Vážení zákazníci,

dovolujeme si Vás upozornit, že na tuto ukázku knihy se vztahují autorská práva, tzv. copyright.

To znamená, že ukázka má sloužit výhradně pro osobní potřebu potenciálního kupujícího (aby čtenář viděl, jakým způsobem je titul zpracován a mohl se také podle tohoto, jako jednoho z parametrů, rozhodnout, zda titul koupí či ne).

Z toho vyplývá, že není dovoleno tuto ukázku jakýmkoliv způsobem dále šířit, veřejně či neveřejně např. umisťováním na datová média, na jiné internetové stránky (ani prostřednictvím odkazů) apod.

redakce nakladatelství BEN – technická literatura
redakce@ben.cz

3 PŘÍKLADY PROGRAMŮ V JAZYCE C PRO AVR

Kdo zná svůj cíl,
cestu najde.

Laotse

Následující příklady programů ukazují práci s kompilátorem C firmy IAR pro rodinu AVR. Zvolili jsme typické aplikace, které se velmi často používají a které ukazují práci s periferními zařízeními. Ponechali jsme je co nejjednodušší, aby zbytečně neztěžovaly pochopení funkce a poskytovaly prostor pro vlastní rozšiřování. Příklady byly vytvořeny pro AT90S8515 (-v1) s modelem paměti small a byly spojovány (linkovány) pomocí spojovacího souboru lnk512s.xcl. Ve vývojovém kitu (Starter Kit) STK200 od firmy Atmel jsou tyto příklady ihned spustitelné. Bližší pokyny je možno nalézt v souborech Readme.txt v příslušných adresářích na doprovodném CD.

Navíc k základní architektuře popsané v kap. 2.3.1 přistupují nyní i periferní složky jako UART, časovač (timer) a vstupní/výstupní porty. Příslušné řídicí registry periferií jsou vysvětleny u příkladů. Zdrojový kód těchto příkladů je na doprovodném CD a smí být dál používán.

3.1 ŘÍZENÍ PŘERUŠENÍ UART

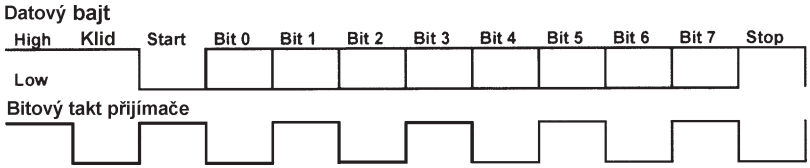
Všeobecně

Název UART znamená »Universal Asynchronous Receiver and Transmitter« (univerzální asynchronní přijímač a vysílač). Základní úlohou jednotky UART je vyslat datové slovo (bajt) převzaté od CPU a samostatně je sériově (bit po bitu) vyslat na linku TxD (Transmit), popřípadě načíst bajt z linky RxD (Receive). Protože na přijímač se přivádí jen datová linka a nikoliv bitový hodinový signál, hovoří se o asynchronním přenosu.

Když byl vyslán nebo přijat kompletní bajt, může o tom být jednotka CPU informována prostřednictvím přerušení, aby předala jednotce UART nový bajt nebo od něj bajt odebrala. Existují také ryze softwarová řešení, které celou funkci UART řeší pomocí dvou bitů portu. Pak toho má CPU přirozeně mnohem více na práci. Při použití UART však tyto funkce přebírá hardware a odpovídající kód i čas zpracování CPU se ušetří.

Sériový přenos dat začíná bitem s nejnižší váhou (LSB – Least Significant Bit). Každý bit je na lince udržován přesně určenou dobu, až se nakonec vydá poslední bit. K tomu dostává UART interní hodinový signál pro posuv bitů z mikrokontroléru.

