Protocolo I2C

Manoel Morais Lemos Neto*, Guilherme Araujo Machado do Nascimento† *Universidade Federal de Santa Catarina (UFSC) †Universidade Federal de Santa Catarina (UFSC)

***Resumo*—A fim de aprimorar os conceitos abordados sobre sistemas de comunicação, este artigo abordará a implementação do protocolo I2C, utilizando como hardware as placas Arduino e Raspberry pi 3. Além disso, também serão implementados os métodos de detecção de erros Checsum e CRC, juntamente com uma interface com a análise estatísticas dos mesmos.**

***Palavras-chave*—Sistemas de Comunicação, Protocolo, I2C, Arduino, Raspberry Pi, Checksum, CRC, Mecatrônica, Sistemas Embarcados.**

I. Introdução

O I2C ou I²C(Inter-Integrated Circuit), foi criado em 1982 como um simples sistema de barramento interno para a construção de controles eletrônicos em vários chips Philips. Hoje, este protocolo se encontra amplamente difundido pelo mundo, implementado em mais de 1000 CIs fabricados por mais de 50 empresas distintas. Devido sua versatilidade ele é usado em várias arquiteturas de controle tais como System Management Bus(SmBus), Power Bus(PMBus), Intelligent Platform Management Interface(IPMI), Advanced Telecom Computing Architecture(ATCA) entre outras [1].

Utilizado principalmente para conectar periféricos de baixa velocidade a placas-mães, microcontroladores e afins [2]. Para isso é necessário que haja suporte para esse protocolo tanto pela unidade controladora quanto pelos periféricos, seja via software através da técnica bit-bang, a qual emula bit a bit o protocolo, ou via hardware no próprio SoC ou utilizando CI's externos como o SC16IS750 [1].

Neste trabalho será apresentada uma maneira de conectar um *Raspberry pi* a um *Arduino* através deste protocolo, além de uma análise gráfica de dois métodos de detecção de erros: **Checksum** e **CRC**(Cyclic Redundant Check). Com o auxílio deles será possível ver a implicância do ruído na sequência de bits da mensagem transferida.

II. Fundamentação Teórica

A. I²C

O I2C é um dos protocolos de comunicação serial síncronos mais usado no mundo[3]. É muito adequado para comunicações entre periféricos integrados para transferência de dados de baixa/média velocidade. Este módulo é amplamente utilizado em vários controladores, sensores e alguns outros circuitos integrados [4].

Ele é um barramento serial multi-mestre, ou seja, é capaz de possuir múltiplos mestres e é usado principalmente para conectar periféricos de baixa velocidade utilizando apenas duas linhas de transmissão, uma **SDA**(serial data line) responsável pela transmissão de dados bidirecionalmente e uma **SCL**(serial clock line) com o clock gerado pelo mestre. além disso é importante ressaltar que o mesmo trabalha com tensões entre +5V e +3,3V.

Cada dispositivo conectado ao barramento é um software endereçável por um endereço único e simples representando a relação mestre / escravo que existe o tempo todo; Mestres podem operar como transmissores-mestres ou como mestres-receptores[5]. Ele também é capaz de detectar colisões, algo comum para a topologia barramento com mais de um mestre, prevenindo que o sistema pare abruptamente.

O mestre é o dispositivo que inicia a transferência de dados no barramento e gera o clock que permite a transferência dos dados. Nesse instante, qualquer dispositivo endereçado é considerado um escravo. As condições de START e STOP são sempre geradas pelo mestre, que funcionam da seguinte forma: Após o START o barramento é considerado ocupado, e algum tempo após STOP ter sido identificado o barramento é considerado disponível.

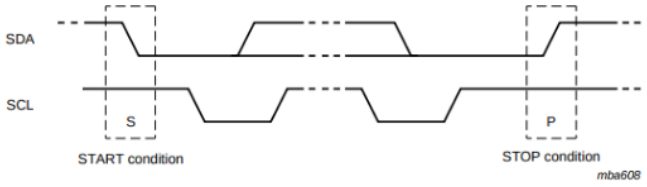

Figura 1. Condições de START e STOP. Retirado de [1]

Todo byte colocado na linha de transmissão SDA, deve ter 8 bits de comprimento. No entanto , não há restrição para quantos bytes que ela pode transmitir e cada byte deve ser acompanhado por um bit de reconhecimento. Os dados são transmitidos com o Most Significant bit(MSB) primeiro. Se um escravo não puder receber ou transmitir um pacote de dados completo pois estava realizando outra função, por exemplo, atendendo a solicitação de uma interrupção interna, ele poderá manter o SCL com sinal baixo forçando um estado de espera no mestre. A transferência então continua quando o escravo está pronto para outro conjunto de dados e libera o SCL [1].

B. Detecção de Erros

Sistemas de comunicação devem ser capazes de transmitir dados de maneira precisa, e para algumas situações é necessário que os dados enviados e recebidos sejam idênticos. No entanto, nem sempre isso ocorre, este fato se dá devido a uma série de fatores que podem alterar um ou mais bits de uma

Universidade Federal de Santa Catarina (UFSC) (UFSC). Correspondência aos autores: Manoel Morais Lemos Neto (email: manoel.morais@grad.ufsc.br), Guilherme Araujo Machado do Nascimento (email:guilhermeamn1@gmail.com)

mensagem. Portanto, para esse tipo de situação é crucial a aplicação de mecanismos de detecção e correção de erros.

A informação que chega devidamente no receptor é denominada transinformação, enquanto a parte perdida durante a transmissão é chamada de equivocação. Já a parte da informação que não tem ligação alguma com a fonte é chamada de dispersão[6]. São termos importantes que auxiliam no entendimento da composição da mensagem, e como a dispersão não possui relação com a mensagem original ela pode ser facilmente emulada.

O conceito mais importante na detecção e correção de erros é a redundância. Para sermos capazes de detectar ou corrigir erros, precisamos enviar alguns bits extras redundantes junto com os dados. Esses bits redundantes são acrescentados pelo emissor e posteriormente retirados pelo receptor. Sua presença possibilita que o receptor detecte ou corrija bits corrompidos[7].

Em se tratando de código de blocos, as mensagens são divididas em blocos de tamanho igual a k, e cada um deles recebe então o nome de palavras de dado(datawords), que no geral é a mensagem gerada na fonte. Adicionando a estes blocos r bits de redundância teremos então novos com o valor n=k+r, que serão chamados de palavras de código(codeword). Que será transmitida pelo canal de comunicação. Para abordar a detecção de erros dessas palavras serão abordadas duas técnicas, sendo elas: checksum e Cyclic Redundant Check(CRC). Abaixo temos uma representação gráfica do texto supracitado.

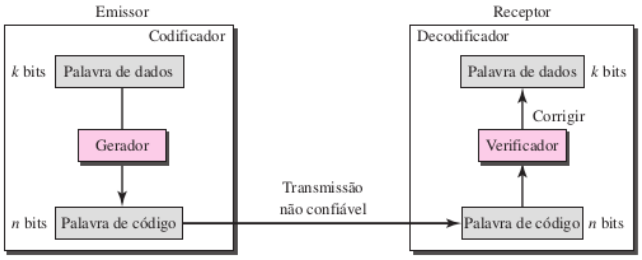

Figura 2. Estrutura do codificador e decodificador na correção de erros. Retirado de[7]

C. CRC — Cyclic Redundant Check

Baseado na teoria de códigos de correção de erros cíclicos, o CRC altera o tamanho da mensagem acrescentado bits de redundância para que posteriormente no decodificador seja possível identificar e corrigir os erros, ele se mostra bastante efetivo na detecção de erros em rajadas. Graças a essa característica ele é bastante utilizado em redes LANs e WANs[7].

Antes de explicar o funcionamento do algoritmo é importante explicitar alguns termos e suas respectivas representações:

- Palavra de dados: *d(x)*
- Palavra de código: *c(x)*
- Síndrome: *s(x)*
- erro: *e(x)*
- Gerador: *g(x)*

Em seu livro[7] Forouzan descreve o algoritmo do CRC da seguinte forma: No codificador, a palavra de dados tem k bits e a palavra de código tem n bits, então o tamanho da palavra de dados é aumentado adicionando-se n – k 0s ao lado direito da palavra. O resultado de n bits alimenta o gerador. O gerador usa um divisor de tamanho n – k + 1, predefinido e estabelecido por ambas as partes. O gerador divide a palavra de dados aumentada pelo divisor (divisão de módulo 2). O quociente da divisão é descartado; o resto (r2r1r0) é anexado à palavra de dados para criar a palavra de código.

O decodificador recebe a palavra de código possivelmente corrompida. Uma cópia de todos os n bits é alimentada no verificador, que é uma réplica do gerador. O resto produzido pelo verificador é uma síndrome de n – k bits que alimenta o analisador lógico de decisão. O analisador tem uma função simples. Se os bits de síndrome forem todos 0s, os (n – k + 1) bits mais à esquerda da palavra de código são aceitos como palavras de dados (interpretado como não sendo um erro); caso contrário, os (n – k + 1) são descartados (erro).

