UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ALYSSON DE LIMA PONCIANO

DESENVOLVIMENTO DE UM MÓDULO DIDÁTICO COM A PLACA FRDM-KL25Z

FORTALEZA

2022

ALYSSON DE LIMA PONCIANO

DESENVOLVIMENTO DE UM MÓDULO DIDÁTICO COM A PLACA FRDM-KL25Z

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.
Orientador: Prof. Dr. Fabrício Gonzalez Nogueira.

FORTALEZA
2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

P854d Ponciano, Alysson de Lima.
Desenvolvimento de um módulo didático com a placa FRDM-KL25Z / Alysson de Lima Ponciano. – 2022.
94 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2022.
Orientação: Prof. Dr. Fabrício Gonzalez Nogueira.

1. Módulo didático. 2. FRDM-KL25Z. 3. Circuitos digitais. 4. Circuitos analógicos. I. Título.
CDD 621.3

ALYSSON DE LIMA PONCIANO

DESENVOLVIMENTO DE UM MÓDULO DIDÁTICO COM A PLACA FRDM-KL25Z

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Aprovada em: __/__/____.

BANCA EXAMINADORA

_____
Prof. Dr. Fabrício Gonzalez Nogueira (Orientador)
Universidade Federal do Ceará (UFC)

_____
Prof. Dr. Luiz Daniel Santos Bezerra
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

_____
Eng. Bruno Luiz Faustino
Universidade Federal do Ceará (UFC)

A Deus, por Ser O Todo Poderoso que me concede tudo o que eu preciso. Meus pais, que são os meus principais investidores de amor, tempo e dedicação.

AGRADECIMENTOS

A Deus, por sempre estar comigo desde a minha criação e ter fornecido forças durante todo esse tempo. Onde toda honra e toda glória sejam dadas ao Seu Santo Nome.

Sou eternamente grato pela minha mãe Eunice, e meu pai Ailo por terem abdicado de viverem os seus sonhos para eu poder viver os meus sonhos.

Aos meus irmãos Ailo Filho e Carlos Eduardo, por todas as lições de vida e companheirismo.

Aos meus amigos/irmãos de igreja Airton Filho, Eliel, Gabrielle e Clara por todos os momentos maravilhosos que vivemos em nome de Jesus Cristo.

Dedico este trabalho ao meu grande amigo Orlando (in memorian), por ter sido um excelente amigo durante todo o tempo que estivemos juntos.

À minha querida amiga Ádria e meu amigo Ramon, por todos os instantes que passamos, seja nas dificuldades ou nas bonanças, mas não perdemos a amizade.

Às famílias dos meus amigos que me deram suporte e palavras de motivação, fica a minha gratidão.

À minha equipe Weslley Barata e Davi, por toda a dedicação, ajuda e risadas, principalmente nas execuções dos trabalhos.

Aos grupos, Grupo de Desenvolvimento Aeroespacial, Ramo Estudantil IEEE UFC Fortaleza, Laboratório de Eletrônica Digital, Grupo de Pesquisa em Automação, Controle e Robótica nos quais fizeram parte da minha construção acadêmica.

Ao Prof. Dr. Fabrício Gonzalez Nogueira, pela excelente orientação e paciência durante todo o trabalho.

Aos colegas da turma que compartilharam ajuda e conhecimento durante essa graduação.

RESUMO

Em suma, este trabalho apresenta a construção de um módulo didático que visa a melhoria continua do aprendizado e da produção de trabalhos científicos tendo como microcontrolador o chip KL25Z128VLK4 e a placa de desenvolvimento FRDM-KL25Z da NXP para realizar todo o processamento dos recursos disponibilizados. O módulo desenvolvido para o curso de Engenharia Elétrica da Universidade Federal do Ceará conta com diodos emissores de luz, botões, potenciômetros, display LCD, circuito condicionador de sinais ajustável, relé optoacoplado e a placa de circuito impresso que contou com boas práticas de construção para acomodar todos os componentes de forma adequada para minimizar os efeitos adversos causados pelos circuitos digitais e analógicos. Por fim, foi notado que após os testes realizados com apoio de diversos aplicativos como EasyEDA, PSIM, Tera Term, MCUXpresso e MCUXpresso SDK, o módulo didático proposto atende os requisitos para a utilização em diversas disciplinas devido a sua concepção que supera o uso restrito em Microcontroladores que atualmente conta com o microcontrolador PIC16F628A.

Palavras-chave: módulo didático; FRDM-KL25Z; circuitos digitais; circuitos analógicos.

ABSTRACT

Finally, this work presents the construction of a didactic module that aims at the continuous improvement of learning and production of scientific works using the microcontroller chip KL25Z128VLK4 and the FRDM-KL25Z NXP development board to perform all the processing of the available resources. The module developed for the Electrical Engineering course at the Federal University of Ceará has light-emitting diodes, buttons, potentiometers, LCD display, adjustable signal conditioning circuit, optocoupled relay and the printed circuit board that had good construction practices for properly accommodate all components to minimize the adverse effects caused by digital and analog circuitry. Finally, it was noticed that after the tests carried out with the support of several softwares such as EasyEDA, PSIM, Tera Term, MCUXpresso and MCUXpresso SDK, the proposed didactic module meets the requirements for use in several school subjects due to its design that overcomes the restricted use in Microcontrollers that currently has the PIC16F628A microcontroller.

Keywords: didactic module; FRDM-KL25Z; analog circuits, digital circuits.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DEE	Departamento de Engenharia Elétrica
UFC	Universidade Federal do Ceará
LED	Light-Emitting Diode
LCD	Liquid Crystal Display
LQFP	Low Profile Quad Flat Pack
ARM	Advanced RISC Machine
DMA	Direct Memory Access
ADC	Analog Digital Converter
DAC	Digital Analog Converter
SPI	Serial Peripheral Interface
PWM	Pulse Width Modulation
TPM	Timer PWM Module
PIT	Periodic Interruption Timer
UART	Universal Asynchronous Receiver/Transmitter
RAM	Random Access Memory
GPIO	General Purpose Input Output
RTC	Real Time Clock
LPTMR	Low-Power Timer
I2C	Inter-Integrated Circuit
MTB	Micro Trace Buffer
BME	Bit Manipulation Engine
PCI	Placa de Circuito Impresso
CI	Circuito Integrado
GND	Ground
TH	Through Hole
DIP	Dual In-line Package
SOP	Small Outline Package
RTOS	Real Time Operating System
IDE	Integrated Development Environment
SDK	Software Development Kit
GCC	GNU Compiler Collection

GDB	GNU Debugger
AMPOP	Amplificador Operacional
USB	Universal Serial Bus

LISTA DE SÍMBOLOS

$	Real
%	Porcentagem
Ω	Ômega
©	Copyright
®	Marca Registrada
A	Ampere
R	Resistência
C	Capacitor
I	Corrente Elétrica
V	Tensão
Hz	Frequência
π	Pi
τ	Constante de tempo

SUMÁRIO

1 INTRODUÇÃO

A presença massiva de microcontrolador em todas as esferas da sociedade é unânime, pois isso é devido a integração do chip microprocessador com os diversos periféricos presentes nas placas em projetos de robótica, automação e controle. Assim, surge a necessidade de a universidade como agente propulsor sempre liderar as inovações em tecnologias para criação de novos produtos e a capacitação dos estudantes para atender as demandas solicitadas pelo mercado de trabalho.

Por meio disso, este trabalho envolve a criação de um módulo didático tendo como microcontrolador, o KL25Z128VLK, presente na placa de desenvolvimento FRDM-KL25Z da NXP *Semiconductors* para uso em diversas disciplinas oferecidas pelo Departamento de Engenharia Elétrica (DEE), com intuito de utilizar o conhecimento da linguagem C obtida em Programação Computacional para Engenharia oferecida para o curso de Engenharia Elétrica da Universidade Federal do Ceará (UFC).

Ademais, tal módulo vem para substituir ou complementar o atual microcontrolador de 8 *bits* PIC16F628A da empresa Microchip utilizado em Microprocessadores 1. Dado que, o PIC16F628A possui limitações de recursos que impedem o potencial criativo dos alunos para a criação de novos projetos que envolvam a indústria 4.0, como a utilização de RTOS e protocolos de comunicação diversos.

Assim, o projeto deste módulo concebe a liberdade criativa do aluno com um formato adaptativo da construção do leiaute no qual permite a interação e o pensar no desenvolvimento das atividades requeridas. Sendo uma placa integrada a componentes como Diodos Emissores de Luz (LED), *Display LCD* 16x2, opto acoplador, relé, botões, regulador de tensão, gerador de tensão de referência, potenciômetros e amplificadores operacionais permitem a execução de diferentes tipos de práticas nos laboratórios. Soma-se ainda, os recursos dos periféricos da placa de desenvolvimento FRDM-KL25Z, nos quais permitem atuações em várias disciplinas pela diversificação e poderio computacional.

No capítulo 2, serão explorados o conteúdo histórico, os recursos computacionais e um comparativo entre a FRDM-KL25Z, o PIC16F628A e o módulo didático com a KL25Z. Mostrando assim, o ganho educativo e acadêmico com a inserção desses novos integrantes.

No capítulo 3, será mostrada de forma detalhada a criação de cada função presente no módulo didático juntamente com os esquemáticos e a seleção dos componentes.

No capítulo 4, detalha-se a questão da programação e os aplicativos que são utilizados para realizar essa interface, além de mostrar um breve comparativo entre as

linguagens C e o Assembly que é usado no PIC16F628A.

No capítulo 5, mostra-se a junção do *software* e *hardware* com os testes executados individualmente de cada função.

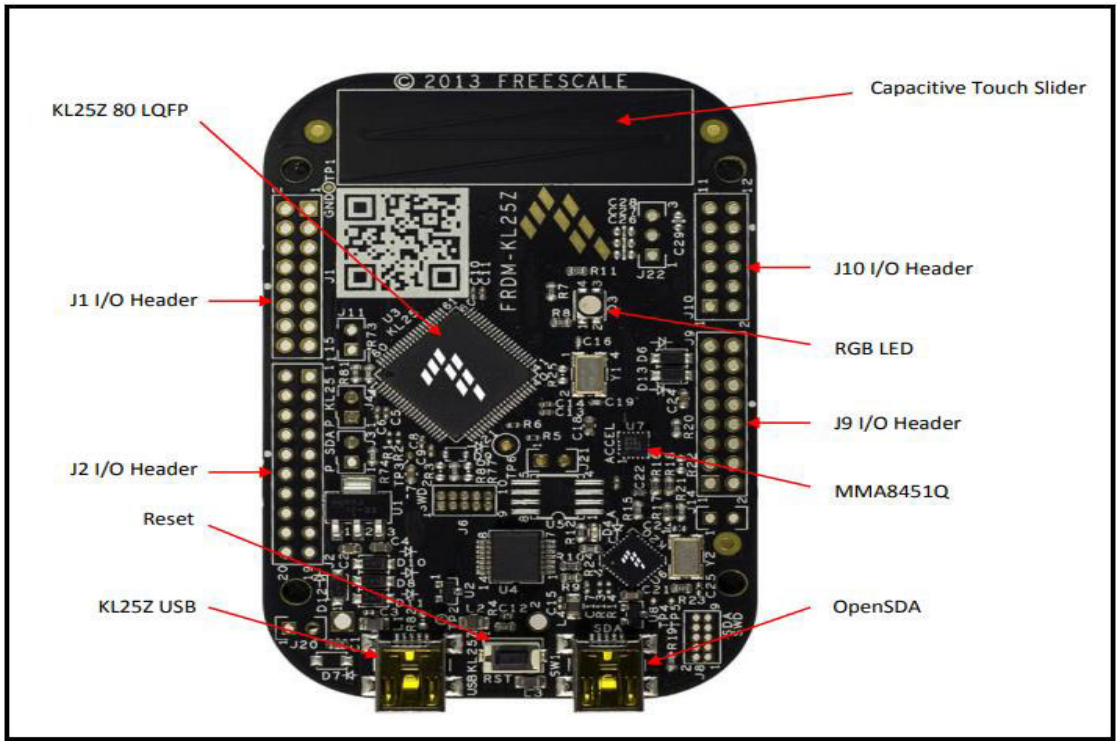
Por fim, conclui-se com a apresentação dos resultados do projeto desenvolvido com detalhes de melhorias e sugestões para futuros trabalhos.

2 PLACA DE DESENVOLVIMENTO FRDM-KL25Z

Em 2015, a NXP *Semiconductors* efetuou uma fusão com a Freescale por um negócio envolvendo 40 bilhões de dólares (SEMICONDUCTORS, 2015), tornando-se assim uma nova gerência sobre os negócios da Freescale e, atualmente, as documentações, aplicativos e outros produtos estão presentes em seu sítio eletrônico. A FRDM-KL25Z é uma placa de desenvolvimento da série de microcontroladores Kinetis L baseada em custo-benefício criada pela Freescale *Semiconductor*, cujo processador é o ARM Cortex-M0+ com frequência máxima de operação de 48 MHz.

Na Figura 1, é possível observar alguns dos recursos presentes na placa de desenvolvimento, como um sensor de toque capacitivo, os pinos de entrada e saída, *LED RGB*, o encapsulamento LQFP 80 do microcontrolador, um acelerômetro de 3 eixos MMA8451Q, botão de *reset* e duas entradas USB de alimentação e programação.

Figura 1 – Funções da KL25Z

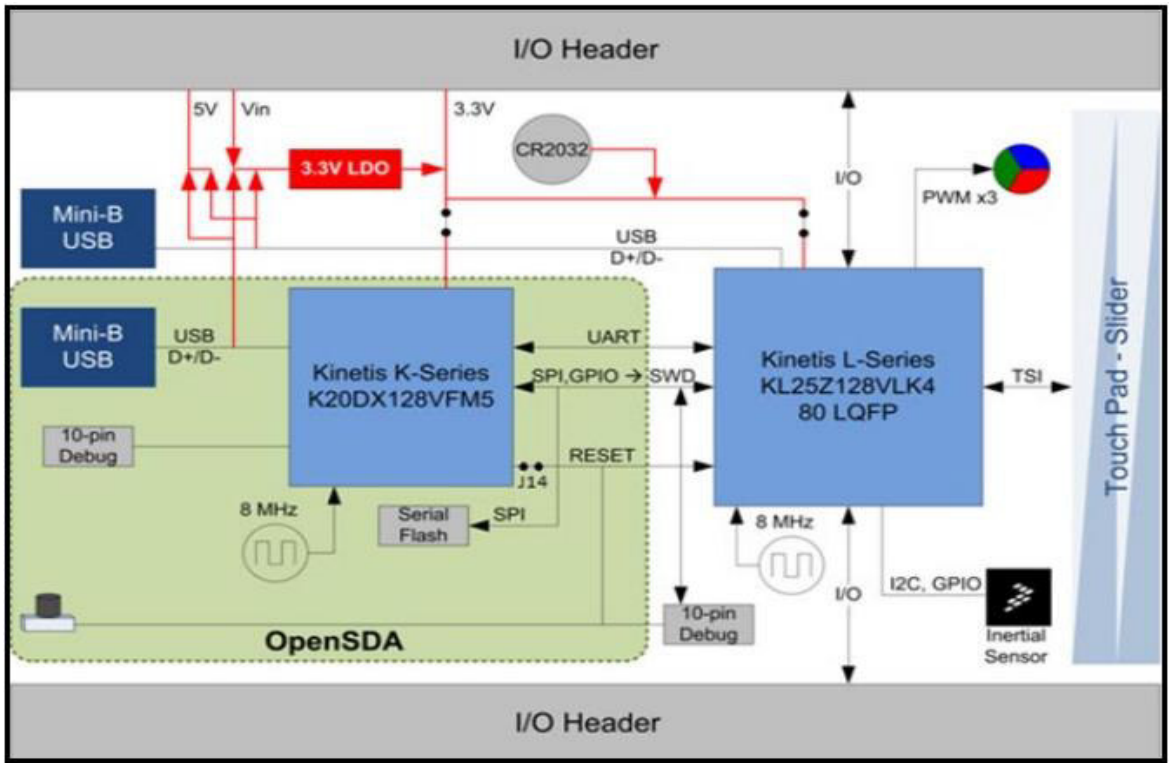


Fonte: adaptado do FRDM-KL25Z *User's Manual* (2013, p. 5).

Na Figura 2 é visto o diagrama de blocos da FRDM-KL25Z, onde pode ser visualizada a presença dos periféricos e recursos na placa de desenvolvimento. Tal dispositivo conta com o OpenSDA que é um mecanismo integrado de depuração e adaptador serial para se

comunicar com o computador via comunicação serial com intuito de programar, comunicar e alimentar.

Figura 2 – Diagrama de blocos da KL25Z



Fonte: adaptado do FRDM-KL25Z *User's Manual* (2013, p. 4).

2.1 Recursos e periféricos

Os recursos e periféricos disponibilizados pela FRDM-KL25z são os seguintes:

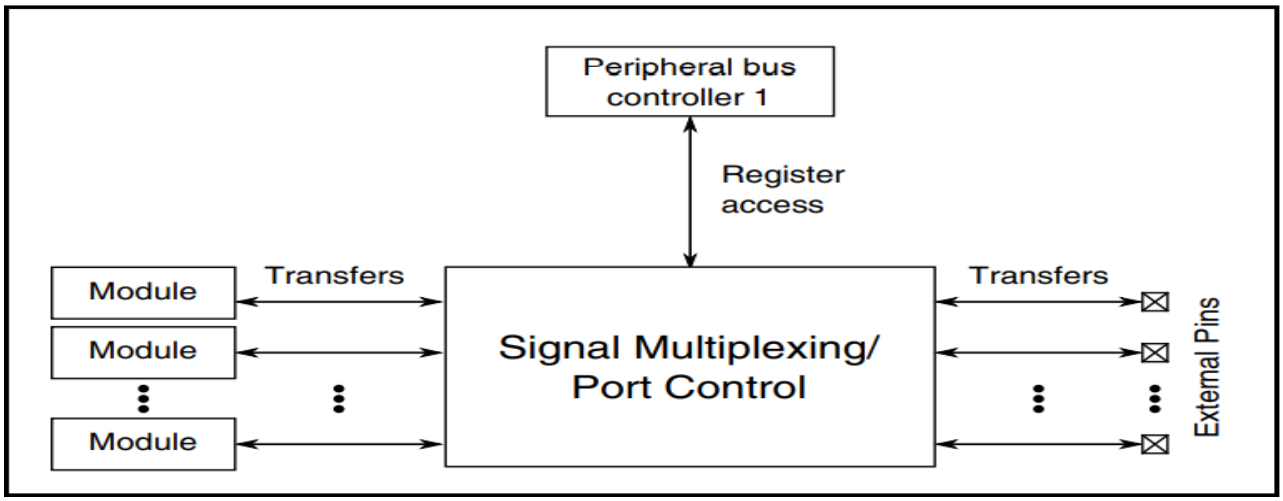
a) processador 32-*bit* ARM Cortex-M0+ de 48 MHz;
b) 128 KB de Memória Flash e 16KB de SRAM;
c) gerenciamento de potência com modos de controle;
d) controlador de acesso direto de memória (DMA);
e) *watchdog timer;*
f) conversor Analógico Digital (ADC) tipo aproximações sucessivas de 16 *bits;*
g) conversor Digital Analógico de 12 *bits* (DAC);
h) comparadores;
i) duas Seriais de Interface Periférica (SPI) de 8 *bits;*
j) um módulo de Modulação por Largura de Pulso/Temporizador (PWM/TPM) com 6 canais;

k) dois módulos de Modulação por Largura de Pulso/Temporizador com 2 canais;
l) dois canais de Temporizador de Interrupção Periódica (PIT).
m) relógio de Tempo Real (RTC);
n) temporizador de baixa potência (LPTMR);
o) dois módulos Circuito Inter-Integrado (I2C);
p) dois módulos padrões Transmissor-Receptor Universal Assíncrono (UART) e um de baixa potência.

Fora os recursos citados anteriormente, como o caso do *LED RGB*. Portanto, nota-se que seu uso é adequado para os laboratórios, salas de aula e em projetos.

Um ponto importante para se mencionar na arquitetura dessa placa é o fato que cada pino pode ser multiplexado para até 8 tipos de alternativas (SEMICONDUCTOR, 2012). Ou seja, tomando como exemplo o pino 1, que corresponde PTE0 como primeira alternativa, UART1_TX na alternativa 3, RTC_CLKOUT na alternativa 4, na alternativa 5 temos o CMP0_OUT e por fim temos na alternativa 6 o I2C1_SDA. Já na Figura 3, é possível entender a multiplexação de sinais, onde há a transferência do módulo para os pinos externos de acordo com o controle de portas. Tal controle fica nas mãos do usuário, logo basta escolher qual módulo será utilizado via *software* em cada pino de acordo com o Semiconductors (2013). Neste projeto de módulo, há amarrações a nível de *hardware* para utilizar os componentes didáticos, mas há a liberdade em vários pinos de desconectar a trilha para ser utilizado como outra função.

Figura 3 – Multiplexação de sinais



Fonte: adaptado do KL25 *Sub-family Reference Manual* (2012, p. 159).

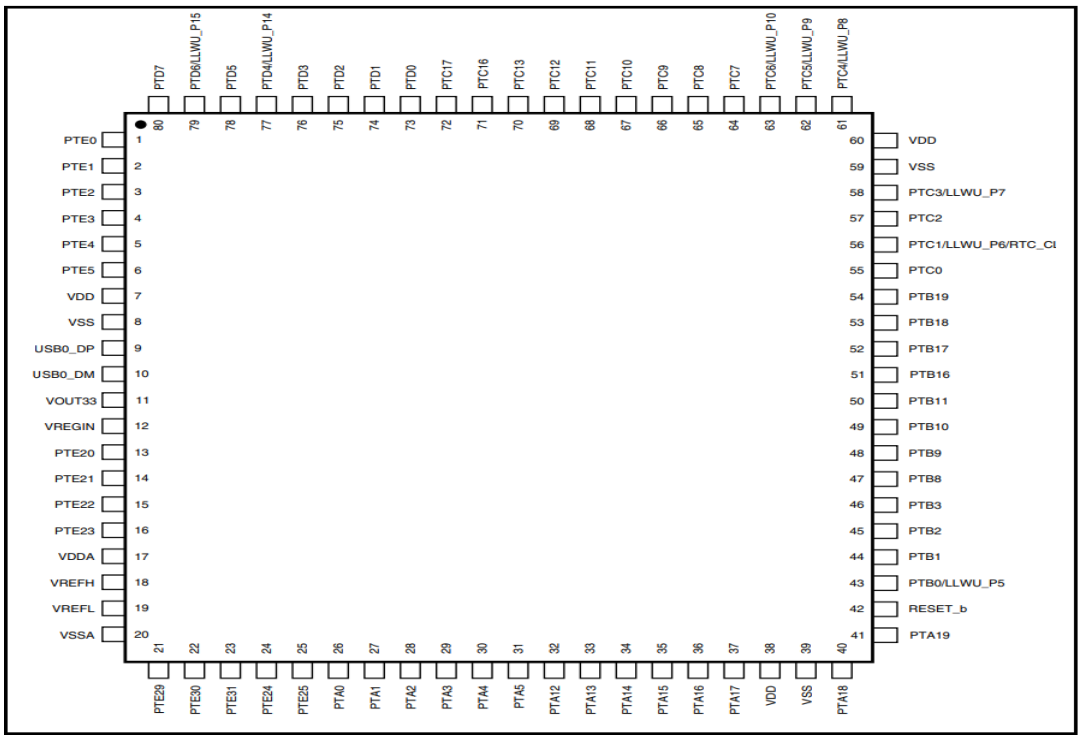
2.2 MKL25Z128VLK4

O chip MKL25Z128VLK4 é o *chip* microcontrolador da placa de desenvolvimento FRDM-KL25Z, o que causa confusão com alguns usuários do ramo da eletrônica. Pois, assim como o Arduino e o ESP32, eles são por definição placas de desenvolvimento que contém periféricos adicionais ao microcontrolador para oferecer ao usuário uma experiência mais simples de montagem e uso. Logo, no caso do Arduino você pode ter variação do ATmega dependendo de qual modelo será usado e no ESP-32 são vistos módulos de chips microcontroladores com funções de *bluetooth* e Wi-Fi.

Em relação ao MKL25Z128VLK4, é fabricado sobre um encapsulamento de 80 pinos do tipo *Low Profile Quad Flat* Pack (LQFP), com frequência de processamento de 48 MHz, memória flash de 128 KB, 16 KB de memória RAM e com limitações de temperatura de -40 a 105 °C.

Com isso, na Figura 4 é possível observar a disposição de cada pino do *chip* com a sua respectiva função. Justamente por seu encapsulamento ser do tipo quadrático, há 20 pinos em cada lado. Assim como, nota-se a existência de PORTA, PORTB, PORTC, PORTD e PORTE que possibilita o trabalho com diversos periféricos e funções presentes em cada pino.

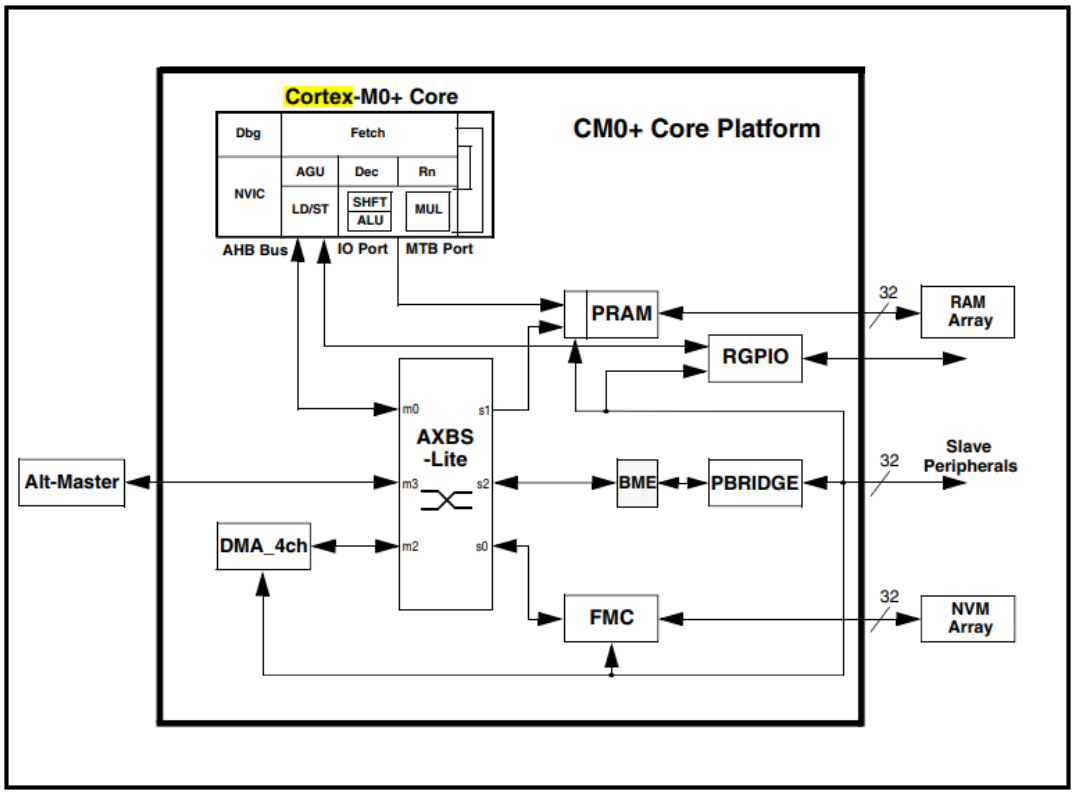
Figura 4 – Microcontrolador MKL25Z128VLK4



Fonte: adaptado do KL25 *Sub-family Reference Manual* (2012, p. 165).