Aby se přijímači ulehčilo rozlišování bajtů a synchronizace, je každý bajt zarámován signálem nazývaným startbit na začátku, který má vždy stav LOW a jedním nebo dvěma stopbity na konci bajtu, které mají vždy stav HIGH. Nevysílá-li se žádný bit, je linka v klidovém stavu na úrovni HIGH. Přenos jednoho bajtu vypadá takto:



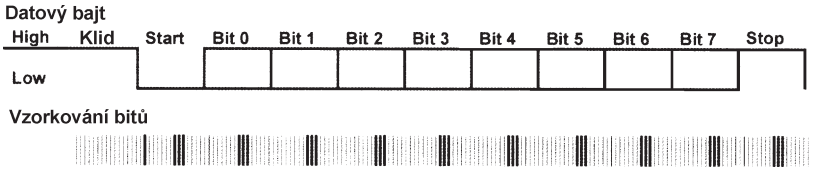
Obr. 3.1 Tvar bajtu pro asynchronní přenos dat

Na straně příjmu se musí jako první rozpoznat sestupná hrana impulzu startbitu (HIGH-LOW), a to co nejpřesněji. Všechny další vzorkovací body následujících bitů jsou vztaženy k tomuto bodu. Hrana je příznakem pro přijímač, že začíná přenos bajtu. Po čekání o délce 1,5násobku doby bitu, tedy uprostřed bitu 0, se načte stav prvního datového bitu do posuvného registru přijímače. Potom se tento registr posune o jedno místo a po dobu jednoho bitu se čeká na další bit, atd., až je načten celý bajt. Jsou zde ještě malé rozdíly v tom, jak často je bit vzorkován, v závislosti na provedení jednotky UART.

U jednotky UART rodiny AVR vzorkuje přijímač každý bit 16×, tedy 16násobkem kmitočtu nastavené přenosové rychlosti (baudrate). Pokud je přijímací linka na úrovni HIGH (klidový stav), rozpozná se první vzorkovací hodnota LOW jako okamžik hrany startbitu. V nejhorším případě s přesností 1/16 doby bitu. Nyní se startbit přesněji vyhodnotí, aby bylo možno potlačit rušivé impulzy (spikes), které by mohly vypadat jako startbit. Až po 7. vzorkování startbitu se nesmí rozpoznat žádná úroveň HIGH, jinak by byl startbit neplatný a nepoužil by se. Vlastní vyhodnocení vzorků k vyhodnocení úrovně se u všech bitů provádí vždy 8., 9. a 10. vzorkem. Nejméně dvě hodnoty musí být u startbitu na úrovni LOW, jinak se i v tomto případě startbit vyhodnotí jako neplatný.

Pokud vyhodnocení startbitu vyšlo, vzorkují se datové bity. K vyhodnocení stavů se používají vzorky 8, 9 a 10. Jako logický stav se vyhodnotí stav získaný nejméně dvěma vzorky. Po načtení všech datových bitů musí být u stopbitu nejméně dvěma ze tří vzorků (8, 9, 10) zjištěn stav HIGH. Pokud tomu tak není, nahodí se bit FE (Frame Error) ve stavovém registru USR (UART Status Register).

Přijatý bajt se přenese do registru přijatých dat UDR. Není-li tento registr prázdný, protože předchozí bajt nebyl odebrán, nahodí UART návěští Overrun Flag (OR) v registru USR. Tato návěští by měla být odpovídajícím způsobem vyhodnocována softwarem.



Obr. 3.2 Vzorkování bitů u AVR

Doba jednoho bitu je rovna převrácené hodnotě přenosové rychlosti (kmitočet):

doba bitu = 1 / přenosová rychlost (bitrate)

V technice přenosu dat se rozlišuje mezi pojmy baudrate a bitrate.

Baudrate: Udává počet změn stavu přenášeného signálu za sekundu a nazývá se také kroková rychlost nebo bitový takt. Baudrate se uvádí v jednotkách Baud.

Bitrate: Udává počet přenesených bitů za sekundu a uvádí se v jednotkách bit/s nebo bps.

U jednotky UART však nehraje tento rozdíl žádnou roli, obě hodnoty jsou stejné. To znamená, že počet přenesených bitů za cyklus jednoho kroku je roven 1 a platí 1 Baud = 1 bit/s. Jen u speciálních přenosových metod jako QAM (Quadratur Amplitude Modulation) nebo QPSK (Quad Phase Shift Keying) se v jednom cyklu přenáší více bitů, takže bitrate není rovna baudrate.

