Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Apucarana
Bacharelado em Engenharia da Computação

TRABALHO DE SISTEMAS EMBARCADOS

ATIVIDADE 1

Timers, interrupções e comunicação serial

Gabriel Finger Conte[1]
Maria Eduarda Pedroso[2]

APUCARANA
ABRIL / 2024

[1] gabcon@alunos.utfpr.edu.br
[2] mpedroso@alunos.utfpr.edu.br

SUMÁRIO

1. INTRODUÇÃO

Hoje em dia encontramos uma enorme variedade de Sistemas Embarcados disponíveis e sendo utilizados para os mais diversos propósitos. Sendo esses uma combinação de hardware e softwares limitados, voltados para uma aplicação específica que deve ser executada continuamente.

Dentre tais aplicações, existem aquelas que necessitam de um alto grau de sincronia entre suas tarefas, como nas redes de telefonia e internet. Onde os dados devem ser enviados na mesma frequência em que são lidos do outro lado da linha, para que seja possível reconstruir a mensagem corretamente.

Portanto, necessita-se de uma forma confiável e precisa para realizar a escrita e leitura dos dados em uma frequência constante. Uma forma de obter tal característica seria através do uso de interrupções de hardware acionadas por um timer interno a um dispositivo, como a biblioteca TimerInterrupt disponibilizada por khoih-prog (2022) para a Arduino IDE.

Além disso, as placas de prototipagem - como o Arduino ATMega 2560 utilizada no presente projeto - tornaram-se extremamente simples projetar e testar diferentes aplicações desenvolvidas sem um alto custo e possibilidade de ter que refazer o hardware desenvolvido.

Tendo isso em mente, o presente trabalho realizado como forma de avaliação parcial para a disciplina de Sistemas Embarcados apresenta um mecanismo de comunicação serial para troca de mensagens informadas pelo usuário entre duas placas Arduino ATMega 2560. Baseando seu funcionamento e sincronia na biblioteca TimerInterrupt. Além de empregar a biblioteca Serial de Arduino (©2024) para a exibição das mensagens trocadas e depuração do código.

2. DESCRIÇÃO DO PROBLEMA

O desafio abordado neste laboratório de sistemas embarcados é a implementação eficiente de comunicação entre placas Arduino utilizando interrupções. Tradicionalmente, a comunicação entre dispositivos embarcados é realizada de forma síncrona ou assíncrona, o

que pode consumir recursos valiosos de processamento e energia, especialmente em sistemas que exigem baixo consumo de energia e latência mínima.

Neste contexto, enfrentamos o problema de estabelecer uma comunicação confiável e eficiente entre duas placas Arduino, utilizando interrupções para otimizar a utilização de recursos e minimizar a latência na transmissão e recepção de dados.

A escolha de usar interrupções para gerenciar a comunicação entre as placas Arduino surge da necessidade de criar um sistema capaz de processar dados em tempo real, mantendo o sincronismo para não perder dados. A utilização de interrupções permite que o microcontrolador Arduino execute outras tarefas enquanto aguarda a ocorrência de eventos específicos, como a chegada de dados para transmissão ou recepção.

Além disso, o uso da biblioteca TimerInterrupt do Arduino e da biblioteca Serial do Arduino proporciona uma estrutura sólida e flexível para implementar a comunicação entre as placas, simplificando o desenvolvimento, trazendo uma saída visual e garantindo a compatibilidade com os recursos oferecidos pelo hardware.

Assim, o problema central abordado neste laboratório é a criação de um sistema de comunicação entre placas Arduino que seja eficiente, confiável e capaz de operar em tempo real, utilizando interrupções para otimizar a utilização de recursos e minimizar a latência na transmissão e recepção de dados.

3. EXEMPLO DE APLICAÇÃO

Para demonstrar a versatilidade dos conceitos desenvolvidos neste laboratório, consideramos dois exemplos práticos de aplicação em sistemas embarcados: um sistema de monitoramento e controle de temperatura em uma fábrica de alimentos e um repetidor Wi-Fi para estender a cobertura de rede em uma área extensa.

No primeiro exemplo, abordamos a necessidade de monitorar e controlar a temperatura em uma fábrica de alimentos para garantir a qualidade e a segurança dos produtos. Utilizando duas placas Arduino, configuramos a placa A para realizar a leitura dos sensores de temperatura e controlar os dispositivos de aquecimento e resfriamento.

Enquanto a placa B recebe os dados de temperatura e os exibe em um display ou os envia para um sistema de monitoramento centralizado. A comunicação entre as placas é

realizada utilizando interrupções, garantindo uma transmissão eficiente de dados em tempo real e minimizando a latência na comunicação.

No segundo exemplo, abordamos o problema de estender a cobertura de uma rede sem fio em uma área extensa onde o sinal Wi-Fi é fraco ou inexistente. Utilizando duas placas Arduino, configuramos a placa A como um ponto de acesso (AP) Wi-Fi e a placa B como um repetidor Wi-Fi.

A placa B recebe o sinal Wi-Fi da placa A e o retransmite para uma área mais ampla, permitindo que dispositivos na área estendida se conectem à rede principal. Novamente, a comunicação entre as placas é realizada utilizando interrupções, garantindo uma transmissão contínua e confiável do sinal Wi-Fi.

Esses exemplos ilustram como os conceitos de comunicação por interrupções podem ser aplicados em uma variedade de contextos em sistemas embarcados, desde o monitoramento de temperatura em ambientes industriais até a extensão da cobertura de redes sem fio.

4. DESENVOLVIMENTO

Para iniciar o desenvolvimento do projeto entendemos que alguns acordos deveriam ser feitos, elencamos alguns tópicos importantes para o funcionamento correto do sistema, sendo eles, qual seria o sinal que o emissor enviará enquanto não estiver enviando mensagem, qual seria o Header e Trailer para uma validação da mensagem enviada, e qual seria o tamanho dessa mensagem.

Depois de algumas discussões chegamos a conclusão que quando o emissor não estiver enviando mensagem ele irá sempre manter seu estado em baixo, o Header e o Trailer de inicio possuem os mesmos valores de “10101” para conseguirmos saber quando a mensagem inicia e quando termina, a mensagem possui 20 caracteres, ou seja seu tamanho em bits será de 160 bits.

