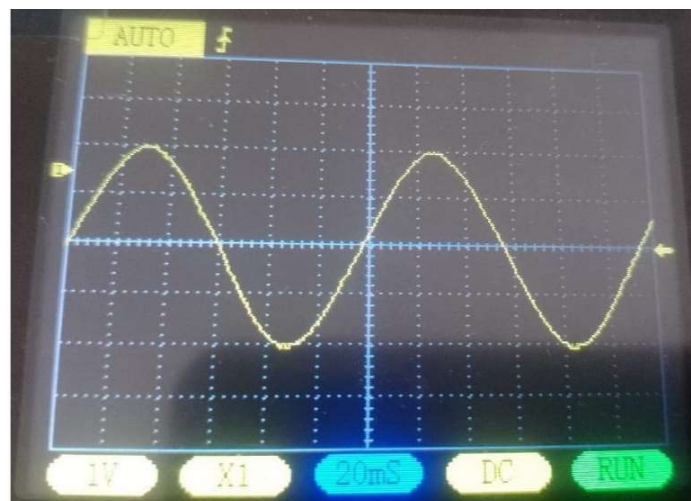
Já nas figuras 25 e 26, podemos ver os mesmos formatos de onda anteriores, só que, desta vez, com o sinal filtrado em passa-baixa. Podemos verificar visualmente que o atraso não é mais percebido.

Figura 24 - Sinal senoidal filtrado - 1 período.



Fonte: autoria própria, 2024.

Figura 25 - Sinal senoidal filtrado - 2 períodos.

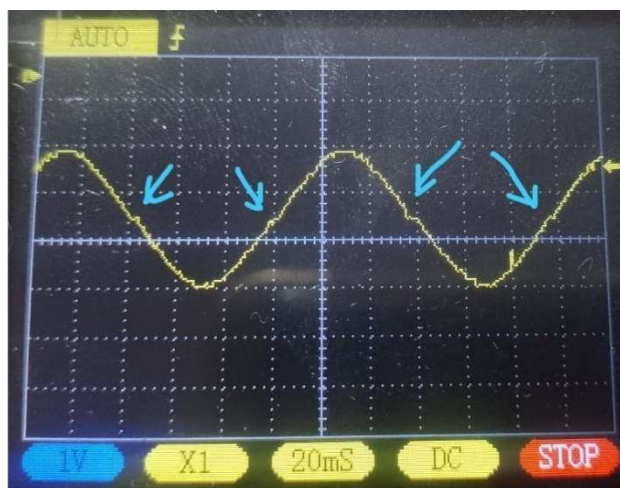


Fonte: autoria própria, 2024.

9.2.3 Sinal senoidal em V4 do circuito C3

Ao analisar o circuito de condicionamento de sinal, não foi encontrado nenhum ponto que pudesse ser suspeito de gerar o atraso do sinal, justamente nos pontos de rampa de subida e rampa de descida do sinal. Então, foi analisado o sinal em V_4 , no circuito C3. Este é ponto de saída do conversor DAC. Este sinal não está filtrado e podemos verificar visualmente no osciloscópio, o sinal também possui um atraso. A figura 27, mostra esse resultado, onde podemos ver o atraso indicado pelas setas desenhadas em azul.

Figura 26 - Sinal senoidal analisado em V4 do circuito 3.



Fonte: autoria própria, 2024.

9.2.4 Modificando as amostragens do sinal

Foi verificado que a causa do atraso no sinal não está no circuito de condicionamento de sinal. Em vista disso, passou-se a verificar o código de geração do sinal em Micropython (O código de execução é mostrado no capítulo 10).

Como o problema estava no atraso do sinal, uma nova lista (***nBinSenoX***) com amostras das amplitudes foi gerada a partir da lista já existente (***nBinSeno***). Para esta nova lista, foram removidas 4 amostras nos índices: 0, 30, 31 e 61. Elas correspondem à primeira amostra, às duas amostras centrais e à última amostra.

Os valores de fração de períodos correspondentes também foram removidos da lista ***tempSeno***. Depois foram somados e a soma foi dividida pelo número de amostras restantes. O resultado da divisão foi somado a cada amostra restante. Estas

operações podem ser verificadas nas linhas 12, 13 e 14 do código abaixo. E assim, foi gerada a lista **tempSenoX**.

Um sinal senoidal que antes possuía 62 amostras, agora possui 58 amostras.

```

1 nBinSenoX = []      # Amostras de 58 amplitudes
2 tempSenoX = []      # Amostras de 58 frações de período
3 t1 = []             # Variável temporária
4 for c in range(len(tempSeno)):
5     if c != 0 and c != 30 and c != 31 and c != 61:
6         tempSenoX.append(tempSeno[c])
7         nBinSenoX.append(nBinSeno[c])
8     else:
9         t1.append(tempSeno[c])
10
11 # Somando nas amostras restantes
12 t2 = sum(t1)/len(tempSenoX)
13 for c in range(len(tempSenoX)):
14     tempSenoX[c] = round(tempSenoX[c] + t2, 4)
15
16 print(nBinSenoX, '\n\n', tempSenoX)

```

Fonte: autoria própria, 2024.

O resultado da modificação *nBinSeno*:

```

[[1,0,0,0,1], [1,0,0,1,0], [1,0,0,1,1], [1,0,1,0,0], [1,0,1,0,1], [1,0,1,1,0], [1,0,1,1,1],
[1,1,0,0,0], [1,1,0,0,1], [1,1,0,1,0], [1,1,0,1,1], [1,1,1,0,0], [1,1,1,0,1], [1,1,1,1,0],
[1,1,1,1,1], [1,1,1,1,0], [1,1,1,0,1], [1,1,1,0,0], [1,1,0,1,1], [1,1,0,1,0], [1,1,0,0,1],
[1,1,0,0,0], [1,0,1,1,1], [1,0,1,1,0], [1,0,1,0,1], [1,0,1,0,0], [1,0,0,1,1], [1,0,0,1,0],
[1,0,0,0,1], [0,1,1,1,0], [0,1,1,0,1], [0,1,1,0,0], [0,1,0,1,1], [0,1,0,1,0], [0,1,0,0,1],
[0,1,0,0,0], [0,0,1,1,1], [0,0,1,1,0], [0,0,1,0,1], [0,0,1,0,0], [0,0,0,1,1], [0,0,0,1,0],
[0,0,0,0,1], [0,0,0,0,0], [0,0,0,0,1], [0,0,0,1,0], [0,0,0,1,1], [0,0,1,0,0], [0,0,1,0,1],
[0,0,1,1,0], [0,0,1,1,1], [0,1,0,0,0], [0,1,0,0,1], [0,1,0,1,0], [0,1,0,1,1], [0,1,1,0,0],
[0,1,1,0,1], [0,1,1,1,0]]

```

Fonte: autoria própria, 2024.

O resultado da modificação *tempSenoX*:

```

[0.0112, 0.0107, 0.0117, 0.0112, 0.0117, 0.0122, 0.0122, 0.0132, 0.0137, 0.0147, 0.0157,
0.0182, 0.0217, 0.0307, 0.0822, 0.0307, 0.0217, 0.0182, 0.0157, 0.0147, 0.0137, 0.0132,
0.0122, 0.0122, 0.0117, 0.0112, 0.0117, 0.0107, 0.0112, 0.0112, 0.0107, 0.0117, 0.0112,
0.0117, 0.0122, 0.0122, 0.0132, 0.0137, 0.0147, 0.0157, 0.0182, 0.0217, 0.0307, 0.0822,
0.0307, 0.0217, 0.0182, 0.0157, 0.0147, 0.0137, 0.0132, 0.0122, 0.0122, 0.0117, 0.0112,
0.0117, 0.0107, 0.0112]

```

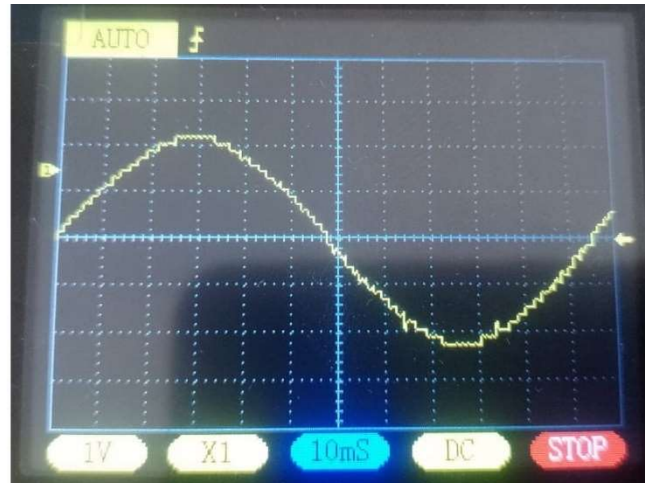
Fonte: autoria própria, 2024.

9.2.5 Resultado do novo sinal senoidal

Assim como na seção 9.1.1, foram gerados novos sinais sem filtro e com filtro, cujos resultados são mostrados respectivamente nas figuras 28 e 29. Visualmente,

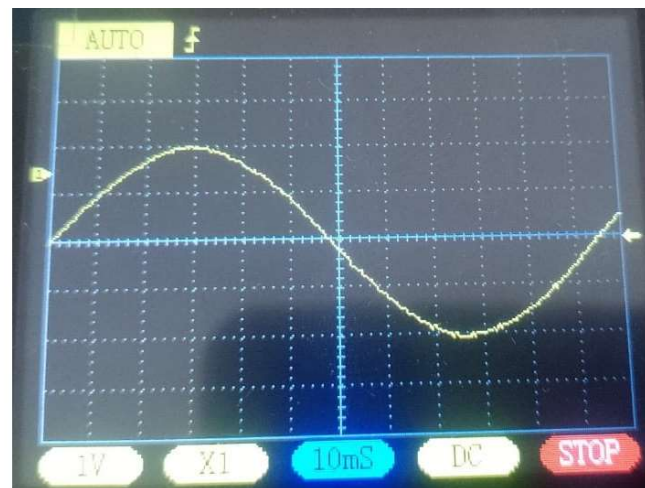
podemos notar uma melhora na qualidade do sinal não filtrado e, conseqüentemente, no sinal filtrado. Para uma análise mais apurada para verificar o atraso do sinal e a qualidade do novo sinal, se faz necessário um osciloscópio de maior precisão.

Figura 27 - Sinal senoidal de 58 amostras, sem filtro passa-baixa - 1 período.



Fonte: autoria própria, 2024.

Figura 28 - Sinal senoidal de 58 amostras e filtrado.



Fonte: autoria própria, 2024.

9.3 Sinal triangular

O resultado da geração do sinal triangular não foi diferente do sinal senoidal, uma vez que a lista de amplitudes é a mesma. A figura 30 mostra como fica este sinal como 62 amostras. Sendo a fração de período:

$$Tn = \frac{1}{62} = 0,01613 \quad \text{Equação 24}$$

Figura 29 - Sinal triangular gerado a partir de 62 amostras.

