Dílenská praxe

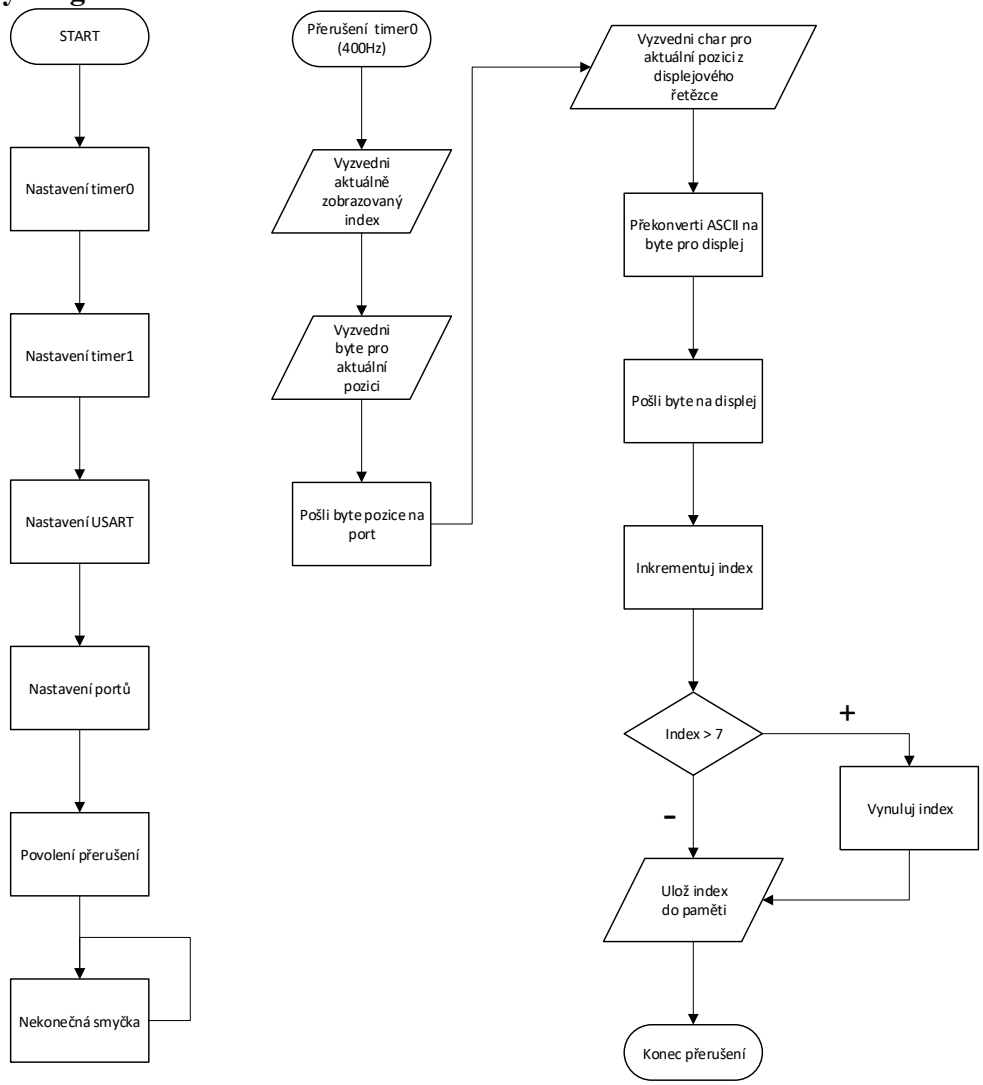
A4	1. Sériová komunikace		
Petřík Vít		1/16	Známka:
11.9. 2019	Datum odevzdání:	2.10. 2019	Odevzdáno:

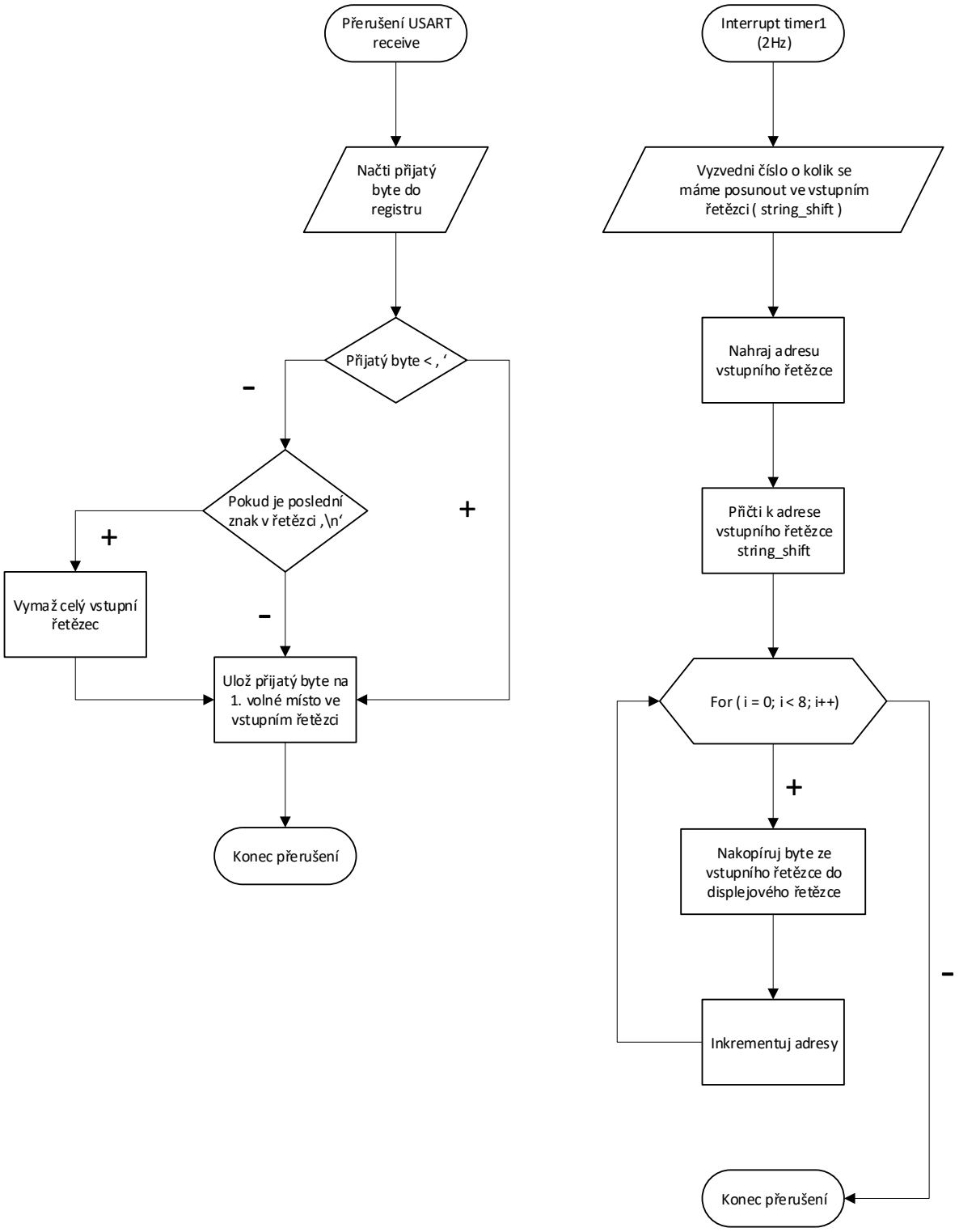
Zadání:

Zpracujte program v programovacím jazyce JSA ATMega128 a C# ovládající sériovou komunikaci mezi jedním přípravkem MB-ATmega128 a PC tak, aby obsahoval nejméně tyto
funkce:

1) stisknuté tlačítko klávesnice počítače se sériovou linkou přenese do přípravku MB-ATmega128,
1. kde se zobrazí. na modulu multiplexovaných 8 LED displejů.
2) zvolte vhodný komunikační protokol
3) komunikační program v přípravku MB-ATmega128 s modulem multiplexovaných 8LED
2. displejů by měl (volitelně) mít funkci „autonegotiation“
4) využití všech vhodných HW možností přípravku MB-ATmega128.
5) sledování chybových stavů**.**

Vývojový diagram:




Přerušení USART receive
Načti přijatý byte do registru
Přijatý byte < , ‘
-
+
Pokud je poslední znak v řetězci ,\n‘
+
-
Vymaž celý vstupní řetězec
Ulož přijatý byte na 1. volné místo ve vstupním řetězci
Konec přerušení
Interrupt timer1 (2Hz)
Vyzvedni číslo o kolik se máme posunout ve vstupním řetězci (string_shift)
Nahraj adresu vstupního řetězce
Přičti k adrese vstupního řetězce string_shift
For (i = 0; i < 8; i++)
+
Nakopíruj byte ze vstupního řetězce do displejového řetězce
Inkrementuj adresy
-
Konec přerušení

Výpis programu:

Příloha 1 – assembler

Příloha 2 – C#

Závěr:

Řešení úlohy funguje dle očekávání. Pro konverzi ASCII znaku do bitové kombinaci pro displej byla využita vyhledávací tabulka. Pro zjednodušení práce jsem vyhledávací tabulku vzal z volně dostupného repozitáře na GitHubu a upravil ji do podoby „.DB tabulky“.

Přílohy:

- Příloha 1 – 10 stran
- Příloha 2 – 2 strany

Příloha 1

```
//displej je připojen na CON1
//USB kabel je připojen do USB-D

/**********definice**********/

.nolist
.Include "m128def.inc"
.list

//definice baud ratu
    .EQU baud_rate_2400 = 383
    .EQU baud_rate_4800 = 191
    .EQU baud_rate_9600 = 95
    .EQU baud_rate_14400 = 63
    .EQU baud_rate_19200 = 47
    .EQU baud_rate_28800 = 31
    .EQU baud_rate_38400 = 23
    .EQU baud_rate_57600 = 15
    .EQU baud_rate_76800 = 11
    .EQU baud_rate_115200 = 7
    .EQU baud_rate_230400 = 3
    .EQU baud_rate_250000 = 1
    .EQU baud_rate_1000000 = 0

//definice prescaleru
    .EQU timer0_prescaler_noclock = 0
    .EQU timer0_prescaler_1 = 1
    .EQU timer0_prescaler_8 = 2
    .EQU timer0_prescaler_32 = 3
    .EQU timer0_prescaler_64 = 4
    .EQU timer0_prescaler_128 = 5
    .EQU timer0_prescaler_256 = 6
    .EQU timer0_prescaler_1024 = 7

//startovací sekvence
    .org 0x0000
    rjmp start

//timer "posouvání textu"
    .org 0x0018
    rjmp timer1a_ctc

//multiplexování displeje
```

```
    .org 0x001E
    rjmp timer0_ctc

//přijmání z USART
    .org 0x0024
    rjmp RXC_interrupt

//nevyužito
    .org 0x0026
    rjmp UDRE_interrupt

//nevyužito
    .org 0x0028
    rjmp TXC_interrupt

/**********konec definic**********/

/**********Definování datových struktur**********/
//konstanty
.CSEG
//bitové kombinaci "černého" displeje
positionToByte:
    .DB 3, 2, 1, 0, 4, 5, 6, 7

/*//bitové kombinaci "bílého" displeje
pozice:
    .DB 0, 1, 2, 3, 4, 5, 6, 7*/

home:
    .DB "        1. ULOHA - VIT PETRIK       ", '\n'

errUnknownChar:
    .DB "        Err - invalid char       ", '\n'

;lookup table pro konverzi charu na bitovou kombinaci pro displej
charToByte:
    .DB 0b00000000, 0b01100001, 0b01000100, 0b01111110
    .DB 0b10110110, 0b01001011, 0b01100010, 0b00000100
    .DB 0b10010100, 0b11010000, 0b10000100, 0b00001110
    .DB 0b00001000, 0b00000010, 0b00000001, 0b01001010
    .DB 0b11111100, 0b01100000, 0b11011010, 0b11110010
    .DB 0b01100110, 0b10110110, 0b10111110, 0b11100000
    .DB 0b11111110, 0b11110110, 0b10010000, 0b10110000
    .DB 0b10000110, 0b00010010, 0b11000010, 0b11001011
    .DB 0b11111010, 0b11101110, 0b00111110, 0b10011100
    .DB 0b01111010, 0b10011110, 0b10001110, 0b10111100
```

```
    .DB 0b01101110, 0b00001100, 0b01111000, 0b10101110
    .DB 0b00011100, 0b10101000, 0b11101100, 0b11111100
    .DB 0b11001110, 0b11010110, 0b11001100, 0b10110110
    .DB 0b00011110, 0b01111100, 0b01111100, 0b01010100
    .DB 0b01101110, 0b01110110, 0b11011010, 0b10011100
    .DB 0b00100110, 0b11110000, 0b11000100, 0b00010000
    .DB 0b01000000, 0b11111010, 0b00111110, 0b00011010
    .DB 0b01111010, 0b11011110, 0b10001110, 0b11110110
    .DB 0b00101110, 0b00001000, 0b00110000, 0b10101110
    .DB 0b00001100, 0b00101000, 0b00101010, 0b00111010
    .DB 0b11001110, 0b11100110, 0b00001010, 0b10110110
    .DB 0b00011110, 0b00111000, 0b00111000, 0b00101000
    .DB 0b01101110, 0b01110110, 0b11011010, 0b01100010
    .DB 0b00001100, 0b00001110, 0b10000000, 0b00000000

//měnitelné přoměnné
.DSEG
//řetězec, který (snad) budeme zobrazovat na displeji
displayString:
    .BYTE 8
index_displayString:
    .BYTE 1

//řetězec, do kterého se bude zapisovat vstup z UART
inputString:
    .BYTE 256
index_inputString:
    .BYTE 1

//posouvání textu
stringShift:
    .BYTE 1

/**********Konec Definování datových struktur**********/


/**********Startovací sekvence**********/

.CSEG
start:
// nastavení zásobníku
    ldi r16, low(RAMEND)
    out spl, R16
    ldi r16, high(RAMEND)
    out sph, R16
```

```
// překopírování "úvodního" řetězce
    ldi XL, low(inputString*2)
    ldi XH, high(inputString*2)
    ldi ZL, low(home*2)
    ldi ZH, high(home*2)
    call copyStringToDisplayString

// nastavení portů