Typické přenosové rychlosti

Pro UART dostaneme při typických přenosových rychlostech s 1 startbitem, 8 datovými bity a 1 stopbitem následující doby bitu a bajtu:

Tab. 3.1

Baudrate	*Doba bitu*	*Doba bajtu (1/8/1/A)*
1200	833 μs	8,33 ms
2400	416 μs	4,16 ms
4800	208 μs	2,08 ms
9600	104 μs	1,04 ms
19200	52 μs	520 μs
38400	26 μs	260 μs
115200	8,6 μs	86 μs

Výhodou tohoto asynchronního přenosu tedy je, že se přijímač s každým novým bajtem může znovu zasynchronizovat. Tím je tento princip přenosu velmi bezpečný a jednoduchý pro softwarové i hardwarové zpracování.

Nevýhodou je, že 20 % přenosové doby (2 bity z 10) se používá pro organizaci samotného přenosu a neobsahuje žádné informace – jedná se o startbit a stopbit. Pro mnoho aplikací přenosu dat je však tato ztráta přenosové doby přijatelná, popřípadě se vykompenzuje volbou vyšší přenosové rychlosti.

Parita

Datový obsah je možno doplnit 9. bitem. Tento bit se pak používá ke kontrole paritou, a to tak, že se nastavuje na hodnotu Odd (lichá parita) nebo Even (sudá parita). Běžné jsou následující údaje k využívání parity:

Even – počet bitů s hodnotou 1 se doplňuje na sudý počet (sudá parita),

Odd – počet bitů s hodnotou 1 se doplňuje na lichý počet (lichá parita),

Mark – paritní bit je pevně nastaven na 1,

Space – paritní bit je pevně nastaven na 0,

None – nepoužívá se žádný paritní bit.

Paritu je možno odpovídajícím způsobem kontrolovat na přijímací straně a porovnáním vyvozovat závěry o chybách přenosu dat. Rozpoznatelnost chyb je však značně sporná, jak lze snadno vidět. Pokud je narušen jen jeden bit, chyba se ještě rozpozná. Ale již dvě bitové chyby mohou vést k tomu, že parita opět souhlasí. Výpověď o bezchybném přenosu je tedy velmi nejistá, takže v praxi se používají spíše jiné, podstatně bezpečnější metody zajištění dat (CRC). Existují jednotky UART, které samostatně vyrábějí a vyhodnocují paritu. Nikoliv ovšem u AVR. Tam je sice 9. bit pro přenos připraven, ale tvoření a vyhodnocování parity musí být zajištěno vlastním softwarem. Protože kontrola paritou se provádí poměrně zřídka, je toto řešení jistě dobrým kompromisem. Nejčastější případ použití asynchronního přenosu dat je 1 startbit, 8 datových bitů, 1 stopbit a žádná parita, stručně **8N1**.

Full duplex/half duplex

Tyto dva pojmy označují chování při vysílání a přijímání dat, popřípadě přenosový protokol dvou komunikujících partnerů:

Full duplex (plný duplex): Vysílání a přijímání je možno provádět současně. Zatímco jedna MCU ještě vysílá, může již probíhat odpověď. Tím je přenosový čas podstatně efektivněji využíván a minimalizován.