Após esses tópicos serem discutidos, escolhemos as portas que seriam utilizadas como envio e recebimento da mensagem e demos início a implementação e lógica dos dois códigos, sendo o primeiro do emissor e o segundo do receptor.

4.1. Lógica e Código do Emissor

Iniciando a lógica implementada no emissor, foi necessário implementar diversas constantes que serão utilizadas ao longo do código, como o intervalo de tempo do Timer 1, o tamanho do cabeçalho e do trailer da mensagem, e o tamanho máximo da mensagem. Todas essas melhor descritas no Anexo A deste relatório.

Em seguida, importamos a biblioteca TimerInterrupt do Arduino e declaramos variáveis globais e funções relacionadas ao Timer 1, essenciais para o funcionamento do sistema de comunicação.

```
Unset
#include "TimerInterrupt.h"

#if USE_TIMER_1
// Declaração de variáveis globais e funções relacionadas ao Timer 1
#endif
```

Uma das funcionalidades centrais do sistema é a função TimerHandler1, responsável pelo tratamento de interrupção do Timer 1. Nesta função, é realizada a lógica de envio da mensagem bit a bit, incluindo o cabeçalho, a mensagem e o trailer. No entanto, vale ressaltar que essa função também desempenha um papel importante na recepção de mensagens. Por exemplo, quando uma placa Arduino está aguardando o recebimento de uma mensagem, ela continua executando esta função, monitorando o estado da comunicação.

```
Unset
void TimerHandler1(unsigned int outputPin = OUT_PORT) {
   static bool toggle1 = false; // Variável para alternar entre
ligado/desligado
```

```
          if (messageReady) { // Se uma mensagem estiver pronta para ser
enviada
            sendNextBit(toggle1);
          } else { // Se nenhuma mensagem está pronta para ser enviada
            toggle1 = false;
          }
```

A função sendNextBit desempenha um papel crucial no processo de envio de mensagens entre as placas Arduino. Ao receber um sinal de controle (toggle1) por referência, esta função é responsável por enviar o próximo bit da mensagem, seja ele parte do cabeçalho, da mensagem principal ou do trailer.

Ao longo da execução da função, são realizadas verificações para determinar qual parte da mensagem está sendo enviada no momento. Inicialmente, é verificado se o cabeçalho da mensagem ainda não foi completamente enviado. Caso positivo, a função envia o próximo bit do cabeçalho, definindo o estado do sinal de controle de acordo com o valor do bit a ser enviado e incrementando o índice do cabeçalho.

Caso o cabeçalho já tenha sido enviado completamente, a função verifica se a mensagem principal ainda está sendo enviada. Se sim, envia o próximo bit da mensagem, ajustando o estado do sinal de controle conforme necessário e incrementando o índice da mensagem.

Se toda a mensagem já foi enviada, a função verifica se o trailer da mensagem ainda não foi completamente enviado. Se sim, envia o próximo bit do trailer, ajustando o estado do sinal de controle de acordo com o valor do bit e incrementando o índice do trailer.

Ao finalizar o envio completo da mensagem, a função atualiza as variáveis de controle e estado, preparando o sistema para o envio de uma nova mensagem, se necessário.

No método setup(), inicializamos as configurações iniciais do sistema, como a definição dos pinos de saída, a inicialização da comunicação serial e a inicialização do Timer.

```
Unset
       void setup() {
         pinMode(outputPin, OUTPUT); // Define o pino de saída como saída
          pinMode(sendMessagePin, OUTPUT); // Define o pino de envio como
saída

         Serial.begin(9600); // Inicializa a comunicação serial
         while (!Serial); // Aguarda a inicialização da comunicação serial

       #if USE_TIMER_1
         ITimer1.init(); // Inicializa o Timer 1

         // Anexa a interrupção ao Timer 1
                  if     (ITimer1.attachInterruptInterval(TIMER1_INTERVAL_MS,
TimerHandler1, sendMessagePin, TIMER1_DURATION_MS)) {
           Serial.print(F("Starting  ITimer1 OK, millis() = "));
           Serial.println(millis());
         } else {
             Serial.println(F("Can't  set  ITimer1.  Select  another  freq.  or
timer"));
         }
       #endif
       }
```

No método loop(), verificamos se há dados disponíveis na porta serial. Se uma mensagem completa for recebida, a função charToBits() é chamada para converter a mensagem em bits e preparar o envio. Quando uma mensagem está pronta para ser enviada, a variável messageReady é ativada, e a função de tratamento de interrupção do Timer 1 é responsável pelo envio bit a bit da mensagem.

```
Unset
       void loop() {
         // Verifica se há dados disponíveis na porta serial
         if (!messageReady) {
           while (Serial.available() > 0) {
```

```
                char incomingChar = Serial.read(); // Lê o próximo caractere
disponível
             if (messageIndex < MESSAGE_SIZE) {
                incomingMessage[messageIndex++] = incomingChar; // Armazena o
caractere na mensagem recebida
               charToBits(incomingChar); // Converte o caractere para bits
             }
               // Se encontrar um caractere de nova linha, a mensagem está
pronta
             if (incomingChar == '\n') {
                 int falta = MESSAGE_SIZE - messageIndex; // Calcula quantos
caracteres faltam para completar a mensagem
               for (int i = 0; i < falta; i++) {
                       incomingMessage[messageIndex++] = ' '; // Completa a
mensagem com espaços em branco
                 charToBits(' '); // Converte o espaço em branco para bits
               }
               messageIndex = 0; // Reseta o índice para a próxima mensagem
                   messageIndexBits = 0; // Reseta o índice para a próxima
mensagem
                  messageReady = true; // Indica que uma nova mensagem está
pronta
               break; // Sai do loop de leitura
             }
           }
         }
       }
```

Por fim, comentando um pouco sobre a função charToBits, ela desempenha um papel fundamental no processo de comunicação entre as placas Arduino, pois é responsável por converter um caractere em sua representação binária, armazenando os bits resultantes em uma variável dedicada. Abaixo temos o código da função no qual…

```
Unset
       void charToBits(char character) {
         for (int i = 7; i >= 0; i--) { // Converte um caractere para bits
              bitsIncomingMessage[messageIndexBits++] = (character >> i) & 1;
// Armazena o bit na mensagem recebida
         }
       }
```

4.2. Lógica e Código Receptor

O dispositivo receptor teve sua lógica base construída no arquivo principal .ino, contando com 2 bibliotecas auxiliares criadas para modularizar questões relacionadas ao recebimento e processamento dos dados e ao uso das interrupções de timer. Podendo o código desenvolvido ser analisado com mais detalhes no Anexo II do presente documento.