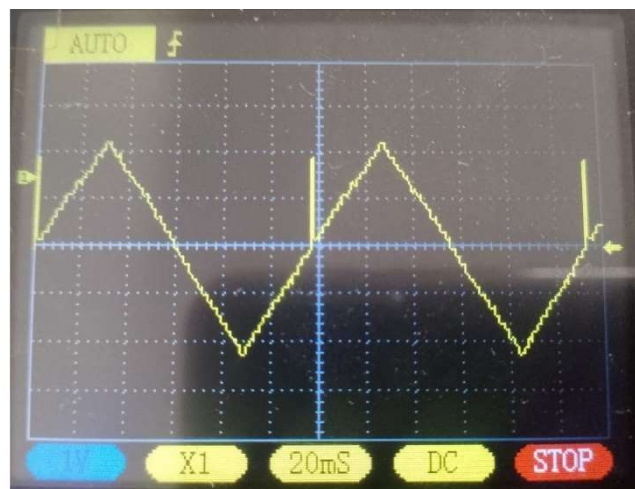


Fonte: autoria própria, 2024.

Já, para o sinal triangular gerado a partir de 58 amostras, temos o resultado mostrado na figura 31, sendo a fração de período:

$$Tn = \frac{1}{58} = 0,01724 \quad \text{Equação 25}$$

Figura 30 - Sinal triangular gerado a partir de 58 amostras.



Fonte: autoria própria, 2024.

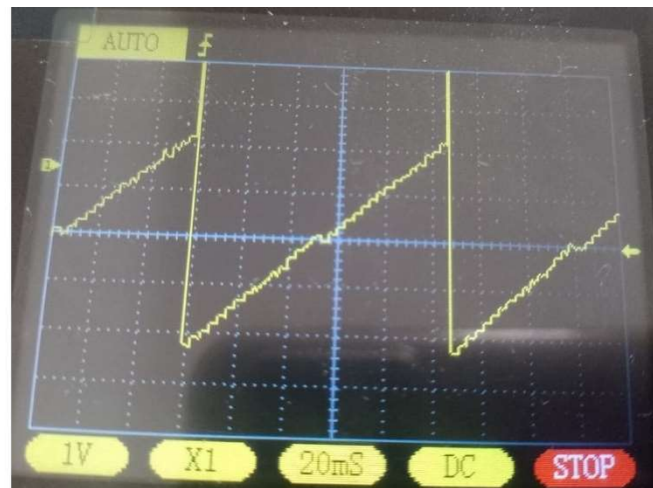
9.4 Sinal dente de serra

O sinal dente de serra de 5 *bits* possui 32 amostras de amplitudes e uma fração de período:

$$T_n = \frac{1}{32} = 0,03125 \quad \text{Equação 26}$$

Verifica-se visualmente que o sinal também possui um atraso no centro da rampa. Verifica-se também um pico de tensão no final desta rampa. Este pico pode ser removido através de um outro circuito para tratar este sinal.

Figura 31 - Sinal dente de serra de 32 amostras.

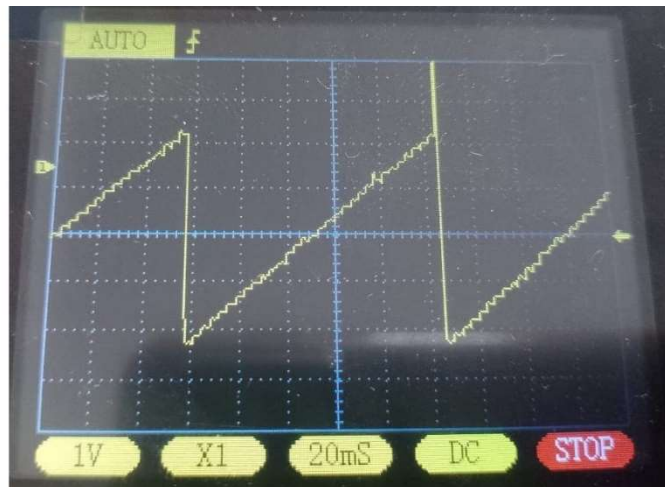


Fonte: autoria própria, 2024.

Assim como no número de amostras dos sinais senoidal e triangular, o número de amostras do sinal dente de serra também foi reduzido. Neste caso, foi removida a amostra de índice 15 (binário: 01111). O resultado desta nova amostragem pode ser visto na figura 33. E a fração de período é:

$$T_n = \frac{1}{31} = 0,03226 \quad \text{Equação 27}$$

Figura 32 - Sinal dente de serra de 31 amostras.



Fonte: autoria própria, 2024.

10 CÓDIGO EM MICROPYTHON

O código do programa escrito em Micropython utilizado para gerar os sinais senoidal, triangular e dente de serra, a partir do Raspberry Pi Pico pode ser verificado a seguir.

As listas de amostragem das amplitudes e das frações do período geradas anteriormente, são adicionadas ao programa.

O programa possui uma função **gera_sinal()** que recebe a frequência e o tipo de sinal que se deseja executar no momento.

A frequência é convertida em um período T em μs (microssegundos) através da equação 28:

$$T = \frac{1}{f} \cdot 1000000 \quad \text{Equação 28}$$

```

1 from machine import Pin
2 from time import sleep, sleep_ms, sleep_us
3 import time
4 # ----- saídas GPIO -----
5 gpioN = [Pin(15, Pin.OUT), Pin(14, Pin.OUT), Pin(13, Pin.OUT), Pin(12, Pin.OUT),
6          Pin(11, Pin.OUT), Pin(10, Pin.OUT), Pin(16, Pin.OUT)]
7 # ----- Listas para os sinais senoidal e triangular -----
8 # Esta lista amplitudes para sinal seno triangular
9 # 62 amostras
10 binSeno = [[1,0,0,0,0], [1,0,0,0,1], [1,0,0,1,0], [1,0,0,1,1], [1,0,1,0,0], [1,0,1,0,1],
11            [1,0,1,1,0], [1,0,1,1,1], [1,1,0,0,0], [1,1,0,0,1], [1,1,0,1,0], [1,1,0,1,1],
12            [1,1,1,0,0], [1,1,1,0,1], [1,1,1,1,0], [1,1,1,1,1], [1,1,1,1,0], [1,1,1,0,1],
13            [1,1,1,0,0], [1,1,0,1,1], [1,1,0,1,0], [1,1,0,0,1], [1,1,0,0,0], [1,0,1,1,1],
14            [1,0,1,1,0], [1,0,1,0,1], [1,0,1,0,0], [1,0,0,1,1], [1,0,0,1,0], [1,0,0,0,1],
15            [1,0,0,0,0], [0,1,1,1,1], [0,1,1,1,0], [0,1,1,0,1], [0,1,1,0,0], [0,1,0,1,1],
16            [0,1,0,1,0], [0,1,0,0,1], [0,1,0,0,0], [0,0,1,1,1], [0,0,1,1,0], [0,0,1,0,1],
17            [0,0,1,0,0], [0,0,0,1,1], [0,0,0,1,0], [0,0,0,0,1], [0,0,0,0,0], [0,0,0,0,1],
18            [0,0,0,1,0], [0,0,0,1,1], [0,0,1,0,0], [0,0,1,0,1], [0,0,1,1,0], [0,0,1,1,1],
19            [0,1,0,0,0], [0,1,0,0,1], [0,1,0,1,0], [0,1,0,1,1], [0,1,1,0,0], [0,1,1,0,1],
20            [0,1,1,1,0], [0,1,1,1,1]]
21 # 58 amostras
22 binSenoX = [[1,0,0,0,1], [1,0,0,1,0], [1,0,0,1,1], [1,0,1,0,0], [1,0,1,0,1], [1,0,1,1,0],
23            [1,0,1,1,1], [1,1,0,0,0], [1,1,0,0,1], [1,1,0,1,0], [1,1,0,1,1], [1,1,1,0,0],
24            [1,1,1,0,1], [1,1,1,1,0], [1,1,1,1,1], [1,1,1,1,0], [1,1,1,0,1], [1,1,1,0,0],
25            [1,1,0,1,1], [1,1,0,1,0], [1,1,0,0,1], [1,1,0,0,0], [1,0,1,1,1], [1,0,1,1,0],
26            [1,0,1,0,1], [1,0,1,0,0], [1,0,0,1,1], [1,0,0,1,0], [1,0,0,0,1], [0,1,1,1,0],
27            [0,1,1,0,1], [0,1,1,0,0], [0,1,0,1,1], [0,1,0,1,0], [0,1,0,0,1], [0,1,0,0,0],
28            [0,0,1,1,1], [0,0,1,1,0], [0,0,1,0,1], [0,0,1,0,0], [0,0,0,1,1], [0,0,0,1,0],
29            [0,0,0,0,1], [0,0,0,0,0], [0,0,0,0,1], [0,0,0,1,0], [0,0,0,1,1], [0,0,1,0,0],
30            [0,0,1,0,1], [0,0,1,1,0], [0,0,1,1,1], [0,1,0,0,0], [0,1,0,0,1], [0,1,0,1,0],
31            [0,1,0,1,1], [0,1,1,0,0], [0,1,1,0,1], [0,1,1,1,0]]
32 # Lista de valores relativos ao tempo de acionado de cada amplitude quantizada
33 # 62 amostras
34 tempSeno = [0.0105, 0.0105, 0.01, 0.011, 0.0105, 0.011, 0.0115, 0.0115, 0.0125,
```

```

35         0.013, 0.014, 0.015, 0.0175, 0.021, 0.03, 0.0815, 0.03, 0.021,
36         0.0175, 0.015, 0.014, 0.013, 0.0125, 0.0115, 0.0115, 0.011, 0.0105,
37         0.011, 0.01, 0.0105, 0.0105, 0.01, 0.0105, 0.01, 0.011, 0.0105,
38         0.011, 0.0115, 0.0115, 0.0125, 0.013, 0.014, 0.015, 0.0175, 0.021,
39         0.03, 0.0815, 0.03, 0.021, 0.0175, 0.015, 0.014, 0.013, 0.0125,
40         0.0115, 0.0115, 0.011, 0.0105, 0.011, 0.01, 0.0105, 0.01]
41 # 58 amostras
42 tempSenoX = [0.0112, 0.0107, 0.0117, 0.0112, 0.0117, 0.0122, 0.0122, 0.0132, 0.0137,
43             0.0147, 0.0157, 0.0182, 0.0217, 0.0307, 0.0822, 0.0307, 0.0217, 0.0182,
44             0.0157, 0.0147, 0.0137, 0.0132, 0.0122, 0.0122, 0.0117, 0.0112, 0.0117,
45             0.0107, 0.0112, 0.0112, 0.0107, 0.0117, 0.0112, 0.0117, 0.0122, 0.0122,
46             0.0132, 0.0137, 0.0147, 0.0157, 0.0182, 0.0217, 0.0307, 0.0822, 0.0307,
47             0.0217, 0.0182, 0.0157, 0.0147, 0.0137, 0.0132, 0.0122, 0.0122, 0.0117,
48             0.0112, 0.0117, 0.0107, 0.0112]
49 # ----- Listas para o sinal dente de serra -----
50 binSerra = [[0,0,0,0,0], [0,0,0,0,1], [0,0,0,1,0], [0,0,0,1,1], [0,0,1,0,0], [0,0,1,0,1],
51             [0,0,1,1,0], [0,0,1,1,1], [0,1,0,0,0], [0,1,0,0,1], [0,1,0,1,0], [0,1,0,1,1],
52             [0,1,1,0,0], [0,1,1,0,1], [0,1,1,1,0], [1,0,0,0,0], [1,0,0,0,1], [1,0,0,1,0],
53             [1,0,0,1,1], [1,0,1,0,0], [1,0,1,0,1], [1,0,1,1,0], [1,0,1,1,1], [1,1,0,0,0],
54             [1,1,0,0,1], [1,1,0,1,0], [1,1,0,1,1], [1,1,1,0,0], [1,1,1,0,1], [1,1,1,1,0],
55             [1,1,1,1,1]]
56
57 binTria = binSeno
58 # ----- Funções dos sinais -----
59 def gera_sinal(freq, sinal):
60     # ----- Definindo o sinal que será impresso -----
61     if sinal == 'seno': # Obtem os dados para impressão do sinal senoidal
62         lista = binSeno
63         tempo = tempSeno
64     elif sinal == 'denteSerra': # Obtém os dados para imprimir o sinal dente de serra
65         lista = binSerra
66         tempo = []
67     for c in range(32): # Gera uma lista de frações de período: 2^5 = 32
68         tempo.append(0.0322) # 1 / 32 = 0.0322
69     else: # Obtem os dados para impressão do sinal triangular
70         lista = binTria
71         tempo = []
72     for c in range(62): # Gera uma lista de frações de período: 2 * (2^5 - 1) = 62
73         tempo.append(0.0161) # 1 / 62 = 0.0161
74     # ----- Imprimindo o sinal -----
75     gpioN[6].value(1) # Aciona a saída para obter 1,65Vcc
76     T = 1000000 / freq # Período em microsegundos
77     for i in range(len(lista)): # Leitura da amplitude do sinal
78         for j in range(len(lista[i])): # Leitura da saída GPIO
79             gpioN[j].value(lista[i][j]) # Aciona a saída GPIO
80             sleep_us(int(T*tempo[i])) # Tempo entre um salto de amplitude e outro
81             print(int(T*tempo[i]))
82     gpioN[6].value(0) # Desativa a saída para 1,65Vcc
83
84 freq = 10
85 T = 1000000 / freq
86 # ----- Laço de acionamentos -----
87 while True:
88     gera_sinal(freq, 'seno') # Frequência e Sinal ('seno', 'binSerra' e 'binTria')