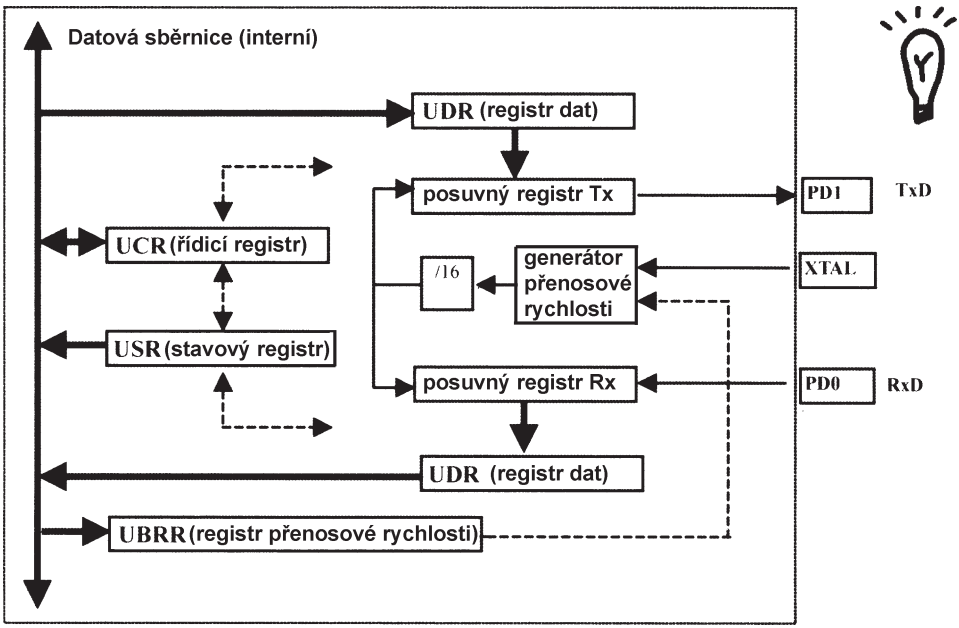
Half duplex (poloduplex): Při jednodušším poloduplexním přenosu neprobíhá vysílání a přijímání současně. Teprve když mikrokontrolér dokončí vysílání, je opět připraven k příjmu. Tato forma přenosu je jednodušší a lze ji realizovat s menším vynaložením prostředků na program a datové buffery. Převážně se používá tento způsob. Způsob přenosu určuje vlastní software. U jednotky UART AVR jsou v každém případě možné obě varianty.

Obr. 3.3 ukazuje zjednodušené blokové schéma jednotky UART v AVR. Vývod RxD (Receive Data – příjem dat) a vývod TxD (Transmit data – vysílání dat) jsou externě přístupné vývody UART. Na všechny registry je možno se obracet přes adresu registru I/O nebo přes adresu RAM. Připomínáme: všechny registry u AVR se promítají i do oblasti RAM.

Datový registr UDR slouží k vysílání a přijímání bajtů a je fyzicky rozdělen do dvou registrů, které se adresují se stejnou adresou. Liší se jen způsobem přístupu pro čtení/zápis. Při zápisu (write) se uplatňuje vysílací registr (Tx) a při čtení přijímací registr (Rx). V závislosti na zápisu/čtení se tedy uplatňují dva zcela odlišné registry.

Tab. 3.2

UDR	*Datový registr*	*bit 7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
Reg. adr.	**0x0C**	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
RAM adr.	**0x2C**	MSB							



Obr. 3.3 Blokové schéma UART AVR

Tab. 3.3 Stavový registr UART

USR	***Stavový registr***	***bit 7***	***6***	***5***	***4***	***3***	***2***	***1***	***0***
Reg. adr.	**0x0B**	R	R	R	R	R	R	R	R
RAM adr.	**0x2B**	RXC	TXC	UDRE	FE	OR	--	--	--

1 = v UDR je přijatý bajt (nezávisle na chybě rámu (frame error)
0 = žádný bajt

1 = vyslání bajtu je hotovo a v UDR není žádný nový bajt
0 = vysílání je ještě aktivní