Começando pelas bibliotecas criadas, a “dataReceiver.h” encapsula as definições das funções criadas para decodificar dados binários recebidos em ASCII. Utilizando para tanto, operações bitwise para remontar o byte de cada um dos caracteres recebidos, seguindo a seguinte lógica em pseudo código:

```
Unset
Instância a variável de retorno com todos os bits nulos
      Para os 7 primeiros bits no byteStream
      Adiciona 0 ou 1 no último bit de ret e desloca tudo para esquerda
      Adiciona o último bit do caractere
Retorna o caractere decodificado
```

Bem como para verificar tanto a integridade do preâmbulo quanto a integridade do trailer utilizados na montagem do pacote a ser recebido pela porta de entrada. Ambos utilizando a mesma lógica, detalhada abaixo, utilizando essencialmente verificações com *if-else*:

```
Unset
Verifica se o preâmbulo/trailer recebido está correto
    Em caso afirmativo retorna true
    Caso contrário:
        Avisa no terminal que houve problema
        Imprime o preâmbulo/trailer recebido para debug
        Retorna false
```

Já a outra biblioteca desenvolvida, “timerConfig.h”, consiste na definição das constantes e parâmetros necessários para utilizar o timer interno da placa Arduino ATMega 2560. Adaptando o exemplo TimerInterruptTest da biblioteca TimerInterrupt de khoih-prog (2022) para configurar a utilização do Timer 1 da placa, com um intervalo de 100 ms, o mesmo do emissor para garantir a sincronia. Mais duas funções que acabaram não sendo utilizadas, sendo elas para habilitar e desabilitar o timer.

Além da lógica da função de callback que será chamada a cada interrupção do timer, responsável basicamente por amostrar um novo bit do meio de comunicação e ativar a flag indicando que um novo dado foi lido, bem como mostrar no terminal serial o dado para debugging.

Partindo agora para o arquivo principal, inicialmente se definem as constantes e variáveis globais que serão utilizadas ao longo do código, sendo elas:

- Constantes
    - Pino de Input de dados
    - Tamanho do Preâmbulo: em bits
    - Tamanho do Trailer: em bits
    - Tamanho da mensagem: número de caracteres/bytes
- Variáveis Globais:
    - Tipo Byte:
        - Vetor de armazenamento dos bits do preâmbulo recebido
        - Matriz de armazenamento dos bits da mensagem recebida, onde cada linha representa um byte, ou seja, um caractere da mensagem.
        - Vetor de armazenamento dos bits do trailer recebido
        - Variável que será armazenado temporariamente todo novo bit lido do meio de comunicação.

- Tipo Bool:
  - Flag que indica novos dados lidos no meio de comunicação
  - Flag que indica que uma mensagem está sendo recebida através do meio de comunicação
  - Flag que indica que uma mensagem já foi lida e está esperando para ser exibida ao usuário
- Tipo Int:
  - Índice do vetor de bits de preâmbulo
  - Vetor com os índices da matriz de bits de da mensagem: {Index do byte atual, Index do bit atual}
  - Índice do vetor de bits de trailer

Após definidas as variáveis e constantes globais, pode-se importar as duas bibliotecas criadas, e em seguida configurar o Setup básico do receptor. O qual pode ser resumido no pseudo código a seguir:

```
Unset
Função de Setup
    Inicia a Comunicação Serial
    Configura o pino para receber os dados da outra placa
    Inicializa o timer
```

Dando continuidade, a parte mais complexa do código devido sua extensão foi a lógica do Loop da placa. Podendo ser dividida em duas estruturas de decisão acionadas por duas flags distintas.

A primeira refere-se à escuta do canal de comunicação, verificando se há uma nova mensagem sendo recebida, e processando o recebimento caso haja. Como descrito no pseudo código abaixo:

```
Unset
A cada novo dado lido pelo Timer (Flag de novo dado)
    Desabilita a flag, indicando que o novo dado já foi processado
    Verifica se já tem uma mensagem sendo recebida
```

```
Em caso negativo:
    Se houve uma variação no canal indica que há uma nova mensagem
    Ativa a flag, indicando que uma nova mensagem está sendo recebida
    Salva o primeiro dado do preâmbulo
Em caso positivo:
    Se ainda está no preâmbulo, salva o dado atualizando o índice
    Se está na parte de dados:
        Salva o bit correspondente na matriz, atualizando o índice
        Verifica se recebeu os 8 bits de um byte da mensagem, passando para o próximo
        em caso afirmativo
    Se ainda está no trailer, salva o dado atualizando o índice
    Se a mensagem terminou:
        Desativa a flag dizendo que tem uma mensagem sendo recebida
        Ativa a flag que indica que há uma nova mensagem que não foi processada
        Reseta os índices
```

Já a segunda refere-se ao tratamento das novas mensagens que acabaram de ser recebidas, verificando se as mesmas foram recebidas corretamente de maneira sincronizada, realizando para isso a análise do preâmbulo e trailer.