E para complementar, dentro do *chip* MKL25Z128VLK4 há um microprocessador que é responsável por realizar todo o processamento de dados. Logo abaixo, na Figura 5 é visto o diagrama de blocos do Cortex-M0+ que informa o relacionamento dos blocos das funções de alocação de memória, periféricos escravos, interrupções, pinos de entrada e saída de uso geral (GPIO) e diversos outros que são essenciais para o funcionamento adequado do microprocessador.

Figura 5 – Diagrama de blocos do Cortex-M0+



Fonte: adaptado do KL25 *Sub-family Reference Manual* (2012, p. 272).

Para Semiconductors (2012), todos os participantes da família Kinetis L possuem um poderoso conjunto de funções analógicas, comunicação, temporização e controles de periféricos, tendo um aumento de nível de integração com o tamanho da memória flash e da quantidade de pinos. A arquitetura e o núcleo possuem os seguintes recursos:

a) processador trabalhando com execução de estado com zero espera para memórias;
b) ciclo único de acesso para entrada e saída;

c) excelente densidade de código em comparação com microcontroladores de 8 *bits* e 16 *bits* reduzindo o uso da memória *flash,* custo operacional e consumo de energia;
d) acesso otimizado para gravação na memória: Acesso em ciclos alternados reduz o consumo de energia;
e) compatibilidade 100% com o ARM Cortex-M0 e ARM Cortex-M3/M4;
f) arquitetura simplificada com 56 instruções e 17 registradores;
g) suporte de terceiros no desenvolvimento de ferramentas e *softwares* para tecnologias ARM.
h) *Micro Trace Buffer* (MTB)*:* Solução de baixo custo computacional para identificação rápida de *bugs e* correções;
i) *Bit Manipulation Engine* (BME): Método de manipulação de *bit* permite a redução de tamanho de código e ciclos para registradores de periféricos baseados em operações baseadas em *bit*.

2.3 Comparativo computacional e educacional

Neste subtópico, visa-se apresentar de forma matricial as diferenças e semelhanças do PIC16F628A e da placa de desenvolvimento FRDM-KL25Z com o módulo didático. Como o Módulo didático conta como base a FRDM-KL25Z, logo a equiparação pode se tornar injusta com relação ao PIC mencionado devido ao custo financeiro e a presença do módulo didático no comparativo. Porém, as diferenças positivas justificam o ganho educacional e computacional para o desenvolvimento acadêmico dos estudantes conforme mostra a Tabela 1.

Tabela 1 – Comparação entre o PIC16F628A e o módulo didático

Atributos	PIC16F628A	Módulo didático com a FRDM-KL25Z
Máxima frequência de operação	20 MHz	48 MHz
Arquitetura	8 *Bits*	32 *bits*
Memória	*Flash:* 2048 (*words*); SRAM: 224 bytes; EEPROM: 128 bytes.	*Flash: 128 KB; SRAM: 16 KB.*

Continua

Atributos	PIC16F628A	Módulo didático com a FRDM-KL25Z
Tensão Operacional	2~5,5V	1,71~3,3V
Quantidade de Pinos	18	80 pinos, sendo 64 disponíveis
Conversor AD	0	Até 24 entradas únicas e 4 diferenciais de 16 *bits.*
Conversor DA	0	1 de 12 *bits;*
PWM	1 de 10 *bits*	2 módulos de 2 canais + 1 módulo de 6 canais de 16 *bits.*
Comparadores	1 de 16 *bits*	6 entradas de comparadores
Timers	2 *timers* de 8 *bits*; 1 *timer* de 16 *bits;*	2 módulos TPM de 2 canais + 1 módulo TPM de 6 canais de 16 *bits;* 2 canais PIT; RTC; LPTMR; Temporizador por *tick* do sistema;
Comunicação	1 Módulo USART;	2 SPI de 8 *bits;* USB; 2 Módulos I2C; 3 Módulos UART;
Periféricos	-	1 Acelerômetro de 3 eixos; 1 *LED RGB;* 1 Sensor de toque capacitivo;
Adicionais	-	1 *Display LCD* 16x2; 1 Condicionador de sinais de dois canais; 10 *LEDS;* 10 botões; 1 relé opto acoplado; Regulador de tensão; Gerador de tensão de referência; Amplificadores Operacionais; Potenciômetros;
Linguagem	Assembly	C/C++

Fonte: elaborada pelo autor.

3 MÓDULO DIDÁTICO

Um módulo didático significa, segundo Colegium (2022), um conjunto de materiais e recursos pedagógicos que melhoram a aprendizagem de um determinado conteúdo. Por meio disto, a construção do módulo didático presente neste trabalho visa a melhoria educacional das diversas disciplinas disponibilizadas pelo Departamento de Engenharia Elétrica da Universidade Federal do Ceará (DEE-UFC), principalmente a de Microprocessadores 1 que conta com o microcontrolador PIC16F628A.

Apesar de que, as placas de desenvolvimento já possuem mecanismos para a viabilização de aulas nos laboratórios, para muitos há a dificuldade de realizar as conexões dos componentes com a placa, resultando desde frustração por não conseguir realizar a prática até a queima de um componente.

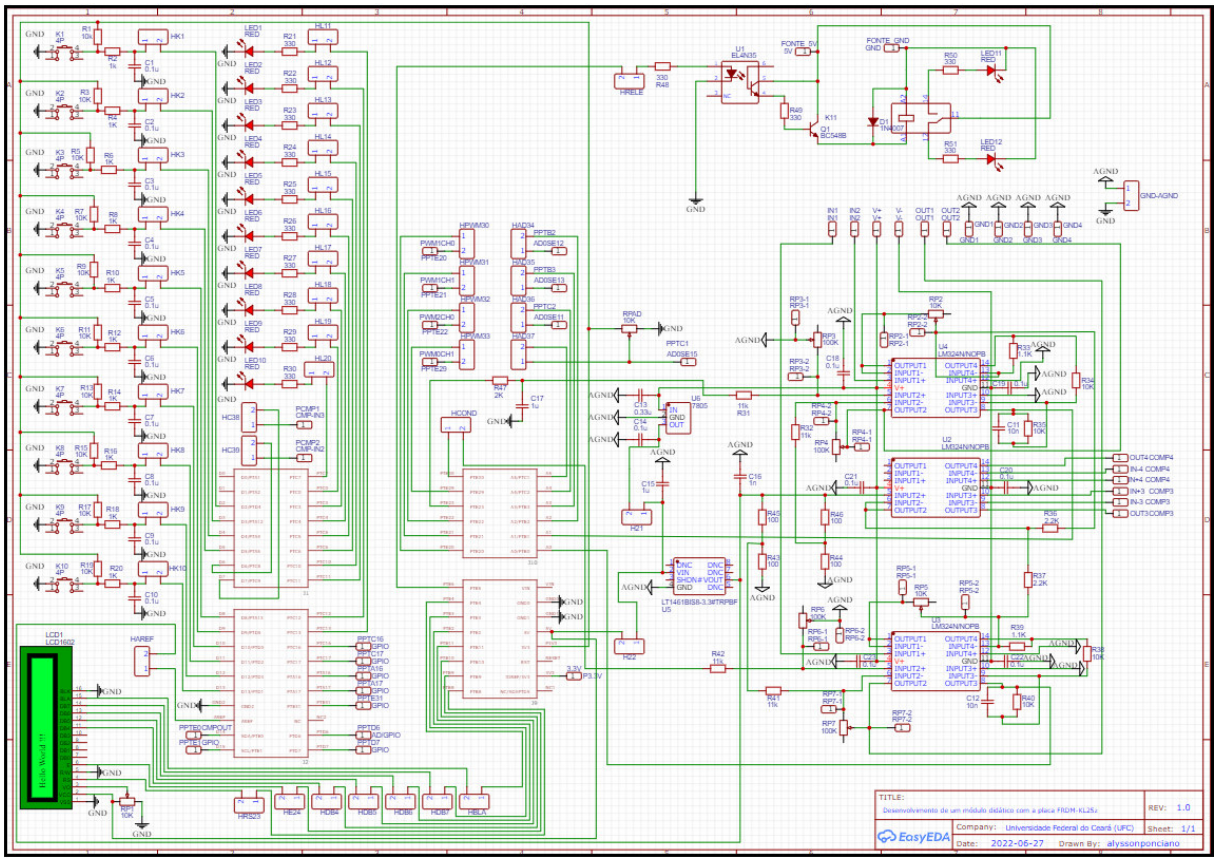
Além disso, pode-se observar em alguns casos, problemas com a *protoboard* em relação às trilhas queimadas, a má fixação dos cabos e os ruídos presentes nesse tipo de prototipagem. Com o módulo didático, os pontos de realizar as conexões, o posicionamento de componentes e os ruídos ocasionados pelas trilhas serão solucionados pela construção da Placa de Circuito Impresso (PCI) e pela soldagem dos componentes.

A construção do módulo didático e a seleção dos equipamentos são baseadas no aproveitamento custo-benefício do projeto, porém alguns componentes se destoam financeiramente como o CI LT1461, gerador de tensão de referência, necessário para um condicionador de sinais de qualidade. Portanto, para poder abranger o máximo de disciplinas possíveis, recursos disponibilizados da FRDM-KL25Z e o potencial criativo dos estudantes há um custo envolvido.

Para o módulo didático, foi pensado no uso do *Display LCD* 16x2 para ter um visualização gráfica de simples uso, botões com atenuação de trepidação mecânica e com possibilidade de interrupções, LEDs para sinalizações, relé opto acoplado para a parte de acionamentos, um circuito condicionador de sinais com 2 canais ajustáveis para aulas de controle, potenciômetro para análise de valores analógicos digitais, 2 amplificadores operacionais, a presença de regulador de tensão e gerador de tensão de referência para alimentação do circuito condicionador de sinais.

A Figura 6 mostra o esquemático de como ficaram as disposições dos componentes no módulo didático, vale observar a presença de pinos fazendo a conexão entre as trilhas justamente para ser habilitada ou desabilitada por meio de micro *jumpers*, permitindo assim a interação do usuário com o *hardware* de uma forma mais interativa.

Figura 6 – Esquemático do Módulo Didático



Fonte: elaborado pelo autor.

O esquemático e o desenho na PCI foram realizados no software gratuito EasyEDA, que possui integração com as empresas LCSC para a compra de componentes e a JLCPCB para a confecção das PCI. O EasyEDA pode ser utilizado tanto na versão web quanto na versão de software, permitindo a conexão de contas para armazenamento de projetos e outros recursos.

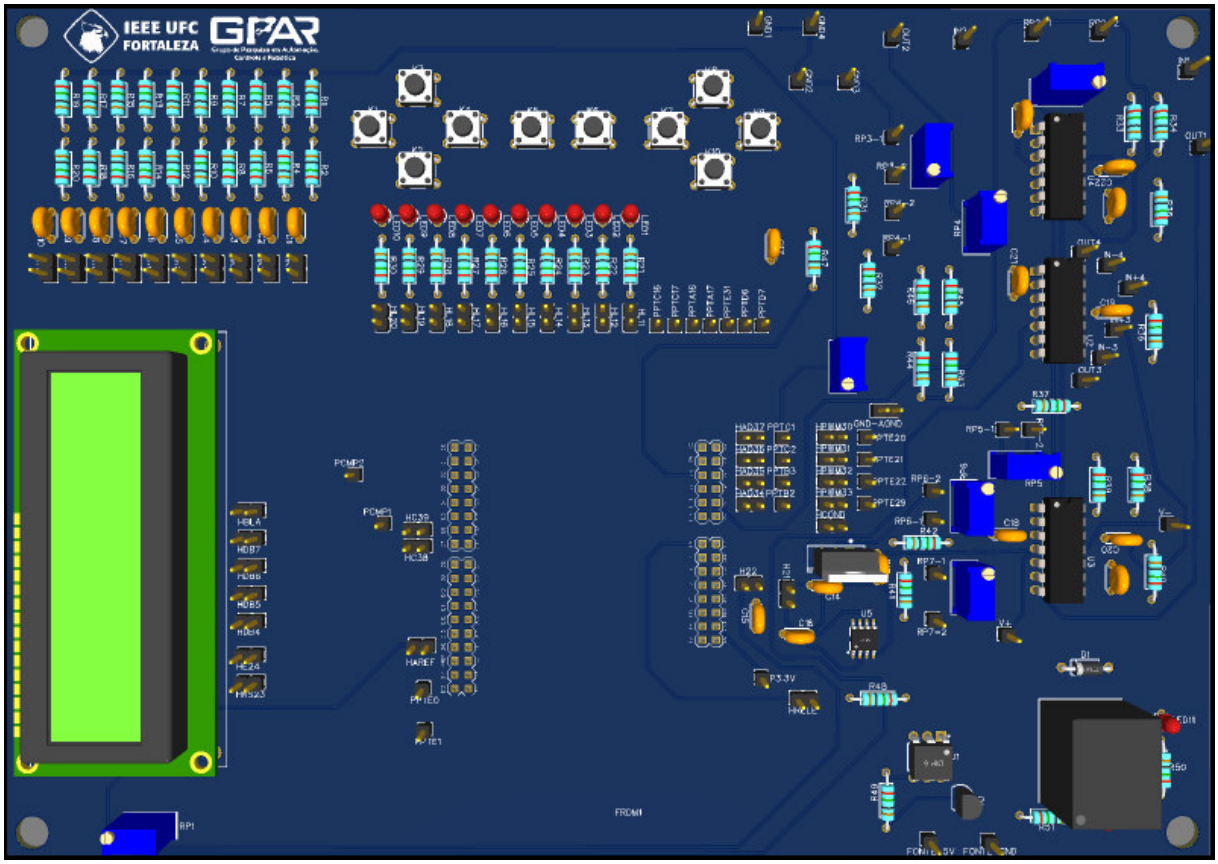
O primeiro passo após a criação do projeto é a seleção e a alocação dos componentes. Após isso, tem-se uma parte fundamental do projeto, pois cada um dos 64 pinos disponibilizados para entrada e saída tem uma função específica, menos algumas exceções que são os pinos de alimentação que se repetem, logo é pensado a distribuição dos componentes para que cada pino seja interligado com eficiência e de forma correta. Por conseguinte, foram realizadas as conexões dos pinos da FRDM-KL25Z com os componentes.

Ainda é possível visualizar a criação de uma conexão GND analógico no circuito condicionador de sinais para se separar da camada do GND digital, porém mantendo ainda um ponto de conexão com o GND digital. Isto serve para diminuir as interferências e manter a qualidade de ambos os sinais.

Por fim, após a finalização do esquemático, foram realizadas o posicionamento dos componentes e as trilhas na PCI, onde já possuem conexões prévias advindas do esquemático. Com isso, foi aplicada uma área de cobre tendo como base o GND sendo comum, pois diminuem as quantidades de conexões, complexidade e melhora a visualização das trilhas.

A Figura 7 mostra como ficou a visualização 3D prévia da PCI montada, logo nota-se a adoção da metodologia *through-hole* (TH), nos quais os pinos dos componentes necessitam atravessar a Placa de Circuito Impressa para efetuar a soldagem. A justificativa para a seleção de tal tecnologia é o fato que ela é mais intuitiva visualmente para os estudantes, embora haja um maior uso da placa pelo tamanho e em alguns casos podem complicar as conexões por serem participantes de duas ou mais camadas. Um ponto importante a ser mencionado neste tipo de visualização é a separação física em blocos de cada funcionalidade da placa. Isso serve para diminuir os impactos dos diferentes tipos de sinais definidos por suas frequências, amplitudes e a característica analógica e digital (SACCO, 2015).

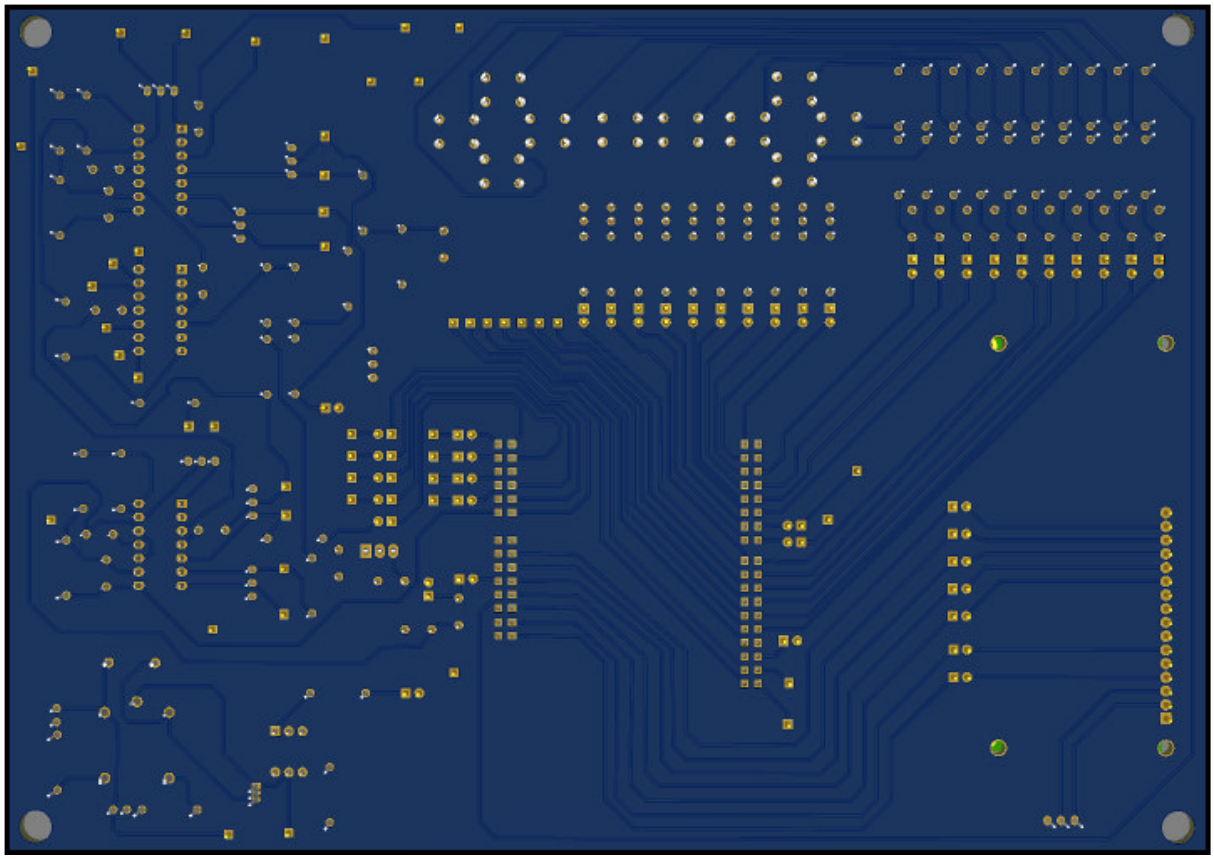
Figura 7 – Visualização frontal do módulo didático



Fonte: elaborado pelo autor.

Já a Figura 8 evidencia a metodologia dos pinos e mostra a complexidade por se tratar de uma PCI em duas camadas, gerando assim uma atenção mais cautelosa para verificar se há conflitos de conexões na parte frontal e traseira. Além disso, observa-se uma maior densidade de trilhas no centro da placa, pois é justamente o posicionamento alocado para a FRDM-KL25Z. Finaliza-se essa análise observando os furos de 6mm nas extremidades para alocação de suportes com intuito de estabilizar a placa em uma superfície.

Figura 8 – Visualização traseira do módulo didático



Fonte: elaborado pelo autor.

Como mencionado anteriormente, todo projeto há um custo envolvido e neste não é diferente. Porém, os custos podem ser minimizados com uma busca de preço mais intensa, troca de fabricante dos componentes e até mesmo em compras de grandes quantidades em diversas lojas de eletrônicas espalhadas pelo país. No caso desse projeto, foi realizado todo o projeto da PCI e por conseguinte gerado o arquivo Gerber para fabricação, sendo reaproveitado alguns componentes sobressalentes de laboratório e compras dos outros itens no mercado local de eletrônica.

Na Tabela 2, é visto todo o custo de fabricação deste módulo didático, onde os preços de grande parte dos componentes são baseados no comércio local e uma pequena parcela em cotações pela internet de baixo volume. Ressalta-se que os valores podem variar de acordo com diversos fatores socioeconômicos e geográficos, além de que não está sendo considerado o preço do frete, assim como o reaproveitamento de peças do laboratório.

Tabela 2 – Descrição dos materiais e custos

Componente	Custo Unitário	Quantidade	Valor Total
Amplificador operacional LM324N	2,20	3	6,60
Espaçador fêmea macho Nylon 6mm	0,90	4	3,60
Botão de toque 6x6	0,25	10	2,50
Capacitor 0,1uF 50V	0,03	17	0,51
Capacitor 0.33uF 50V	1,30	1	1,30
Capacitor 2,2uF 50V	0,14	1	0,14
Capacitor cerâmico 10 nF 50V	0,06	2	1,40
Capacitor poliester 1uF 50V	0,21	2	0,42
CI Optoacoplador 4N35	1,64	1	1,64
Diodo 1N4007	0,23	1	0,23
Display LCD 16x2	26,90	1	26,90
FRDM-KL25Z	97,33	1	97,33
Led Alto Brilho Vermelho	0,40	12	4,80
Micro jumper	0,39	43	16,77
PCI do Módulo didático	18,41	1	18,41
Pinos machos	0,03	136	4,42
Potenciômetro 100k	2,20	4	8,80
Potenciômetro 10k	2,20	4	8,80
Regulador de tensão 7805 5V	2,90	1	2,90
Relé 5V	8,01	1	8,01
Resistor CR25 100Ω	0,05	4	0,20
Resistor CR25 10kΩ	0,05	14	0,70
Resistor CR25 1kΩ	0,05	10	0,50
Resistor CR25 2,2kΩ	0,05	2	0,10
Resistor CR25 330Ω	0,05	14	0,70
Resistor CR25 2kΩ	0,05	1	0,05
Resistor CR25 11kΩ	0,05	4	0,20
Resistor CR25 1,1kΩ	0,05	2	0,10
Transistor BC548	0,40	1	0,40
Barra de pinos fêmea	0,05	64	3,20
Soquete para CI Torneado 14 pinos	2,90	3	8,70
CI LT1461 3.3V	87,11	1	87,11
Custo Total	R$256,59	366	R$317,44

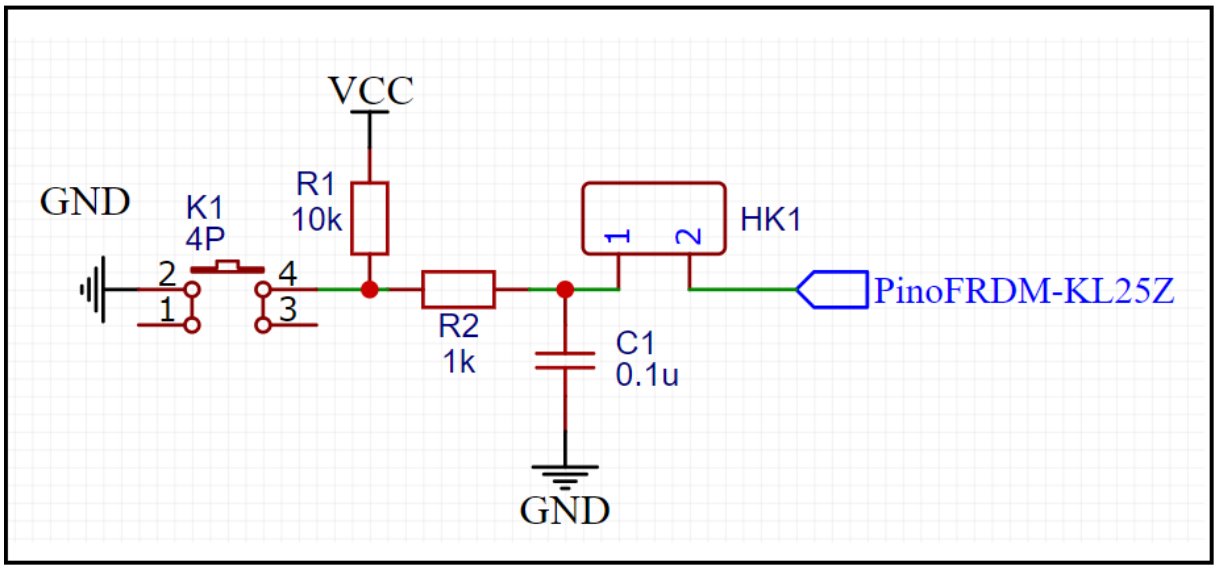
Fonte: elaborado pelo autor.

Ao observar os itens envolvidos, pode-se notar uma diversidade de material e um custo final do projeto de R$317,44, logo verifica-se que o valor médio por componente é de R$ 0,87, justificado pela presença de diversos materiais de baixo custo. Além disso, a Placa de Desenvolvimento FRDM-KL25Z é responsável por cerca de 30% do investimento no projeto e se somar com o CI1461 o custo gira em torno de 60%. Sobrando assim, cerca de 40% nos outros 364 itens presentes na placa. Por fim, o valor é considerado razoável mediante aos recursos que são oferecidos pelo módulo didático e pelo potencial de inserção nas disciplinas do DEE-UFC.

3.1 Botões

O circuito abaixo é similar ao proposto por Eckert (2016) apresentando uma configuração de *pull-up* (Resistor R1) para eliminar um suposto terceiro estágio de indecisão no pressionamento ou não do botão e a função de amenização de trepidação mecânica (*debounce)* por meio do circuito RC marcado pelos componentes C1 e R2. É possível observar na Figura 9, um dos 10 circuitos de botões que existem no módulo didático.

Figura 9 – Esquemático exemplo do circuito do botão



Fonte: elaborado pelo autor.

Sabendo que para um circuito de primeira ordem, o tempo de estabilização é dado por

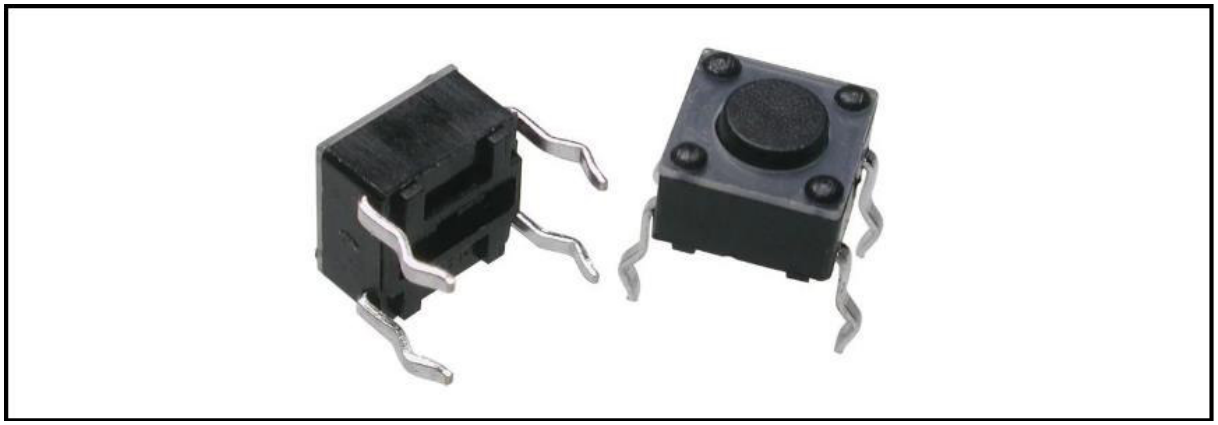
$$\tau = 5 * R2 * C1 = 5 * 1k * 0{,}1 * 10^{-6} = 0{,}5\ ms. \quad (1)$$

Logo, a trepidação mecânica do botão será solucionada via *hardware*, porém poderia ser resolvido de uma forma diferente por meio de programação através da coleta da informação após um atraso gerado pelo usuário. Assim, pode ser testado ao desabilitar o componente HK1 e realizar a montagem de um circuito externo sem a presença do capacitor.