```

Fonte: autoria própria, 2024.

11 CONCLUSÃO

O objetivo do projeto foi alcançado com êxito, tendo sido este o desenvolvimento de um gerador de sinais utilizando como plataforma a placa Raspberry Pi Pico.

O projeto visa gerar sinais analógicos a partir de sinais digitais codificados em números binários, operação na qual cada número binário corresponde a uma fração da tensão de referência que, no caso, é a tensão de 3,3 V.

A codificação em binário foi feita utilizando-se a linguagem de programação Python. Estes valores binários foram adicionados ao código escrito em Micropython que, por sua vez, foi adicionado à plataforma Raspberry Pi Pico.

As saídas GPIOs selecionadas acionam as entradas do circuito de condicionamento de sinal.

A saída do circuito de condicionamento do sinal fornece o sinal selecionado anteriormente na execução do código em Micropython.

A partir da programação inicial, todas as etapas, com exceção da última, forneceram respostas satisfatórias ao visualizarmos o sinal no osciloscópio sem a filtragem.

O sinal gerado na última etapa que não foi satisfatório, pode ser visto na figura 27. A partir disso, foi feita uma modificação na quantidade de amostras codificadas em números binários. E, assim, melhoramos o sinal senoidal antes da filtragem.

Ao analisarmos todo o processo do projeto e as imagens dos sinais dente de serra e triangular, concluímos que este gerador não fornece adequada para estes formatos de onda devido à quantidade de ruídos que acompanham estes sinais. Estes ruídos podem ser reduzidos de forma satisfatória em futuros projetos.

Já, o formato do sinal senoidal, principalmente após a filtragem, possui um formato senoidal praticamente perfeito, levando em conta as condições do projeto.

De todo modo, como um material didático para alunos do ensino médio, técnico e até mesmo para os iniciantes do curso superior, este projeto pode ser útil para:

- Estudos de processamento de sinais;

- Linguagens de programação Python e Micropython;

- Projetos de filtros ativos;

Análise e projetos com amplificadores operacionais, como por exemplo a substituição do amplificador LM741 por um LM324.

Este projeto também fornece uma direção para futuros projetos de geradores de sinais. No caso de um projeto semelhante, pode-se analisar a velocidade de comutação das saídas GPIOs utilizando a linguagem C e comparando a mesma com os resultados gerados a partir da linguagem Microphyton.

Pode-se também trabalhar no circuito de condicionamento de sinal, mudando-se os componentes ou incrementando mais filtros e, até mesmo, confeccionando uma placa de circuito impresso, o que proporcionaria uma redução de ruídos em relação à placa *protoboard*.

Este projeto também pode ser facilmente replicado devido à quantidade de tipos de componentes. O maior custo fica a cargo do Raspberry Pi Pico mas, que se adquirido em boa quantidade, pode-se ter o custo reduzido. Além de ser um microcontrolador com diversas aplicações e possui extensa informações na internet.

Este gerador também pode ser utilizado junto a TV-Box RK3229 ou outro modelo, para identificar novas conexões GPIO.

O projeto da TV-Box também possui extensas possibilidades que podem ser exploradas. Como, por exemplo, o sistema operacional Linux para que se possa executar as GPIOs em tempo real, independentemente da execução de outros serviços dos sistemas Linux. Ou seja, este projeto abre a possibilidade de se desenvolver soluções para reduzir a latência na execução da GPIO.

REFERÊNCIAS

BOYLESTAD, Robert L. Introdução à análise de circuitos / Robert L. Boylestad; revisão técnica Benedito Donizete Bonatto; tradução Daniel Vieira e Jorge Ritter. – 12. ed. – São Paulo: Pearson Prentice Hall, 2012.

BOYLESTAD, Robert L. Dispositivos eletrônicos e teoria de circuitos / Robert L. Boylestad, Louis Nashelsky; tradução Sônia Midori Yamamoto; revisão técnica Alceu Ferreira Alves. – 11. ed. – São Paulo: Pearson Education do Brasil, 2013.

DORF, Richarf C. Introdução aos circuitos elétricos / Richarf C. Dorf, James A. Svoboda; Tradução e revisão técnica: Ronaldo Sérgio de Biasi. - 8 Ed. - Rio de Janeiro: LTC, 2012.

FLEURY, Cláudio Afonso. Quantização de sinais discretos. Disciplina: Processamento digital de sinais. Aula: Amostragem e Quantização. Arquivo: Quantizacao.ipynb [Arquivo lido no Jupyter Notebook 6.5.4]. Brasil. Goiânia – GO, 2024.

HAYKIN, Sinais e Sistemas / Simon Haykin, Barry Van Veen; Tradução: José Carlos Barbosa dos Santos; Bookman, Porto Alegre, 2001.

LATHI, B. P. Sinais e sistemas lineares [recurso eletrônico] / B. P. Lathi; tradução Gustavo Guimarães Parma. – 2. ed. – Dados eletrônicos. – Porto Alegre: Bookman, 2008

MICROPYTHON, Documentação micropython: <<https://docs.micropython.org/en/latest/index.html>>. Acesso em: 24 junho de 2024.

NUMPY, Documentação Numpy: <<https://numpy.org/doc/stable/index.html>>. Acesso em: 24 junho de 2024.

OPPENHEIM, Alan V. Sinais e sistemas / Alan V. Oppenheim, Alan S. Willsky com S. Hamid Nawab; tradução Daniel Vieira; revisão técnica Marcio Eisencraft; Maria D. Miranda. – 2. Ed. - São Paulo: Pearson Prentice Hall, 2010.

OPPENHEIM, Alan V. Processamento em tempo discreto de sinais / Alan V. Oppenheim, Ronald W. Schafer; tradução Daniel Vieira; revisão técnica Marcio Eisencraft e Maria D. Miranda. – 3. ed. – São Paulo: Pearson Education do Brasil, 2012.

PERTENCE, Eletrônica analógica: amplificadores operacionais e filtros ativos: teoria, projetos, aplicações e laboratório / Antonio Pertence Júnior. -- 6ª Ed. -- Porto Alegre: Bookman, 2003.

PYTHON, Documentação Python: <<https://www.python.org/doc/>>. Acesso em: 24 junho de 2024.

RASPBERRY, Pi. Documentação Raspberry Pi:
<<https://www.raspberrypi.com/documentation/microcontrollers/>>. Acesso em: 24
junho de 2024.

SADIKU, Fundamentos de circuitos elétricos [recurso eletrônico] / Charles K.
Alexander, Matthew N. O. Sadiku; tradução: José Lucimar do Nascimento; revisão
técnica: Antônio Pertence Júnior. – 5. ed. – Dados eletrônicos. – Porto Alegre:
AMGH, 2013.

SADIKU, Matthew N. O. Análise de circuitos elétricos com aplicações [recurso
eletrônico] / Matthew N. O. Sadiku, Sarhan M. Musa, Charles K. Alexander; tradução:
Luiz Carlos do Nascimento, Warley de Sousa Sales; revisão técnica: Antonio
Pertence Júnior. – Dados eletrônicos. – Porto Alegre: AMGH, 2014.

TOCCI, Sistemas digitais: princípios e aplicações, 10ª edição / Ronald J. Tocci, Neal
S. Widmer, Gregory L. Moss; tradução: Cláudia Martins; revisão técnica: João
Antonio Martino. -- 10ª ed. -- São Paulo: Pearson Prentice Hall, 2007.

APÊNDICE

APÊNDICE A – Relatório 2 do projeto de trabalho de conclusão de curso 2023-2 -
Identificando e utilizando portas GPIO da TB-Box descaracterizada.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE GOIÁS
CAMPUS GOIÂNIA
DEPARTAMENTO DE ÁREAS ACADÊMICAS IV
COORD. DO BACHARELADO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

RELATÓRIO 2 DO PROJETO DE TRABALHO DE CONCLUSÃO DE CURSO 2023-
2

IDENTIFICANDO E UTILIZANDO PORTAS GPIO
DA TV-BOX DESCARACTERIZADA

ÁREA DO CONHECIMENTO – ENGENHARIA DE CONTROLE E AUTOMAÇÃO
SUB-ÁREA – SISTEMAS ELETRÔNICOS DE MEDIDAS E DE CONTROLE

ALUNO: MATEUS MORAIS AGUIRRE

ORIENTADOR: PROF. DR. CLÁUDIO AFONSO FLEURY

GOIÂNIA
NOVEMBRO-2023

ÍNDICE

1 INTRODUÇÃO	68
2 DECODIFICANDO O NÚMERO DO GPIO	70
3 DISPONIBILIDADE DE CADA GPIO	71
4 IDENTIFICANDO A GPIO NA PLACA DA TV-BOX.....	73
5 CONEXÕES EXTERNAS DAS GPIOs.....	78
6 CODIFICANDO O ACESSO A GPIO ATRAVÉS DO PYTHON + SHELL SCRIPT	81
7 APLICAÇÃO PRÁTICA DA GPIO	84
8 CRIANDO UM SHELL SCRIPT E UM SCRIPT PYTHON A PARTIR DO TERMINAL LINUX	87
9 REFERÊNCIAS	88

1 INTRODUÇÃO

1.1 Apresentação do relatório e objetivos

O objetivo deste relatório é apresentar o processo de identificação das conexões GPIO (General Purpose Input/Output) na placa da TV-BOX RK3229 descaracterizada e apresentar um exemplo de aplicação das GPIO como entrada (input) e saída (output). Este processo foi dividido em 6 etapas. Sendo resumidamente como:

1.1.1 Decodificando o número do GPIO

O processo de identificação na placa iniciou-se com a identificação do número codificado (identificador) do GPIO no Datasheet e a quantidade de GPIOs do processador RK3229, que são 128 (0 a 127) GPIOs.

