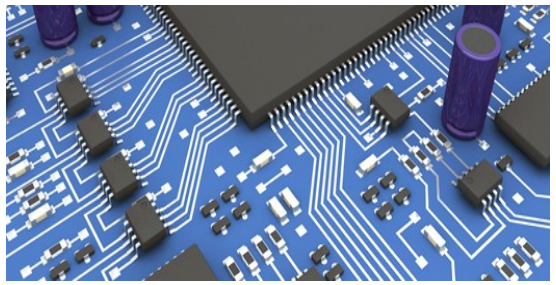
Aplikace Embedded systémů v Mechatronice



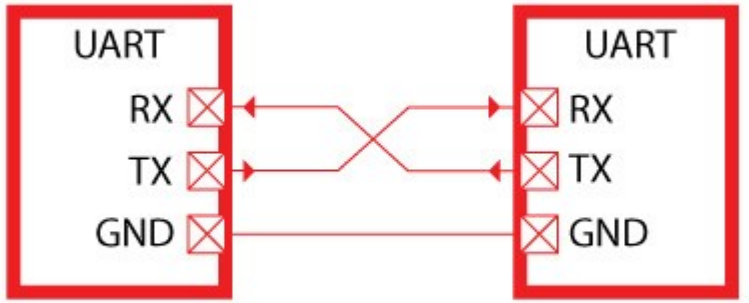
MPLAB® X IDE

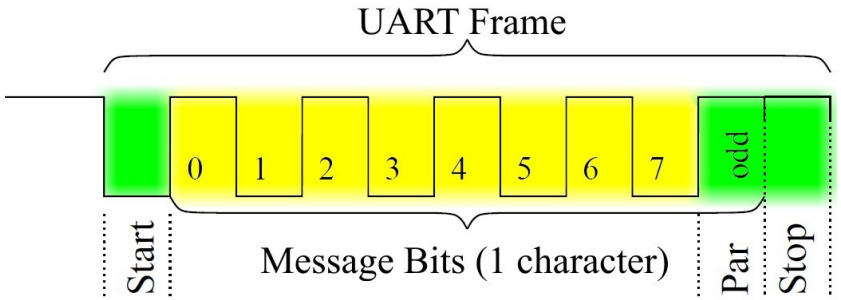
Michal Bastl
A2/713a

- UART (universal asynchronous receiver/transmitter) je nízkoúrovňový komunikační protokol, v kontextu MCU pak logický obvod, který tento protokol realizuje.
- UART definuje komunikační protokol, respektive chování logického obvodu (periferie).
- Protokol UART je v principu typu point-to-point, tedy komunikují spolu dvě protistrany existuje i rozšíření pro vícestranou komunikaci.
- UART je v základu asynchronní, tedy strany nejsou vzájemně synchronizovány hodinovým signálem. Pro přenos dat se používá vždy jeden vodič pro odesílání a pro příjem dat. Často se nezívají TX-transmit a RX-receive.
- Komunikace je sériová a je přenášena po jednotlivých bitech. Zpráva má typicky jeden bajt, tedy 8-bit.
- Délka každého bitu je dána zvolenou komunikační rychlostí v baudech, což je modulační rychlost.
- Vodič má v klidovém stavu logickou 1 a začíná start bitem, což je opak, tedy logická nula.
- Následuje série datových bitu, tedy typicky 8-bit.
- Datové slovo může být doplněno bitem paritním. Parita je jednoduchou kontrolou, která může rozeznat chyby. Je sudá, nebo lichá. Počet jedniček ve zprávě je sudý a nebo lichý. To zařídím právě nastavením paritního bitu.
- Konec je signalizován stop bitem a ten má klidovou úroveň tedy log 1.