Além disso, as trilhas de todos os botões são interligadas com pinos que possuem interrupção externa liberada, permitindo assim uma maior variedade de utilização do circuito para os usuários.

A Figura 10 mostra o botão utilizado no projeto, sendo que ele possui dimensões de 6x6x4,3mm e esquema PTH que é a mesma tecnologia de pinagem através do furo.

Figura 10 – Botão de toque 6x6x4,3mm

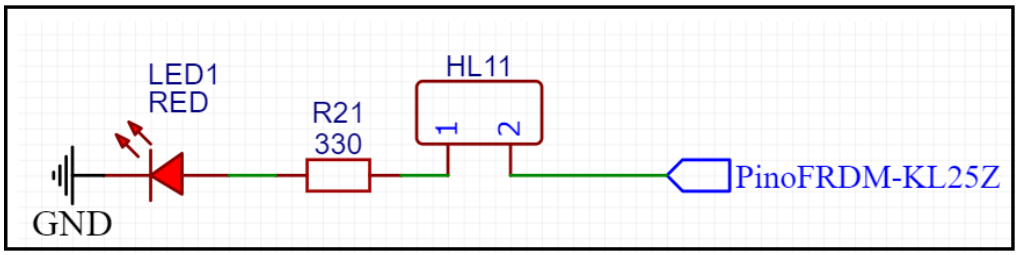


Fonte: adaptado da AutoCore Robótica.

3.2 Diodos Emissores de Luz

Ao contrário do circuito dos botões que são sinais de entradas que enviam dados para os pinos da KL25Z, o esquemático presente na Figura 11 é um circuito de sinal de saída, ou seja, a informação parte da placa de desenvolvimento para o LED. O acionamento está definido para ligar com o sinal 1 e desligar com o sinal 0, pois isso é devido ao GND que está conectado ao catodo. Caso, houvesse um 3,3 V no anodo e o pino da FRDM-KL25Z no catodo, logo seria necessário inverter o acionamento passando a ser desligado com 1 e ligado com 0.

Figura 11 – Esquemático exemplo do circuito do diodo emissor de luz



Fonte: elaborado pelo autor.

Para alimentar um LED é comumente utilizado uma corrente por volta de 20mA, onde é suficiente para prevenir uma queima de um pino da KL25Z que suporta até 25mA (SEMICONDUCTORS, 2014), quanto a queima do próprio LED. Para isso, foi utilizado de forma suficiente um resistor de 330 Ω para realizar a limitação da corrente, já que

$$I = \frac{3{,}3V - 0}{330\Omega} = 10mA. \quad (2)$$

Com a Figura 12 é possível observar que o modelo utilizado é um LED de alto brilho vermelho 5mm do tipo PTH.

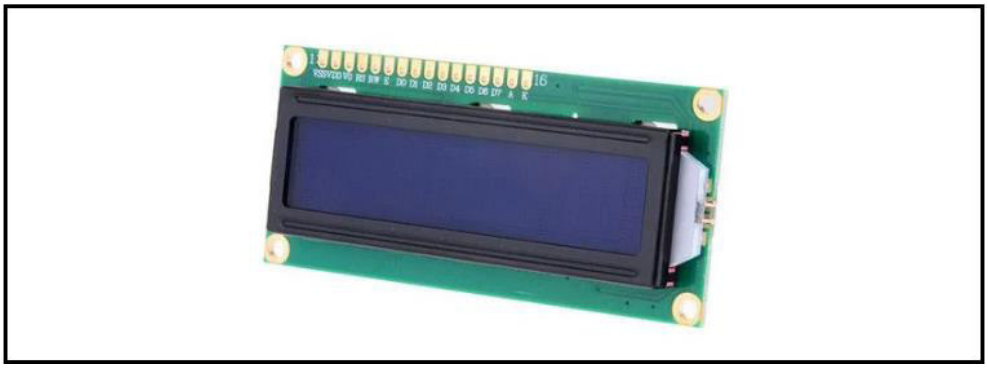
Figura 12 – LED alto brilho vermelho 5mm

Fonte: adaptado da Autocore Robótica.

3.3 Display LCD 16x2

O *Display LCD 16x2* utilizado no projeto possui uma luz de fundo azul e um controlador HD44780, sendo bastante utilizado em diversos projetos de prototipagem devido a sua configuração mais simplificada de uso em comparação com outros *Displays* superiores. A Figura 13 mostra o produto com as conexões de alimentação VSS, VDD, V0, A e K, sendo respectivamente o GND, 5V, tensão de regulagem do contraste, tensão positiva de regulagem da luz de fundo e GND de regulagem da luz de fundo.

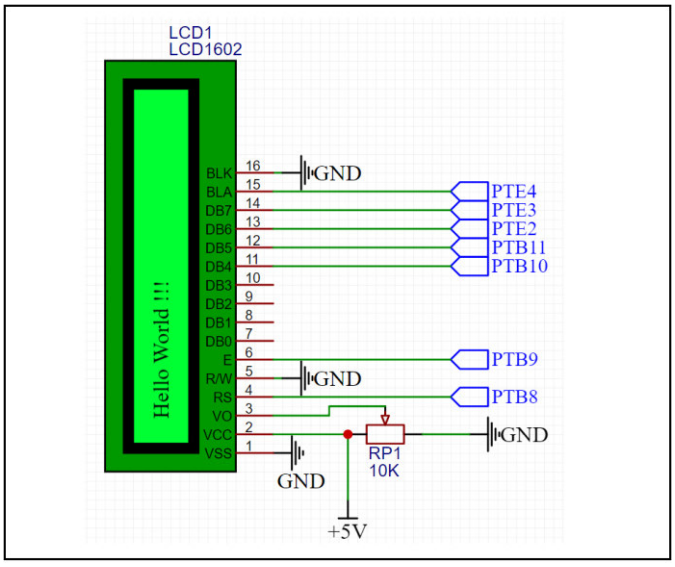
Figura 13 – Display LCD 16x2 com luz de fundo azul



Fonte: adaptado da Autocore Robótica.

Após os pinos de alimentação, em ordem da esquerda para direita, existem os pinos de controles e dados que são: RS, RW, E, D0, D1, D2, D3, D4, D5, D6 e D7. Conforme Learningmicro (2018), o pino “RS” é o Seletor de Registrador, “RW” é o pino de escrita e leitura e o “E” sendo o Habilitador. O seletor de registrador juntamente com o habilitador possui função importantíssima no controle do Display, pois desde as configurações de iniciar o LCD até realizar o deslocamento são passadas por eles. Já o pino de leitura e escrita fica aterrado justamente por sempre estar em uma configuração de escrita das informações, sendo desnecessário o uso com a função de leitura que seria com o nível lógico alto. Ressalta-se que a tensão de 5V é somente para alimentação do display e não para os pinos de dados, logo não é necessário nenhum ajuste de nível lógico. A Figura 14 mostra como ficou a conexão do display com a KL25Z.

Figura 14 – Esquemático do circuito do Display LCD 16x2



Fonte: elaborado pelo autor.

A metodologia utilizada para essas conexões estão de acordo com (LEARNINGMICRO, 2018), onde são utilizados apenas os pinos DB7, DB6, DB5 e DB4 para o repasse de comandos ou dados. O ponto positivo é a redução da quantidade de conexões, porém há um acréscimo de linhas de comando na parte de programação, já que são repassadas as sequências *upper nibble* e *lower nibble.* Ou seja, há um ganho de *hardware* com 4 pinos livres, mas um leve desgaste no *software* pelo tamanho do código e processamento.

3.4 Relé com optoacoplador

Um dos equipamentos mais utilizados em prototipagem envolvendo acionamentos e automação é o Relé. Conforme Braga (2012), o relé eletromecânico pode ser definido com um dispositivo comutador eletromecânico, portanto por ser um dispositivo que comuta para correntes elevadas em comparação com outros circuitos integrados, logo é bastante utilizado em acionamentos.

Já o optoacoplador 4N35 consiste segundo Semiconductors (2010), sendo um LED infravermelho de arsenieto de gálio e um fototransistor NPN de silício. Sua função é isolar eletricamente o circuito de entrada com o de saída, já que o acionamento se dá de forma luminosa sem contato elétrico entre os dois circuitos.

Ao juntar os dois componentes, temos uma conexão entre o circuito de potência e o circuito de controle com as seguintes características.

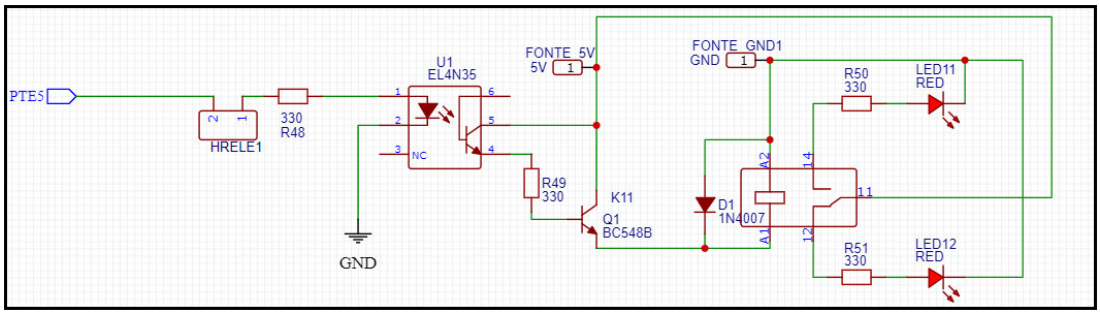
Primeiramente, o circuito de entrada da parte de controle não sofre nenhum dano caso haja algum problema na parte de potência, em seguida há a possibilidade de acionar cargas de corrente elevada sem haver queima do pino da placa de desenvolvimento. Ademais, os GNDs dos dois circuitos não podem ser interligados justamente por causa da isolação proveniente do optoacoplador.

Além do optoacoplador e do relé, a presença do transistor e do diodo são fundamentais para circuitos que envolvam grandes correntes e comutações, pois o optoacoplador possui limitação de corrente em seus terminais fazendo com que não supra energeticamente o relé para usos de corrente elevadas. Assim, a solução é utilizar um transistor para polarizar juntamente com o optoacoplador e permitir passagem de corrente diretamente da fonte. Já o diodo serve para fornecer um caminho para a dissipação da corrente durante as comutações do relé.

Na Figura 15 é possível observar como o circuito foi montado, com destaque para o diodo de roda livre próximo ao relé, a necessidade de uma fonte externa de 5V para alimentar

o relé e os contatos normalmente aberto e o normalmente fechado que contam com um LED em cada ramo.

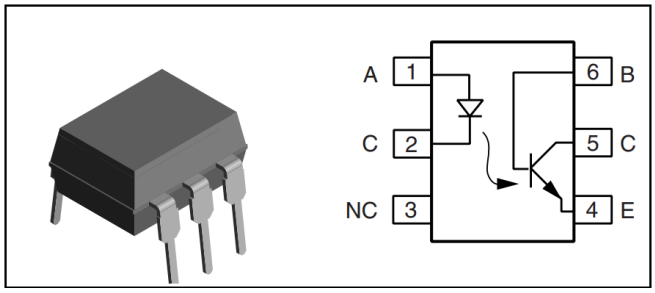
Figura 15 – Esquemático do circuito do relé com optoacoplador



Fonte: elaborado pelo autor.

O circuito integrado do optoacoplador 4N35 utilizado no projeto se encontra na Figura 16, nela é visto que possui a pinagem *Dual In-line Package* (DIP) e PTH, onde é um componente de linhas duplas com pinos que atravessam a placa.

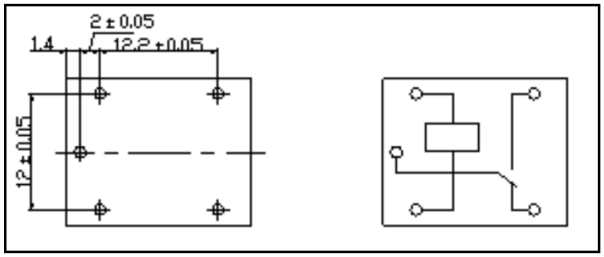
Figura 16 – CI optoacoplador 4N35



Fonte: adaptado de Vishay Semiconductors.

O relé eletromecânico de 5V usado é um modelo de 5 pinos com encapsulamento do tipo PTH, sendo dois das bobinas, 1 comum, 1 normalmente fechado e o outro normalmente aberto.

Figura 17 – Relé eletromecânico



Fonte: adaptado da Songle Relay.

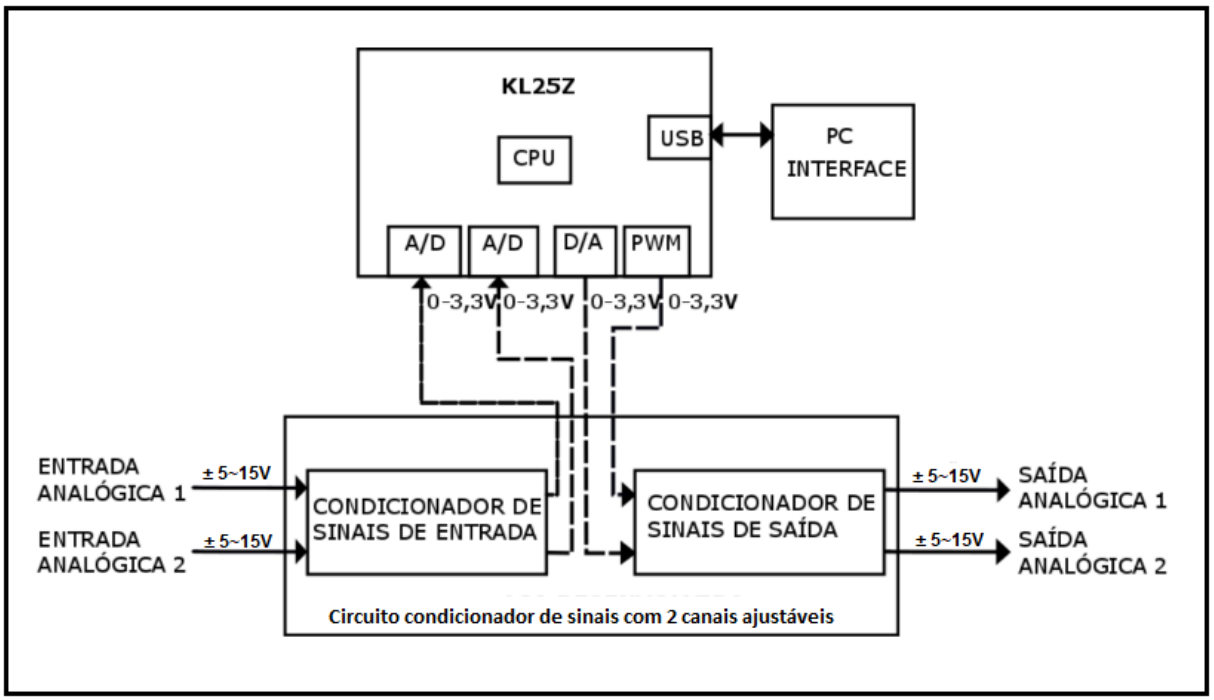
3.5 Condicionador de sinais com 2 canais de ganhos ajustáveis

O projeto do circuito tem como base o trabalho de Mota (2016), no qual ele apresenta um condicionador de sinais com 2 canais com ganhos fixos. Os ganhos fixos se dão pelo fato de utilizar resistores fixos e não variáveis, assim não pode ser realizado nenhum ajuste de achatamento ou ampliação dos valores capturados de amplitude de tensão.

A função do circuito condicionador de sinais é capturar um sinal de tensão geralmente maior que o valor suportado pelos pinos do microcontrolador, em seguida condicionar para que se reduza seu nível de tensão em um valor suportado e que seja lido por um conversor analógico digital para realizar tarefas de controle. Por fim, é gerado um valor para o conversor digital analógico ou PWM conforme o valor de entrada capturado pelo microcontrolador e passar um por um processo para transforma-lo no nível de tensão similar ao de entrada.

A Figura 18 mostra em formato de diagrama de blocos o circuito condicionador de sinais, sendo bastante similar ao proposto por Mota (2016), porém com os ajustes dos sinais de entrada e saída fazendo com que o sinal condicionado não seja apenas de 15 V, mas também valores até 5 V sejam positivos ou negativos.

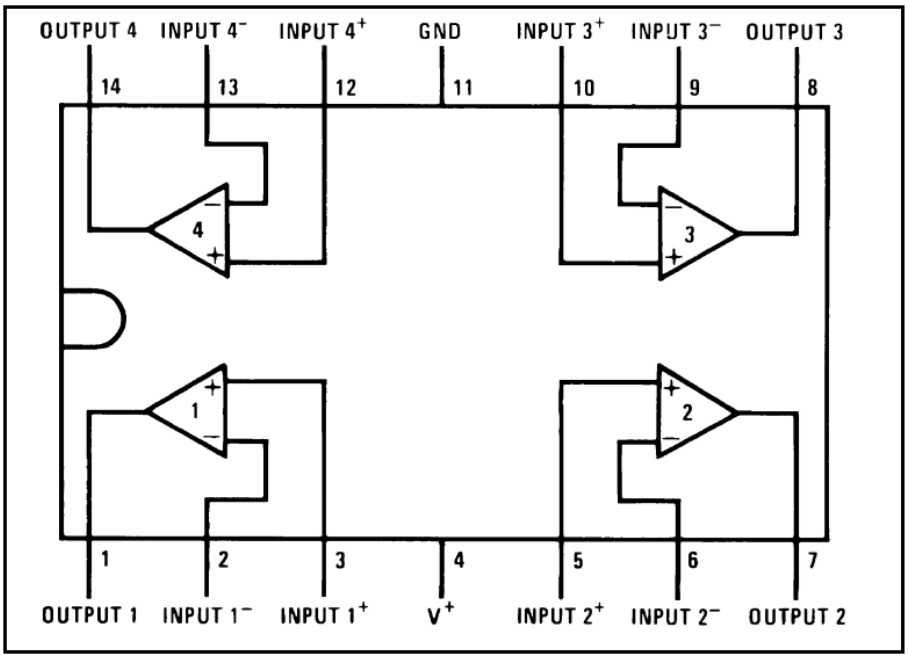
Figura 18 – Diagrama de blocos do circuito condicionador de sinais



Fonte: adaptado de Francisco Jandysley Aguiar Mota (2016, p. 15).

Na Figura 19 é possível observar o circuito integrado LM324N, onde é composto por 4 amplificadores operacionais e duas entradas de tensão positiva e negativa para regulação dos seus limites inferiores e superiores. Este equipamento é fundamental no condicionamento de sinais devido aos seus arranjos disponibilizados nas literaturas que permitem reduções, amplificações, inversões e outros recursos.

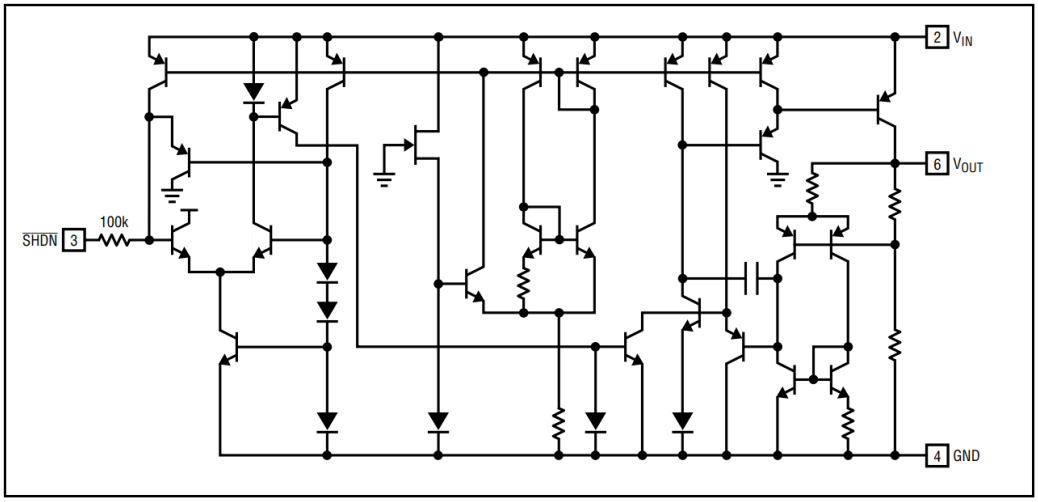
Figura 19 – Circuito Integrado LM324N



Fonte: adaptado da Texas Instruments.

O circuito integrado mostrado na Figura 20 é o único com encapsulamento SOP (*Small Outline Package)*, no qual é um tamanho reduzido de um DIP (*Dual In-line Package)* sendo aplicado em somente uma camada da PCI. Este CI é gerador de tensão de referência, LT1461 3,3V, cuja função é a estabilização com uma precisão altíssima da tensão em um valor de referência que no caso deste projeto é em 3,3V. Essa estabilização é super importante para o funcionamento adequado para leituras de ADC de 16 *bits,* pois a variação de tensão é tão pequena que o valor não é perceptível pela resolução do ADC.

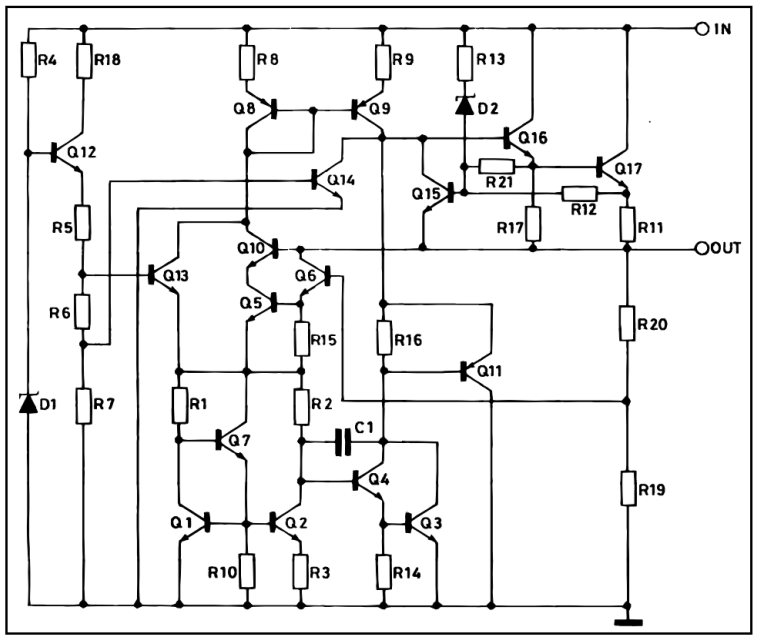
Figura 20 – Circuito Integrado LT1461 3,3V



Fonte: adaptado da Analog Devices.

Além disso, temos na Figura 21, o circuito integrado regulador de tensão em 5V L7805. A função dele é captar a tensão de entrada de alimentação dos amplificadores operacionais e reutilizar para a alimentação do LT1461, fazendo com que o nível de tensão seja reduzido para o limite operacional do gerador de tensão de referência. Em alguns casos, pode ser necessária a aplicação de um dissipador de calor na parte metálica do CI devido ao seu sobreaquecimento.

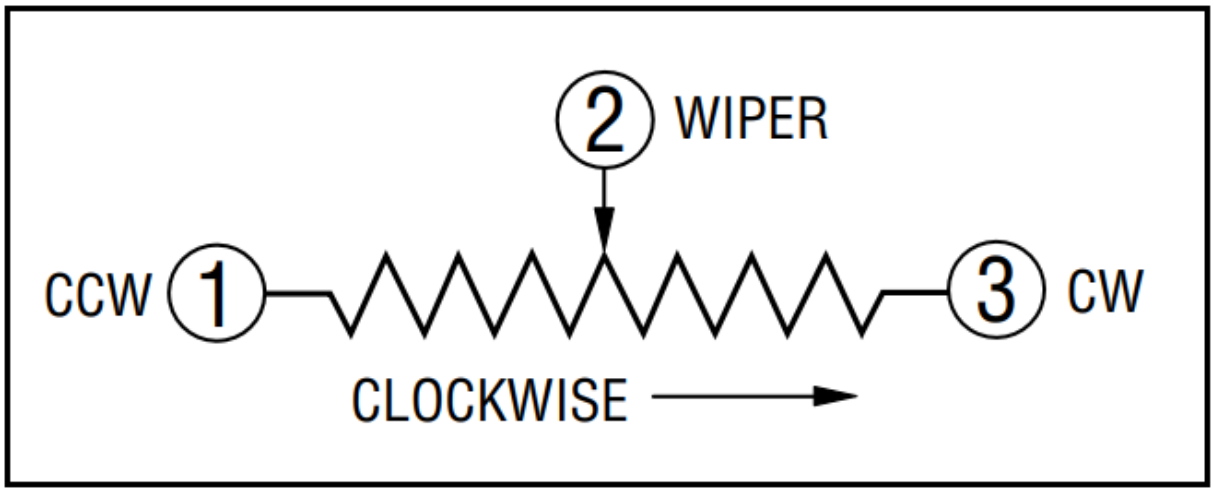
Figura 21 – Circuito Integrado L7805



Fonte: adaptado da STMicroelectronics.

Os trimpots ou resistores variáveis são equipamentos que permitem o ajuste dos níveis de resistência entre seus polos de acordo com a necessidade dos projetos. Na Figura 22, mostra-se um trimpot com ajuste localizado na parte superior do equipamento. O equipamento é comercializado com um valor de resistência, logo a medição de resistência do pino 1 e 2 é um valor complementar da medição do pino 2 e 3 para o valor total do equipamento. Ressaltam-se que estes equipamentos sofrem com precisão da mesma forma que os resistores fixos.

Figura 22 – Resistores variáveis

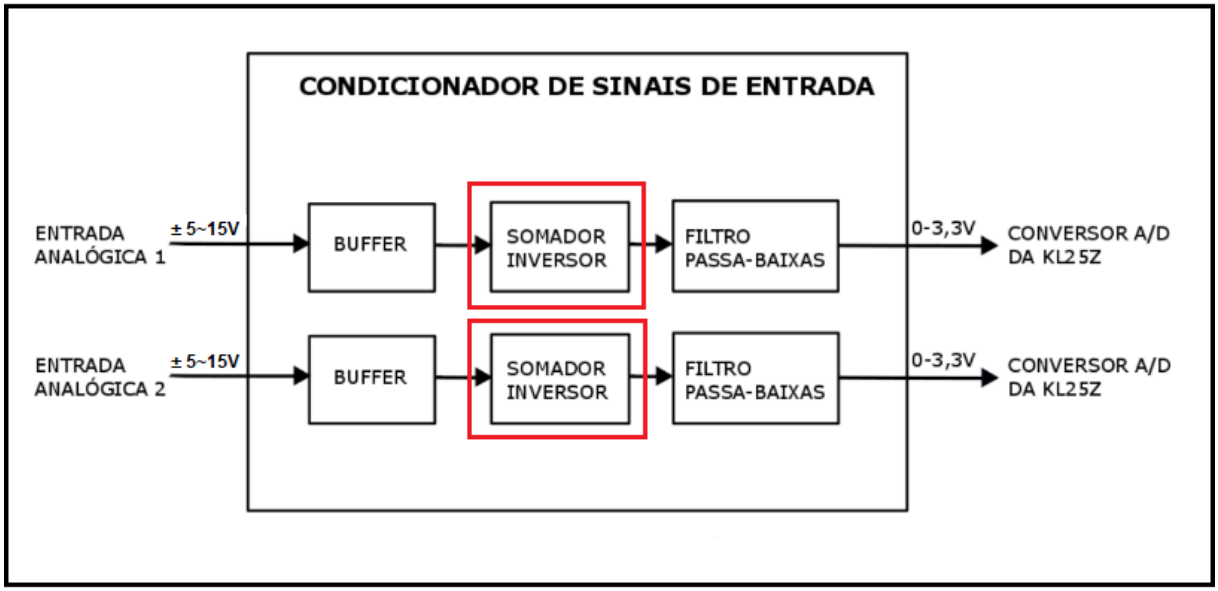


Fonte: adaptado da Bourns.