A operação de divisão é realizada utilizando a operação XOR(OU exclusivo) representada simbolicamente por $\oplus$. O mesmo processo descrito pode ser melhorado com a utilização de polinômios, pois dessa forma é possível criar uma simplificação bastante significativa da palavra de código. Este processo requer a conversão das palavras escritas na forma de 0s e 1s para polinômios. Considera-se todos os bits da palavra de dados como coeficientes de um polinômio, o que permite eliminar todos os termos atrelados aos bits de valor 0. Quanto aos de valor 1, estes representarão um valor do tipo x^i, sendo i o grau do polinômio, cujos valores são decrescentes da esquerda para a direita da palavra de dados, e o maior grau possível é dado pelo número de bits contidos na mesma menos 1. Abaixo temos um exemplo retirado de [7].

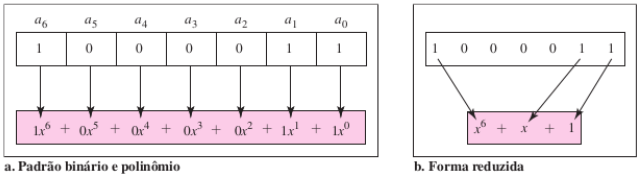

Figura 3. Representação polinômial de uma palavra binária. Retirado de [7]

D. Checksum

Assim como o CRC, o checksum também se baseia no conceito de redundância. Seu conceito é simples e consiste no envio de pacotes de palavras de mesmo tamanho, juntamente com o valor de sua soma de todas as palavras. No receptor será recalculada essa soma e então será efetuada uma comparação com o valor da soma recebida, caso sejam iguais não há presença de erro, em caso de discrepância entre os valores presume-se que ocorreu um erro. O método pode também ser usado para verificação na transmissão de imagens, como uma forma de pré-classificação das imagens recebidas pelo receptor da comunicação em aplicações de reconhecimento de imagem e visão computacional[8].

Segundo [7] podemos facilitar o trabalho do receptor da seguinte maneira: se enviarmos o negativo (complemento) da soma, denominada checksum. O receptor pode somar todos os números recebidos (inclusive a soma de verificação). Se o resultado for 0, ele supõe que não existem erros; caso contrário, um erro foi detectado. Além disso, é necessário o uso da aritmética de complemento de 1. Nessa aritmética, podemos representar números sem sinal entre 0 e $2^n - 1$, usando apenas n bits. Se o número tiver mais de n bits, os bits extras mais à esquerda precisam ser adicionados aos n bits mais à direita (wrapping). Na aritmética de complemento um, um número negativo pode ser representado invertendo-se todos os bits (transformando 0 em 1 e 1 em 0). Isso é o mesmo que subtrair o número por $2^n - 1$. Abaixo temos um exemplo do funcionamento deste algoritmo.

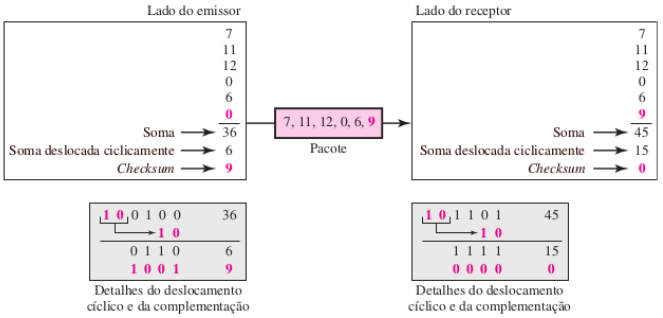

Figura 4. Exemplo prático do algoritmo Checksum. Retirado de [7]

Apesar de sua simplicidade, o checksum é considerado um dos métodos menos caros de detecção de erros[9], por isso ainda é bastante utilizado na internet. No entanto, aos poucos ele vem sendo substituído pelo CRC. Pois seu modelo tradicional conta com um número pequeno de bits(16)[7]. Ademais ele também é incapaz de detectar, por exemplo, quando uma palavra é incrementada e outra é decrementada pelo mesmo valor, os dois erros não serão detectados, visto que a soma e o checksum permanecerão idênticos. Da mesma forma que, se os valores de várias palavras forem incrementados, mas se a mudança total for múltipla de 65535, a soma e o checksum serão alterados, o que significa que erros não serão detectados[7].

III. Montagem do Experimento

A. Ferramentas necessárias

Hardware:

- Raspberry Pi 3B
- Arduíno UNO
- 3 jumpers macho-fêmea
- Fonte de alimentação 5V/2.5A

Software:

- Raspberry Pi OS
- PyQt5
- Arduino IDE

Montagem:

Com os itens acima, a conexão entre os hardwares foi realizada conforme ilustra as figuras a seguir:

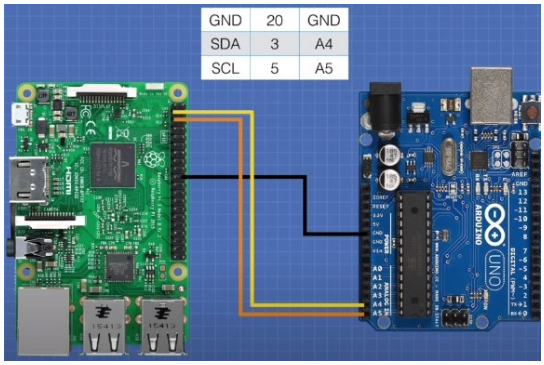

Figura 5. Conexão dos pinos. Adaptado de [10]

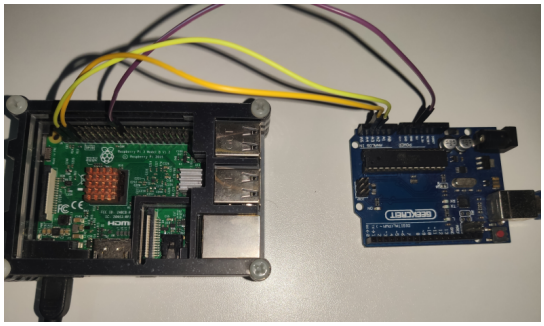

Figura 6. Montagem dos hardwares

B. Implementações

O raspberry pi foi configurado para ser o mestre durante toda a execução, enquanto o arduino atuará como escravo. Principalmente em razão da diferença no suporte de tensão dos dois componentes, visto que, o raspberry funciona a +3.3V se o arduino enviasse uma tensão de +5V o mini pc poderia vir a queimar.

O codificador e decodificador para o *CRC* pode ser implementado tanto em hardware quanto em software [11]. Neste trabalho, as implementações foram realizadas em software. No Arduíno, por ter suporte a linguagem C++, duas classes foram criadas para representar os métodos, com seus respectivos nomes. A classe *CRC* possui como atributos a *codeword*, o gerador e as variáveis *n*, *k* e *r*, além de métodos para calcular divisão entre binários, operação lógica *XOR* e um método que representa a decodificação. A classe *Checksum* possui como atributos o somatório dos valores do pacote, o tamanho da palavra e o valor do *checksum*, além de métodos que calculam a soma do pacote recebido, convertem binário para decimal e vice-versa, calculam o complemento de um número binário, somam dois binários e um método que representa o receptor da comunicação.

O código cíclico *CRC* foi dividido em duas partes: uma para o codificador e também mestre, Raspberry Pi 3, e outra para o decodificador e escravo, Arduíno UNO. A coleta de dados necessária para o processo de codificação pode ser resumida da seguinte maneira: O usuário insere através da interface a *dataword* e o gerador desejado, e de acordo com o modo escolhido dentre os disponíveis:*CRC*, *CRC-8* ou *CRC-10*. Para o caso dos 3 últimos, um gerador padrão é automaticamente

preenchido na interface para facilitar o processo, contudo o usuário permanece com a liberdade de alterá-lo, mas é importante salientar que o uso de um gerador inadequado afeta significativamente na eficiência do algoritmo, portanto é aconselhável não alterá-los de maneira arbitrária, a não ser é claro, para fins de testes. Este gerador padrão segue a seguinte tabela[7]:

Tabela 10.7 *Polinômios-padrão*

Nome	*Polinômio*	*Aplicação*
CRC-8	$X^8 + x^2 + x + 1$	Cabeçalho ATM
CRC-10	$X^{10} + x^9 + x^5 + x^4 + x^2 + 1$	ATM AAL
CRC-16	$X^{16} + x^{12} + x^5 + 1$	HDLC
CRC-32	$X^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$	LANs

Figura 7. Principais polinômios na geração de *CRC*

Por falta de suporte nos hardwares utilizados para o experimento, o *CRC-16* e *CRC-32* não puderam ser implementados. Através da interface, há também a possibilidade de escolher se quer que um ruído seja inserido na transmissão e de quantos bits será o erro que este ruído causará.