UART

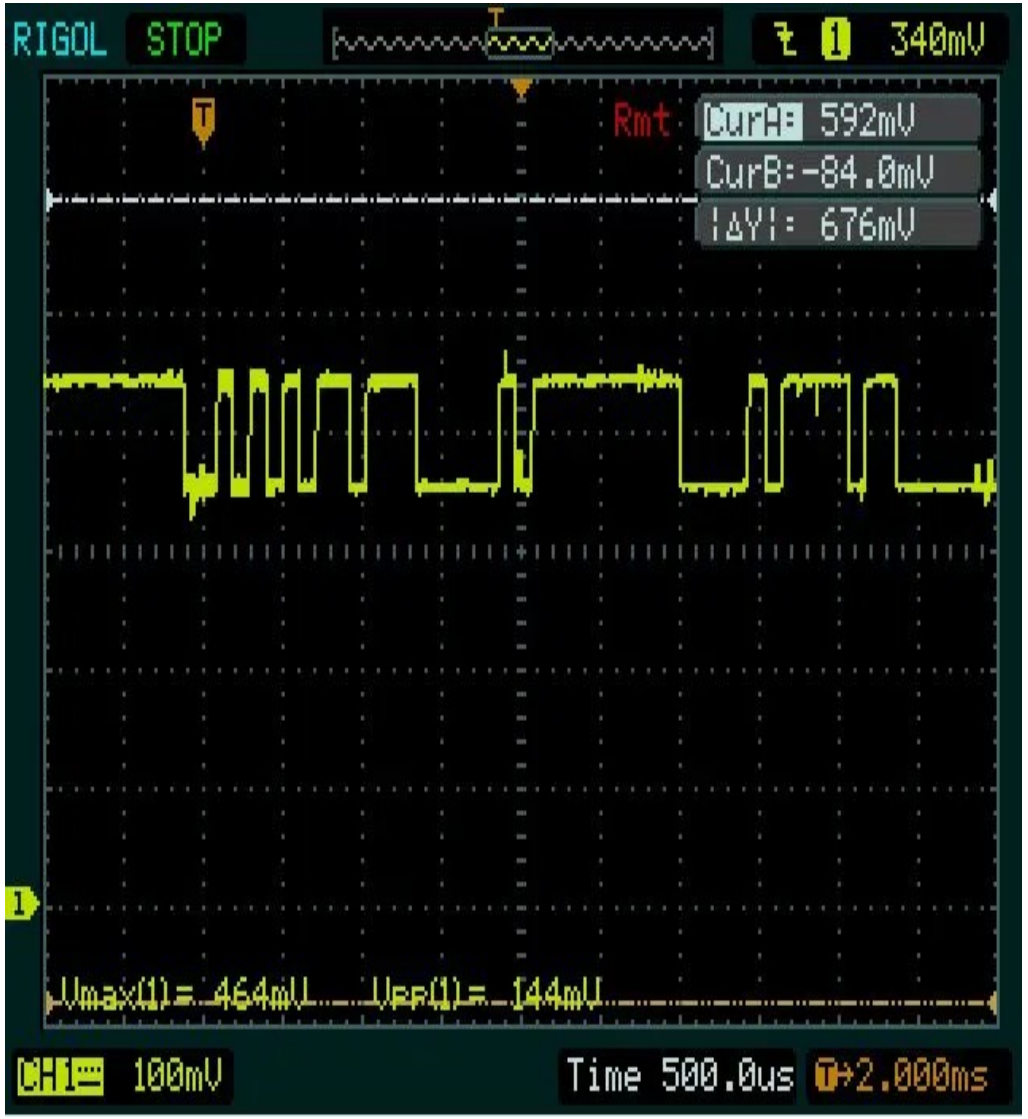
- UART *Universal Asynchronous Receiver and Transmitter*
- Jedná se o sériovou sběrnici
- Asynchronní znamená, že není sdílený CLOCK signál pro komunikující zařízení
- Je tedy nutné znát modulační rychlost na zařízeních
- zařízení by však měli mít společnou GND viz obrázek
- Standardně 8-bit dat na zprávu
- Zpráva je doplněna o start a stop bit a případně kontrolu parity





