Resumo projetos Camada Física

TX = write (escreve)

RX = read (leitura)

Projeto loop back

Intuito do projeto: realizar um envio de uma imagem atraves da porta rx e, após a recepção da imagem, duplicá-la e salvá-la por meio do tx.

* Sobre a interface física:

Possui baudrate de 9600, o byte size é de 8, entretanto possui apenas um start bit e um stop bit (não possui parity bit). Dessa forma em cada pacote o byte size é de 10 bits.

* Sobre a aplicação:

Txbuffer = armazena os dados a serem enviados

* Sobre enlaceTx:

A **def thread** fica rodando enquanto **threadStop** não for **True**, ou seja, **def threadKill** não for ativada. Assim, ele fica num while que, se **threadMutex** for **True,** ou seja, se a aplicação estiver rodando e recebendo informações, ele fica escrevendo oque foi colocado na variável Txbuffer.

**def Thread:** é uma função viva, a qual fica fora do processador, permitindo que ela envie informações sem depender do resto, rodando constantemente fora do software.

Mesmo sendo fora da aplicação ela consegue complartilhar variáveis e informações

A **def sendBuffer** recebe o **data** enviado pela aplicação->enlace, o qual, nesse caso é o tamanho da imagem, e manda para a **def Thread** para escrever as informações

**def sendBuffer:** está fora da variável viva, então ele salva o data em uma variável em comum e ativa o threadMutex como True para que seja possível rodar na def Thread

* Sobre enlaceRX:

Possui também uma função viva (**def Thread**), o qual, enquanto a função não for parada, ele gera um while o qual adiciona os dados que foram lidos pela função em self.buffer.

Assim em buffer, estão todos os dados que foram lidos pela função.

**def getBufferLen** pega o comprimento do dado lido

**def getAllBuffer** pausa a def Thread e pega todo o dado que estava armazenado em buffer, após isso ele apaga o buffer e continua a Thread

**def getNData** é uma das funções mais importantes a qual por meio do size da informação passada por meio de argumento na aplicação, ela consegue chamar a **def getBuffer**, a qual seleciona qual o comprimento de informação desejado pelo client, atualiza o buffer para não ocorrer a repetição de dados. Assim, a getNData é chamada pelo enlace na **def getData** para retornar os dados pedidos pelo client.

* Sobre enlace:

**def sendData**: a aplicação chama a def para enviar o tamanho do arquivo, entretanto essa def se localiza no enlace, assim ela por meio da linha “self.tx.sendBuffer(data)” chama o sendBuffer do enlaceTX para começar a leitura de dados no tx

**def** **getData**: a função é chamada na aplicação recebendo como argumento o tamanho do arquivo em int que deseja ser puxado, assim essa função localizada no enlace, puxa o data vindo do enlaceRX pela função getNData, por meio da linha “data = self.rx.getNData(size)”, retornando o data em si e seu comprimento.

Possui as funções **enable** e **disable** as quais ativam e desativam as portas conectoras do arduino

Projeto Client-Server

Intuito do projeto: diferentemente do loop back, nesse projeto a aplicação é dividida em duas partes, sendo uma para o client e outra para o server.Primeiramente o client deve enviar via trasmissão uart uma imagem, assim o server deve resceber a mesma, salvar o arquivo recebido e enviar de volta para o client, assegurando-se que os numeros de bits estão corretos e que a transmissão foi bem sucedidas.

* Sobre a aplicação-client e a aplicação-server:

Imagem escolhida pelo client e armazenada na variável **txbuffer**, assim esse tamanho em bytes é enviado por meio do **sendData**, o qual esta localizado no enlace que chama a **def sendBuffer**, localizado no enlaceTx. O **sendBuffer** juntamente com a **def Thread** realizam a escrita dos dados que chegaram e guardam eles na **transLen**.

Assim, o server chama esses dados por meio do **getData**, o qual se localiza no enlace, que puxa os dados lidos do enlaceRX; assim o tamanho da imagem é salvo no server para que a imagem possa ser copiada e em seguida enviado novamente pra o client.

Após isso os dois valores de comprimento de imagem são comparados para averiguar se a imagem foi transmitida com sucesso.

Projeto Fragmentação, hand-shake e datagrama

Intuito do projeto: definição de uma protocolo, onde não pode enviar nenhum dado sem a existência de um datagrama

* Datagrama:

Sempre quando for enviar um pacote de dados é necessário tem um **HEAD**, o qual possui informações sobre o payload e o dado, possuindo sem 10 bytes. Em segundo lugar é necessário enviar um **PAYLOAD**, o qual varia em seu tamanho dependendo do seu tipo de pacote (nesse trabalho ele variava de 0-128 bytes) e carrega os dados que o client deseja transmitir. E, por fim o **EOP**, o qual é o end of packge, possuindo 4 bytes, os quais são valores fixos definidos pelo programador.