1.1.2 Disponibilidade de cada GPIO

Na etapa seguinte, scripts em Python e Shell foram utilizados no terminal Linux da TV-Box para exportar cada porta GPIO e configurá-la como entrada ou saída. A finalidade desta etapa era determinar a disponibilidade de cada pino para o experimento. Após a execução dos scripts, constatou-se que 17 GPIOs apresentaram comportamentos inesperados, indicando que são provavelmente reservadas para funções internas do sistema. Diante desse resultado, a análise foi focada nas 111 GPIOs que responderam conforme o esperado.

1.1.3 Identificando a GPIO na placa da TV-Box

Esta etapa teve por objetivo localizar fisicamente na placa da TV-Box, cada uma das 111 GPIOs disponíveis. Para isso, um sinal elétrico pulsado foi gerado em cada GPIO, utilizando um LED de 5 mm como indicador visual. O cátodo do LED foi conectado a um resistor de 1 k Ω e ao GND da placa, enquanto o ânodo foi utilizado para sondar os pontos de teste correspondentes a cada GPIO na placa, permitindo a visualização do acendimento do LED quando a GPIO estava ativa.

1.1.4 Identificando a GPIO na placa da TV-Box

No capítulo 7, foi realizada uma demonstração prática da aplicação da TV-Box como plataforma para o controle de dispositivos de maior potência. Para isso, foram utilizadas duas GPIOs configuradas como entradas digitais (uma para o botão e outra para o LDR) e uma GPIO configurada como saída digital para controlar um relé de estado sólido. O relé, por sua vez, era responsável por comandar o acionamento de uma lâmpada de 220 VAC. O experimento consistiu em acender a lâmpada ao pressionar o botão ou ao detectar uma variação na luminosidade ambiente através do LDR.

2 DECODIFICANDO O NÚMERO DO GPIO

Ao analisar o Datasheet do processador RK3229 da Rockchip, verificou-se que há 128 portas GPIO (0 a 127). A identificação destas portas não segue o formato padrão: gpio0, gpio1, gpio2, ... gpio127 que é o formato de acesso a GPIO através do Shell do Linux. No Datasheet as GPIOs estão codificadas no formato: GPIO_D1 que é a conexão GPIO 25. Para converter o formato do identificador da GPIO, seguiu-se o processo:

O formato do código no Datasheet é: $GPIO(X_1)_{-}(X_2)(X_3)$

A decodificação:

Número do $GPIO = 32 * X_1 + 8 * X_2 + X_3$.

Onde X_2 , pode assumir os valores: $A = 0, B = 1, C = 2$ e $D = 3$.

Exemplo: $GPIO_D1 = GPIO(32 * 0 + 8 * 3 + 1) = GPIO25$.

No processo inverso, para obtermos o código GPIO a partir do número, fazemos a divisão inteira por 32 para obtermos X_1 , e divisão inteira por 8, para obtermos X_2 .

Exemplo: Obtendo o código do $GPIO70$.

$$X_1 = \frac{numGPIO}{32} = \frac{70}{32} = 2,187 = \text{Inteiro: } X_1 = 2$$

$$X_2 = \frac{numGPIO - 32X_1}{8} = \frac{70 - 32 * 2}{8} = 0,75 = \text{Inteiro: } X_2 = 0 = A$$

$$X_3 = numGPIO - 32 * X_1 - 8 * X_2 = 70 - 32 * 2 - 8 * 0 = 6$$

Assim: $GPIO70 = GPIO2_A6$.

3 DISPONIBILIDADE DE CADA GPIO

3.1 Acessando a GPIO a partir do terminal Linux

Ao pesquisar na internet sobre acesso a GPIO a partir do buscador Google, aparecem muitas informações de acesso a GPIO do Raspberry Pi e do Arduino, utilizando as linguagens C e Python.

Em um primeiro momento, para acessar as GPIOs da TV-Box, foram utilizados os mesmos métodos e bibliotecas utilizados na plataforma Raspberry Pi Pico estudado anteriormente. Devido às diferenças de plataformas Raspberry e TX2 RK3229, ocorreram erros de incompatibilidade durante a execução dos programas em Python.

Foi pesquisado sobre como acessar uma GPIO diretamente a partir do sistema Linux. Através desta pesquisa, ficou-se conhecendo a linguagem Shell Script que pode ser utilizada através de um script salvo na extensão *.sh* quanto diretamente no terminal Shell do Linux. É uma linguagem pouco conhecida por isso a necessidade de um estudo mais aprofundado nessa linguagem. De início, os acessos a GPIOs da TV-Box começaram pelo sistema SYSFs, onde as GPIOs são acessadas a partir do diretório: ***/sys/class/gpio***.

Passou-se a utilizar a linguagem Shell Script com a linguagem Python. Com o Linux na TV-Box, a linguagem Python oferece mais possibilidades que a linguagem MicroPython utilizada no Raspberry Pi Pico.

3.2 Acessando a GPIO a partir do Python + Shell Script

Utilizando um script em Python, passou-se a acessar através da linguagem Shell Script, os diretórios de cada GPIO. O processo consistia em exportar cada GPIO, configurar a direção (entrada ou saída) e, posteriormente, alterar o seu valor lógico.

A partir do diretório: ***cd /sys/class/gpio***, foram exportadas as 128 GPIOs (de 0 até 127) do processador RK3229. Como exemplo, foi utilizada a porta *gpio5*, para exportar. Para isto, foi executado o comando: ***cd /sys/class/gpio;echo 5 > export***.

Passou-se ao processo de alterar a direção (*int/out*) da GPIO. Para direção *out*, foi utilizado o comando: ***cd /sys/class/gpio/gpio5;echo out > direction***.

E para a direção *in*, o comando: ***cd /sys/class/gpio/gpio5;echo in > direction***.

Ao tentar alterar a direção da porta gpio15, a TV-Box desliga indicando que esta porta pode ser de uso exclusivo da TV-Box. Portanto, ela foi deixada de lado das investigações.

A alteração do valor lógico (*value*) via código de comando, só é possível quando a porta está na direção de saída (*output*). Caso esteja na direção de entrada (*input*), o valor lógico só pode ser alterado via comando externo a placa. Este processo é detalhado no exemplo de aplicação no capítulo 7.

O valor lógico (0 ou 1) é alterado utilizando o comando (*echo 0* ou *echo 1*): ***cd /sys/class/gpio/gpio5;echo 1 > value.***

Ao alterar o valor lógico da porta gpio30, a TV-Box trava e não responde a nenhum comando, tendo assim que desligar e religar a TV-Box. Esta porta também foi deixada de lado.

Uma outra forma de alterar o valor lógico de uma GPIO configurada como entrada é modificando seu estado ativo (*active low*). Ao definir o estado ativo como 0 ou 1, o valor lógico da GPIO é automaticamente invertido.

Desta etapa do processo, verificou-se as 17 portas GPIO que são de uso exclusivo da TV-Box, ou que estão com algum defeito não identificado que impede o acesso as mesmas. São elas: 15, 30, 35, 38, 43, 45, 49, 88, 90, 94, 95, 105, 106, 116, 117, 118 e 120.

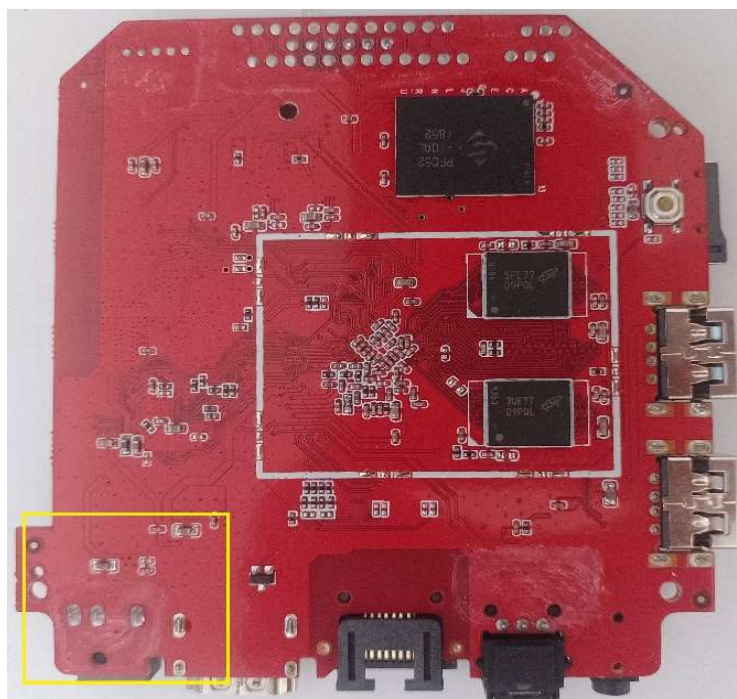
O processo de localização física das GPIOs na placa seguiu utilizando as demais 111 GPIOs que estão disponíveis no sistema.

4 IDENTIFICANDO A GPIO NA PLACA DA TV-BOX

A terceira etapa consistiu em identificar as 111 GPIO diretamente na placa da TV-BOX. Para isto, foi gerado um sinal de saída com uma frequência não específica na GPIO e que pode ser definida ao utilizar o código da figura 4 na linha 34. Este sinal aciona o LED de 5 mm utilizado como identificador visual, ligado em serie com um resistor (figura 3). O cátodo do LED foi conectado a um resistor de 1 k Ω e ao GND da placa, enquanto o ânodo foi utilizado para sondar os pontos de teste correspondentes a cada GPIO na placa, permitindo a visualização do acendimento do LED quando a GPIO estava ativa.

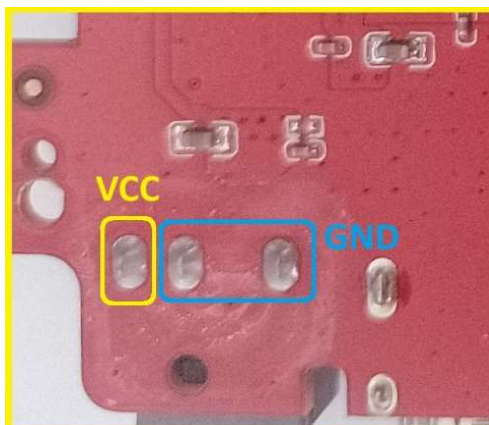
As identificações dos GNDs e ponto de tensão de 5 Vcc, foram feitas analisando os pontos de soldagens da conexão de alimentação externas da placa, como mostrado nas figuras 1 e 2.