3.5.1 Circuito de entrada

O trabalho apresentado por Mota (2016), no circuito de entrada são apresentados os seguintes estágios para o condicionamento do sinal. Primeiramente, há a entrada analógica do sinal de tensão, em seguida existe amplificador de ganho unitário para aumentar a isolação entre dois estágios e eliminar o efeito de carga entre estágios (ALEXANDER; SADIKU, 2013). O retângulo vermelho presente na Figura 23 mostra o somador inversor, onde tem como resultado a inversão do sinal de entrada, em contrapartida será tratada a diminuição da amplitude do sinal de acordo com os ganhos selecionados (NILSSON; RIEDEL 2015) e é justamente o ponto interferido para possibilitar o ajuste variável do condicionador de sinais. Por fim, há um filtro passa-baixa para reverter o sinal invertido no estágio anterior e atenuar frequências acima da frequência de corte.

Figura 23 – Diagrama de blocos do circuito condicionador de sinais de entrada



Fonte: adaptado de Francisco Jandysley Aguiar Mota (2016, p. 17).

O circuito somador inversor é crucial no condicionamento do sinal, já que ele irá reduzir o nível de tensão para o valor desejado. Além disso, há a possibilidade de ter um ajuste de ganho na entrada alterando apenas um componente. Logo, é possível comprovar conforme a Equação 3 que fala sobre a tensão de saída em um circuito somador inversor. Normalmente, existe um sinal negativo na fórmula, porém ele será retirado tendo em vista o filtro passa-baixas no estágio seguinte que inverterá novamente o sinal de entrada. Assim, a relação para saber qual o valor da resistência mediante a entrada fica como

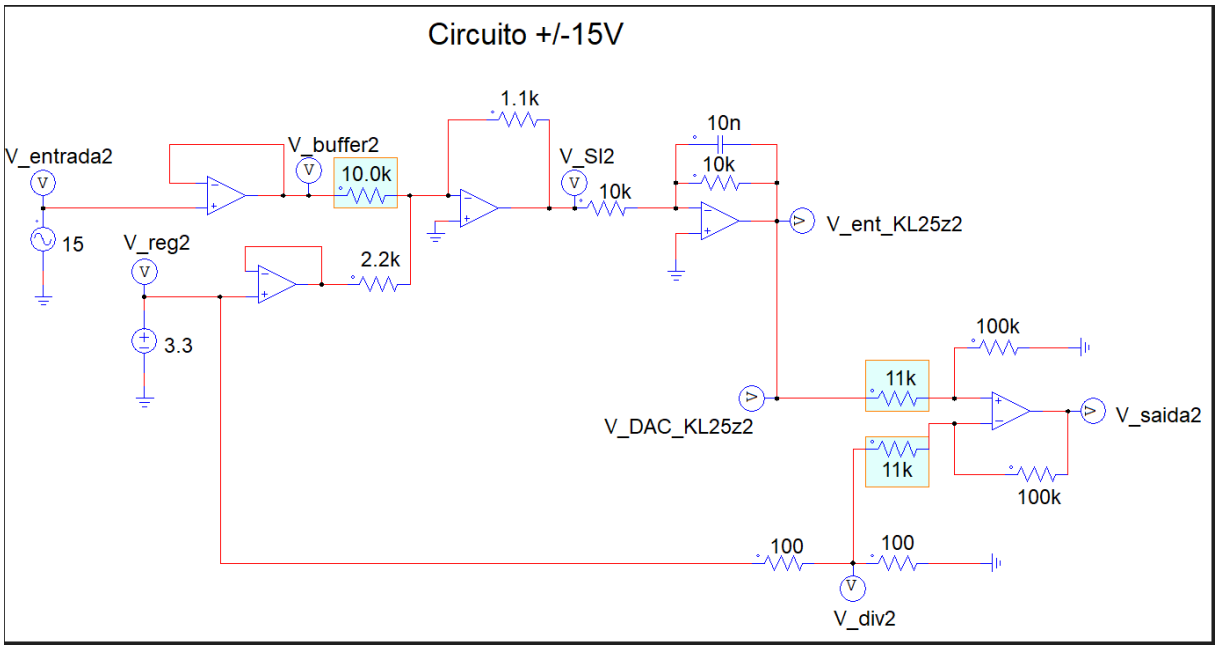
$$v_o = \left(\frac{R_f}{R_a} * v_a + \frac{R_f}{Rb} * v_b\right)$$

$$3{,}3V = \left(\frac{1{,}1k}{2{,}2k} * 3{,}3V + \frac{1{,}1k}{Rb} * V_{in}\right) \quad (3)$$

$$\frac{V_{in}}{Rb} = 0{,}0015\frac{V}{\Omega}.$$

Portanto, é visto que a relação da tensão de entrada com o resistor variável de entrada é de 0,0015, assim para uma tensão de entrada de 15V o resistor deve ser de 10kΩ e para uma tensão de 5V basta um resistor de 3,3kΩ. Isso é possível observar na Figura 24, onde necessita de apenas um resistor variável para realizar tal procedimento.

Figura 24 – Esquemático do circuito condicionador de entrada



Fonte: elaborado pelo autor.

Um ponto importante a ser mencionado é a presença do deslocamento da parte negativa da onda para estar dentro dos limites capturáveis do microcontrolador que no caso é entre 0 e 3,3V. A responsabilidade de realizar esse deslocamento está nas mãos do circuito inversor com o resistor de 2,2k e a fonte de 3,3V. Para isso, é necessário um gerador de tensão de referência, onde a função dele é estabilizar com muita precisão a tensão em 3,3V, pois o regulador de tensão possui muita instabilidade nessa fixação ocasionando assim a transposição de ruídos para outros estágios do circuito, logo haverá uma estabilidade com o CI LT1461. O resistor de 2,2k serve justamente para fixar um valor de 1,65V de deslocamento da onda, retirando assim a parte negativa e deixando apenas o ajuste do resistor variável com intuito de achatar ou expandir a onda conforme a tensão de entrada.

Finaliza-se o condicionador de sinais de entrada após passar por todos os estágios anterior, assim é permitido o tratamento do sinal via programação e *hardware* pelo conversor analógico digital da KL25Z.

Segundo o teorema de Nyquist, um sinal pode ser recuperado com uma frequência de amostragem superior a duas vezes a frequência do sinal (ALEXANDER; SADIKU, 2013). Portanto, ao analisar a Equação 4 que é a frequência de corte do filtro passa-baixas vista a necessidade de usar frequências maiores que 3,2 kHz, sendo que uma ótima condição é captar cerca de 10 vezes a frequência do sinal, logo a amostragem recomendada pode ser de 16 kHz.

$$f_c = \frac{1}{2\pi * C * R} = \frac{1}{2\pi * 10k * 10nF} = 1591{,}55\ Hz. \quad (4)$$

Entretanto, é importante observar que podem ser usados sinais com frequências inferiores à de corte, desde que se mantenha a recomendação de utilizar uma frequência de amostragem de 10 vezes a do sinal para permitir a reconstituição do referido.

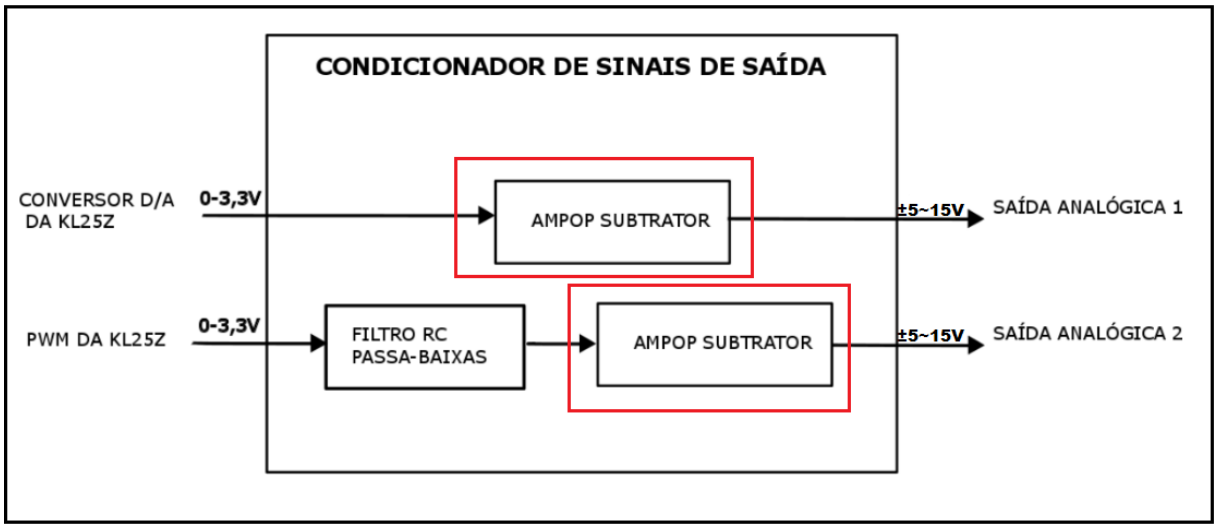
3.5.2 Circuito de saída

O circuito condicionador de sinais de saída possui dois canais assim como o de entrada, porém há um estágio a mais no circuito que pertence a saída pelo PWM que é o filtro RC passa-baixas. O PWM pode gerar ondas quadradas com uma certa frequência e ciclos de trabalhos, porém isso não é o suficiente para gerar um sinal analógico. Logo, é necessária a utilização do filtro RC passa-baixas para converter as ondas quadradas em valores analógicos de acordo com o ajuste de ciclos de trabalho do PWM.

Em meio a isso, é possível observar que enquanto o DAC possui apenas o amplificador operacional subtrator, o PWM possui um estágio a mais. A utilização do PWM é pelo fato que a placa possui apenas um DAC, portanto, para evitar a compra de um componente a mais para realizar esse serviço, foi então utilizado esse método com o PWM com filtro passa-baixas.

Ao analisar a Figura 25, é visto que a faixa de tensão produzida é entre 0 e 3.3V devido a limitação do próprio microcontrolador, logo no estágio do ampop subtrator é necessário que além do aumento do nível de tensão de saída seja importante também o deslocamento para a parte negativa com intuito de replicar a onda de entrada do condicionador de sinais, porém agora com a captação do sinal pelo microcontrolador para realizar medições de controle.

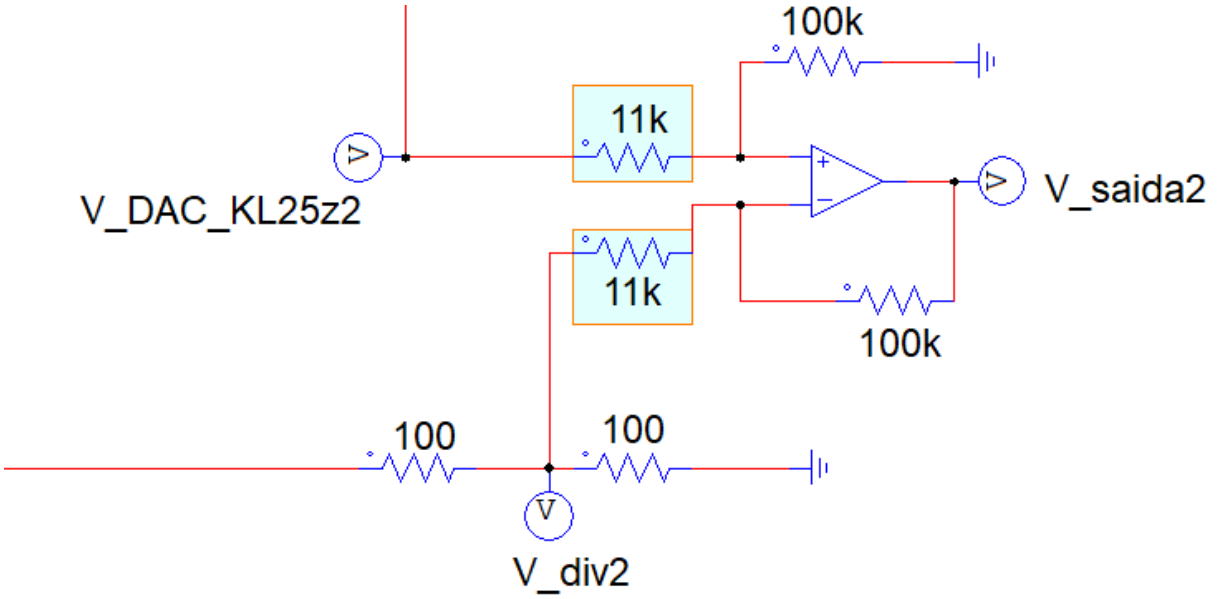
Figura 25 – Diagrama de blocos do circuito condicionador de sinais de saída



Fonte: adaptado de Francisco Jandysley Aguiar Mota (2016, p. 23).

A configuração de um ampop subtrator pode ser visualizada na Figura 26, onde é possível observar que há apenas um estágio na saída, o que sinaliza que o circuito simulado se refere ao conversor digital analógico. Para transformar o circuito abaixo para ser alimentado por um PWM basta adicionar um filtro passa-baixas após o voltímetro V_DAC_KL25z.

Figura 26 – Esquemático do circuito condicionador de saída



Fonte: elaborado pelo autor.

Assim como o circuito de entrada possui resistor variável para possibilitar entradas de tensões diferentes, a saída necessita da mesma forma com intuito de igualar a onda de entrada.

Por ser um circuito subtrator e possuir pares de resistor com valores iguais, logo a fórmula pode ser reduzida para:

$$v_o = \frac{R_b}{R_a}(V_+ - V_-)$$

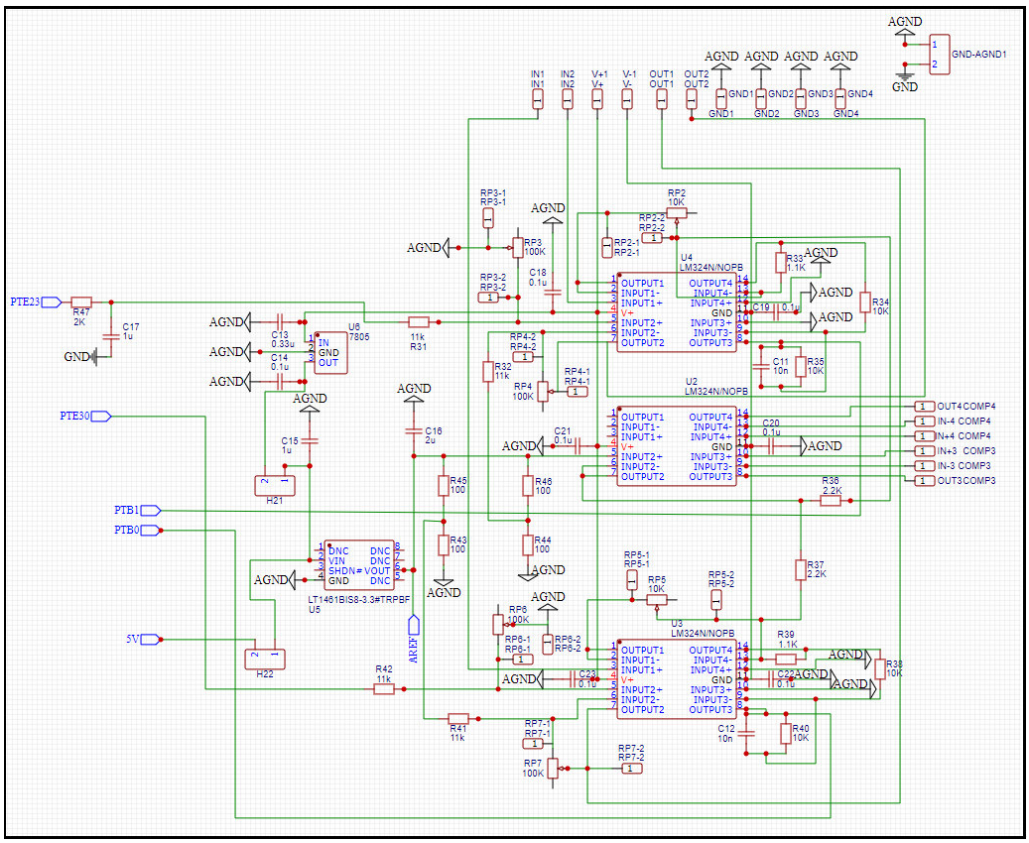
$$V_O = \frac{R_B(3{,}3 - 1{,}65)}{11k} \tag{5}$$

$$\frac{Vo}{R_b} = 0{,}00015\,\frac{V}{\Omega}.$$

3.5.3 Esquemático do circuito

O esquemático do circuito é fundamental para identificar os pontos de conexões de entrada e saída, além de poder identificar o funcionamento do mesmo. Logo, a Figura 27 mostra como está a disposição dos equipamentos e suas respectivas conexões.

Figura 27 – Esquemático do circuito condicionador de saída



Fonte: elaborado pelo autor.

É importante visualizar nesse esquemático alguns pontos, sendo eles:

a) terra analógico (AGND);
b) pontos de medições de resistência dos potenciômetros (RPx-1 e RPx-2);
c) entradas e saídas tipo pino macho para fácil conexão com entradas do tipo garra ou fêmea;
d) presença de um regulador de tensão (7805) e um gerador de tensão de referência (LT1461);
e) presença dos capacitores de acoplamento;
f) separação física dos outros recursos presentes nas placas.

O primeiro ponto se refere ao GND analógico, onde é necessário para quando existem a presença de circuitos digitais e analógicos na mesma placa. O intuito é realizar dois planos de cobres separados, porém como há a limitação física de duas camadas para tal procedimento, logo se adotou um plano de terra digital e conexão entre os GND analógicos.

Em seguida, há os pontos de medições dos potenciômetros para ajustar as entradas e saídas conforme as fórmulas apresentadas. Existem 6 potenciômetros para o condicionador de sinais, logo são necessários 12 pontos de medição de resistência, sendo dois para cada potenciômetro. O ajuste da resistência deve ser feito com o circuito desenergizado devido a configuração de medição de resistência para não queimar nenhum componente.

Devido à falta de componentes e a necessidade de uma placa mais robusta, a utilização dos conectores do tipo banana foram desconsiderados para esse projeto e foi dada entrada numa solução de baixo custo que são os pinos machos. Embora não sejam os melhores conectores devido a sua simplicidade, possuem fácil integração com o tipo garra presente em osciloscópios e por serem mais abrangente.

No quarto ponto, mostra-se que o próprio circuito alimenta ele mesmo, pois a entrada de V+ para alimentação dos ampops também faz a alimentação de um regulador de tensão CI 7805 com intuito de regular entradas de 15V para 5V e por conseguinte o LT1461, como gerador de tensão de referência, estabiliza tensões para 3.3V na saída.

As presenças dos capacitores de acoplamentos servem para manter os níveis de tensão e evitar que surtos possam comprometer os resultados esperados.

A presença de diversos tipos de frequência, formato de sinais e níveis de tensão faz com que a placa de circuito impresso sofra com diferentes tipos interferências (SACCO, 2015). Logo, o circuito condicionador de sinais fica em um espaço diferente dos outros recursos, assim como cada recurso possui um espaço definido para evitar essas interferências.

4 PROGRAMAÇÃO

A placa de desenvolvimento FRDM-KL25Z, que possui o microcontrolador KL25Z128VLK, suporta as linguagens de programação C e C++ e a seleção fica de acordo com as configurações iniciais de projeto. Além disso, tem suporte para *baremetal* e *FreeRTOS*, no qual permite a utilização de um sistema operacional em tempo real para aplicações mais avançadas, porém no caso deste trabalho será usado a função *baremetal* por não necessitar de um *RTOS.* O termo *baremetal,* segundo Vmware (2022), significa que não há um sistema operacional entre o *hardware* e o *software*, suprindo assim as necessidades deste projeto.

Devido a disciplina de Programação Computacional para Engenharia ser ensinada a linguagem de programação C, os códigos disponibilizados para usos dos recursos deste trabalho também serão na mesma linguagem. Isso se baseia no fundamento no qual o estudante possui um conhecimento prévio em C, com possibilidade de aprimoramento de acordo com o exercício das atividades previstas em aplicações de diferentes laboratórios.

Porém, fica livre ao aluno e ao professor a definição de qual linguagem será utilizada, desde que haja uma conversão da escrita de programação C para C++ de forma que permaneça a lógica para o funcionamento dos componentes presentes no módulo didático. Portanto, evidencia cada vez mais a flexibilidade e melhoria na capacidade criativa dos alunos.

O aplicativo utilizado como *Integrated Development Environment* (IDE) será o MCUXpresso disponibilizado pela própria NXP, em adição será utilizado um *Software Development Kit* (SDK) também oferecida pela NXP. A IDE será o ambiente de programação enquanto a SDK será um facilitador dos usos dos periféricos (Semiconductors, 2017).

Como o Assembly é uma linguagem para maximizar a eficiência dos processos que era totalmente necessário na época do lançamento do PIC16F628A, já hoje há um poderio computacional muito maior que a diferença do uso de ambas as linguagens são pequenas e irrelevante na maioria dos casos de utilização no que tange a processamento de código. Porém, em questão de usabilidade a linguagem C é muito superior ao Assembly devido a sua simplificação na programação. Logo, a linguagem C se torna mais atrativa para os usuários do que o Assembly.

A Tabela 3 mostra um leve comparativo entre as duas linguagens, sendo o lado esquerdo a linguagem Assembly do PIC e o lado direito sendo o C. É possível notar que de acordo com o nosso aprendizado ao longo do tempo, a linguagem C oferece uma facilidade maior de entendimento em comparação com o Assembly e esse sentimento é mais intenso de acordo com a complexidade do código.

Tabela 3 – Exemplo código Assembly PIC

Variáveis antes do comando
W EQU 0x49;
TESTE EQU 0xA1;
Comando de soma
ADDWF TESTE,0
Variáveis após o comando
W=0x49;
TESTE= 0xEA
	Variáveis antes do comando
	Int valor1 = 50;
	Int valor2 = 40;
	Int valor3 = 0;
	Comando de soma
	valor3 = valor1 + valor2
	Variáveis após o comando
	valor1 = 50;
	valor2 = 40;
	valor3= 90;

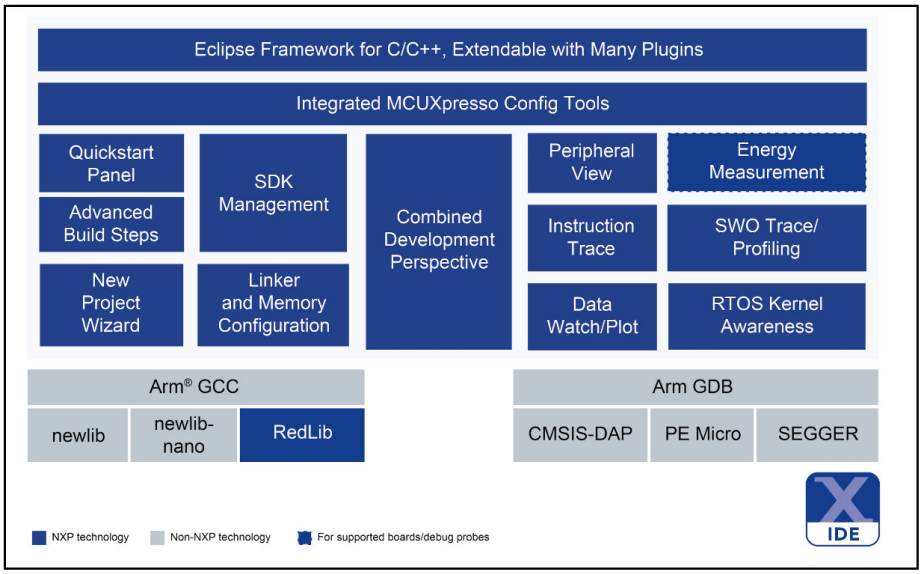
Fonte: elaborado pelo autor.

4.1 MCUXpresso

O MCUXpresso é uma IDE baseada em Eclipse para uso em microcontroladores com núcleos ARM Cortex-M, onde oferece recursos avançados para compilação, edição e depuração e diversas outras funcionalidades de gerenciamento de *clock,* alocação de memória, rastreamento de código, pinos e outras especialidades (SEMICONDUCTORS, 2022).

A Figura 28 mostra o diagrama de blocos da MCUXpresso, é visto a base em Eclipse para C e C++, a relação da IDE com as suas funcionalidades desde a criação de um novo projeto com o novo *Wizard* até a mensuração de energia, onde são tecnologias desenvolvidas pela própria NXP, enquanto há também envolvimento com as ferramentas ARM GCC e GDB para compilação e depuração de código (DEVELOPER, 2022).

Figura 28 – Diagrama de blocos MCUXpresso IDE



Fonte: adaptado de Ambiente de Desenvolvimento Integrado (IDE) MCUXpresso (2022, p. 2).

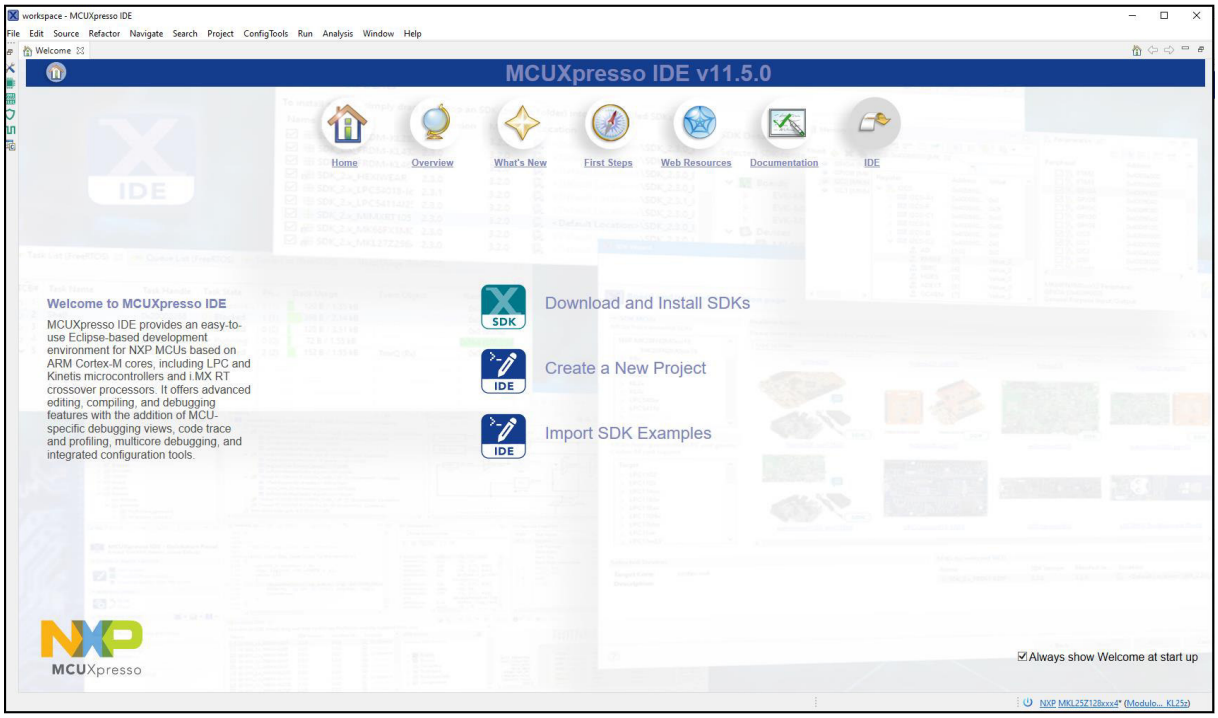
O procedimento de instalação do MCUXpresso é bastante simples, basta o usuário seguir os seguintes passos:

a) entre no site da NXP voltado para o MCUXpresso IDE;
b) cadastre-se e acesse com a sua conta no site da NXP, após confirmar o seu e-mail, clique em *download*;
c) clique em *download* irá direcionar para o aplicativo, onde de fato será o arquivo a ser baixado;
d) em seguida, leia os termos e aceite-os para dar sequência na instalação;
e) selecione o *software* correspondente com o seu sistema operacional, onde após o clique o aplicativo será baixado;
f) leia os termos de licença e de *software* e aceite-os para continuar;
g) selecione as pastas de destino, o nome do arquivo de atalho e se deseja que ele fique na área de trabalho;
h) será mostrado um resumo das opções selecionadas e clicando em *Install*, os arquivos serão instalados;
i) serão pedidas permissões dos *drivers* para o funcionamento da IDE, aceite-os para garantir o pleno funcionamento do MCUXpresso;
j) por fim, a instalação será concluída e o aplicativo estará pronto para uso.

