MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

DESENVOLVIMENTO PRELIMINAR DO PROGRAMA DE INTERFACE DO EXPERIMENTO ELISA/EQUARS

RELATÓRIO FINAL DE PROJETO DE INICIAÇÃO CIENTÍFICA

(PIBIC/CNPq/INPE)

Verônica Maria da Silva (FATEC SJC – Jessen Vidal, Bolsista PIBIC/CNPq)
E-mail: veronicka_ms@hotmail.com

Ing Hwie Tan (LAP/CTE/INPE, Orientadora)
E-mail: ing.tan@inpe.br

Julho de 2016

PUBLICADO POR:
Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/6921
Fax: (012) 3208-6919
E-mail: pubtc@sid.inpe.br

VERÔNICA MARIA DA SILVA

DESENVOLVIMENTO PRELIMINAR DO PROGRAMA DE INTERFACE DO EXPERIMENTO ELISA/EQUARS

Trabalho de Iniciação Científica apresentado ao Instituto Nacional de Pesquisas Espaciais (INPE).

Orientadora: Dra. Ing Hwie Tan

São José dos Campos
2016

Dados Internacionais de Catalogação na Publicação (CIP)

Instituto Nacional de Pesquisas Espaciais (INPE), Serviço de Informação e Documentação (SID).

Cutter **Desenvolvimento preliminar do Programa de Interface do Experimento ELISA/EQUARS/ . – São José dos Campos : INPE, Versão: 2016-02-29. xv + 103 p. ; (INPE-13269-MAN/45)**

1. Manual. 2. Estilo 3. LaTeX. 4. MSWord. 5. Normas I. Título.

CDU 000.000

AGRADECIMENTOS

Agradeço a minha família, especialmente a minha mãe Jorgina pela ajuda e esforço diário, não me deixando desistir. Agradeço aos meus amigos, em especial ao Marcel Ogawa, pela grande ajuda no LabVIEW, ao Rodrigo Takeshi e ao Marcos Hideki, que me ajudaram, incentivaram e me encorajaram a concluir este trabalho. Agradeço aos pesquisadores do INPE, especialmente a minha orientadora de Iniciação Científica, Ing Hwie Tan, ao meu supervisor de estágio, Júlio Guimarães Ferreira, e pela Primavera Botelho pelo apoio a pesquisa e a confiança. E agradeço também a todos os meus professores da FATEC, em especial ao Antônio Egydio e Emanuel Mineda pela ajuda e encorajamento.

RESUMO

Este trabalho tem como objetivo a elaboração de um programa para testar e validar o software a ser implantado no submódulo de interface do experimento ELISA (*Electrostatic Energy Analyser*). Estes testes de validação visam garantir o seu perfeito funcionamento após o lançamento. Os satélites operam em um ambiente agressivo, do ponto de vista da radiação ionizante, dos gradientes térmicos, do vácuo e das solicitações mecânicas no lançamento, exigindo assim que as entidades embarcadas sejam submetidas a um processo longo e complexo de qualificação, de modo que elas desempenhem suas funcionalidades com uma alta confiabilidade. Todo este cuidado deve ser aplicado também aos softwares embarcados. O controle e a aquisição de dados do experimento ELISA envolvem: 1) um submódulo de interface no qual um microprocessador é programado para controlar as diversas funções que o experimento deve realizar e no qual os dados obtidos são temporariamente armazenados; 2) o computador de bordo do satélite (OBC - OnBoard Computer) que controla os subsistemas da plataforma e se comunica com cada experimento, requisitando dados e enviando telecomandos através de uma interface serial RS422; e 3) e a estação terrestre, que recebe dados enviados pelo OBC e envia telecomandos ao satélite através de ondas de rádio. No trabalho foi realizado a comunicação serial entre um software desenvolvido em linguagem Python e um desenvolvido no LabVIEW. O desenvolvimento do programa na linguagem Python tem a função de simular as respostas que a interface do experimento ELISA envia ao computador de bordo, já o programa no LabVIEW tem a função de simular o computador de bordo ao receber mensagens de dados do experimento e enviar telecomandos. Os testes ocorrerem em dois computadores distintos, cada um contendo um programa, e a comunicação serial entre eles é realizada utilizando, conversores USB-Serial, adaptadores RS232/RS422 e conectores com pinagem cross-over específicas. A comunicação utiliza mensagens construídas seguindo o protocolo de comunicação estabelecido para a Plataforma Multimissão (PMM) do INPE.

Palavras-Chave: Python; LabVIEW; Experimento ELISA; Satélites; OBC; Comunicação Serial.

ELISA/EQUARS INTERFACE PROGRAM EXPERIMENT PRELIMINARY DEVELOPMENT

ABSTRACT

This work has as objective the elaboration of a program to test and validate the software that will be implemented on the interface sub-module of the ELISA (Electrostatic Energy Analyzer) experiment. These validation tests aim to assure the perfect operation after launch. Satellites work in an aggressive environment of ionized radiation, thermal gradients in vacuum and have to endure the mechanical stresses during launch, requiring a long and complex qualification process to assure high reliability in performing their mission. All this caution should also be applied to embedded softwares. The ELISA experiment control and data acquisition involves: 1) An interface sub-module in which a microprocessor is programmed to control the functionalities that the experiment should execute and temporarily store the measured data; 2) The satellite onboard computer (OBC) that controls the platform subsystems, and by using RS422 serial communication, notifies the experiments requesting data and sending commands; 3) a terrestrial station that receives the data sent by the OBC and sends to the satellite remote commands using radio waves. Using softwares developed in LabVIEW and Python languages, a serial communication was made. The software developed in Python Language simulates the responses that the ELISA experiment interface sends to the OBC. The LabVIEW software has the function of simulating the OBC when it receives data packages of the experiment and sends commands. Tests occur in two distinct computers, each one of them hosting one program, with the serial communication between the two programs occurring using USB-Serial converters, RS232/RS422 adaptors and connectors with the proper cross over pin connections. The communication message frames followed the protocol established for INPE´s Multi-Mission Platform (MMP).

Keywords: Python; LabVIEW; ELISA Experiment; Satellites; OBC; Serial Communication.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

OBC	*Onboard Computer*
ELISA	*Electrostatic Energy Analyser*
INPE	Instituto Nacional de Pesquisas Espaciais
GPIB	*General Purpose Interface Bus*
ACDH	*Attitude and Control Data Handling Subsystem*
HVS	*High Voltage Supply*
PLC	*Payload Computer*
GICs	Correntes Geomagnéticas Induzidas
AMAS	Anomalia Magnética do Atlântico Sul

SUMÁRIO

1- INTRODUÇÃO

Este capítulo visa explicar qual a motivação para a construção deste trabalho e seu objetivo.

1.1- Motivação

O Instituto Nacional de Pesquisas Espaciais (INPE) é uma instituição que busca produzir ciência e tecnologia nas áreas espacial e do meio ambiente, além de oferecer produtos e serviços singulares. Um de seus principais objetivos é o lançamento de satélites, como os satélites de coleta de dados SCD – 1 e 2, e de imageamento da terra CBERS – 1, 2, 2B e 4. Estão previstos também o lançamento de satélites científicos. Em particular a missão EQUARS tem como objetivo o monitoramento global da atmosfera na região equatorial, enfatizando processos dinâmicos, fotoquímicos, e mecanismos de transporte de energia entre a baixa, média e alta atmosfera e ionosfera (INPE, 2006). Um dos experimentos da missão EQUARS, objeto de estudo deste trabalho, é o ELISA (*Electrostatic Energy Analyser*), um analisador eletrostático de energias que medirá o espectro de energias de elétrons presentes na ionosfera equatorial terrestre (INPE, 2006), com especial ênfase na precipitação de elétrons na Anomalia Magnética do Atlântico Sul (AMAS). Este fenômeno resulta num aumento da ionização em camadas de menor altitude na ionosfera, afetando a propagação de ondas eletromagnéticas e por consequência vários aspectos da telecomunicação.

Para garantir o perfeito funcionamento do experimento ELISA, assim como para todos os subsistemas do satélite, o processo comumente referido como controle de qualidade ou garantia de qualidade é um requisito mandatório, pois o suporte e a manutenção em operação de vôo é zero. Além disso, os satélites operam em um ambiente agressivo, do ponto de vista da radiação ionizante, dos gradientes térmicos, do vácuo e das solicitações mecânicas no lançamento, exigindo assim que as entidades embarcadas sejam submetidas a um processo longo e complexo de qualificação, de modo que elas desempenhem suas funcionalidades com uma alta confiabilidade. Todo este cuidado deve ser aplicado também aos softwares embarcados (Fortescue,2011).

O controle e a aquisição de dados do experimento ELISA envolvem: 1) um submódulo de interface no qual um microprocessador é programado para controlar as diversas funções que o experimento deve realizar e no qual os dados obtidos são temporariamente

armazenados; 2) o computador de bordo do satélite (OBC - *OnBoard Computer*) que controla os subsistemas da plataforma e se comunica com cada experimento requisitando dados e enviando telecomandos através de uma interface serial RS422; e 3) e a estação terrestre, que recebe dados enviados pelo OBC e envia telecomandos ao satélite através de ondas de rádio.

1.2- Objetivo Geral

O objetivo deste trabalho é a elaboração de um programa para testar e validar o software a ser implantado no submódulo de interface do experimento ELISA, e com isso garantir o seu perfeito funcionamento após o lançamento. Como o circuito de interface ainda não foi construído um outro programa será elaborado para substituí-lo e testar a comunicação.

1.2.2- Objetivos Específicos

Os objetivos específicos desse trabalho incluem:

a) Desenvolver um programa no LabVIEW para simular o OBC ao receber mensagens de dados do experimento e enviar telecomandos;
b) Desenvolver um programa na linguagem Python para simular as respostas que a interface enviará ao OBC, ou seja, simular o envio de mensagens de dados, e o recebimento de telecomandos;
c) Seguir protocolo de comunicação, para comunicar, via cabo serial, os dois programas que serão implantados em dois notebooks distintos, como mostra a Figura 1.

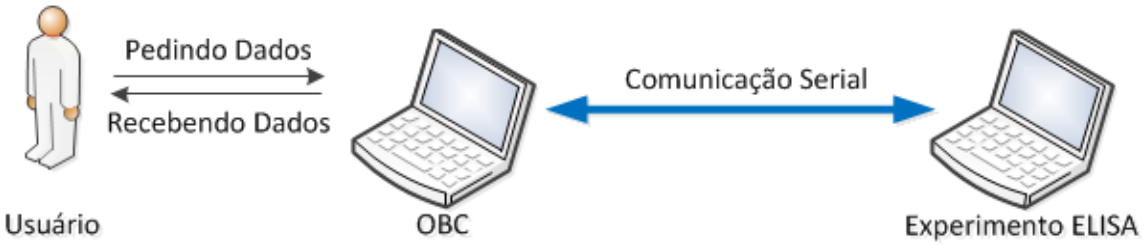


Figura 1 - Exemplo da comunicação entre os notebooks

Fonte: Produção da Autora (2016)

1.3- Organização do Trabalho

Este trabalho está organizado nos seguintes capítulos:

Capítulo 2: Fundamentação Teórica – descreve o funcionamento do experimento ELISA, os fundamentos, características e protocolos da comunicação serial, e as linguagens LabVIEW e Python que foram utilizadas.

Capítulo 3: Desenvolvimento – descreve os requisitos e formatação das mensagens de dados que devem ser geradas pelo experimento, a arquitetura dos dois programas elaborados e a montagem da comunicação entre eles.

Capítulo 4: Resultados e Discussão.

Capítulo 5: Conclusão.

Capítulo 6: Trabalhos Futuros.

2- FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a revisão bibliográfica feita a fim de fundamentar o trabalho.

A seção 2.1 apresenta uma introdução ao Experimento ELISA, seu princípio de funcionamento e como será seu comportamento no espaço. A seção 2.2 apresenta o conceito da comunicação de dados, os componentes necessários para uma comunicação, a direção do fluxo de dados, o que é comunicação serial e seus padrões RS232, RS422 e RS485. A seção 2.3 apresenta uma introdução ao protocolo de comunicação PMM e seu funcionamento. A seção 2.4 descreve o teste de validação e a seção 2.5 apresenta as tecnologias utilizadas para a criação de todo o sistema para teste.

2.1- Experimento ELISA

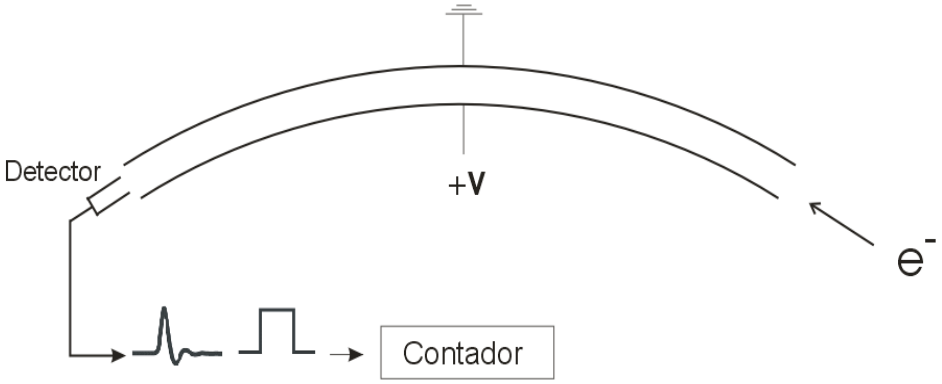
O experimento ELISA consiste em um analisador eletrostático de energias que deverá ser lançando no satélite científico EQUARS, a ser construído no INPE. Este experimento medirá o espectro de energias dos elétrons presentes na ionosfera equatorial terrestre, em especial na região da Anomalia Magnética do Atlântico Sul.

2.1.1- Princípio de Funcionamento do ELISA

O princípio de funcionamento do analisador eletrostático de energias se baseia nas trajetórias quase circulares que os elétrons incidentes na sua janela de entrada farão ao serem submetidos a um campo elétrico radial produzido por placas cilíndricas concêntricas nas quais se aplica uma diferença de tensão. Os elétrons que conseguem emergir desta região cilíndrica entre placas são coletados por um detector, que produz um pulso elétrico, que por sua vez será amplificado e contado eletronicamente por um contador. O número de pulsos (ou seja, de elétrons) é contado durante um intervalo de tempo, ao fim do qual esse número é armazenado, a tensão entre as placas (que seleciona a energia que será coletada) é variada, e volta-se a contar os pulsos em outra energia durante o mesmo período de tempo, e assim sucessivamente, se obtém o espectro da intensidade do fluxo de elétrons em 16 faixas de energia. Obtido um espectro, a tensão entre placas retorna ao valor inicial, e o processo se repete indefinidamente (ICD, 2015).

A Figura 2 mostra o esquema de funcionamento do analisador eletrostático.

Figura 2 - Esquema de funcionamento do analisador eletrostático

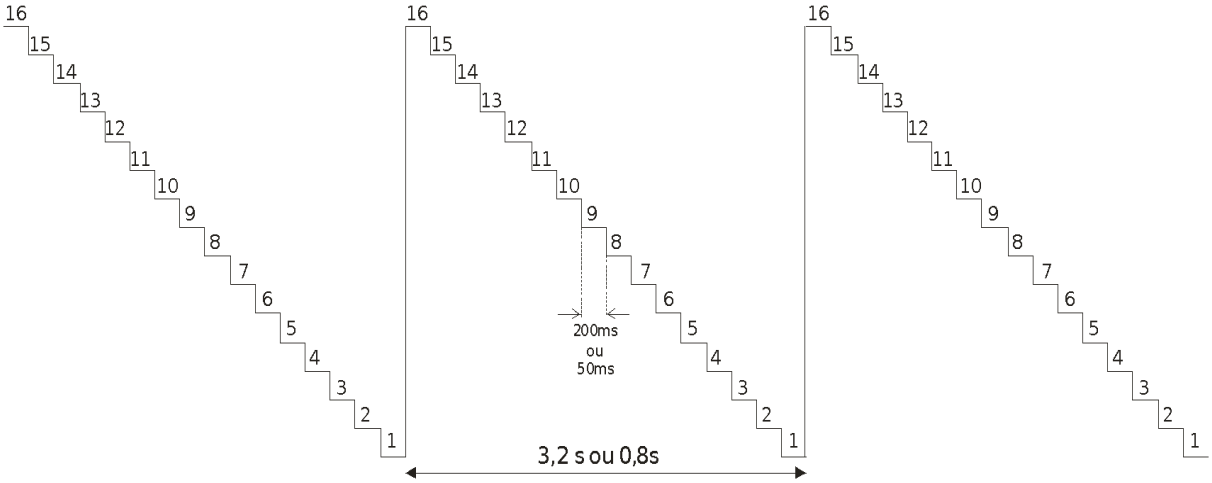


Fonte: TAN (2015)

O experimento ELISA produz então, uma série de números (contagens) que têm de ser armazenados, identificados, e periodicamente têm de ser transferidos ao computador de bordo do satélite que se encarregará de transmitir os dados aos centros de recepção em terra. Além das contagens, outras grandezas de monitoramento devem ser periodicamente medidas, armazenadas e transmitidas: a tensão entre placas, a tensão de alimentação do detector, e os valores medidos pelo detector durante calibrações periódicas. Mudanças no modo de operação (regular ou modo de monitoração/calibração) serão feitas automaticamente, ao passo que mudanças no tempo de varredura das tensões nas placas poderão ser eventualmente feitas por telecomando (ICD, 2015).

A Figura 3 mostra o esquema temporal da varredura de 16 níveis de tensão entre as placas (os níveis de tensão na realidade decaem exponencialmente, a figura mostra decaimento linear por simplicidade).

Figura 3 - Esquema temporal da varredura



Fonte: TAN (2015)

Todo o funcionamento do experimento e sua interação com o computador de bordo é controlado por um módulo de interface, composto por um microcontrolador com diversos circuitos auxiliares (memórias, conversores ADC e outros). O experimento terá dois analisadores que serão embarcados (com direções de visada perpendiculares), e cada um deles produzirá números de contagens que ocuparão 3 bytes de 8 bits (chegando à contagem máxima de aproximadamente $1{,}7 \times 10^7$ contagens) (TAN, 2011).

2.1.2- Comportamento do ELISA no Espaço

Ao ser lançado o satélite com os experimentos a bordo, todos devem permanecer desligados até que o computador de bordo saia do estado de *standby*.

Para ligar o experimento liga-se seu submódulo de eletrônica. A seguir checa-se o tempo de varredura que pode ou não ser alterado por um telecomando. O tempo de varredura é gravado na mensagem de dados.

No modo de monitoramento, o amplificador é ligado ao gerador de pulsos e são medidas as tensões de varredura das placas e a tensão de alimentação do detector *Channeltron*.

Existem 2 analisadores de elétrons, os quais chamamos de X e Y, que medem os espectros de energia de elétrons que incidem com direção paralela e perpendicular ao campo magnético terrestre.

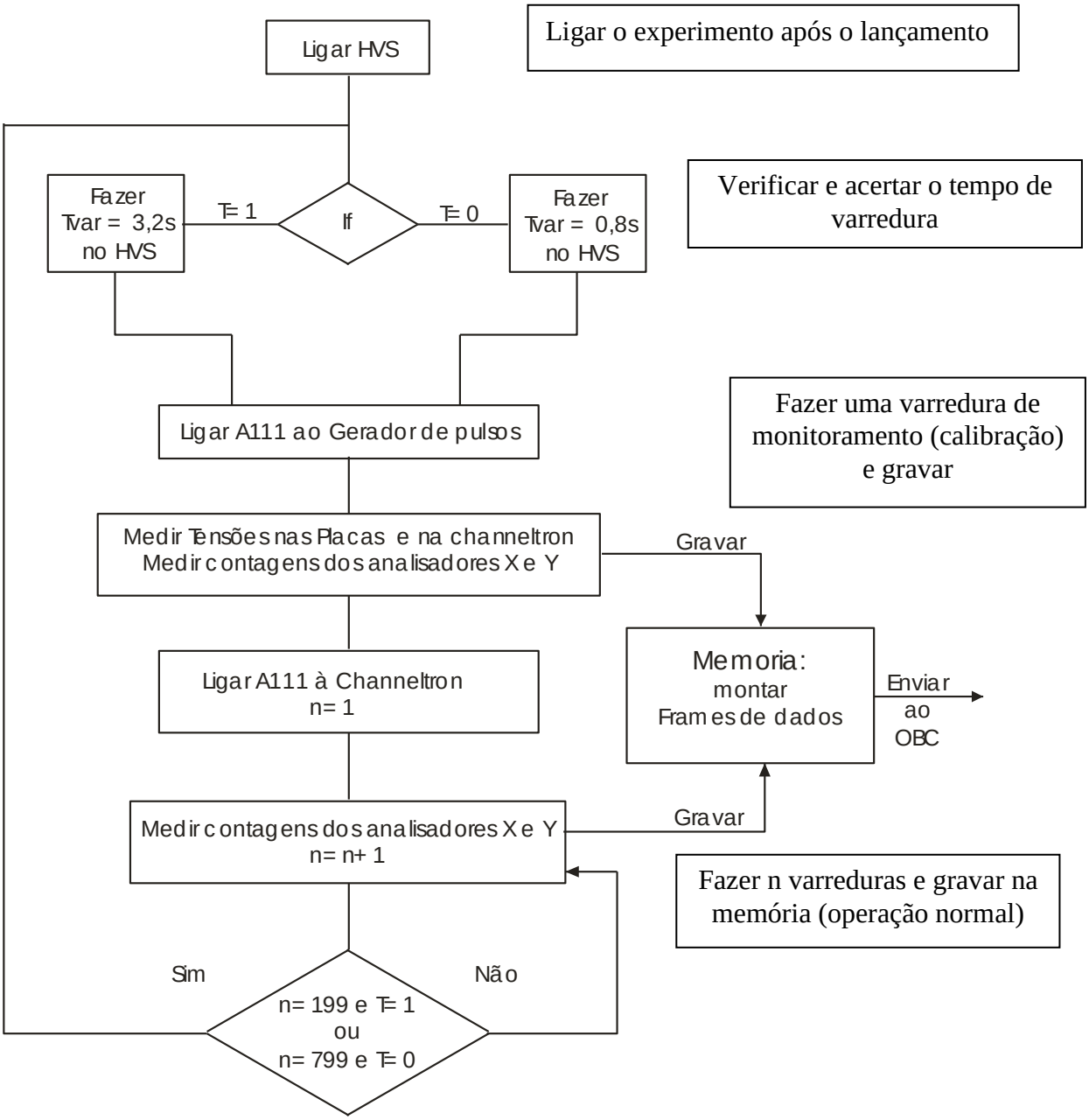
As contagens dos analisadores X e Y também são medidas e gravadas. Como o amplificador está ligado ao gerador de pulsos com frequência conhecida, neste modo de monitoramento, todas as contagens são iguais e de acordo com a frequência do gerador.

Após essa varredura de monitoramento o amplificador é então ligado à *Channeltron* e iniciam-se N varreduras dos analisadores X e Y, onde as contagens medidas para cada valor de tensão nas placas (são 16 medidas por varredura para cada analisador) são armazenadas temporariamente na memória para serem eventualmente enviados ao OBC. Após o fim da varredura N, volta-se para o início do processo, onde a interface pode ou não receber um telecomando com mudança no tempo de varredura. O processo é repetido indefinidamente.