Figura 1 – Vista inferior da placa.



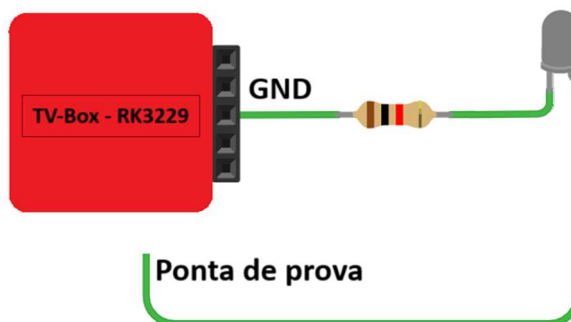
Fonte: autoria própria 2023.

Figura 2 – 5VCC e GND.



Fonte: autoria própria 2023.

Figura 3 – Circuito de verificação da conexão GPIO na placa.



Fonte: autoria própria 2023.

A melhor e mais eficiente forma de identificar fisicamente uma conexão GPIO na placa da TV-Box é utilizando um osciloscópio. Este processo foi executado em um momento que não havia a disponibilidade de um osciloscópio e que se mostrou eficiente.

O não uso do osciloscópio e o êxito na identificação física das GPIO utilizando a ponta de prova, demonstra que o custo financeiro pode ser reduzido. Alguém que não saiba utilizar, ou não tenha um osciloscópio também pode executar a mesma tarefa.

O script Python + Schell Script, utilizando para acessar a verificar a disponibilidade de cada GPIO (excluindo as 17 mencionadas anteriormente) é o arquivo **VerificGPIO.py** mostrado na figura 4.

Figura 4 – Script de verificação de disponibilidade da GPIO: **VerificGPIO.py**.