    ldi r16, 0xff
    out ddrb, r16
    ldi r16, 0b0000_1111
    out ddrd, r16

// inicializace USART
    ldi r17, high(baud_rate_9600)
    ldi r16, low(baud_rate_9600)
    call USART_Init

// inicializace timeru0 na 400Hz
    ldi r16, (timer0_prescaler_256<<CS00)
    ldi r17, 144
    call timer0_Init

// inicalizace timer1 na 2hz
    call timer1_Init

// zapnutí interruptů
    sei

/* aby program nedělal nějaký blbosti tak ho zacyklíme, protože všechno řešíme
 přes interrupty
prostě černá díra -> nikdy to nevyleze pryč */
cerna_dira:
    rjmp cerna_dira

/**********Konec startovací sekvence**********/

/**********Inicializace USART**********/

// r17 - vrchní byte baud rate, r16 - spodní byte baud rate
// rámec - 1 start bit, 8 data bitů, 1 stop bit, žádný parity check
USART_Init:
    sts UBRR0H, r17
    out UBRR0L, r16
    ldi r16, (1<<RXCIE0)|(0<<TXCIE0)|(0<<UDRIE0)|(1<<RXEN0)|(1<<TXEN0)
    out UCSR0B,r16
```

```
    ldi r16, (1<<UCSZ01)|(1<<UCSZ00)
    sts UCSR0C,r16
ret

/**********Konec inicializace USART**********/

/**********Inicializace Timer0**********/

//r16 - precaler, r17 - OCR
timer0_Init:
    ldi r18, (1<<WGM01)
    or r18, r16
    out TCCR0, r18
    out OCR0, r17
    in r17, TIMSK
    ldi r16, (1<<OCIE0)
    or r16, r17
    out TIMSK, r16
ret

/**********Konec inicializace Timer0**********/

/**********Inicializace Timer1**********/

timer1_Init:
    ldi r16, high(7200)
    out OCR1AH, r16
    ldi r16, low(7200)
    out OCR1AL, r16
    ldi r16, (1<<WGM12)|(5<<CS10)
    out TCCR1B, r16
    in r17, TIMSK
    ldi r16, (1<<OCIE1A)
    or r16, r17
    out TIMSK, r16
ret

/**********Konec inicializace Timer1**********/

/**********Posouvání textu**********/

timer1a_ctc:
/*vezmeme posunovací index, přičteme ho k adrese vstupního
řetězce a nakopírujeme následujících 8 bytů ze vstupního řetězce
do zobrazovacího řetězce*/
    ldi ZL, low(stringShift*2)
```

```
    ldi ZH, high(stringShift*2)
    ld r16, Z
    mov r17, r16
    inc r17
    st Z, r17
    clr r18
    ldi ZL, low(inputString*2)
    ldi ZH, high(inputString*2)
    add ZL, r16
    adc ZH, r18
    ldi XL, low(displayString*2)
    ldi XH, high(displayString*2)
//začínáme kopírovat
copy:
    ld r17, Z+
    st X+, r17
//ošéfujeme, zda je znak platný
    cpi r17, ' '
    brlo default3
    inc r18
    cpi r18, 8
    brlt copy
//pokud jsme už nakopírovali dostatek znaků tak ukončíme přerušení
    reti
default3:
    ldi ZL, low(stringShift*2)
    ldi ZH, high(stringShift*2)
    ldi r17, 0
    st Z, r17
    cpi r16, 0
    brne timer1a_ctc
reti

/**********Konec posunování textu**********/

/**********Zobrazování na displeji**********/

timer0_ctc:
//Načtení aktuální pozice, její inkrementace a zpět uložení
    ldi ZL, low(index_displayString*2)
    ldi ZH, high(index_displayString*2)
    ld r16, Z
    mov r17, r16
    inc r17
//Ošéfování přetečení
    cpi r17, 8
```

```
    brlt default1
    ldi r17, 0
default1:
    st Z, r17
/*index nám furt zůstává r16
načteme bitovou kombinaci pro pozici
a pošleme ji na port*/
    ldi r17, 0
    out portB, r17
    ldi ZL, low(positionToByte*2)
    ldi ZH, high(positionToByte*2)
    add ZL, r16
    adc ZH, r17
    lpm r17, Z
    out portD, r17
    clr r17
//načteme char z proměnné displayString
    ldi ZL, low(displayString*2)
    ldi ZH, high(displayString*2)
    add ZL, r16
    adc ZH, r17
    ld r16, Z
//zkontrolujeme, zda se jedná o platný znak
    cpi r16, ' '
    brsh default2
        ldi ZL, low(index_displayString*2)
        ldi ZH, high(index_displayString*2)
        ldi r16, 0
        st Z, r16
        rjmp timer0_ctc
/*překonvertíme char na bitovou kombinaci pro displej
konverzi děláme pomocí lookup tabulky charToByte*/
default2:
    call charToByteConversion
    out portB, r16
//hodíme to na port