Neste modo de operação não é necessário monitoramento contínuo das tensões nas placas, basta marcar o início da varredura para saber em qual ponto corresponde a leitura. As medidas são armazenadas temporariamente até serem transferidas ao Computador de Bordo (ICD, 2015).

Abaixo, na Figura 4 está o diagrama de blocos com o fluxo de funcionamento do experimento.

Figura 4 - Diagrama de blocos da operação do experimento ELISA



Fonte: TAN (2015)

A evolução temporal dos eventos descritos nesse diagrama de blocos pode ser vista na seção 3.2.2.

2.2- Comunicação de Dados

A comunicação de dados é a troca de dados entre dispositivos por intermédio de algum tipo de meio de transmissão, como um cabo condutor formado por fios. Para que a comunicação de dados ocorra, os dispositivos devem fazer parte de um sistema de comunicação, composto da combinação de *hardware* (equipamentos físicos) e *software* (programas) (FOROUZAN, 2008).

Para que um sistema de comunicação seja eficiente, é fundamental que tenha as seguintes características:

1. **Entrega (delivery)**: os dados devem ser entregues no destino correto e devem ser recebidos somente pelo dispositivo ou usuário destinado.
2. **Confiabilidade/Precisão**: a entrega dos dados deve ser garantida pelo sistema. São pouco úteis dados que são modificados ou corrompidos durante uma transmissão.
3. **Tempo de atraso/Sincronização**: os dados devem ser entregues pelo sistema em um tempo finito e predeterminado. No caso de transmissões multimídia, como vídeo e áudio, os dados devem ser entregues praticamente no mesmo instante em que foram produzidos, isto é, sem atrasos significativos.
4. ***Jitter***: refere-se à variação do tempo de chegada das mensagens. É o atraso desigual na entrega das mensagens de áudio e vídeo.

2.2.1- Componentes

Um sistema básico de comunicação de dados é composto pelos seguintes elementos (FOROUZAN, 2008):

1. **Mensagem:** é a informação a ser transmitida. Podem ser números, figuras, áudio, vídeo, etc.;
2. **Transmissor:** é o dispositivo que envia a mensagem de dados. Pode ser um computador, um rádio, um telefone, etc.;
3. **Receptor:** é o dispositivo que recebe a mensagem. Pode ser um computador, um rádio, um telefone, etc.;
4. **Meio:** é o caminho físico por onde viaja a mensagem da origem até o receptor;
5. **Protocolo:** é um conjunto de regras que governa a comunicação de dados. Representa um acordo entre os dispositivos que se comunicam. Sem um protocolo, dois dispositivos podem estar conectados, mas, sem se comunicar.

Esse acordo entre o emissor e o receptor se baseia em (MAZIDI, 2009):

- Como os dados são embalados;
- Quantos bits constituem uma mensagem;
- Quando os dados começam e terminam.

2.2.2- Direção do Fluxo de Dados

Uma comunicação entre dois dispositivos pode acontecer de três maneiras diferentes (FOROUZAN, 2008):

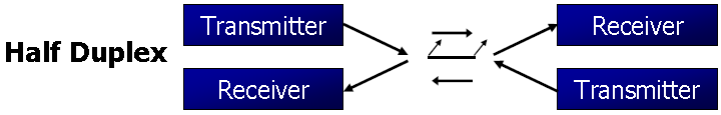
- ***Simplex***: a comunicação é unidirecional, como em uma rua de mão única. Somente um dos dispositivos é capaz de transmitir e o outro só receber, como mostra a Figura 5.

Figura 5 – Mostrando o *Simplex*

Fonte: MAZIDI (2009)

- ***Half-Duplex***: cada dispositivo pode transmitir e receber, mas nunca ao mesmo tempo. Enquanto um transmite o outro recebe e vice-versa, como mostra a Figura 6. Toda a capacidade do canal é dada ao dispositivo que estiver transmitindo no momento.

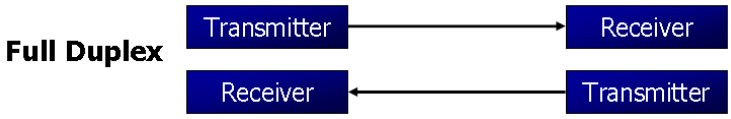
Figura 6 - Mostrando o *Half Duplex*



Fonte: MAZIDI (2009)

- ***Full-duplex***: ambos dispositivos podem transmitir e receber simultaneamente, como mostra a Figura 7. A capacidade do canal deve ser dividida em ambas as direções.

Figura 7 - Mostrando o *Full Duplex*



Fonte: MAZIDI (2009)

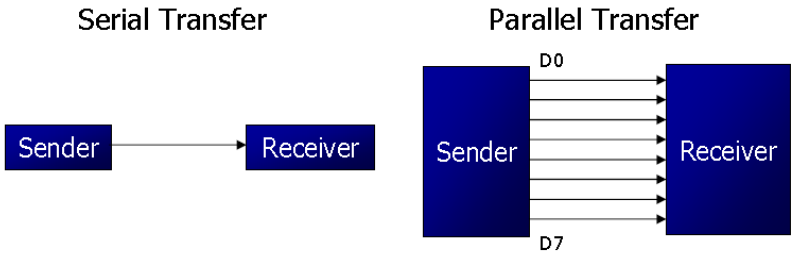
2.2.3- Comunicação Serial

A transferência de dados pode ser feita de duas maneiras (MAZIDI, 2009):

- **Paralela**: oito ou mais fios condutores são utilizados para transferir dados para outro dispositivo em poucos metros de distância;
- **Serial**: para transmitir dados para outro dispositivo em longa distância, é utilizado o método serial, o qual os dados são enviados um bit por vez.

A Figura 8 mostra o envio de dados em cada método.

Figura 8 - Envio de dados em Serial e Paralelo



Fonte: MAZIDI (2009)

A comunicação de dados seriais usa dois métodos (MAZIDI, 2009):

- **Método Síncrono**: o emissor e o receptor devem estar num estado de sincronia durante a transmissão;
- **Método Assíncrona**: transmite dados que podem ser transmitidos intermitentemente, não contínuos, em um fluxo estável.

A comunicação serial é realizada usando 3 linhas de transmissão: terra, transmissão, recepção. E podem ser empregados os três sistemas de comunicação: *Simplex, Half-Duplex, Full-Duplex* (NATIONAL INSTRUMENTS, 2015).

2.2.3.1- Características Importantes

Para duas portas de comunicação, são necessários (NATIONAL INSTRUMENTS, 2015):

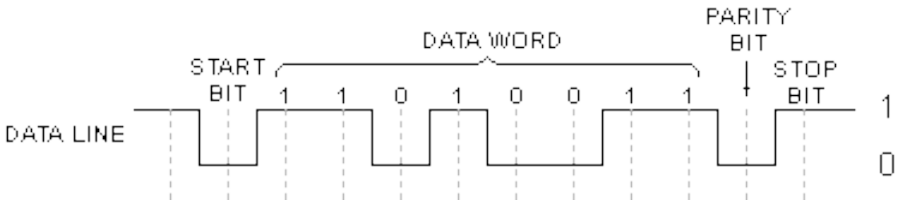
- **Taxa de Transmissão (*Baud Rate*):** medida de velocidade para comunicação. Indica o número de bits transmitidos por segundo. Por exemplo, 200 *baud rate* são 200 bits por segundo;
- **Bit de Início (*Start Bit*):** A linha de dados tem dois estados, 0 ou 1 (desligado e ligado, respectivamente). Uma linha de dados está sempre em 1. Quando o instrumento ou computador quer enviar dados que define a linha para 0 - este é o bit de início. Os bits após o bit de início são, portanto, os bits de dados;
- **Bit de Dado: (*Data Bit*):** é uma medida dos bits atuais de uma transmissão;
- **Bits de Parada (*Stop Bit*):** sinaliza o fim da comunicação para uma única mensagem;
- **Paridade:** uma forma simples de verificação de erro. Existem quatro tipos de paridade: par, impar, marcada e espaçada.
    - **Par e ímpar:** a porta serial irá definir o bit de paridade (o ultimo bit depois dos bits de dados) para um valor que garanta que a transmissão tenha um número par ou ímpar de bits de lógica alta. Exemplo: se o dado for 011, para a paridade par o bit será 0 para manter o número de bits de lógica alta par. Se a paridade for ímpar, o bit de paridade será 1, resultando em 3 bits de lógica alta.
    - **Marcada e espaçada:** não verificam os bits de dados, mas definem o bit de paridade alto para paridade marcada ou baixo para paridade espaçada. Isso permite ao dispositivo receptor saber o estado de um bit e descobrir se um ruído está corrompendo o dado ou se os *clocks* do transmissor receptor estão dessincronizados.

Os dados a serem transmitidos ou recebidos devem seguir a devida ordem (SILVA, et. Al, 2013):

- Bit de Início
- Bits de Dados
- Bit de Paridade
- Bits de Parada

A Figura 9 é um exemplo de um sinal de envio/recebimento.

Figura 9 - Sinal de envio/recebimento



Fonte: ELETRONICA (1999)

2.2.4- Cabo Serial *Cross-over*

São usados para conectar dispositivos DTE para DTE, ou DCE para DCE (como definidos abaixo). O termo cabo *null-modem* é muitas vezes utilizado para referir cabo *cross-over*, mas um cabo *null-modem* é só uma variedade de cabos *cross-over* (ETUTORIALS, 2016).

- **DTE** - termo para *Data Terminal Equipment*, equipamento de terminação de dados, é o equipamento onde os dados terminam e onde podem ser iniciados. DTEs geram e recebem os dados (SNNANGOLA, 2010).
- **DCE** - termo para *Data Communications Equipment*, equipamento, responsável pela comunicação de dados. Realiza algumas tarefas importantes na transmissão de dados entre dois dispositivos como determinar a frequência de *clock*, a determinação dos erros de transmissão e a codificação, etc, ou seja, como se envia e se recebem os dados (SNNANGOLA, 2010).

Para que haja uma conexão, é necessário que se conecte um DTE com um DCE ou que use um cabo *cross-over* entre dois dispositivos DTE.

As portas seriais do computador geralmente são configuradas como DTE; modens, mouse, scanners, etc, são geralmente dispositivos DCE, por isso se conecta a um PC com um cabo direto. Quando se conecta dois PCs DTE, é necessário um cabo *cross-over* (ETUTORIALS, 2016).

2.2.5- Padrões RS

RS, sigla para *Recommended Standard* (padrão recomendado), é um padrão de comunicação serial, desenvolvido pela EIA (Electronics Industry Association), por isso o padrão também é denominado EIA. A EIA é desenvolvedora dos padrões de comunicação: RS-232(EIA-232), RS-422(EIA-422) e RS-485(EIA-422) (SILVA, et. al, 2013).

2.2.5.1- RS-232

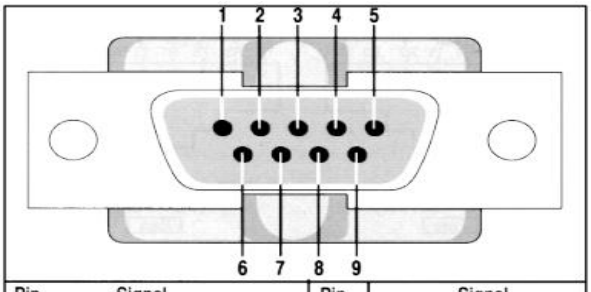
O padrão de comunicação RS-232 foi criado em 1962 para a comunicação entre uma máquina de escrever eletromecânica e um modem. Em 1969, após algumas revisões, o protocolo RS-232 se tornou mais popular para a utilização em dispositivos de comunicação, onde se tornou o padrão de comunicação serial mais utilizado até o fim dos anos 90, quando começou a ser substituído pelo USB (*Universal Serial Bus*) (SILVA, et. Al, 2013).

Apesar da alta velocidade e praticidade proporcionada pelo USB, a robustez do RS-232 faz com que ele ainda seja muito utilizado na indústria, medicina e em aplicações mais específicas, como impressoras fiscais, leitores de códigos de barra, GPS's e modems de satélite (SILVA, et. al, 2013).

O padrão especifica dois tipos de conectores: DB-9 e DB-25, de nove e vinte e cinco pinos, respectivamente, onde nem todos os pinos são atribuídos a sinais.

As Figuras 10 e 11 mostram a pinagem de cada conector.

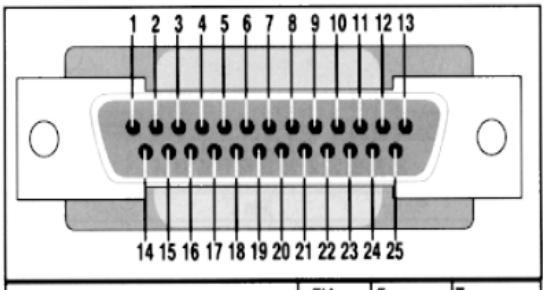
Figura 10 - Pinagem do conector DB-9



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Fonte: SILVA, et. al (2013)

Figura 11 - Pinagem do conector DB-25



Pin	Description	EIA CKT	From DCE	To DCE
1	Frame Ground	AA		
2	Transmitted Data	BA		D (Data)
3	Received Data	BB	D	
4	Request to Send	CA		C (Control)
5	Clear to Send	CB	C	
6	Data Set Ready	CC	C	
7	Signal Gnd/Common Return	AB		
8	Rcvd. Line Signal Detector	CF	C	
11	Undefined			
12	Secondary Rcvd. Line Sig. Detector	SCF	C	
13	Secondary Clear to Send	SCB	C	
14	Secondary Transmitted Data	SBA		D
15	Transmitter Sig. Element Timing	DB	T (Timing)	
16	Secondary Received Data	SBB	D	
17	Receiver Sig. Element Timing	DD	T	
18	Undefined			
19	Secondary Request to Send	SCA		C
20	Data Terminal Ready	CD		C
21	Sig. Quality Detector	CG		C
22	Ring Indicator	CE	C	
23	Data Sig. Rate Selector (DCE)	CI	C	
23	Data Sig. Rate Selector (DTE)	CH		C
24	Transmitter Sig. Element Timing	DA		T
25	Undefined			

Fonte: SILVA, et. al (2013)

A Tabela 1 apresenta os sinais utilizados e sua função.

Tabela 1 - Sinais utilizados nos conectores para a comunicação

Sinal	Descrição
Frame Ground	Terra da carcaça
Transmitted Data	Dados transmitidos pelo DTE
Received Data	Dados recebidos pelo DTE
Request to Send (RTS)	Pedido do DTE para início da transmissão pelo DCE
Clear to Send	Resposta do DCE ao sinal RTS
Data Set Ready (DSR)	Indicador de funcionamento (Power On)
Signal Ground / Common Return	Terra para Sinal (Referência para DTE e DCE)
Received Line Signal Detector (DCD)	Indica o estado da linha de recepção
Transmitter Signal Element Timing	*Clock* do Transmissor

Receiver Signal Element Timing	*Clock* do Receptor
Data Terminal Ready (DTR)	DTE pronto para receber dados
Signal Quality Detector	Qualidade do Sinal
Ring Indicator	Indicador de Chamada (e.g. Modem)
Data Signal Rate Selector	Seleção de taxa de transmissão

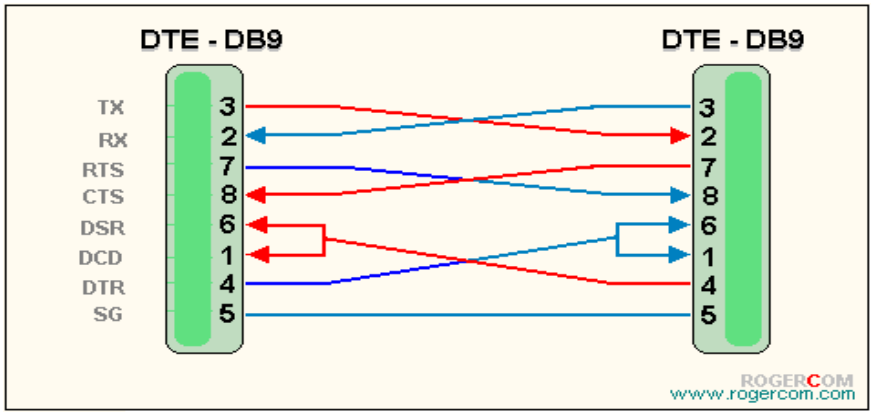
Fonte: SILVA, et. al (2013)

O mais utilizado ultimamente é o conector D-B9 em sua configuração mais simples, chamada 3-wire, com apenas os pinos de transmissão (Transmitted Data), recepção (Received Data) e Terra (Signal Ground), sendo feito o controle de fluxo de dados, sincronização e sinalização através de software. Outra configuração popular é a 5-wire, onde, além dos 3 fios anteriores, *Transmitted Data*, *Received Data* e *Signal Ground*, utiliza-se do DSR (Indicador de funcionamento) e RTS (Pedido do DTE para início da transmissão pelo DCE) para controle do fluxo de dados.

Um cuidado a ser tomado ao montar um conector, é saber que o pino de recepção do DCE será conectado ao pino de transmissão do DTE, no caso do DB-9, o pino 2 deve ir no 3, e vice-versa, fazendo com que os fios de transmissão e recepção sejam invertidos. Apesar de básico, este é uma das maiores causas de falha de comunicação de usuários amadores (SILVA, et. al, 2013).

A Figura 12 mostra como os pinos devem ser conectados para a transmissão entre dois computadores.

Figura 12 - Conexão dos pinos para uma comunicação entre dois computadores



Fonte: ROGERCOM (2006)

2.2.5.2- RS-485

O RS-485 é um padrão de comunicação multiponto, permitindo-se a conexão de até 32 dispositivos num simples cabo de par trançado. Ele é baseado na transmissão diferencial de dados, através de um par de fios de par trançado com resistores de terminação para balanceamento. Dessa forma, se um ruído é introduzido na linha, ele é induzido nos dois fios de modo que a diferença entre A e B é quase nula. Outra vantagem da transmissão diferencial é que diferentes potenciais de terra são, até certo ponto, ignorados pelos transmissores e receptores. O protocolo RS-485 é ideal para transmissão em altas velocidades, longas distâncias e em ambientes propícios a interferência eletromagnética. Ele permite a comunicação entre vários elementos participantes em uma mesma rede de dados e é uma melhoria do RS-422 (SILVA, et. al, 2013).

Funciona da seguinte forma a transmissão diferencial: qualquer transmissor RS-485 possui dois canais independentes conhecidos como A e B, que transmitem níveis iguais de tensão, mas com polaridades opostas (VOA e VOB ou simplesmente VA e VB). É importante que a rede seja ligada com a polaridade correta. Embora os sinais sejam opostos, não existe um loop de corrente, ou seja, um não é retorno do outro. Cada sinal tem seu retorno pela terra ou por um terceiro condutor de retorno, porém o sinal deve ser lido pelo receptor de forma diferencial sem referência à terra ou ao condutor de retorno. Este sinal diferencial, lido em relação ao ponto central da carga, é que é interpretado como sinal de transmissão. Se a diferença for superior a 200mV, então tem-se nível lógico 1. Caso a diferença seja inferior a -200mV, então considera-se nível lógico 0. No intervalo de -200mV a 200mV o nível lógico é indefinido e interpretados como ruído (SILVA, et. al, 2013).

2.2.5.3- RS-422

O RS-422 é uma conexão serial que utiliza um sinal elétrico diferencial como a do RS-485, diferente dos sinais desbalanceados referenciados ao fio terra com o RS-232. A transmissão diferencial utiliza duas linhas para cada sinal transmitido ou recebido, resulta em maior imunidade a ruído e maiores distâncias comparada ao RS-232 e é capaz de interligar um dispositivo transmissor a até 10 receptores.

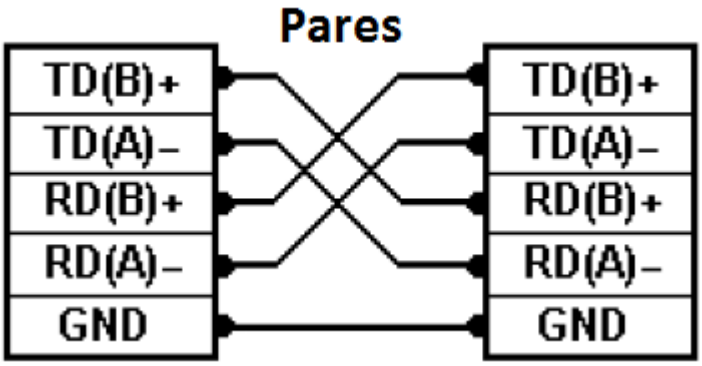
Ele utiliza dois pares trançados, sendo um utilizado no sentido do transmissor para os receptores e o outro no sentido contrário. A utilização de dois pares trançados permite que ocorra ao mesmo tempo transmissão entre o transmissor e o receptor, a possibilidade de

transmissão e recepção simultânea caracteriza o RS-422 como *full-duplex* (SILVA, et. al, 2013).

O padrão não possui um conector definido para uso e é basicamente uma extensão do RS-232 com sinal diferencial para transmissão e recepção de dados (SILVA, et. al, 2013).

A Figura 13 mostra como os pinos devem ser conectados em uma comunicação entre dois computadores. Os números de pinos para as ligações RS-422 podem variar de acordo com o modelo.

Figura 13 - Pinagem RS-422



Fonte: B&BELETRONICS (2003)

A pinagem do RS-422 e sua descrição é mostrada na Tabela 2.

Tabela 2 - Descrição da pinagem RS-422

Sinal	Descrição
T+	*Transmit Data* +
T-	*Transmit Data* -
R-	*Request To Send* -
R+	*Request To Send* +
GND	*Ground*

Fonte: SERIALCOMM

2.2.5.4- Adaptadores

O protocolo de comunicação RS vem sendo substituído pelo USB em pequenas distâncias, por este motivo vários computadores, principalmente notebooks, não são mais produzidos com entradas para os conectores DB utilizados em comunicação serial, somente computadores específicos. O protocolo USB é mais rápido, possui conectores mais simples de usar e tem um melhor suporte por software (*Plug & Play*). Mesmo com estas vantagens, o protocolo RS ainda continua sendo utilizado em periféricos para pontos de venda (caixas registradoras, leitores de códigos de barra ou fita magnética) e para a área industrial (dispositivos de controle remoto, medidores), já que possui alta confiabilidade e fácil implementação (SILVA, et. al, 2013).

Como alternativa para o uso do protocolo RS, existem adaptadores para portas USB, como mostra a Figura 14, que podem ser utilizados para conectar teclados ou mouses PS/2, uma ou mais portas seriais e uma ou mais portas paralelas (SILVA, et. al, 2013).

Figura 14 - Adaptador USB Serial

Fonte: TRENDNET

Para utilização do protocolo serial RS422 foi utilizado um outro adaptador que converte o sinal serial RS232 para o padrão RS422. Para que a comunicação aconteça é necessário um conector *cross-over* de maneira que os pinos de transmissão sejam conectados aos pinos de recepção.

Os pinos conectados para a comunicação no RS-422 são mostrados na Tabela 3.