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Sat Jul  8 11:07:22 2023
4  @author: Mateus Moraes de Aguirre
5  """
6  import time
7  import os
8  import subprocess as subp
9
10 # Diretório de acesso a exportação das GPIOs
11 diretorio = '/sys/class/gpio'
12 # -----
13 def estado(numGPIO):
14     endereco = 'cd ' + diretorio + f'/gpio{numGPIO};cat '
15     temp1 = subp.Popen(endereco+'value', stdout=subp.PIPE, stderr=subp.PIPE, shell=True, universal_newlines=True)
16     p2, temp2 = temp1.communicate()
17     p2 = p2.split()
18     return print(f'GPIO{numGPIO} - Nível: {p2[0]}')
19 # -----
20 # Define as função de alteração do valor de Lógico 0 ou 1, de cada GPIO
21 def on(numGPIO):
22     os.system(f'echo 1 > {diretorio}/gpio{numGPIO}/value')
23
24 def off(numGPIO):
25     os.system(f'echo 0 > {diretorio}/gpio{numGPIO}/value')
26 # -----
27 numGPIO = int(input('Informe o número do GPIO: '))
28 acao = str(input('Exportar [e] e consultar ou importar [i]? '))
29 # -----
30 if acao == 'e' or acao == 'E':
31     os.system(f'echo {numGPIO} > {diretorio}/export') # Exporta o GPIO
32     os.system(f'echo out > {diretorio}/gpio{numGPIO}/direction')
33 # -----
34     freq = int(input('Informe a frequência: '))
35     T = 1/freq
36     duty = int(input('Informe o DutyCle [%]: '))
37     duty = (duty/100)
38     tempoON = duty * T
39     tempoOFF = T - tempoON
40 # -----
41     while True:
42         on(numGPIO)
43         estado(numGPIO)
44         time.sleep(tempoON)
45         off(numGPIO)
46         estado(numGPIO)
47         time.sleep(tempoOFF)
48 # -----
49 else:
50     os.system(f'echo {numGPIO} > {diretorio}/unexport') # Importa o GPIO

```

Fonte: autoria própria 2023.

Ao se executar o programa **VerificGPIO.py** e escolher a opção de exportar a GPIO, o programa executa comandos de liga e desliga de um LED em uma certa frequência. O tempo que o LED deve ficar ligado é definido pelo ciclo de trabalho (*duty cycle*). A ideia é fazer uma modulação por largura de pulso PWM (*Pulse Width Modulation*).

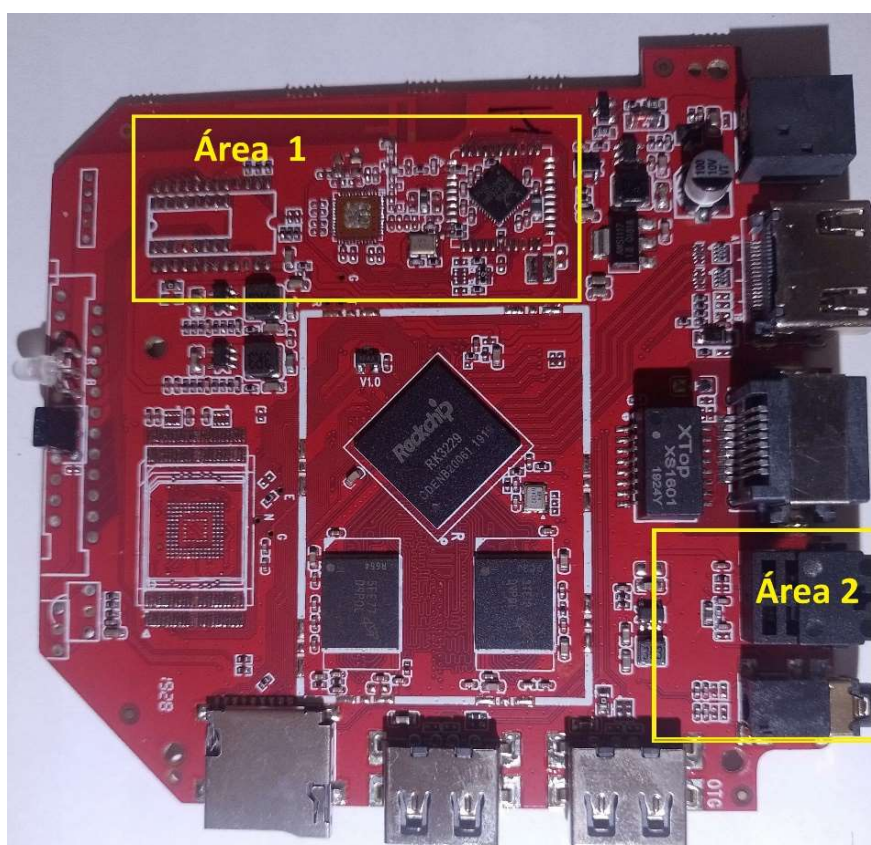
O programa solicita ao usuário que informe o número do GPIO que se deseja analisar (linha 27) e se deseja exportar ou importar (linha 28). No caso deste teste trabalho, a escolha foi exportar o GPIO. Depois solicita que o usuário informe a frequência (linha 34) e a porcentagem de *duty cycle* (linha 36).

O ciclo de acionamentos do LED somente se encerrará quando o usuário digitar o comando Ctrl+C,

Com o programa através do comando **while True**, acionando o LED da ponta de prova, a identificação física da GPIO na placa consiste em ir testando cada ponto de soldagem na placa. Ao conectar a ponta de prova no local que correspondente ao GPIO selecionado, o LED irá piscar conforme a frequência definida anteriormente.

Antes dos testes a placa foi fotografada para que cada GPIO identificada pudesse ser marcada na fotografia conforme a figura 6. Após alguns testes, foram identificados 10 pontos de conexões portas GPIOs. São elas: 2, 24, 25, 28, 40, 103, 107, 108, 109, 127. Estas GPIOs podem ser vistas nas áreas 1 e 2 demarcadas na figura 5 que por sua vez, são vistas em detalhe nas figuras 6 e 7.

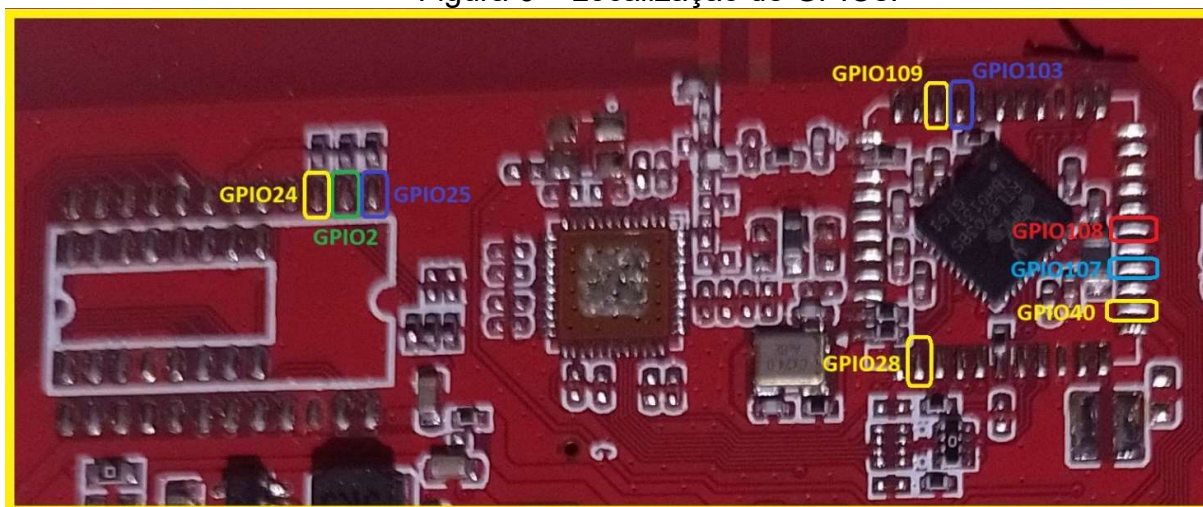
Figura 5 – Vista geral da placa com as áreas 1 e 2, das localizações da GPIO identificadas.



Fonte: autoria própria 2023.

A área 1 é mostrada na Figura 6, onde são identificadas as conexões GPIO: 2, 24, 25, 28, 40, 103, 107, 108 e 109.

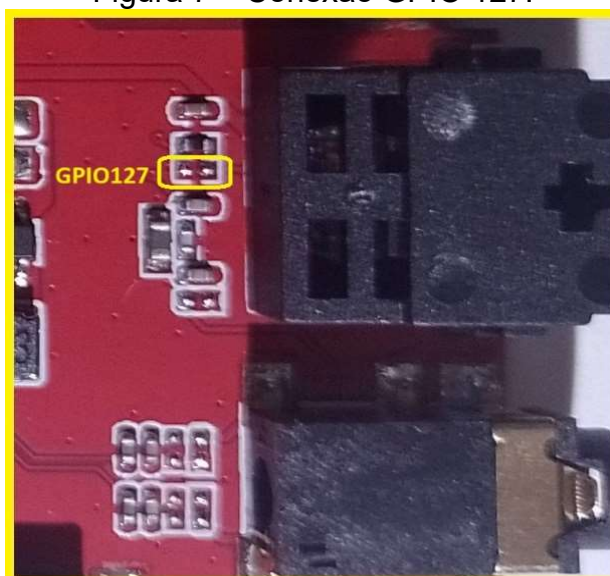
Figura 6 – Localização de GPIOs.



Fonte: autoria própria 2023.

A área 2, mostrada na Figura 7, mostra a localização da conexão GPIO 127.

Figura 7 – Conexão GPIO 127.



Fonte: autoria própria 2023.

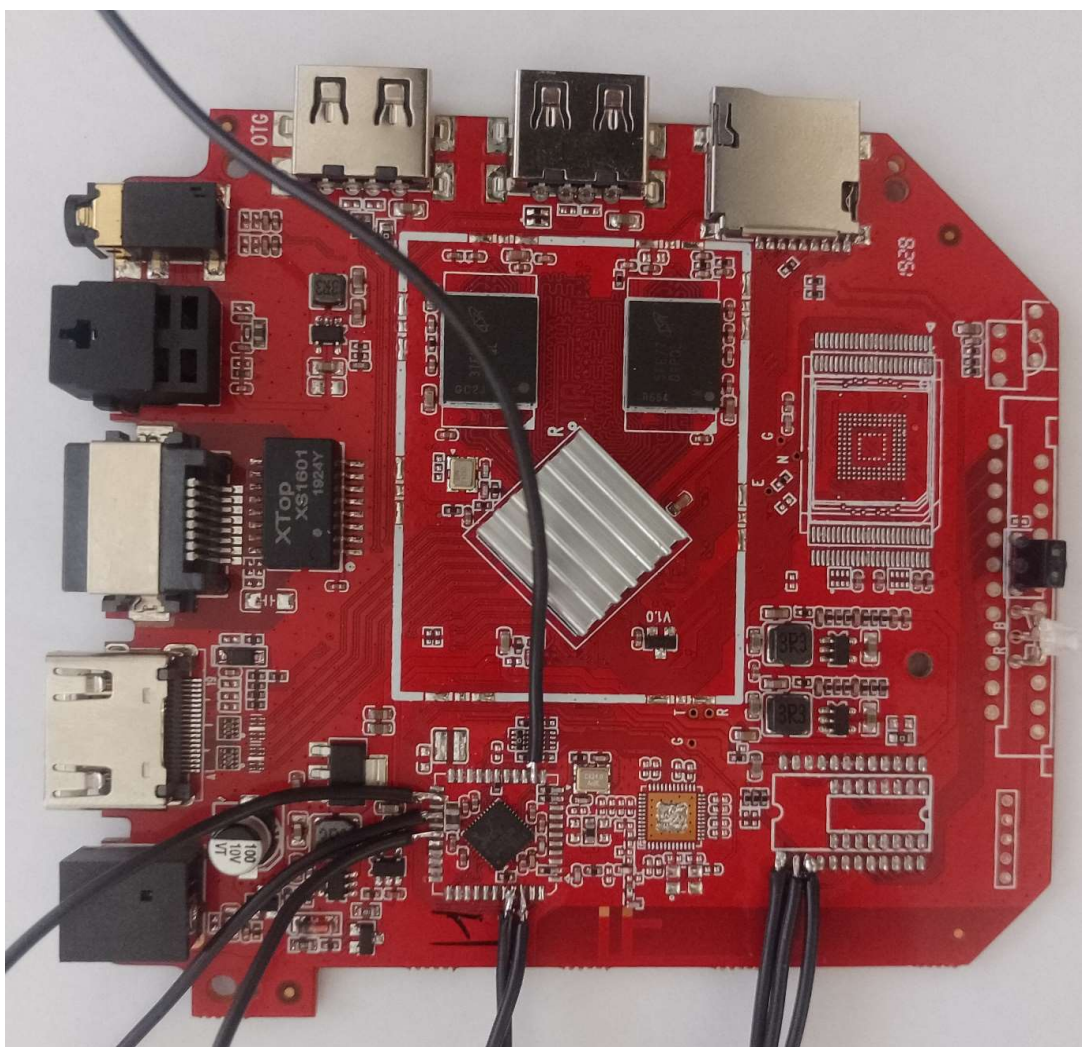
5 CONEXÕES EXTERNAS DAS GPIOS

Após a identificação das 10 portas GPIO na placa, a quarta etapa foi a do processo de soldagem de cabos em cada conexão GPIO identificada, ao barramento de conexões do tipo fêmea visto na figura 10.

Devido à ausência de um cabo de bitola adequada no momento, foi utilizado um cabo de bitola acima do adequado para este processo. Por isto, recomenda-se soldar um cabo AWG30 ou diâmetro menor. Recomenda-se também possuir habilidades de soldagens de placas de circuito.

A Figura 8 mostra os cabos soldados nas conexões GPIOs menos na conexão 127 que foi soldada, mas devido a fragilidade física do ponto de contato, a solda não suportou as forças externas e se rompeu. Ficando soldadas somente 9 portas GPIOs.

Figura 8 – Fios soldados nas conexões GPIO da placa.

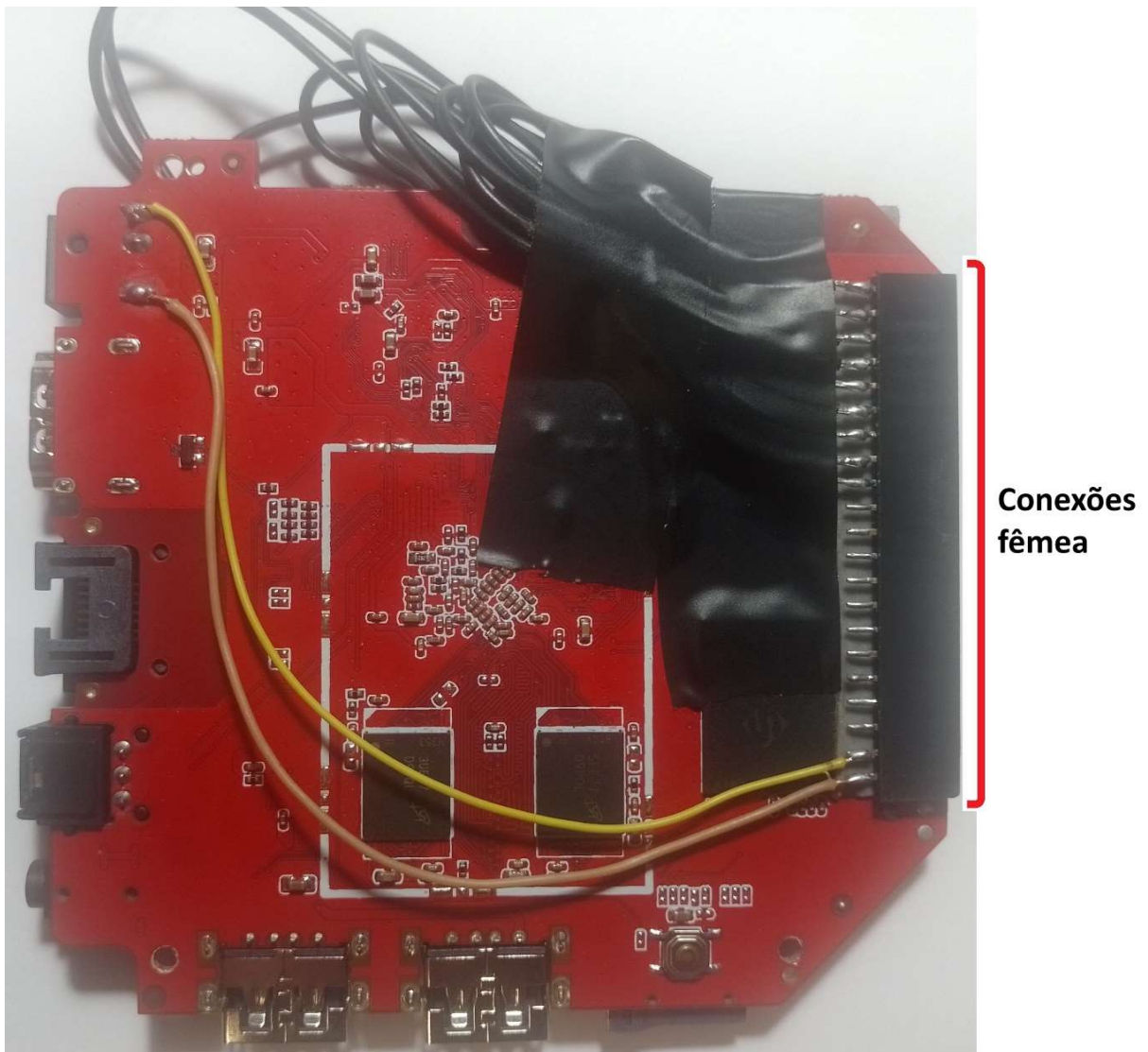


Fonte: autoria própria 2023.

Para evitar a quebra das demais conexões, uma fita de dupla face foi fixada sobre os pontos de soldagem. E uma fita isolante foi posta de forma que os cabos fiquem imobilizados sobre a placa. O detalhe do barramento de conexão fêmea e a passagem da fita isolante na placa são mostrados na figura 9.

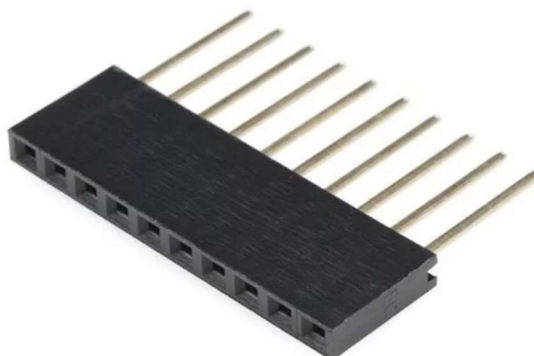
Desta forma, a placa está pronta para o uso das portas GPIO.

Figura 9 – Conexão fêmea e imobilização dos fios na placa.



Fonte: autoria própria 2023.

Figura 10 – Barramento fêmea utilizado na placa.



Fonte: <https://www.eletrogate.com/barra-de-pinos-macho-femea-empilhavel-10-pinos>, 2023.

6 CODIFICANDO O ACESSO A GPIO ATRAVÉS DO PYTHON + SHELL SCRIPT

A quinta etapa consistiu na criação de uma biblioteca em código Python + Shell Script para acessar e alterar as características de uma GPIO.

A biblioteca consiste em um script escrito na linguagem Python que acessa o diretório que se localizam as GPIOs, onde, através das funções chamadas, a GPIO é modificada através de comandos Shell Script.

O script Python + Shell Script é mostrado na Figura 11.

Figura 11 – Código Python + Shell Script de acesso e modificação das GPIO.

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Sun Aug 27 19:12:41 2023
4
5  @author: mateus
6  """
7  ...
8
9  O programa contém funções para exportar/importar um GPIO, alterar a sua direção e o seu valor lógico (0/1).
10 1 - As funções recebem as indicações de ações e as executam. São elas:
11     1.1 - Uma função que recebe um número GPIO para exportar (export/unexport). obs: unexport é desnecessário até
12           o momento.
13     1.2 - Uma função que altera a sua direção (out/in).
14     1.3 - Uma função que altera o seu valor lógico (0/1).
15 2 - Antes, do programa executar a ação de exportar verifica-se, se já está exportado. Se não tiver exportado,
16     executa a ação do item 3. Depois executa a ação do item 5.
17 3 - Exporta o GPIO (echo 'número do gpio' > export)
18 4 - Se já estiver exportado, executa a ação do item 5.
19 5 - Após exportado, o programa executa a função para alterar a sua direção (out/in).
20 6 - Se a direção informada for OUT, então é permitido alternar o seu valor lógico entre 0 e 1.
21 7 - Se a direção informada for IN, então a alteração do valor lógico não é permitida.
22
23 OBS:
24     1 - O comando Unexport é dispensável. Isto porque se o número do GPIO informado já estiver exportado, e não for
25         utilizado para alterar a sua direção ou valor lógico. Então não há necessidade de mudar seu estado.
26     2 - O valor lógico (0/1), só pode ser alterado se a direção do GPIO estiver em OUT. Caso esteja IN, ocorrerá
27         um erro.
28
29 Números de GPIO não utilizados
30 num = [15, 30, 35, 38, 43, 45, 49, 88, 90, 94, 95, 105, 106, 116, 117, 118 e 120]
31 GPIOs encontradas para uso: 2, 24, 25, 28, 40, 103, 107, 108, 109, 127
32 OBS: A solda da conexão da GPIO 127, se rompeu. Portanto, ela esta inabilitada até que a solda seja refeita
33 ...
34 import os
35 import subprocess as subp
36
37 # Diretório de acesso as porta GPIO's
38 diretorio = '/sys/class/gpio'
39
40 # -----
41 # Verifica o estado do GPIO: export ou unexport
42 def estado(numGPIO):
43
44     endereco = 'cd ' + diretorio + ' & ls ' # Acessa e retorna os dados do diretório
45     temp1 = subp.Popen(endereco, stdout=subp.PIPE, stderr=subp.PIPE, shell=True, universal_newlines=True)
46     p2, temp2 = temp1.communicate()
47     n = p2.split()
48
49     verifica = -1
50
51     # Verifica se o número informado já está exportado.
52     # Para isto, o número deve ser separado da string.
53     for c in range(len(n)):
54         if type(n[c]) == str and 'gpio' in n[c] and n[c][4:] != 'gpio':
55             numero = int(n[c][4:])
56             if numero == numGPIO:
57                 return 'exportado'
58                 verifica = numero
59
60     # Se o número informado não estiver exportado. Então está ação exporta.
61     if verifica == -1:
62         return 'exportar'
63 # -----

