Inatel Instituto Nacional de Telecomunicações	**RELATÓRIO 10**	**Data: / /**
	Disciplina: E209	
	Prof: Yvo Marcelo Chiaradia Masseli **Monitores: João Lucas/Luan Siqueira/Maria Luiza/ Lucas Lares/Rafaela Papale**	
Conteúdo: Microcontrolador ATMega328p		
Tema: Canal Serial - UART		
Nome:	**Matrícula:**	**Curso:**

OBJETIVOS:

- Utilizar as ferramentas de simulação para desenvolver programas para o ATMega328p.
- Desenvolver um programa que faz uso do canal serial UART do ATMega328p.
- Utilizar as entradas e saídas do ATMega328p com circuitos de aplicação.

Parte Teórica

Canal de comunicação serial assíncrona - UART

A comunicação serial é o tipo de comunicação onde os bits da palavra de dados não são transmitidos de uma única vez e, sim, um bit a cada momento. As comunicações seriais podem variar em função do modo de uso do canal de comunicação e pelo sincronismo da informação.

A comunicação pode ser considerada síncrona ou assíncrona. No caso do modo síncrono existe um sinal de clock que é transmitido junto com o sinal de dados. No caso assíncrono, não existe sinal de clock e o dado transmitido e recebido corretamente porque ambos lados operam com o mesmo tempo de duração de bit. Esse tempo de duração de bit é conhecido como **baudrate** ou taxa de dados. A taxa de dados, medida em **bps – bits por segundo**, é escolhida em função da quantidade/fluxo de dados que vão ser transmitidos e recebidos pelo canal de comunicação. Quanto maior o fluxo de dados, maior deve ser a taxa de dados. Porém, quanto maior a taxa de dados, maior a taxa de erros que pode acontecer na comunicação caso a distância seja muito grande. É aconselhável não utilizar distância de comunicação entre TX e RX superiores a 5m. Um detalhe importante, como descrito anteriormente, é que por ser uma comunicação assíncrona, a taxa de dados utilizada deve ser a mesma no TX e no RX. Caso isso não aconteça, o que for transmitido por um lado(transmissor) não será entendido pelo outro(receptor).

O modo utilizado aqui é o modo assíncrono – muito conhecido como UART (Universal Asynchronous Receiver Transmitter), onde o dado tem início com um pulso de start, seguido pelos bits de dados e finalizado pelo pulso de stop. O padrão utilizado no PC é o padrão RS232C, que padroniza os níveis de tensão, a taxa de dados e a pinagem dos conectores que irão interligar os lados da comunicação. A Figura 1 ilustra um exemplo de aplicação de comunicação entre um PC e um periférico, que pode ser um microcontrolador, um leitor de código de barras, impressora térmica, entre outros. Na ilustração, verifica-se o cruzamento que ocorre entre os pinos TX/RX para que a comunicação possa acontecer, sendo omitida a linha de referência de tensão (GND) que deve ser a mesma para ambos os lados.

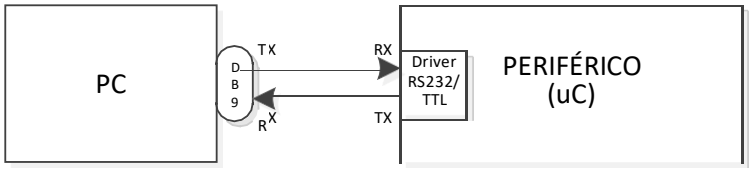

Figura 1 – Exemplo de uso da comunicação RS232 entre PC e um periférico.

Uma observação pode ser feita sobre o padrão RS232: a interface USB está, cada vez mais, sendo utilizada no lugar da interface RS232. Porém, para a interconexão entre periféricos microcontrolados o padrão RS232 ainda é bastante utilizado. Devido a essa tendência, os computadores novos e, principalmente, os notebooks, não possuem porta RS232. Porém isso não limita o seu uso pois existem cabos conversores USB/Serial a custos relativamente baixos que não prejudicam, no geral, a performance de comunicação. Esses cabos conversores são conectados na porta USB do computador e criam uma porta serial "virtual" que, para as aplicações funciona com uma porta real.

O Atmega328p possue uma porta serial (conhecida com USART), enquanto outros microcontroladores podem possuir várias portas sendo UART ou USART.

No Atmega328p os pinos **PD0 e PD1** são usados para comunicação com o computador ou outro microcontrolador qualquer com suporte a comunicação serial. Conectar qualquer coisa a esses pinos pode interferir nessa comunicação, incluindo causar falhas na gravação da placa.

Enfim, os parâmetros mais importantes de uma comunicação serial UART são: taxa de dados **(baudrate)**, número de bits, número de stop bits e paridade. Desses parâmetros, o mais comum de ser alterado é o **baudrate** que influência, geralmente, a distância máxima de comunicação e taxa real de comunicação do sistema: quanto maior o **baudrate**, mais rápido ocorre a comunicação de um determinado pacote de dados, porém, menor a distância máxima permitida entre o TX e o RX. Assim, costuma-se dizer que uma serial opera no padrão 9600,8,N,1. O que isso significa? 9600 é a taxa de dados**(9600 bps)**, 8 é o número de bits de dados (um byte), N é que não está sendo utilizada a paridade(quantidade de bits "1" na palavra de dados) e 1 é a quantidade de bits de stop (fim de comunicação).

São exemplos de sistemas que usam UART: módulos de comunicação GSM/GPRS, módulos GPS, sensores, displays alfanuméricos, impressoras, leitores de códigos de barras.

