Universidade Federal de Pernambuco
Departamento de Eletrônica e Sistemas

ES455 - Desenvolvimento de Sistemas Embarcados de Tempo Real — 1º Semestre de 2024
Prof. Hermano Cabral — 23/maio/2024

# 1ª Atividade

## 1 Introdução

Em uma comunicação serial, não existe uma fronteira entre grupos de bytes, já que o que é recebido é uma sequência de bytes ininterrupta. Isso não é um problema se os dados que estamos interessados consistem de bytes e a única relação entre eles é de ordem temporal, ou seja, quem veio primeiro. Esse é o caso, por exemplo, se estamos transmitindo os dados de um sensor onde cada dado é exatamente um byte. Mesmo que a recepção e interpretação desses dados comece no meio de uma transmissão, os dados subsequentes recebidos serão corretamente decodificados, independente dos outros bytes não-recebidos.

Considere, por outro lado, o caso em que há $n$ sensores, cujos dados, novamente de comprimento de um byte cada, devem ser transmitidos por comunicação serial na ordem dos sensores, ou seja, transmitimos primeiro o dado do primeiro sensor, depois o do segundo e assim por diante, até chegar ao último sensor, após o qual o processo se reinicia transmitindo um novo dado do primeiro sensor. Neste caso, se a recepção dos dados se inicia de forma não-sincronizada à transmissão, o primeiro byte recebido será de um sensor desconhecido, e portanto não saberemos como interpretá-lo.

Algo similar acontece quando os dados de um sensor consistem de mais de um byte cada. Digamos que o sensor é um conversor A/D que produz 2 bytes para cada amostra. Assuma ainda que o conversor opera de forma contínua, e que os dados gerados são enviados por uma comunicação serial. Se a recepção for iniciada em um tempo arbitrário, há uma chance de 50% do primeiro byte recebido ser o segundo byte do conjunto de dois bytes referentes a um dado. Se isso ocorrer, e não há como detectar caso ocorra, a interpretação desse dado e de todos os seguintes será feita de forma errônea.

O problema em ambos os casos ocorre devido a uma falta de sincronia entre transmissão e recepção que é intrínseca na comunicação serial. Existem várias formas de resolver este problema, desde usando canais *out-of-band* (como os pares de sinais RTS/CLS e DTS/DSR) ou caracteres especiais (como o XON/XOFF) para o controle de fluxo de dados até o uso de um caractere especial, denominado de caractere de sincronismo, para indicar a separação entre duas ou mais partes dos dados, chamados de pacotes.

Com o controle de fluxo, o receptor pode instruir o transmissor para só começar uma transmissão de um pacote de dados quando sinalizado. Assim, o receptor estará preparado para ler os dados do pacote desde o início, sendo assim capaz de decodificar corretamente o pacote. Observe que, nesta solução, a ação parte do receptor que controla o fluxo de dados. O transmissor deve responder ao pedido de início de transmissão enviando um pacote de dados desde o início, ou seja, ele deve descartar quaisquer bytes de pacotes

Por outro lado, o caractere de sincronismo pode ser usado pelo transmissor para indicar o início de um novo pacote de dados. O receptor então só tem que esperar um caractere de sincronismo aparecer

para identificar corretamente o início do pacote. Uma vez identificado o início, a decodificação dos dados pode ocorrer de forma correta. É óbvio que os bytes recebidos antes do caractere de sincronismo sofrem do mesmo problema de interpretação mencionado acima.

Neste exercício iremos implementar duas soluções, a do caractere de sincronismo e a dos caracteres especiais XON/XOFF para controle de fluxo por software. Iremos nos basear em algumas ideias do protocolo HLDC (do termo em inglês *High Level Data Link Control*), em particular para os valores dos caracteres usados para estas funções.