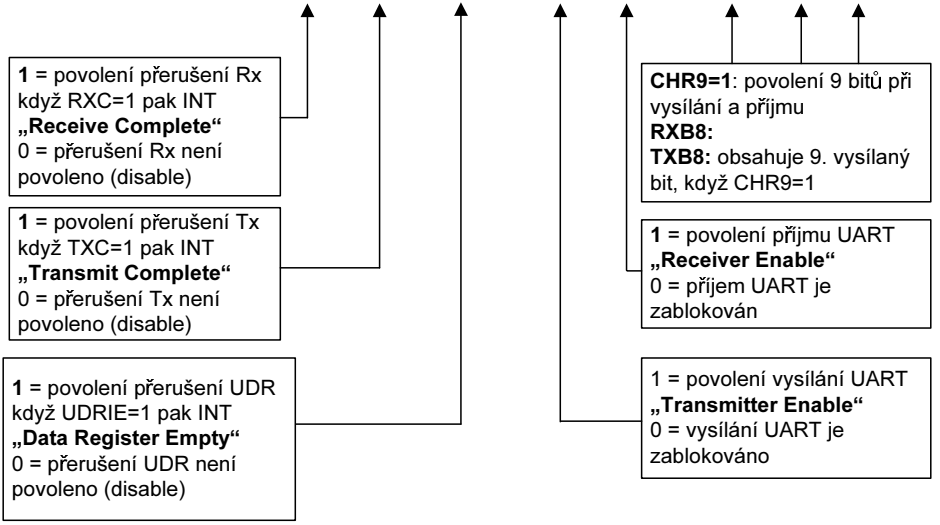
1 = vysílaný bajt z UDR je převzat do Tx registru, UDR znovu naplnit!
0 – UDR není ještě volný

1 = zjištěna chyba rámu (frame error) (žádný stopbit)
0 = není chyba

1 = přeběh přijímacího bufferu, UDR byl přepsán
0 = není přeběh (overrun)

Tab. 3.4 Řídicí registr UART

UCR	Řídicí registr	bit 7	6	5	4	3	2	1	0
Reg. adr.	0x0A	R/W	R/W	R/W	R/W	R/W	R/W	R	W
RAM adr.	0x2A	RXCIE	TXCIE	UDREIE	RXEN	TXEN	CHR9	RXB8	TXB8



Tab. 3.5 Registr rychlosti přenosu UART

UBRR	Registr rychlosti přenosu	bit 7	6	5	4	3	2	1	0
Reg. adr.	0x0A	W	W	W	W	W	W	W	W
RAM adr.	0x2A	MSB							LSB

Přenosová rychlost (Baudrate) se získává pomocí jednoduchého děliče kmitočtu z kmitočtu krystalu hodin AVR. Frekvenční dělič se skládá z pevného děliče 16 a nastavitelného děliče 0 až 255. Tato volně nastavitelná část se zanáší do registru UBRR. Potřebný dělič je možno vypočítat pomocí následujícího vzorce:

*UBRR / (16 * Baudrate) – 1*

Pro některé hodinové kmitočty a některé přenosové rychlosti jsou dělicí hodnoty UBRR uvedeny v *tab. 3.2*. Požadované rychlosti přenosu ovšem nelze přesně odvodit z jakýchkoliv hodinových kmitočtů krystalu, protože UBRR je celočíselný dělič. Chyba přenosové rychlosti by však neměla být větší než 2 %, neboť tato chyba se sčítá s každým bitem a u stopbitu (10. bit) může být již 20 %. K tomu přistupuje ještě nepřesnost při rozpoznávání sestupné hrany ve startbitu, která činí asi 6 %. Tak dostaneme celkovou chybu 26 %. Pro 3 z 16 vzorkovacích časových bodů by bit měl být stabilní, to je ještě dalších 18,75 %. Tím se již dostáváme do blízkosti 50 % bitové doby a vzorkuje se ve středu bitu. Proto platí, že chyba přenosové rychlosti by neměla být větší než 2 %.

Zjistí-li se tedy lichá hodnota UBRR, měla by se ještě jednou zjistit skutečná přenosová rychlost a procentuální odchylka se zaokrouhlenou hodnotou UBRR.

*Baudrate = fclock/16 * (UBRR – 1)*

Tab. 3.6 Hodnoty děliče UBRR

Krystal [MHz]	***1200***	***2400***	***9600***	***19 200***	***38 400***	***115 200***
1,8432	96	47	11	5	3	0
4,0	208 (0,6 %)	103 (0,2 %)	25 (0,2 %)	12 (0,2 %)	-- (> 2 %)	-- (> 2 %)
7,3728	--	191	47	23	12	3
8,0	--	207 (0,2 %)	51 (0,2 %)	25 (0,2 %)	-- (> 2 %)	-- (> 2 %)
14,7456	--	--	95	47	24	7

Jestliže přenosovou rychlost nelze přesně nastavit, je třeba uvést i procentuální odchylku. Při vysokých kmitočtech krystalu nelze nízké přenosové rychlosti nastavit, protože 8bitový dělič UBRR pak již nestačí.