No processo de codificação, primeiramente o tamanho da palavra de dados é calculado, assim como a quantidade de bits de redundância. Este último, é calculado utilizando a seguinte fórmula [7]:

$$r = n - k$$

Sendo r a quantidade de bits de redundância, n o tamanho da palavra de código e k tamanho da palavra de dados. Após os cálculos, são então adicionados r *0s* na *dataword*. Esta nova palavra com os bits de paridade adicionados é divida pelo gerador utilizando-se palavras de (n – k + 1) bits. O resto dessa divisão é então anexado à palavra de dados, criando a palavra de código, a qual será enviada para o decodificador através da comunicação *I2C*. Caso a opção de inserir erros tenha sido selecionada, uma função é chamada para gerar uma palavra composta por zeros e uma quantidade especificada pelo usuário de uns, gerados em posições aleatórias. A operação *XOR* é realizada entre a *codeword* correta e esta palavra gerada, para transmitir os erros gerados para a palavra de código. Esta então é finalmente enviada ao decodificador.

No decodificador, nesse caso o Arduíno, a *codeword* é recebida assim como o gerador, devido ao fato de que o gerador deve ser o mesmo para os dois comunicadores. O mesmo cálculo das variáveis n, k e r feitas no codificador, são também feitos aqui. A palavra de código recebida através da comunicação *I2C* é dividida pelo gerador e o resto dessa divisão, também chamado de síndrome, é analisado. Caso o usuário tenha selecionado a opção de transmissão com ruído, haverá bits 1 na síndrome e a *dataword* será descartada. Caso contrário, a síndrome será composta apenas de bits 0, e a *dataword* será aceita.

Para o método de detecção *Checksum*, o código também foi dividido em 2 partes, sendo o Raspberry Pi o emissor e o Arduíno o receptor. Na interface, o usuário insere os números que compõem o pacote que deseja ser enviado e o *checksum* é inicializado com zero. O somatório desses valores é calculado, convertido para binário e então é feita a soma deslocada ciclicamente, utilizando palavras de 4 bits. Como a implementação foi feita com string, as últimas 4 posições são a parte superior da soma, e o restante das posições são a parte inferior. Este valor é então passado por uma função que retorna o complemento de um número binário, sendo este retorno o valor do *checksum*. Por último, o Raspberry envia o pacote inserido pelo usuário juntamente com o *checksum* para o Arduíno.

No lado do receptor, o pacote é recebido, sendo os primeiros números o pacote original digitado pelo usuário, e o último dígito o *checksum*. O processo realizado aqui é semelhante ao do emissor: a soma dos digítos é calculada, convertida para binário, a soma deslocada ciclicamente é realizada e o complemento desta é calculado, da mesma forma explicada acima, e armazenado no *checksum*. A diferença aqui é a verificação do valor do *checksum*, a fim de analisar se há erros na transmissão. Caso este valor tenha dado diferente de zero, significa que os dados foram corrompidos.

C. Interface Gráfica

Com o auxílio do framework *PyQt5*, foi desenvolvida uma interface gráfica com aspecto amigável para o usuário.

Figura 8. Aba *CRC* da interface gráfica

- Para o *CRC*, há 3 modos que podem ser selecionados: *CRC*, *CRC-8* e *CRC-10*
- O usuário deve selecionar o modo desejado e entrar com os bits de entrada, assim como o gerador
- Para os casos do *CRC-8* e *CRC-10*, ao serem selecionados o gerador é automaticamente preenchido com um gerador padrão, conforme Figura 7.
- Na área de "Análise Estatística"há a opção de "Inserir ruído", que ao ser clicada aparece um espaço para inserir a quantidade de erros desejada. Esta opção irá gerar erros em posições aleatórias da *codeword*, antes de ser enviada ao receptor
- Há também a opção de "Simular", conforme será explicado na seção Resultados

- Na área de "Gráficos", temos as opções de visualizar o gráfico tanto do codificador quanto do decodificador. Para o caso do *CRC* estes gráficos representam a palavra de código no emissor e no receptor, respectivamente. Vale observar que se não houver erros durante a transmissão, estes gráficos serão idênticos

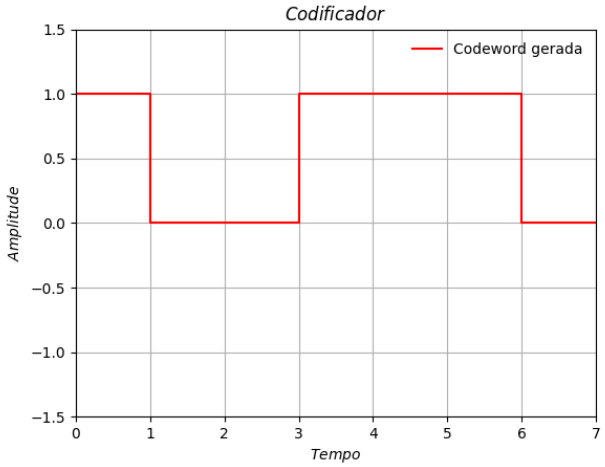

Figura 9. Exemplo de sinal do Codificador

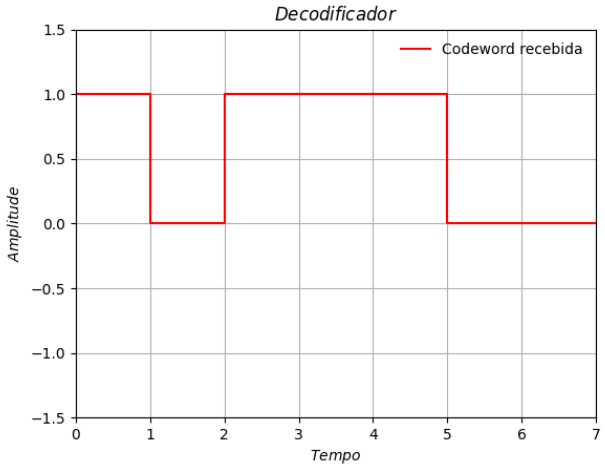

Figura 10. Exemplo de sinal do Decodificador

O exemplo acima demonstra um caso em que há erros durante a transmissão. Vale ressaltar que caso não ocorra erros, ambos os gráficos serão idênticos.

- Para o modo *Checksum*, a interface oferece espaço para o usuário entrar com o pacote que deseja transmitir. Para nossa implementação, a transmissão recebe pacotes de 5 dígitos
- Na área de "Análise Estatística"há a opção de "Simular", a qual foi explicada mais detalhadamente na seção Resultados
- Assim como no outro modo, há a opção "Inserir ruído", a qual para esse modo irá gerar um valor aleatório entre 1 e 10 para somar ao checksum previamente calculado, antes de enviá-lo ao receptor

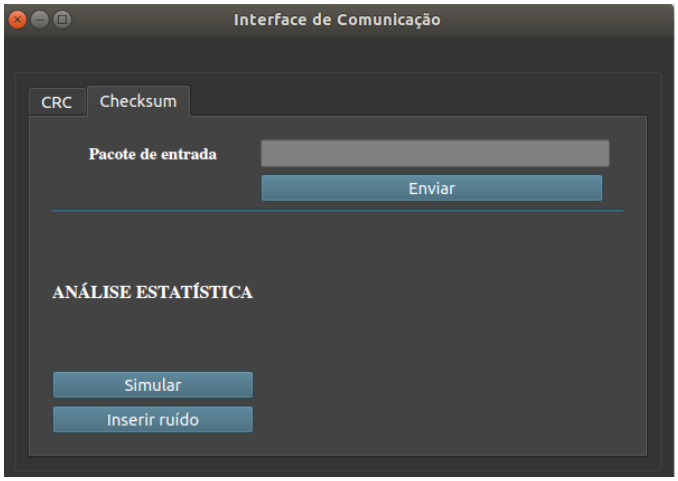

Figura 11. Aba *Checksum* da interface gráfica

IV. Resultados

Um modo de simulação foi implementado para melhorar a visualização dos resultados dos métodos de detecção de erros. Na interface há a opção de "Simular", a qual gera 100 *datawords* aleatórios para o modo *CRC* escolhido. 50% delas utiliza 2 erros injetados e os outros 50% sem erro nenhum. O gerador deve ser fornecido pelo usuário, de acordo com o *CRC* escolhido. Através desse procedimento podemos analisar a porcentagem de sucesso na detecção dos erros no receptor na nossa implementação do método. Para o *Checksum* o processo é semelhante, porém dessa vez um dígito aleatório é somado ao pacote. 100 pacotes são gerados aleatoriamente, com metade deles possuindo um erro injetado. Os dados necessários para análise estatística são gravados pelo Arduíno em um arquivo do tipo "txt". Um código na linguagem Python é responsável por ler estes arquivos gerados pelo Arduíno, extrair a quantidade de erros detectados e calcular a eficiência do gerador, para o caso do *CRC* e da implementação, para o caso do *Checksum*, de acordo com a seguinte fórmula:

$$Eficiência = \frac{\text{Erros detectados}}{\text{Ruídos injetados}}$$

Por fim, outro *script* é responsável pelo *plot* do gráfico, contendo os resultados da simulação tanto das versões disponíveis do CRC quanto do Checksum. A Figura 13 representa a saída de uma simulação qualquer.