UART

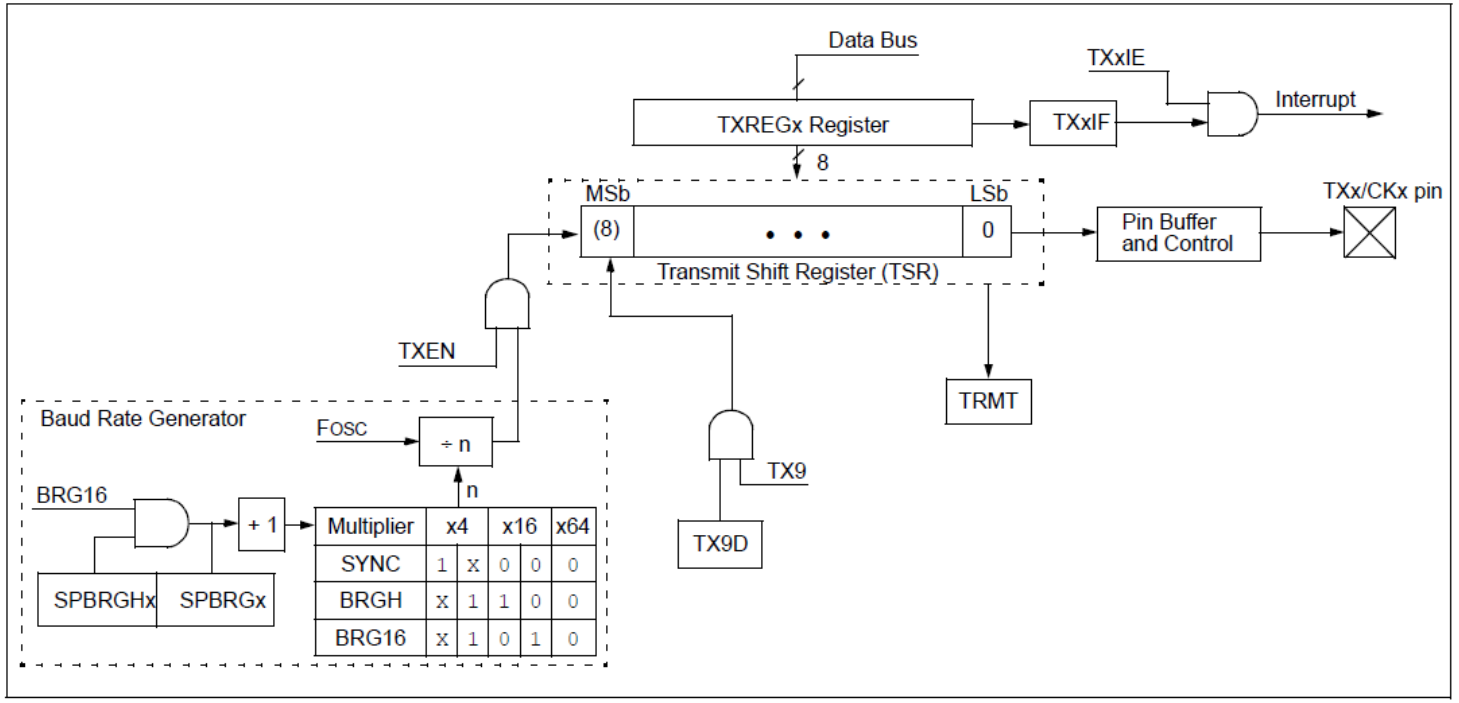
- Sběrnice ke komunikaci používá rozdílné úrovně napětí na vodičích.
- TX – vysílání.
- RX – příjem.
- V našem případě je 3,3V logická 1 a 0V log 0
- Začátek zprávy je uvozen Start bitem 0 a konec Stop bitem 1
- Může obsahovat paritní byt
- Parita je sudá, nebo lichá. Ve zprávě je sudý počet jedniček, nebo lichý (zajistím právě paritním bitem)