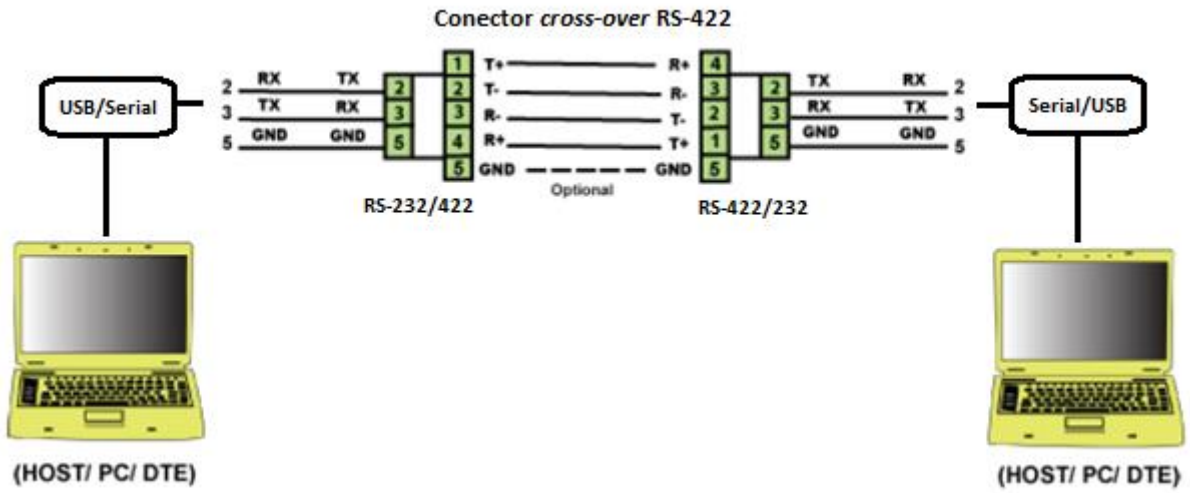
Tabela 3 - Conexão dos pinos do RS-422

Pino	Sinal	Sinal	Pino
1	T+	R+	4
2	T-	R-	3
3	R-	T-	2
4	R+	T+	1
5	GND	GND	5

Fonte: SERIALCOMM

A Figura 15 mostra a conexão entre dois computadores utilizando o RS-422.

Figura 15 - Conexão entre dois computadores utilizando o conector cross-over RS-422



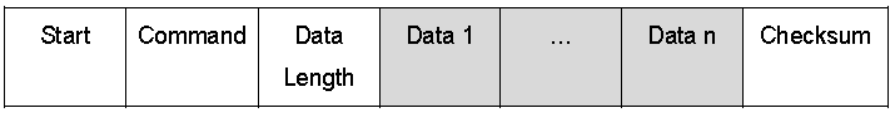
Fonte: SERIALCOMM / Modificação da Autora (2016)

2.3- Protocolo de Comunicação

O protocolo de comunicação a ser seguido no satélite EQUARS, no qual será lançado o experimento ELISA, ainda não foi definido. Por esse motivo o protocolo a ser seguido nesse trabalho obedece às regras estabelecidas para a PMM (Plataforma Multimissão), uma plataforma desenvolvida pelo INPE (PMM, 2001).

Como descrito no documento da PMM (PMM, 2001), a mensagem deve ser composta de um cabeçalho (*header*), uma pacote de dados (*data packet*) e uma palavra de verificação (*checksum*). O cabeçalho é composto de um Início (*Start*), um comando (*Command*), e um comprimento de dados (*Data Length*). A mensagem de dados é composta de até 255 dados de 8 bits. Todo o esquema da mensagem é mostrado na Figura 16.

Figura 16 - Esquema do formato da mensagem



Fonte: PMM (2001)

A mensagem é composta de:

- Início (*Start*) (2 bytes): EB90 (número hexadecimal que representa o experimento);
- Comando (*Command*) (1 byte);
- *Data Lenght* (1 byte contendo o tamanho de uma parte da mensagem em bytes, excluindo o cabeçalho e a soma de verificação (*checksum*), 0 a 255).
- Dado 1 (1 byte);
- Dados até 255 (1 byte);
- Dado de confirmação (*Checksum*) (2 bytes).

O protocolo dever ser do tipo mestre-escravo (*master-slave*), onde o computador de bordo deve ser o mestre e os diversos experimentos como o ELISA, o escravo (PMM, 2001).

2.3.2- Tipo de Mensagem

Os tipos de mensagem a serem enviados do OBC para o experimento devem ter, ao menos, os seguintes comandos e sua representação em hexadecimal:

- *Data request* (Pedido de dados): 01H
- *Data send* (Dado enviado): 02H
- Comandos específicos (comandos a ser suportados pelo equipamento)

Os tipos mensagens do experimento para o OBC podem ser:

- *Acknowledge* (Reconhecido), deve ser enviado ao OBC em resposta a mensagem de comando: 03H
- *Data message* (Mensagem de dado): 04H
- *Error message* (Mensagem de erro): 06H

O cálculo do checksum utilizado neste trabalho é feito da seguinte maneira:

- Somam-se os valores numéricos em cada byte = S
- Divide-se S por 256 e arredonda-se esse valor = D
- **Checksum** = S - 256×D

Nem todos os comandos acima são obrigatórios em uma comunicação.

2.4- Testes de Validação de Experimento

Antes de um satélite ser lançado junto com seus experimentos, todos os subsistemas precisam passar por testes exaustivos, tanto em hardware como em software, para que não haja erros ao serem lançados ou quando estiverem em órbita. Este teste de validação é uma das funções dos equipamentos de suporte em solo EGSE (*Eletronic Ground Support Equipment*), o qual é o equipamento de suporte usado para fornecer serviços de apoio e suporte operacional para vários satélites e veículos (ESC AEROSPACE).

Um dos serviços do EGSE é testar funções do computador de bordo, utilizando determinadas ferramentas para a construção do mesmo.

Um dos programas muito utilizados para a construção do OBC é o LabVIEW, o qual fornece funções e serviços que atendem as necessidades para a realização do teste.

A comunicação entre o OBC e o circuito de interface do experimento será feita através de comunicação serial. Para fazer o teste de validação do circuito de interface do experimento independentemente do OBC, essa comunicação serial será simulada implementando dois programas em dispositivos distintos (laptops) ligados por um cabo serial RS422. Em um dos laptops será implementado um programa em LabVIEW simulando o OBC, e no outro laptop será implementado um programa em Python simulando o circuito de interface.

Como tipicamente os laptops não possuem conectores para RS422, dois adaptadores USB/RS232 e RS232/RS422 foram adquiridos.

2.5- Tecnologias Utilizadas

Neste projeto serão usados o LabVIEW e a linguagem Python para o desenvolvimento da aplicação.

2.5.1- Python

Python é uma linguagem criada para produzir código bom e fácil de maneira rápida. Ressaltam os seguintes objetivos da linguagem (PYSCIENCE, 2008):

- Baixo uso de caracteres especiais, o que torna muito parecida com pseudocódigo executável;
- Uso de endentação para marcar blocos;
- Quase nenhum uso de palavras-chave voltadas para compilação;
- etc.

Python suporta múltiplos paradigmas de programação. A programação procedimental pode ser usada para programas simples e rápidos, mas estruturas de dados complexas, como tuplas, listas e dicionários, estão disponíveis para facilitar o desenvolvimento de algoritmos complexos. Python suporta técnicas de orientação objeto, inclusive sobrecarga de operadores e herança múltipla (PYSCIENCE, 2008).

Possui uma imensa biblioteca padrão (*Python Package Index* (PyPI)) que contém classes, métodos e funções para a realização de diversas tarefas. Ambas bibliotecas padrão do Python e os módulos contribuídos pela comunidade permitem infinitas possibilidades (PYTHON, 2016).

Python é uma linguagem livre e multiplataforma, ou seja, programas escritos rodam na maioria das plataformas existentes sem nenhuma modificação.

2.5.1.1- PySerial

O site oficial do pySerial (2015) diz que, o pySerial é um módulo em Python que encapsula o acesso para a porta serial. Ele fornece *back-ends* (tudo que é necessário para utilizar) para Python rodando no Windows, OSX, Linux, BSD (possivelmente qualquer

sistema compatível com POSIX) e IronPython. O módulo chamado "serial" seleciona automaticamente o *back-end* apropriado. É uma biblioteca de fácil instalação e utilização.

Características segundo o site:

- A interface baseia a mesma classe em todas as plataformas;
- Acesso às configurações de porta através de propriedades Python;
- Suporte para diferentes tamanhos de byte, bits de parada, paridade e controle de fluxo;
- Trabalha com ou sem receber timeout;
- Arquivo como API com "*read*" (ler) e "*write*" (escrever) ("*readline*" (ler linha) etc. também suportado);
- Os arquivos 100% Python;
- A porta está configurada para transmissão de binário. Sem remoção de NULL byte, tradução para CR-LF etc. (que são muitas vezes habilitados para POSIX.) Isso faz com que este módulo seja universalmente útil.
- Compatível com biblioteca de I/O;
- RFC 2217 cliente (experimental), servidor fornecido nos exemplos.

Requerimento:

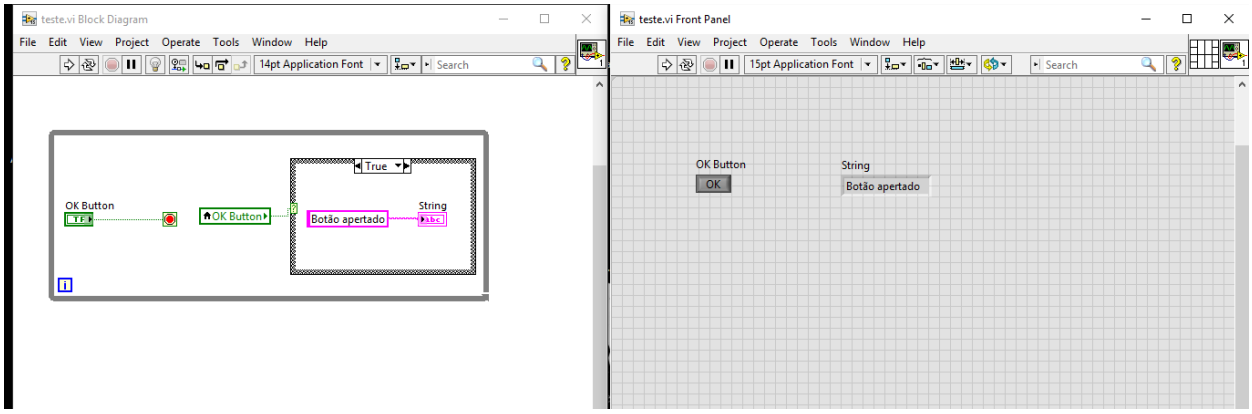
- Python 2.7 ou mais recente, incluindo Python 3.4 e mais recentes;
- "*Java Communications*" (JavaComm) ou uma extensão compatível com Java/ Jython.

2.5.2- LabVIEW

Segundo Regazzi et al. (2005), o LabVIEW é uma plataforma de programação da NI - National Instruments poderosa e flexível, que permite ser executada em vários sistemas operacionais, tais como Windows, MacOs, Linux, etc. É uma linguagem intuitiva e fácil, não necessitando de grandes conhecimentos em programação. Permite a interação com outras ferramentas matemáticas, de simulação, controle e supervisão, utilizadas em processos simples ou de grande porte.

O LabVIEW permite a simulação de instrumentos e equipamentos, tornando a tela do computador um instrumento virtual com a mesma aparência do equipamento ou desenho utilizado no modelo. Os painéis do programa são criados em ambientes gráficos com objetos, que são ferramentas de desenho. Estes objetos são interligados como em um fluxograma, sem a necessidade de escrever linhas de código, como mostra a Figura 17. Depois de compilado no sistema, o programa pode rodar em vários dispositivos (REGAZZI, et al., 2005).

Figura 17 - Exemplo do uso do LabVIEW



Fonte: Produção da Autora (2016)

2.5.3- MVC – Model – View - Controller

O padrão de arquitetura de *software* utilizado no programa em Python foi o MVC, que separa sua aplicação em três camadas (RAMOS, 2015):

- **Model:** Manipulação dos dados. Responsável pela validação dos dados e pela sua leitura e escrita;
- **View:** interação com o usuário. Exibe os dados;
- **Controller:** camada de controle. Responsável por receber todas as requisições do usuário.

A Figura 18 mostra o funcionamento do MVC

Figura 18 - Funcionamento do MVC

Fonte: DEVMEDIA

3- DESENVOLVIMENTO

Este capítulo visa informar todas as etapas para a elaboração do projeto.

A seção 3.1 apresenta os requisitos Funcionais e não funcionais do sistema. A seção 3.2 apresenta a montagem das mensagens de dados do ELISA, seguindo a documentação da PMM. A seção 3.3 apresenta qual o formato das mensagens de dados. A seção 3.4 apresenta o tempo de varredura e a quantidade de mensagens que são enviados em cada mensagem e a seção 3.5 apresenta as medidas para a simulação das mensagens do ELISA.

3.1- Requisitos

Para a elaboração de todo o sistema, foi necessário o levantamento dos requisitos funcionais juntamente com a pesquisadora responsável pelo projeto, Dra. Ing Hwie Tan. Em entrevista, foram levantados todos os requisitos que o projeto necessita.

3.1.1- Requisitos Funcionais

Segue abaixo os requisitos levantados para o sistema.

Sendo: responsável

Gostaria que: houvessem testes da comunicação serial entre os dispositivos (*notebooks*) seguindo o protocolo de comunicação.

Para que: teste e valide o protocolo de comunicação serial entre o experimento e o computador de bordo.

Sendo: responsável

Gostaria que: simulasse o computador de bordo (programa em LabVIEW).

Para que: simule sua comunicação serial com o circuito de interface do experimento.

Sendo: responsável

Gostaria que: simulasse o circuito de interface do experimento ELISA (programa em Python).

Para que: simule sua comunicação com o computador de bordo.

Fonte: Produção da Autora (2016).

Sendo: Usuário

Gostaria que: houvesse uma interface fácil e amigável do simulador do computador de bordo e do simulador do experimento.

Para que: o teste e a validação ocorram facilmente, sem problemas de manuseio.

Fonte: Produção da Autora (2016)

3.1.2- Requisitos Não Funcionais

É necessário que cada programa esteja em computadores distintos, caso não esteja, é possível que o envio de mensagens geradas pelo programa em Python não aconteça e cause um travamento no sistema, por sobrecarga de memória.

Para que a comunicação aconteça, é necessário um cabo serial *cross-over*, ou seja, um cabo cruzado para comunicação entre dois computadores.

3.2- Montagem das Mensagens de Dados do ELISA

Como a plataforma da missão EQUARS e seu computador de bordo (OBC) ainda não foram definidos, este trabalho seguirá o protocolo de comunicação estabelecido para a Plataforma Multimissão (PMM) em desenvolvimento no INPE.

Baseado na documentação existente para a PMM, foram criadas as mensagens para a comunicação entre o experimento e o computador de bordo.

3.2.1- Formato da Mensagem

A Tabela 4 mostra como é o formato das mensagens do OBC para o ELISA

Tabela 4 - Formato das mensagens do OBC para o ELISA

Start		***Cmd***	***Length***	***Checksum***	
EB_H	**95_H**	**XX**	**XX**	**CKS1**	**CKS2**

Fonte: Produção da Autora (2016)

Início (*Start*)

- EB_H: início da mensagem
- 95_H: número do Experimento

Comando (*Cmd*)

- XX: comando, ou seja, o que o experimento deve fazer, se é um envio de dados, pedido de dados, etc.

Lenght

- XX: tamanho da mensagem.

Checksum

- CKS1 e CKS2: soma de checagem.

A Tabela 5 mostra como é o formato das mensagens do ELISA para o OBC.

Tabela 5 - Formato das mensagens do ELISA para o OBC

Start		***Cmd***	***Length***	***SubCmd***	***Dados***	***Checksum***	
EB_H	**95_H**	**XX**	**XX**	**XX**	**N**	**CKS1**	**CKS2**

Fonte: Produção da Autora (2016)

Início (*Start*) - igual ao anterior

Comando (*Cmd*) – igual ao anterior

Lenght – igual ao anterior

Subcomando (SubCmd) – igual ao anterior

- Subcomando, ou seja, algo específico que deve ser feito de um comando.

Dados

- Dados que a mensagem enviará.

Checksum

- CKS1 e CKS2: mensagem de confirmação

Como foi descrito na seção 2.1.2 (Figura 4) a estrutura dos dados do experimento ELISA se baseia em varreduras feitas na tensão das placas. Cada varredura terá 16 valores de tensão nas placas e para cada tensão o número de contagens é armazenado. Uma varredura de monitoramento (*Housekeeping*) será feita a cada 640 segundos (10 minutos e 40 segundos), e o tempo de varredura (Tvarr) poderá ser de 3.2s ou 0,8s. Desse modo, para Tvarr = 3,2s, uma varredura de monitoramento é seguida de 199 varreduras em operação normal, e para Tvarr = 0,8s a varredura de monitoramento é seguida de 799 varreduras em operação normal.

Cada mensagem de dados contém:

5 bytes (*Header*) + 7 bytes (data e hora) = 12 bytes + 2 (CKS) = 14 bytes

Data e hora serão preenchidas pelo OBC.

Os dados propriamente ditos terão em cada varredura:

- Em modo de monitoramento (*housekeeping*):
  - ✓ Parâmetros: 1 byte (Tempo de varredura e modo de operação ocupando 1 bit cada)
  - ✓ Tensão fornecida às *channeltrons*: 1 byte (~ 2,5 V)
  - ✓ Tensões fornecidas às placas dos dois analisadores: 16 bytes
  - ✓ Contagens medidas pelos 2 detectores: 2x16x3bytes = 96 bytes
  - ✓ Total: 1+1+16+96 = 114 bytes

- Em modo normal de operação:
  - ✓ Parâmetros: 1byte (Tempo de varredura e modo de operação ocupando 1bit cada)
  - ✓ Número da varredura dentro do loop: 2 bytes (max 1024)
  - ✓ Contagens medidas pelos 2 detectores: 2x16x3bytes = 96 bytes.
  - ✓ Total: 1+2+96 = 99 bytes

Como cada mensagem terá no máximo 255 bytes de dados propriamente ditos, ele terá no máximo duas varreduras.

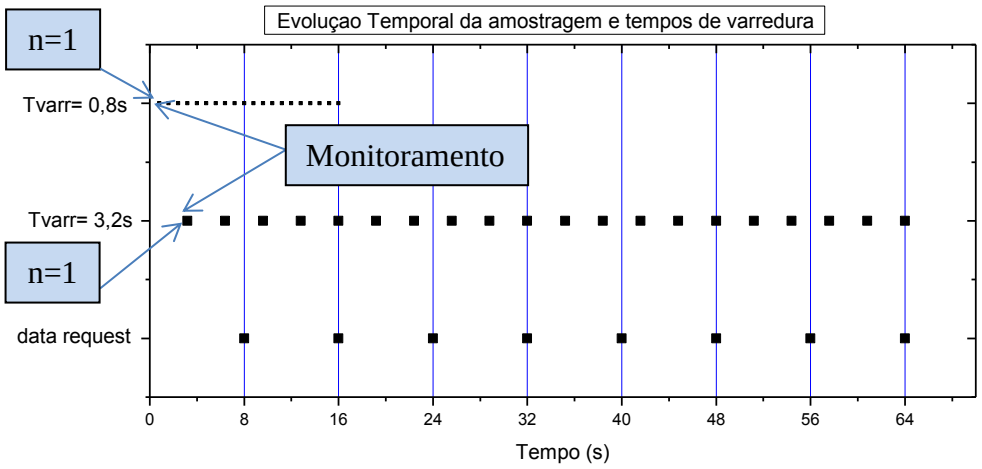
Note-se que as mensagens de dados, podem conter (ou não, na maioria das vezes) a varredura de monitoramento, que ocorre apenas uma vez em 200 (ou 800) varreduras.

3.2.2- Tempo de Varredura e Quantidade de Mensagens Enviadas

A quantidade de mensagens enviadas dependerá do tempo de varredura (Tvarr) do experimento que poderá ser de 3,2s ou de 0,8s. O experimento será lançado com Tvarr = 3,2s e só será modificado se houver necessidade, por telecomando. Será assumido que o OBC fará uma amostragem de cada experimento a cada 8 segundos, isto é, enviará um comando de "*data request*" a cada 8s. Logicamente para Tvarr = 0,8s a o pacote de mensagens enviadas será maior do que para Tvar = 3,2s.

As figuras 19 e 20 mostram a evolução no tempo dos eventos esquematizados na figura 4 (operação do experimento ELISA no espaço) no início e no final do loop de N varreduras para os dois tempos de varredura possíveis.

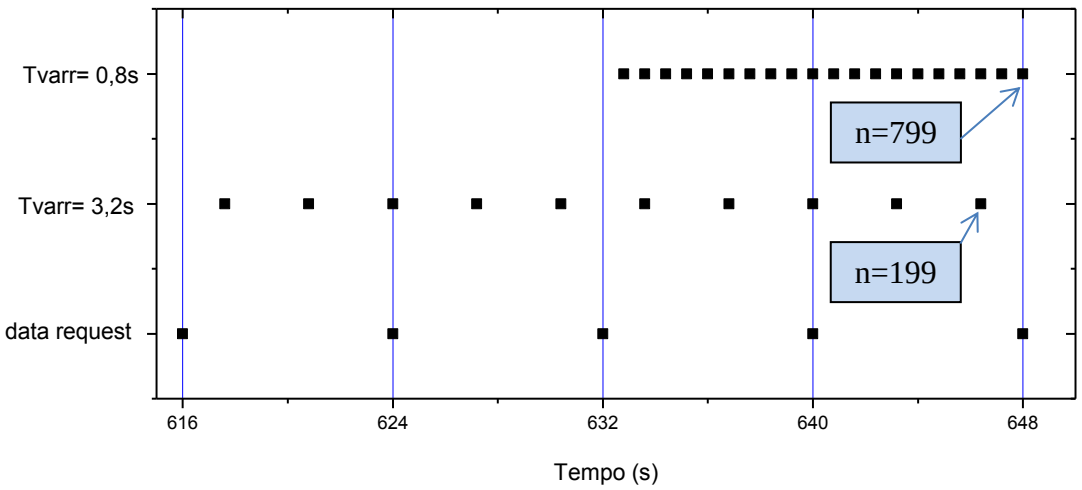
Figura 19 - Evolução temporal da amostragem e tempos–início de n varreduras



Fonte: Tan (2016)

A Figura 20 mostra a contagem dos números da varredura.

Figura 20 - Evolução temporal da amostragem e tempos de varredura–final de n varreduras



Fonte: Tan (2016)

Pela evolução mostrada acima, para um tempo de amostragem dos dados feitos pelo OBC de 8 segundos (*data request* a cada 8 segundos, mostrados pelas linhas azuis), vemos

que para o tempo de varredura igual a 3,2 segundos na primeira e segunda amostragem é possível enviar uma mensagem com dados de duas varreduras, já na terceira teríamos que enviar um pacote de dados com três varreduras e depois os pacotes teriam sucessivamente duas e três varreduras. Como cada mensagem com no máximo 255 bytes comporta apenas duas varreduras, as mensagens enviadas ao OBC teriam alternadamente um pacote com 2 mensagens (uma com duas varreduras e uma com 1 varredura) e um pacote com uma mensagem (com duas varreduras).