A Figura 13 ilustra os resultados obtidos de eficiência na detecção de erros utilizando 3 geradores diferentes para cada um dos métodos *CRC*, *CRC-8* e *CRC-10*. Para a realização dos testes, os geradores utilizados para o *CRC* foram 1011, 1001 e 0001. Para o *CRC-8* os geradores utilizados foram 100000111, 100000001 e 000000001. Para o último modo, os geradores 1100011011, 0100000011 e 0100000000 foram escolhidos. Para os casos do *CRC-8* e *CRC-10* o primeiro gerador foi retirado da tabela de polinômios padrão, de acordo

com a Figura 7. Para os outros geradores a escolha foi feita baseada nos critérios de eficiência a seguir:

Um bom polinômio gerador precisa apresentar as seguintes características:

1. **Ter pelo menos dois termos.**
2. **O coeficiente do termo x^0 deve ser 1.**
3. **Não deve ser divisível por $x^t + 1$, para t entre 2 e $n - 1$.**
4. **Ter o fator $x + 1$.**

Figura 12. Critérios de eficiência para polinômio gerador. Retirado de [7]

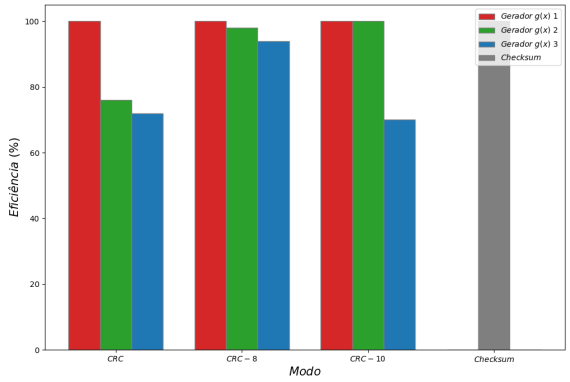

Figura 13. Gráfico de Eficiência x Modo

Como a ideia era testar a eficiência de diferentes geradores, alguns desses critérios foram propositalmente descumpridos para o segundo e terceiro gerador de cada caso, como por exemplo, alguns possuem menos de dois termos, outros possuem o coeficiente do termo x^0 nulo ou então não possuem o fator $x + 1$.

Para o caso do *Checksum*, o gráfico serve apenas para ilustrar a eficiência da implementação do método, visto que não há um gerador interferindo na detecção de erros, mostrando quantos erros foram detectados em relação aos pacotes enviados, que como demonstra o gráfico foi de 100% dos casos.

V. Conclusões

A implementação da comunicação ocorreu de maneira simples, sem muitas complicações. Deixando claro o motivo deste protocolo ser tão difundido no universo dos sistemas embarcados. Já a detecção de erros se mostrou bastante trabalhosa, pois demandava uma boa base teórica, assim como meios eficientes de testar sua exatidão. Através das simulações feitas e dos gráficos gerados, ficou nítida a importância de geradores adequados para o CRC, que para cada aplicação possuem suas vantagens e desvantagens. Quanto ao Checksum, os dados obtidos foram exatamente os esperados, com uma precisão igual ou aproximadamente 100%, ou seja, dos 50 erros injetados ele conseguia captar todos ou praticamente todos. Com estes métodos de detecção de erros implementados, é possível garantir uma linha de transmissão com maior confiabilidade para o usuário, não apenas para o protocolo I2C, como para todos os outros.

Referências

[1] *I²C-bus specification and user manual*, 6th ed., https://www.nxp.com/docs/en/user-guide/UM10204.pdf, NXP Semiconductors, april 2014.

[2] *Protocolo I2C*, http://www.univasf.edu.br/~romulo.camara/novo/wp-content/uploads/2013/11/Barramento-e-Protocolo-I2C.pdf, UNIVASF, november 2013.

[3] M. P. Kumar and U. Rani. Nelakuditi, "Iot and i2c protocol based m-health medication assistive system for elderly people," in *2019 IEEE 16th India Council International Conference (INDICON)*, 2019, pp. 1–4.

[4] C. Liu, Q. Meng, T. Liao, X. Bao, and C. Xu, "A flexible hardware architecture for slave device of i2c bus," in *2019 International Conference on Electronic Engineering and Informatics (EEI)*, 2019, pp. 309–313.

[5] *I²C MANUAL*, 24th ed., https://www.nxp.com/docs/en/application-note/AN10216.pdf, Philips Semiconductors, march 2003.

[6] J. Rochol, *Comunicação de Dados*, 22nd ed. bookman, 2012.

[7] B. A. Forouzan, *Comunicação de Dados e Redes de Computadores*, 4th ed. McGraw-Hill, 2010.

[8] D. Fedorov, "Information technology of image recognition using checksums," in *Proceedings of International Conference on Modern Problem of Radio Engineering, Telecommunications and Computer Science*, 2012, pp. 429–429.

[9] N. Saxena and E. McCluskey, "Analysis of checksums, extended-precision checksums, and cyclic redundancy checks," *IEEE Transactions on Computers*, vol. 39, no. 7, pp. 969–975, 1990.

[10] D. Workshop. I2c with arduino and raspberry pi - two methods. Youtube. [Online]. Available: https://www.youtube.com/watch?v=me7mhrRbspk&t=1366s

[11] A. S. Panda and G. L. Kumar, "Comparison of serial data-input crc and parallel data-input crc design for crc-8 atm hec employing mlfsr," in *2014 International Conference on Electronics and Communication Systems (ICECS)*, 2014, pp. 1–4.

APÊNDICE A
INTERFACE.PY

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# Raspberry Pi Master for Arduino Slave
# Connects to Arduino via I2C

import sys
import random
import unicodedata
import datetime
import os
import math
from PyQt5 import QtCore, QtWidgets, QtGui
from PyQt5.QtWidgets import QApplication, QLabel, QWidget
from PyQt5.QtGui import QPalette, QColor
from scipy import signal
import matplotlib.pyplot as plt
import numpy as np
from smbus import SMBus
import CRC
import Checksum
from time import sleep

addr = 0x8 # bus address
bus = SMBus(1) # indicates /dev/i2c-1

class Application(QWidget):
    def __init__(self, parent=None):
        super(Application,self).__init__()
        self.setWindowTitle("Interface de Comunica  o")

        #------------atributos-----------#
        self.bits = 0
        self.active_erro = False
        self.active_erro_checksum = False
        self.qtd_erro = 0
        self.original_codeword = []
        self.codeword_plot = []
        self.crc_modes = ["CRC","CRC-8","CRC-10"]
        self.mode = "CRC"

        #------------------fonte-------------------#
        self.main_font = QtGui.QFont('Times',11, QtGui.QFont.Bold)
        self.secundary_font = QtGui.QFont('Times',12, QtGui.QFont.Bold)
        self.button_color = 'QPushButton {background-color: #365F73;}'

        self.Interface()

        self.layout = QtWidgets.QVBoxLayout()
        self.layout.addWidget(self.formGroupBox1)
        self.setLayout(self.layout)

    def Interface(self):
        self.formGroupBox1 = QtWidgets.QGroupBox()

        #--------------layout--------------------#
        layout = QtWidgets.QHBoxLayout()

        #-------------------tabs--------------------#
        self.tabs = QtWidgets.QTabWidget()
        self.tabs.addTab(self.CRCTabUI(), "CRC")
        self.tabs.addTab(self.ChecksumTabUI(), "Checksum")
        self.tabs.currentChanged.connect(self.tabsIndex)
        layout.addWidget(self.tabs)

        self.formGroupBox1.setLayout(layout)

    def CRCTabUI(self):
        crcTab = QtWidgets.QWidget()

        #--------------layouts--------------#
        layout = QtWidgets.QGridLayout()
        layout.setContentsMargins(20,20,20,20)

```

```
        #--------------separator------------#
        separator = QtWidgets.QFrame()
        separator.setStyleSheet("background-color: #365F73")
        separator.setFrameShape(QtWidgets.QFrame.HLine)
        separator.setFrameShadow(QtWidgets.QFrame.Sunken)

        separatorVertical = QtWidgets.QFrame()
        separatorVertical.setStyleSheet("background-color: #365F73")
        separatorVertical.setFrameShape(QtWidgets.QFrame.VLine)
        separatorVertical.setFrameShadow(QtWidgets.QFrame.Sunken)

        #----------------entrada-----------------#
        self.entrada_crc = QtWidgets.QLineEdit()
        self.entrada_crc.setAlignment(QtCore.Qt.AlignLeft)
        self.entrada_crc.setStyleSheet('QLineEdit { background-color: gray; color: black}')
        label = QtWidgets.QLabel("Bits de entrada")
        label.setFont(self.main_font)
        label.setAlignment(QtCore.Qt.AlignCenter)
        layout.addWidget(label, 0, 0)
        layout.addWidget(self.entrada_crc, 0, 1, 1, 2)

