15. USART

Neste capítulo, é apresentada a USART, um periférico com características muito flexíveis e amplamente utilizado na comunicação serial dos microcontroladores com o mundo externo. O seu emprego é exemplificado na comunicação do ATmega com um computador, através de uma porta USB, na comunicação entre módulos básicos de rádio frequência e na comunicação com dispositivos *bluetooth*. No Arduino, a USART é usada principalmente para a gravação do ATmega328 através do computador.

A *Universal Synchronous and Asynchronous serial Receiver and Transmitter* é um módulo de comunicação serial com inúmeras possibilidades de configurações de trabalho, o que lhe permite ser aplicada em uma infinidade de sistemas eletrônicos. Como por exemplo, nas comunicações RS232 e RS485, que apesar de não serem mais utilizadas em computadores pessoais, são, ainda, largamente usadas em sistemas industriais de controle. A grande vantagem da USART é que muitos dispositivos eletrônicos modernos suportam seu protocolo de comunicação.

15.1 USART DO ATMEGA328

As principais características do módulo de comunicação USART do ATmega328 são:

- Operação *Full Duplex* (registradores independentes de recepção e transmissão).
- Operação síncrona ou assíncrona.
- Operação síncrona com *clock* mestre ou escravo.
- Gerador de taxa de comunicação de alta resolução (*Baud Rate Generator*).

- Suporta *frames* seriais com 5, 6, 7, 8 ou 9 bits de dados e 1 ou 2 bits de parada.
- Gerador de paridade par ou ímpar e conferência de paridade por hardware.
- Detecção de colisão de dados e erros de *frames*.
- Filtro para ruído, incluindo bit de início falso e filtro digital passa-baixa.
- Três fontes separadas de interrupção (transmissão completa, recepção completa e esvaziamento do registrador de dados).
- Modo de comunicação assíncrono com velocidade duplicável.
- Pode ser utilizada como interface SPI mestre.

Para gerar a taxa de comunicação no modo mestre, é empregado o registrador UBRR0 (USART *Baud Rate Register* 0). Um contador decrescente trabalhando na velocidade de *clock* da CPU é carregado com o valor de UBRR0 cada vez que chega a zero ou quando o UBRR0 é escrito. Assim, um pulso de *clock* é gerado cada vez que esse contador zera, determinando a taxa de comunicação (*baud rate*). O transmissor dividirá o *clock* de *baud rate* por 2, 8 ou 16, de acordo com o modo programado. A taxa de transmissão de saída é usada diretamente pela unidade de recepção e recuperação de dados. Na tab. 15.1, são apresentadas as equações para o cálculo da taxa de comunicação (bits por segundo - bps) e para o cálculo do valor de UBRR0 para cada modo de operação usando a fonte de *clock* interna (para valores previamente calculados, ver a tab. C1 do apêndice C).

Tab. 15.1 – Equações para o cálculo do registrador UBRR0 da taxa de transmissão.

Modo de operação	Equação para o cálculo da taxa de transmissão	Equação para o cálculo do valor de UBRR0
Modo Normal Assíncrono (U2X0 = 0)	$TAXA = \frac{f_{osc}}{16(UBRR0 + 1)}$	$UBRR0 = \frac{f_{osc}}{16.TAXA} - 1$
Modo de Velocidade Dupla Assíncrono (U2X0 = 1)	$TAXA = \frac{f_{osc}}{8(UBRR0 + 1)}$	$UBRR0 = \frac{f_{osc}}{8.TAXA} - 1$
Modo Mestre Síncrono	$TAXA = \frac{f_{osc}}{2(UBRR0 + 1)}$	$UBRR0 = \frac{f_{osc}}{2.TAXA} - 1$

Para duplicar a taxa de comunicação no modo assíncrono, basta ativar o bit U2X0 do registrador UCSR0A. Esse bit deve ser colocado em zero na operação síncrona.

Emprega-se um sinal externo de *clock* no pino XCK para os modos síncronos de recepção escravo. A frequência máxima desse sinal está limitada pelo tempo de resposta da CPU, devendo seguir a seguinte condição:

$$f_{XCK} < \frac{f_{osc}}{4} \quad (15.1)$$

A frequência do sistema (f_{osc}) depende da estabilidade da fonte de *clock*. É recomendado usar alguma margem de segurança na eq. 15.1 para evitar uma possível perda de dados.

Quando o modo síncrono é usado (UMSEL0=1), o pino XCK será empregado como *clock* de entrada (escravo) ou saída (mestre). O princípio básico é que o dado de entrada (no pino RXD) é amostrado na borda oposta do XCK quando a borda do dado de saída é alterada (pino TXD). O bit UCPOL0 do registrador UCRS0C seleciona qual borda de XCK é usada para a amostragem do dado e qual é usada para a mudança do dado. Na fig. 15.1, o efeito do bit UCPOL0 é apresentado.

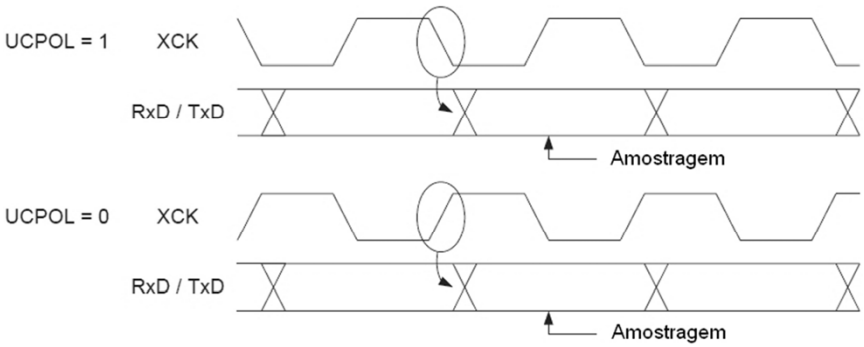

Fig. 15.1 – Efeito do bit UCPOL0 na amostragem dos dados.