```

```

63 # -----
64 # Exporta o GPIO.
65 # A função recebe o número do GPIO e ação de entrada ou saída (input/out). Então o número é exportado e a ação é feita.
66 def GPIO(numGPIO, acao):
67     acao = acao.upper()
68     if estado(numGPIO) == 'exportar':
69         os.system(f'echo {numGPIO} > {diretorio}/export')
70     if acao == 'OUT':
71         os.system(f'echo out > {diretorio}/gpio{numGPIO}/direction')
72     if acao == 'IN':
73         os.system(f'echo in > {diretorio}/gpio{numGPIO}/direction')
74
75     # Caso o GPIO for definido como entrada. O seu valor active Low deve ser alterado para 1, caso esteja em zero.
76     endr = f'cat {diretorio}/gpio{numGPIO}/active_low'
77     t1 = subp.Popen(endr, stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True, universal_newlines=True)
78     p2, t2 = t1.communicate()
79     n = p2.split()
80     if n[0] == '0':
81         os.system(f'echo 1 > {diretorio}/gpio{numGPIO}/active_low')
82 # -----
83 # Define as função de alteração do valor de Lógico 0 ou 1, de cada GPIO
84 def on(numGPIO):
85     os.system(f'echo 1 > {diretorio}/gpio{numGPIO}/value')
86
87 def off(numGPIO):
88     os.system(f'echo 0 > {diretorio}/gpio{numGPIO}/value')
89 # -----
90 # A função valor(), verifica o estado do valor Lógico da GPIO.
91 # Esta função é utilizada para verificar o acionamento de um comando externo a TV Box.
92 def valor(numGPIO):
93
94     endr1 = f'cat {diretorio}/gpio{numGPIO}/active_low'
95     t1 = subp.Popen(endr1, stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True, universal_newlines=True)
96     p2, t2 = t1.communicate()
97     n = p2.split()
98
99     #os.system(f'cat {diretorio}/gpio{numGPIO}/value')
100     endr2 = f'cat {diretorio}/gpio{numGPIO}/value' # Acessa e retorna os dados do diretório
101     a1 = subp.Popen(endr2, stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True, universal_newlines=True)
102     b2, a2 = a1.communicate()
103     m = b2.split()
104
105     return m[0] # A saída retorna uma string
106 # -----

```

Fonte: autoria própria 2023.

Além de acessar o diretório **/sys/class/gpio**, o script contém as funções:

estado(<número do GPIO>) – Esta função local utilizada exclusivamente no próprio script. Ela recebe o número do GPIO que se deseja utilizar e verificar o seu estado atual, verificar se está exportado ou não. Caso não esteja exportado, então a ação é exportar este número. A verificação é feita para evitar erros de exportação.

GPIO(<número do GPIO>, <ação>) – Esta função é utilizada externamente a este script. É utilizada pelo usuário para acessar a GPIO ao informar o número e a ação que se deseja executar que no caso é a aplicação de entrada (*input*) ou saída (*output*) do GPIO.

A função **GPIO()** acessa a função **estado()**, caso o número não esteja exportado, então o comando da linha 69, executa a exportação (*export*) através de um comando Shell Script.

on(<número do GPIO>) – Quando o GPIO é utilizado como saída, esta função recebe o número GPIO e altera o valor lógico (*value*) para o nível lógico alto (1).

off(<número do GPIO>) – Esta função executa o comando inverso ao função

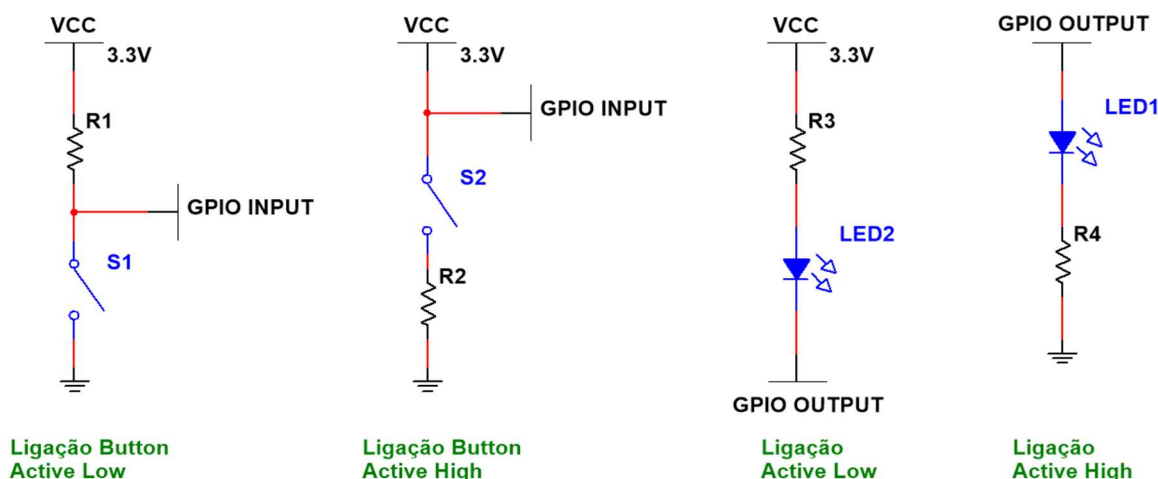
on(). Ela altera o nível lógico para baixo (0).

valor(<número do GPIO>) – Quando uma porta GPIO está configurada para entrada (*input*) de sinal, através da função **GPIO()**, a função **valor()** faz a leitura do nível lógico (*value*) e retorna este valor (0 ou 1). O que vai ser feito desse valor depende do objetivo do programa escrito pelo usuário.

7 APLICAÇÃO PRÁTICA DA GPIO

Para o uso de microcontroladores como o Raspberry Pi e o Arduino, temos dois tipos de ligações para entrada e saída de sinal. São elas a ligação do tipo: **Active Low** e **Active High**, como mostrado nos circuitos da Figura 12.

Figura 12 – Ligações Active Low e Active High.



Fonte: autoria própria 2023.

Para o uso da TV-Box RK3229 como um microcontrolador utilizamos as mesmas formas de ligações. Sendo ligações de entrada do sinal do tipo **Active Low** e as ligações para saída de sinal do tipo **Active High**.

Na sexta etapa foi realizada uma demonstração prática da aplicação da TV-Box como plataforma para o controle de dispositivos de maior potência. Para isso, foram utilizadas duas GPIOs configuradas como entradas digitais (uma para o botão e outra para o LDR) e uma GPIO configurada como saída digital para controlar um relé de estado sólido. O relé, por sua vez, era responsável por comandar o acionamento de uma lâmpada de 220 VAC. O experimento consistiu em acender a lâmpada ao pressionar o botão ou ao detectar uma variação na luminosidade ambiente através do sensor foto resistor (LDR).

A figura 13 mostra o circuito de acionamento da lâmpada. A figura 14, mostra o script *ligaLampada.py*, que contém a lógica de acionamentos e está instalado na TV-Box. Este programa importa a biblioteca *gpioRK.py* para acessar as suas funções, descritas anteriormente no capítulo 6.

O circuito possui os seguintes componentes:

- 1 – TV-Box RK2339;
 - 1 – Relé de estado sólido SSR-40 DA. Tensão de acionamento 3-32 VDC.
- Tensão acionada 24-380 VCA, 40 A;
- 1 – Lâmpada LED de 220 VCA – 9 W;
 - 1 – Botão push button;
 - 1 – Sensor foto resistivo LDR;
 - 2 – Resistor de 1 k Ω .