Para o tempo de varredura igual a 0,8 segundos, a cada amostragem um pacote de dados com dez varreduras (com exceção da primeira que teria 9, mas isso não irá ocorrer porque o experimento será lançado com tempo de varredura igual a 3,2 segundos) é sempre enviado, com cinco mensagens de duas varreduras cada.

Mesmo quando há mudança no tempo de varredura, isso só ocorrerá no final de um loop de 200 varreduras (um monitoramento e 199 normais) (ou 800 varreduras para tvar = 0,8s) que dura 10 minutos 40 segundos.

A Tabela 7 mostra a evolução temporal da comunicação e a quantidade de varreduras por mensagem.

Tabela 6 - Evolução temporal da comunicação OBC – ELISA

Comando OBC -> ELISA	Resposta ELISA -> OBC
TurnOn	**Acknowledge**
Data Request	Data Message – Data (Contagens) (pacote c/2 varr.)
Data Request	Data Message – Data (Contagens) (pacote c/2 varr.)
Data Request	Data Message – Data (Contagens) (pacote c/3 varr.)
Data Request	Data Message – Data (Contagens) (pacote c/2 varr.)
Data Request	Data Message – Data (Contagens) (pacote c/3 varr.)
Data Request	Data Message – Data (Contagens) (pacote c/2 varr.)
Data Request	Data Message – Data (Contagens) (pacote c/3 varr.)
Data Request	Data Message – Data (Contagens) (pacote c/2 varr.)
Data Sent – T varredura = 0,8s	**Acknowledge**
Data Request	Data Message – Data (Contagens) (pacote c/10 varr.)
Data Request	Data Message – Data (Contagens) (pacote c/10 varr.)
Data Request	Data Message – Data (Contagens) (pacote c/10 varr.)
Data Request	Data Message – Data (Contagens) (pacote c/10 varr.)
Data Sent – T varredura = 3,2s	**Acknowledge**
Data Request	Data Message – Data (Contagens) (pacote c/3 varr.)
Data Request	Data Message – Data (Contagens) (pacote c/2 varr.)
Data Request	Data Message – Data (Contagens) (pacote c/3 varr.)
Data Request	Data Message – Data (Contagens) (pacote c/2 varr.)

Fonte: Tan (2016)

3.2.3- Mensagens

A Tabela 8 informa os números dos comandos e respostas entre o experimento ELISA e o OBC.

Tabela 7 - Comandos e respostas entre o experimento e o OBC

COMANDO DAS MENSAGENS (OBC → ELISA)		**COMANDO DE RESPOSTA (ELISA → OBC)**		
01_H	**DATA REQUEST** (Pede as varreduras)	04_H	0_H	**Data Message ⇒ Data**
01_H	**DATA REQUEST** (Pede as varreduras)	04_H	12_H	**Data Message ⇒ Data**
02_H	**DATA SEND** (Tempo de Varredura)	03_H		**Acknowledge**
07_H	**RESET** (Reset o experimento**)**	03_H		**Acknowledge**
$0B_H$	**TURN ON** (Liga o experimento)	03_H		**Acknowledge**
	TURN OFF (Desliga o experimento)	03_H		**Acknowledge**

Fonte: Produção da Autora (2016)

A Tabela 9 informa o formato das mensagens que serão trocados entre o experimento ELISA e o OBC.

Tabela 8 - Mensagens trocadas na comunicação.

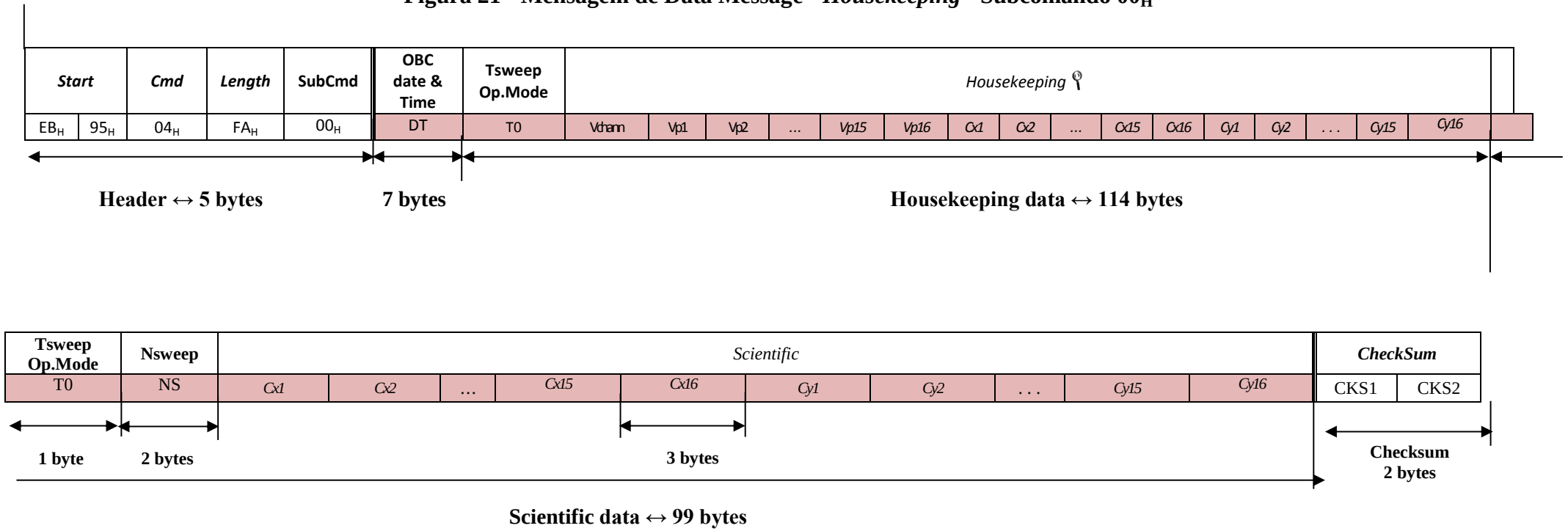
<table>
<tr><th>FORMATO DAS MENSAGENS DE COMANDO
(OBC → ELISA)</th><th>FORMATO DAS MENSAGENS DE RESPOSTAS
(ELISA → OBC)</th></tr>
<tr><td>
Data Request
<table>
<tr><th colspan="2">Start</th><th>Cmd</th><th>Length</th><th colspan="2">CheckSum</th></tr>
<tr><td>EB_H</td><td>95_H</td><td>01_H</td><td>00_H</td><td>CKS1</td><td>CKS2</td></tr>
</table>
</td><td>
Data Message ⇒ Data
<table>
<tr><th colspan="2">Start</th><th>Cmd</th><th>Length</th><th>Scd</th><th>Field</th><th colspan="2">CheckSum</th></tr>
<tr><td>EB_H</td><td>95_H</td><td>04_H</td><td>FA_H</td><td>00_H</td><td>*</td><td>CKS1</td><td>CKS2</td></tr>
</table>
Data Message ⇒ Data
<table>
<tr><th colspan="2">Start</th><th>Cmd</th><th>Length</th><th>Scd</th><th>Field</th><th colspan="2">CheckSum</th></tr>
<tr><td>EB_H</td><td>95_H</td><td>04_H</td><td>FA_H</td><td>12_H</td><td>*</td><td>CKS1</td><td>CKS2</td></tr>
</table>
</td></tr>
<tr><td>
Data Send - Tvarredura
<table>
<tr><th colspan="2">Start</th><th>Cmd</th><th>Length</th><th>Field</th><th colspan="2">CheckSum</th></tr>
<tr><td>EB_H</td><td>95_H</td><td>02_H</td><td>$1D_H$</td><td>1_H</td><td>CKS1</td><td>CKS2</td></tr>
</table>
</td><td>
Acknowledge
<table>
<tr><th colspan="2">Start</th><th>Cmd</th><th>Length</th><th colspan="2">CheckSum</th></tr>
<tr><td>EB_H</td><td>95_H</td><td>03_H</td><td>00_H</td><td>CKS1</td><td>CKS2</td></tr>
</table>
</td></tr>
<tr><td>
Turn On
<table>
<tr><th colspan="2">Start</th><th>Cmd</th><th>Length</th><th colspan="2">CheckSum</th></tr>
<tr><td>EB_H</td><td>95_H</td><td>$0B_H$</td><td>00_H</td><td>CKS1</td><td>CKS2</td></tr>
</table>
</td><td>
Acknowledge
<table>
<tr><th colspan="2">Start</th><th>Cmd</th><th>Length</th><th colspan="2">CheckSum</th></tr>
<tr><td>EB_H</td><td>95_H</td><td>03_H</td><td>00_H</td><td>CKS1</td><td>CKS2</td></tr>
</table>
</td></tr>
<tr><td>
Turn Off
<table>
<tr><th colspan="2">Start</th><th>Cmd</th><th>Length</th><th colspan="2">CheckSum</th></tr>
<tr><td>EB_H</td><td>95_H</td><td>01_H</td><td>00_H</td><td>CKS1</td><td>CKS2</td></tr>
</table>
</td><td>
Acknowledge
<table>
<tr><th colspan="2">Start</th><th>Cmd</th><th>Length</th><th colspan="2">CheckSum</th></tr>
<tr><td>EB_H</td><td>95_H</td><td>03_H</td><td>00_H</td><td>CKS1</td><td>CKS2</td></tr>
</table>
</td></tr>
</table>

Fonte: Produção da Autora (2016)

As Figuras 21 e 22 mostam de maneira específica uma mensagem de dados com uma varredura de *housekeeping* e uma de modo normal (duas varreduras em modo normal), respectivamente.

*** Data Message mensagem (256 Bytes) - Subcomando 00_H:**

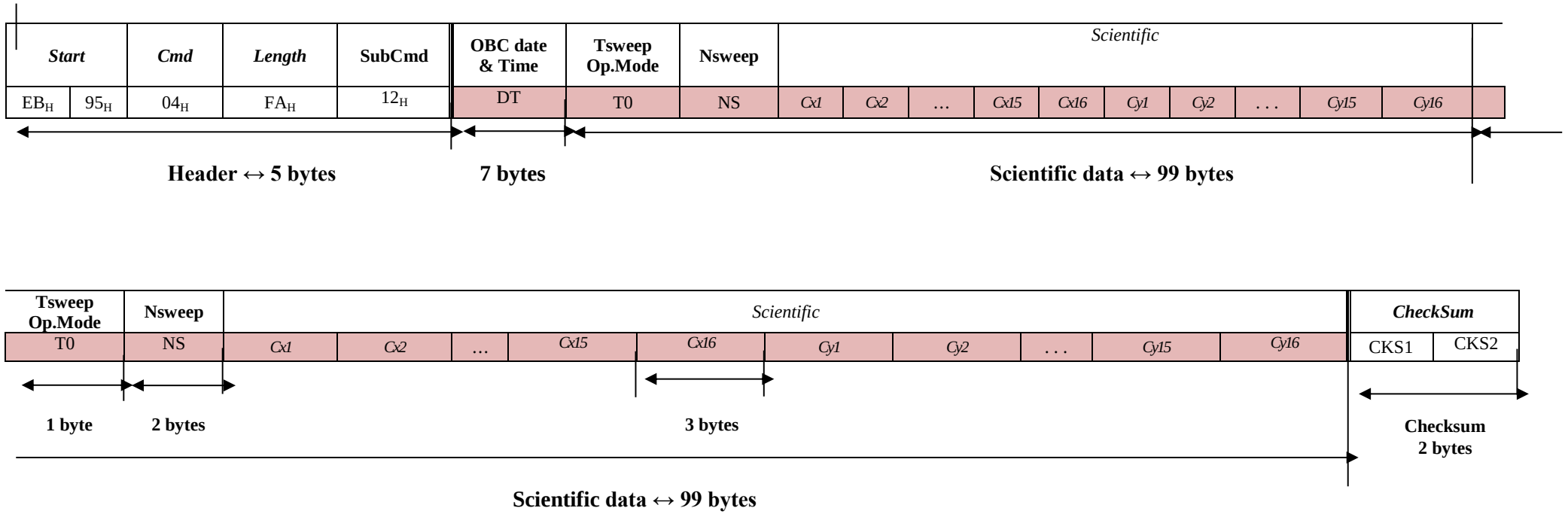
Figura 21 - Mensagem de Data Message - *Housekeeping* - Subcomando 00_H



Fonte: Produção da Autora (2016)

*** Data Message mensagem (256 Bytes) – Subcomando 12_H:**

Figura 22 - Mensagem de Data Message – Modo Normal - Subcomando 12_H



Fonte: Produção da Autora (2016)

Tabela 9 - Definição das siglas das mensagens

Definição das Siglas	
VP	Voltagem das Placas
Scd	Subcomando
OBC date & time	OBC Data e hora
Tsweep Op.Mode	Modo de operação
Housekeeping	Modo de monitoração/calibração.
Scientific	Dados Científicos
Nsweep	Número da varredura
CheckSum	Soma de confirmação
DT	Dado da hora e data
T0	Bit verificador se está em modo de *Houskeeping* ou normal.
NS	Dado do número da Varredura.
Cx1 … Cx15	Contagens (16 valores) do analisador X
Cy1 … Cy15	Contagens (16 valores) do analisador Y
CKS1 e CKS2	Dado do *CheckSum*

Fonte: Produção da Autora (2016)

3.2.4- Simulação das Mensagens do ELISA

Para o programa que simulará o circuito de interface e preparará as mensagens, vamos montar os vetores com as medidas que seriam feitas:

Voltagem das Placas (Vp): máximo de 2,7V, decaindo exponencialmente até 0,09V

Vp = 2,7 exponencial (-nk) n=0 -> Vp = 2,7 n=15 -> Vp = 0,09

0,09 = 2,7 exponencial (-15k)

Ln (0,09/2,7) = ln(-15k) = -15k

k = 0,2267

n=15 -> $Vp = 2,7x2,718^{-15x0,2267} = 0,0900945$

n=14 -> $Vp = 2,7x2,718^{-14x0,2267} = 0,1130$

n=13 -> $Vp = 2,7x2,718^{-13x0,2267} = 0,1418$ etc..

Vamos digitalizar, isto é, transformar os números em números inteiros, e maximizar a precisão dessa digitalização já que o maior número que é possível escrever ocupando 1 byte é 127.

Para isso multiplicamos a voltagem da placa por 47 (o que leva o valor máximo de 2,7 para muito perto de 127), e arredondamos para o número inteiro mais próximo, como mostra a seguinte tabela:

Voltagem da *Channeltron*: 2,5 * 47 ~= 117 = 7F hexadecimal.

A Tabela 11 mostra os valores digitalizados e seu valor em hexadecimal.

Tabela 10 - Valores digitalizados das placas

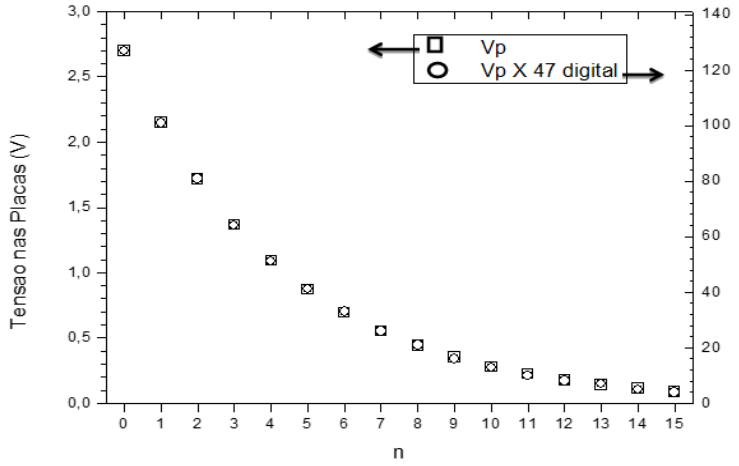
n	Vp	Vp x 47	Vp arredondado	Hexadecimal
0	2,7	126,9	127	7F
1	2,15233	101,16	101	65
2	1,71575	80,64037	81	51
3	1,36773	64,28327	64	40
4	1,0903	51,24404	51	33
5	0,86914	40,8497	41	29
6	0,69285	32,56374	33	21
7	0,55231	25,95851	26	1A
8	0,44028	20,69308	21	15
9	0,35097	16,49569	16	10
10	0,27978	13,14971	13	D
11	0,22303	10,48242	10	A
12	0,17779	8,35616	8	8
13	0,14173	6,6612	7	7
14	0,11298	5,31004	5	5
15	0,09006	4,23295	4	4

Fonte: Produção da Autora (2016)

O gráfico na Figura 23 mostra que os valores arredondados reproduzem bem o comportamento dos valores reais.

Para recuperar as voltagens da placa original, basta dividir os valores digitalizados (arredondados para números inteiros) por 47.

Figura 23 - Margem de precisão da digitalização da tensão



Fonte: Tan (2016)

Para construir os outros vetores podemos usar a sequência exponencial de números inteiros criada acima e multiplicar por números arbitrários:

Por exemplo, as contagens dos dois analisadores em geral também têm uma queda exponencial (seria o esperado).

Para um dos analisadores podem ser iguais aos números da quarta coluna (com máximo em 127) que ocupa apenas 1 byte, e o dobro disso no outro.

A Tabela 12 mostra a contagem do analisador X digitalizado.

Tabela 11 - Contagem do analisador X digitalizado

n	Contagem	Contagem x 47	Contagem arredondada	Hexadecimal
0	2,7	126,9	127	7F
1	2,15233	101,16	101	65
2	1,71575	80,64037	81	51
3	1,36773	64,28327	64	40
4	1,0903	51,24404	51	33
5	0,86914	40,8497	41	29
6	0,69285	32,56374	33	21
7	0,55231	25,95851	26	1A
8	0,44028	20,69308	21	15
9	0,35097	16,49569	16	10
10	0,27978	13,14971	13	D
11	0,22303	10,48242	10	A
12	0,17779	8,35616	8	8

13	0,14173	6,6612	7	7
14	0,11298	5,31004	5	5
15	0,09006	4,23295	4	4

Fonte: Produção da Autora (2016)

A Tabela 13 mostra a contagem do analisador Y digitalizado.

Tabela 12 - Contagem do analisador Y digitalizado

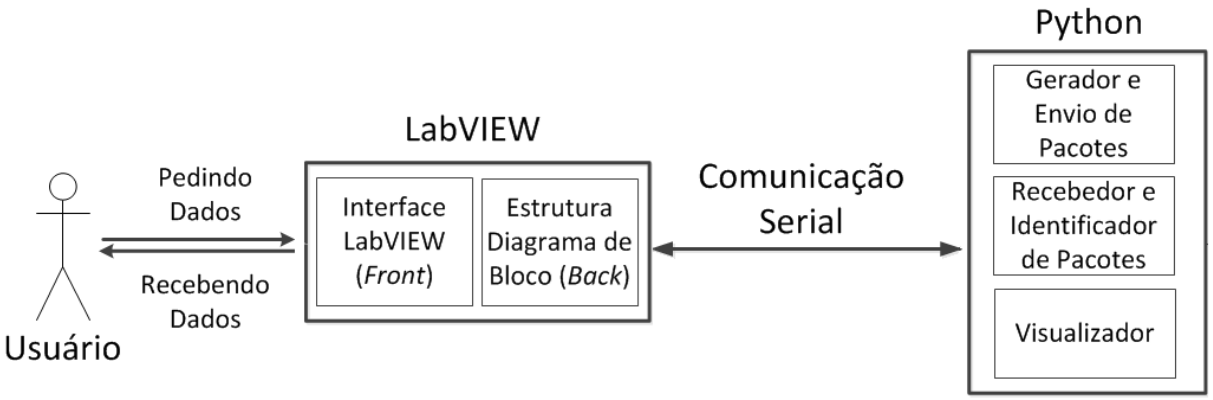
n	Contagem	Contagem x 94	Contagem arredondada	Hexadecimal
0	2,7	253,8	254	FE
1	2,15233	202,31902	202	CA
2	1,71575	161,2805	161	A1
3	1,36773	128,56662	129	81
4	1,0903	102,4882	102	66
5	0,86914	81,69916	82	52
6	0,69285	65,1279	65	41
7	0,55231	51,91714	52	34
8	0,44028	41,38632	41	29
9	0,35097	32,99118	33	21
10	0,27978	26,29932	26	1A
11	0,22303	20,96482	21	15
12	0,17779	16,71226	17	11
13	0,14173	13,32262	13	D
14	0,11298	10,62012	11	B
15	0,09006	8,46564	8	8

Fonte: Produção da Autora (2016)

3.3- Arquitetura

A Figura 24 mostra a arquitetura do sistema.

Figura 24 - Arquitetura do sistema



Fonte: Produção da Autora (2016)

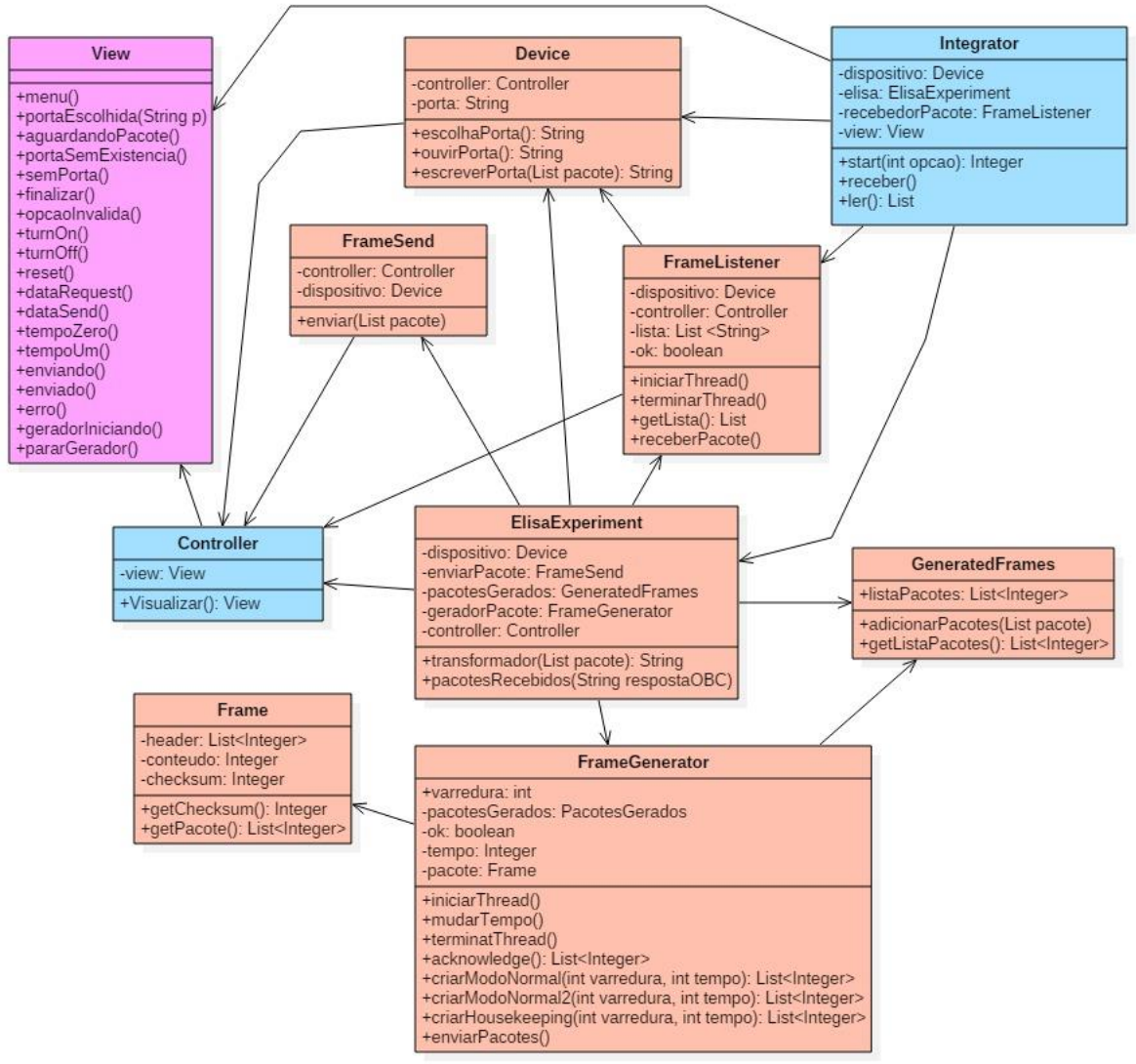
A explicação da arquitetura será feita por módulos, começando pelo Python com seus submódulos, e em seguida o LabVIEW com seus submódulos. O esquema dos módulos ficará assim:

- Módulo 1 – Python
  - Submódulo 1 – Gerador e Recebedor de Mensagens
  - Submódulo 2 – Identificador de Mensagens
  - Submódulo 3 – Visualizador
  - Submódulo 4 - Integrador
- Módulo 2 – LabVIEW
  - Submódulo 1 – Mensagens
  - Submódulo 2 – Recebedor de mensagens
  - Submódulo 3 - Gerador de Mensagens
- Módulo 3 – Comunicação Serial

3.3.1- Módulo 1 – Python

Todo o funcionamento do Circuito de Interface do Experimento ELISA foi simulado em um programa feito na linguagem Python. Sua estrutura foi montada utilizando o padrão MVC, *model-view-controller*. A Figura 25 mostra o diagrama de classe do sistema.