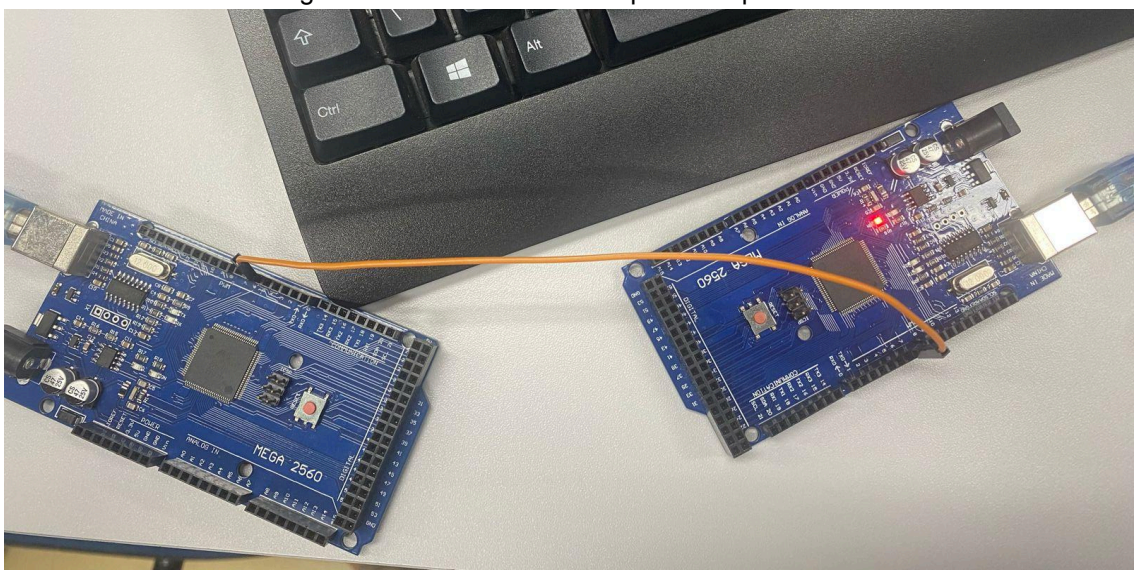
Bem como decodificando e exibindo a mensagem recebida no terminal serial. Sendo a lógica em si esplanada no pseudo código subsequente:

```
Unset
Se terminou de receber uma mensagem (flag de mensagem não processada)
    Exibe o preâmbulo recebido (debugging)
    Verifica se o preâmbulo foi recebido corretamente
    Exibe a mensagem recebida decodificada
    Imprime os bits da mensagem recebida (debugging)
    Exibe o trailer recebido (debugging)
    Verifica se o trailer foi recebido corretamente
    Demarca o fim da mensagem com um "#"
    Desativa a flag informando que a mensagem já foi lida
```

5. RESULTADOS

A montagem do circuito, conforme ilustrada na Figura 5.1, foi realizada com sucesso e cada placa foi conectada a um computador diferente para permitir a observação dos terminais no Arduino IDE.

Figura 5.1 - Circuito montado para o experimento



Fonte: autoria própria

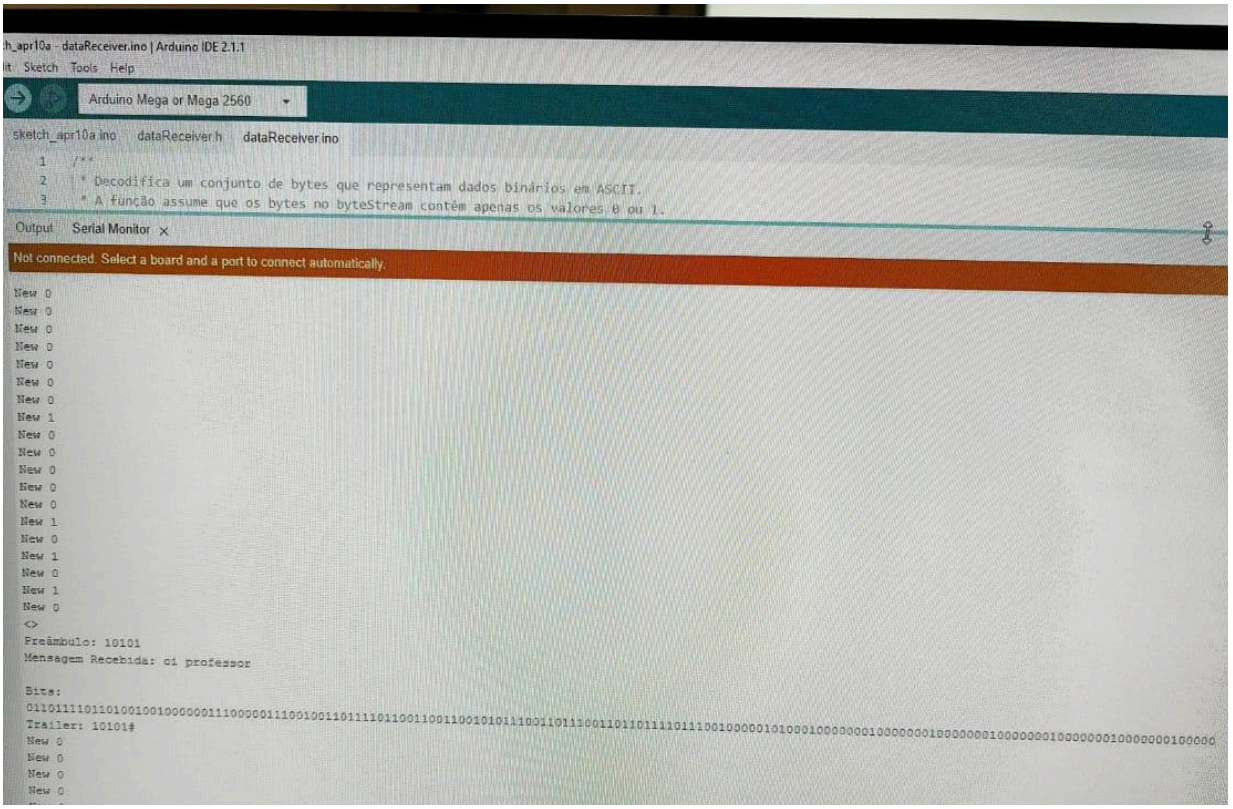
O teste inicial consistiu no envio da mensagem "oi professor" do emissor para o receptor. O sistema desenvolvido funcionou conforme esperado, com a mensagem sendo recebida e decodificada corretamente pelo receptor, como mostrado na Figura 5.2. Este resultado confirma a capacidade do sistema de codificar uma mensagem em uma stream de bits no emissor, transmiti-la com sucesso e decodificá-la de volta em uma string no receptor, exibindo-a apropriadamente.

Para a realização do experimento, uma mensagem de 20 caracteres foi escolhida com uma taxa de transmissão de 100 caracteres por segundo, o que resultou em um intervalo de 100 ms entre envios de caracteres. A possibilidade de ajustar o tamanho da mensagem, modificando uma constante nas configurações de ambas as placas, foi verificada e validada.