UART

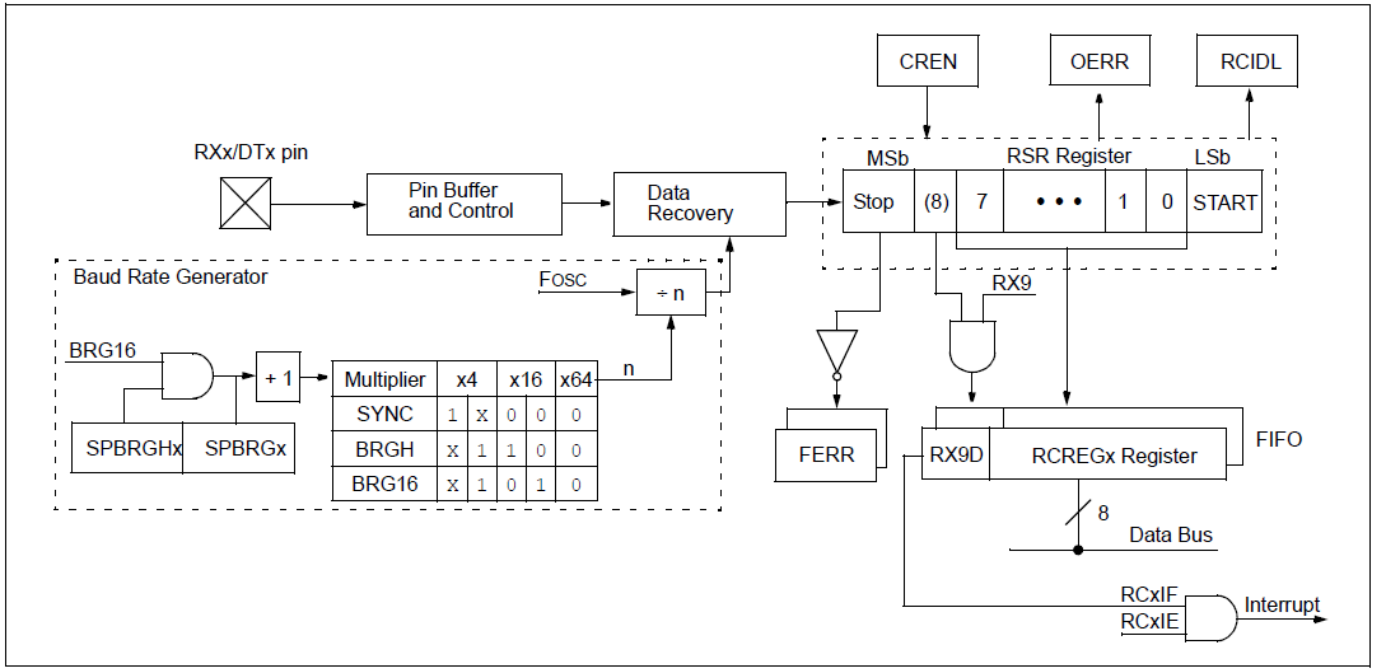
- TSR je odesílací posuvný registr a programátor s ním nepracuje
- Programátor zapisuje datovou zprávu do registru TXREGx
- Periferie obsahuje generátor modulační rychlosti
- TXxIF je vystaven do logické jedničky, dokud nedojde k posunu mezi registry TXREGx a TSR.
- Potom dojde automaticky k odeslání
- Mohu tedy kontrolovat, kdy je možné data zapsat.

FIGURE 16-1: EUSART TRANSMIT BLOCK DIAGRAM



UART

- RSR je posuvný příjmací registr, programator jej přimo nepoužívá
- Používá RCREGx, který má formu FIFO fronty a uchová dvě zprávy
- Generátor modulační rychlosti je stejný jako pro odesílání
- RcxIF bit je vyvolán, pokud přijímací registr obsahuje data
- Lze tedy vyvolat přerušení na příchod dat
- BIT OERR nás informuje, že došlo k přetečení přijímacího FIFO registru a periferie se zastaví
- Pomůže vypnout a zapnout SPEN bit



Inicializace UART periferie

REGISTER 16-1: TxSTAx: TRANSMIT STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN(1)	SYNC	SENDB	BRGH	TRMT	TX9D
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

```
TXSTAbits.TXEN = 1;      // enable TX
TXSTA1bits.SYNC = 0;
```

- V registru TxSTAx je pro základní mod třeba nastavit TXEN bit
- TX9 je nastavení režimu 9bitu
- TX9D je potom tento bit obsahující např. Paritu
- Je zde SYNC bit, kterým nastavujeme mod přenosu (v našem případě 0 async)

Inicializace UART periferie

REGISTER 16-2: RCSTAx: RECEIVE STATUS AND CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-0
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

- SPEN je seriál port enabled
- CREN je zapnutí příjmání
- OERR bit detekující přetečení
- Pokud došlo k přetečení je třeba resetovat periferii
- To se dela tak, že ji vypnu a zapnu

```
RCSTAbits.SPEN = 1;     // enable UART peripheral
RCSTAbits.CREN = 1;     // enable RX (aka Continuous receive)
```