* Handshake:

Também conhecido como o contato inicial entre o client e o server, trata-se da confirmação enviada pelo client para ter certeza que o server está pronto para receber as informações. Caso ele esteja, o mesmo envia uma confirmação para o client avisando que o envio de dados pode começar.

Caso essa resposta não seja recebida depois de um período de tempo determinado, o client deve enviar outro handshake caso queria tentar novamente o contato, ou, caso contrário, encerrar a aplicação.

* Fragmentação:

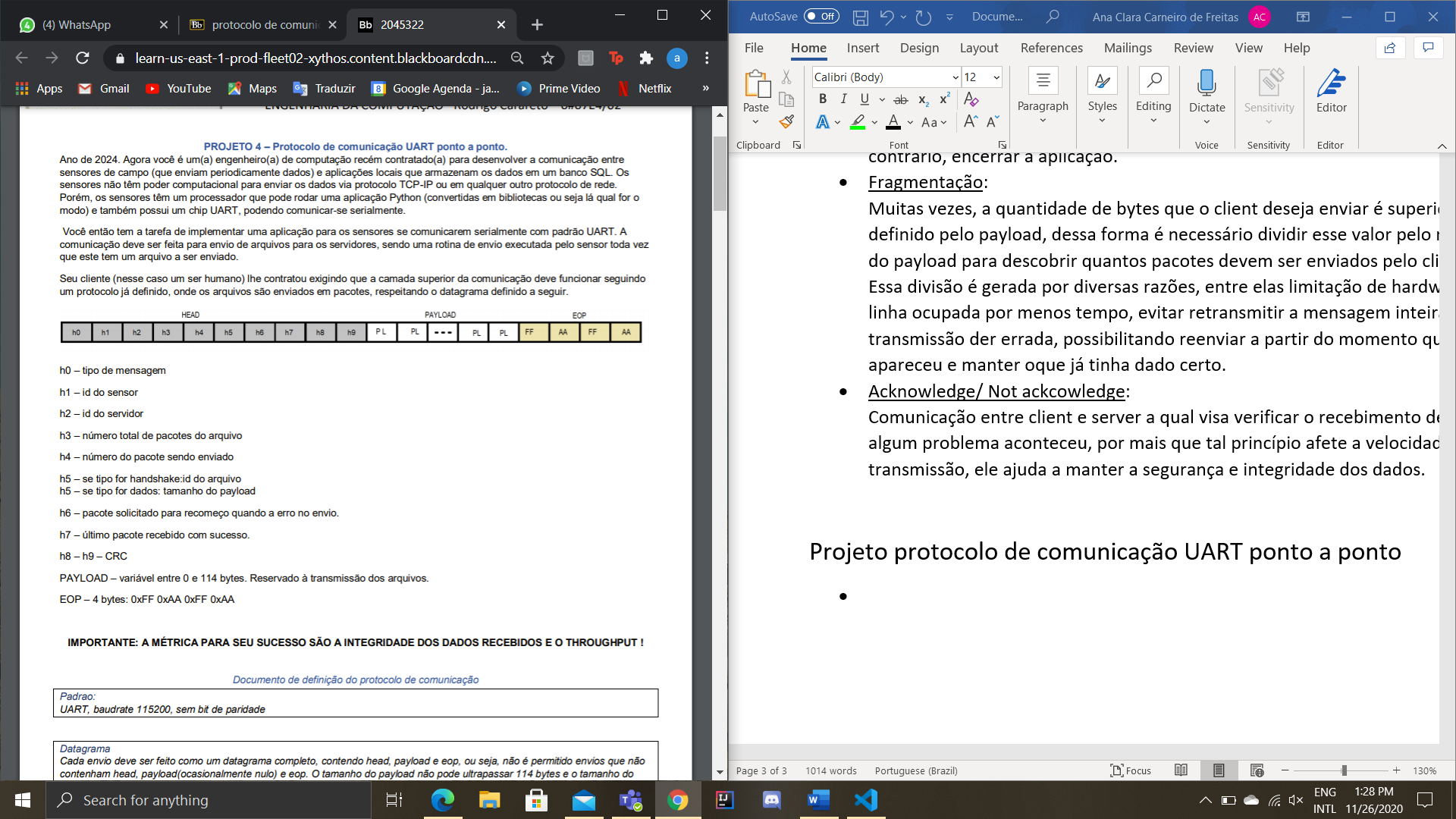
Muitas vezes, a quantidade de bytes que o client deseja enviar é superior ao número definido pelo payload, dessa forma é necessário dividir esse valor pelo numero máximo do payload para descobrir quantos pacotes devem ser enviados pelo client.