Experimentos adicionais foram realizados para testar diferentes intervalos de timer, começando com 1000 ms e reduzindo progressivamente. No entanto, foi observado que

intervalos inferiores a 100 ms comprometeram a sincronia entre as placas, levando à perda de informação. Isso demonstra que, enquanto o sistema é capaz de operar eficientemente dentro de certos limites de tempo, a estabilidade da comunicação é desafiada à medida que o intervalo de tempo diminui, tornando o sistema impraticável para taxas de transmissão muito altas sob as condições atuais do experimento.

Figura 5.2 - Mensagem decodificada no terminal serial do receptor



Fonte: autoria própria

6. CONCLUSÃO

O laboratório de sistemas embarcados dedicado ao uso de interrupções para comunicação entre placas Arduino proporcionou uma compreensão mais aprofundada sobre a implementação eficiente de sistemas de comunicação em tempo real. Ao longo do desenvolvimento do projeto, exploramos as funcionalidades da biblioteca TimerInterrupt do Arduino, destacando a importância das interrupções na otimização de recursos e na minimização da latência na transmissão e recepção de dados.

Através da análise detalhada do código implementado, foi possível compreender melhor as etapas envolvidas no processo de comunicação, com ênfase especial no Header e o Trailer para detecção de erros na recepção de mensagens. A função charToBits e decodeData, em particular, desempenhou um papel crucial ao converter caracteres em sua representação binária, e o inverso também, garantindo uma melhor experiência e validação do projeto conseguindo analisar a consistência na codificação e decodificação dos dados.

Em resumo, o laboratório proporcionou uma experiência prática valiosa no desenvolvimento de sistemas embarcados, demonstrando como as interrupções podem ser aproveitadas para criar sistemas de comunicação robustos e eficientes. O conhecimento adquirido durante esta atividade é de grande relevância para a construção de soluções embarcadas em uma variedade de aplicações, desde o monitoramento industrial até a ampliação de redes de comunicação sem fio.

7. REFERÊNCIAS

khoih-prog. 2022. TimerInterrupt [Software repository]. GitHub. Disponível em: https://github.com/khoih-prog/TimerInterrupt. Acesso em: 11 de abr. de 2024.
Arduino. ©2024. Serial [Documentação]. Disponível em: https://www.arduino.cc/reference/en/language/functions/communication/serial/. Acesso em: 11 de abril de 2024.

ANEXO I - CÓDIGO EMISSOR

C/C++

```
#define TIMER_INTERRUPT_DEBUG 0
#define _TIMERINTERRUPT_LOGLEVEL_ 0
#define USE_TIMER_1 true
#define HEADER "10101"
#define TRAILER "10101"
#define TAMHEADER 4
#define TAMTRAILER 4
#define MESSAGE_SIZE 20
#define USING_LOOP_TEST false
#define TIMER1_INTERVAL_MS 100
#define TIMER1_FREQUENCY (float)(1000.0f / TIMER1_INTERVAL_MS)
#define SERIAL_BAUD_RATE 9600

#include "TimerInterrupt.h" // Inclui a biblioteca TimerInterrupt.h

#if !defined(LED_BUILTIN) // Define o pino do LED embutido se não estiver definido
#define LED_BUILTIN 13
#endif

#if !defined(OUT_PORT) // Define o pino de saída se não estiver definido
#define OUT_PORT 7
#endif

#if USE_TIMER_1 // Se estiver usando o Timer 1, define variáveis e funções relacionadas

static bool messageReady = false; // Indica se uma mensagem está pronta para ser enviada
volatile int messageHeader = 0; // Índice para o cabeçalho da mensagem
volatile int messageTrailer = 0; // Índice para o trailer da mensagem
volatile int message = 0; // Índice para os bits da mensagem
char bitsIncomingMessage[MESSAGE_SIZE * 8 + 8]; // Array para armazenar os bits da mensagem recebida
unsigned int outputPin1 = LED_BUILTIN; // Pino para o LED embutido
unsigned int sendMessagePin = OUT_PORT; // Pino para enviar a mensagem
```

```
unsigned int outputPin = A0; // Pino de saída
char incomingMessage[MESSAGE_SIZE + 1]; // Array para armazenar a mensagem recebida
volatile int messageIndex = 0; // Índice para a mensagem
volatile int messageIndexBits = 0; // Índice para os bits da mensagem

void sendNextBit(bool &toggle1) {
  if (messageHeader <= TAMHEADER) { // Se ainda não enviou todo o cabeçalho da mensagem
    // Envia o próximo bit do cabeçalho
    if (HEADER[messageHeader] == '1') {
      toggle1 = true;
      Serial.println("\nHEADER 1");
    } else {
      toggle1 = false;
      Serial.println("\nHEADER 0");
    }
    messageHeader++; // Incrementa o índice do cabeçalho
  } else { // Se o cabeçalho já foi enviado
    if (message < (MESSAGE_SIZE * 8)) { // Se ainda não enviou toda a mensagem
      // Envia o próximo bit da mensagem
      if (bitsIncomingMessage[message] == 0) {
        toggle1 = false;