Inicializace UART periferie

1)Pozor na ANSELx registr u pinu RX/TX v našem případě je to RC6/7

2)Nastavení pinů pomocí TRISx jako vstupu

3)Nastavení modulační rychlosti do SPBRG vzoreček v datasheetu

4)Nastavení synchronizace SYNC bitu na 0

5)Zapnutí periferie TXEN-vysílač; CREN-příjmač; SPEN-periferie

6)Pokud chci využívat přerušení na příchod znaku tak RC1IE, příznak je RC1IF

Inicializace UART periferie

Piny RX a TX je třeba v TRISx registru nastavit jako I/O

```
ANSELC = 0x00;        // vypnuti analogovych funkci na PORTC
TRISCbits.TRISC6 = 1;   // TX pin jako vstup
TRISCbits.TRISC7 = 1;   // rx pin jako vstup

/*baudrate*/
SPBRG1 = 51;          // (32_000_000 / (64 * 9600)) - 1

TXSTA1bits.SYNC = 0;     // nastaveni asynchroniho modu
RCSTA1bits.SPEN = 1;    // zapnuti UART
TXSTA1bits.TXEN = 1;    // zapnuti TX
RCSTA1bits.CREN = 1;    // zapnuti RX
```

Uart je již „komplexnější“ periferie.
Na PIC18 může fungovat i v synchronním módu
Omezíme se na základní nastavení!!

EXAMPLE 16-1: CALCULATING BAUD RATE ERROR

For a device with FOSC of 16 MHz, desired baud rate of 9600, Asynchronous mode, 8-bit BRG:

$$Desired\ Baud\ Rate = \frac{FOSC}{64([SPBRGHx:SPBRGx] + 1)}$$

Solving for SPBRGHx:SPBRGx:

$$X = \frac{\frac{FOSC}{Desired\ Baud\ Rate}}{64} - 1$$

$$= \frac{\frac{16000000}{9600}}{64} - 1$$

$$= [25.042] = 25$$

$$Calculated\ Baud\ Rate = \frac{16000000}{64(25 + 1)}$$

$$= 9615$$

$$Error = \frac{Calc.\ Baud\ Rate - Desired\ Baud\ Rate}{Desired\ Baud\ Rate}$$

$$= \frac{(9615 - 9600)}{9600} = 0.16\%$$

UART použití

Zápis dat na sběrnici:

```
while(1){
   If ( RC1IF ){
      LATD2 ^= 1;     // LED
      TXREG1 = RCREG1;     // precist a poslad zpet
   }
}
```

- Podmínka kontroluje zda přejímací registr obsahuje data
- Pokud převrátím led
- Překlopím data z přejímacího registru do vysílacího

UART použití

```
void putch(unsigned char data){
    while(!TX1IF);
    TXREG1 = data;
}

while(1){
     __delay_ms(500);
     printf("ahoj\n");
}
```

- Funkci putch(char) mohu používat pro zápis bajtu
- Tutot funkci však používá i funkce printf ze standardní knihovny
- Ve funkci je čekání Busy waiting dokud není TXREG připraven na zápis
- Poznám to kontrolou příznaku přerušení TX1IF

UART použití

Pomocí přerušení:

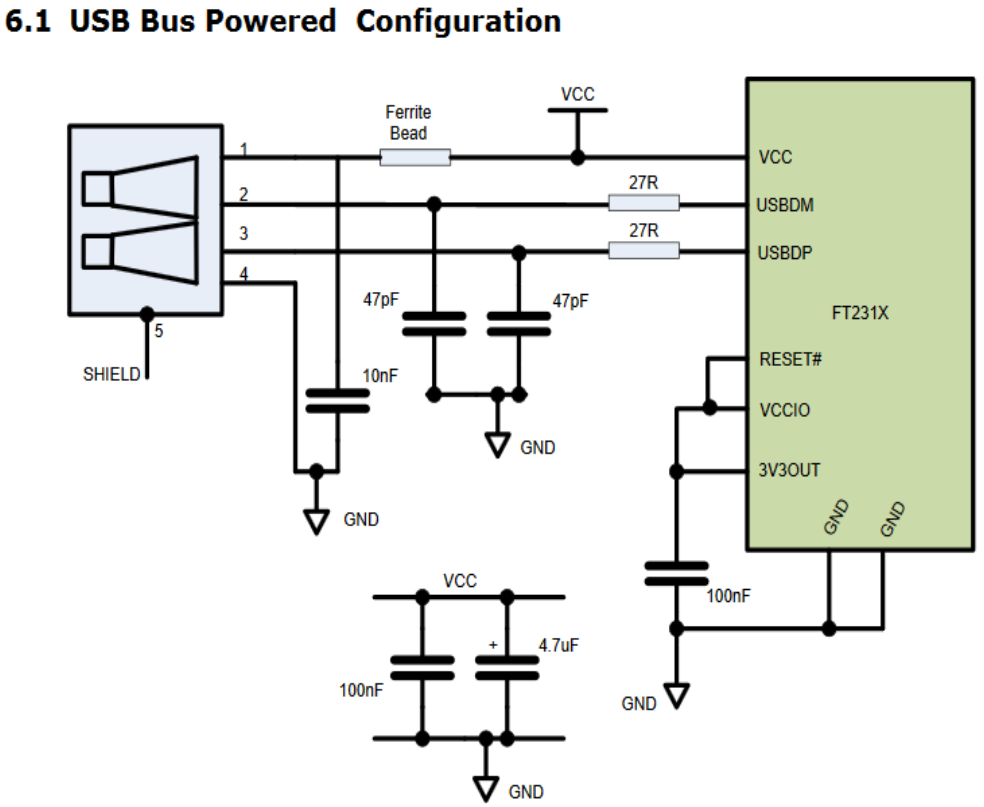
```
  RCIE = 1;          //RX interrupt enable
  PEIE = 1;          // global interrupt enable
  GIE = 1;           // peripheral interrupt enable

void __interrupt() RC_ISR_HANDLER(void){

    if(RC1IF && RC1IE){
       TXREG1 = RCREG1;        //vymena dat
        RC1IF = 0;             //smaze se i při cteni RCREG

    }

}
```

- Pokud chci pro příjem využít přerušení, tedy vyzvednout data krátce po jejich příchodu
- Stačí zapnout globální povolení GIE
- PEIE přerušení od periferii
- A přerušení na příchod znaku RC1IE pro UART1
- V přerušení pak mohu zkontrolovat, zda je to opravdu ono vyvolání a vyzvednout znak z registru RCREG

Hardware

Zákaznické obvody pro UART → USB:

- sběrnice UART lze pomocí specializovaných obvodů převést na usb
- na EduKitu je k tomuto účelu použit obvod FTDI 231
- tento obvod obsahuje vlastní mikrokontroler a lze jej pomocí obslužné aplikace nastavovat
- lze mu například přiřadit jméno, zvolit funkce pro vlastní GPIO apod
- Samotný obvod má cenu 40kč
- V podstatě se jedná o převodník UART<->USB



Zadání 2. DÚ

Zadání:

Vytvořte jednoduchý model digitálně řízeného signalizačního majáku:

maják má tři základní režimy: svítí trvale, bliká, nesvítí
maják má nastavitelnou rychlost blikání v rozsahu 1-20 Hz
maják je řízen prostřednictvím rozhraní UART pomocí sady příkazů, z nichž každý začíná písmenem A a končí tečkou '.'.

- Odevzdávejte ve formatu: Prijmeni_Jmeno_body.zip

Maják je obsluhován pomocí této sady příkazů:

AC. (continuous) – nastaví režim kontinálního svícení
AB. (blink) – nastaví režim blikání
AON. – start svícení, dle nastaveného režimu
AOFF. – vypne svícení, blikání
ASnn. (sequence) – maják nn-krát zabliká a poté se vypne
AFnn. (frequency) – nastavení frekvence blikání
AF?. – vypíše nastavenou hodnotu frekvence
A?. – vypíše stav majáku (zapnuto/vypnuto + nastavený režim)

Pozn:
Přepnutí režimu v zapnutém stavu způsobí okamžitou změnu operace.
blikání emulujte na LEDkách na kitu