O grupo de bits é transmitido/recebido em um bloco (*frame*) composto pelos bits de dados, bits de sincronização (bits de início e parada) e, opcionalmente, por um bit de paridade para a conferência de erro. A USART aceita várias combinações possíveis de formato de dados:

- Um bit de início.
- 5, 6, 7, 8 ou 9 bits de dados.
- Bit de paridade par, ímpar ou nenhum.
- Um ou dois bits de parada.

Um *frame* inicia com um bit de início, seguido pelo bit menos significativo (LSB). Seguem os outros bits de dados, num total de até 9 bits. O *frame* termina com o bit mais significativo (MSB). Se habilitado, o bit de paridade é inserido após os bits de dados, antes dos bits de parada. Após a transmissão completa dos bits, pode-se seguir outra transmissão ou aguardar-se nova transmissão. O formato do número total de bits (*frame)* é ilustrado na fig. 15.2.

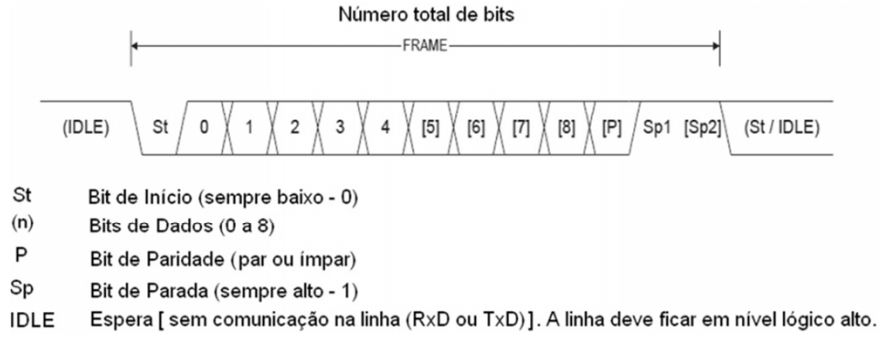

Fig. 15.2 – Formato do *frame* da USART.

O formato do *frame* é definido pelos bits UCSZ02:0, UPM01:0 e USBS0 nos registradores UCSR0B e UCSR0C. O transmissor e o receptor usam a mesma configuração.

Antes de qualquer comunicação, a USART deve ser inicializada. Os detalhes de configuração necessitam ser ajustados nos registradores específicos, detalhados posteriormente. A seguir, são apresentados exemplos para trabalho com a USART a partir de códigos extraídos do manual do fabricante.

```
//============================================================================== //
//        INICIALIZANDO A USART                                                 //
//============================================================================== //
#define FOSC    1843200        //Frequência de trabalho da CPU
#define BAUD    9600
#define MYUBRR FOSC/16/BAUD-1
//------------------------------------------------------------------------------
void main(void)
{
     ...
     USART_Init(MYUBRR);
     ...
}
//------------------------------------------------------------------------------
void USART_Init(unsigned int ubrr)
{
     UBRR0H = (unsigned char)(ubrr>>8);  //Ajusta a taxa de transmissão
     UBRR0L = (unsigned char)ubrr;
     UCSR0B = (1<<RXEN0)|(1<<TXEN0);     //Habilita o transmissor e o receptor
     UCSR0C = (1<<USBS0)|(3<<UCSZ00);    //Ajusta o formato do frame:
                                         //8 bits de dados e 2 de parada
}
//==============================================================================


//============================================================================== //
//        ENVIANDO FRAMES COM 5 A 8 BITS                                        //
//============================================================================== //
void USART_Transmit(unsigned char data)
{
     while(!( UCSR0A & (1<<UDRE0)));//Espera a limpeza do registr. de transmissão
     UDR0 = data;                   //Coloca o dado no registrador e o envia
}
//==============================================================================


//============================================================================== //
//        ENVIANDO FRAMES COM 9 BITS                                            //
//============================================================================== //
void USART_Transmit(unsigned int data)
{
     while(!(UCSR0A &(1<<UDRE0)));//Espera a limpeza do registr. de transmissão

     UCSR0B &= ~(1<<TXB80);       //Copia o 9° bit para o TXB8

     if(data & 0x0100)
          UCSR0B |= (1<<TXB80);

     UDR0 = data;                //Coloca o dado no registrador e o envia
}
//==============================================================================
```

```
//============================================================================== //
//        RECEBENDO FRAMES COM 5 A 8 BITS                                       //
//============================================================================== //
unsigned char USART_Receive(void)
{
    while(!(UCSR0A & (1<<RXC0)));  //Espera o dado ser recebido
    return UDR0;                   //Lê o dado recebido e retorna
}
//==============================================================================

//============================================================================== //
//        RECEBENDO FRAMES COM 9 BITS                                           //
//============================================================================== //
unsigned int USART_Receive(void)
{
    unsigned char status, resh, resl;    //Espera o dado ser recebido
    while(!(UCSR0A &(1<<RXC0)));//Obtêm o status do 9° bit, então, o dado do registr.
    status = UCSR0A;
    resh = UCSR0B;
    resl = UDR0;

    if(status & (1<<FE0)|(1<<DOR0)|(1<<UPE0)) //Se ocorrer um erro retorna -1
        return -1;

    resh = (resh >> 1) & 0x01;               //Filtra o 9° bit, então, retorna
    return ((resh << 8) | resl);
}
//==============================================================================

//============================================================================== //
//  LIMPANDO O REGISTRADOR DE ENTRADA (quando ocorre um erro p. ex.)            //
//============================================================================== //
void USART_Flush(void)
{
    unsigned char dummy;
    while(UCSR0A & (1<<RXC0)) dummy = UDR0;
}
//==============================================================================
```