Figura 25 - Diagrama de Classes



Fonte: Produção da Autora (2016)

Cores:

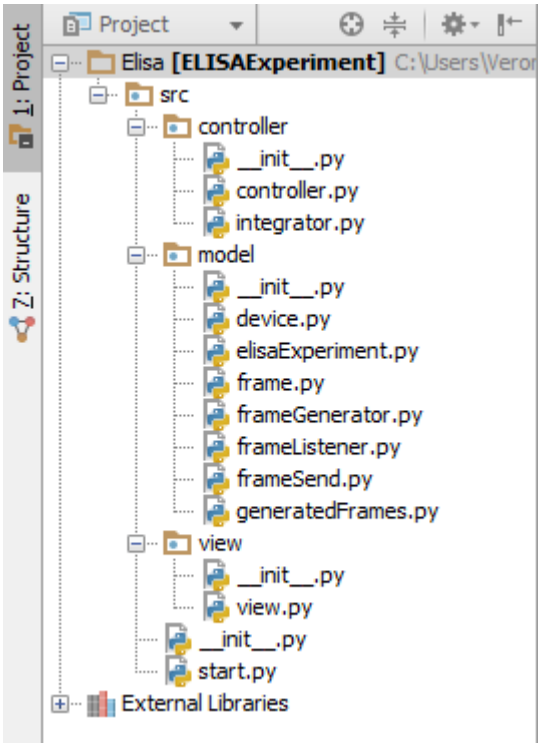
Salmão: *Model*

Rosa: *View*

Verde: *Controller*

A Figura 26 mostra estrutura do programa em Python.

Figura 26 - Estrutura do programa em Python



Fonte: Produção da Autora (2016)

3.3.1.1- Submódulo 1 – Recebedor e Identificador de Mensagens

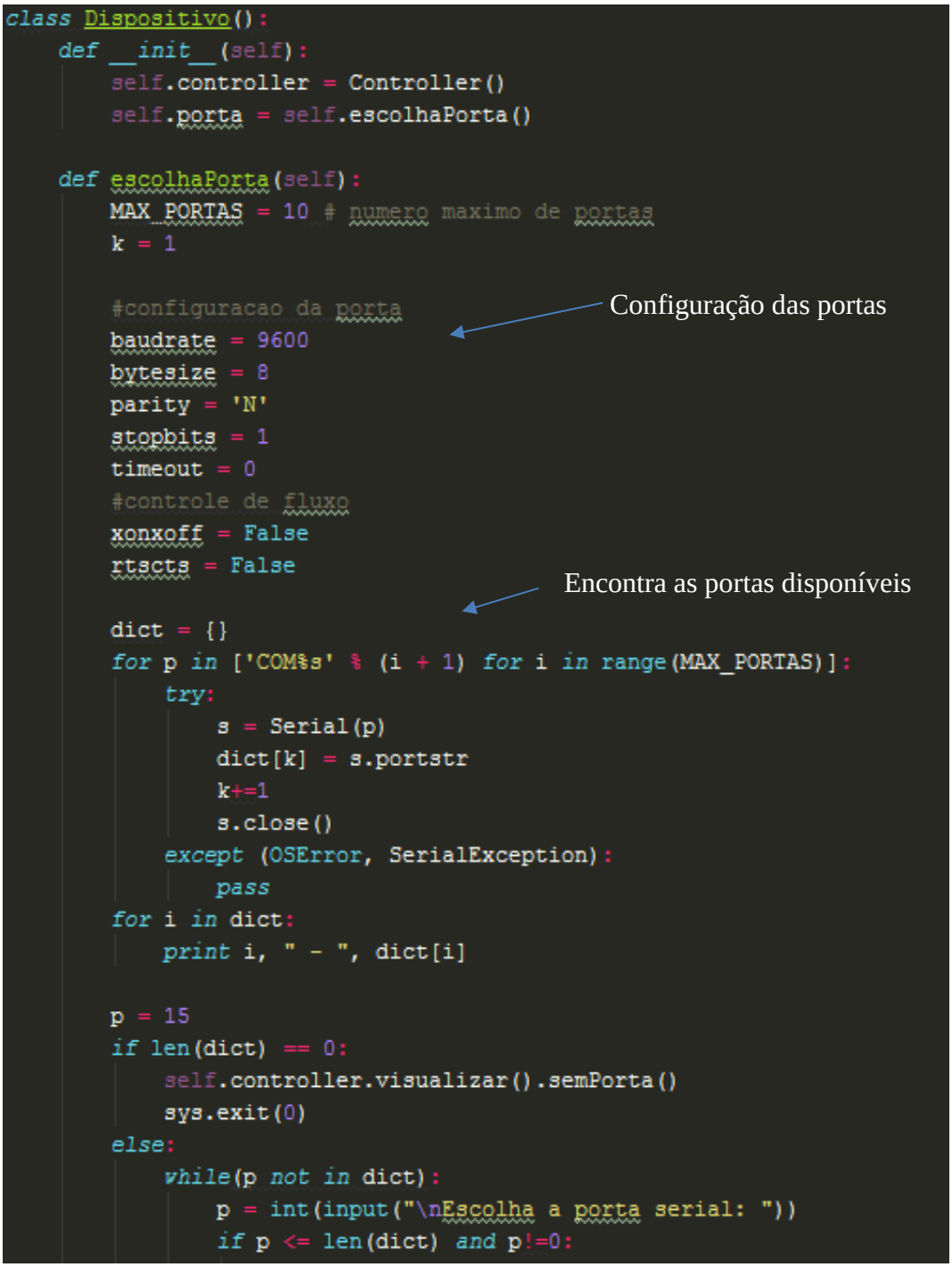
As classes responsáveis pelo recebimento e identificação das mensagens são:

- **Device:** Responsável por toda comunicação serial. A classe dispositivo, Figura 27, configura e abre a porta serial para a comunicação, e é por ela que se recebe e envia as mensagens. As configurações para a porta serial são as seguintes:
    - *baudrate*: 9600
    - *barity*: 8
    - *parity*: 'N' (significa *None*, sem paridade)
    - *stopbits*: 1
    - *timeout*: 0
    - controle de fluxo: xonxoff e rtscts = False

Estes números são valores padrões da comunicação serial, mas podem ser alterados dependendo do dispositivo. Todos os dispositivos que precisam ser comunicados via serial

são necessários estarem com os mesmos valores de configuração da porta para que haja comunicação.

Figura 27 – Trecho do código da classe Device




```
class Dispositivo():
    def __init__(self):
        self.controller = Controller()
        self.porta = self.escolhaPorta()

    def escolhaPorta(self):
        MAX_PORTAS = 10 # numero maximo de portas
        k = 1

        #configuracao da porta
        baudrate = 9600
        bytesize = 8
        parity = 'N'
        stopbits = 1
        timeout = 0
        #controle de fluxo
        xonxoff = False
        rtscts = False

        dict = {}
        for p in ['COM%s' % (i + 1) for i in range(MAX_PORTAS)]:
            try:
                s = Serial(p)
                dict[k] = s.portstr
                k+=1
                s.close()
            except (OSError, SerialException):
                pass
        for i in dict:
            print i, " - ", dict[i]

        p = 15
        if len(dict) == 0:
            self.controller.visualizar().semPorta()
            sys.exit(0)
        else:
            while(p not in dict):
                p = int(input("\nEscolha a porta serial: "))
                if p <= len(dict) and p!=0:
```

Fonte: Produção da Autora (2016)

- **FrameListener**: responsável abrir uma Thread (forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas concorrencialmente) para receber mensagens vindas da classe dispositivo e adicionar em uma lista de mensagens recebidas.
- **Integrator**: a classe Integrador cria outra Thread para sempre pegar a lista de mensagens recebidas da classe FrameListener e enviar para a classe ElisaExperiment. Ela é chamada de Integrator, pois é responsável por integrar classes, classes ElisaExperiment e FrameListener com a classe Device. Possui as opções de escolha da classe View para o início do programa.
- **ElisaExperiment**: essa classe transforma os dados recebidos em hexadecimal em decimal, após isto verifica o tipo de mensagem recebido e realiza sua devida operação. Suas operações são:
    - **Turn On**: liga o gerador de mensagens;
    - **Turn Off**: desliga o gerador de mensagens;
    - **Data Request**: envia as mensagens gerados;
    - **Data Send**: muda o tempo da varredura ou para 3,2 ou para 0,8 segundos; e
    - **Reset:** muda os valores para os de início.

A Figura 28 mostra a classe ElisaExperiment.

Figura 28 – Trecho do código da classe ElisaExperiment


```
class ExperimentoElisa(object):

    def __init__(self, dispositivo):
        self.__dispositivo = dispositivo
        self.__enviarPacote = EnviarPacote(self.__dispositivo)
        self.__pacotesGerados = PacotesGerados()
        self.__geradorPacote = GeradorPacote(self.__pacotesGerados)
        self.__controller = Controller()

    def transformador(self, pacote):
        f = ''
        for i in pacote:
            f += i
        respostaOBC = [ord(b) for b in f]
        if len(respostaOBC) > 1:
            print respostaOBC
            self.pacotesRecebidos(respostaOBC)

    def pacotesRecebidos(self, respostaOBC):

        if respostaOBC[2] == 11:
            self.__controller.visualizar().turnOn()
            self.__enviarPacote.enviar(self.__geradorPacote.acknowledge())
            self.__geradorPacote.iniciarThread()
            self.__controller.visualizar().geradorIniciando()

        elif respostaOBC[2] == 1:
            self.__controller.visualizar().dataRequest()
            a = self.__pacotesGerados.getListaPacotes()
            self.__enviarPacote.enviar(a)
            print a

        elif respostaOBC[2] == 2:
            self.__controller.visualizar().dataSend()

            self.__enviarPacote.enviar(self.__geradorPacote.acknowledge())
            if respostaOBC[4] == 0:
                self.__controller.visualizar().tempoZero()
                self.__geradorPacote.mudarTempo(0x0)
```

Fonte: Produção da Autora (2016)

3.3.1.2- Submódulo 2 – Gerador e Envio de Mensagens

As classes responsáveis pela geração e envio de mensagens são:

- **FrameGenerator**: Inicia uma Thread para gerar mensagens e enviar para classe GeneratedFrames. As mensagens são geradas respeitando o tempo de varredura e a quantidade de varreduras em cada mensagem conforme definido no subcapítulo Montagem das Mensagens de Dados do ELISA, como mostra a Figura 29. Caso a operação Data Send seja solicitada na classe ElisaExperiment, o tempo de varredura é alterado, caso seja a operação Reset o tempo de varredura retorna para 0,8, valor padrão, e o número da varredura retorna para zero.

Figura 29 – Trecho do código da classe FrameGenerator

```
def enviarPacotes(self):
    t = 0x0
    while self.__ok:
        if t == 0x0:
            if self.varredura == 0xc8 or self.varredura == 0x0:
                # housekeeping
                sleep(6.4) # 2 * 0.8, pois e o tempo de cada varredura
                t = self.__tempo

                #pacote com o housekeeping e uma varredura
                self.varredura = 0x0 #caso a varredura seja igual a 200 ela e zerada
                self.__pacotesGerados.adicionarPacotes(self.criarHousekeeping(self.varredura, t))
                self.varredura = 0x1

            else:
                # modo normal

                # pacote com 2 varreduas
                sleep(6.4)
                self.__pacotesGerados.adicionarPacotes(self.criarModoNormal(self.varredura, t))

                self.varredura += 0x1
                #pacote com 2 varreduras
                sleep(6.4)
                self.__pacotesGerados.adicionarPacotes(self.criarModoNormal(self.varredura, t))

                self.varredura += 0x1
                #pacote com 1 varredura
                sleep(3.2)
                self.__pacotesGerados.adicionarPacotes(self.criarModoNormal2(self.varredura, t))
                self.varredura += 0x1

        else:
            if self.varredura == 0x320 or self.varredura == 0x0:
                # housekeeping
                sleep(1.6)  # 2 * 1.6, pois e o tempo de cada varredura
                t = self.__tempo
```

Pega as mensagens geradas e envia para a classe ElisaExperiment para serem enviados.

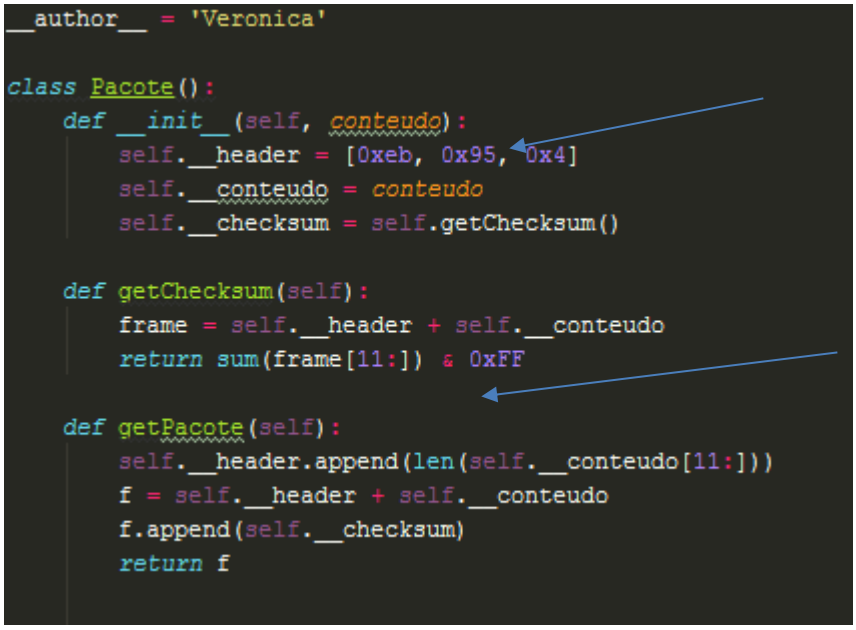
O tempo de criação das mensagens é simulado usando a função *sleep*, que pausa a função pelo tempo determinado.

As mensagens só serão enviadas depois desse tempo.

Fonte: Produção da Autora (2016)

- **Frame**: estrutura de uma mensagem. Possui o *header* e calcula o *checksum* da mensagem, explicado no capítulo 2. A Figura 30 mostra a classe Frame.

Figura 30 - Código da classe Frame



Fonte: Produção da Autora (2016)

- **GeneratedFrames**: é uma lista com todas as mensagens gerados pela classe FrameGenerator. Assim que a mensagem é enviada para o OBC, elas são apagadas para não sobrecarregar o sistema.
- **FrameSend**: responsável por enviar as mensagens para o OBC sempre que a classe ElisaExperiment recebe algum pedido.

3.3.1.3- Submódulo 3 – Visualizador

A classe responsável pela visualização das mensagens para o usuário:

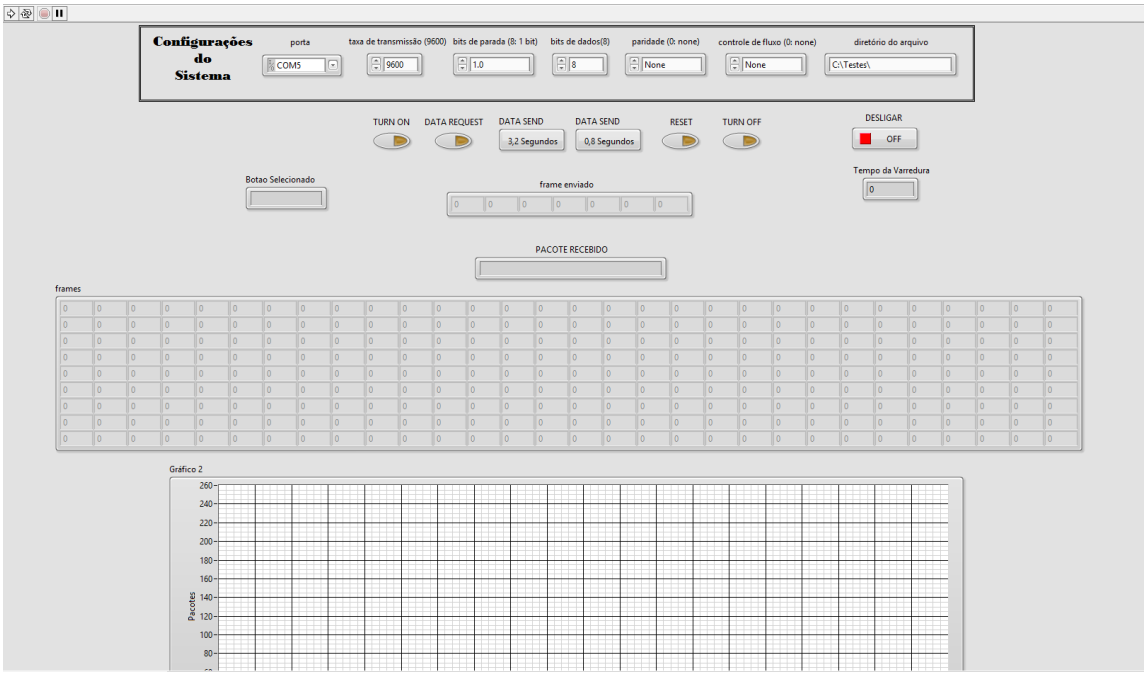
- **View**: Responsável por toda a interação com o usuário. Exibe mensagens na tela.

3.3.2- Módulo 2 – LabVIEW

A estrutura do programa em LabVIEW é sequencial, ou seja, não obedece a estrutura MVC. Possui dois modos de interação, o modo de interface, também chamado de *Front Panel*, e o modo de diagrama de blocos, também chamado de *Block Diagram*.

A Figura 31 mostra a interface do programa, parte visual onde haverá a interação com o usuário.

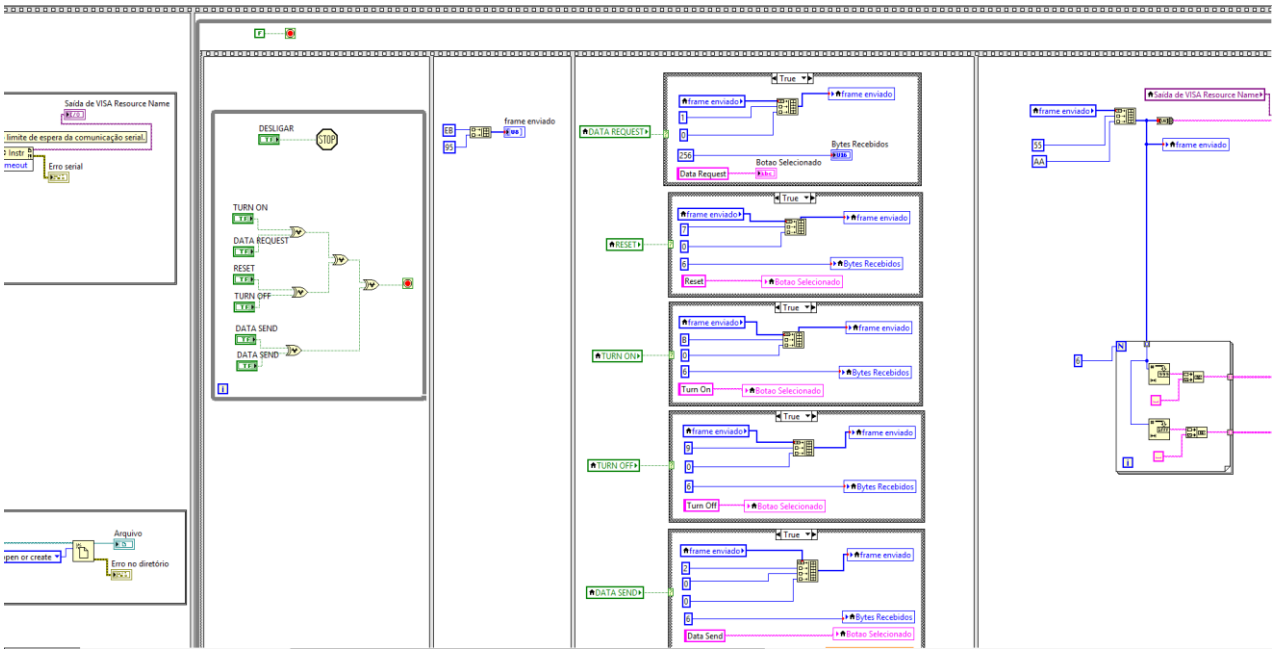
Figura 31 - Interface do programa em LabVIEW



Fonte: Produção da Autora (2016)

Figura 32 exibe seu diagrama de bloco, onde está toda lógica do funcionamento do programa.