        #--------------gerador--------------#
        self.gerador = QtWidgets.QLineEdit()
        self.gerador.setStyleSheet('QLineEdit { background-color: gray; color: black}')
        self.gerador_label = QtWidgets.QLabel("Gerador g(x)")
        self.gerador_label.setFont(self.main_font)
        self.gerador_label.setAlignment(QtCore.Qt.AlignCenter)
        layout.addWidget(self.gerador_label, 1, 0)
        layout.addWidget(self.gerador, 1, 1, 1, 2)

        #------------enviar--------------#
        buttonEnviar = QtWidgets.QPushButton("Enviar")
        buttonEnviar.setStyleSheet(self.button_color)
        buttonEnviar.setSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Fixed)
        layout.addWidget(buttonEnviar, 2, 1)
        buttonEnviar.clicked.connect(self.send)

        #------------combobox-------------#
        cb = QtWidgets.QComboBox()
        cb.addItems(["CRC","CRC-8","CRC-10"])
        cb.currentIndexChanged.connect(self.combo)
        layout.addWidget(cb, 0, 3)

        layout.addWidget(separator, 3, 0, 1, 5) # horizontal separator
        layout.addWidget(separatorVertical, 4, 1, 5, 1) # vertical separator

        #--------------an lises-------------#
        analysis_title = QtWidgets.QLabel("AN LISE ESTAT STICA")
        analysis_title.setFont(self.secundary_font)
        analysis_title.setAlignment(QtCore.Qt.AlignCenter)
        layout.addWidget(analysis_title, 4, 0)

        #----------------simular-----------------#
        buttonSimular = QtWidgets.QPushButton("Simular")
        buttonSimular.setStyleSheet(self.button_color)
        layout.addWidget(buttonSimular, 5, 0)
        buttonSimular.clicked.connect(self.automatizado)

        #---------simula  o de an lise de erros------------#
        buttonSimulacao = QtWidgets.QPushButton("Inserir ru do")
        buttonSimulacao.setStyleSheet(self.button_color)
        layout.addWidget(buttonSimulacao, 6, 0)
        buttonSimulacao.clicked.connect(self.simulacao)

        #------------op  es de erro--------------#
        self.erro_crc = QtWidgets.QLineEdit()
        self.erro_crc.setPlaceholderText("Quantidade de erros")
        self.erro_crc.setStyleSheet('QLineEdit { background-color: gray; color: black}')
        layout.addWidget(self.erro_crc, 7, 0)
        self.erro_crc.setHidden(True)

        self.buttonSaveError_crc = QtWidgets.QPushButton("Salvar")
        self.buttonSaveError_crc.setStyleSheet(self.button_color)
        layout.addWidget(self.buttonSaveError_crc, 8, 0)
        self.buttonSaveError_crc.clicked.connect(self.saveError)
        self.buttonSaveError_crc.setHidden(True)

```

```
        #--------------gr ficos-------------#
        plot_title = QtWidgets.QLabel("GR FICOS")
        plot_title.setFont(self.secundary_font)
        plot_title.setAlignment(QtCore.Qt.AlignCenter)
        layout.addWidget(plot_title, 4, 2)

        # codificador
        buttonCodificador = QtWidgets.QPushButton("Codificador")
        buttonCodificador.setStyleSheet(self.button_color)
        buttonCodificador.setSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Expanding)
        layout.addWidget(buttonCodificador, 5, 2, 2, 1)
        buttonCodificador.clicked.connect(self.codificador)

        # decodificador
        buttonDecodificador = QtWidgets.QPushButton("Decodificador")
        buttonDecodificador.setStyleSheet(self.button_color)
        buttonDecodificador.setSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Preferred)
        layout.addWidget(buttonDecodificador, 7, 2, 2, 1)
        buttonDecodificador.clicked.connect(self.decodificador)

        crcTab.setLayout(layout)
        return crcTab

    def ChecksumTabUI(self):
        checksumTab = QtWidgets.QWidget()

        #--------------layouts--------------#
        layout = QtWidgets.QGridLayout()
        layout.setContentsMargins(20,20,20,20)

        #--------------separator------------#
        separator = QtWidgets.QFrame()
        separator.setStyleSheet("background-color: #365F73")
        separator.setFrameShape(QtWidgets.QFrame.HLine)
        separator.setFrameShadow(QtWidgets.QFrame.Sunken)

        separatorVertical = QtWidgets.QFrame()
        separatorVertical.setStyleSheet("background-color: #365F73")
        separatorVertical.setFrameShape(QtWidgets.QFrame.VLine)
        separatorVertical.setFrameShadow(QtWidgets.QFrame.Sunken)

        #----------------entrada-----------------#
        self.entrada_checksum = QtWidgets.QLineEdit()
        self.entrada_checksum.setAlignment(QtCore.Qt.AlignLeft)
        self.entrada_checksum.setStyleSheet('QLineEdit { background-color: gray; color: black}')
        label = QtWidgets.QLabel("Pacote de entrada")
        label.setFont(self.main_font)
        label.setAlignment(QtCore.Qt.AlignCenter)
        layout.addWidget(label, 0, 0)
        layout.addWidget(self.entrada_checksum, 0, 1, 1, 2)

        #------------enviar--------------#
        buttonEnviar = QtWidgets.QPushButton("Enviar")
        buttonEnviar.setStyleSheet(self.button_color)
        buttonEnviar.setSizePolicy(QtWidgets.QSizePolicy.Expanding, QtWidgets.QSizePolicy.Fixed)
        layout.addWidget(buttonEnviar, 2, 1)
        buttonEnviar.clicked.connect(self.send)

        layout.addWidget(separator, 3, 0, 1, 5) # horizontal separator

        #--------------an lises-------------#
        analysis_title = QtWidgets.QLabel("AN LISE ESTAT STICA")
        analysis_title.setFont(self.secundary_font)
        #analysis_title.setAlignment(QtCore.Qt.AlignCenter)
        layout.addWidget(analysis_title, 4, 0)

        #----------------simular-----------------#
        buttonSimular = QtWidgets.QPushButton("Simular")
        buttonSimular.setStyleSheet(self.button_color)
        layout.addWidget(buttonSimular, 5, 0)
        buttonSimular.clicked.connect(self.automatizado)

        #---------simula  o de an lise de erros------------#
        self.buttonSimulacaoChecksum = QtWidgets.QPushButton("Inserir ru do")
        self.buttonSimulacaoChecksum.setStyleSheet(self.button_color)
        self.buttonSimulacaoChecksum.setCheckable(True) # toggle button
        layout.addWidget(self.buttonSimulacaoChecksum, 6, 0)
```

```
        self.buttonSimulacaoChecksum.clicked.connect(self.simulacao)

        checksumTab.setLayout(layout)
        return checksumTab

    def tabsIndex(self):
        if self.tabs.currentIndex() == 0: self.mode = "CRC"
        elif self.tabs.currentIndex() == 1: self.mode = "Checksum"

    def automatizado(self):
        print("Simula  o autom tica")
        if self.mode in self.crc_modes:
            message = QtWidgets.QMessageBox(QtWidgets.QMessageBox.Information,'Aviso','Esta simula  o
    gera 100 datawords aleat rios. 50% deles utiliza 2 erros injetados. Os outros 50% sem erro.')
        elif self.mode == "Checksum":
            message = QtWidgets.QMessageBox(QtWidgets.QMessageBox.Information,'Aviso','Esta simula  o
    gera 100 pacotes aleat rios. 50% deles utiliza erros injetados. Os outros 50% sem erro.')
        message.exec_()
        g_x = ""
        number_iterations = 100
        if self.mode in self.crc_modes:
            if(self.gerador.text() == ""): QtWidgets.QMessageBox.warning(self, 'Error', 'Entre com o
    gerador g(x)')
            else: g_x = str(self.gerador.text())
        for i in range(number_iterations):
            if self.mode in self.crc_modes and g_x != "":
                codeword = CRC.simulation(i,g_x,self.mode)
                codeword_array = [int(x) for x in codeword] # converte pra array de int
                codeword_with_g_x = [int(x) for x in g_x]
                codeword_with_g_x.insert(0,9) # pra indicar pro receptor que aqui come a o g(x)
                codeword_array += codeword_with_g_x
                bus.write_i2c_block_data(addr,0,codeword_array) # envia pro arduino com offset 0
                sleep(1) # precisa dar sleep, se n o a comunica  o falha
                print("\n")
            elif self.mode == "Checksum":
                pack = Checksum.simulation(i)
                bus.write_i2c_block_data(addr,1,pack) # envia pro arduino com offset 1
                sleep(1) # precisa dar sleep, se n o a comunica  o falha
                print("\n")
        self.gerador.clear()
        print("----------------Simula  o encerrada--------------------\n")

    def combo(self,i):
        self.mode = "CRC"
        if i == 0: self.gerador.setText("")
        elif i == 1:
            self.gerador.setText("100000111")
            self.mode = "CRC-8"
        elif i == 2:
            self.gerador.setText("1100011011")
            self.mode = "CRC-10"

    def save(self):
        retorno = 0 # retorna 0 se erro

        try:
            self.entry_crc = str(self.entrada_crc.text()) # pega string de entrada
            self.bits = retorno = [int(x) for x in self.entry_crc] # converte str pra array de int
            print("Bits de entrada: ", self.bits)
        except ValueError: # entrada n o num rica ou vazia
            QtWidgets.QMessageBox.warning(self, 'Error', 'Entrada inv lida! Digite novamente.')