Já na Figura 29 é possível observar a interface inicial da IDE após a instalação. Nela é visível tópicos como *download* e instalação de SDKs, criação de novos projetos, importações

de códigos exemplos pela SDK, além de documentações para iniciantes e usuários de níveis avançados. Ressalta-se que a visualização pode variar de acordo com a versão instalada pelo usuário, sendo que na construção deste trabalho está sendo utilizada a versão 11.5.0 da IDE MCUXpresso e já possui um projeto em andamento como se nota no canto inferior direito.

Figura 29 – Interface inicial MCUXpresso IDE



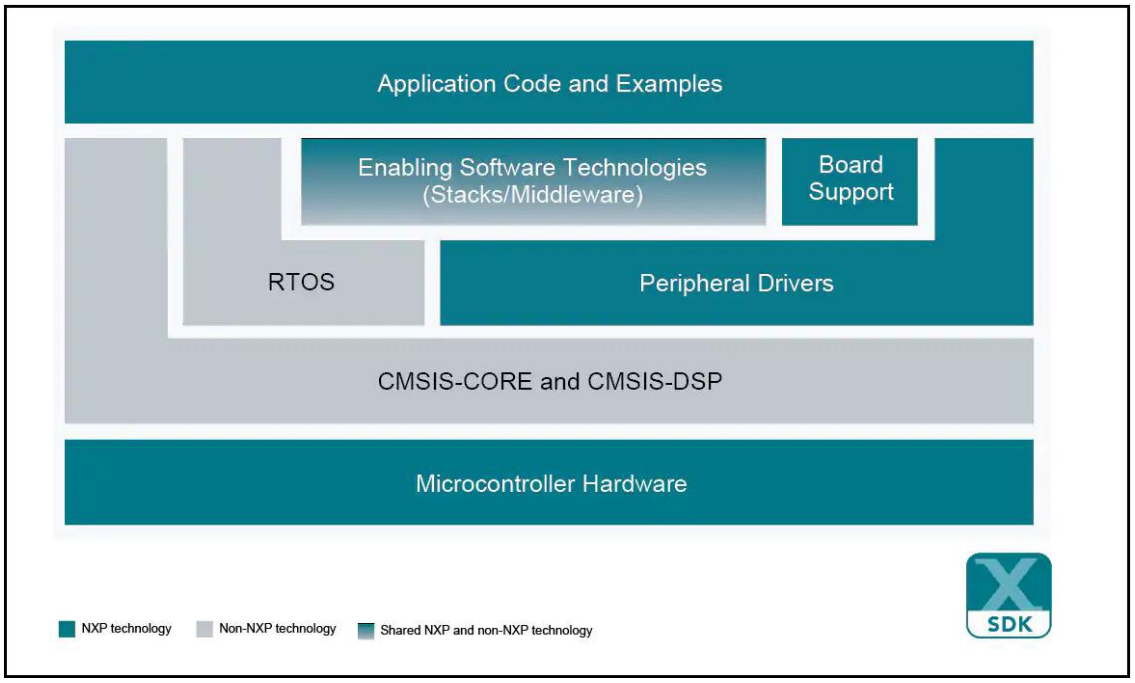
Fonte: elaborado pelo autor.

4.2 MCUXpresso SDK

Como foi mencionado anteriormente, a SDK é um pacote de ferramentas que irá auxiliar no desenvolvimento dos códigos, na otimização e facilitação quanto ao uso do MCUXpresso. Entretanto, a SDK não é necessária para conseguir realizar a programação e muito menos para depurar o código, no entanto a curva de aprendizagem torna-se mais dificultosa devido a sua ausência.

A Figura 30 mostra o diagrama de blocos da SDK do MCUXpresso, onde é possível visualizar de uma forma simplificada a interação dos códigos com o *hardware* do microcontrolador.

Figura 30 – Diagrama de blocos MCUXpresso SDK



Fonte: adaptado de Kit de Desenvolvimento de Software MCUXpresso (SDK) (2022, p. 2).

A própria SDK fornece códigos exemplos para usos de recursos como ADC, DAC, PWM e outros drivers, além de tornar a programação em um nível mais alto que o C/C++ no que tange as configurações de bits para habilitar e modificar os periféricos nas placas suportadas. Ou seja, oferecem recursos para simplificar certas rotinas que a certo ponto poderia dificultar o aprendizado e o uso da placa FRDM-KL25Z.

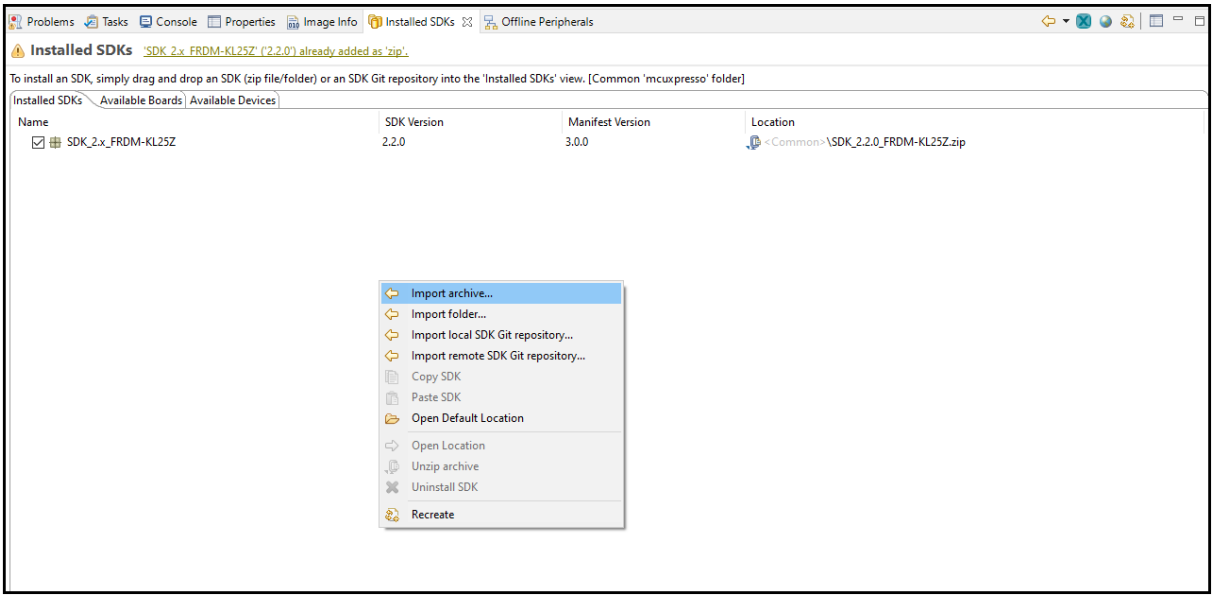
A sequência para baixar e instalar a SDK no MCUXpresso segue abaixo:

a) Acesse o sítio eletrônico MCUXpresso SDK *Builder* e clique em *Acess my SDK Dashboard*;
b) Em *Select Development Board*, busque no pesquisar e selecione a MKL25Z128xxx4;
c) Clique na opção *Build* MCUXpresso SDK, em seguida escolha o sistema operacional do seu computador;
d) Escolha o MCUXpresso como IDE e a opção de baremetal é suficiente para este trabalho, no entanto o usuário pode escolher outras opções como o RTOS;
e) Por fim, selecione todos os recursos que você deseja, caso queira adicionar mais recursos futuramente, logo deve ser seguido os passos anteriores para criar um novo item no *dashboard;*

f) Após apertar em *Download SDK* e aceitar os termos, clique em *Download SDK Archive* e o arquivo será baixo no formato .zip ou .rar.

Após o término do *download*, inicie o MCUXpresso IDE e aguarde a sua inicialização. Caso seja o primeiro uso, logo fecha a tela de bem-vindo e na parte inferior há uma janela similar ao que mostra a Figura 31. Clique com o botão direito no campo em branco de *Installed SDKs*, selecione *import archive*, após isso busque o arquivo baixado da SDK e aguarde a instalação no MCUXpresso. Finalizado o procedimento, o usuário terá acesso aos recursos disponibilizados pela SDK.

Figura 31 – Instalação da SDK



Fonte: elaborado pelo autor.

Com a finalização desses passos, o usuário está apto para utilizar o aplicativo MCUXpresso para programar o módulo didático. Por fim, dentro das pastas dos arquivos instalados no computador será possível observar documentos que possam ajudar na introdução e até mesmo na complementação do conhecimento para usuários de níveis mais avançados.

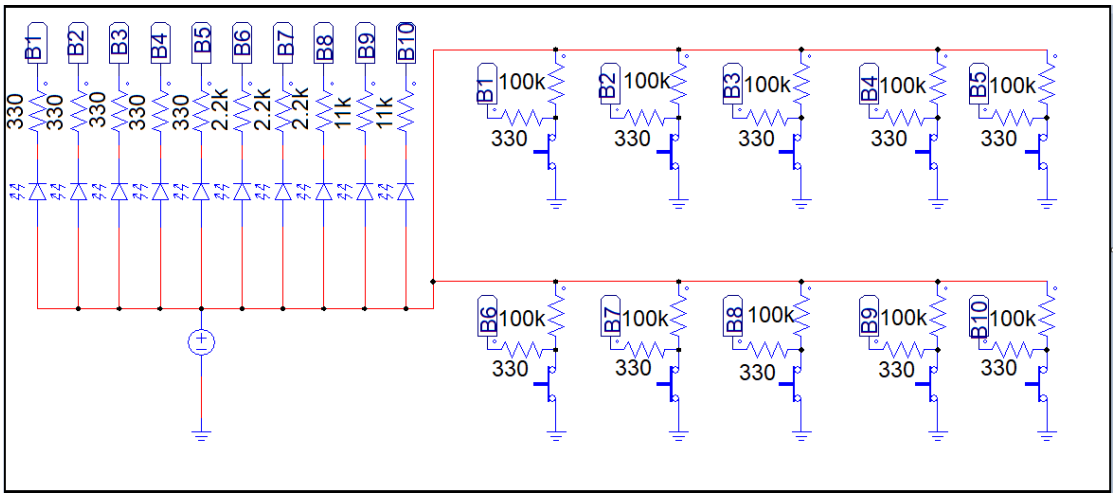
5 TESTES E SIMULAÇÕES

As propostas da criação desse módulo didático foram simuladas no *software* PSIM e/ou implementado um circuito similar ao desenhado para realizar os testes necessários para validar o projeto. Ressalta-se que serão mostrados os testes de forma isolada, ou seja, todos os componentes não serão testados de uma única vez. Tal motivo para esse teste de forma isolada é pelo fato de que haverá muita poluição visual e a dificuldade de montar todo o projeto em uma protoboard para atender todas as propostas do módulo didático.

5.1 LEDs e Botões

O primeiro circuito mostrado será a interação entre os LEDs e botões que estão presentes na Figura 32. Nesse caso em específico, o esquemático informará que as diferenças entre o módulo e a simulação estão na falta do capacitor para retirar o efeito de *bouncing* e a presença de resistores variados para mostrar a interação com a sua luminosidade.

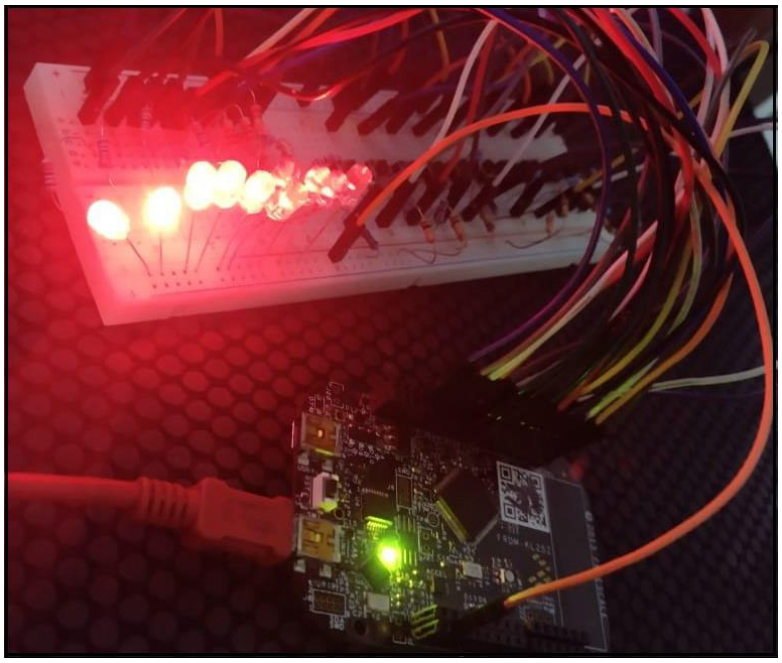
Figura 32 – Circuito simulado LEDs e botões



Fonte: elaborado pelo autor.

Já a Figura 33 mostra como ficou o circuito após a montagem prevista, nota-se que todos os LEDs estão ligados e a programação estava de forma que quando houvesse o pressionar do botão, o LED correspondente apaga. Além disso, é visível que os resistores com resistências maiores possuem uma menor intensidade luminosa em comparação com os outros de resistência menor.

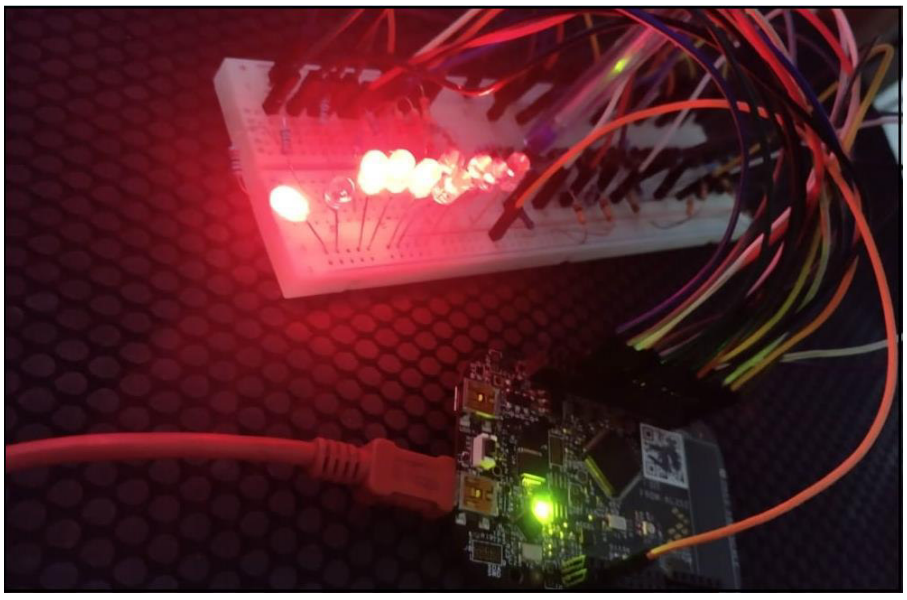
Figura 33 – Circuito montado LEDs e botões.



Fonte: elaborado pelo autor.

O teste realizado na Figura 34 corresponde ao pressionar um dos botões, o sinal lido é interpretado pela KL25Z, conforme o código disponibilizado no Apêndice A. Após a leitura do sinal, a interpretação do código é que o estado do pino do LED correspondente tornar-se-ia estado lógico baixo. Em suma, o LED apaga ao pressionar do botão. No caso, o botão que está localizado na ordem 2 apagou o LED da mesma ordem.

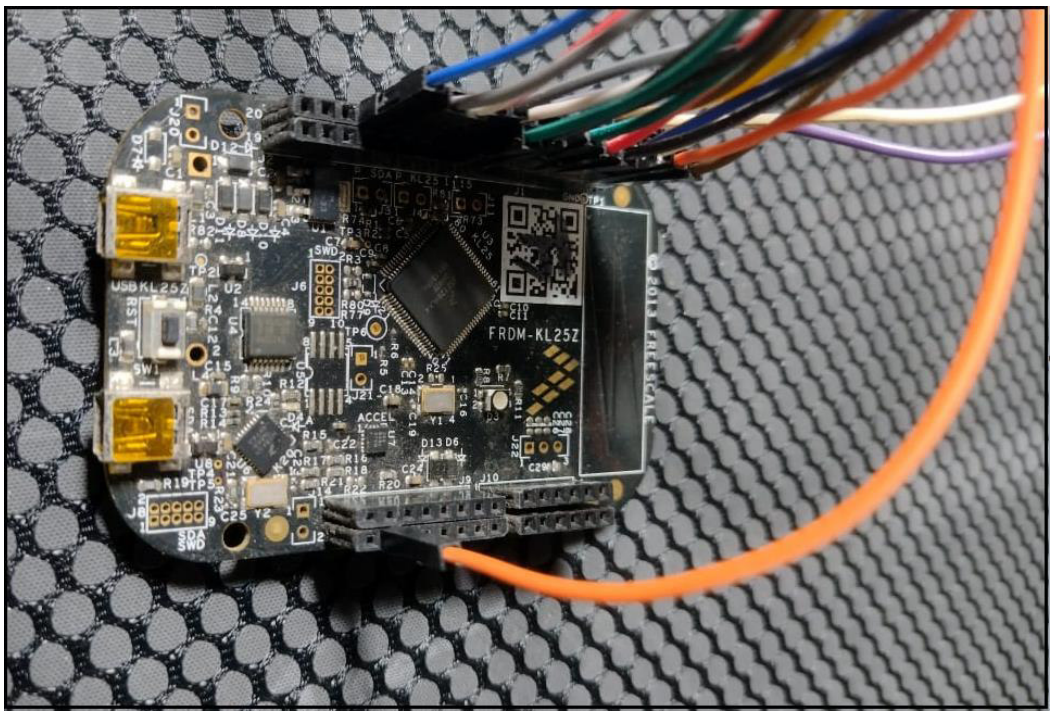
Figura 34 – Circuito testado LEDs e botões.



Fonte: elaborado pelo autor.

Por fim, foi utilizada a mesma pinagem dos LEDs e botões previstos no módulo didático. A visualização fica mais fácil na Figura 35. É possível notar que em um circuito simples como esse a densidade de componentes é moderada e que há uma eficiência maior de uso dos recursos e de tempo nos laboratórios caso a prototipagem já estivesse realizada.

Figura 35 – Visualização da placa no circuito LEDs e botões.

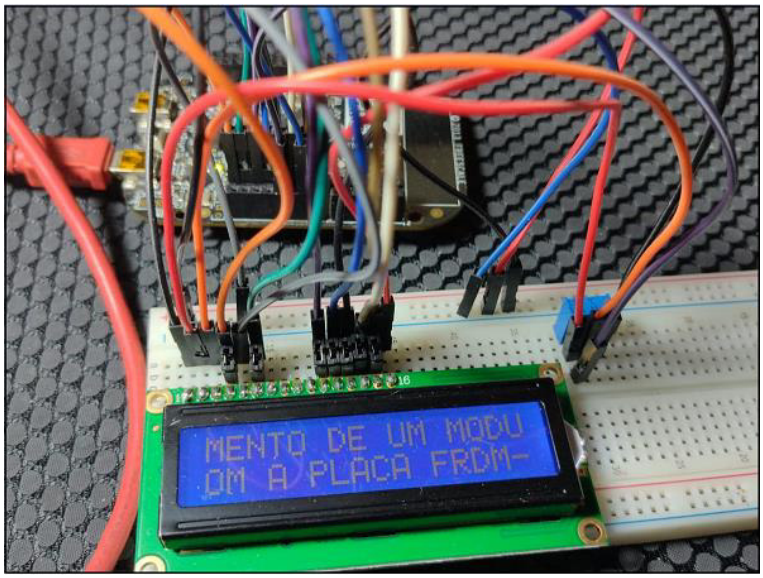


Fonte: elaborado pelo autor.

5.2 Circuito LCD

O circuito do LCD montado é igual ao proposto para o módulo didático, pois a Figura 36 apresenta as mesmas pinagens e inclusive são utilizados os conectores do tipo macho com interligação de *microjumpers* para exibir a ideia de modularização dos componentes e seccionamento das conexões.

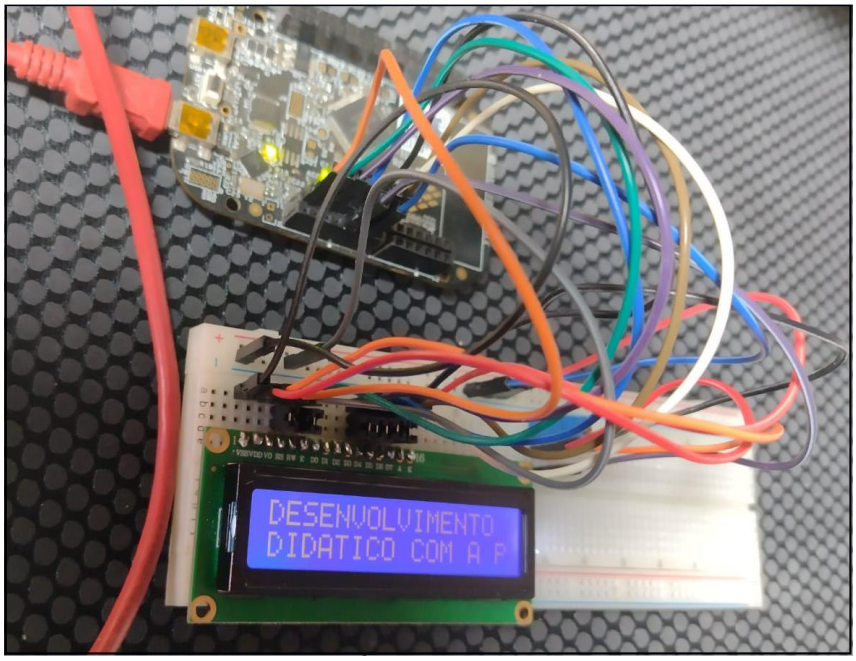
Figura 36 – Circuito LCD ligado.



Fonte: elaborado pelo autor.

O código disponibilizado no Apêndice B contêm funções diversas para o uso do LCD, desde a inicialização do equipamento até a realização do deslocamento. Na Figura 37, é realizado um teste utilizado a função de deslocamento para a esquerda.

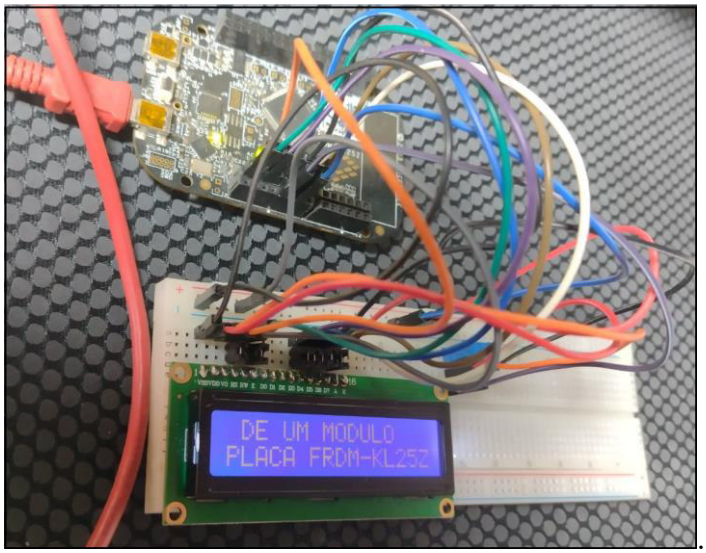
Figura 37 – Circuito LCD com deslocamento para esquerda.



Fonte: elaborado pelo autor.

O deslocamento conclui-se na Figura 38, com isso é possível utilizar 40 caracteres tanto na linha superior quanto a inferior, desde que seja utilizado um mecanismo para visualizar os outros caracteres que não estão presentes na primeira visualização.

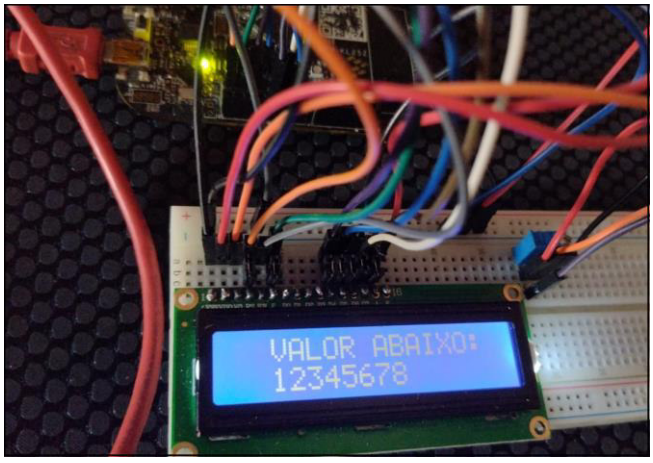
Figura 38 – Final do deslocamento dos caracteres do circuito LCD.



Fonte: elaborado pelo autor.

Por fim, foi utilizado uma função do código para impressão de números no display com deslocamento, para isso foi necessário realizar uma conversão do valor inteiro para decompor em cada número em caractere limitado a um tamanho de 8 espaços. Essa visualização encontra-se na Figura 39, onde na linha superior há um texto e embaixo há o número imposto no código.

Figura 39 – Circuito LCD testado com número.

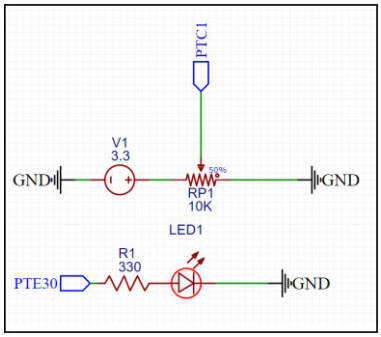


Fonte: elaborado pelo autor.

5.3 Circuito ADC e DAC

O Pino PTC1 da FRDM-KL25Z está conectado a um trimpot, a utilização se dá para a leitura de sinais analógicos pelo conversor ADC que está presente naquele pino. Além disso, pode ser utilizado um conversor DAC, por exemplo, para gerar uma saída analógica em função da interpretação do sinal capturado. A Figura 40 mostra o circuito simulado no EASYEDA.