A faixa de operação do receptor é dependente do descasamento entre a taxa de comunicação do transmissor e a sua taxa de comunicação, gerada internamente (*baud rate*). Se o transmissor estiver enviando bits muito rapidamente, ou muito devagar, ou ainda, se o *clock* gerado internamente não tiver a frequência base similar a do transmissor, o receptor não será capaz de sincronizar os *frames* relativos ao bit de início. Isso gerará um erro na taxa de recepção, que, segundo a Atmel, deve estar entre ± 1% e ± 3% (ver manual do componente, tabelas detalhadas são apresentadas no apêndice C). O erro é calculado por:

$$Erro[\%] = \frac{Taxa\ de\ Transmissão\ Aproximada}{Taxa\ de\ Transmissão\ Ideal}.100\% \qquad (15.2)$$

TRANSMISSÃO DE DADOS

Uma transmissão é iniciada ao se escrever o dado a ser transmitido no registrador de I/O da USART – UDR0. O dado é transferido de UDR0 para o registrador de deslocamento de transmissão quando ele está no estado *idle* (ocioso) ou imediatamente após o último bit de parada do frame anterior ter sido deslocado para a saída.

Se o registrador de deslocamento do transmissor está vazio, o dado é transferido do registrador UDR0 para o registrador de deslocamento. Neste momento, o bit UDRE0 (USART *Data Register Empty* 0) no registrador UCSR0A é posto em 1 lógico. Quando esse bit está em 1 lógico, a USART está pronta para receber o próximo caractere.

No ciclo de *clock* da taxa de comunicação seguinte à operação de transferência para o registrador de deslocamento, o bit de início é deslocado para o pino TXD, seguindo o dado com o LSB primeiro. Quando o último bit de parada é deslocado para a saída, o registrador de deslocamento é carregado se algum novo dado foi escrito no UDR0 durante a transmissão. Durante a carga, UDRE0 é posto em 1 lógico. Se não há nenhum dado novo no registrador UDR0 para ser enviado quando o bit de parada é deslocado, o *flag* UDRE0 permanecerá em 1 lógico até UDR0 ser escrito novamente. Se nenhum dado novo foi escrito e o bit de parada estiver presente em TXD pelo tempo de um bit, o *flag* de transmissão completa (TXC0) em UCSR0A é posto em 1.

O bit TXEN0 no registrador UCSR0B habilita o transmissor da USART quando em 1 lógico. Se esse bit é posto em 0 lógico, o pino PD1 pode ser usado para I/O de dados. Quando TXEN0 é posto em 1 lógico, o transmissor da USART é conectado a PD1, o qual é forçado a ser um pino de saída independente da configuração do bit DDD1 em DDRD.

RECEPÇÃO DE DADOS

O receptor inicia a recepção de dados quando um bit de início válido é detectado. Os bits seguintes são amostrados e deslocados para o registrador de deslocamento de recepção até o primeiro bit de parada de um *frame* ser recebido. Neste momento, o conteúdo do registrador de deslocamento é movido para o registrador de recepção e o bit RXC0 em UCSR0A é posto em 1 lógico. O dado recebido pode ser lido a partir do registrador UDR0.

Quando o bit RXEN0 no registrador UCSR0A está em 0 lógico, a recepção está desabilitada. Isso significa que o terminal PD0 pode ser usado como um pino de I/O. Quando RXEN0 está em 1 lógico, o receptor da USART está conectado a PD0, o que o força a ser uma entrada, independente da configuração do bit DDD0 em DDRD. Quando PD0 é forçado como entrada pela USART, o PORTD pode ainda ser usado para controlar o resistor de *pull-up* do pino.

REGISTRADORES DA USART

UDR0 – USART I/O *Data Register*:

Bit		7	6	5	4	3	2	1	0
	UDR0 (leitura)	RXB[7:0]							
	UDR0 (escrita)	TXB[7:0]							
Lê/Escreve		L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0

Os registradores de recebimento e envio de dados compartilham o mesmo endereço lógico. Entretanto, um leitura é feita no UDR0 de leitura e uma escrita, no UDR0 de escrita, o hardware se encarrega da distinção. Para frames com 5, 6 ou 7 bits, os bits não utilizados serão ignorados e na recepção colocados em zero. O UDR0 só deve ser escrito quando o bit UDRE0 do registrador UCSR0A estiver ativo.

UCSR0A – USART *Control and Status Register A*:

Bit		7	6	5	4	3	2	1	0
	UCSR0A	RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0
Lê/Escr.		L	L/E	L	L	L	L	L/E	L/E
Valor Inicial		0	0	1	0	0	0	0	0

Bit 7 – RXC0 – USART *Receive Complete*
Este bit é posto em 1 lógico quando um caractere recebido é transferido do registrador de deslocamento de recepção para o registrador de recepção. O bit RXC0 é posto em 0 lógico na leitura de UDR0.

Bit 6 – TXC0 – USART *Transmit Complete*
Este bit é posto em 1 lógico quando um caractere completo (incluindo o bit de parada) no registrador de deslocamento de transmissão for transferido e nenhum dado foi escrito em UDR0. Este bit é especialmente útil em interfaces de comunicação *half-duplex*, onde um aplicativo de transmissão deve entrar no modo de recepção e liberar o barramento de comunicação imediatamente após completar a transmissão. O bit TXC0 é posto em 0 lógico pelo hardware ao executar o vetor correspondente de tratamento de interrupção. Alternativamente, o bit TCX0 é limpo pela escrita de 1 lógico.