        Serial.println("\nMENSAGEM 0");
      } else {
        toggle1 = true;
        Serial.println("\nMENSAGEM 1");
      }
      message++; // Incrementa o índice da mensagem
    } else { // Se a mensagem foi enviada completamente
      if (messageTrailer <= TAMTRAILER) { // Se ainda não enviou todo o trailer da mensagem
        // Envia o próximo bit do trailer
        if (TRAILER[messageTrailer] == '1') {
          toggle1 = true;
          Serial.println("\TRAILER 1");
        } else {
          toggle1 = false;
```

```
          Serial.println("\TRAILER 0");
        }
        messageTrailer++; // Incrementa o índice do trailer
      } else { // Se o trailer já foi enviado completamente
        // Finaliza o envio da mensagem
        messageReady = false;
        toggle1 = false;
        messageHeader = 0;
        messageTrailer = 0;
        message = 0;
      }
    }
  }
}

// Função de tratamento de interrupção do Timer 1
void TimerHandler1(unsigned int outputPin = OUT_PORT) {
  static bool toggle1 = false; // Variável para alternar entre ligado/desligado

#if (TIMER_INTERRUPT_DEBUG > 1) // Se o debug estiver ativado
  Serial.print("ITimer1 called, millis() = ");
  Serial.println(millis());
#endif

  if (messageReady) { // Se uma mensagem estiver pronta para ser enviada
    sendNextBit(toggle1);
  } else { // Se nenhuma mensagem está pronta para ser enviada
    toggle1 = false;
  }

  digitalWrite(outputPin, toggle1); // Define o pino de saída conforme o valor de toggle1
  Serial.print("output menssage ");
  Serial.println(toggle1);
}

#endif
```

```
#if USING_LOOP_TEST
#define TIMER1_DURATION_MS (10UL * TIMER1_INTERVAL_MS)
#else
#define TIMER1_DURATION_MS 0
#endif



void charToBits(char character) {
  Serial.print("\nEm bits");
  for (int i = 7; i >= 0; i--) { // Converte um caractere para bits
    Serial.print((character >> i) & 1); // Shifta o caractere e extrai o bit
      bitsIncomingMessage[messageIndexBits++] = (character >> i) & 1; // Armazena o bit na
mensagem recebida
  }
}

void setup() {
  pinMode(outputPin1, OUTPUT); // Define o pino do LED embutido como saída
  pinMode(outputPin, OUTPUT); // Define o pino de saída como saída
  pinMode(sendMessagePin, OUTPUT); // Define o pino de envio como saída

  Serial.begin(9600); // Inicializa a comunicação serial
  while (!Serial); // Aguarda a inicialização da comunicação serial

  // Exibe informações sobre o sistema
  Serial.print(F("\nStarting TimerInterruptTest on "));
  Serial.println(BOARD_TYPE);
  Serial.println(TIMER_INTERRUPT_VERSION);
  Serial.print(F("CPU Frequency = "));
  Serial.print(F_CPU / 1000000);
  Serial.println(F(" MHz"));

#if USE_TIMER_1
```

```
            ITimer1.init(); // Inicializa o Timer 1

            // Anexa a interrupção ao Timer 1
            if (ITimer1.attachInterruptInterval(TIMER1_INTERVAL_MS, TimerHandler1, sendMessagePin,
TIMER1_DURATION_MS)) {
              Serial.print(F("Starting  ITimer1 OK, millis() = "));
              Serial.println(millis());
            } else {
              Serial.println(F("Can't set ITimer1. Select another freq. or timer"));
            }
          #endif
          }

          void loop() {
            // Verifica se há dados disponíveis na porta serial
            if (!messageReady) {
              while (Serial.available() > 0) {
                char incomingChar = Serial.read(); // Lê o próximo caractere disponível
                Serial.print("\nCaractere recebido: ");
                Serial.print(incomingChar);
                if (messageIndex < MESSAGE_SIZE) {
                      incomingMessage[messageIndex++] = incomingChar; // Armazena o caractere na
mensagem recebida
                  charToBits(incomingChar); // Converte o caractere para bits
                }
                // Se encontrar um caractere de nova linha, a mensagem está pronta
                if (incomingChar == '\n') {
                    int falta = MESSAGE_SIZE - messageIndex; // Calcula quantos caracteres faltam para
completar a mensagem
                  Serial.print("\nFALTA");
                  Serial.println(falta);
                  for (int i = 0; i < falta; i++) {
                      incomingMessage[messageIndex++] = ' '; // Completa a mensagem com espaços em
branco
                    charToBits(' '); // Converte o espaço em branco para bits
                  }
```

```
            messageIndex = 0; // Reseta o índice para a próxima mensagem
            messageIndexBits = 0; // Reseta o índice para a próxima mensagem
            messageReady = true; // Indica que uma nova mensagem está pronta
            break; // Sai do loop de leitura
          }
        }
      }
    }
```