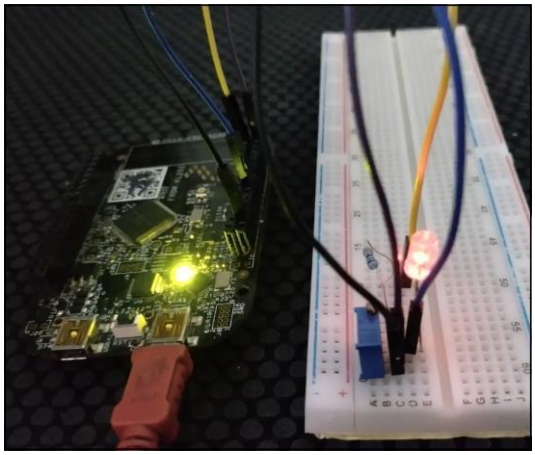
Figura 40 – Circuito leitura ADC com base no potenciômetro.



Fonte: elaborado pelo autor.

A configuração da prototipagem do teste ficou de acordo com a Figura 41, onde é possível observar a leitura do sinal analógico pelo pino intermediário do trimpot e a alimentação via conversor DAC para o LED. A intensidade da luminosidade era de acordo com a variação das resistências do potenciômetro. Para a simulação desse circuito, foram utilizados os códigos que estão no Apêndice C, D e E.

Figura 41 – Circuito leitura ADC montado.

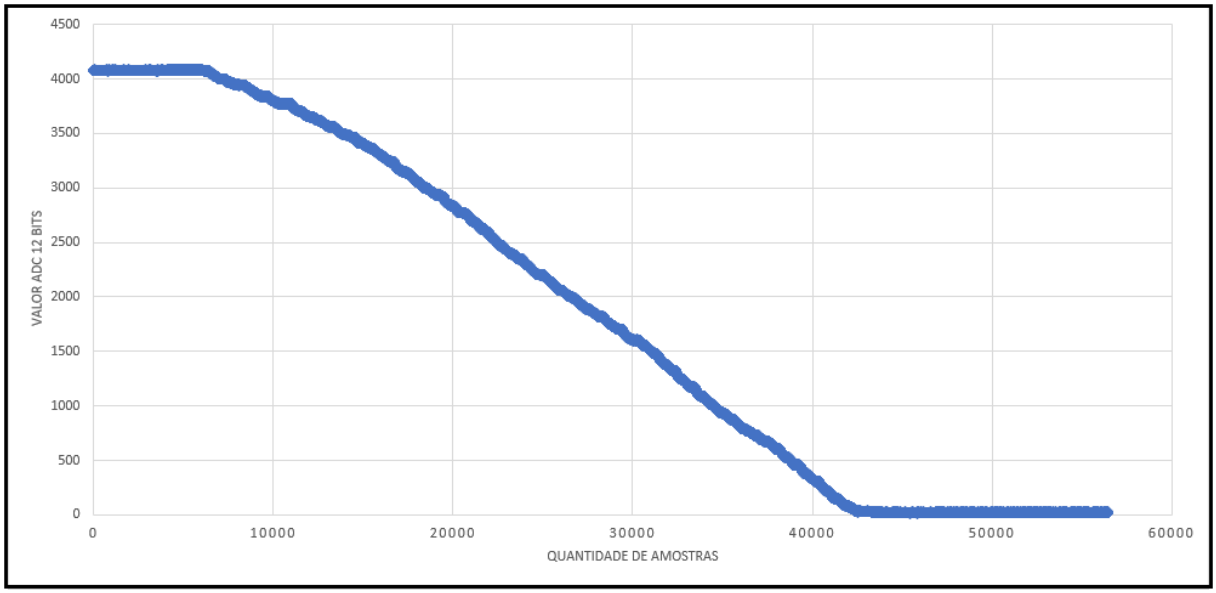


Fonte: elaborado pelo autor.

Para simular a captura do conversor ADC de 12 *bits,* foi utilizado o aplicativo Teraterm para a captura do *log* de leitura do sinal impresso no monitor serial, após isso foram coletados os dados e realizado um gráfico para visualizar a interpretação do conversor ADC mediante o ajuste do trimpot ao longo do tempo.

Conforme mostra o Gráfico 1, é possível observar os limites superiores e inferiores do eixo y que estão entre os valores de 0 a 4096, que é o valor mínimo de leitura do microcontrolador e o valor máximo para uma resolução de 12 *bits*. Por fim, ressalta-se que a leitura não é totalmente linear devido a questão mecânica do trimpot e o manuseio do usuário para a realização do teste.

Gráfico 1 – Circuito leitura ADC montado



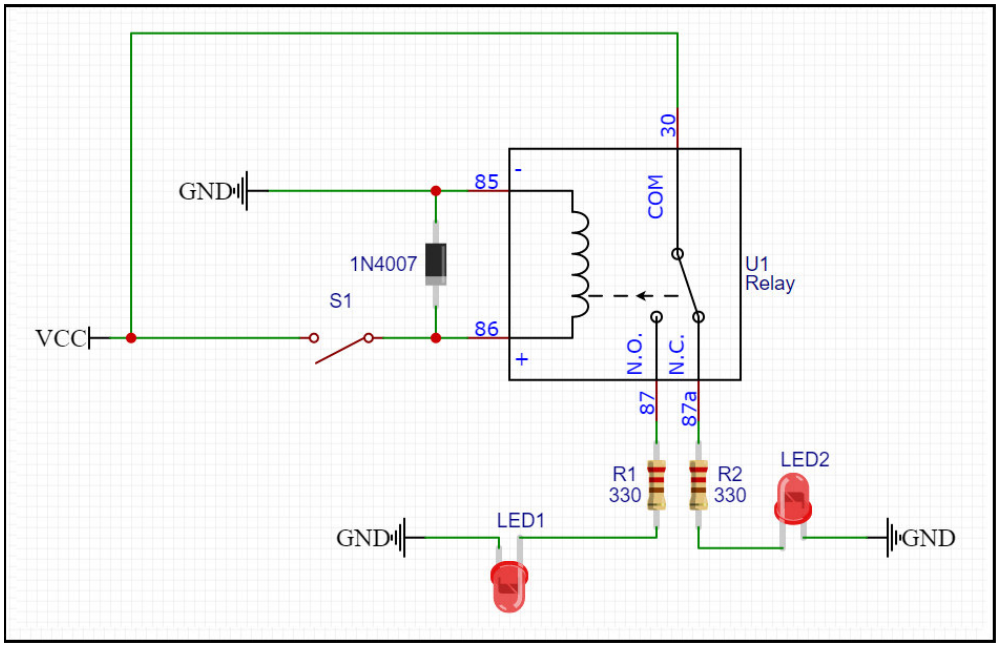
Fonte: elaborado pelo autor.

5.4 Circuito Relé

O circuito relé optoacoplado proposto do módulo didático possui uma isolação entre o sinal digital para o sinal de potência através do optoacoplador e o relé utilizado possui uma tensão de 5 V em suas características técnicas. Porém, o circuito testado será com um relé de 3.3V sem o optoacoplador devido a falta do componente projetado.

O esquemático do circuito para a simulação de um circuito similar ao proposto se encontra na Figura 42.

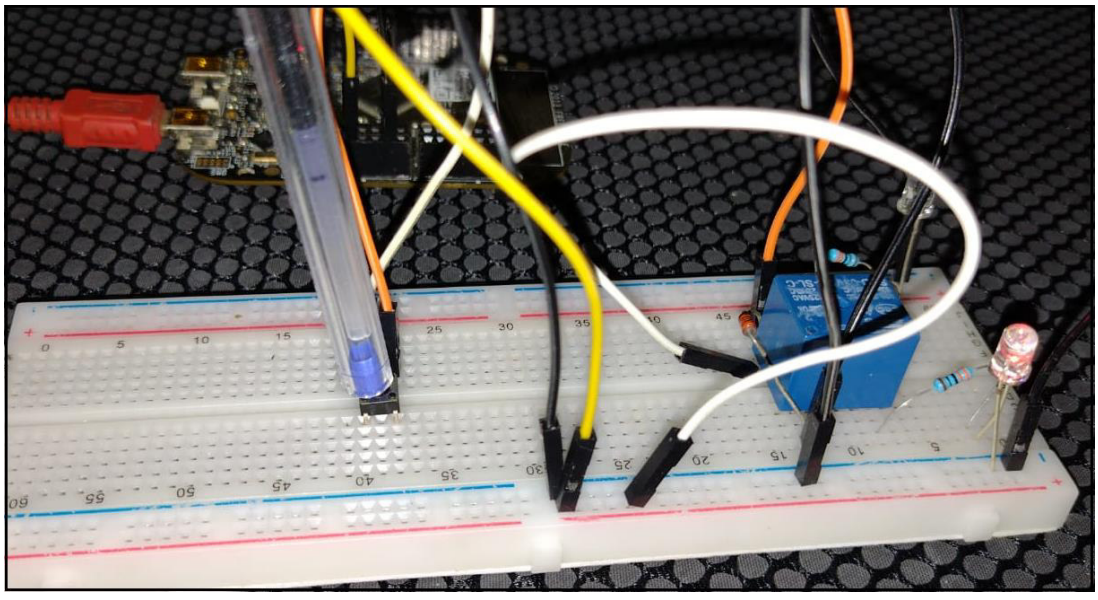
Figura 42 – Esquemático do circuito simulado do relé.



Fonte: elaborado pelo autor.

O funcionamento do circuito é bastante simplificado, para o circuito simulado ao pressionar o botão o LED1 é ligado enquanto o LED2 está desligado. Porém, ao deixar botão na configuração aberta, o LED2 é ligado e nesse caso o LED1 é desligado. A Figura 43 mostra a situação quando o botão está pressionado.

Figura 43 – Circuito relé com botão pressionado.

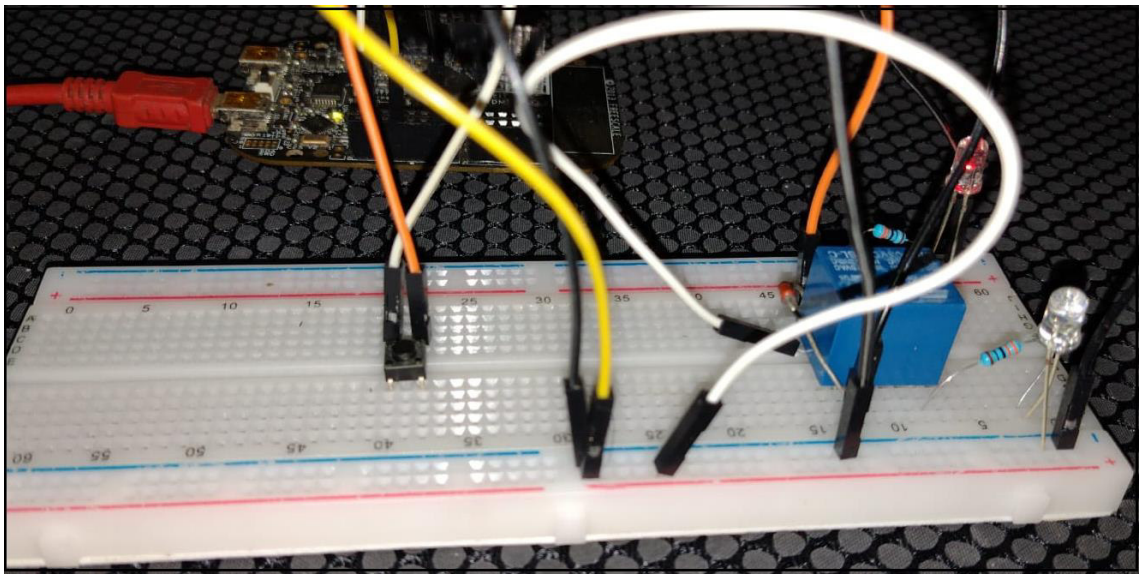


Fonte: elaborado pelo autor.

A proposta de acionamento do módulo didático é por um sinal de estado lógico alto para o optoacoplador em comparação com o circuito simulado, isso faz com que a condição lógica de acionamento seja maior devido a inserção da programação.

Por fim, pode ser visto na Figura 44, o último estado do circuito simulado que é o não pressionar do botão.

Figura 44 – Circuito relé com botão pressionado.



Fonte: elaborado pelo autor.

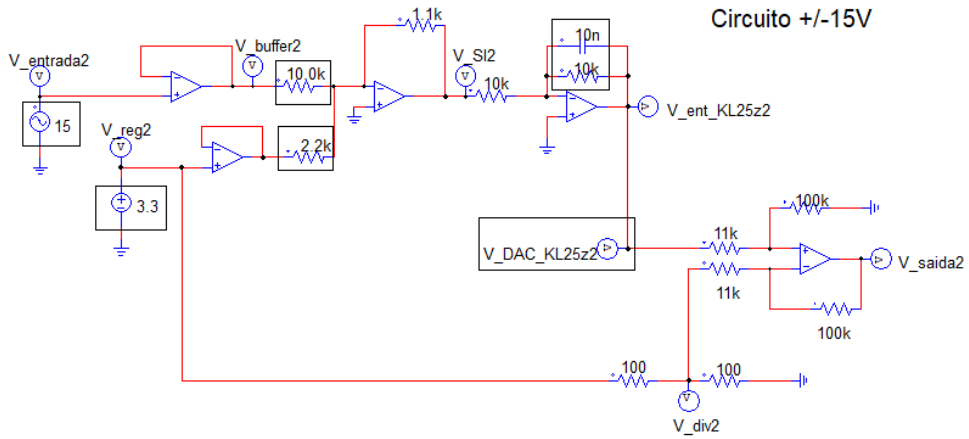
5.5 Circuito condicionador de sinais

Primeiramente, quando é realizado simulação é necessário entender que alguns pontos não serão condizentes com a realidade devido à natureza da construção de cada equipamento. É evidente que devem ser utilizados os principais mecanismos para evitar que esses tipos de problemas descaracterizem o resultado ideal, de tal forma a seleção de equipamentos e os cuidados nos ajustes são fundamentais para minimizar os efeitos.

Mas pode-se esperar que as fontes não são ideais, ou seja, possuem ruídos e por si só já afetam o comportamento do circuito. Os resistores apresentam faixas de erros toleráveis geralmente entre 5% e 10%. Os resistores variáveis podem trepidar e variar seus ajustes, assim como há o filtro passa-baixas que ocasiona um deslocamento angular entre o sinal de saída em relação ao de entrada. Além de que, há também as configurações e requisitos para a captura de um sinal adequado pelo ADC, já que a resolução de 16 *bits* consegue captar variações menores em relação ao de 12 *bits*.

O esquemático simplificado do circuito simulado no PSIM está conforme mostra a Figura 45. Nele é possível observar alguns pontos importantes como destacado anteriormente para o experimento real do circuito. Além do desvio entre a entrada da placa e o sinal que deve ser gerado pelo DAC ou PWM.

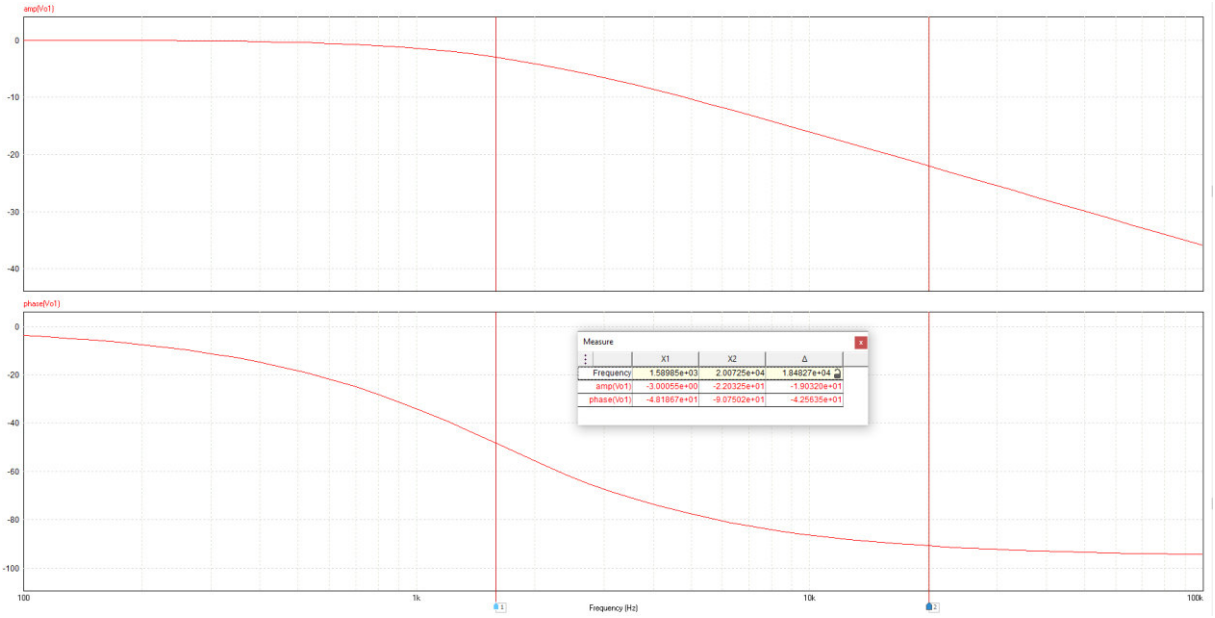
Figura 45 – Circuito condicionador de sinais simplificado simulado



Fonte: elaborado pelo autor.

Na Figura 46, é possível observar o diagrama de Bode do circuito passa-baixas na entrada do condicionador de sinais e o seu comportamento para diferentes frequências.

Figura 46 - Diagrama de Bode



Fonte: elaborado pelo autor.

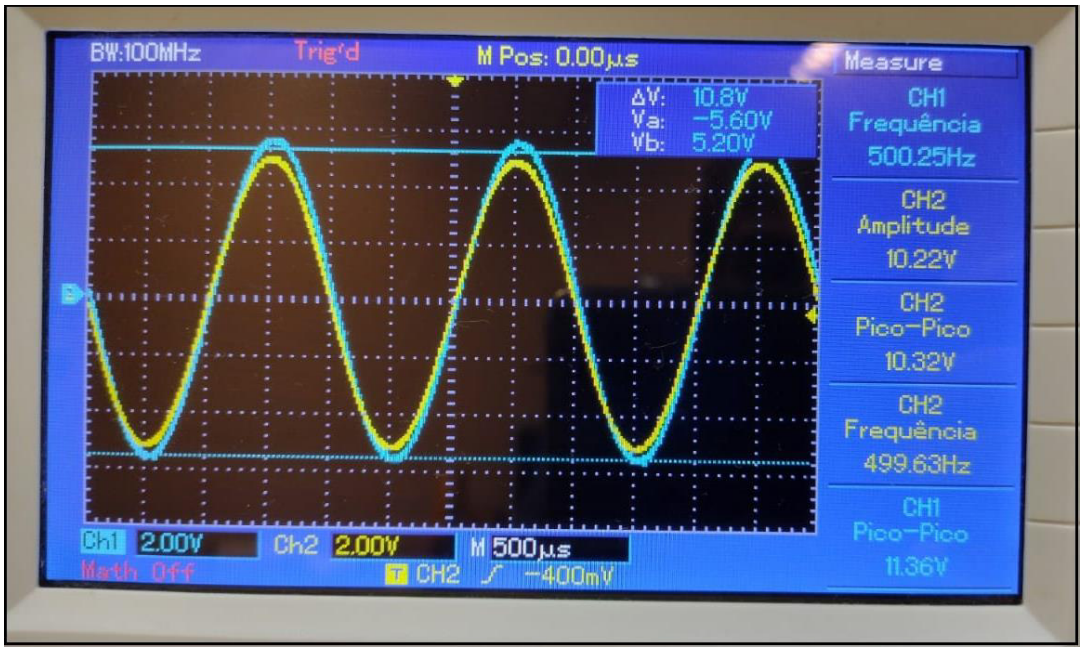
O circuito condicionador de sinais foi testado em laboratório com auxílio de fontes de bancada, osciloscópio e gerador de sinais como pode ser visualizado na Figura 47. Para isso foi realizado todo o ajuste de tensão +15V e -15V para a fonte simétrica, 3,3V para a fonte de bancada, um gerador de sinal com onda senoidal com frequência de 500Hz para tensões de 20V e 10V pico a pico.

Figura 47 – Circuito experimental do condicionador de sinais

Fonte: elaborado pelo autor.

Na Figura 48, já é possível observar os fatos citados anteriormente no que tange a diferença de um circuito simulado para um circuito em condições reais. Pois, mesmo sem o capacitor no filtro passa-baixas é notável que a onda de entrada é diferente da de saída perante aos erros toleráveis inerentes aos equipamentos usados.

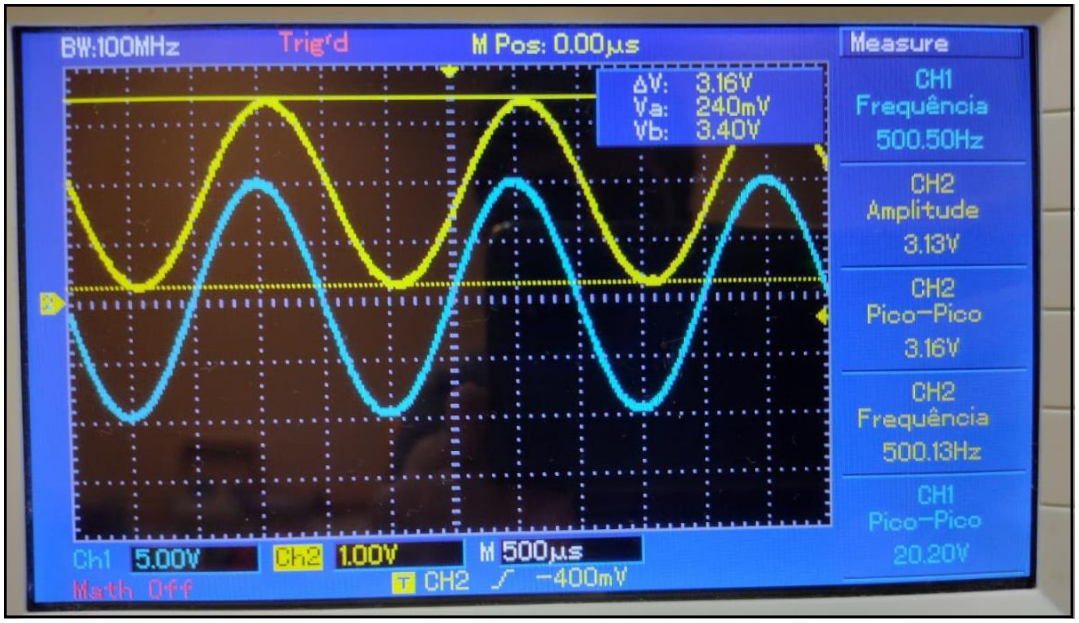
Figura 48 – Erros experimentais do circuito condicionador de sinais



Fonte: elaborado pelo autor.

Na Figura 49, temos a onda na cor amarela sendo o sinal interpretado pelo conversor ADC, enquanto o sinal em azul é o valor de entrada. Para isso, foi testado com o gerador de sinal com tensão em 10V pico a pico. Ressalta-se que para esse teste o capacitor do filtro passa-baixas já está no seu devido local, por isso há uma diferença de tempo perceptível entre as ondas.

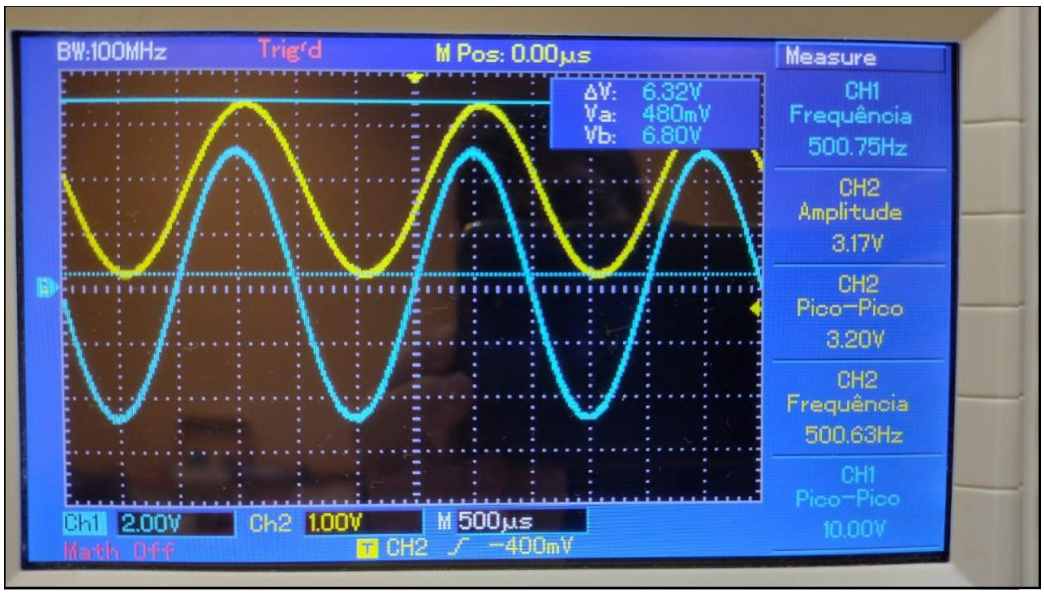
Figura 49 – Circuito de entrada do condicionador de sinais com 20V pico a pico



Fonte: elaborado pelo autor.

Já pela Figura 50, é vista uma simulação semelhante a anterior, porém agora com o gerador de sinais com onda de 10V pico a pico.

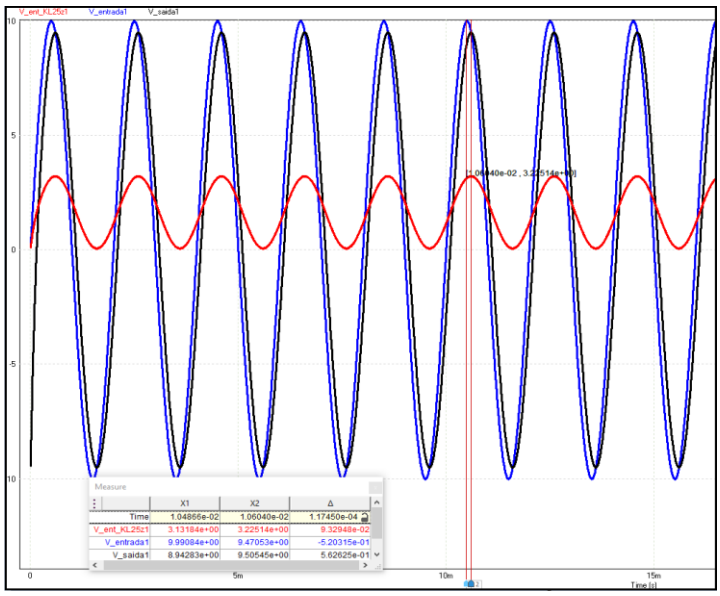
Figura 50 – Circuito de entrada do condicionador de sinais com 10V pico a pico



Fonte: elaborado pelo autor.

Após essa introdução de simulações experimentais, será mostrado o resultado do circuito simulado. A onda em azul é o sinal de entrada, a onda em preta o sinal de saída e em vermelho é o valor na entrada do circuito ADC. A Figura 51 mostra o resultado da simulação.