Figura 32 - Diagrama de bloco LabVIEW



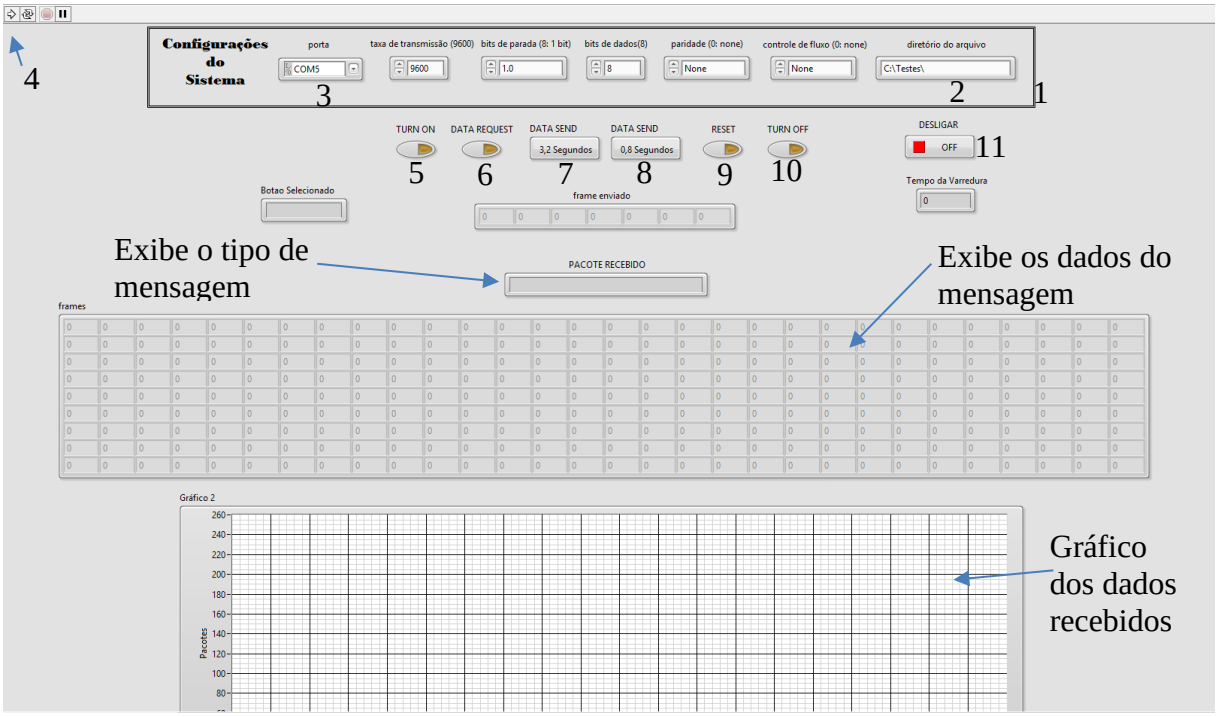
Fonte: Produção da Autora (2016)

3.3.2.1- Interface LabVIEW

O funcionamento da interface é explicado nas figuras a seguir.

Figura 33 mostra o funcionamento do programa em LabVIEW. As partes enumeradas na figura são a seguir explicadas.

Figura 33 - Funcionamento do programa em LabVIEW



Fonte: Produção da Autora (2016)

A escolha da porta e do diretório devem ser feitas antes de executar o programa.

1. Configurações da porta. Valores padrões, já configurados para a comunicação com o Programa em Python.
    - Taxa de transmissão: 9600
    - Bit de parada: 1.0
    - Bit de dados: 8
    - Paridade: None
    - Controle de Fluxo: None
2. Local onde será guardado os arquivos em texto.
3. Seleciona a porta serial para a comunicação.
4. Executa o programa do LabVIEW
5. **TURN ON:** envia uma mensagem para o programa em Python, o qual entende que deve ligar o gerador de mensagens.
6. **DATA REQUEST:** envia uma mensagem pedindo as mensagens de dados geradas.
7. **DATA SEND:** envia uma mensagem para que mude o tempo de varredura do programa em Python para 3.2 segundos

8. **DATA SEND:** envia uma mensagem para que mude o tempo de varredura do programa em Python para 0,8 segundos.
9. **RESET:** envia uma mensagem para que retorne o programa em Python com os valores iniciais.
10. **TURN OFF:** envia uma mensagem para que desligue o gerador de mensagens. As mensagens geradas que ainda estiverem na memória, não são apagados e podem ser recebidos pelo LabVIEW.
11. **STOP:** Desliga o programa.

3.3.2.2- Estrutura Diagrama de Bloco

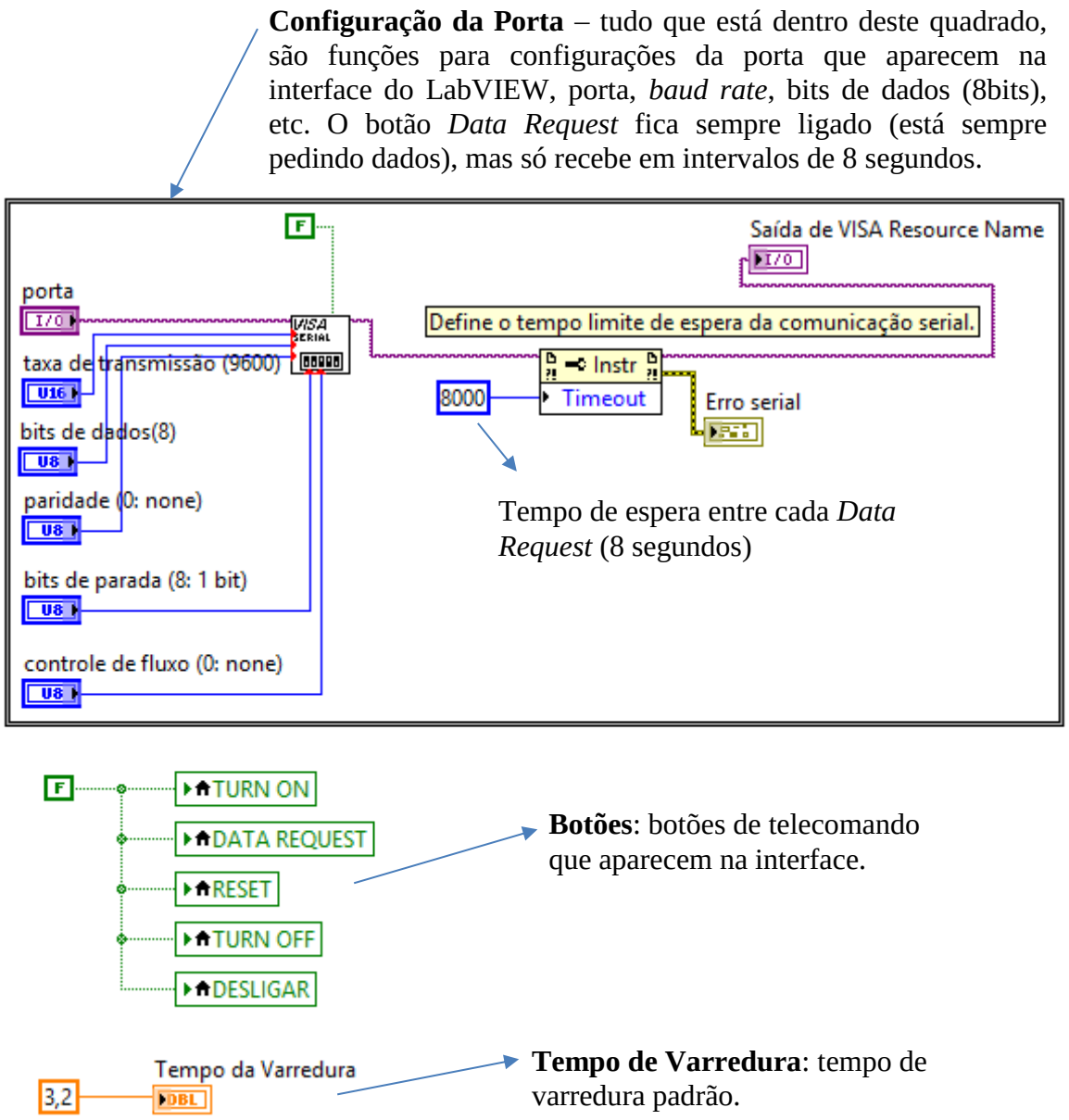
Para abrir a Estrutura Diagrama de Bloco do LabVIEW, onde se programa toda a lógica do sistema, é preciso clicar em Window –> Block Diagram.

Os subcapítulos a seguir mostram como foi montado o sistema que simula o OBC.

3.3.2.2.1- Configuração da porta

A Figura 34 mostra a configuração da porta serial e do arquivo em texto.

Figura 34 - Configuração da porta LabVIEW



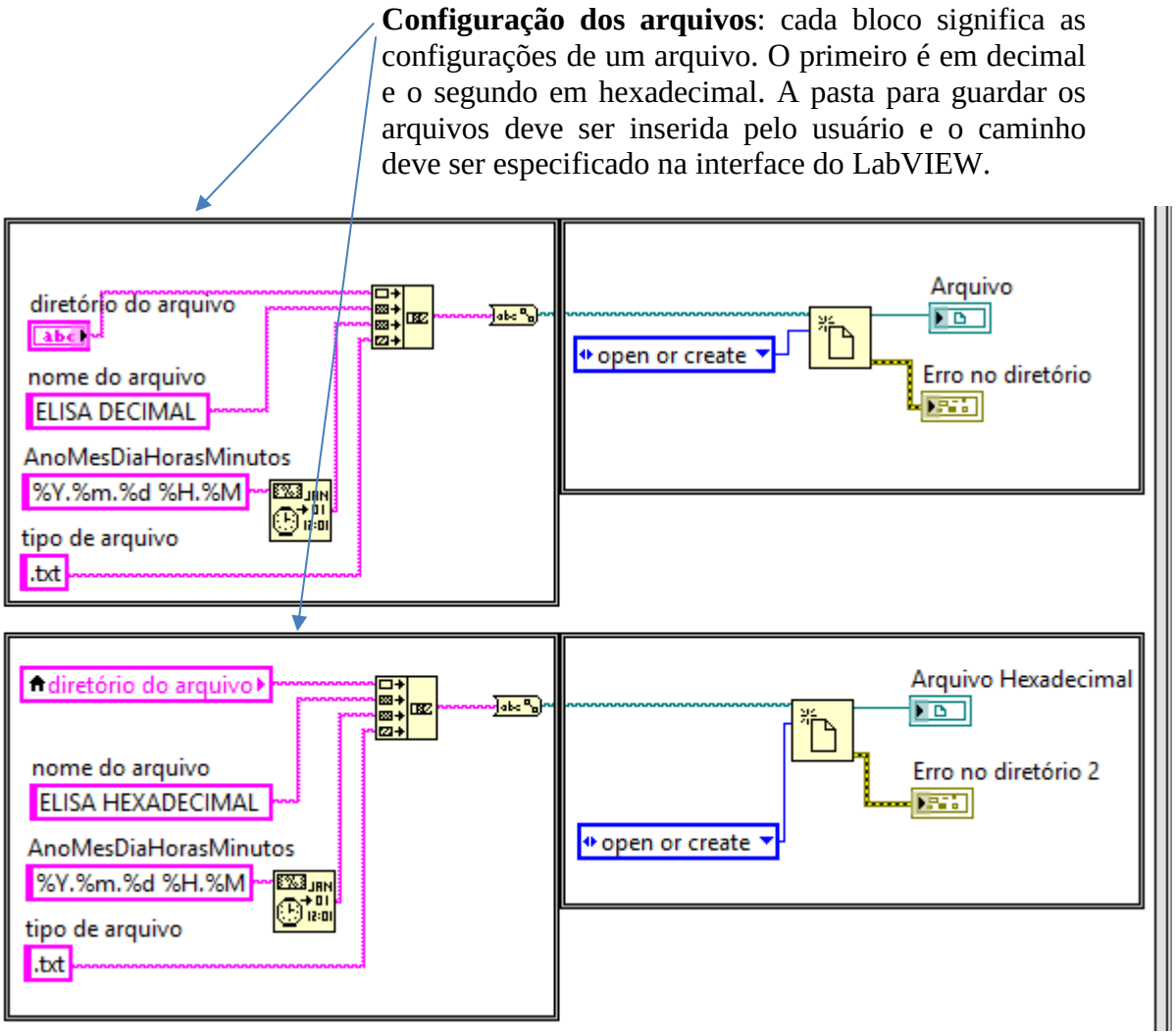
Fonte: Produção da Autora (2016)

3.3.2.2.2- Criação dos arquivos em texto

Serão criados dois arquivos com as mensagens recebidas e enviadas, um em decimal e outro em hexadecimal.

A Figura 35 mostra a configuração para a criação dos arquivos em texto.

Figura 35 - Configuração da criação dos arquivos



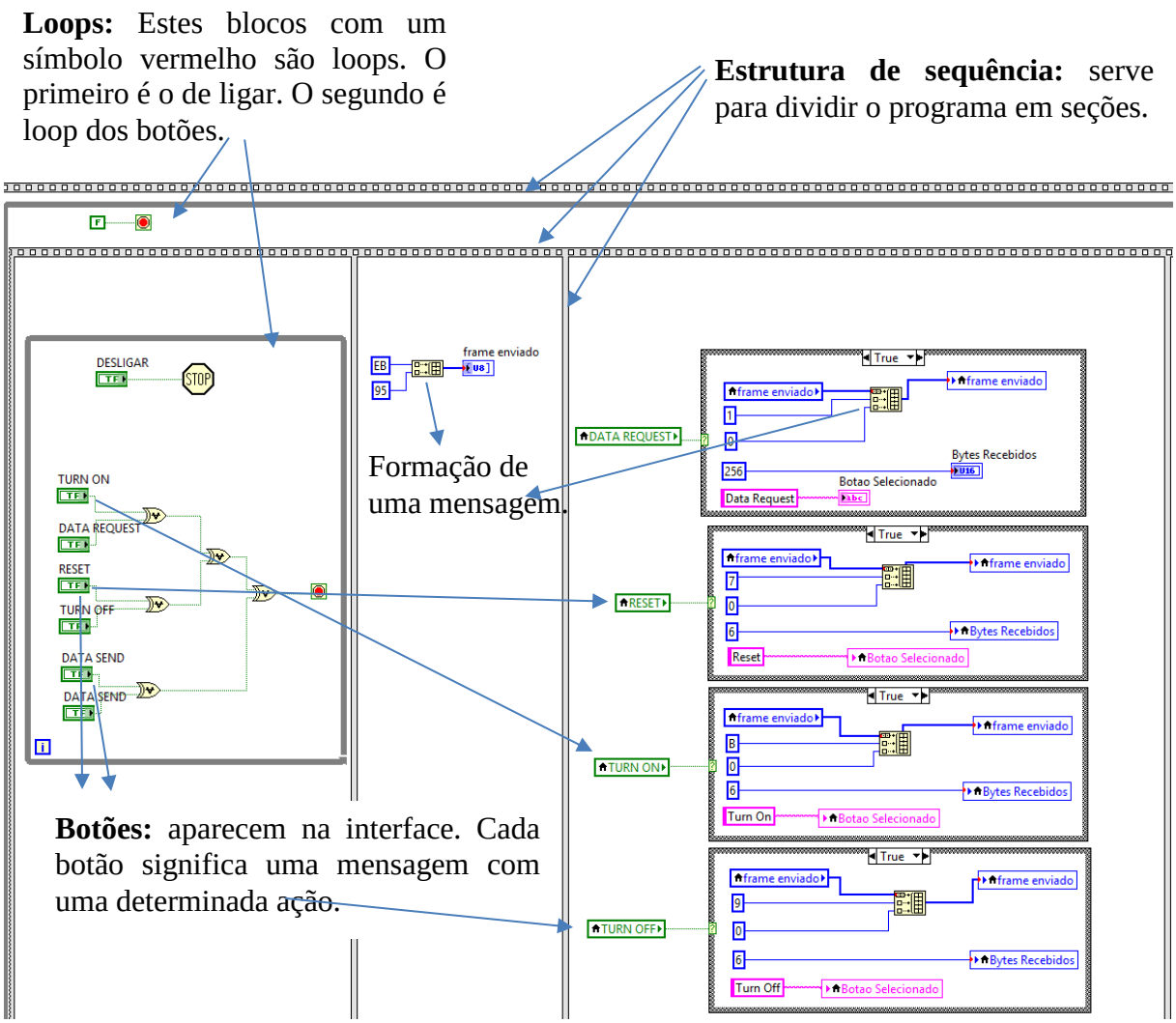
Fonte: Produção da Autora (2016)

3.3.2.2.3- Botões de telecomando

Os botões ficam dentro de um loop que é iniciado sempre que um botão é pressionado. Enquanto o botão estiver pressionado, sua determinada função é habilitada repetidamente. Há um outro loop, o que liga o programa, ele fica habilitado até que o programa é finalizado.

A Figura 36 mostra a estrutura dos botões e o funcionamento da construção das mensagens de cada uma.

Figura 36 - Botões e suas funções LabVIEW



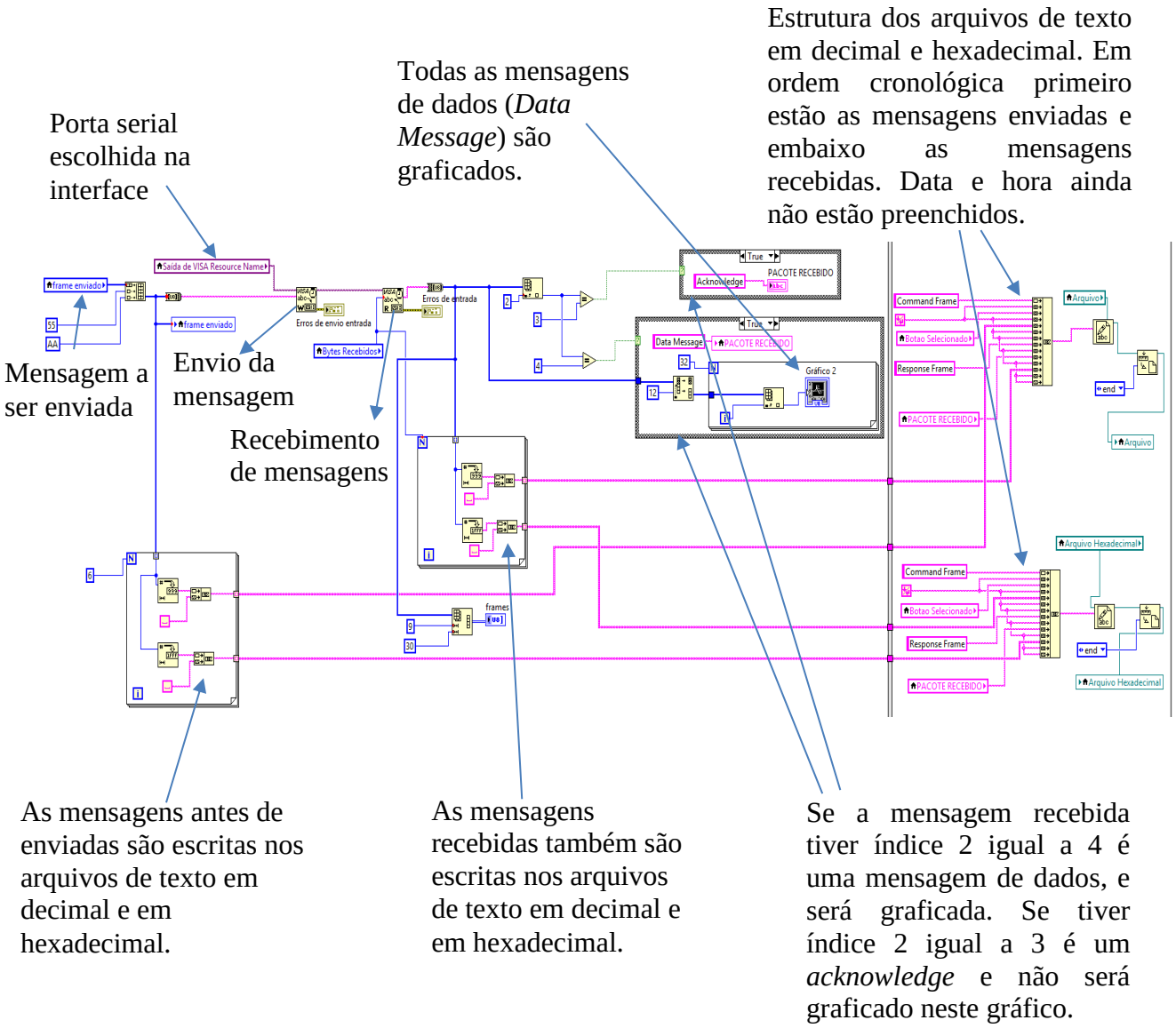
Fonte: Produção da Autora (2016)

3.3.2.2.4- Envio e recebimento de mensagens

O LabVIEW só permitirá a entrada de dados quando uma mensagem for enviada, caso nenhuma seja ele não receberá nada.

A Figura 37 mostra toda a estrutura do LabVIEW para envio e recebimento de mensagens e a escrita no arquivo em texto configurado no início.

Figura 37 - Envio e recebimento de dados LabVIEW



Fonte: Produção da Autora (2016)

3.4- Montagem dos Conectores e Comunicação

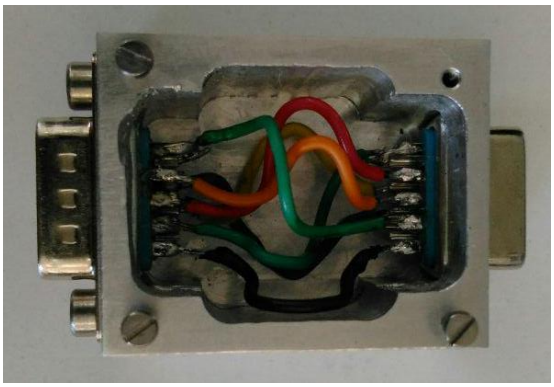
Este subcapítulo mostra os conectores montados e como é feita a comunicação.

A comunicação entre os dispositivos ocorreu utilizando dois conectores *cross-over* diferentes: um para o caso de comunicação usando RS-232 e outro para comunicação por RS-422. Em ambos os casos é necessário o uso de adaptadores pois os laptops utilizados somente possuem saída USB.

O teste inicial de comunicação foi feito utilizando RS-232, já que não necessita de outros conversores RS-232 para RS-422 para realizar a comunicação.

A Figura 38 mostra o conector RS-232 montado para a comunicação com a pinagem *cross-over*, como explicado na Figura 12 do capítulo 2.

