Atividade_08 - Livro AVR e Arduino – Técnicas de Projeto
Capítulo: 15 (USART)

Título: Usando a porta serial para controlar dispositivos

Objetivos: Aprender a usar a porta serial dos microcontroladores da Atmel. Aprender a usar a porta serial dirigida a interrupções.

Nesta prática utilizaremos o Tinkercad para simular um circuito simples usando o microcontrolador Atmega328, utilizado nas placas Arduino UNO. Desta vez, programaremos usando um código C para acender e apagar LEDs usando a porta serial. Faremos também toda a comunicação ser dirigida a interrupções.

1. Procedimentos:

1. Acesse sua conta no Tinkercad (tinkercad.com) e vá para a aba circuits (https://www.tinkercad.com/circuits). Você pode utilizar o projeto disponível em https://www.tinkercad.com/things/cMgQuhBoQHC como exemplo de como configurar uma porta serial que utiliza chamadas bloqueantes.

2. Ligue dois LEDs, um na porta PB5 (13), um na porta PB4 (12).

3. Agora você deve fazer com que o LED na porta PB5 (13) acenda ao se enviar 'A13' e apague ao se enviar 'S13'. O LED na porta PB4 (12) deve alternar de estado toda vez que 'D12' for recebido.

4. Agora você deve modificar o código de forma a enviar um texto pela porta serial de tempos em tempos. Este código deve rodar dentro do while() do main(). O texto será a saída do millis() e ele será enviado a cada 100ms. Utilize a função `init()` para que `millis()` funcione.

Modelo de função principal. Note que o processamento dos comandos dos leds não se encontra nesta função.

```
int main() {
init();
configura_serial(9600);
while(1) {
  _delay_ms(100);
  envia_millis(millis()); // não bloqueante
}}
```

5. Todo o código da porta serial deve ser dirigido a interrupções. A resposta dos leds deve ser imediata quando se envia os comandos para seu controle.

6. Cole o código fonte do microcontrolador ao final deste arquivo e inclua a imagem de seu design. Importante: Deixe seu circuito público no Tinkercad e cole o link para ele aqui:

Tinkercad: https://www.tinkercad.com/things/73PwR5eoWFm-copy-of-portaserial/editel?sharecode=xWyWd17Y2g4ZXyklaKoRvx9lOuoqn8tznbjoLjsiY8w