Bit 5 – UDRE0 – USART *Data Register Empty*
Este bit é posto em 1 lógico quando um caractere escrito no UDR0 é transferido para o registrador de deslocamento de transmissão. Um lógico neste bit indica que o transmissor está pronto para receber um novo caractere para transmissão. O bit UDRE0 é posto em 0 lógico na leitura de UDR0. Quando uma interrupção acionada pela transmissão de dados é usada, a rotina de interrupção por USART *Data Register Empty* deve escrever em UDR0 a fim de limpar UDRE0, se não o fizer, uma nova interrupção ocorrerá a cada vez que a rotina de interrupção terminar. UDRE0 é posto em 1 lógico durante a inicialização para indicar que o transmissor está pronto.

Bit 4 – FE0 – *Frame Error*
Indica se existe um erro no *frame* recebido. Este bit deve sempre ser zerado quando se escreve no registrador UCSR0A.

Bit 3 – DOR0 – *Data OverRun*
Ocorre quando o registrador de entrada está cheio, não foi lido e um novo bit de início é detectado. Este bit deve sempre ser zerado quando se escreve no registrador UCSR0A.

Bit 2– UPE0 – USART *Parity Error*
Indica se existe um erro de paridade no dado recebido e fica ativo até UDR0 ser lido. Este bit deve sempre ser zerado quando se escreve no registrador UCSR0A.

Bit 1 – U2X0 – *Double the* USART *transmission speed*
Este bit só tem efeito no modo de operação assíncrona.

Bit 0 –MPCM0 – *Multi-processor Communication Mode*
Habilita a comunicação com vários processadores. Quando ativo, todos os *frames* recebidos serão ignorados se não contiverem uma informação de endereço.

UCSR0B – USART *Control and Status Register B*:

Bit		7	6	5	4	3	2	1	0
	UCSR0B	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80
Lê/Escreve		L/E	L/E	L/E	L/E	L/E	L/E	L	L/E
Valor Inicial		0	0	0	0	0	0	0	0

Bit 7 – RXCIE0 – RX *Complete Interrupt Enable*
Este bit habilita a interrupção por recepção de dados completa (bit RXC0). Uma interrupção de recepção será gerada somente se o bit RXCIE0, o bit I (SREG) e o bit RXC0 estiverem ativos.

Bit 6 – TXCIE0 – TX *Complete Interrupt Enable*
Este bit habilita a interrupção por transmissão de dados completa (bit TXC0). Uma interrupção de transmissão será gerada somente se o bit TXCIE0, o bit I (SREG) e o bit TXC0 estiverem ativos.

Bit 5 – UDRIE0 – USART *Data Register Empty Interrupt Enable*
Este bit habilita a interrupção por registrador de dados vazio (bit UDRE0). Uma interrupção por registrador de dados vazio será gerada somente se o bit UDRIE0, o bit I (SREG) e o bit UDRE0 estiverem ativos.

Bit 4 – RXEN0 – *Receiver Enable*
Este bit habilita a recepção da USART. O receptor irá alterar a operação normal do pino RXD. Desabilitando o receptor, ocorrerá o esvaziamento do registrador de entrada, invalidando os bits FE0, DOR0 e UPE0.

Bit 3 – TXEN0 – *Transmitter Enable*
Este bit habilita a transmissão da USART. O transmissor irá alterar a operação normal do pino TXD. A desabilitação do transmissor só terá efeito após as transmissões pendentes serem completadas.

Bit 2 – UCSZ02 – *Character Size*
Este bit, combinado com os bits UCSZ01:0 do registrador UCSR0C, ajusta o número de bits de dados do *frame*.

Bit 1 – RXB80 – *Receive Data Bit 8*
É o nono bit de dados recebidos quando o *frame* for de 9 bits. Deve ser lido antes dos outros bits do UDR0.

Bit 0 – TXB80 – *Transmit Data Bit 8*
É o nono bit de dados a ser transmitido quando o *frame* for de 9 bits. Deve ser escrito antes dos outros bits no UDR0.

UCSR0C - USART *Control and Status Register C*:

Bit		7	6	5	4	3	2	1	0
	UCSR0C	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01	UCSZ00	UCPOL0
Lê/Escreve		L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	1	1	0

Bit 7:6 – UMSEL01:0 – USART *Mode Select*
Estes bits selecionam o modo de operação da USART, conforme tab. 15.2.

Tab. 15.2 – Ajuste dos bits UMSEL01:0 para o modo de operação da USART.

UMSEL01	UMSEL00	Modo de operação
0	0	assíncrono
0	1	síncrono
1	0	reservado
1	1	SPI mestre

Bits 5:4 – UPM01:0 – *Parity Mode*
Estes bits habilitam e ajustam o gerador de paridade e de conferência. Se habilitado, o transmissor irá gerar e enviar automaticamente o bit de paridade em cada *frame;* o receptor irá gerar o valor de paridade para comparação. Se uma desigualdade for detectada, o bit UPE0 torna-se ativo. Na tab. 15.3, são apresentadas as possíveis configurações para os bits UPM01:0.

Tab. 15.3 – Bits UPM01:0.

UPM01	UPM00	Modo de Paridade
0	0	Desabilitado
0	1	Reservado
1	0	Habilitado, paridade par
1	1	Habilitado, paridade ímpar

Bit 3 – USBS0 – *Stop Bit Select*
Este bit seleciona o número de bits de parada a serem inseridos pelo transmissor (USBS0=0, 1 bit de parada; USBS0=1, 2 bits de parada).