reti

/**********Konec zobrazování na displeji**********/

/**********Příjem dat**********/

RXC_interrupt:
//načteme přijatý byte a načteme adresy řetězců
    in r16, UDR0
    clr r18
```

```
    ldi XL, low(index_inputString*2)
    ldi XH, high(index_inputString*2)
    ld r17, X
    ldi ZL, low(inputString*2)
    ldi ZH, high(inputString*2)
    add ZL, r17
    adc ZH, r18
    ld r18, Z
    cpi r16, ' '
    brlo false1
//pokud je přijatý byte větší nebo roven ' '
        cpi r18, '\n'
        brne end2
/*pokud je poslední znak ve vstupním řetězci '\n'
tak vymažeme vstupní řetězec a vynulujeme indexy*/
            ldi r17, 0
            call eraseString
            ldi ZL, low(inputString*2)
            ldi ZH, high(inputString*2)
end2:
//Nahrajeme přijatý byte do vstupního řetězce a inkrementujeme index
        cpi r16, 127
        brsh end1
        st Z, r16
        inc r17
        st X, r17
//opustíme přerušení
        rjmp endOfRCXinterrupt
//pokud je přijatý byte menší jak ' '
false1:
    cpi r16, '\n'
    brne end1
//pokud je znak '\n' tak ho nahrajeme do vstupního řetězce a opustíme přerušen
í
        st Z, r16
        ldi XL, low(inputString*2)
        ldi XH, high(inputString*2)
        rjmp endOfRCXinterrupt
//pokud je znak pro nás neznámý, vypíšeme errorovou hlášku
end1:
        call eraseString
        clr r17
        st X, r17
        ldi XL, low(inputString*2)
        ldi XH, high(inputString*2)
        ldi ZL, low(errUnknownChar*2)
```

```
        ldi ZH, high(errUnknownChar*2)
        clr r19
        ldi r18, 20
        call copyStringToDisplayString
        inc r21
        st X, r21
        ldi ZL, low(inputString*2)
        ldi ZH, high(inputString*2)
        add ZL, r21
        adc ZH, r19
        ldi r16, ' '
        st Z, r16
endOfRCXinterrupt:
    ldi ZL, low(stringShift*2)
    ldi ZH, high(stringShift*2)
    ldi r17, 0
    st Z, r17
    call timer1a_ctc
reti

/**********Konec příjmu dat**********/

UDRE_interrupt:
reti

TXC_interrupt:
reti

//převede char na bitovou kombinaci pro displej
charToByteConversion:
    subi r16, ' '
    ldi ZL, low(charToByte*2)
    ldi ZH, high(charToByte*2)
    add ZL, r16
    adc ZH, r17
    lpm r16, Z
ret

//vymaže celý vstupní řetězec
eraseString:
    ldi ZL, low(inputString*2)
    ldi ZH, high(inputString*2)
    ldi R20, 0
    clr r19
return1:
    st Z+, r20
```

```
    inc r19
    breq exit1
    rjmp return1
exit1:
ret

/*X - adresa displayový řetězec
Z - adresa řetězce, ze kterého kopírujeme*/
copyStringToDisplayString:
clr r21
return:
    inc r21
    lpm r20, Z+
    st X+, r20
    cpi r20, '\n'
    brne return
    ldi XL, low(index_inputString*2)
    ldi XH, high(index_inputString*2)
    dec r21
    st X, r21
ret
```

Příloha 1

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.IO.Ports;


namespace serialPortController
{
    /// <summary>
    /// Interakční logika pro MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        private SerialPort port;
        public MainWindow()
        {
            InitializeComponent();
            updateCOMlist();
        }

        private void updateCOMlist()
        {
            string[] ports = SerialPort.GetPortNames();

            // Display each port name to the console.
            foreach (string port in ports)
            {
                selectCOMport.Items.Add(port);
                Console.WriteLine(port);
            }
        }
```

```
        private void ConnectToDevice_Click(object sender, RoutedEventArgs e)
        {
            port = new SerialPort(selectCOMport.Text, 9600, Parity.None, 8, St
opBits.One);
            port.Open();
        }

        private void SnadToDevice_Click(object sender, RoutedEventArgs e)
        {
            port.WriteLine(stringTextbox.Text);
        }
    }
}
```