Příklad programování UART

UART je možno programovat různými způsoby. Nejjednodušší způsob je jistě polling, u kterého se prostřednictvím UART předává bajt a stále se provádějí dotazy, zda již byl odeslán. Polling lze použít i v opačném směru a dotazovat se, zda bajt již přišel. Takové příklady ovšem mají v praxi jen malý význam, neboť výhoda samostatně pracující jednotky UART se tím ztrácí. Následující jednoduchý příklad ukazuje, že ani přerušením řízený UART nemusí být složitý a náročný. Může současně sloužit jako základ pro vlastní rozšíření programů pro UART.

Základem jsou dvě samostatné rutiny přerušení, jedna pro příjem UartRxIntHandler(void) a druhá pro vysílání UartTxHandler(void). Každá z funkcí přerušení má svůj vlastní datový buffer o velikosti 8 bajtů, který je navržen jako kruhový buffer. Jakmile se dosáhne konce bufferu, začne se do něj opět od začátku zapisovat nebo z něj číst. Velikost bufferu by proto měla být navržena na hodnotu modulo 2 (2, 4, 8, 16, …) , neboť tak lze po přeběhu znovu provádět justaci s nejmenšími náklady. Datový buffer však nevolte příliš velký, vždy dbejte na to, aby zůstal dostatek RAM pro jiné funkce.