Bits 2:1 – UCSZ01:0 – *Character Size*
Estes bits, combinados com o bit UCSZ02 do registrador UCSR0B, ajustam o número de bits de dados no *frame* do transmissor e receptor, conforme tab. 15.4.

Tab. 15.4 – Ajuste dos bits UCSZ01:0.

UCSZ02	UCSZ01	UCSZ00	Tamanho do Caractere
0	0	0	5 bits
0	0	1	6 bits
0	1	0	7 bits
0	1	1	8 bits
1	0	0	reservado
1	0	1	reservado
1	1	0	reservado
1	1	1	9 bits

Bit 0 – UCPOL0 – *Clock Polarity*
Este bit é usado somente no modo síncrono. Deve ser zero quando o modo assíncrono é usado. Ele ajusta o sinal de *clock* (XCK) para amostragem e saída de dados, conforme tab. 15.5.

Tab. 15.5 – Ajustando a polaridade do *clock*.

UCPOL0	Mudança do Dado Transmitido (saída do pino TxD0)	Amostragem do Dado Recebido (entrada do pino RxD0)
0	borda de subida de XCK	borda de descida de XCK
1	borda de descida de XCK	borda de subida de XCK

UBRR0L e UBRR0H - USART *Baud Rate Register*

Bit		15	14	13	12	11	10	9	8
	UBRR0H	-	-	-	-	UBRR[11:8]			
	UBRR0L	UBRR[7:0]							
Bit		7	6	5	4	3	2	1	0
Lê/Escreve		L	L	L	L	L/E	L/E	L/E	L/E
		L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
Valor Inicial		0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0

Bits 15:12 – Reservado
Devem ser zero quando se escreve em UBRR0H.

Bits 11:0 – UBRR011:0: USART *Baud Rate Register*
Este é o registrador de 12 bits que contém o valor da taxa de comunicação. Qualquer transmissão em andamento será corrompida se houver mudança desse valor. Qualquer escrita atualiza imediatamente a taxa de comunicação, ver a tab. 15.1.

15.2 USART NO MODO SPI MESTRE

A USART do ATmega328 também pode ser empregada como interface SPI mestre com as seguintes características:

- Operação *Full Duplex*, transferência síncrona de dados.
- Suporta todos os modos de operação da SPI: 0, 1, 2 e 3.
- Escolha do primeiro bit para transmissão (LSB ou MSB).
- Gerador de *clock* de alta resolução.
- Frequência máxima igual à metade da CPU.
- Permite interrupções.

Nas tabs. 15.6 e 15.7, são apresentadas as equações para o cálculo da taxa de transmissão e as configurações dos bits para os diferentes modos de operação da SPI.

Tab. 15.6 – Equações para calcular a taxa de comunicação da USART no modo SPI mestre.

Equação para o cálculo da taxa de transmissão	Equação para o cálculo do valor de UBRR0
$TAXA = \frac{f_{osc}}{2(UBRR0 + 1)}$	$UBRR0 = \frac{f_{osc}}{2.TAXA} - 1$

Tab. 15.7 – Bits de configuração para os modos SPI.

UCPOL0	UCPHA0	Borda do SCK	Borda do SCK	MODO SPI
0	0	Amostragem (subida)	Ajuste (descida)	0
0	1	Ajuste (subida)	Amostragem (descida)	1
1	0	Amostragem (descida)	Ajuste (subida)	2
1	1	Ajuste (descida)	Amostragem (subida)	3

Funções exemplo são apresentadas a seguir, conforme manual do fabricante.

```
//---------------------------------------------------------------------------------------
void USART_Init( unsigned int baud )
{
     UBRR0 = 0;

     DDRD |= (1<<PD4);    //pino XCK como saída, habilita o modo mestre

     //Modo MSPI e SPI no modo 0
     UCSR0C = (1<<UMSEL01)|(1<<UMSEL00)|(0<<UCPHA0)|(0<<UCPOL0);
     UCSR0B = (1<<RXEN0)|(1<<TXEN0);     //habilita a transmissão e recepção

     // Ajuste da taxa de transmissão
     // IMPORTANTE: A taxa de transmissão deve ser ajustada após a habilitação
     UBRR0 = baud;  //a variável baud deve conter o valor da taxa
}
//---------------------------------------------------------------------------------------
unsigned char USART_TXRX(unsigned char dado)
{
     while(!( UCSR0A & (1<<UDRE0)) );    //espera a transmissão ser completada

     UDR0 = data;                        //envia o dado

     while(!(UCSR0A & (1<<RXC0)) );      //espera o dado ser recebido

     return UDR0;                        //retorna o dado recebido
}
//---------------------------------------------------------------------------------------
```

15.3 COMUNICAÇÃO ENTRE O MICROCONTROLADOR E UM COMPUTADOR

Quem começa os seus passos como programador, logo se depara com algum programa para a comunicação entre um computador e o mundo externo. Antigamente, empregavam-se as portas paralela ou serial disponíveis no hardware (onde se colocavam a impressora e o mouse, por exemplo). Na atualidade, os computadores não dispõem mais dessas portas, o domínio é das portas USB (*Universal Serial Bus*). Entretanto, isso não impede que dispositivos que usem porta paralela ou serial sejam conectados ao computador. Uma solução é o emprego de cabos conversores que podem ser ligados diretamente às portas USB. Na fig. 15.3, são

apresentados os terminais de um cabo conversor serial/USB[39] (COMTAC®) empregado na comunicação RS232.