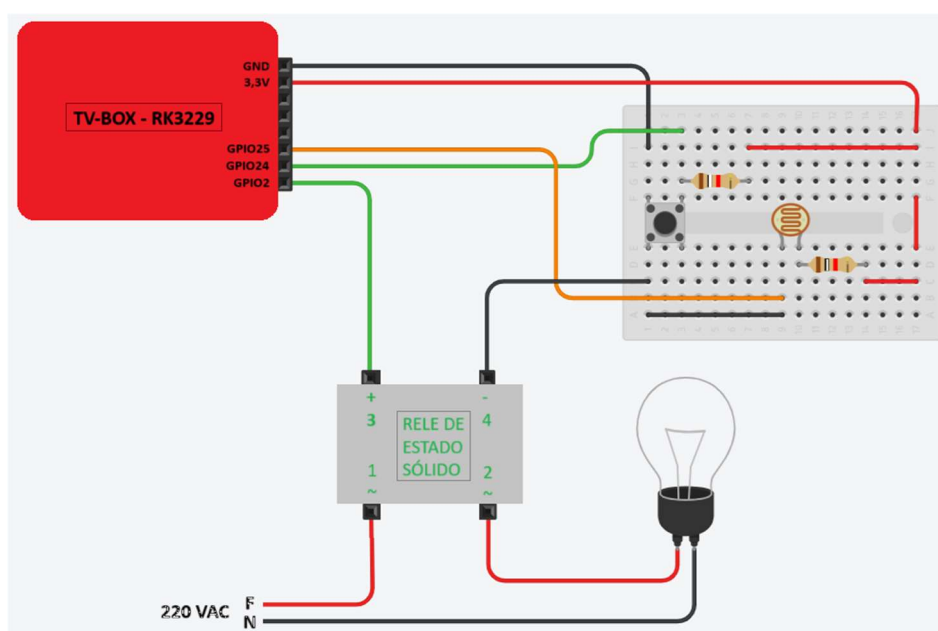
O acionamento do circuito acontece da seguinte forma:

Caso o ambiente esteja com luminosidade, o sensor LDR irá deixar passar corrente. O valor lógico na entrada da *GPIO 25* será de nível alto (1). Então a lâmpada não irá ligar. Mas, caso o botão seja acionado o nível lógico na entrada *GPIO 24* será alto (1), e assim a lâmpada irá ligar.

Caso o ambiente esteja sem luminosidade. Então, no sensor LDR não terá corrente elétrica circulando. O valor lógico na entrada 25 será baixo (0). E assim a lâmpada ficará ligada até que o ambiente volte a ter luminosidade natural.

A lâmpada não está ligada diretamente a saída *GPIO 2* mas sim ao relê de estado sólido que, por sua vez, é controlado pelo sinal da porta *GPIO 2*.

Figura 13 – Circuito de acionamento de uma lâmpada.



Fonte: autoria própria 2023.

Figura 14 – Lógica de acionamento da lâmpada.

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Sat Nov 18 17:29:55 2023
4
5  @author: Mateus Moraes de Aguirre
6  """
7
8  Este programa tem o objetivo de demonstrar algumas funções do uso da TV Box RK3229, modificada para o uso das
9  portas GPIO.
10 A lógica de controle está neste script. E a lógica de acesso a GPIO da TX Box, está na biblioteca gpioRK
11
12 As portas disponíveis são: 2, 24, 25, 28, 40, 103, 107, 108 e 109.
13
14 Este programa controla o acionamento de uma lâmpada LED.
15 O controle é feito através de um botão liga/desliga e um sensor de luz LDR.
16 Caso ambiente esteja com luz, o sensor LDR vai estar acionado e a lâmpada vai estar desligada. Mas, caso o botão
17 seja pressionado, a lâmpada irá ligar por tempo programado.
18 Caso o LDR não esteja acionado, a lâmpada ficará ligada até que o LDR seja acionado novamente.
19 O botão só pode ser acionado uma certa quantidade de vezes. Caso esceda, o programa irá terminar.
20 """
21
22 from gpioRK import * # Biblioteca de acesso as GPIO na TV box
23 import time
24
25 # Definindo o nome para cada conexão GPIO
26 led1 = 2 # O Led 1 está, conectado a porta GPIO 2
27 botoa1 = 24 # O botão 1 está, conectado a porta GPIO 24
28 sensor1 = 25 # O sensor 1 (DLR), está conectado a porta GPIO 25
29
30 # Definindo a função de cada porta GPIO (entrada ou saída)
31 GPIO(led1, 'out') # Led acionado como saída
32 GPIO(btoa1, 'in') # Botão acionado como entrada
33 GPIO(sensor1, 'in') # Sensor acionado como entrada
34
35 # Rotina para entrada e saída dos sinal
36 cont = 0
37 while True:
38     if valor(sensor1) == '0': # Verifica se o sensor está desligado. Caso esteja, o Led é ligado
39         on(led)
40     elif valor(btoa1) == '1': # Verifica o acionamento do botão para ligar o Led
41         time.sleep(0.4) # Tempo de espera do contato fechado
42         on(led1) # Liga o Led
43         time.sleep(0.8) # Liga o Led por um determinado tempo
44         cont = cont + 1 # Conta a quantidade de acionamentos do botão
45     else:
46         off(led1) # Desliga o Led, caso nenhuma ação anterior for executada
47     if cont == 4: # Encerra o programa após a quantidade de acionamentos do botão
48         off(led1)
49         break
50
51 print('FIM DO TESTE!\n')

```

Fonte: autoria própria 2023.

8 CRIANDO UM SHELL SCRIPT E UM SCRIPT PYTHON A PARTIR DO TERMINAL LINUX

8.1 Criando um Shell Script

Para criar um Shell Script foram utilizadas as sequências de comandos descritas a seguir. A partir do terminal Linux, deve-se abrir um editor para escrever o script e já definir um nome e tipo de extensão do arquivo. Que neste caso a extensão é **.sh**. Estas ações são a mesma tanto para criar um script em Shell Script quanto em Python.

Para um Shell Script que acessa o diretório que contém as GPIO e exibe o seu conteúdo, o comando digitado é:

```
sudo nano nome_do_arquivo.sh
```

Com o editor aberto, o programa deve conter na primeira linha: ***#!/bin/bash***

```
#!/bin/bash
```

```
cd /sys/class/gpio
```

```
ls
```

Com o programa escrito, para salvar o arquivo utiliza-se a sequência de comandos: ***Ctrl + O*** depois ***Enter***.

Para sair utiliza-se: ***Ctrl + X***.

Para voltar novamente ao editor, utiliza-se novamente o comando:

```
# sudo nano nome_do_arquivo.sh.
```

Para executar o programa, caso esteja em outro diretório usa-se o comando:

```
chmod A+X nome_do_arquivo.hs.
```

Se o terminal estiver acessando o mesmo diretório que está salvo o script, basta digitar o comando nome do arquivo com sua extensão: ***nome_do_arquivo.sh***.

8.2 Criando um script Python

O processo para gerar, editar e salvar um script Python é semelhante ao processo para o Shell Script.

Digite o comando: ***sudo nano nome_do_arquivo.py***.

Estando no mesmo diretório do arquivo salvo, deve-se usar o comando com a versão do Python instalada: ***python3 nome_do_arquivo.py***.

9 REFERÊNCIAS

AUTODESK Tinkercad: <https://www.tinkercad.com/> Acesso em 19 de novembro de 2023.

Bash for Loop, o primeiro passo na automação no Linux:
<<https://imasters.com.br/desenvolvimento/bash-for-loop-primeiro-passo-na-automacao-no-linux>> Acesso em: 11 de julho de 2023.

Curso de Raspberry Pi Pico - Introdução e Hello World.:
<https://www.youtube.com/watch?v=_mzMmfH0Yfs&list=PL4LRmt3d7UxbA0op_JVq8ifW9U71BQOx3> Acesso em: 11 de julho de 2023.

CRIE SEU PRÓPRIO LINUX - TV Box:
<<https://www.youtube.com/watch?v=8Ch7nUBiQa0> > Acesso em: 11 de julho de 2023.

Conheça o Sysfs no Linux: <<https://www.certificacaolinux.com.br/conheca-o-sysfs-no-linux/>> Acesso em: 11 de julho de 2023.

Elder Vicente de Paulo Sobrinho – Instalação do Linux na TX2.:
<<https://colab.research.google.com/drive/12H454cBhw3xDux9VxbK3AEj3Q7kZd-gj?usp=sharing#scrollTo=rv1p0uRff4W9>> Acesso em: 11 de julho de 2023.

Firefly-RK3399 Installation: <<https://dev.intelrealsense.com/docs/firefly-rk3399-installation>> Acesso em: 11 de julho de 2023.

Gerando PWM na Raspberry Pi: <<https://embarcados.com.br/gerando-pwm-na-raspberry-pi/#:~:text=O%20clock%20do%20PWM%20da,base%20de%2019%2C2%20MHz.>> Acesso em: 11 de julho de 2023.

Installing MicroPython: <<https://problemsolvingwithpython.com/12-MicroPython/12.02-Installing-MicroPython/>> Acesso em: 11 de julho de 2023.

Introdução ao Shell Script no Linux: <<https://www.devmedia.com.br/introducao-ao-shell-script-no-linux/25778>> Acesso em: 11 de julho de 2023.

JUCA, Sandro Juca. Aplicações Práticas de sistemas embarcados Linux utilizando Raspberry Pi [recurso eletrônico] / Sandro Jucá e Renata Pereira, 1 ed. - Rio de Janeiro: PoD, 2018: < <http://sanusb.org/arquivos/embarcadoslinux.pdf>> Acesso em: 11 de julho de 2023.

MicroPython: <<https://micropython.org/>> Acesso em: 11 de julho de 2023.

MicroPython - class Pin – control I/O pins:

<<https://docs.micropython.org/en/v1.9.3/wipy/library/machine.Pin.html>> Acesso em: 11 de julho de 2023.

MicroPython for Kids – Button: <<https://www.coderdojotc.org/micropython/basics/03-button/>> Acesso em: 11 de julho de 2023.

OpenWrt – GPIO: <https://openwrt.org/docs/techref/hardware/port.gpio> Acesso em 18 de novembro de 2023.

Python IDE for beginners: <<https://thonny.org/>> Acesso em: 11 de julho de 2023.

PyRock - Python GPIO Library: <<https://wiki.radxa.com/News/2014/10/pyRock-python-gpio-library>> Acesso em: 11 de julho de 2023.

Raspberry Pi: <<https://www.raspberrypi.com/products/raspberry-pi-pico/>> Acesso em: 11 de julho de 2023.

Raspberry Pi Pico W - high-resolution pinout and specs:

<<https://mischianti.org/2022/09/16/raspberry-pi-pico-w-high-resolution-pinout-and-specs/>> Acesso em: 11 de julho de 2023.

RPi GPIO Code Samples: <https://elinux.org/RPi_GPIO_Code_Samples#sysfs> Acesso em: 11 de julho de 2023.

subprocess — Gerenciamento de subprocessos: <<https://docs.python.org/pt-br/dev/library/subprocess.html>> Acesso em: 11 de julho de 2023.

Testando o RASPBERRY PI PICO | Tutorial de configuração com MicroPython:

<<https://www.youtube.com/watch?v=UiyLg3YQR3s>> Acesso em: 11 de julho de 2023.

Usando GPIO e PWM no Linux Embarcado: <<https://embarcados.com.br/usando-gpio-e-pwm-no-linux-embarcado/>> Acesso em: 11 de julho de 2023.

Usando Node.js e o framework Express para acessar os pinos de GPIO da Colibri VF61 via Web: <<https://embarcados.com.br/node-js-e-express-colibri-vf61/>> Acesso em: 11 de julho de 2023.

Utilizando PWM para controlar LED no Raspberry Pi com Guizero:

<<https://tecnicas.com/utilizando-pwm-para-controlar-led-no-raspberry-pi-com-guizero/>> Acesso em: 11 de julho de 2023.

4 MicroPython - PWM (Modulación por ancho de pulso) | ESP8266 y ESP32:

<<https://www.youtube.com/watch?v=-OmWCgxDvLo>> Acesso em: 11 de julho de 2023.