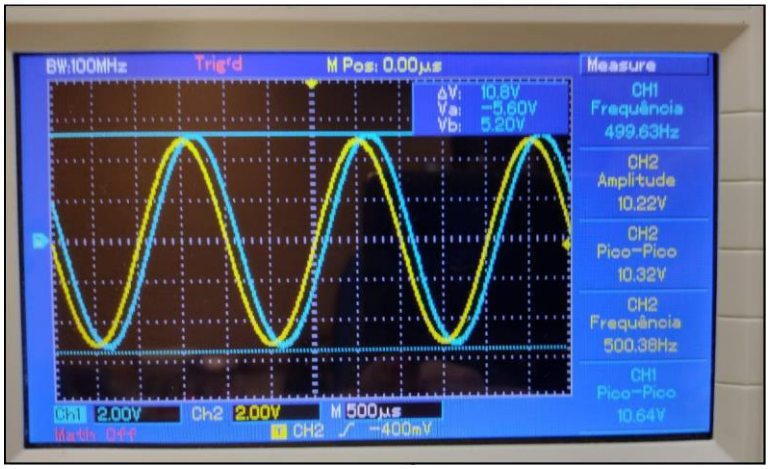
Figura 51 – Circuito simulado do condicionador de sinais com 20V pico a pico



Fonte: elaborado pelo autor.

Com a Figura 52, comprova-se os fatos observados no circuito simulado e que os erros toleráveis conseguem distorcer levemente o esperado, porém existem soluções para minimizar tais ocasiões. O código utilizado para o circuito condicionador de sinais está no Apêndice F.

Figura 52 – Circuito experimental do condicionador de sinais com 20V pico a pico



Fonte: elaborado pelo autor.

Mediante essas influências negativas que ocasionam em circuitos reais, é interessante que os resistores utilizados sejam de precisão ou utilizar potenciômetros com ajustes precisos. Além disso, no projeto inicial estava sendo previsto a utilização de regulador de tensão (AMS1117 3,3V) para a entrada de 3,3V, porém como a sua saída não é precisa foi contornado para um gerador de tensão de referência LT1461 versão 3,3V. Assim, os usos desses componentes estão previstos no projeto do módulo didático para melhorar a experiência em circuitos que necessitam de uma precisão nos testes.

6 CONCLUSÃO

A proposta do módulo didático desde a sua concepção visava a utilização dos seus recursos de uma forma mais otimizada e permitindo a interação humano-computador. Com os recursos de construção de uma placa de circuito impresso, utilização de pinos e *microjumpers,* modularização dos componentes da placa, utilização de aplicativos que maximizam o potencial criativo dos usuários e a linguagem selecionada faz com que esses objetivos iniciais estejam cumpridos.

As simulações e os resultados obtidos nos circuitos dos botões, LEDs, LCD, Leitura ADC por potenciômetro, relé e o condicionador de sinais são considerados um sucesso mesmo com algumas divergências ocasionadas pelos modelos reais de tais equipamentos, já que algumas soluções se encontram no projeto do modelo didático final.

A utilização do módulo em práticas de acionamentos por microcontrolador na disciplina de Microprocessadores 1, a coleta de dados para a modelagem de um circuito na disciplina de Controle Discreto e a integração com os laboratórios de Eletrônica Analógica com os recursos do ADC e DAC são exemplos de propostas de melhorias devido ao uso do módulo didático.

Com a finalização deste trabalho foi possível observar que toda a parte de simulações, experimentos e projetos se conclui em um módulo didático preparado para receber diversos tipos de atividades em aulas em laboratórios quanto na realização de trabalhos de cunho científico.

As sugestões de continuidade para este trabalho são a utilização da linguagem C++, busca por componentes mais precisos, expansão dos limites operacionais do condicionador de sinais, utilização de componentes para promover a proteção dos equipamentos, inserir fontes de alimentação alternativas para a alimentação elétrica do módulo e a adição de novos circuitos para ampliar o potencial do módulo didático.

REFERÊNCIAS

ALEXANDER, Charles K.; SADIKU, Matthew. **Fundamentos de Circuitos Elétricos**. 5. ed. New York: Amgh, 2013. 896 p.

BOURNS. **Resistores variáveis**. Disponível em: https://www.filipeflop.com/img/files/download/Datasheet_Potenciometro_Multivoltas_3296.pdf. Acesso em: 01 dez. 2022.

BRAGA, Newton C.. **Relés**: circuitos e aplicações. [S. L.]: Editora Newton C. Braga, 2012. 150 p. Disponível em: https://books.google.com.br/books?hl=pt-BR&lr=&id=DXAEDgAAQBAJ&oi=fnd&pg=PA12&dq=rel%C3%A9+newton+braga&ots=ELmVcO3kZp&sig=74xw_IxhGXKggVWPyTeVC15-Te0#v=onepage&q=rel%C3%A9%20newton%20braga&f=false. Acesso em: 02 nov. 2022.

COLEGIUM. **Módulos didáticos digitais**. 2022. Disponível em: https://www.colegium.com/modulos-didaticos-digitais/?lang=pt-br. Acesso em: 02 nov. 2022.

DEVELOPER, Arm. **Arm GNU Toolchain**. Disponível em: https://developer.arm.com/Tools%20and%20Software/GNU%20Toolchain. Acesso em: 02 nov. 2022.

DEVICES, Analog. **Circuito Integrado LT1461 3,3V**. Disponível em: https://www.analog.com/media/en/technical-documentation/data-sheets/LT1461.pdf. Acesso em: 01 dez. 2022.

ECKERT, Guilherme Wagner. **PROJETO DE HARDWARE E SOFTWARE PARA A PLATAFORMA FRDM-KL25Z**. 2016. 167 f. TCC (Graduação) - Curso de Engenharia Elétrica, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2016. Disponível em: https://edisciplinas.usp.br/pluginfile.php/4093515/mod_resource/content/1/Projeto%20de%20Hardware%20e%20Software%20utilizando%20FRDM%20KL25Z.pdf. Acesso em: 23 jul. 2022.

INSTRUMENTS, Texas. **Circuito Integrado LM324N**. Disponível em: https://www.filipeflop.com/img/files/download/Datasheet_lm324-n.pdf. Acesso em: 01 dez. 2022.

LEARNINGMICRO. **Interfacing 16×2 LCD with KL25Z Series MCU**. 2018. Disponível em: https://learningmicro.wordpress.com/interfacing-lcd-with-kl25z-freedom-board/. Acesso em: 02 nov. 2022.

MOTA, Francisco Jandysley Aguiar. **DESENVOLVIMENTO DE UM CIRCUITO CONDICIONADOR DE SINAIS PARA APLICAÇÃO EM SISTEMAS DE CONTROLE**. 2016. 43 f. TCC (Graduação) - Curso de Engenharia Elétrica, Universidade Federal do Ceará, Fortaleza, 2016.

NILSSON, James W.; RIEDEL, Susan A.. **Circuitos Elétricos**. 10. ed. [S. L.]: Pearson, 2015. 816 p.

RELAY, Songle. **Relé eletromecânico**. Disponível em: https://www.filipeflop.com/img/files/download/Datasheet_Rele_Songle_SRD-05VDC-SL-C.pdf. Acesso em: 01 dez. 2022.

ROBÓTICA, Autocore. **Botão de Toque 6x6mmx4,3mm**. Disponível em: https://www.autocorerobotica.com.br/botao-de-toque-6x6mm. Acesso em: 21 set. 2022

ROBÓTICA, Autocore. **Display Lcd 16x2 com fundo azul**. Disponível em: https://www.autocorerobotica.com.br/display-lcd-16x2-hd44780. Acesso em: 21 set. 2022.

ROBÓTICA, Autocore. **Led Alto Brilho 5mm Vermelho**. Disponível em: https://www.autocorerobotica.com.br/led-alto-brilho-5mm-vermelho. Acesso em: 21 set. 2022.

SACCO, Francesco. **10 mandamentos da PCB**. 2015. Disponível em: https://embarcados.com.br/10-mandamentos-da-pcb/. Acesso em: 02 nov. 2022.

SEMICONDUCTORS, Nxp. **Diagrama de blocos MCUXpresso SDK**. Disponível em: https://www.nxp.com/design/software/development-software/mcuxpresso-software-and-tools-/mcuxpresso-software-development-kit-sdk:MCUXpresso-SDK?#documentation. Acesso em: 02 dez. 2022.

SEMICONDUCTORS, Nxp. **FRDM-KL25Z User's Manual**. 2013. Disponível em: https://www.nxp.com/webapp/sps/download/preDownload.jsp?render=true. Acesso em: 02 nov. 2022.

SEMICONDUCTORS, Nxp. **Getting Started with MCUXpresso SDK**. 2017. Disponível em: https://community.nxp.com/pwmxy87654/attachments/pwmxy87654/kinetis-design-studio/9324/1/Getting%20Started%20with%20MCUXpresso%20SDK.pdf. Acesso em: 02 nov. 2022.

SEMICONDUCTORS, Freescale. **Kinetis KL25 Sub-Family**: 48 mhz cortex-m0+ based microcontroller with usb. 48 MHz Cortex-M0+ Based Microcontroller with USB. 2014. Disponível em: https://www.nxp.com/docs/en/data-sheet/KL25P80M48SF0.pdf. Acesso em: 02 nov. 2022.

SEMICONDUCTOR, Freescale. **KL25 Sub-Family Reference Manual**. 2012. Disponível em: https://edisciplinas.usp.br/pluginfile.php/4510188/mod_resource/content/1/KL25P80M48SF0RM.pdf. Acesso em: 13 ago. 2022.

SEMICONDUCTORS, Nxp. **MCUXpresso Integrated Development Environment (IDE)**. 2022. Disponível em: https://www.nxp.com/design/software/development-software/mcuxpresso-software-and-tools-/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDE. Acesso em: 02 nov. 2022.

SEMICONDUCTORS, Nxp. **NXP and Freescale Announce $40 Billion Merger**. 2015. Disponível em: https://www.nxp.com/company/about-nxp/nxp-and-freescale-announce-40-billion-merger:NW-FREESCALE-40BILLION-MERGE. Acesso em: 02 nov. 2022.

SEMICONDUCTORS, Vishay. **Optocoupler, Phototransistor Output, with Base Connection**. 2010. Disponível em: https://www.vishay.com/docs/81181/4n35.pdf. Acesso em: 02 nov. 2022.

STMICROELECTRONICS. **Circuito Integrado L7805**. Disponível em: https://storage.googleapis.com/baudaeletronicadatasheet/L7800.pdf. Acesso em: 01 dez. 2022.

VMWARE. **What is a bare metal hypervisor?** Disponível em: https://www.vmware.com/topics/glossary/content/bare-metal-hypervisor.html. Acesso em: 02 nov. 2022.

APÊNDICE A – CÓDIGO GPIO

```
/*
 * gpio.h
 *
 *  Created on: 27 de ago de 2022
 *      Author: Alysson
 */

#ifndef GPIO_H_
#define GPIO_H_
// Bibliotecas -----------------------------------------------------------------
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "int.h"
#include "adc16.h"
#include "fsl_pit.h"
#include "fsl_port.h"
#include "fsl_common.h"
#include "fsl_adc16.h"
#include "fsl_gpio.h"
#include "fsl_port.h"

// Funções ---------------------------------------------------------------------
// Função para habilitar um pino em específico como entrada
// Exemplo: gpi_init (PORTA, GPIOA, 3) Habilita o PORTA, GPIOA, Pino 3 como
entrada
void gpi_init(PORT_Type* port_, GPIO_Type* gpio_, int pino_);

// Função para habilitar um pino em específico como saída com um valor definido
// Exemplo: gpo_init (PORTA, GPIOA, 3, 1) Habilita o PORTA, GPIOA, Pino 3 como
saída com valor 1 (ligado)
void gpo_init(PORT_Type* port_, GPIO_Type* gpio_, int pino_, uint8_t onoff_);

// Função para ligar um pino/ definir o estado lógico como alto (Saída 1).
// Att: Função funciona somente para pinos definidos como saída.
// Exemplo:liga (GPIOA, 3)
void liga (GPIO_Type* gpio, int pino);

// Função para desligar um pino/ definir o estado lógico como baixo (Saída 0).
// Att: Função funciona somente para pinos definidos como saída.
void desliga (GPIO_Type* gpio, int pino);

// Função padrão para iniciar todos os botões e leds com uma configuração
padrão.
// Botões como entrada e LEDs como saída;
void init_botoesled();



#endif /* GPIO_H_ */
```

```c
/*
 * gpio.c
 *
 *  Created on: 27 de ago de 2022
 *      Author: Alysson
 */

// Bibliotecas ----------------------------------------------------------------
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "int.h"
#include "adc16.h"
#include "fsl_pit.h"
#include "fsl_port.h"
#include "fsl_common.h"
#include "fsl_adc16.h"
#include "fsl_gpio.h"
#include "gpio.h"

// Variáveis ------------------------------------------------------------------
GPIO_Type* gpio;
PORT_Type* port;
int pino;
uint8_t onoff;

// Funções --------------------------------------------------------------------
// Recebe o tipo de PORT, GPIO e PINO
// Cria variáveis locais e habilita os clocks para todos os PORTS independente
do selecionado
// Inicia o pino com as configurações de entrada
void gpi_init(PORT_Type* port_, GPIO_Type* gpio_, int pino_){
	gpio = gpio_;
	pino = pino_;
	port=port_;
	gpio_pin_config_t entrada;
	entrada.pinDirection=kGPIO_DigitalInput;
	PORT_SetPinMux(port, pino, kPORT_MuxAsGpio);
	CLOCK_EnableClock(kCLOCK_PortA);
	CLOCK_EnableClock(kCLOCK_PortB);
	CLOCK_EnableClock(kCLOCK_PortC);
	CLOCK_EnableClock(kCLOCK_PortD);
	CLOCK_EnableClock(kCLOCK_PortE);
	GPIO_PinInit(gpio, pino, &entrada);

}

// Recebe o tipo de PORT, GPIO e PINO e o estado de saída
// Cria variáveis locais e habilita os clocks para todos os PORTS independente
do selecionado
// Inicia o pino com as configurações de saída
void gpo_init(PORT_Type* port_, GPIO_Type* gpio_, int pino_, uint8_t onoff_){
	gpio = gpio_;
	pino = pino_;
	onoff = onoff_;
```

```
	port=port_;
	gpio_pin_config_t saida;
	saida.outputLogic=onoff;
	saida.pinDirection=kGPIO_DigitalOutput;
	PORT_SetPinMux(port, pino, kPORT_MuxAsGpio);
	CLOCK_EnableClock(kCLOCK_PortA);
	CLOCK_EnableClock(kCLOCK_PortB);
	CLOCK_EnableClock(kCLOCK_PortC);
	CLOCK_EnableClock(kCLOCK_PortD);
	CLOCK_EnableClock(kCLOCK_PortE);
	GPIO_PinInit(gpio, pino, &saida);
}

// Recebe o tipo de  GPIO e PINO
// Coloca o pino selecionado para nível lógico alto
void liga (GPIO_Type* gpio, int pino){
	GPIO_WritePinOutput(gpio, pino, 1);

}

// Recebe o tipo de  GPIO e PINO
// Coloca o pino selecionado para nível lógico baixo
void desliga (GPIO_Type* gpio, int pino){
	GPIO_WritePinOutput(gpio, pino, 0);

}



// Função para iniciar todos os botões como pinos de entrada
// e o LEDS como pinos de saída.
void init_botoesled(){
	// Habilita os clocks dos botões e leds
	CLOCK_EnableClock(kCLOCK_PortA);
	CLOCK_EnableClock(kCLOCK_PortC);
	CLOCK_EnableClock(kCLOCK_PortD);
	// Cria a configuração dos botões e inicia cada pino dos botões no módulo
	gpio_pin_config_t botao;
	botao.pinDirection=kGPIO_DigitalInput;
	GPIO_PinInit(GPIOA, 13, &botao);
	GPIO_PinInit(GPIOA, 12, &botao);
	GPIO_PinInit(GPIOA, 5, &botao);
	GPIO_PinInit(GPIOA, 4, &botao);
	GPIO_PinInit(GPIOD, 4, &botao);
	GPIO_PinInit(GPIOD, 5, &botao);
	GPIO_PinInit(GPIOD, 0, &botao);
	GPIO_PinInit(GPIOD, 2, &botao);
	GPIO_PinInit(GPIOD, 3, &botao);
	GPIO_PinInit(GPIOD, 1, &botao);
	// Define os pinos dos PORTs abaixo como GPIO para ser de uso geral.
	PORT_SetPinMux(PORTA, 13, kPORT_MuxAsGpio);
	PORT_SetPinMux(PORTA, 5, kPORT_MuxAsGpio);
	PORT_SetPinMux(PORTA, 4, kPORT_MuxAsGpio);
	PORT_SetPinMux(PORTA, 12, kPORT_MuxAsGpio);
	PORT_SetPinMux(PORTD, 4, kPORT_MuxAsGpio);
	PORT_SetPinMux(PORTD, 5, kPORT_MuxAsGpio);
	PORT_SetPinMux(PORTD, 0, kPORT_MuxAsGpio);
	PORT_SetPinMux(PORTD, 2, kPORT_MuxAsGpio);
	PORT_SetPinMux(PORTD, 3, kPORT_MuxAsGpio);
```

```
        PORT_SetPinMux(PORTD, 1, kPORT_MuxAsGpio);
        // Cria a configuração dos LEDS como saída
        gpio_pin_config_t led;
        led.outputLogic=0;
        led.pinDirection=kGPIO_DigitalOutput;
        // Inicia os pinos com a configuração de LED
        GPIO_PinInit(GPIOC, 7, &led);
        GPIO_PinInit(GPIOC, 0, &led);
        GPIO_PinInit(GPIOC, 3, &led);
        GPIO_PinInit(GPIOC, 4, &led);
        GPIO_PinInit(GPIOC, 5, &led);
        GPIO_PinInit(GPIOC, 6, &led);
        GPIO_PinInit(GPIOC, 10, &led);
        GPIO_PinInit(GPIOC, 11, &led);
        GPIO_PinInit(GPIOC, 12, &led);
        GPIO_PinInit(GPIOC, 13, &led);
        // Define os pinos dos PORTs abaixo como GPIO para ser de uso geral.
        PORT_SetPinMux(PORTC, 7, kPORT_MuxAsGpio);
        PORT_SetPinMux(PORTC, 0, kPORT_MuxAsGpio);
        PORT_SetPinMux(PORTC, 3, kPORT_MuxAsGpio);
        PORT_SetPinMux(PORTC, 4, kPORT_MuxAsGpio);
        PORT_SetPinMux(PORTC, 5, kPORT_MuxAsGpio);
        PORT_SetPinMux(PORTC, 6, kPORT_MuxAsGpio);
        PORT_SetPinMux(PORTC, 10, kPORT_MuxAsGpio);
        PORT_SetPinMux(PORTC, 11, kPORT_MuxAsGpio);
        PORT_SetPinMux(PORTC, 12, kPORT_MuxAsGpio);
        PORT_SetPinMux(PORTC, 13, kPORT_MuxAsGpio);


}



/** Arquivo fonte
 * @file    Modulo_FRDM_KL25z.c
 * @brief   Application entry point.
 */
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "int.h"
#include "adc16.h"
#include "fsl_pit.h"
#include "fsl_port.h"
#include "fsl_common.h"
#include "fsl_adc16.h"
#include "fsl_gpio.h"
#include "gpio.h"

int main(void) {


    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();
```

```
    BOARD_InitDebugConsole();
init_botoesled();
// Laço infinito para que os LEDs apaguem de acordo com o botão apertado

    while(1) {

      if (GPIO_ReadPinInput(GPIOD, 1)==0)
      {
            liga(GPIOC, 13);
      }
      else
      {
            desliga(GPIOC, 13);
      }

      if (GPIO_ReadPinInput(GPIOD, 3)==0)
      {
            liga(GPIOC, 12);
      }
      else
      {
            desliga(GPIOC, 12);
      }

      if (GPIO_ReadPinInput(GPIOD, 2)==0)
      {
            liga(GPIOC, 11);
      }
      else
      {
            desliga(GPIOC, 11);
      }

      if (GPIO_ReadPinInput(GPIOD, 0)==0)
      {
            liga(GPIOC, 10);
      }
      else
      {
            desliga(GPIOC, 10);
      }

      if (GPIO_ReadPinInput(GPIOD, 5)==0)
      {
            liga(GPIOC, 6);
      }
      else
      {
            desliga(GPIOC, 6);
      }

      if (GPIO_ReadPinInput(GPIOA, 13)==0)
      {
            liga(GPIOC, 5);
      }
      else
      {
            desliga(GPIOC, 5);
      }
```

```
      if (GPIO_ReadPinInput(GPIOA, 5)==0)
      {
            liga(GPIOC, 4);
      }
      else
      {
            desliga(GPIOC, 4);
      }

      if (GPIO_ReadPinInput(GPIOA, 4)==0)
      {
            liga(GPIOC, 3);
      }
      else
      {
            desliga(GPIOC, 3);
      }

      if (GPIO_ReadPinInput(GPIOA, 12)==0)
      {
            liga(GPIOC, 0);
      }
      else
      {
            desliga(GPIOC, 0);
      }

      if (GPIO_ReadPinInput(GPIOA, 4)==0)
      {
            liga(GPIOC, 7);
      }
      else
      {
            desliga(GPIOC, 7);
      }

    }
    return 0 ;
}
```