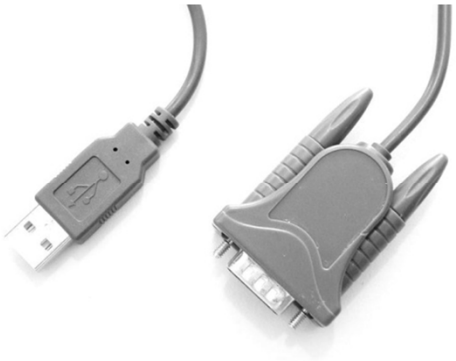

Fig. 15.3 – Terminais de um cabo para ligar um dispositivo com comunicação RS232 a um computador com entrada USB.

O fabricante do cabo conversor fornece o programa de instalação do seu dispositivo no computador *(driver*). Esse programa instala uma porta serial virtual no computador para ser utilizada pela USB. Após a instalação do *driver*, é necessário um programa para realizar a comunicação. Esse programa pode ser desenvolvido pelo usuário ou, em aplicações simples, pode-se empregar um programa gratuito. Nas versões anteriores do sistema operacional Windows XP, era disponibilizada uma ferramenta com essa função, chamada de hiperterminal. Em versões posteriores do Windows, onde essa ferramenta não existe, é necessário utilizar outro programa.

Neste trabalho, foi empregado o programa gratuito Hércules da HW *Group* (www.hw-group.com). Na fig. 15.4, é apresentada a janela de configuração para a comunicação serial, onde se define qual porta virtual

[39] O desempenho conseguido com um conversor serial/USB é inferior ao da USB isoladamente. Literalmente, a comunicação não pode ser denominada USB, pois o protocolo e as taxas de transmissão são diferentes.

(criado pelo programa do cabo conversor) será empregada (neste caso, a COM3), qual a taxa de transmissão (9600 bps) e as outras características da comunicação que se deseja realizar.

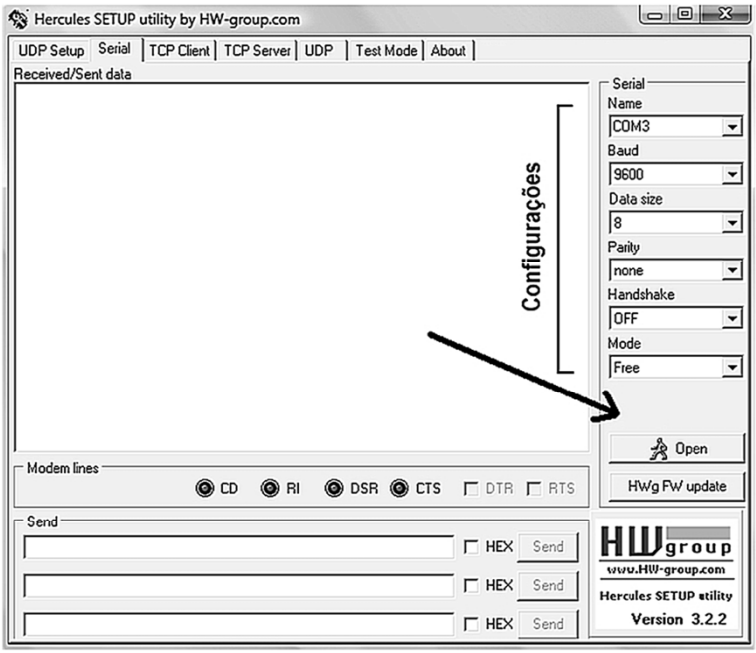

Fig. 15.4 – Configurando o programa Hercules para uma comunicação RS232.

Após todos os programas instalados e configurados, a comunicação com o dispositivo externo desejado pode ser feita. Todavia, os níveis lógicos que o circuito do cabo conversor entende, são os níveis de tensão do protocolo RS232 e, portanto, deve ser utilizado um circuito para adequar os níveis de tensão do microcontrolador aos níveis compreendidos pelo conversor, como por exemplo, o CI MAX232, como apresentado na fig. 15.5 (a alimentação desse CI é a mesma do microcontrolador). O circuito externo deve dispor do conector DB-9 fêmea para a conexão do cabo.

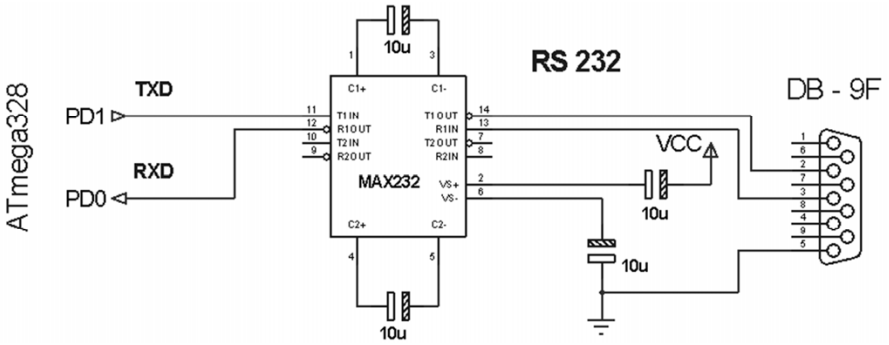

Fig. 15.5 – Circuito para adequação da tensão proveniente de um microcontrolador para a comunicação RS232.

O uso do cabo conversor é uma solução barata e de fácil aquisição. O seu porém, é a necessidade de utilização do MAX232 ou equivalente. Outra possibilidade, é o uso de um circuito integrado dedicado para realizar a comunicação diretamente com os níveis de tensão do circuito microcontrolado, como por exemplo, o FT232. Circuitos conversores desse tipo podem ser encontrados prontos para uso, mas o seu custo geralmente é superior ao dos cabos conversores.