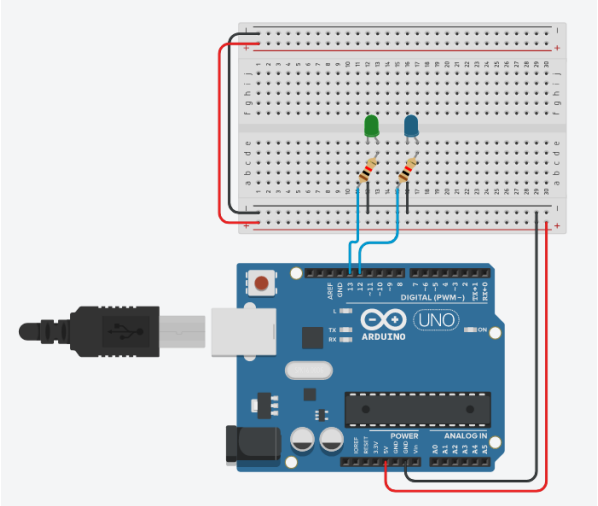
Código:

```
#define F_CPU 16000000UL  //define a frequencia do microcontrolador - 16MHz

#include <avr/io.h>       //definições do componente especificado
#include <util/delay.h>   //biblioteca para o uso das rotinas de _delay_ms e _delay_us()
#include <avr/pgmspace.h>   //para o uso do PROGMEM, gravação de dados na memória flash
#include <stdio.h>
#include <string.h>


#define BAUD   2400    //taxa de 2400 bps
```

```
#define MYUBRR  F_CPU/16/BAUD-1

#define BUFFLEN 32
// buffer de dados de envio na porta serial
char buff[BUFFLEN];
char receptBuffer[4] = {0};
// pos indica a posição corrente no buffer
// end indica a posição final no buffer com dados válidos
// note que end pode ser menor que pos, já que o buffer é circular
// cntbuff indica quantos caracteres válidos há no buffer
short pos=0,end=0,cntbuff=0;

//------------------------------------------------------------------------
void USART_Inic(unsigned int ubrr0)
{
  UBRR0H = (unsigned char)(ubrr0>>8); //Ajusta a taxa de transmissão
  UBRR0L = (unsigned char)ubrr0;

  UCSR0A = 0;//desabilitar velocidade dupla (no Arduino é habilitado por padrão)
  UCSR0B |= (1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0);
  UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);/*modo assíncrono, 8 bits de dados, 1 bit de parada,
sem paridade*/



  sei();
}

//------------------------------------------------------------------------
void EscreveAsync(char *dados, short len) {
  // desabilita temporariamente interrupção
  UCSR0B &= ~(1<<UDRIE0);
  // não deixa enfileirar mais dados que o tamanho do buffer
  len %= BUFFLEN;

  // separa a cópia dos dados para o buffer em duas
  // partes se necessário for
  cntbuff += len;
  short cnt = BUFFLEN-end;
  cnt = cnt>len?len:cnt;

  memcpy(buff+end, dados, cnt);
  if (cnt<len)
    memcpy(buff, dados+cnt, len-cnt);

  end += len;
  end %= BUFFLEN;
  // habilita interrupção para (re)iniciar o envio
  UCSR0B |= (1<<UDRIE0);
}

//------------------------------------------------------------------------

void envia_millis(unsigned long mil) {
  char millisStr[12];
  sprintf(millisStr, "%lu\n", mil);
  EscreveAsync(millisStr, strlen(millisStr));
}

//------------------------------------------------------------------------
ISR(USART_UDRE_vect) {
  // enquanto houver dados no buffer
```

```
  if (cntbuff>0) {
    UDR0 = buff[pos]; // envia
    pos++;
    cntbuff--;
    pos %= BUFFLEN; // buffer circular
  } else {
    // se não houver o que enviar, desliga interrupção
    UCSR0B &= ~(1<<UDRIE0);
  }
}

ISR(USART_RX_vect) {
  receptBuffer[0] = receptBuffer[1];
  receptBuffer[1] = receptBuffer[2];
  receptBuffer[2] = UDR0; // Recebe o dado do serial

  if(!strcmp(receptBuffer, "A13")){
    PORTB |= 0b00100000;
  } else if(!strcmp(receptBuffer, "S13")){
    PORTB = 0b00000000;
  } else if(!strcmp(receptBuffer, "D12")){
    PORTB ^= 0b00010000;
  }
}


//------------------------------------------------------------------------
int main() {

        init();

        DDRB |= (1<<PB4)| (1<<PB5);      //define PB4 e PB5 como saída

        USART_Inic(MYUBRR);

        while(1) {
                _delay_ms(100);
                envia_millis(millis()); // não bloqueante
        }
}
```

Arduino:



ATENÇÃO: Documente seu código. Cada linha/bloco deve deixar explícito o seu papel.
ATENÇÃO: Na versão final do seu projeto, as funções pinMode(), digitalWrite() e digitalRead() são proibidas. O uso delas fará a nota atribuída ser zero. Utilizar USART com envio e recebimento dirigido a interrupções é obrigatório.

Proibido utilizar classes prontas que encapsulem funcionalidades da porta serial, como a classe Serial do Arduino.

RÚBRICA:
10% Circuito;
40% LEDs funcionam como descrito (são ativados via serial sem delays);
50% O valor do millis() é enviado no tempo certo sem afetar as demais funções;
Valor desta atividade na média: 0.7