Considerações importantes

Para enviarmos dados de sensores ou variáveis numéricas precisamos antes convertê-los. Imagine que você queira mandar o valor de um sensor de temperatura que varia de 0 a 700 graus para o PC. Para isso você deve pegar o valor do sensor, convertê-lo em uma **"STRING"** através do padrão **ASCII**.

Lembre-se que nos arquivos fornecidos o tamanho esperado da mensagem a ser recebida é variável logo ele deve ser modificado de acordo com seu projeto e de acordo com o que você espera receber.

Funções para serem utilizadas

```
//não é recomendado que o tamanho dos vetores seja maior
//que 32 por motivos de limitações físicas e tempo
char mensagem_tx[20];
char mensagem_rx[32];
int tam = 0;
int TAMANHO = tamanho que você queira; // inicie com 1 para mensagens simples
```

Configuração da UART

```
void UART_Init(unsigned int ubrr)
{
    // Configura a  baud rate */
    UBRR0H = (unsigned char)(ubrr>>8);
    UBRR0L = (unsigned char)ubrr;
    // Habilita a recepcao, tranmissao e interrupcao na recepcao */
    UCSR0B = (1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0);
    // Configura o formato da mensagem: 8 bits de dados e 1 bits de stop */
    UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
}
```

Envio de mensagens

```
void UART_Transmit(char* dados)
{
    // Envia todos os caracteres do buffer dados ate chegar um final de linha
    while (*dados != 0) {
        while(!(UCSR0A & (1<<UDRE0))); // Aguarda a transmissão acabar
        // Escreve o caractere no registro de tranmissão
        UDR0 = *dados;
        // Passa para o próximo caractere do buffer dados
        dados++;
    }
}
```

Recepção de mesagens

```
ISR(USART_RX_vect) {
    // Escreve o valor recebido pela UART na posição pos_msg_rx do buffer msg_rx
    msg_rx[pos_msg_rx++] = UDR0;
    if (pos_msg_rx == tamanho_msg_rx)
        pos_msg_rx = 0;
}
```

Exemplo de utilização:

```
#include <stdio.h>
#include <stdlib.h>

#define FOSC    16000000U // Clock Speed
#define  BAUD 9600
#define MYUBRR  FOSC / 16 / BAUD - 1
#define botao   (1 << PD4)

char msg_tx[20];
char msg_rx[32];
int pos_msg_rx = 0;
int tamanho_msg_rx = 3;
unsigned int x = 0, valor = 0;

//Prototipos das funcoes
void UART_Init(unsigned int ubrr);
void UART_Transmit(char *dados);

int main(void)
{
    UART_Init(MYUBRR);
    sei();
    PORTD |= botao;

    UART_Transmit("Digite 'ola':\n");

    x = 0;
    while (x == 0)
    {
        if ((msg_rx[0] == 'o') &&
            (msg_rx[1] == 'l') &&
            (msg_rx[2] == 'a'))
        {
            x = 1;
        }
    }

    UART_Transmit("Digite '250':\n");

    x = 0;
    valor = 0;
    while (x == 0)
    {
        valor = (msg_rx[0] - 48) * 100 +
                (msg_rx[1] - 48) * 10 +
                (msg_rx[2] - 48) * 1;

        if (valor == 250)
            x = 1;
```

```
    }

    x = 0;
   UART_Transmit("Aperte o botao:\n");

    // Super-loop
   while (1) {
        if ((PIND & botao) == 0) // O botao foi pressionado?
        {
          // Se sim, envia mensagem
            UART_Transmit("Hello World!\n");
            x++;

            UART_Transmit("num vezes botao press: ");
            itoa(x, msg_tx, 10);
            UART_Transmit(msg_tx);
            UART_Transmit("\n");

            _delay_ms(500); // Aguarda um tempo para evitar o bounce
        }
   }
}

ISR(USART_RX_vect)
{
    // Escreve o valor recebido pela UART na posição pos_msg_rx do buffer msg_rx
    msg_rx[pos_msg_rx++] = UDR0;
    if (pos_msg_rx == tamanho_msg_rx)
        pos_msg_rx = 0;
}

void UART_Transmit(char *dados)
{
    // Envia todos os caracteres do buffer dados ate chegar um final de linha
    while (*dados != 0)
    {
        while (!(UCSR0A & (1 << UDRE0))); // Aguarda a transmissão acabar
        // Escreve o caractere no registro de tranmissão
        UDR0 = *dados;
        // Passa para o próximo caractere do buffer dados
        dados++;
    }
}

void UART_Init(unsigned int ubrr)
{
    // Configura a  baud rate */
    UBRR0H = (unsigned char)(ubrr >> 8);
    UBRR0L = (unsigned char)ubrr;
    // Habilita a recepcao, tranmissao e interrupcao na recepcao */
```

```
    UCSR0B = (1 << RXEN0) | (1 << TXEN0) | (1 << RXCIE0);
    // Configura o formato da mensagem: 8 bits de dados e 1 bits de stop */
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);
}
```

Parte Prática

Programa 1)
Crie um programa que quando o botão for pressionado **(interrupção externa)** envie o número de vezes que ele foi pressionado **(incluindo a atual)** para o PC através da UART. E toda vez que o computador enviar o comando **"Z"** devemos zerar a contagem.

Programa 2)
Crie um programa para controlar dois leds, **um vermelho e um verde**. toda vez que o computador enviar o comando **"0"** o vermelho liga e o verde desliga e **"1"** o verde liga e vermelho desliga.

Programa 3)
Crie um programa que receba do computador o valor do DUTY CYCLE do PWM (0 a 100) e acione um LED verde de acordo com a potência.