O Arduino Duemilenove emprega um FT232 e o Arduino Uno utiliza um ATmega8U2 com USB para a comunicação com um computador (ver a fig. 3.2). Dessa forma, a plataforma Arduino pode ser ligada diretamente ao computador com o uso de apenas um cabo USB. Quando o Arduino é conectado ao computador, o *driver* do dispositivo conversor deve ser instalado e uma porta COM será associada automaticamente a ele.

Com a plataforma adequadamente instalada, basta escrever o código para o microcontrolador e conectá-lo ao computador. Na fig. 15.6, é apresentada a comunicação realizada entre o computador e o Arduino Uno. O microcontrolador transmite a mensagem inicial: "Transmitindo primeira mensagem para o computador! Digite agora – Para sair <*>". O programa microcontrolado retorna qualquer caractere recebido com os símbolos "->" seguidos do caractere. Também é possível enviar uma palavra inteira pelo

programa de comunicação, a qual aparecerá na janela do programa com a cor rosa.

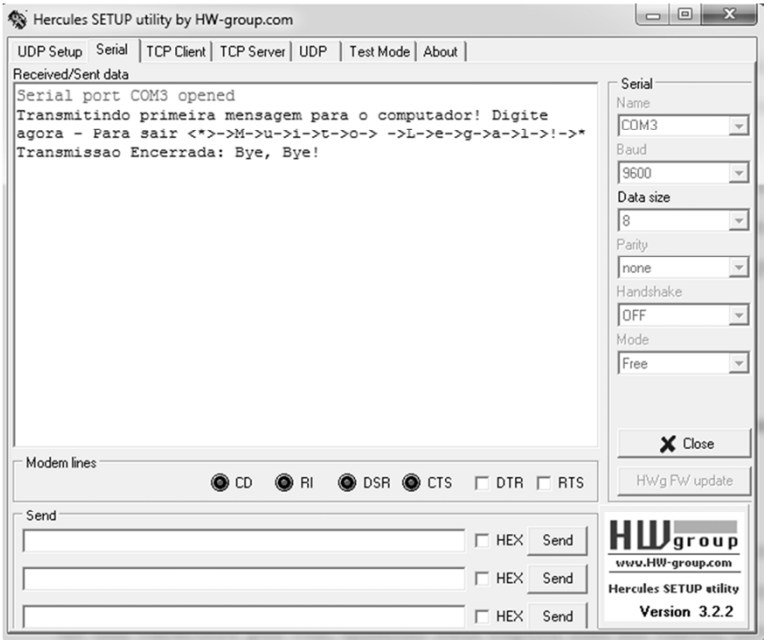

Fig. 15.6 – Janela do programa teste de comunicação entre um computador e Arduino Uno.

O programa utilizado para a comunicação no exemplo acima é apresentado a seguir. O programa foi desenvolvido para esperar o recebimento e o envio de um caractere por vez. Quando essa espera não for viável, a interrupção da USART deve ser empregada.

def_principais.h (arquivo de cabeçalho do programa principal)

```
#ifndef _DEF_PRINCIPAIS_H
#define _DEF_PRINCIPAIS_H

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

#include <avr/io.h>
#include <util/delay.h>
#include <avr/pgmspace.h>
```

```
//Definições de macros para o trabalho com bits
#define   set_bit(y,bit)  (y|=(1<<bit))
#define   clr_bit(y,bit)  (y&=~(1<<bit))
#define   cpl_bit(y,bit)  (y^=(1<<bit))
#define   tst_bit(y,bit)  (y&(1<<bit))

#endif
```

USART_PC.c (programa principal)

```
//=============================================================================== //
//             COMUNICAÇÃO SERIAL ENTRE O ARDUINO E UM COMPUTADOR                  //
//=============================================================================== //
#include "def_principais.h"
#include "USART.h"

const char primeira_msg[] PROGMEM = "Transmitindo primeira mensagem para o computador!
                                                    Digite agora - Para sair <*>\n\0";
const char segunda_msg[] PROGMEM = "\nTransmissao Encerrada: Bye, Bye!\0";
//-------------------------------------------------------------------------------
int main()
{
     unsigned char dado_recebido;

     USART_Inic(MYUBRR);

     escreve_USART_Flash(primeira_msg);

     do
     {
          dado_recebido= USART_Recebe();      //recebe caractere
          USART_Transmite('-');
          USART_Transmite('>');
          USART_Transmite(dado_recebido);     //envia o caractere recebido
     }while(dado_recebido!='*');

     escreve_USART_Flash(segunda_msg);

     while(1);//laço infinito
}
//-------------------------------------------------------------------------------
```

USART.h (arquivo de cabeçalho do USART.c)

```
#ifndef _USART_H
#define _USART_H

#include "def_principais.h"

#define BAUD    9600      //taxa de 9600 bps
#define MYUBRR  F_CPU/16/BAUD-1
#define tam_vetor    5    //número de digitos individuais para a conversão por ident_num()
#define conv_ascii   48   /*48 se ident_num() deve retornar um número no formato ASCII
                                                          (0 para formato normal)*/

void USART_Inic(unsigned int ubbr0);
void USART_Transmite(unsigned char dado);
unsigned char USART_Recebe();
void escreve_USART(char *c);
```

```
void escreve_USART_Flash(const char *c);
void ident_num(unsigned int valor, unsigned char *disp);

#endif
```