        self.entrada_crc.clear() # limpa a entrada
        return retorno

    def save_checksum(self):
        retorno = 0 # retorna 0 se erro

        self.entry_checksum = str(self.entrada_checksum.text()) # pega string de entrada
        print("Pacote de entrada: ", self.entry_checksum)

        if len(self.entry_checksum) == 0: QtWidgets.QMessageBox.warning(self, 'Error', 'Entrada inv lida!
    Digite novamente.')
        else: retorno = 1

        self.entrada_checksum.clear() # limpa a entrada
        return retorno
```

```
301
302     def send(self):
303         if self.mode in self.crc_modes and self.save(): # se entrada v lida e CRC
304             print("\n-----------------MODE = {}----------------".format(self.mode))
305             g_x = str(self.gerador.text())
306             codeword, original = CRC.encoder(self.entry_crc,g_x,self.qtd_erro)
307             codeword_array = [int(x) for x in codeword] # converte pra array de int
308             self.original_codeword = [int(x) for x in original] # pra plotar o codeword original
309             self.codeword_plot = [int(x) for x in codeword] # pra plotar o codeword
310             codeword_with_g_x = [int(x) for x in g_x]
311             codeword_with_g_x.insert(0,9) # pra indicar pro receptor que aqui come a o g(x)
312             codeword_array += codeword_with_g_x
313             bus.write_i2c_block_data(addr,0,codeword_array) # envia pro arduino com offset 0
314             self.gerador.clear()
315             print("-------------------------------------------------\n")
316         elif self.mode == "Checksum" and self.save_checksum(): # se entrada v lida e Checksum
317             print("\n-----------------MODE = {}----------------".format(self.mode))
318             in_list = self.entry_checksum.split()
319             if len(in_list) < 5: QtWidgets.QMessageBox.warning(self, 'Error', 'Entrada inv lida! Pacote
    menor que 5.')
320             else:
321                 somatorio = Checksum.calculate_sum(in_list)
322                 checksum = Checksum.emissor(somatorio,4,self.active_erro_checksum)
323                 in_list.append(checksum) # adiciona o checksum no array pra enviar
324                 bus.write_i2c_block_data(addr,1,in_list) # envia pro arduino com offset 1
325             print("-------------------------------------------------\n")
326
327     def plot(self, array, name, type):
328         y = array # bits
329         y.insert(0,0) # ajuste pro plot, insere um 0 na primeira posi  o do array
330
331         x = [i for i in range(len(y))] # posicoes no tempo, a cada 1 seg, tamanho de acordo com a entrada y
332
333         plt.step(x,y,'r',label=type)
334         plt.title("${}$".format(name))
335         plt.xlabel('$Tempo$')
336         plt.ylabel('$Amplitude$')
337         plt.grid(True, which='both')
338         plt.legend(frameon=False)
339         plt.xlim(0,len(x) - 1)
340         plt.ylim(-1.5,1.5)
341         plt.show()
342
343     def codificador(self):
344         print("Codificador")
345
346         if len(self.original_codeword) != 0: self.plot(self.original_codeword,"Codificador","Codeword
    gerada")
347         else: QtWidgets.QMessageBox.warning(self, 'Error', 'Primeiro digite a entrada e pressione Enviar.')
348
349     def decodificador(self):
350         print("Decodificador")
351
352         if len(self.codeword_plot) != 0: self.plot(self.codeword_plot,"Decodificador","Codeword recebida")
353         else: QtWidgets.QMessageBox.warning(self, 'Error', 'Primeiro digite a entrada e pressione Enviar.')
354
355     def simulacao(self):
356         print("Inserir ru do")
357         if self.mode in self.crc_modes:
358             if self.active_erro:
359                 self.erro_crc.setHidden(True)
360                 self.buttonSaveError_crc.setHidden(True)
361                 self.active_erro = False
362                 self.qtd_erro = 0
363             else:
364                 self.erro_crc.setHidden(False)
365                 self.buttonSaveError_crc.setHidden(False)
366                 self.active_erro = True
367         elif self.mode == "Checksum":
368             if self.buttonSimulacaoChecksum.isChecked():
369                 self.buttonSimulacaoChecksum.setStyleSheet("background-color : lightblue")
370                 self.active_erro_checksum = True
371             else:
372                 self.buttonSimulacaoChecksum.setStyleSheet(self.button_color)
373                 self.active_erro_checksum = False
374
375     def saveError(self):
```

```
        if self.erro_crc.text() != "": self.qtd_erro = int(self.erro_crc.text())
        print("Quantidade de erros escolhida: ", self.qtd_erro)

def dark_theme(app):
    dark_palette = QPalette()

    dark_palette.setColor(QPalette.Window, QColor(53, 53, 53))
    dark_palette.setColor(QPalette.WindowText, QtCore.Qt.white)
    dark_palette.setColor(QPalette.Base, QColor(25, 25, 25))
    dark_palette.setColor(QPalette.AlternateBase, QColor(53, 53, 53))
    dark_palette.setColor(QPalette.ToolTipBase, QtCore.Qt.white)
    dark_palette.setColor(QPalette.ToolTipText, QtCore.Qt.white)
    dark_palette.setColor(QPalette.Text, QtCore.Qt.white)
    dark_palette.setColor(QPalette.Button, QColor(53, 53, 53))
    dark_palette.setColor(QPalette.ButtonText, QtCore.Qt.white)
    dark_palette.setColor(QPalette.BrightText, QtCore.Qt.red)
    dark_palette.setColor(QPalette.Link, QColor(42, 130, 218))
    dark_palette.setColor(QPalette.Highlight, QColor(42, 130, 218))
    dark_palette.setColor(QPalette.HighlightedText, QtCore.Qt.black)
    app.setPalette(dark_palette)

    app.setStyleSheet("QToolTip { color: #ffffff; background-color: #2a82da; border: 1px solid white; }")

if __name__ == "__main__":
    app = QApplication(sys.argv)
    app.setStyle('Fusion')
    dark_theme(app)

    window = Application()
    window.resize(600,400)

    window.show()

    sys.exit(app.exec_())
```

APÊNDICE B
CRC.PY

```
import random

def gera_certo(qtd = 6):
  dataword = ""
  for i in range(qtd):
    dataword += str(random.randint(0,1))
  print("Dataword gerada = ", dataword)
  return dataword

"""
50% das palavras geradas contem 2 erros injetados
"""
def simulation(value,g,mode):
  if mode == "CRC": qtd = 4
  elif mode == "CRC-8": qtd = 10
  elif mode == "CRC-10": qtd = 11

  dataword = gera_certo(qtd)
  codeword = ""

  if value % 2 != 0: codeword,_ = encoder(dataword,g,2)
  else: codeword,_ = encoder(dataword,g)

  return codeword

def XOR(a, b):
  if a == b: return '0'
  else: return '1'

def divisao(a,b,k,n):
  dividendo = ""
  divisor = ""
  quociente = ""
  resto = ""
  prox = ""
  i = 0

  for m in range(len(b)): dividendo += a[m]
```

```
39
40    while i < n:
41      if dividendo[0] == '1': divisor = b
42      elif dividendo[0] == '0':
43        divisor = ""
44        for l in range(len(b)): divisor += '0'
45
46      for j in range(1,len(b)): prox += XOR(dividendo[j],divisor[j])
47
48      if i == 0: i += len(b)
49      else: i += 1
50
51      if i < n: prox += a[i]
52
53      dividendo = prox
54      prox = ""
55
56    resto = dividendo
57    return resto
58
59  """
60  Insere qtd 1s em um bin rio 000000 e faz XOR com o codeword
61  @param: qtd -> quantidade de erros
62      size -> tamanho da palavra
63  @return: codeword com erro nas p o s i  e s geradas a l e a t r i a s
64  """
65  def gera_erros(codeword,qtd,size):
66    send = ""
67    positions = random.sample(range(0,size),qtd)
68    print("posicoes: ", positions)
69    for i in range(size):
70      if i in positions: send += '1'
71      else: send += '0'
72
73    codeword_erro = ""
74    for i in range(size): codeword_erro += XOR(codeword[i],send[i])
75    print("Codeword com erro gerado = ", codeword_erro)
76    return codeword_erro
77
78  """
79  @param: qtd_erro -> se != 0, envia codeword com qtd_erro erros
80  @return: codeword -> codeword gerada, com ou sem erro
81       original -> codeword original, caso a  o p   o com erro tenha sido selecionada (usado para o g r f i c o)
82             caso  o p   o  sem erro selecionada, original == codeword
83  """
84  def encoder(dataword,g,qtd_erro = 0):
85    print("----------Encoder--------------")
86    k = len(dataword)
87    r = len(g) - 1
88    n = r + k # r = n - k
89    bits_paridade = ""
90    for i in range(r):
91      bits_paridade += "0"
92    print("Bits de paridade = ",bits_paridade)
93
94    data_paridade = ""
95    data_paridade += dataword + bits_paridade
96    print("Dataword com bits de paridade = ",data_paridade)
97
98    resto = divisao(data_paridade,g,k,n)
99
100   codeword = dataword + resto
101   print("Codeword = ", codeword)
102
103   original = codeword
104   if(qtd_erro): return gera_erros(codeword,qtd_erro,n), original
105   else: return codeword, original
106
107 if __name__ == "__main__":
108   g_x = "1011"
109
110   bits = input('Digite a entrada de bits: ')
111   encoder(bits,g_x)
```

APÊNDICE C
CHECKSUM.PY