UART.c

```
/* Include-soubory
 ------------------------*/
#include <io8515.h> // Definice registrů pro AT90S8515
#include <ina90.h> // Intrinsické funkce pro ICCA90

/* UCR – UART Control Register: Definice bitů
 ---------------------------------------------------------------------------------------------*/
#define   RXCIE   0x80   // Bit 7
#define   TXCIE   0x40   // Bit 6
#define   UDRIE   0x20   // Bit 5
```

```
#define   RXEN    0x10   // Bit 4
#define   TXEN    0x08   // Bit 3
#define   CHR9    0x04   // Bit 2
#define   RXB8    0x02   // Bit 1
#define   TXB8    0x01   // Bit 0

/* USR – stavový registr UART: definice bitů
 ------------------------------------------------------------------------------*/
#define   RXC     0x80   // Bit 7
#define   TXC     0x40   // Bit 6
#define   UDRE    0x20   // Bit 5
#define   FE      0x10   // Bit 4
#define   OVR     0x08   // Bit 3

/* UBRR – registr přen. rychlosti UART: definice přen. rychl. pro 4 MHz
 ------------------------------------------------------------------------------*/
#define   BAUD_1200    208
#define   BAUD_2400    103
#define   BAUD_9600    25
#define   BAUD_19200   12

/* Definice datového bufferu UART
 ------------------------------------------*/
#define BUFFER_SIZE   8       // velikost modulo 2: jen 2,4,8,16,32 Bajtů
#define BUFFER_MASK           (BUFFER_SIZE-1)

/* Obecné definice typů
 ------------------------------------------------------------------------------*/
typedef enum{ FALSE=0, TRUE=1} Boolean_t ;

typedef struct{     unsigned char UartData[ BUFFER_SIZE ];
                    unsigned char NumberOfBytes;
                    unsigned char UartStatus; } Uart_t;

/* Statické proměnné
 ---------------------------*/
static unsigned char              RxBuf[ BUFFER_SIZE ];
static volatile unsigned char     RxProducer;      // příjem: Int-Service je
                                                   // výrobce
static volatile unsigned char     RxConsumer;      // příjem: Hlavní program je
                                                   // konzument
static unsigned char              TxBuf[ BUFFER_SIZE ];
static volatile unsigned char     TxProducer;      // vysílání: Hlavní program
                                                   // je výrobce
static volatile unsigned char     TxConsumer;      // vysílání: Int-Service je
                                                   // konzument
```

```
/* Prototypy funkcí
-------------------------------------*/
void InitUart( unsigned char baudrate );
Boolean_t ReceiveData( Uart_t * Data );
Boolean_t TransmitData( Uart_t * Data );
void Delay( void );

/* Input Pins, Port B:  sfrb PINB  = 0x16;
Data Direction Register, Port B:  sfrb DDRB  = 0x17;
Data Register, Port B:  sfrb PORTB  = 0x18; */

void Delay( void )
{
        unsigned int i= 0x7FFF;
        while(--i);
}

/* Main – jednoduchý příklad  použití pro UART a Timer
 ---------------------------------------------------------------------------------------------*/
void C_task main( void )
{
        Uart_t UartData;

        DDRB  = 0xFF ;              // Konfiguruje Port B jako výstup
        PORTB = 0xAA;

        InitUart( BAUD_9600 );    // Baudrate = 9600 Baud při frekvenci krystalu
                                  // 4MHz
        _SEI();                   // Povolení všech přerušení => Enable UART
                                  // Interrupts
        for( ;; )                 // Nekonečná smyčka
{
                if( ReceiveData( & UartData ) )
                {
                        TransmitData( & UartData ); // Echo přijatého bajtu
                }

        PORTB++;            // led switch
        Delay();
 }
}

/* Inicializace UART
 ------------------------------------------------*/
void InitUart( unsigned char baudrate )
{
```

```
        UBRR = baudrate;                // nastavit Baudrate

        // povolení přerušení UART
        UCR =  TXEN | RXEN | RXCIE; // ==> Vysílání/příjem/Empf.INT

        RxConsumer  = 0;                // Vynulování indexu bufferu
        RxProducer  = 0;
        TxConsumer  = 0;
        TxProducer  = 0;
}

/*  Empfangs-Interrupt Handler
 ---------------------------------------- */
interrupt [UART_RX_vect] void UartRxIntHandler( void )
{
unsigned char data;

data = UDR;                     // čtení přijímaných dat
RxBuf[RxProducer] = data;       // uložení přijatého bajtu do bufferu
RxProducer++;                   // nastavit index Bufferu
RxProducer &=BUFFER_MASK; // Index omezit a popř. na začátek bufferu

if (RxProducer == RxConsumer )
{
            // Chyba! přeběh přijímacího bufferu
        }
}

/*  Sende-Interrupt Handler
 ---------------------------------------- */
interrupt [UART_UDRE_vect] void UART_TX_interrupt( void )
{
        // Test, zda vysílání není hotovo
        if ( TxConsumer != TxProducer )
{
            ++TxConsumer                    // aktualizovat index bufferu
            TxConsumer &=BUFFER_MASK;  // omezit index a ev. na začátek
                                            // bufferu
            UDR = TxBuf[TxConsumer];        // předání bajtu
        }
  else
{
            UCR &= ~UDRIE;          // Zablokovat přerušení UDRE interrupt
        }
}
```

```
/* přijímací funkce UART
 ---------------------------------------- */
Boolean_t ReceiveData( Uart_t * Data )
{
        unsigned char i=0;

        // Smyčka k předání přijatých dat
        while (( RxConsumer != RxProducer ) && ( i < BUFFER_SIZE ) )
        {
                Data->UartData[i++] = RxBuf[RxConsumer++];

                RxConsumer &=BUFFER_MASK;       // omezit Consumer index a
                                                // ev. na začátek bufferu
        }

        Data->NumberOfBytes = i ;

        if( i ) return TRUE ;
        else return FALSE ;
}

/* Vysílací funkce
---------------------------------------*/
Boolean_t TransmitData( Uart_t * Data )
{
        unsigned char i=0;

        while (( i < Data->NumberOfBytes ) && ( i < BUFFER_SIZE ) )
        // předání přijímaných dat
 {
                TxBuf[TxProducer++] = Data->UartData[i++] ;

                TxProducer &=BUFFER_MASK; // omezit Producer-Index a
                                          // popř. na začátek bufferu
 }

        UDR = TxBuf[TxConsumer];          // spustit vysílání
        ++TxConsumer;
        TxConsumer &=BUFFER_MASK;       // omezit Consumer-Index

        UCR |=  UDRIE ;
        Data->NumberOfBytes = 0;
        return TRUE ;
}
```