USART.c (arquivo com as funções para o trabalho com a USART)

```
#include "USART.h"
//------------------------------------------------------------------------------------
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); //Habilita a transmissão e a recepção
     UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);/*modo assíncrono, 8 bits de dados, 1 bit de
                                                              parada, sem paridade*/
}
//------------------------------------------------------------------------------------
void USART_Transmite(unsigned char dado)
{
     while (!( UCSR0A & (1<<UDRE0)) );   //espera o dado ser enviado
     UDR0 = dado;                        //envia o dado
}
//------------------------------------------------------------------------------------
unsigned char USART_Recebe()
{
     while (!(UCSR0A & (1<<RXC0)));      //espera o dado ser recebido
     return UDR0;                        //retorna o dado recebido
}
//------------------------------------------------------------------------------------
void escreve_USART(char *c)              //escreve String (RAM)
{
   for (; *c!=0;c++) USART_Transmite(*c);
}
//------------------------------------------------------------------------------------
void escreve_USART_Flash(const char *c)  //escreve String (Flash)
{
   for (;pgm_read_byte(&(*c))!=0;c++) USART_Transmite(pgm_read_byte(&(*c)));
}
//------------------------------------------------------------------------------------
//Conversão de um número em seus digitos individuais
//------------------------------------------------------------------------------------
void ident_num(unsigned int valor, unsigned char *disp)
{
     unsigned char n;

     for(n=0; n<tam_vetor; n++)
          disp[n] = 0 + conv_ascii;      //limpa vetor para armazenagem dos digitos
     do
     {
       *disp = (valor%10) + conv_ascii;  //pega o resto da divisão por 10
        valor /=10;                      //pega o inteiro da divisão por 10
        disp++;

     }while (valor!=0);
}
//------------------------------------------------------------------------------------
```

Exercícios:

15.1 – Elaborar um programa para realizar uma comunicação serial (RS232) entre o AVR e um computador, conforme exemplo da fig. 15.7. O programa deve escrever "TESTE SERIAL" na primeira linha do LCD e mandar uma mensagem ao terminal de recepção. Os caracteres digitados nesse terminal devem ser escritos na segunda linha do LCD. Quando a escrita estiver completa, deve ser apagada e reiniciada. Devem ser empregados 2400 bps, 8 bits de dados, sem paridade e 1 bit de parada.

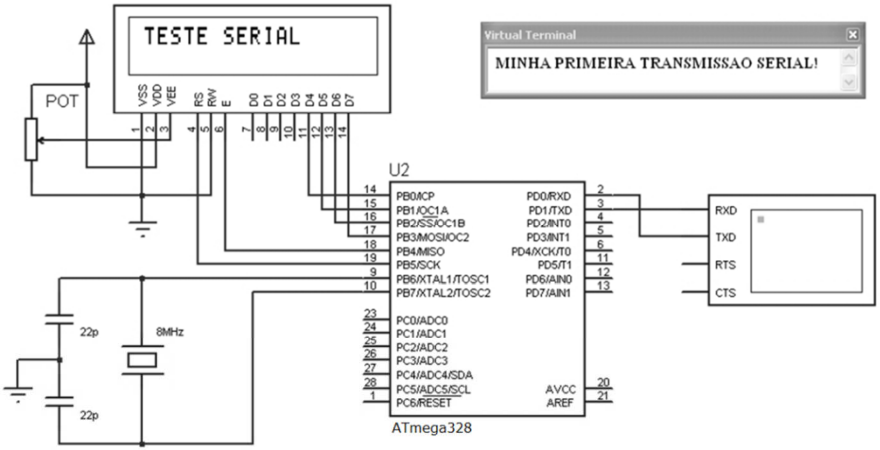

Fig. 15.7 – Comunicação serial: possível simulação no Proteus.

15.2 – Faça um programa para controlar, através do computador, 4 LEDs ligados aos pinos PB0:4 do ATmega328. Imprima na tela do hiperterminal uma mensagem semelhante a da fig. 15.8, crie sua própria imagem.

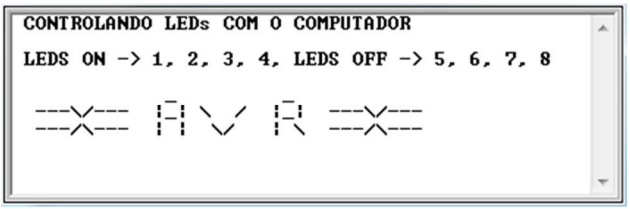

Fig. 15.8 – Mensagem recebida no computador.

A ideia é: quando o número 1 for pressionado, o LED ligado ao pino PB0 é acionado, quando o número for 5 ele será desligado (comando similares para os demais LEDs).

Para gerar a mensagem da fig. 15.8 é necessário escrever cada linha individualmente na programação, tal como:

```
const char msg1 [] PROGMEM = " CONTROLANDO LEDs COM O COMPUTADOR\0";
const char msg2 [] PROGMEM = " LEDS ON -> 1, 2, 3, 4, LEDS OFF -> 5, 6, 7, 8\0";
const char msg3 [] PROGMEM = "                 _          _ \0";
const char msg4 [] PROGMEM = "  ---\\/---  |_| \\  /  |_|  ---\\/---\0";
const char msg5 [] PROGMEM = "  ---/\\---  | |  \\/   | \\  ---/\\---\0";
```

Obs.: caso seja feita uma simulação no Proteus, para o deslocamento do cursor para uma nova linha é necessário enviar o valor 0x0D para o seu terminal virtual.

*A programação exigida por este exercício é muito utilizada para a automação de sistemas, onde é necessário enviar e receber dados ou comandos de um microcontrolador.

15.3 – Altere o programa exemplo apresentado anteriormente de tal forma que ele realize a comunicação com o computador empregando as interrupções da USART.