ANEXO II - CÓDIGO RECEPTOR

- **Arquivo Principal “receptorSinal.ino”**

```
C/C++
/**
 * @file receptorSinal.ino
 * @brief Implementação do receptor de sinal.
 * @author Gabriel Finger Conte e Maria Eduarda Pedroso
 *
 * Este arquivo contém a implementação do receptor de sinal, responsável por receber e processar mensagens
 * binárias em ASCII. Ele utiliza interrupções de timer para ler os dados do canal de comunicação e decodificá-los.
*/

/* Definind os pinos a serem utilizados */
#define INPUT_PIN 6

/* Definindo constantes necessárias */
#define PREAMBULE_LENGTH 5
#define TRAILER_LENGHT 5
#define DATA_LENGHT 20

/* Variáveis globais */
byte receivedPreambule[PREAMBULE_LENGTH]; /* Armazena os bits do Preâmbulo recebido */
byte receivedData[DATA_LENGHT][8]; /* Armazena os bits da Mensagem recebida */
byte receivedTrailer[TRAILER_LENGHT]; /* Armazena os bits do Trailer Recebido */
volatile byte newData = 0x00; /* Armazena o novo bit lido do meio de comunicação */

volatile bool newUnprocessedData = false; /* Flag que indica novos dados lidos no meio de comunicação */
bool incomingMessage = false; /* Flag que indica que uma mensagem está sendo recebida através do meio de comunicação */
bool unreadMessage = false; /* Flag que indica que uma mensagem já foi lida e está esperando para ser exibida ao usuário */
```

```
int incomingPIndex = 0; /* Index do vetor de bits de preâmbulo */
int incomingMIndex[2] = { 0, 0 }; /* Index do matriz de bits de da mensagem  {Index do byte atual, Index
do bit atual} */
int incomingTIndex = 0; /* Index do vetor de bits de trailer */

/* Inclusão de Bibliotecas */
#include "dataReceiver.h" /* Funções com a Lógica do Receptor */
#include "timerConfig.h" /* Configuração e Funções com o Timer */

/* Função de Setup */
void setup() {
  /* Inicia a Comunicação Serial */
  Serial.begin(115200);
  while (!Serial);

  /* Configura o pino para receber os dados da outra placa */
  pinMode(INPUT_PIN,INPUT);

  /* Inicializa o timer */
  #if USE_TIMER_1
    ITimer1.init();
    if (ITimer1.attachInterruptInterval(TIMER_INTERVAL_MS, TimerHandler)) {
      Serial.print(F("Starting  ITimer1 OK, millis() = "));
      Serial.println(millis());
    } else
      Serial.println(F("Can't set ITimer1. Select another freq. or timer"));
  #endif

}  // setup

/* Função de Loop */
void loop() {

  /* A cada novo dado lido pelo Timer */
  if (newUnprocessedData) {
```

```
/* Indica que o novo dado já foi processado */
newUnprocessedData = false;

/* Se não uma mensagem que está sendo recebida */
if (!incomingMessage) {
  /* Se houve uma variação no canal, indica que há uma nova mensagem */
  if (newData) {
    /* Indica que tem uma mensagem vindo */
    incomingMessage = true;

    /* Salva o dado do preâmbulo */
    receivedPreambule[incomingPIndex++] = newData;
  }  // if newData
} else {
  /* Se tem uma mensagem sendo recebida e está no preâmbulo */
  if (incomingPIndex < PREAMBULE_LENGTH) {
    /* Salva o dado do preâmbulo */
    receivedPreambule[incomingPIndex++] = newData;
  } else if (incomingMIndex[0] < DATA_LENGHT) {
    /* Se está na parte dos dados, salva o bit correspondente */
    receivedData[incomingMIndex[0]][incomingMIndex[1]++] = newData;

    /* Se recebeu os 8 bits de um byte da mensagem, passa para o próximo */
    if (incomingMIndex[1] >= 8) {
      incomingMIndex[1] = 0;
      incomingMIndex[0]++;
    }  // if
  } else if (incomingTIndex < TRAILER_LENGHT) {
    /* Se está no trailer, salva no trailer */
    receivedTrailer[incomingTIndex++] = newData;
  } else {
    /* Se terminou a mensagem, desativa a flag que tem mensagem sendo recebida */
    incomingMessage = false;

    /* Ativa a flag de uma nova mensagem que não foi processada */
    unreadMessage = true;
```

```
      /* Resetar os Indexes */
      incomingPIndex = 0;
      incomingMIndex[0] = 0;
      incomingMIndex[1] = 0;
      incomingTIndex = 0;

      /* Imprime um símbolo indicando o fim*/
      Serial.println("<>");
    }  // if incomingPIndex
  }    // if incomingMessage
}  // if newUnprocessedData

/* Se terminou de receber uma mensagem */
if (unreadMessage) {
  /* Debugging - Exibe o preâmbulo */
  Serial.print("Preâmbulo: ");
  for (int i = 0; i < PREAMBULE_LENGTH; i++) {
    Serial.print(receivedPreambule[i]);
  }  // for

  /* Verifica se o Preâmbulo foi recebido corretamente */
  verifyPreambulo(receivedPreambule);

  /* Exibe a mensagem recebida decodificada */
  Serial.print("\nMensagem Recebida: ");
  for (int i = 0; i < DATA_LENGHT; i++) {
    Serial.print(decodeData(receivedData[i]));
  }  // for

  /* Debuggin - Imprime os Bits Recebidos da Mensagem */
  Serial.println("\nBits:");
  for (int i = 0; i < DATA_LENGHT; i++) {
    for(int j = 0; j < 8; j++){
      Serial.print(receivedData[i][j]);
    }// for j
```

```
    }// for i

    /* Debuggin - Exibe o trailer */
    Serial.print("\nTrailer: ");
    for (int i = 0; i < PREAMBULE_LENGTH; i++) {
      Serial.print(receivedTrailer[i]);
    }  // for

    /* Verifica se o trailer foirecebido corretamente */
    verifyTrailer(receivedTrailer);

    /* Demarca o fim da mensagem */
    Serial.println("#");

    /* Desativa a flag informando que a mensagem já foi lida */
    unreadMessage = false;
  }  // if

}  // loop
```

- **Biblioteca “dataReceiver.h”**

```
C/C++
/**
 * @file dataReceiver.h
 * @brief Declaração das funções para decodificação de dados binários em ASCII e verificação de preâmbulo e trailer.
 * @author Gabriel Finger Conte e Maria Eduarda Pedroso
 *
 * Este arquivo contém a declaração das funções para decodificar dados binários em ASCII,
 * verificar a integridade do preâmbulo e verificar a integridade do trailer em um
 * processo de comunicação.
*/
```

```
#ifndef dataReceiver_h
#define dataReceiver_h

/**
 * Decodifica um conjunto de bytes que representam dados binários em ASCII.
 * A função assume que os bytes no byteStream contêm apenas os valores 0 ou 1.
 * Os bytes são decodificados bit a bit para formar um caractere ASCII.
 * @param byteStream Um array de bytes contendo os dados binários a serem decodificados.
 * @return O caractere decodificado a partir dos dados binários.
 */
char decodeData(byte *byteStream);


/**
 * Verifica se o preâmbulo recebido está correto.
 * A função compara os bytes recebidos com um padrão específico para o preâmbulo.
 * Em caso de divergência, exibe uma mensagem de aviso no terminal.
 * @param byteStream Um array de bytes contendo o preâmbulo a ser verificado.
 * @return true se o preâmbulo estiver correto, false caso contrário.
 */
boolean verifyPreambulo(byte *byteStream);


/**
 * Verifica se o trailer recebido está correto.
 * A função compara os bytes recebidos com um padrão específico para o trailer.
 * Em caso de divergência, exibe uma mensagem de aviso no terminal.
 * @param byteStream Um array de bytes contendo o trailer a ser verificado.
 * @return true se o trailer estiver correto, false caso contrário.
 */
boolean verifyTrailer(byte *byteStream);


#endif
```