## 1.1 Codificando caracteres especiais

Assumindo que os dados transmitidos por comunicação são binários, ou seja, cada byte pode assumir qualquer valor entre 0 e 255, imediatamente vemos que um problema aparece para definirmos caracteres com sentidos especiais, como o de sincronismo ou os de XON/XOFF. Qualquer valor que escolhermos será um dado válido. Nesta situação, o que se comumente faz, inclusive no protocolo HDLC, é definir um *caractere de escape*, que é usado como prefixo aos caracteres especiais. Observe que o próprio caractere de escape é um valor válido para o dado transmitido, e por isso ele também deve ser tratado como caractere especial.

Devemos então definir um procedimento especial, chamado de *byte stuffing*, que deve ser usado para evitar conflito ao receber algum dos caracteres especiais. O procedimento consiste dos seguintes passos:

- Se se deseja enviar o caractere de sincronismo para finalizar o grupo de butes anteriores para começar outro, devemos transmitir o valor referente a ele.
- Se, na sequência de dados sendo transmitida, aparecer os valores dos caracteres de sincronismo ou de escape, devemos transmitir dois bytes, onde o primeiro é o caractere de escape, e depois o caractere em questão.
- Para os caracteres XON/XOFF, devemos primeiro transmitir o caractere de escape, e depois o caractere em questão.

Observe que esses caracteres extras são adicionados à sequência de dados, de modo que a sequência recebida deve ser decodificada para obtenção da sequência de dados original. As regras acima garantem que sempre seremos capazes de decodificar a sequência de dados recebida de forma inequívoca, seguindo os passos abaixo em um processo chamado *byte destuffing*:

- No início da recepção, para sincronizar, busque por um caractere de sincronismo que não seja precedido por um caractere de escape. Esse caractere denotará o início de um pacote de bytes. Descarte este caractere de sincronismo.
- Após este sincronismo inicial, para cada byte recebido, verifique se ele é um caractere de:

  **sincronismo** Neste caso, informe a finalização do pacote anterior e início de novo pacote. Descarte o caractere.

  **escape** Descarte o caractere e analise o próximo byte recebido.

  - Caso seja o caractere XOFF ou XON, pare ou recomece, respectivamente, o envio de dados. Não retorne o caractere.
  - Se for algum outro caractere, retorne-o como o próximo byte da sequência transmitida.

Existem várias formas de informar da recepção de um caractere de sincronismo ou de XON/XOFF. Na próxima seção definiremos uma delas.

# 2 Descrição da atividade

Você deve implementar funções para transmissão e recepção por comunicação serial com um caractere de sincronismo e dois caracteres de controle de fluxo, XON/XOFF. O caractere de sincronismo serve para delimitar os pacotes, i.e., a recepção de um caractere de sincronismo representa o encerramento da recepção dos bytes de um pacote e início da recepção dos bytes de um novo pacote.

Para os valores numéricos destes caracteres, use os valores-padrão estabelecidos pelos protocolo HDLC e RS232, que são 0x7e para o caractere de sincronismo, 0x11 para XON e 0x13 para XOFF. Seguindo HDLC, use o caractere de escape 0x7d como prefixo para os caracteres anteriores, segundo descrito acima.

As funções a serem implementadas são:

- uint8_t write(uint8_t* buf, uint8_t n, int8_t close_packet) — Função para transmitir os n bytes do buffer buf de acordo com as instruções acima. O máximo valor de n deve ser 255. A função deve bloquear até transmitir todos os bytes ou até receber o caractere XOFF, e deve retornar o número de bytes transmitidos, o que, no caso de recebimento do caractere XOFF, pode ser menos do que o valor solicitado n. Se a função for chamada depois de recebido o caractere XOFF, mas antes de ser recebido um caractere XON, ela deve retornar imediatamente com o valor de retorno 0. Se a variável close_packet for diferente de zero, ao término da transmissão dos n bytes, um caractere de sincronisom deve ser enviado.