APÊNDICE B – CÓDIGO LCD

```
#ifndef _lcd_H_
#define _lcd_H_
//-----------------------------------------------------------------------
// Variables
//-----------------------------------------------------------------------
//const unsigned char initLCD[8];
//extern const unsigned char upper_line[];
//volatile long count;
//volatile bool pitIsrFlag[2];

#include "fsl_port.h"
#include "fsl_gpio.h"
#include "adc.h"
//-----------------------------------------------------------------------
// Function Prototypes
//-----------------------------------------------------------------------
//Função: gera atraso, recebe como parâmetro a variável tempo.
//Ex.: Se passar a variável 2, retornará um delay de 2ms.
void delay(unsigned long time);

//Função: Habilita o LCD.
void enable(void);

//Função: Inicializa e habilita o uso do LCD, além habilitar o PIT para gerar a
função delay.
//SEMPRE que for usar o LCD, deve introduzir esse código como primeira ação.
void lcd_init();

//Função: Faz a configuração de inicialização do LCD.
//Ela já está introduzida dentro do lcd_init.
void SetUp ();

//Função: Envia as instruções para o LCD usando o high nibble (D4-d7).
void instruction (unsigned char x);

//Função: Responsável por enviar os dados para o LCD.
void lcd_data(unsigned char x);

//Função: Passa os caracteres de texto para a função info.
void text (unsigned char *b);

//Função: Recebe os caracteres para enviar para a função lcd_data com intuito de
imprimir no lcd.
void info(unsigned char x);

//Função: Habilitação dos pinos do LCD. Está dentro do lcd_init
void LCD_Pin_Enable(void);

//Função: Habilita o módulo PIT,canal 0, temporização de 1ms, com interrupção
ativada.
//Interrupção por pit: Adiciona ++ a uma variável contador para gerar a função
delay.
void pit_Init(void);

//Função: Limpa a tela do lcd, porém não limpa as variáveis de impressão.
```

```
void limpar(void);

//Função: Imprime na linha de cima do LCD
//O tamanho máximo é de 40bytes, caso ultrapasse 40 caracteres,
//a mensagem será escrita na linha de baixo.
//Parâmetro: Passar a mensagem entre " "
//Exemplo: imprimir_cima("Teste do LCD");
//Atenção: Cuidado com a sobrescrita de informação
//OBS:Existe a função de limpar os caractereses no vetor upper_line.
void imprimir_cima (unsigned char* upper_line[]);

//Função: Imprime na linha de baixo do LCD
//O tamanho máximo é de 40bytes por linha, caso ultrapasse 40 caracteres,
//a mensagem será escrita na linha de cima.
//Parâmetro: Passar a mensagem entre " "
//Exemplo: imprimir_baixo("Realizando teste do LCD");
//Atenção: Cuidado com a sobrescrita de informação
//OBS:Existe a função de limpar os caractereses no vetor lower_line.
void imprimir_baixo (unsigned char* lower_line[]);

//Função: Imprime na linha de cima e de baixo do LCD
//O tamanho máximo é de 40bytes por linha, caso ultrapasse 40 caracteres haverá
sobrescrita na outra linha.
//O primeiro parâmetro é para a linha de cima e o segundo para a linha de baixo.
//Parâmetro: Passar a mensagem entre " "
//Exemplo: imprimir("Teste do LCD","Realizando teste do LCD");
//OBS:Existe a função de limpar os caractereses no vetor lower_line e
upper_line.
void imprimir(unsigned char* upper_line[],unsigned char* lower_line[]);

//Função: Imprime valores no LCD na linha de escolha, sendo 0 para a linha de
baixo e outro valor para linha de cima.
//Parâmetro: _valor recebe uma variável do tipo inteiro e imprime na linha _x.
//Exemplo: int teste = 50;
//Imprimir na linha de cima: imprimir_valor (0,teste);
//Imprimir na linha de baixo: imprimir_valor (1,teste);
//OBS: o tamanho máximo de _valor é 8 dígitos.
void imprimir_valor(int _x, int _valor);

//Função: Limpa o vetor upper_line.
void limparcima();

//Função: Limpa o vetor lower_line.
void limparbaixo();

//Função: Desloca as linhas upper e lower para a esquerda uma vez com um tempo
1s.
//Atenção: Cada linha tem 40bytes, logo demorará 40 segundos para fazer a volta
completa.
//OBS: Ajuste o tempo do delay interno da função para um tempo menor de
deslocamento.
void deslocar_esquerda(void);

//Função: Desloca as linhas upper e lower para a direita uma vez com um tempo
1s.
//Atenção: Cada linha tem 40bytes, logo demorará 40 segundos para fazer a volta
completa.
//OBS: Ajuste o tempo do delay interno da função para um tempo menor de
deslocamento.
```

```
void deslocar_direita(void);



#endif



//-------------------------------------------------------------------------
// Includes
//-------------------------------------------------------------------------
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "lcd.h"
#include "fsl_pit.h"
#include "adc.h"
//-------------------------------------------------------------------------
// Macros
//-------------------------------------------------------------------------
// Definições que serão utilizados para desligar e ligar pinos dos LCD.
#define      BOARD_PIT_INSTANCE        0
#define      LCD_D7_ON           GPIO_WritePinOutput(GPIOE, 3, 1)
#define      LCD_D7_OFF          GPIO_WritePinOutput(GPIOE, 3, 0)
#define      LCD_D6_ON           GPIO_WritePinOutput(GPIOE, 2, 1)
#define      LCD_D6_OFF          GPIO_WritePinOutput(GPIOE, 2, 0)
#define      LCD_D5_ON           GPIO_WritePinOutput(GPIOB, 11, 1)
#define      LCD_D5_OFF          GPIO_WritePinOutput(GPIOB, 11, 0)
#define      LCD_D4_ON           GPIO_WritePinOutput(GPIOB, 10, 1)
#define      LCD_D4_OFF          GPIO_WritePinOutput(GPIOB, 10, 0)
#define      LCD_ENABLE_ON       GPIO_WritePinOutput(GPIOB, 9, 1)
#define      LCD_ENABLE_OFF      GPIO_WritePinOutput(GPIOB, 9, 0)
#define      LCD_RS_ON           GPIO_WritePinOutput(GPIOB, 8, 1)
#define      LCD_RS_OFF          GPIO_WritePinOutput(GPIOB, 8, 0)
//-------------------------------------------------------------------------
// Variáveis
//-------------------------------------------------------------------------
// Variáveis upper_line e lower_line com tamanho de 40 caracteres.
unsigned char  upper_line[39] = "FINALMENTE";
unsigned char  lower_line[39] = "OBRIGADO DEUS";

// Comandos de inicialização do LCD
const unsigned char  initLCD[8]={0X03, 0x02, 0x28, 0x0C,0x06,0x01,0x00};
// Variável de contagem
volatile long              count;
// Variável de sinalização de flag
volatile bool              pitIsrFlag[2] = {false};
extern volatile long count;
extern int long countadc;
// Variável que receberá a transformação de variáveis em dígitos.
int digito[7]={0};

//-------------------------------------------------------------------------
// Interrupções
//-------------------------------------------------------------------------
```

```
// Interrupção por PIT no canal 0.
void PIT_IRQHandler(void)
{
      if (PIT_GetStatusFlags(PIT, kPIT_Chnl_0)){
      //Limpa a Flag do PIT.
    PIT_ClearStatusFlags(PIT, kPIT_Chnl_0, kPIT_TimerFlag);
    pitIsrFlag[0] = true;
    count++;
      }

}

//-------------------------------------------------------------------------
// Funções
//-------------------------------------------------------------------------
// Inicialização do LCD
void lcd_init()
{
      // Habilita o clock do PORTA,PORTB e PORTC
      CLOCK_EnableClock(kCLOCK_PortA);
      CLOCK_EnableClock(kCLOCK_PortB);
      CLOCK_EnableClock(kCLOCK_PortE);

      // Configuração de ligar os pinos do LCD
      gpio_pin_config_t config_lcd_liga;
      config_lcd_liga.outputLogic=1;
      config_lcd_liga.pinDirection=kGPIO_DigitalOutput;

      // Configuração de desligar os pinos do LCD
      gpio_pin_config_t config_lcd_desliga;
      config_lcd_desliga.outputLogic=0;
      config_lcd_desliga.pinDirection=kGPIO_DigitalOutput;

      // Seleciona os MUX da UART (Comunicação pela SDA) e GPIO para os pinos
do LCD.
      PORT_SetPinMux(PORTA, 1, kPORT_MuxAlt2); /* PORTA1 está configurado como
UART0_RX */
      PORT_SetPinMux(PORTA, 2, kPORT_MuxAlt2); /* PORTA2 está configurado como
UART0_TX */
      PORT_SetPinMux(PORTB, 9, kPORT_MuxAsGpio); /* PORTB9 está configurado
como  GPIOB9 */
      PORT_SetPinMux(PORTB, 8, kPORT_MuxAsGpio); /* PORTB8 está configurado
como  GPIOB8 */
      PORT_SetPinMux(PORTB, 10, kPORT_MuxAsGpio); /* PORTB10 está configurado
como  GPIOB10 */
      PORT_SetPinMux(PORTB, 11, kPORT_MuxAsGpio); /* PORTB11 está configurado
como  GPIOB11 */
      PORT_SetPinMux(PORTE, 2, kPORT_MuxAsGpio); /* PORTE2 está configurado
como  GPIOE2 */
      PORT_SetPinMux(PORTE, 3, kPORT_MuxAsGpio); /* PORTE3 está configurado
como  GPIOE3 */
      PORT_SetPinMux(PORTE, 4, kPORT_MuxAsGpio); /* PORTE4 está configurado
como  GPIOE4 */

      // Inicializa os Pinos GPIO com o padrão inicial de setup.
      GPIO_PinInit(GPIOB, 9, &config_lcd_liga); //ENABLE
      GPIO_PinInit(GPIOB, 8, &config_lcd_desliga); //RS
      GPIO_PinInit(GPIOB, 10, &config_lcd_liga);//D4
      GPIO_PinInit(GPIOB, 11, &config_lcd_liga);//D5
```

```
      GPIO_PinInit(GPIOE, 2, &config_lcd_liga);//D6
      GPIO_PinInit(GPIOE, 3, &config_lcd_liga);//D7
      GPIO_PinInit(GPIOE, 4, &config_lcd_liga);//D7

      //Chama a função de habilitar os pinos do LCD
    LCD_Pin_Enable();
    //Chama a função de inicializar o PIT
    pit_Init();
    //Necessita de 2ms para chamar a função Setup
      delay(2);
      //Chama a função de configuração
      SetUp();

}

//-------------------------------------------------------------------------
// Função auxiliar de habilitar
void enable(void){
      LCD_ENABLE_ON;
      delay(1);
      LCD_ENABLE_OFF;
}

//-------------------------------------------------------------------------
// Função que realiza a configuração inicial do LCD
// Envia as instruções necessárias para o uso do LCD
void SetUp(){
      unsigned char a = 0;
      while(initLCD[a])
            {
                  instruction(initLCD[a]);
                  delay(1);
                  a++;
            }
}

//-------------------------------------------------------------------------
//Função de instruções
void instruction(unsigned char x){
            LCD_RS_OFF;
            lcd_data(x&0xF0);
            enable();
            lcd_data((x<<4)&0xF0);
            enable();
}

//-------------------------------------------------------------------------
// Função de dados para o LCD
void lcd_data(unsigned char x)
{
      //Bit 7
      if (x&0x80)
      {
        LCD_D7_ON;
      }
      else
      {
        LCD_D7_OFF;
      }
```

```
	//Bit 6
	if (x&0x40)
	{
	  LCD_D6_ON;
	}
	else
	{
	  LCD_D6_OFF;
	}

	//Bit 5
	if (x&0x20)
	{
	  LCD_D5_ON;
	}
	else
	{
	  LCD_D5_OFF;
	}

	//Bit 4
	if (x&0x10)
	{
	  LCD_D4_ON;
	}
	else
	{
	  LCD_D4_OFF;
	}
}

//---------------------------------------------------------------------
// Função para texto
void text (unsigned char *b){
	while(*b)
	{
		info(*b);
		b++;
	}
}

//---------------------------------------------------------------------
// Função complementar de texto
void info(unsigned char x)
{
	if (x == '') x = 238;
	LCD_RS_ON;
	lcd_data(x&0xF0);
	enable();
	lcd_data((x<<4)&0xF0);
	enable();
}


//---------------------------------------------------------------------
// Função para inicializar os pinos com os valores corretos.
void LCD_Pin_Enable(void){
	LCD_ENABLE_ON;
```

```
	LCD_RS_OFF;
	LCD_D7_ON;
	LCD_D6_ON;
	LCD_D5_ON;
	LCD_D4_ON;
}

//---------------------------------------------------------------------
// Função para gerar um atraso.
// Parâmetro: Recebe o valor inteiro em milisegundos.
void delay(unsigned long time){
	PRINTF("delay\n");
	while (count <= time){}
	PRINTF("count>time\n");
	count = 0;
}

// Função limpar tela
void limpar (void){
	// Instrução para limpar a tela. Necessita de 2 ms.
	instruction(0x01);
	delay(2);
	// Instrução para deixar na linha 0, coluna 0.
	instruction(0x00);
	}

// Imprime texto na linha superior do lcd.
void imprimir_cima (unsigned char* upper_line[])
{
	// Limpa os valores no vetor upper.
	limparcima();
	// Seleciona a primeira linha
	instruction(0x80);
	// Escreve na linha de cima.
	text((unsigned char *)&upper_line[0]);

}
// Imprime texto na linha inferior
void imprimir_baixo (unsigned char* lower_line[])
{
	// Limpa os valores no vetor lower.
	limparbaixo();
	// Seleciona a segunda linha
	instruction(0xC0);
	// Escreve na linha debaixo.
	text((unsigned char *)&lower_line[0]);

}

// Imprime na linha de cima e debaixo do LCD.
void imprimir (unsigned char* upper_line[],unsigned char* lower_line[])
{
	// Limpa os vetores upper e lower.
	limparcima();
	limparbaixo();
	// Seleciona a primeira linha e escreve.
	instruction(0x80);
	text((unsigned char *)&upper_line[0]);
	// Seleciona a segunda linha e escreve.
```

```
	instruction(0xC0);
	text((unsigned char *)&lower_line[0]);

}

// Imprime valores na linha selecionada.
// Converte a variável em dígitos de um número.
void imprimir_valor(int _x, int _valor)
{
	int _aux =_valor;
	if (_x==0)
		{
		limparbaixo();
		instruction(0xC0); // Linha debaixo
		digito[0]=_aux/10000000;
		digito[1]=(_aux-(digito[0]*10000000))/1000000;
		digito[2]=(_aux-((digito[0]*10000000)+(digito[1]*1000000)))/100000;
		digito[3]=(_aux-
((digito[0]*10000000)+(digito[1]*1000000)+(digito[2]*100000)))/10000;
		digito[4]=(_aux-
((digito[0]*10000000)+(digito[1]*1000000)+(digito[2]*100000)+(digito[3]*10000)))
/1000;
		digito[5]=(_aux-
((digito[0]*10000000)+(digito[1]*1000000)+(digito[2]*100000)+(digito[3]*10000)+(
digito[4]*1000)))/100;
		digito[6]=(_aux-
((digito[0]*10000000)+(digito[1]*1000000)+(digito[2]*100000)+(digito[3]*10000)+(
digito[4]*1000)+(digito[5]*100)))/10;
		digito[7]=(_aux-
((digito[0]*10000000)+(digito[1]*1000000)+(digito[2]*100000)+(digito[3]*10000)+(
digito[4]*1000)+(digito[5]*100)+(digito[6]*10)));
			for (int _i=0; _i<8;_i++)
			{
				lower_line[_i] = (char) digito[_i]+0x30;
			}
		text((unsigned char *)&lower_line[0]);
		}
		else
		{
		limparcima();
		instruction(0x80); // Linha de cima
		digito[0]=_aux/10000000;
		digito[1]=(_aux-(digito[0]*10000000))/1000000;
		digito[2]=(_aux-((digito[0]*10000000)+(digito[1]*1000000)))/100000;
		digito[3]=(_aux-
((digito[0]*10000000)+(digito[1]*1000000)+(digito[2]*100000)))/10000;
		digito[4]=(_aux-
((digito[0]*10000000)+(digito[1]*1000000)+(digito[2]*100000)+(digito[3]*10000)))
/1000;
		digito[5]=(_aux-
((digito[0]*10000000)+(digito[1]*1000000)+(digito[2]*100000)+(digito[3]*10000)+(
digito[4]*1000)))/100;
		digito[6]=(_aux-
((digito[0]*10000000)+(digito[1]*1000000)+(digito[2]*100000)+(digito[3]*10000)+(
digito[4]*1000)+(digito[5]*100)))/10;
		digito[7]=(_aux-
((digito[0]*10000000)+(digito[1]*1000000)+(digito[2]*100000)+(digito[3]*10000)+(
digito[4]*1000)+(digito[5]*100)+(digito[6]*10)));
			for (int _i=0; _i<8;_i++)
```

```
                    {
                          upper_line[_i] = (char) digito[_i]+0x30;
                    }
                    text((unsigned char *)&upper_line[0]);
      }}
// Função que limpa os valores do vetor upper.
void limparcima()
{
      for (int _i=0; _i<40;_i++)
      {
                          upper_line[_i] = ' ';
                          }

}
// Função que limpa os valores do vetor lower.
void limparbaixo()
{
      for (int _i=0; _i<40;_i++)
      {
                          lower_line[_i] = ' ';
                          }

}

// Desloca uma vez para a esquerda
void deslocar_esquerda(void)
{
      instruction(0x18);
      delay(1000);
}
// Desloca uma vez para a direita
void deslocar_direita(void)
{
      instruction(0x1E);
      delay(1000);
}

//-------------------------------------------------------------------------


void pit_Init(void){
      // Configuração do PIT canal 0
      pit_config_t chn0Confg;
      chn0Confg.enableRunInDebug=false;

      // Inicializa o pit com as configuração para o canal 0.
      PIT_Init(PIT, &chn0Confg);

      // Configura o timer do PIT canal 0
      // Definido para 1ms.
      // Clock do PIT = 24Mhz.
      PIT_SetTimerPeriod(PIT, kPIT_Chnl_0, 240000);

      // Iniciliza o timer do PIT canal 0
      PIT_StartTimer(PIT, kPIT_Chnl_0);

      // Habilita a interrupção para o PIT canal 0
      PIT_EnableInterrupts(PIT, kPIT_Chnl_0, kPIT_TimerInterruptEnable);
```

```
      // Habilita a interrupção PIT NVIC
      NVIC_EnableIRQ(PIT_IRQn);
}



//-------------------------------------------------------------------------

#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "lcd.h"
#include "fsl_pit.h"
#include "fsl_port.h"
#include "fsl_common.h"
#include "adc.h"
#include "fsl_gpio.h"


int main(void) {
    /* Init board hardware. */
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();

    BOARD_InitDebugConsole();
    lcd_init();
    limpar();

    imprimir_cima("VALOR ABAIXO:");
    imprimir_valor(0, 12345678);

      PRINTF("TERMINO");
    while(1) {

deslocar_esquerda();

        __asm volatile ("nop");
    }
    return 0 ;
}
```

APÊNDICE C – CÓDIGO ADC

```
/*
 * adc16.h
 *
 *  Created on: 20 de ago de 2022
 *      Author: Alysson
 */

#ifndef ADC16_H_
#define ADC16_H_


#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "fsl_pit.h"
#include "fsl_port.h"
#include "fsl_common.h"
#include "fsl_adc16.h"
#include "fsl_gpio.h"
#include "int.h"

void init_adc();

uint32_t get_conversao(uint32_t chNumero);

void init_amostragem(int Hz);

void parar_amostragem();

#endif /* ADC16_H_ */
//-----------------------------------------------------------------------------
/*
 * adc16.c
 *
 *  Created on: 20 de ago de 2022
 *      Author: Alysson
 */

#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "fsl_pit.h"
#include "fsl_port.h"
#include "fsl_common.h"
#include "fsl_adc16.h"
#include "fsl_gpio.h"


bool adcrun=false;
```

```
adc16_channel_config_t chconfigadc;
float frequencia=0;
int countadc=0;


void init_adc(){
	CLOCK_EnableClock(kCLOCK_PortE);
	CLOCK_EnableClock(kCLOCK_PortB);
	CLOCK_EnableClock(kCLOCK_Adc0);
	adc16_config_t configadc;
	configadc.clockDivider=kADC16_ClockDivider1;
	configadc.clockSource=kADC16_ClockSourceAlt0;
	configadc.enableContinuousConversion=true;
	configadc.enableHighSpeed=false;
	configadc.enableLowPower=false;
	configadc.longSampleMode=kADC16_LongSampleDisabled;
	configadc.referenceVoltageSource=kADC16_ReferenceVoltageSourceVref;
	configadc.resolution=kADC16_ResolutionSE12Bit;
	ADC16_Init(ADC0, &configadc);
	ADC16_EnableHardwareTrigger(ADC0, false);
	ADC16_DoAutoCalibration(ADC0);
}

void init_amostragem(int Hz){
	// Configuração do PIT canal 1
	pit_config_t chn1Confg;
	chn1Confg.enableRunInDebug=false;
	// Inicializa o pit com as configuração para o canal 1.
	PIT_Init(PIT, &chn1Confg);
	uint32_t frequencia=(24000000/Hz);
	// Configura o timer do PIT canal 1
	// Frequencia máxima de clock 24Mhz
	// Limite máximo permitido pro adc 2.4MHz.
	// Motivo= Sample and Hold.
	PIT_SetTimerPeriod(PIT, kPIT_Chnl_1, frequencia);
	// Inicializa o timer do PIT canal 0
	PIT_StartTimer(PIT, kPIT_Chnl_1);
	// Habilita a interrupção para o PIT canal 0
	PIT_EnableInterrupts(PIT, kPIT_Chnl_1, kPIT_TimerInterruptEnable);
	// Habilita a interrupção PIT NVIC
	NVIC_EnableIRQ(PIT_IRQn);

}

uint32_t get_conversao(uint32_t chNumero){
	chconfigadc.enableDifferentialConversion=false;
	chconfigadc.enableInterruptOnConversionCompleted=false;
	chconfigadc.channelNumber=chNumero;
	ADC16_SetChannelConfig(ADC0, kADC16_ChannelMuxA, &chconfigadc);
	return ADC16_GetChannelConversionValue(ADC0, kADC16_ChannelMuxA);
}

void parar_amostragem(){
	PIT_StopTimer(PIT, kPIT_Chnl_1);
	adcrun=true;
}
```

```
//-------------------------------------------------------------------------------
/**
 * @file    Modulo_FRDM_KL25z.c
 * @brief   Application entry point.
 */
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "int.h"
#include "adc16.h"
#include "fsl_pit.h"
#include "fsl_port.h"
#include "fsl_common.h"
#include "fsl_adc16.h"
#include "fsl_gpio.h"
#include "gpio.h"

int main(void) {


    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();
    BOARD_InitDebugConsole();
    int valor=0;
    init_adc();
    DAC_Ini();
   // init_amostragem(1);
    while(1) {

       valor=get_conversao(15);
       DAC_Valor(valor);

       PRINTF("%d \r\n ",valor);
  //   delay2();
    }
    return 0 ;
}
```

APÊNDICE D – CÓDIGO DAC

```
/*
 * dac.h
 *
 *  Created on: 21 de ago de 2022
 *      Author: Alysson
 */

#ifndef DAC_H_
#define DAC_H_

#include "fsl_dac.h"

// inicia e Habilita o Conversor digital analógico
void DAC_Ini(void);

// Envia um valor inteiro para o DAC
// O valor será convertido em tensão
void DAC_Valor(int dacvalor);

// Envia um valor de tensão direto para o DAC;
void DAC_Tensao(float volts);

//-------------------------------------------------------------------------------


#endif /* DAC_H_ */

/*
 * dac.c
 *
 *  Created on: 21 de ago de 2022
 *      Author: Alysson
 */

#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "fsl_pit.h"
#include "fsl_port.h"
#include "fsl_common.h"
#include "fsl_adc16.h"
#include "fsl_gpio.h"
#include "fsl_dac.h"
#include "dac.h"

dac_config_t dac_cfg;


void DAC_Ini(void)
{
	dac_cfg.enableLowPowerMode=false;
	dac_cfg.referenceVoltageSource=kDAC_ReferenceVoltageSourceVref2;
```

```
	DAC_Init(DAC0, &dac_cfg);
	DAC_Enable(DAC0, true);
	DAC_SetBufferReadPointer(DAC0, 0);
}

void DAC_Valor(int dacvalor)
{
	DAC_SetBufferValue(DAC0, 0, dacvalor);
}

void DAC_Tensao(float volts)
{
	uint16_t dacvalor = (volts*4096.0f)/3.31f;
	DAC_SetBufferValue(DAC0, 0, dacvalor);
}
```

APÊNDICE E – CÓDIGO INTERRUPÇÃO

```
/*
 * int.h
 *
 *  Created on: 20 de ago de 2022
 *      Author: Alysson
 */

#ifndef INT_H_
#define INT_H_

#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "fsl_pit.h"
#include "fsl_port.h"
#include "fsl_common.h"
#include "fsl_adc16.h"
#include "fsl_gpio.h"

#define PIT_INT PIT_IRQHandler

void PIT_INT(void);

void delay1(unsigned long time);

void delay2(void);


#endif /* INT_H_ */
//-----------------------------------------------------------------------------

/*
 * int.c
 *
 *  Created on: 20 de ago de 2022
 *      Author: Alysson
 */

#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "fsl_pit.h"
#include "fsl_port.h"
#include "fsl_common.h"
#include "fsl_adc16.h"
#include "fsl_gpio.h"
#include "int.h"
```

```
int count1=0;
int count2=0;


void PIT_INT(void)
{
	if (PIT_GetStatusFlags(PIT, kPIT_Chnl_0))
	{
		PIT_ClearStatusFlags(PIT, kPIT_Chnl_0, kPIT_TimerFlag);
		count1++;
	}

	if (PIT_GetStatusFlags(PIT, kPIT_Chnl_1))
	{
		PIT_ClearStatusFlags(PIT, kPIT_Chnl_1, kPIT_TimerFlag);
		count2++;
	}

}

void delay1(unsigned long time){
while (count1 <= time){}
count1 = 0;
}

void delay2(void){
while (count2 < 1){}
count2 = 0;

}
```

APÊNDICE F – CÓDIGO CONDICIONADOR DE SINAIS

```
/*
 * DAC_ADC_PWM.c
 *
 *  Created on: 18 de out de 2022
 *      Author: Alysson
 */
//-------------BIBLIOTECAS --------------------
 #include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "fsl_adc16.h"
#include "fsl_dac.h"
#include "fsl_tpm.h"
#include "fsl_common.h"
#include "fsl_port.h"
//-------------DEFINIÇÕES --------------------
#define BASEADC ADC0
#define TIPOADC 0U
#define CANALADC 8U /* PTB0, ADC0_SE8 */
#define CANALADC2 9U /* PTB1, ADC0_SE9 */
#define BASE_DAC DAC0
#define IRQ_ADC ADC0_IRQn
#define IRQH_ADC ADC0_IRQHandler
#define VREF_BRD 3.300
#define SE_12BIT 4096.0
#define SE_16BIT 65536.0
#define BASETPM TPM2 /* PTE23, FTM2_ch1*/
#define CANALTPM 1U
#define TPM_SOURCE_CLOCK CLOCK_GetFreq(kCLOCK_PllFllSelClk)
//-------------PROTÓTIPOS --------------------
static void DAC_ADC_Init(void);
void delay(void);
//-------------VARIÁVEIS --------------------

/* Variável status flag ADC*/
volatile bool g_Adc16ConversionDoneFlag = false;
/* Resetando o valor da conversão AD*/
volatile uint32_t g_Adc16ConversionValue = 0;
/* Declaração da configuração do canal ADC*/
adc16_channel_config_t g_adc16ChannelConfigStruct;
volatile uint8_t updatedDutycycle = 0;
volatile float TensaoPWM = 0;

//-------------DAC ADC INIT --------------------
static void DAC_ADC_Init(void)
{
	CLOCK_EnableClock(kCLOCK_PortA);                           /* Port A
Clock Gate Control: Clock enabled */
	CLOCK_EnableClock(kCLOCK_PortE);
	CLOCK_EnableClock(kCLOCK_PortB);
	CLOCK_EnableClock(kCLOCK_PortC);
	CLOCK_EnableClock(kCLOCK_PortD);
```

```
      PORT_SetPinMux(PORTB, 0, kPORT_PinDisabledOrAnalog); // ADC
      PORT_SetPinMux(PORTB, 1, kPORT_PinDisabledOrAnalog); // ADC
      PORT_SetPinMux(PORTE, 1, kPORT_MuxAlt2);            /* PORTA1 (pin 27) is
configured as UART0_RX */
        PORT_SetPinMux(PORTA, 2, kPORT_MuxAlt2);           /* PORTA2 (pin 28)
is configured as UART0_TX */
        PORT_SetPinMux(PORTE, 23, kPORT_MuxAlt3); // PWM
        PORT_SetPinMux(PORTE, 30, kPORT_PinDisabledOrAnalog); //DAC
/* Declaração da configuração do ADC*/
adc16_config_t adc16ConfigStruct;
/* Declaração da configuração do DAC*/
dac_config_t dacConfigStruct;
DAC_GetDefaultConfig(&dacConfigStruct);
/* Inicialização do DAC*/
DAC_Init(BASE_DAC, &dacConfigStruct);
/* Habilita a saída do DAC*/
DAC_Enable(BASE_DAC, true); /* PINO PTE30*/
/* Configuração do ADC. */
adc16ConfigStruct.clockDivider = kADC16_ClockDivider1;
adc16ConfigStruct.resolution = kADC16_ResolutionSE12Bit;
adc16ConfigStruct.clockSource= kADC16_ClockSourceAlt0;
adc16ConfigStruct.enableHighSpeed = true;
adc16ConfigStruct.longSampleMode = kADC16_LongSampleDisabled;
adc16ConfigStruct.enableContinuousConversion = false;
adc16ConfigStruct.enableLowPower = false;
adc16ConfigStruct.enableAsynchronousClock=false;
adc16ConfigStruct.referenceVoltageSource=kADC16_ReferenceVoltageSourceVref;
/* Inicia o ADC*/
ADC16_Init(BASEADC, &adc16ConfigStruct);
/* Desabilita o trigger do hardware do ADC*/
ADC16_EnableHardwareTrigger(BASEADC, false);
ADC16_DoAutoCalibration(BASEADC);
/* Configura o canal do adc, SINAL de entrada no PTE20 e habilita interrupção
com o termino da conversão*/
g_adc16ChannelConfigStruct.channelNumber = CANALADC;
g_adc16ChannelConfigStruct.enableInterruptOnConversionCompleted = false;
g_adc16ChannelConfigStruct.enableDifferentialConversion = false;

}
void IRQH_ADC(void)
{
    g_Adc16ConversionDoneFlag = true;
    g_Adc16ConversionValue = ADC16_GetChannelConversionValue(BASEADC, TIPOADC);
}
int main(void) {
tpm_config_t tpmInfo;
tpm_chnl_pwm_signal_param_t tpmParam;
tpm_pwm_level_select_t pwmLevel = kTPM_HighTrue;
// Configura a frequencia do PWM para 24 kHz
tpmParam.chnlNumber = (tpm_chnl_t)CANALTPM;
tpmParam.level = pwmLevel;
tpmParam.dutyCyclePercent = updatedDutycycle;
float voltRead;
int valordac=0;
/* Init board hardware. */
BOARD_InitBootPins();
BOARD_InitBootClocks();
BOARD_InitBootPeripherals();
BOARD_InitDebugConsole();
```

```
CLOCK_SetTpmClock(1U);
TPM_GetDefaultConfig(&tpmInfo);
TPM_Init(BASETPM, &tpmInfo);
TPM_SetupPwm(BASETPM, &tpmParam, 1U, kTPM_CenterAlignedPwm, 24000U,
TPM_SOURCE_CLOCK);
TPM_StartTimer(BASETPM, kTPM_SystemClock);
/* Habilita a interrupção do ADC*/
EnableIRQ(IRQ_ADC);
/* Chama a função DAC_ADC*/
DAC_ADC_Init();

/* Laço infinito*/
      while (1)
        {
             /* Zera a flag de conversão*/
        g_Adc16ConversionDoneFlag = false;
        g_adc16ChannelConfigStruct.channelNumber = CANALADC;
        g_adc16ChannelConfigStruct.enableInterruptOnConversionCompleted = false;
        /* Seta as configurações do canal*/
        ADC16_SetChannelConfig(BASEADC, TIPOADC, &g_adc16ChannelConfigStruct);
/* Aguarda a flag de conversão do ADC*/
        /* Printa o valor da conversão do ADC
        /PRINTF("\r\n Valor ADC PORTE20: %d", g_Adc16ConversionValue);*/

        // o valor da saída do PTE30 será o valor da conversão do ADC
        valordac= (g_Adc16ConversionValue/SE_12BIT)*SE_12BIT;
        DAC_SetBufferValue(BASE_DAC, 0U, valordac);
        /* Converte o valor em tensão com base no 3.3V da placa*/
        voltRead = (float)(g_Adc16ConversionValue * (VREF_BRD / SE_12BIT));
        /* Printa o Valor de tensão no pino PORTE30: */

        g_Adc16ConversionDoneFlag = false;
        g_adc16ChannelConfigStruct.channelNumber = CANALADC2;
        g_adc16ChannelConfigStruct.enableInterruptOnConversionCompleted = true;

        ADC16_SetChannelConfig(BASEADC, TIPOADC, &g_adc16ChannelConfigStruct);
        while (!g_Adc16ConversionDoneFlag)
             {
             }
        updatedDutycycle = (g_Adc16ConversionValue / SE_12BIT);
        TensaoPWM= (updatedDutycycle * 3.3);

        // Desabilita o canal antes de atualizar o PWM
        TPM_UpdateChnlEdgeLevelSelect(BASETPM, (tpm_chnl_t)CANALTPM, 0U);

        // Atualiza o ciclo PWM
        TPM_UpdatePwmDutycycle(BASETPM, (tpm_chnl_t)CANALTPM,
kTPM_CenterAlignedPwm, updatedDutycycle);

        // Inicia o canal com o valor atualizado do PWM
        TPM_UpdateChnlEdgeLevelSelect(BASETPM, (tpm_chnl_t)CANALTPM, pwmLevel);
             }
return 0;
}
```