- **Implementação da Biblioteca “dataReceiver.ino”**

```C/C++
/**
 * @file dataReceiver.ino
 * @brief Funções para decodificar dados binários em ASCII e verificar preâmbulo e trailer.
 * @author Gabriel Finger Conte e Maria Eduarda Pedroso
 *
 * Este arquivo contém as definições das funções para decodificar dados binários em ASCII,
 * verificar a integridade do preâmbulo e verificar a integridade do trailer em um
 * processo de comunicação.
*/

/**
 * Decodifica um conjunto de bytes que representam dados binários em ASCII.
 * A função assume que os bytes no byteStream contêm apenas os valores 0 ou 1.
 * Os bytes são decodificados bit a bit para formar um caractere ASCII.
 * @param byteStream Um array de bytes contendo os dados binários a serem decodificados.
 * @return O caractere decodificado a partir dos dados binários.
 */
char decodeData(byte *byteStream) {
  /* Instância a variável de retorno com todos os bits nulos */
  char ret = 0x00;

  /* Para os 7 primeiros bits no byteStream */
  for (int8_t i = 0; i < 7; i++) {
    /* Adiciona 0 ou 1 no último bit de ret e desloca tudo para esquerda */
    ret = (ret | byteStream[i]) << 1;
  }  // for

  /* Adiciona o último bit do caractere */
  ret = (ret | byteStream[7]);

  /* Retorna o caractere decodificado */
  return ret;
}  // decodeData
```

```
/**
 * Verifica se o preâmbulo recebido está correto.
 * A função compara os bytes recebidos com um padrão específico para o preâmbulo.
 * Em caso de divergência, exibe uma mensagem de aviso no terminal.
 * @param byteStream Um array de bytes contendo o preâmbulo a ser verificado.
 * @return true se o preâmbulo estiver correto, false caso contrário.
 */
boolean verifyPreambulo(byte *byteStream) {
  /* Verifica se o preâmbulo recebido está correto */
  if (
       byteStream[0] == 1 && byteStream[1] == 0 && byteStream[2] == 1 && byteStream[3] == 0 &&
byteStream[4] == 1) {
    /* Em caso afirmativo retorna true */
    return true;
  }  // if

  /* Caso contrário, avisa no terminal que houve problema */
  Serial.println("Warning - Preâmbulo Incorreto, ocorreu um problema na comunicação");

  /* Imprime o Preâmbulo recebido para debug */
  for (int8_t i = 0; i < 5; i++) {
    Serial.print(byteStream[i]);
  }  // for

  /* Retorna falso */
  return false;
}  // verifyPreambulo


/**
 * Verifica se o trailer recebido está correto.
 * A função compara os bytes recebidos com um padrão específico para o trailer.
 * Em caso de divergência, exibe uma mensagem de aviso no terminal.
 * @param byteStream Um array de bytes contendo o trailer a ser verificado.
 * @return true se o trailer estiver correto, false caso contrário.
```

```
 */
boolean verifyTrailer(byte *byteStream) {
  /* Verifica se o trailer recebido está correto */
  if (
       byteStream[0] == 1 && byteStream[1] == 0 && byteStream[2] == 1 && byteStream[3] == 0 &&
byteStream[4] == 1) {
    /* Em caso afirmativo retorna true */
    return true;
  }  // if

  /* Caso contrário, avisa no terminal que houve problema */
  Serial.println("Warning - Trailer Incorreto, ocorreu um problema na comunicação");

  /* Imprime o Preâmbulo recebido para debug */
  for (int8_t i = 0; i < 5; i++) {
    Serial.print(byteStream[i]);
  }  // for

  /* Retorna falso */
  return false;
}  // verifyTrailer
```

- **Biblioteca “timerConfig.h”**

```
C/C++
/**
 * @file timerConfig.h
                    *                    @note                    Adaptado                    de
https://github.com/khoih-prog/TimerInterrupt/blob/master/examples/TimerInterruptTest/TimerInterruptTest
.ino
 * @brief Configurações do timer.
 * @author Gabriel Finger Conte e Maria Eduarda Pedroso
 *
```

```
 * Este arquivo contém as definições das constantes relacionadas ao timer, bem como a inclusão da biblioteca TimerInterrupt.h
 * e a declaração das funções para habilitar, desabilitar e lidar com interrupções do timer.
*/

#ifndef timerConfig_H
#define timerConfig_H

/* Definição de constantes de Timer */
#define TIMER_INTERRUPT_DEBUG 2
#define _TIMERINTERRUPT_LOGLEVEL_ 2
#define USE_TIMER_1 true
#define TIMER_INTERVAL_MS 100


#if (defined(__AVR_ATmega644__) || defined(__AVR_ATmega644A__) || defined(__AVR_ATmega644P__) || defined(__AVR_ATmega644PA__) || defined(ARDUINO_AVR_UNO) || defined(ARDUINO_AVR_NANO) || defined(ARDUINO_AVR_MINI) || defined(ARDUINO_AVR_ETHERNET) || defined(ARDUINO_AVR_FIO) || defined(ARDUINO_AVR_BT) || defined(ARDUINO_AVR_LILYPAD) || defined(ARDUINO_AVR_PRO) || defined(ARDUINO_AVR_NG) || defined(ARDUINO_AVR_UNO_WIFI_DEV_ED) || defined(ARDUINO_AVR_DUEMILANOVE) || defined(ARDUINO_AVR_FEATHER328P) || defined(ARDUINO_AVR_METRO) || defined(ARDUINO_AVR_PROTRINKET5) || defined(ARDUINO_AVR_PROTRINKET3) || defined(ARDUINO_AVR_PROTRINKET5FTDI) || defined(ARDUINO_AVR_PROTRINKET3FTDI))
#define USE_TIMER_2 true
#warning Using Timer1
#else
#define USE_TIMER_3 true
#warning Using Timer3
#endif

/* Importando a Biblioteca do Timer */
#include "TimerInterrupt.h"

/**
 * Função para desabilitar o temporizador.
```

```
*/
void disableTimer();

/**
 * Função para reabilitar o temporizador.
*/
void enableTimer();

/**
  *Função que irá ser executada pelo Timer
*/
void TimerHandler();

#endif
```

- **Implementação da Biblioteca “timerConfig.ino”**

```
C/C++
/**
 * @file timerConfig.ino
 * @brief Implementação das funções relacionadas à configuração do timer.
 * @author Gabriel Finger Conte e Maria Eduarda Pedroso
 *
 * Este arquivo contém a implementação das funções para desabilitar, habilitar o
 * temporizador e lidar com a interrupção do timer.
*/

/**
 * Função para desabilitar o temporizador.
*/
void disableTimer() {
  ITimer1.disableTimer();
}  // disableTimer
```

```
/**
 * Função para reabilitar o temporizador.
*/
void enableTimer() {
  ITimer1.enableTimer();
}  // enableTimer

/**
  *Função que irá ser executada pelo Timer
*/
void TimerHandler() {

  /* Lê o novo dado */
  newData = digitalRead(INPUT_PIN);
  /* Ativa a flag informando que tem um novo dado não lido */
  newUnprocessedData = true;
  /* Debuggin - Imprimindo no terminal serial o novo dado lido */
  Serial.print("New ");
  Serial.println(newData);

}  // TimerHandler
```