- uint8_t read(uint8_t* buf, uint8_t n) — Função para receber os bytes de um pacote até um máximo de n bytes, colocando-os no buffer buf. O máximo valor de n deve ser 254 (observe que não é 255). A função deve bloquear até receber todos os bytes do pacote, e deve retornar o número de bytes recebidos até o caractere de sincronismo, mas sem incluí-lo. Caso o número máximo n de bytes seja atingido sem que se receba o caractere de sincronismo, a função deve retornar o valor n+1. Se esta função for chamada depois da função flow_off(), mas antes da função flow_on(), ela deve retornar imediatamente com o valor 0.

- void flow_off() — Função para enviar um caractere XOFF.

- void flow_on() — Função para enviar um caractere XON.

- uint8_t is_flow_on() — Função para detectar se a transmissão de dados pode ser feita ou não (quando o fluxo estiver desligado, depois de receber o caractere XOFF, mas antes de receber XON). Ela retorna o valor 1 quando a transmissão pode ser feita, e 0, caso contrário.

## 2.1 Testes a serem executados

No seu código, você deve executar alguns testes:

1. Inicie o hardware com o led apagado e com a interface serial configurada para transmissão e recepção.

2. Envie a string "Universidade Federal de Pernambuco\nDepartamento de Eletrônica e Sistemas" com um caractere de sincronismo no final.

3. Envie toda a string "Universidade Federal de Pernambuco\nDepartamento de Eletrônica e Sistemas" usando um buffer com comprimento de 10 bytes. Envie o caractere de sincronismo apenas ao final da string.

4. Receba um pacote consistindo de uma string repetida um certo número de vezes. Após 300 bytes recebidos, envie um caractere XOFF. Se a sequência sendo recebida parar, pisque o LED da placa a uma taxa de 1 Hz por 5 segundos. Caso contrário, acenda o LED de forma contínua por 5 segundos.

5. Após os 5 segundos acima, apague o LED e repita a ação.

6. Após os 5 segundos acima, apague o LED e transmita a string binária formada por 0x00, 0x01, 0x11, 0x02, 0x13, 0x04, 0x7e, 0x05, 0x7d, 0x06 de forma ininterrupta e sem adicionar o caractere de sincronismo. Leve em conta que um caractere XOFF pode ser recebido durante a transmissão. Caso receba este caractere, acenda o LED (faça isso da função main()) e, obviamente, suspenda a transmissão. Monitore para detectar quando a transmissão pode ser reiniciada, quando então você deve reiniciar a transmissão de onde parou. Neste momento, apague o LED.

7. Em um loop infinito, execute os passos abaixo:

   (a) Receba um pacote e compare com a string "Desenvolvimento de sistemas embarcados". Se forem iguais, acenda o LED da placa. Caso contrário, apague o LED.

   (b) Usando um buffer com comprimento de 10 bytes, receba um pacote (o pacote pode ter mais de 10 bytes) e compare novamente com a string "Desenvolvimento de sistemas embarcados". Se forem iguais, acenda o LED da placa. Caso contrário, apague o LED.

# 3 Instruções para geração e gravação do código

Baixe o arquivo zip apropriado para o seu microcontrolador e descompacte-o em uma pasta. Você encontrará o arquivo main.c que contém um esqueleto de programa. Implemente o que se pede na seção anterior.

Para compilar e gravar o código, as instruções são as mesmas da atividade anterior. Usaremos o programa make que executará os comandos necessários para nós. No terminal, vá para a pasta onde se encontra o seu código e execute o comando

```
make
```

Isto compilará, lincará e gravará o código em seu microcontrolador. Caso não queira fazer a gravação, você pode gerar apenas o arquivo hex (o arquivo a ser baixado para o arduino através do comando avrdude) com o comando

```
make hex
```

O arquivo hex será criado dentro da pasta com nome build.

Você pode apagar todos os arquivos binários gerados com o comando

```
make clean
```

Isso deixará a pasta com apenas o código em C e os arquivos necessários para o programa make.