```
import random

def gera_certo(qtd = 5):
  pack = []
  for i in range(qtd):
    pack.append(str(random.randint(0,10)))
  print("Pacote gerado = ", pack)
  return pack

"""
50% das palavras geradas contem erros injetados
"""
def simulation(value):
  pack = gera_certo()
  somatorio = calculate_sum(pack)
  if value % 2 != 0: checksum = emissor(somatorio,4,True)
  else: checksum = emissor(somatorio,4)
  pack.append(checksum)
  return pack

def complemento(s):
  ret = ""
  t = len(s)
  for i in range(t):
    if s[i] == '0': ret += '1'
    elif s[i] == '1': ret += '0'
  return ret

"""
Convert int to binary (4 bits)
"""
def toBinary(n):
  return "{0:04b}".format(n)

def toDecimal(n):
  decimalNumber = 0
  i = 0
  remainder = 0
  while n != 0:
    remainder = n % 10
    n /= 10
    decimalNumber += remainder*pow(2,i)
    i += 1
  return decimalNumber

def binaryStringToint(s):
    return int(s,2)

def soma(a,b):
    return toBinary(a + b)

def calculate_sum(g):
    for i in range(len(g)): g[i] = int(g[i])
    return sum(g)

"""
Gera um n mero  aleat rio pra somar com o pacote enviado
"""
def gera_erro(checksum):
  erro = random.randint(1,10)
  print("Erro gerado = ", erro)
  return checksum + erro

"""
Faz o c lculo da soma no checksum
Ex.: 11011 -> 1011
      +  1
"""
def checksum_sum(word,size):
  word_length = len(word)
  n = word_length - size # n meros da frente que tem que retirar
  sup = ""
  inf = ""
  for i in range(n,word_length): sup += word[i] # n meros que v o somar com os retirados
  for i in range(n): inf += word[i] # retirados
  s = soma(binaryStringToint(sup),binaryStringToint(inf))
  return s
```

```

def emissor(somatorio,size,error = False):
  print("\n----------Emissor----------")
  word = ""
  word += toBinary(somatorio) # soma convertida em bin rio
  word_length = len(word)
  # soma em bin rio <= tamanho da palavra
  if word_length <= size: checksum = binaryStringToint(complemento(word)) # soma vai ser o pr prio word
  else: # soma em bin rio > tamanho da palavra
    s = checksum_sum(word,size)
    while len(s) > size: # enquanto a soma der maior que size bits (palavras de size bits)
      s = checksum_sum(s,size)
    checksum = binaryStringToint(complemento(s))
  print("Checksum = ", checksum)
  if error: return gera_erro(checksum)
  else: return checksum

if __name__ == "__main__":
  package = input("Digite o pacote de entrada: ")
  in_list = package.split()
  somatorio = calculate_sum(in_list)
  checksum = emissor(somatorio,4)
```

APÊNDICE D
CHECKSUM.H

```
#ifndef CHECKSUM_H
#define CHECKSUM_H

#include <Arduino.h>

class Checksum
{
public:
    Checksum(int s) : _sum(0), _size(s), _checksum(0) {}
    ~Checksum() {}
    void receptor(int package[])
    {
        Serial.println("------------Receptor--------------");
        Serial.print("Pacote recebido: "); for(int i = 0; i < 6; i++) Serial.print(String(package[i]) + "\t
    "); Serial.println();
        calculate_sum(package);
        String word = toBinary(_sum);
        //Serial.print("WORD = "); Serial.println(word);
        String s = checksum_sum(word);
        while(s.length() > _size) s = checksum_sum(s);
        //Serial.print("Soma = "); Serial.println(s);
        if(s.length() < 4) s = adjust(s);
        //Serial.print("NOVA SOMA = "); Serial.println(s);
        _checksum = binaryStringToint(complemento(s));
        Serial.print("Checksum = "); Serial.println(_checksum);
        if(_checksum != 0) Serial.println("ERRO: Dados corrompidos");
    }
    /*
     * Faz o c lculo da soma no checksum
     *  Ex.: 11011 -> 1011
     *               +  1
     */
    String checksum_sum(String word)
    {
      int word_length = word.length();
      int n = word_length - _size; // numeros da frente que tem que retirar
      String sup = "";
      String inf = "";
      for(int i = n; i < word_length; i++) sup += word[i]; // numeros que vao somar com os retirados
      if(n > 0) for(int i = 0; i < n; i++) inf += word[i]; // retirados
      else inf = "0"; // se palavra for menor ou igual ao tamanho da palavra inf    0
      //Serial.print("sup = "); Serial.println(sup);
      //Serial.print("inf = "); Serial.println(inf);
      String s = soma(binaryStringToint(sup),binaryStringToint(inf));
      return s;
    }
    /*
     * Palavras de 4 bits, completa o bin rio  at   4 bits
     */
    String adjust(String s)
```

```
    {
      int i = s.length();
      String new_s = "";
      while(i != 4)
      {
        new_s += '0';
        i++;
      }
      return new_s + s;
    }
    /*
     * Inverte uma string de tr s pra frente
     */
    String reverse(String s)
    {
      String r = s;
      for(int i = 0; i < s.length(); i++) r[i] = s[s.length() - 1 - i];
      return r;
    }
    /*
     * Converte string de bin rio pra int
     * (https://stackoverflow.com/questions/2343099/convert-binary-format-string-to-int-in-c)
     */
    int binaryStringToint(String s)
    {
      char* start = &s[0];
      int total = 0;
      while (*start)
      {
       total *= 2;
       if (*start++ == '1') total += 1;
      }
      return total;
    }
    String soma(int a, int b) { return toBinary(a + b); }
    String complemento(String s)
    {
        String ret = "";
        int t = s.length();
        for(int i = 0; i < t; i++)
        {
            if(s[i] == '0') ret += '1';
            else if(s[i] == '1') ret += '0';
        }
        return ret;
    }
    String toBinary(int n)
    {
        String r;
        while(n != 0) { r = (n % 2 == 0 ? "0" : "1") + r; n /= 2; }
        return r;
    }
    int toDecimal(int n)
    {
        int decimalNumber = 0, i = 0, remainder;
        while (n!=0)
        {
            remainder = n%10;
            n /= 10;
            decimalNumber += remainder*pow(2,i);
            ++i;
        }
        return decimalNumber;
    }
    void calculate_sum(int g[]) { for(int i = 0; i < 6; i++) _sum += g[i]; }
    /*
     * Limpa todos os atributos
     */
    void clean()
    {
      _sum = 0;
      _checksum = 0;
    }

private:
    int _sum;
    int _size; // tamanho da palavra
```

```
    int _checksum;
};

#endif
```

APÊNDICE E
CRC.H

```
#ifndef CRC_H
#define CRC_H

#include <Arduino.h>

class CRC
{
private:
    String _codeword; // sa da
    String _g; // gerador g(x)
    int _n; // n = k + r
    int _k; // bits de entrada
    int _r;
public:
    CRC() : _codeword(""), _g(""), _n(0), _k(0), _r(0) {}
    ~CRC() {}
    void decoder(String codeword, String g_x)
    {
      Serial.println("----------------Decoder---------------");
      Serial.println("Codeword recebida: " + codeword);
      Serial.println("Gerador: " + g_x);

      _codeword = codeword;
      _g = g_x;
      _r = _g.length() - 1;
      _n = _codeword.length();
      _k = _n - _r;

      String resto = divisao(_codeword, _g);
      bool error = false;

      for(int i = 0; i < (_k - 1); i++)
      {
          if(resto[i] == '1'){ error = true; }
      }

      if(error) Serial.println("d(x) descartada");
      else Serial.println("d(x) aceita");
    }
    String divisao(String a, String b)
    {
      String dividendo = "";
      String divisor = "";
      String quociente = "";
      String resto = "";
      String next = "";
      int i = 0;

      int b_length = b.length();
      for(int k = 0; k < b_length; k++) dividendo += a[k];

      while(i < _n)
      {
          if(dividendo[0] == '1') divisor = _g;
          else if(dividendo[0] == '0')
          {
              divisor = "";
              for(int l = 0; l < b_length; l++) divisor += '0';
          }

          for(int j = 1; j < b_length; j++) next += XOR(dividendo[j],divisor[j]);

          if(i == 0) i += b_length;
          else i++;

          if(i < _n) next += a[i];

          dividendo = next;
```

```
          next = "";
      }

      resto = dividendo;

      return resto;
    }
    char XOR(char a, char b)
    {
      if(a == b) return '0';
      else return '1';
    }
    /*
     * Limpa todos os atributos
     */
    void clean()
    {
      _codeword = "";
      _g = "";
      _n = 0;
      _k = 0;
      _r = 0;
    }
};