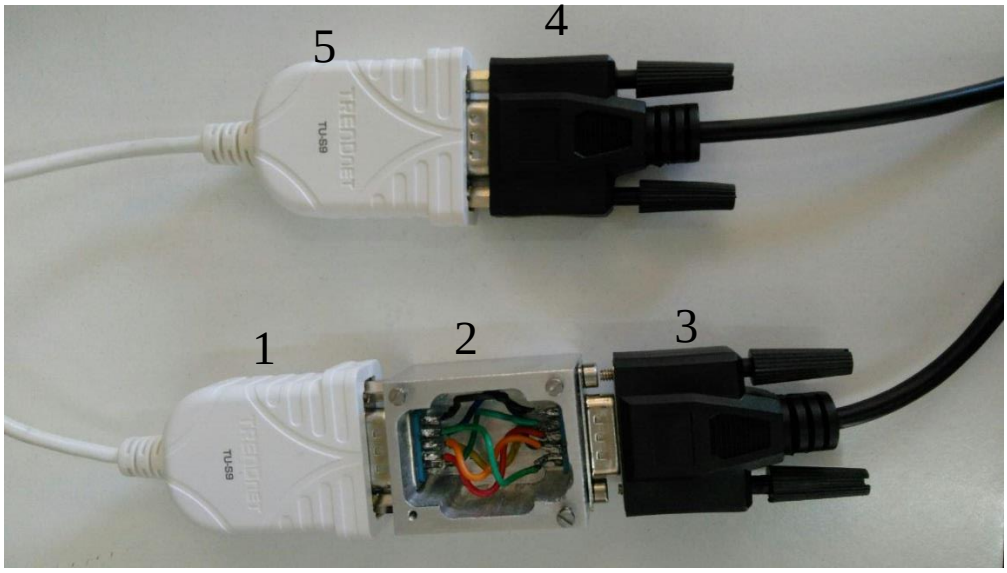
Figura 38 - Conector *cross-over* RS-232



Fonte: Produção da Autora (2016)

Para a comunicação, foram utilizados dois adaptadores USB-Serial-RS232 (modelo TU-S9 da Trendnet) e o conector *cross-over* mostrado na figura 38. A comunicação foi montada conforme mostra a Figura 39.

Figura 39 - Ligação dos cabos e conectores

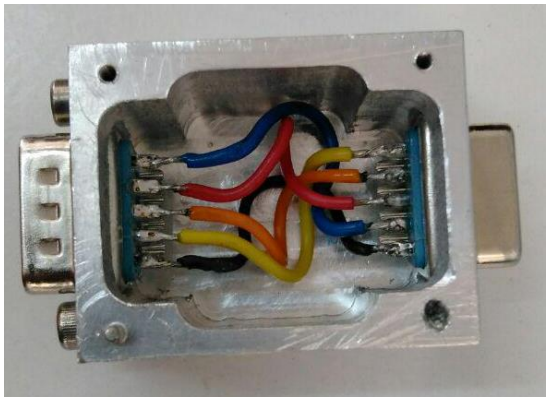


Fonte: Produção da Autora (2016)

Explicando brevemente a figura, o conversor USB-Serial (1) é conectado no conector *cross-over* RS-232 (2), que se conecta no cabo de extensão (3 - 4), o qual é conectado ao outro conversor USB-Serial (5). Cada conversor USB-Serial se conecta em um notebook.

Para testar a comunicação via RS422, utilizou-se o mesmo adaptador USB-serial-RS-232 acrescido de um outro conversor RS-232 – RS422 (conversor modelo COM-422-PE9 da SerialComm). Neste caso a inversão dos pinos é diferencial e a Figura 40 mostra o conector *cross-over* montado, com a pinagem invertida como explicado no capítulo 2 (Figuras 13 e 14).

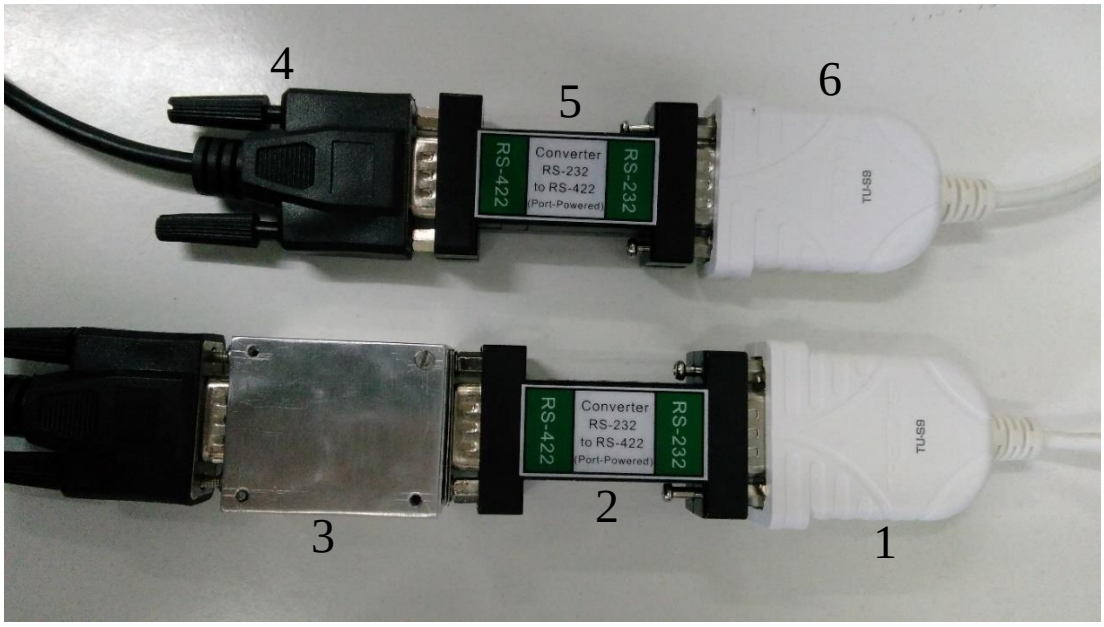
Figura 40 - Conector *cross-over* RS-422



Fonte: Produção da Autora (2016)

A comunicação foi montada conforme mostra a Figura 41.

Figura 41 - Ligação dos cabos e conectores RS-422



Fonte: Produção da Autora (2016)

Explicando brevemente a figura, o conversor USB-Serial (1) é conectado ao conversor RS-232 para RS-422 (2), que se conecta no conector montado para conexão *cross-over* (3), o qual se conecta a um fio de extensão (4), que se conecta a outro conversor RS-422 para RS-232 (5), para que retorne sendo RS-232 e se conecte ao outro conversor USB-Serial (6). Cada conversor USB-Serial se conecta em um notebook.

4- RESULTADOS

Este capítulo apresenta os resultados e discussões do desenvolvimento feito no capítulo 3, demonstra a comunicação entre dois computadores, o envio e o recebimento de mensagens, registro no arquivo em texto e o cumprimento dos requisitos levantados.

4.1- Demonstração da Comunicação Serial

A montagem da comunicação serial, isto é, os dois notebooks conectados pelos adaptadores USB/RS-232 + cabo de extensão + conector *cross-over* + adaptador RS-232/USB, foi testada utilizando o programa HyperTerminal. Neste teste, um texto digitado em um dos notebooks aparece na tela do outro notebook ao qual o primeiro está conectado.

O mesmo teste foi feito na montagem que utiliza o padrão RS-422, isto é, dois notebooks conectados por adaptadores USB/RS-232 + adaptador RS-232/422 + cabo de extensão + conector *cross-over* + adaptador RS-422/232 + adaptador RS-232/USB.

4.2- Demonstração do Envio e Recebimento de Mensagens

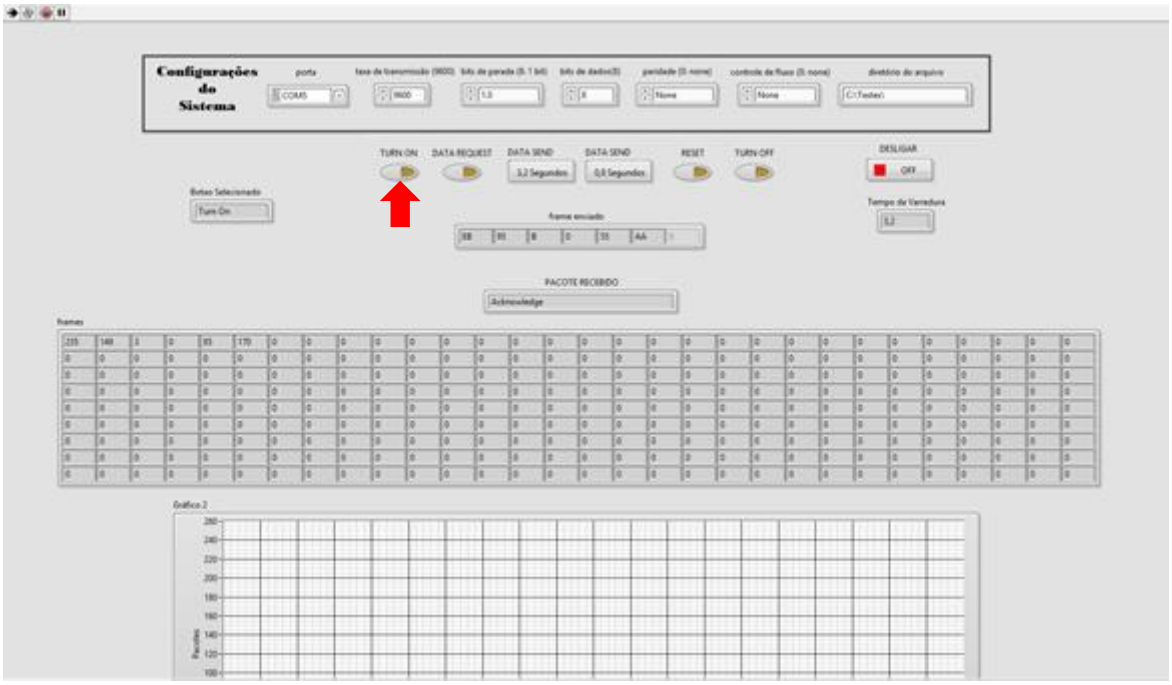
Este subcapítulo mostra a interação do programa em LabVIEW (OBC) com o programa em Python (Experimento) e o arquivo com as mensagens enviadas e recebidas.

4.2.1- Ação dos Botões

O usuário aperta o botão TURN ON (Ligar) no programa em LabVIEW que envia a mensagem *Turn On* para o programa em Python. O experimento recebe a mensagem e envia outra mensagem para o programa em LabVIEW de *acknowledge*, que confirma que ligou. O programa em Python inicia o gerador de mensagens, o qual começa a gerar mensagens a cada 0.8 ou 3.2 segundos, dependendo do tempo de varredura escolhido. O tempo padrão é 3.2s.

A Figura 42 mostra o envio da mensagem de TURN ON e o recebimento do *acknowledge*. A flecha vermelha indica o botão TURN ON.

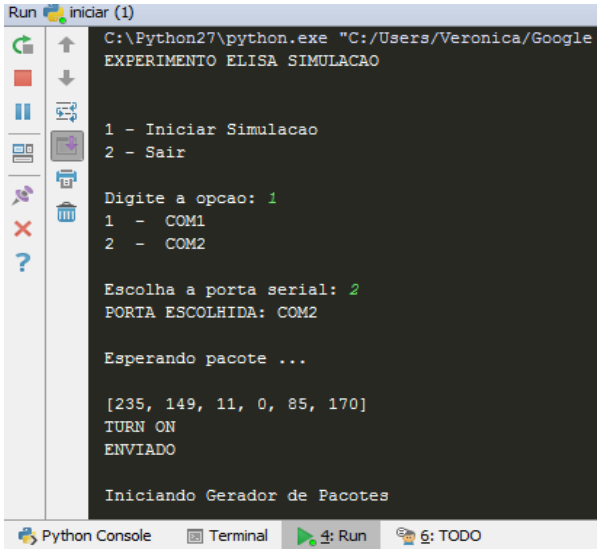
Figura 42 - LabVIEW - botão TURN ON



Fonte: Produção da Autora (2016)

A Figura 43 mostra, no programa em Python, o recebimento da mensagem de TURN ON, a mensagem de que o gerador de mensagem iniciou e a mensagem “ENVIADO” que significa o envio do *acknowledge*.

Figura 43 - Python - Resposta botão TURN ON

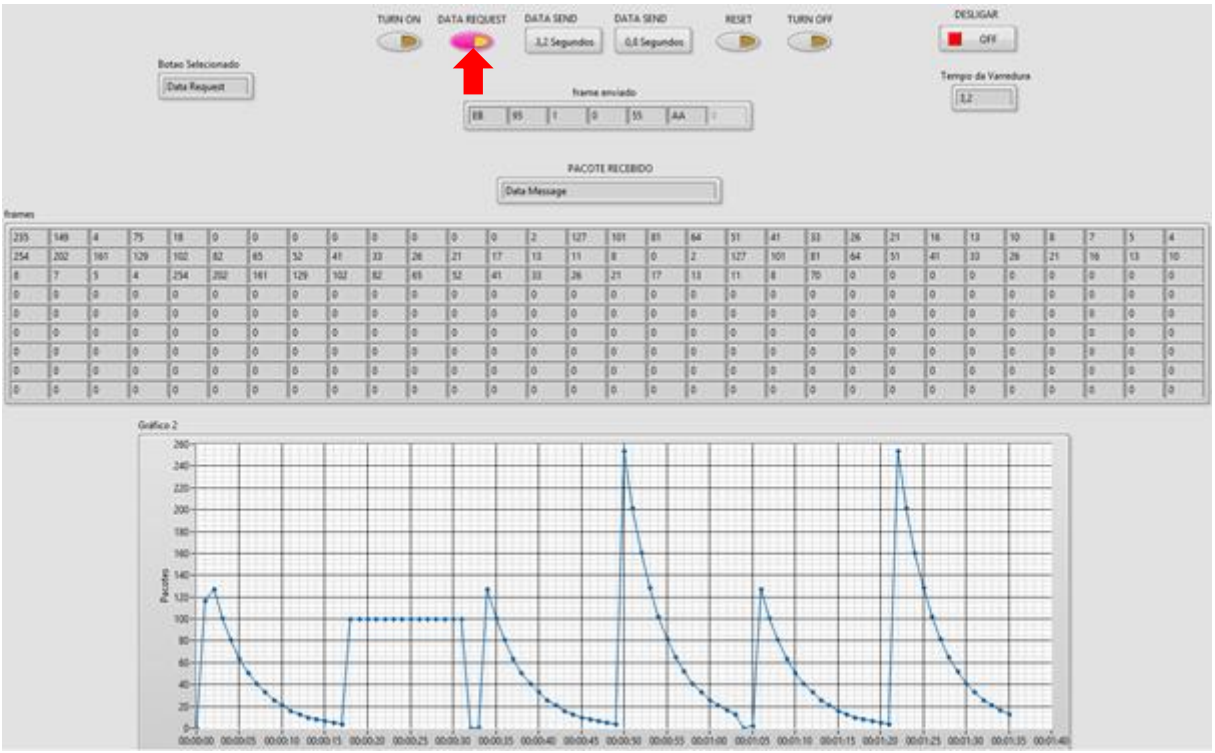


Fonte: Produção da Autora (2016)

O usuário aperta o botão DATA REQUEST (Pedir Dados) no programa em LabVIEW que envia uma mensagem para o programa em Python. O programa em Python então, recebe a mensagem de pedido de dados e começa a enviar as mensagens geradas enquanto o botão estiver habilitado no programa LabVIEW.

A Figura 44 mostra a ação do botão DATA REQUEST, o qual resulta no recebimento de uma mensagem de dados com os dados gerados pelo gerador de mensagens no programa Python e a visualização dos mesmos em forma de gráficos.

Figura 44 - LabVIEW - botão DATA REQUEST



Fonte: Produção da Autora (2016)

Como pode ser visto na imagem, os gráficos das mensagens de *Data Message* decaem exponencialmente, conforme calculado no capítulo anterior.

A Figura 45 mostra a ação do aperto do botão DATA REQUEST, as mensagens que irão ser enviados e a mensagem “Enviado” que significa que enviou para o programa em LabVIEW.

Figura 45 - Python -Resposta botão DATA REQUEST

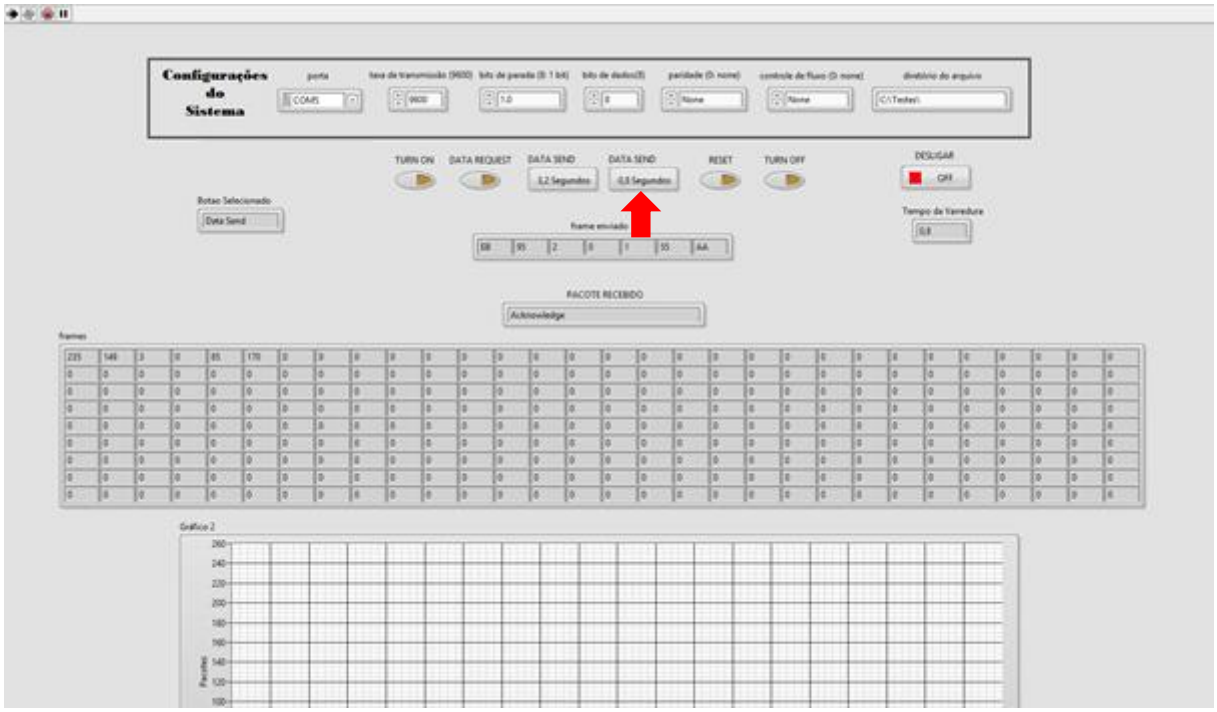
```
[235, 149, 1, 0, 85, 170]
DATA REQUEST
ENVIADO
[235, 149, 4, 91, 0, 0, 0, 0, 0, 0, 0, 0, 0, 117, 127, 101, 81, 64, 51, 41, 33, 26, 21
[235, 149, 1, 0, 85, 170]
DATA REQUEST
ENVIADO
[235, 149, 4, 75, 18, 0, 0, 0, 0, 0, 0, 0, 0, 2, 127, 101, 81, 64, 51, 41, 33, 26, 21,
[235, 149, 1, 0, 85, 170]
DATA REQUEST
ENVIADO
```

Fonte: Produção da Autora (2016)

O usuário aperta o botão DATA SEND (Envio de Dados) no programa em LabVIEW que envia uma mensagem para o experimento contendo 0 ou 1, que significa a mudança do tempo de varredura. O programa em Python recebe a mensagem e muda o tempo de varredura, ou seja, muda o tempo para a criação de mensagens de dados.

A Figura 46 mostra a ação do botão DATA SEND 0.8 segundos, e o recebimento da mensagem de *acknowledge*.

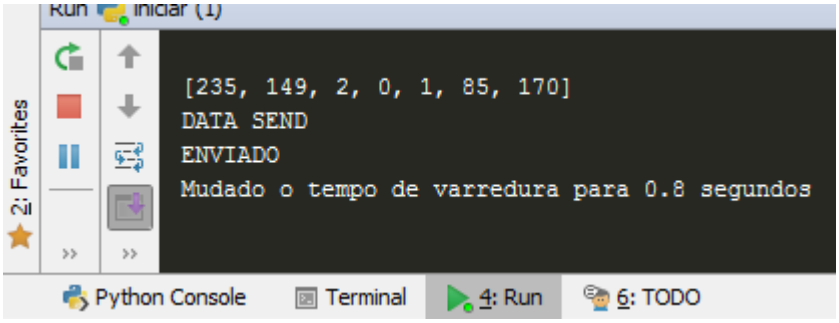
Figura 46 - LabVIEW - botão DATA SEND (0.8)



Fonte: Produção da Autora (2016)

A Figura 47 mostra a ação do aperto do botão DATA SEND 0.8 segundos, a mensagem de mudança do tempo de varredura para 0.8 segundos e a mensagem “ENVIADO” que significa o envio do *acknowledge*.

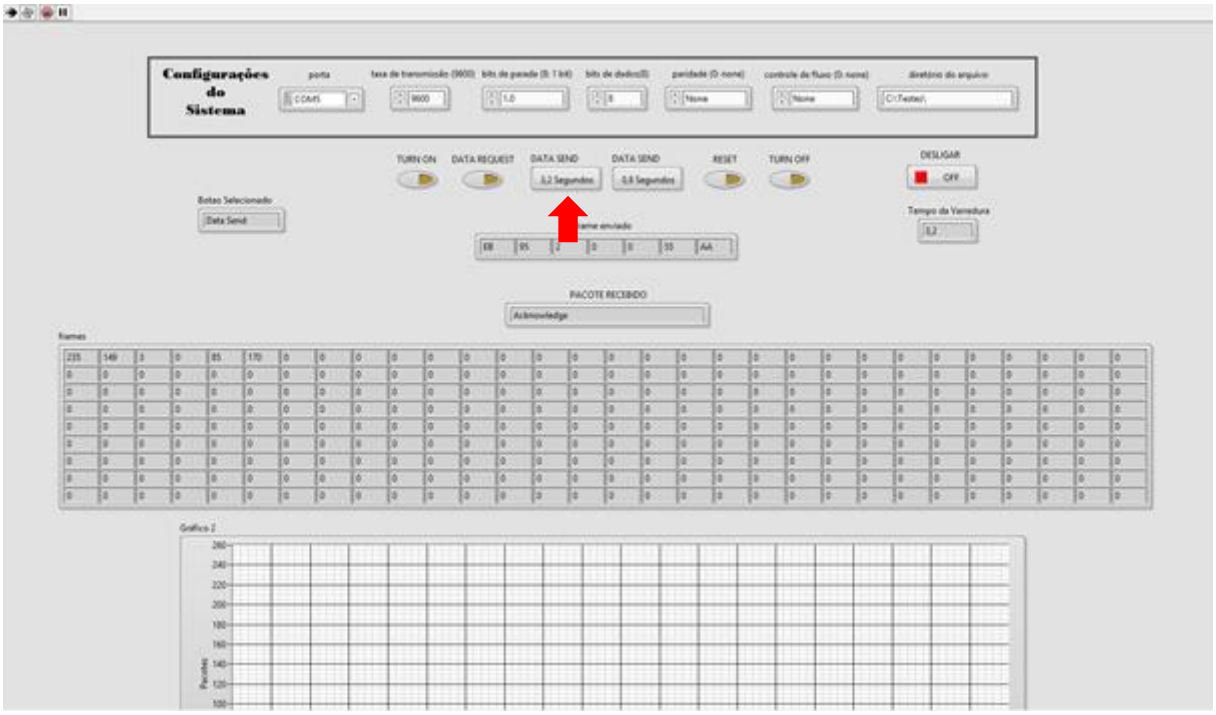
Figura 47 - Python - resposta botão DATA SEND (0.8)



Fonte: Produção da Autora (2016)

A Figura 48 mostra a ação do botão Data Send 3.2 segundos, e o recebimento da mensagem de *acknowledge*.