Essa divisão é gerada por diversas razões, entre elas limitação de hardware, mandar a linha ocupada por menos tempo, evitar retransmitir a mensagem inteira quando a transmissão der errada, possibilitando reenviar a partir do momento que o erro apareceu e manter oque já tinha dado certo.

* Acknowledge/ Not ackcowledge:

Comunicação entre client e server a qual visa verificar o recebimento de pacotes ou se algum problema aconteceu, por mais que tal princípio afete a velocidade da transmissão, ele ajuda a manter a segurança e integridade dos dados.

Projeto protocolo de comunicação UART ponto a ponto



Serialização UART

Nos outros projetos anteriores que possuiam ligação entre arduinos, não foi preciso fazer esse código em c++ pois o próprio chip uart do arduino ou o codigo em python já faziam isso, como por exemplo escrever a onda da letra a

* sw\_uart.h (aparentemente é tipo uma interface física)
* sw\_uart.ccp

O due\_sw\_uart\_TX para escrever um caracter chama a função **sw\_uart\_write\_string** localizada dentro do sw\_uart.ccp, a qual recebe o caracter a ser transformado e o envia para **sw\_uart\_write\_data**. Tal função faz um for que fica em looping durante o comprimento da string, ou seja, ele fica mandando mandando informações para **sw\_uart\_write\_byte** até acabar a string desejada.

Assim, quando o caracter for enviado a função write\_byte vai escrever a informação considerando o startbit, parity bit e stop bit. Além do fato de que ele espera meio período após o startbit para comecar a escrever as informaçoes.

Dentro dessa função existe a **calc\_even\_parity** o qual verifica se a paridade enviada foi a mesma que a recebida, assim se for um valor par significa que a paridade é 1, se for impar significa que a paridade é 0. Assim, essa verificação é feita por quem envia e quem recebe e ambos os valores tem que coincidir.

Após isso a função **sw\_uart\_receive\_byte** é chamada para receber os dados, assim, enquanto o pin\_RX estiver HIGH ou seja estiver recebendo informações para a escrita dos dados, a função vai ficar rodando. Esta checa se apareceu o start bit, depois recebe os dados respectivos ao caracter, depois ve se ter paridade ou não e por fim o stop bit. Se tudo der certo, data = aux e a função retorna o SW\_UART\_SUCCESS, o qaul printa na tela o caracter certo enviado pelo tx

Projeto DTMF (dual tone multi frequency)

Intuito do projeto: software que gera o som de uma tecla de celular ao ser pressionado, esse sinal é composto por **duas frequências**, ou seja, duas senoides as quais serão somadas por meio do Fourier

* emissor:

por meio do generateSin foi gerado dois senos com as respectivas frequencias sonoras, após isso a parte plotada em y das duas senoides escolhidas eram somadas para representar o sinal captado

* receptor:

recebe o audio enviado pelo emissor e faz o fourrier nele para descobrir as frequencias dominantes, no caso deveriam ser dois picos dominantes por serem dois senos.

Após isso, com o apoio do peakutils foi possível descobrir os indexes e a frequencia dos picos e, com isso, descobrir qual tecla teria sido pressionada

Projeto Modulação AM

Conceito importante: a modulação AM consiste na mudança da amplitude da onda sonora por meio da multiplicação com a carrier, que é uma onda de frequencia maior que a do sinal emitido.

* Normalização do áudio:

Para realizar a normalização do audio é necessário multiplicá-lo por uma constante a qual permita deixá-lo dentro do range desejado. Para isso foi encontrado o maior e o menor valor atingidos pelo audio e, em seguida, escolher o mais abrangente. Após isso essa constante foi dividida por 1 e multiplicada pelo audio.

* Áudio filtrado:

Foi realizado um filtro passa baixo com o intuito de filtrar todas as frequencias acima do valor estipulado (sem essas frequencias mais altas o som tende a ficar mais opaco)

* Modulação do áudio:

Para realizar a modulação do áudio, o mesmo foi multiplicado pela portadora

* Demodulação do áudio:

Para demodulá-lo foi realizado novamente a multiplicação pela portadora

Projeto modulação FM e Digital

* Transmissor:

Primeiramente é escolhido um texto para realizar a modulação, assim um socket é aberto e modula a mensagem recebida transformando-a em bytes. Assim, esses bytes são codificados em fase, sendo assim cada bit uma fase, os quais são representados na constelação, a qual divide em 0s e 1s.

Assim, a partir do momento que essa onda possui suas inversões de fase, o trasmissor transmite ela modulada por meio de um áudio.

* Receptor:

Assim, o receptor irá receber o audio transmitido e realizar sua demodulação, decodificação e analisar suas variações de fase, a fim de descobrir o que é 0 e o que é 1.

Após isso, ele realiza a constelação com as fases definidas, decodifica os 0s e 1s para caracteres e transmite a mensagem de forma digital.