#endif
```

APÊNDICE F
I2C_SLAVE.INO

```
// Include the Wire library for I2C
#include <Wire.h>
#include "CRC.h"
#include "Checksum.h"

String package;
String g_x;
bool gerador = false;
CRC c;
int times = 0;
String mode = "CRC";
int pack[6];
int pos = 0;
Checksum cs(4);

void setup() {
  // Join I2C bus as slave with address 8
  Wire.begin(0x8);

  // Call receiveEvent when data received
  Wire.onReceive(receiveEvent);

  Serial.begin(9600);
}

// Function that executes whenever data is received from master
void receiveEvent(int howMany)
{
  while(Wire.available()) // loop through all but the last
  {
    int c = Wire.read(); // receive byte as a character

    if(times == 0) // primeiro caracter    o offset, 0 pra CRC e 1 pra Checksum
    {
      times = 1;
      if(c == 0) mode = "CRC";
      else if(c == 1) mode = "CHECKSUM";
    }
    else
    {
      if(mode == "CRC")
      {
        // tem que ser nessa ordem, pq quando for 9,  s   descarta o valor e pega a partir do pr ximo
        if(gerador) g_x += String(c);
        else if(c == 9) gerador = true;
        else package += String(c);
```

```
      }
      else if(mode == "CHECKSUM")
      {
        pack[pos] = c;
        pos += 1;
      }
    }
  }
  Serial.println("-----------------MODE = " + mode + "----------------");

  if(mode == "CRC") c.decoder(package,g_x);
  else if(mode == "CHECKSUM") cs.receptor(pack);
  package = "";
  g_x = "";
  times = 0;
  pos = 0;
  gerador = false;
  c.clean();
  cs.clean();

  Serial.println("------------------------------------------------");
}

void loop() {
  delay(100);
}
```

APÊNDICE G
EFFICIENCY.PY

```
"""
L  o arquivo de sa da do Ardu no, extrai o n mero de erros detectados e calcula a efici ncia
"""

import sys

"""
Search for the given string in file and return lines containing that string, along with line numbers
(https://thispointer.com/python-search-strings-in-a-file-and-get-line-numbers-of-lines-containing-the-
    string/)
"""
def search_string_in_file(file_name, string_to_search):
    line_number = 0
    list_of_results = []
    # Open the file in read only mode
    with open(file_name, 'r') as read_obj:
        # Read all lines in the file one by one
        for line in read_obj:
            # For each line, check if line contains the string
            line_number += 1
            if string_to_search in line:
                # If yes, then add the line number & line as a tuple in the list
                list_of_results.append((line_number, line.rstrip()))
    # Return list of tuples containing line numbers and lines where string is found
    return list_of_results

"""
Escreve as efici ncias em um arquivo
"""
def write_file(efficiency,mode):
    file = open("Output/eficiencias_{}.txt".format(mode),"a")
    file.write(str(efficiency) + '\n')
    file.close()

"""
Conta quantas linhas tem no arquivo
"""
def count_lines(file):
    nonempty_lines = [line.strip("\n") for line in file if line != "\n"]
    line_count = len(nonempty_lines)
    file.close()
    print(line_count)
    return line_count

"""
Abre o arquivo de sa da do Ardu no e calcula a efici ncia do gerador/modo
```

```
@param: mode -> CRC, CRC-8, CRC-10 ou CHECKSUM
"""
def read_file(mode):
    crc_modes = ["CRC","CRC-8","CRC-10"]

    # acha as linhas onde tem resultado d(x)
    if mode in crc_modes: search = "d(x)"
    elif mode == "CHECKSUM": search = "Checksum"
    matched_lines = search_string_in_file("Output/output_{}.txt".format(mode),search)
    total = len(matched_lines) # n mero total de pacotes enviados

    num_rejeitou = 0
    if mode in crc_modes:
        # verifica quantas d(x) foram descartadas
        for elem in matched_lines:
            if elem[1] == "d(x) descartada": num_rejeitou += 1

    elif mode == "CHECKSUM":
        # pegando s   o valor do checksum na linha encontrada
        checksums = []
        for elem in matched_lines:
            checksums.append([int(x) for x in elem[1].split() if x.isdigit()][0])

        # quantos pacotes tiveram erro detectado
        for checksum in checksums:
            if checksum != 0: num_rejeitou += 1

    efficiency = (num_rejeitou / 50) * 100
    print("Descartadas: ", num_rejeitou, "de 50")
    print("Efici ncia = ", efficiency, "%")

    write_file(efficiency,mode)

if __name__ == "__main__":
    modes = ["CRC","CRC-8","CRC-10","CHECKSUM"]

    if len(sys.argv) < 2:
        print("Usage: python3 efficiency.py [mode]")
        print("[CRC,CRC-8,CRC-10,CHECKSUM]")
    else:
        mode = sys.argv[1]
        if mode not in modes:
            print("Modo digitado n o existe. Digite novamente.")
            print("Usage: python3 efficiency.py [mode]")
            print("[CRC,CRC-8,CRC-10,CHECKSUM]")
        else: read_file(mode)
```

APÊNDICE H
PLOT.PY

```
"""
Plota o gr fico de efici ncia de todos os modos/geradores
"""

import matplotlib.pyplot as plt
import numpy as np
import sys

def read_efficiency(mode):
    file = open("Output/eficiencias_{}.txt".format(mode),"r")
    content = file.readlines()
    content = [line.strip("\n") for line in content if line != "\n"] # remove \n
    values = [float(x) for x in content] # convert to float array
    return values

def plot():
    values_crc = read_efficiency("CRC")
    values_crc8 = read_efficiency("CRC-8")
    values_crc10 = read_efficiency("CRC-10")
    values_checksum = read_efficiency("CHECKSUM")

    # set width of bar
    barWidth = 0.25
    fig = plt.subplots(figsize =(12, 8))

    # set height of bar
```

```
    # [crc,crc8,crc10,checksum]
    generator = []
    for i in range(3): generator.append([values_crc[i],values_crc8[i],values_crc10[i],0])

    generator_checksum = [0,0,0,values_checksum[0]]

    # Set position of bar on X axis
    br1 = np.arange(len(generator[0]))
    br2 = [x + barWidth for x in br1]
    br3 = [x + barWidth for x in br2]
    br4 = [x + barWidth for x in br1]

    # Make the plot
    plt.bar(br1, generator[0], color ='tab:red', width = barWidth,
            edgecolor ='grey', label ='$Gerador$ $g(x)$ 1')
    plt.bar(br2, generator[1], color ='tab:green', width = barWidth,
            edgecolor ='grey', label ='$Gerador$ $g(x)$ 2')
    plt.bar(br3, generator[2], color ='tab:blue', width = barWidth,
            edgecolor ='grey', label ='$Gerador$ $g(x)$ 3')
    plt.bar(br4, generator_checksum, color ='tab:gray', width = barWidth,
            edgecolor ='grey', label ='$Checksum$')

    # Adding Xticks
    plt.xlabel('$Modo$', fontsize = 15)
    plt.ylabel('$Eficincia$ (%)', fontsize = 15)
    plt.xticks([r + barWidth for r in range(len(generator[0]))],
            ['$CRC$', '$CRC-8$', '$CRC-10$', '$Checksum$'])

    plt.legend()
    plt.savefig('Figuras/plot.png')
    plt.show()

if __name__ == "__main__":
    plot()
```

APÊNDICE I
READ_ARDUINO.PY

```
##############
## Script listens to serial port and writes contents into a file
##############
## requires pySerial to be installed
"""
Grava a sa da do Ardu no em um arquivo txt
"""
import serial
import sys

serial_port = '/dev/ttyACM0'
baud_rate = 9600 #In arduino, Serial.begin(baud_rate)
modes = ["CRC","CRC-8","CRC-10","CHECKSUM"]
name = ""

if len(sys.argv) < 2:
    print("Usage: python3 read_arduino.py [mode]")
    print("[CRC,CRC-8,CRC-10,CHECKSUM]")

else:
    if sys.argv[1] not in modes:
        print("Modo digitado n o existe. Digite novamente.")
        print("Usage: python3 read_arduino.py [mode]")
        print("[CRC,CRC-8,CRC-10,CHECKSUM]")
    else:
        mode = sys.argv[1]
        write_to_file_path = "Output/output_{}.txt".format(mode)

        print("MODE = " + mode)

        output_file = open(write_to_file_path, "w")
        ser = serial.Serial(serial_port, baud_rate)
        while True:
            line = ser.readline()
            line = line.decode("utf-8") #ser.readline returns a binary, convert to string
            print(line)
            output_file.write(line)
```