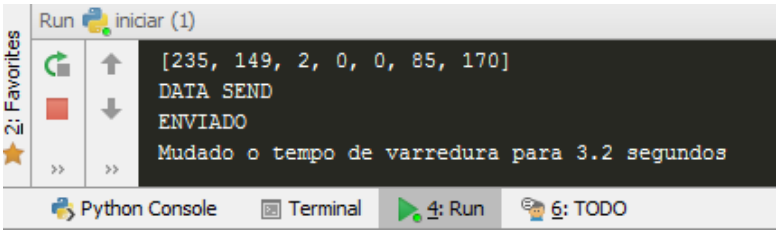
Figura 48 -LabVIEW - botão DATA SEND (3.2)



Fonte: Produção da Autora (2016)

A Figura 49 mostra a ação do aperto do botão DATA SEND 3.2 segundos, a mensagem de mudança do tempo de varredura para 3.2 segundos e a mensagem “ENVIADO” que significa o envio do *acknowledge*.

Figura 49 - Python - Resposta botão DATA SEND (3.2)



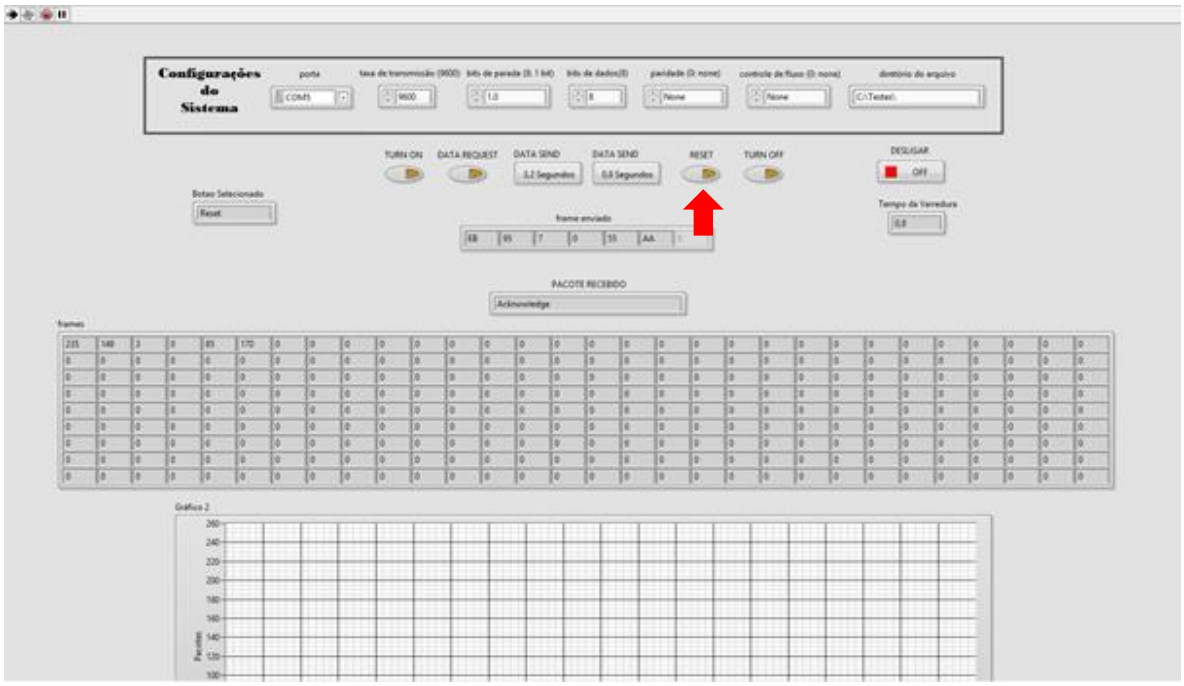
Fonte: Produção da Autora (2016)

O usuário aperta o botão RESET (Reinicia) no programa em LabVIEW, o qual envia uma mensagem para o programa em Python. O programa em Python recebe a mensagem, reinicia o gerador de mensagens e coloca o tempo de varredura com o valor padrão e envia

outra mensagem para o programa em LabVIEW de *acknowledge*, que confirma que retornou com os valores iniciais.

A Figura 50 mostra a ação do botão RESET, e o recebimento do *acknowledge*.

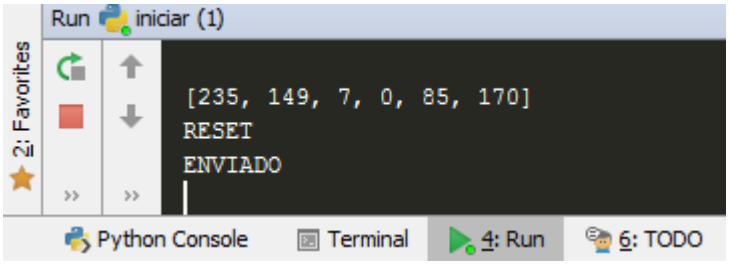
Figura 50 - LabVIEW - botão RESET



Fonte: Produção da Autora (2016)

A Figura 51 mostra a ação do aperto do botão RESET e a mensagem “ENVIADO” que significa o envio do *acknowledge*.

Figura 51 - Python - resposta botão RESET



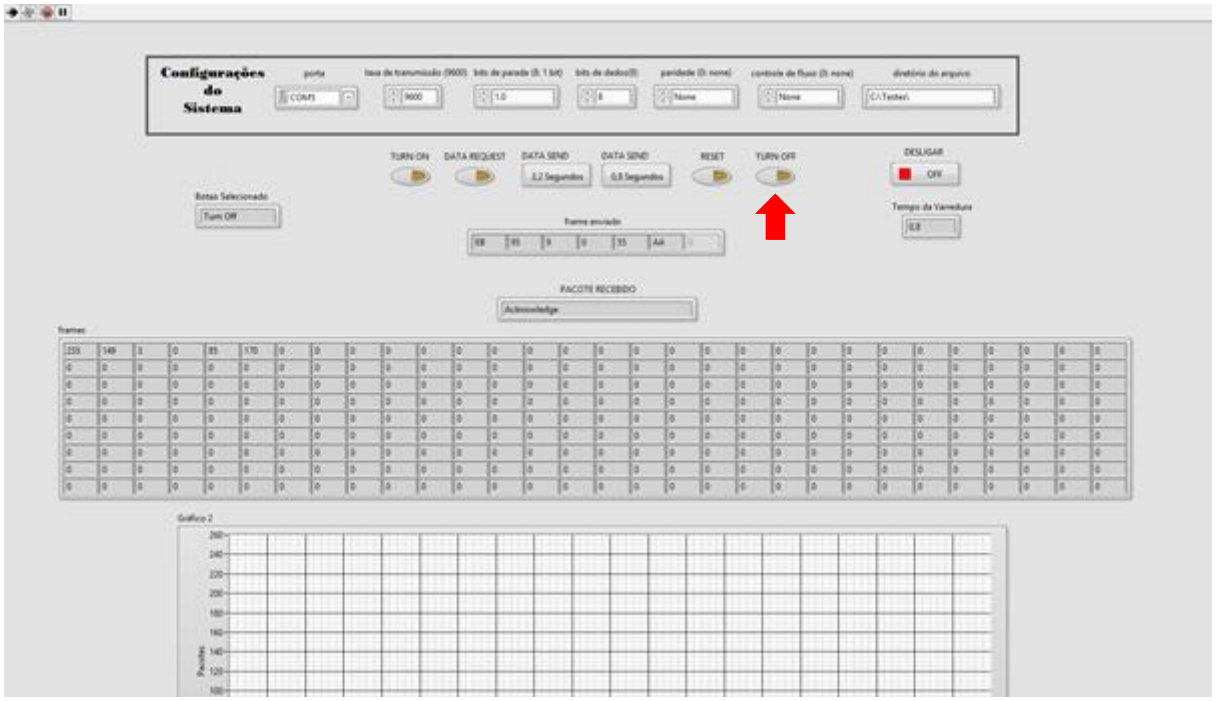
Fonte: Produção da Autora (2016)

O usuário aperta o botão TURN OFF (flecha vermelha) no programa em LabVIEW que envia uma mensagem para o programa em Python. O programa em Python recebe a

mensagem, para que desligue o gerador de mensagens e envia outra mensagem para o programa em LabVIEW de *acknowledge*, confirmando que desligou.

A Figura 52 mostra a ação do botão TURN OFF e o recebimento do *acknowledge*.

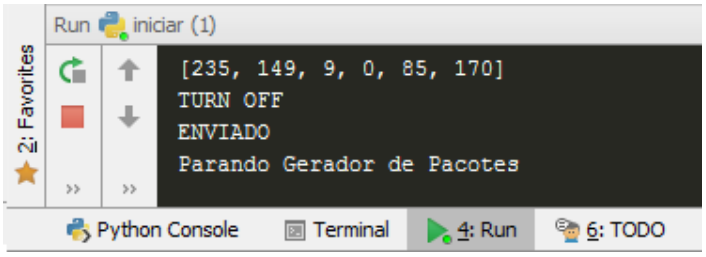
Figura 52 - LabVIEW - Botão TURN OFF



Fonte: Produção da Autora (2016)

A Figura 53 mostra a ação no programa em Python do aperto do botão TURN OFF no LabVIEW, a mensagem da parada do gerador de mensagens e a mensagem “ENVIADO” que significa o envio do acknowledge.

Figura 53 - Python - Resposta botão TURN OFF



Fonte: Produção da Autora (2016)

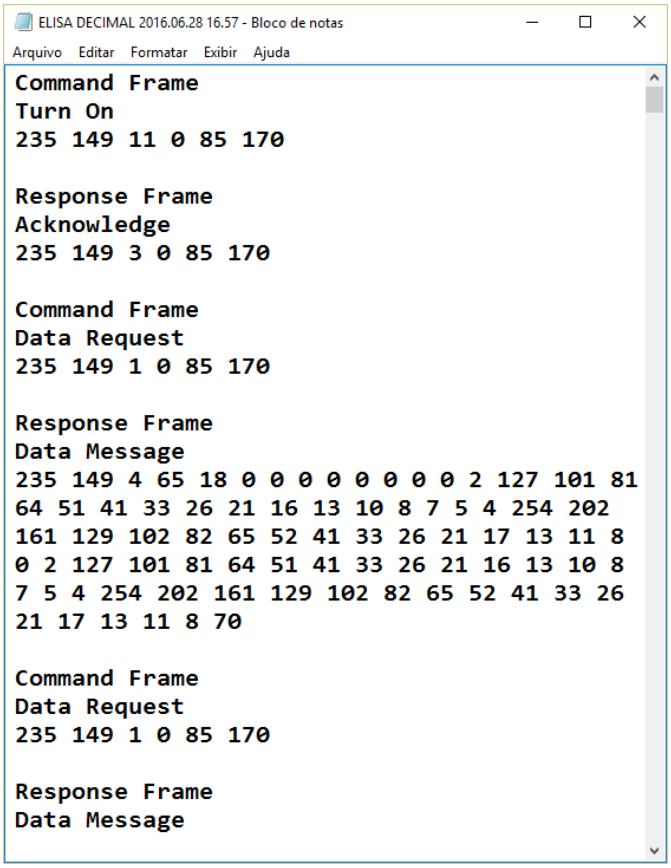
4.2.2- Checagem das mensagens enviadas e recebidas

O cálculo do *checksum* é um dos procedimentos mais importantes ao receber mensagens, pois verifica a integridade dos dados. Porém não foi aplicado ao programa LabVIEW, pois o mesmo apresentou problemas ao calcular em tempo real.

Para testar a integridade das mensagens enviadas e recebidas foi checado visualmente os arquivos de texto que são criados pelo programa em LabVIEW com todos as mensagens de entrada e saída.

A Figura 54 mostra o arquivo de texto em decimal com as mensagens enviadas e recebidas.

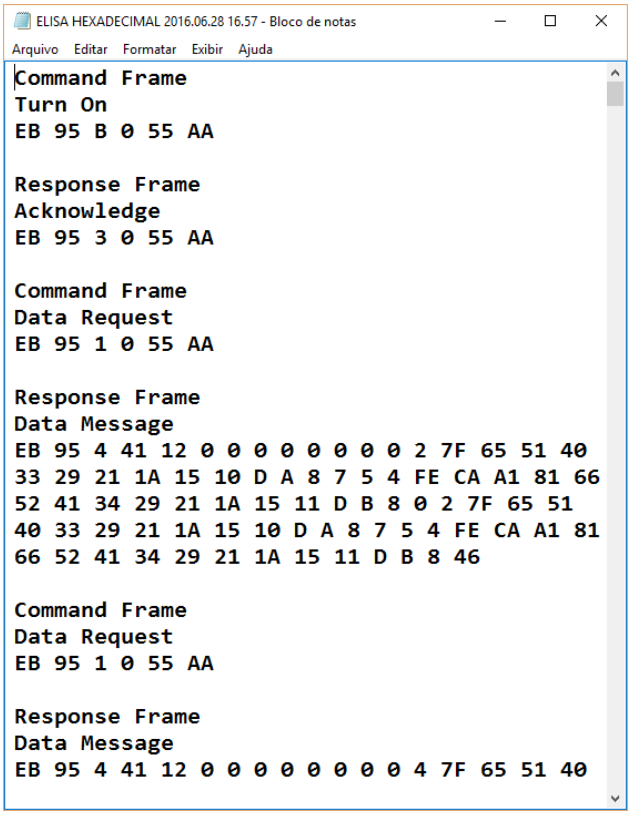
Figura 54 - Arquivo de texto em decimal



Fonte: Produção da Autora (2016)

A Figura 55 mostra o arquivo de texto em hexadecimal com todas as mensagens enviadas e recebidas.

Figura 55 - Arquivo de texto em hexadecimal



Fonte: Produção da Autora (2016)

A Tabela 14 apresenta os testes realizados em decimal.

Tabela 13 - Tabela de teste das mensagens recebidos

Mensagem enviado	Mensagem Esperada	Mensagem Recebida
235 149 11 0 85 170	235 149 3 0 85 170	235 149 3 0 85 170
235 149 2 0 0 85	235 149 3 0 85 170	235 149 3 0 85 170
235 149 2 0 1 85	235 149 3 0 85 170	235 149 3 0 85 170
235 149 7 0 85 170	235 149 3 0 85 170	235 149 3 0 85 170
235 149 9 0 85 170	235 149 3 0 85 170	235 149 3 0 85 170
235 149 1 0 85 170	235 149 4 91 0 0 0 0 0 0 0 0 0 117 127 101 81 64 51 41 33 26 21 16 13 10 8 7 5 4	235 149 4 91 0 0 0 0 0 0 0 0 0 117 127 101 81 64 51 41 33 26 21 16 13 10 8 7 5 4

	100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 0 0 127 101 81 64 51 41 33 26 21 16 13 10 8 7 5 4 254 202 161 129 102 82 65 52 41 33 26 21 17 13 11 8 118	100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 0 0 127 101 81 64 51 41 33 26 21 16 13 10 8 7 5 4 254 202 161 129 102 82 65 52 41 33 26 21 17 13 11 8 118
235 149 1 0 85 170	235 149 4 75 18 0 0 0 0 0 0 0 0 2 127 101 81 64 51 41 33 26 21 16 13 10 8 7 5 4 254 202 161 129 102 82 65 52 41 33 26 21 17 13 11 8 0 2 127 101 81 64 51 41 33 26 21 16 13 10 8 7 5 4 254 202 161 129 102 82 65 52 41 33 26 21 17 13 11 8 70	235 149 4 75 18 0 0 0 0 0 0 0 0 2 127 101 81 64 51 41 33 26 21 16 13 10 8 7 5 4 254 202 161 129 102 82 65 52 41 33 26 21 17 13 11 8 0 2 127 101 81 64 51 41 33 26 21 16 13 10 8 7 5 4 254 202 161 129 102 82 65 52 41 33 26 21 17 13 11 8 70
235 149 1 0 85 170	235 149 4 41 18 0 0 0 0 0 0 0 0 3 127 101 81 64 51 41 33 26 21 16 13 10 8 7 5 4 254 202 161 129 102 82 65 52 41 33 26 21 17 13 11 8 36	235 149 4 41 18 0 0 0 0 0 0 0 0 3 127 101 81 64 51 41 33 26 21 16 13 10 8 7 5 4 254 202 161 129 102 82 65 52 41 33 26 21 17 13 11 8 36

Com base nos testes, os seguintes requisitos foram satisfeitos:

- ✓ Testes de comunicação serial entre os dispositivos (notebooks) seguindo o protocolo de comunicação;
- ✓ Simulação do computador de bordo (programa em LabVIEW); e
- ✓ Simulação do circuito de interface do experimento ELISA (programa em Python).

5- CONCLUSÃO

O desenvolvimento dos programas irá ajudar nos testes de validação do circuito de interface do experimento ELISA que será lançado no satélite EQUARS para pesquisas de perturbações aeronomicas em especial na zona onde ocorre a Anomalia Magnética do Atlântico Sul. Os testes feitos no experimento são de extrema importância pois garantem um perfeito comportamento no espaço.

O objetivo principal do trabalho foi realizar a comunicação serial entre um software desenvolvido em linguagem Python, simulando o experimento, e um desenvolvido no LabVIEW, simulando o computador de bordo.

O desenvolvimento do programa na linguagem Python foi realizado para simular as respostas que a interface do experimento ELISA envia ao computador de bordo, já o programa no LabVIEW foi feito para simular o computador de bordo ao receber mensagens de dados do experimento e enviar telecomandos.

Os testes ocorreram em dois computadores distintos, cada um contendo um programa, e a comunicação entre eles foi realizada utilizando cabo serial, conversores USB-Serial e conectores com a pinagem específica para que ocorra a comunicação seguindo o protocolo de comunicação.

Os resultados obtidos com os testes atingiram todos os objetivos levantados no capítulo 1. Os programas produzidos estão prontos para o teste real com o circuito de interface que será construído.

6- TRABALHOS FUTUROS

O programa desenvolvido em LabVIEW deverá ser utilizado para testar o circuito de interface do experimento ELISA, quando este for construído. Estes testes serão muito importantes especialmente durante a fase de qualificação espacial do experimento, em que todos os circuitos eletrônicos são submetidos a testes de vibração e de operação em ambiente de vácuo e transiente térmico.

REFERÊNCIAS

(SILVA, et. Al, 2013) SILVA, Bruno Saraiva; PEREIRA, Maurício de Castro e BUCHMANN, Rafael Mazza. Padrões de Comunicação Serial Clássicos: RS-232, RS-422 e RS-485, pp 2 a 19.

(HSW, 2014) TYSON, Jeff. HOW STUFF WORKS, COMO TUDO FUNCIONA. Disponível em: <http://tecnologia.hsw.uol.com.br/criptografia7.htm>. Acesso em: 02/2016.

(ETUTORIALS, 2016) ETUTORIALS.ORG. Disponível em: <http://etutorials.org/Misc/pc+hardware/Chapter+22.+Serial+Communications/22.3+Serial+Cables/>. Acesso em: 05/2016.

(INPE, 2006) INPE, 2006. EQUARS - Equatorial Atmosphere Research Satellite. Disponível em: <http://www.laser.inpe.br/equars/>. Acesso em: 12/2015.

(ELETRONICA, 1999) ELETRONICA – UGR. Disponível em: < http://electronica.ugr.es/~amroldan/modulos/temas_tecnicos/rs232/>. Acesso em: 06/2016.

(FOROUZAN, 2008) Forouzan, Behrouz A. Comunicação de Dados e Redes de Computadores. Quarta Edição 2008, pp 3 a 8.

(MAZIDI, 2009) MAZIDI, Muhammad Ali; MAZIDI, Janice Gillispie. The 8051 Microcontroller na Embedded Systems, pp. 358 a 370.

(REGAZZI, et. al, 2005) REGAZZI, Rogério Dias; PEREIRA, Paulo Sérgio e SILVA JR, Manoel Feliciano. Soluções Práticas de Instrumentação e Automação - Utilizando a Programação Gráfica LabVIEW, pp. 2.

(PYSCIENCE, 2008) PYSCIENCE. PyScience – Brasil. Disponível em: <http://pyscience-brasil.wikidot.com/python:python-oq-e-pq>. Acesso em: 05/2016.

(PYTHON, 2016) PYTHON. Disponível em: <https://www.python.org/about/>. Acesso em: 05/2016.

(FORTESCUE, 2011) P. Fortescue, G. Swinerd, J. Stark, editors, Spacecraft Systems Engineering, capítulo 2, 2011

(ICD, 2015) ICD (Interface Control Document) do Experimento ELISA. Biblioteca INPE, 2015.

(TAN, 2011) Tan, Ing. Construção da Interface (ELIF) do Experimento ELISA, parte integrante do Micro-satelite Cientifico LATTES, Projeto Básico.

(RAMOS, 2015) RAMOS, Allan. MVC – Afinal, o que é?. Disponível em: <http://tableless.com.br/mvc-afinal-e-o-que/>. Acesso em: 05/2016.

(PYSERIAL, 2015) PYSERIAL. Disponível em: <http://pythonhosted.org/pyserial/pyserial.html>. Acesso em: 05/2016.

(NATIONAL INSTRUMENTS, 2015) NATIONAL INSTRUMENTS, 2015. Disponível em: <http://digital.ni.com/public.nsf/allkb/32679C566F4B9700862576A20051FE8F>. Acesso em: 10/2015.

(WUEST, 2007) M. Wüest, D. Evans, J. Mc Fadden, W. Kasprzak, L. Brace, B. Dichter, W. Hoegy, A. Lazarus, A. Masson, O. Vaisberg, *Review of Instruments*, in: M. Wüest, D. Evans, R. von Steiger (Eds), *Calibration of Particle Instruments in Space Physics*. The International Space Science Institute, ESA Communications (2007), pp. 50-52

(PMM, 2001) A820000-SPC-002 V06 MMP DESIGN & CONSTRUCTION SPECIFICATION. Multi-mission platform - Design and construction Specification, 2001, pp. 27 a 31.

(ITASAT, 2015) ITA, 2015. Disponível em: <http://www.itasat.ita.br/?q=content/egse>. Acesso em: 03/2016.

(ESC AEROSPACE) ESC AEROSPACE. Disponível em: < http://www.esc-aerospace.com/?page_id=458>. Acesso em: 03/2016.

(ROGERCOM, 2006) ROGERCOM.COM. Disponível em:
< http://www.rogercom.com/PortaSerial/PortaSerial.htm>. Acesso em: 06/2016.

(B&BELETRONICS, 2003) B&BEletronics, Make The Right Connections.RS-422 Converter Connections for Extending RS-232.

(SERIALCOMM) SerialComm, Data Conversion Experts. Datasheet dos conversores RS-422.

(TRENDNET) Trendnet. Disponível em: <http://www.trendnet.com/products/USB-adapters/TU-S9>. Acesso em: 11/2015.

(DEVMEDIA) DEVMEDIA.COM.BR, Artigo Java Magazine 77 - Grails: do Groovy à Web – Parte 3. Disponível em: www.devmedia.com.br/ artigo-java-magazine-77-grails-do-groovy-a-web-parte-3/16160. Acesso em: 03/2016.