



Díleňská praxe

A4	7. Model křížovanky		
Petrík Vít		1/18	Známka:
11. 3. 2020	Datum odevzdání:	1. 4. 2020	Odevzdáno:



Zadání:

Zpracujte program v programovacím jazyce C# ovládající model křižovatky tak, aby obsahoval nejméně tyto funkce:

- 1) funkce řízení světél křižovatky respektuje pravidla silničního provozu.
- 2) pomocí tlačítek modelu přepínejte režim denní/noční (volitelně plná/zjednodušená křižovatka).
- 3) na monitoru počítače zobrazujte aktuální stav světél křižovatky, případně režimu činnosti křižovatky.

Postup:

- Nadefinování datových struktur.
- Tvorba komponentů pro simulaci křižovatky.
- Pospojování datových struktur programu.
- Otestování základní funkčnosti programu.
- Zadefinování konfigurací křižovatky dle dopravních předpisů.
- Implementace nočního režimu.
- Navržení spojení desek Velleman a modelu křižovatky.
- Import DLL knihovny Zápis ovládání modelu křižovatky dle dokumentace.

Propojení křižovatky a desek Velleman:

1. deska		2. deska	
Pin	Význam	Pin	Význam
0	Č. dopředu	0	Č. dopředu
1	Ž. dopředu	1	Ž. dopředu
2	Z. dopředu	2	Z. dopředu
3	Č. doleva	3	Z. doprava
4	Ž. doleva	4	Č. přechod
5	Z. doleva	5	Z. přechod
6	Č. přechod	6	Tlačítko
7	Z. přechod	7	Tlačítko



Schéma zapojení:

LABORATOŘE MIKROPROCESOROVÉ TECHNIKY

název úlohy: Světelná křižovatka

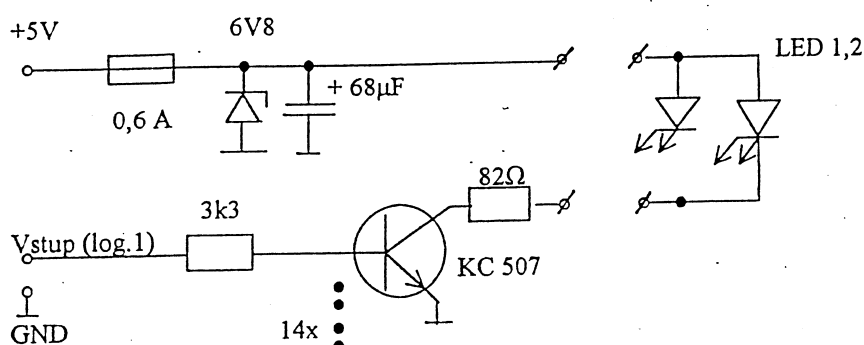
Technické údaje: Napájení : +5V/ max. 0,8 A

: Ovládání : všechny vstupy a výstupy v úrovních TTL

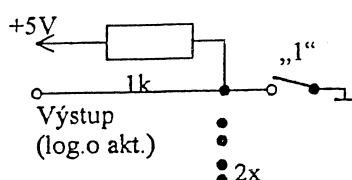
14 x vstup, světlo svítí při log. 1 na vstupu

2 x výstup, po stisku funkčního tlačítka „1“ nebo „2“ je na příslušném výstupu log. 0

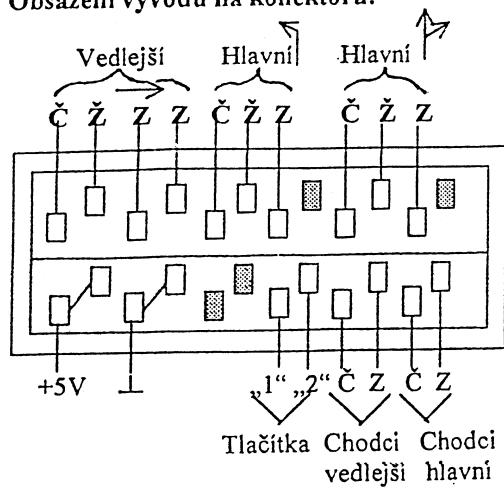
Vnitřní zapojení – jednoho vstupu



- jednoho výstupu



Obsazení vývodu na konektoru:

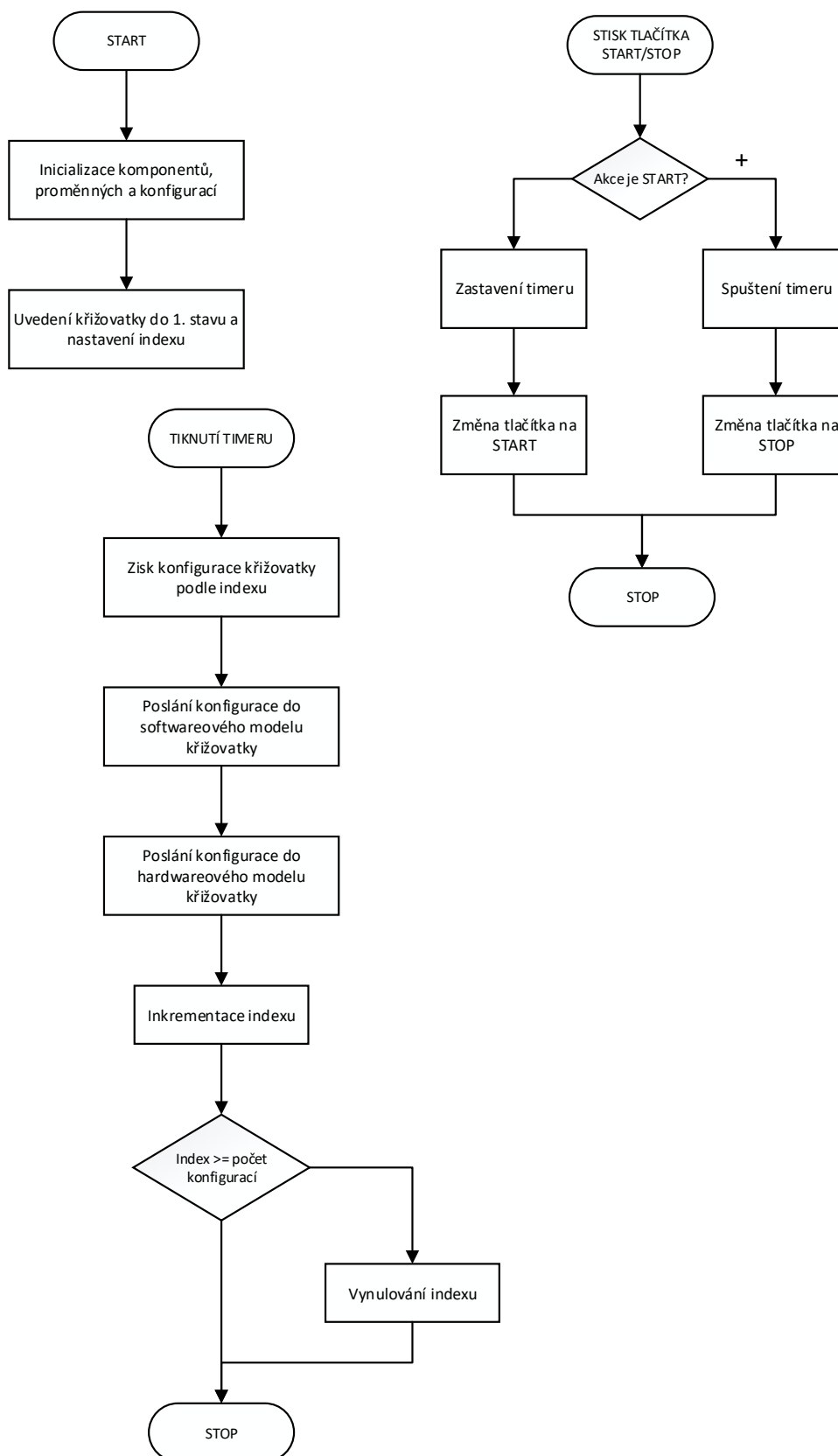


Tlačítko stisknuto → log. 0

Světlo rozsvítit → log. 1



Vývojový diagram:





Výpis programu:

- Příloha 1 – Form1.cs
- Příloha 2 – IntersectionConfigurations.cs
- Příloha 3 – IntersectionHW.cs
- Příloha 4 – signalLight.cs
- Příloha 5 – semafor-Main.cs
- Příloha 6 – IntersectionSV.cs

Okomentován je jenom kód v příloze 1. Další soubory dle mého komentovat není třeba.

Závěr:

Program funguje tak jak má. Pro zhotovení softwareového modelu křižovatky jsem využil tvorbu vlastních UI komponentů. Komponenta křižovatky se tedy skládá z obrázu a semaforů. Každý semafor je komponenta, která je složena z komponent samostatných signálních světel.

Vzhledem k situaci, za které byl program napsán nedokáži potvrdit ani vyvrátit, zda je ovládání skutečné křižovatky funkční, či nikoliv.

Předávání konfigurace křižovatky jsem vyřešil třídou, která obsahuje všechny důležité informace o stavu semaforů. Jednotlivé konfigurace jsou zapsány do listu.

Vzhledem k tomu, že jsem C# nepracoval delší dobu, chvíli mi trvalo než jsem se opět rozkoukal v prostředí Visual Studia. Troufám si tvrdit, že samotný kód je nejlepší, který jsem kdy v C# napsal. Samozřejmě něco by stále šlo vylepšit, ale to už je mimo mé programátorské schopnosti.

Protože mám zkušenosti s vývojem webových aplikací pomocí front-endových javascriptových frameworků, velice jsem si oblíbil možnost tvorby vlastních uživatelských komponent, která je mi sobě vlastní. Je to super způsob jak decentralizovat řídicí logiku uživatelského prostředí, zvláště v případě mnohonásobného použití stejného prvku.

Přílohy:

- Příloha 1 – 3 strany
- Příloha 2 – 1 strana
- Příloha 3 – 4 strany
- Příloha 4 – 2 strany
- Příloha 5 - 2 strany
- Příloha 6 – 1 strana



Příloha 1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Library;

namespace winFormUI
{
    public partial class Form1 : Form
    {
        //definice proměnných
        private static List<List<IntersectionConfiguration>> configurations;
        private static IntersectionHW IntersectionHW = new IntersectionHW();

        private static int Index { get; set; }
        private static Boolean Run { get; set; }

        public Form1()
        {
            //inicializace
            InitializeComponent();
            initializeConfigurations();

            buttonCheckTimer.Stop();

            //nastavení listboxu
            rezimListBox.Items.Add("Denní režim");
            rezimListBox.Items.Add("Noční režim");
            rezimListBox.SelectedIndex = 0;

            //nastavení dalších komponent
            setButtonToStart();
            nextConfigurationTimer.Stop();
            Run = false;

            //výchozí konfigurace
            loadConfiguration(0, rezimListBox.SelectedIndex);
            Index = 1;
        }

        //inicializace konfigurací
        private static void initializeConfigurations()
        {
            ...spousta konstantních definic zabírající hrozně moc řádků...
        }

        //reakce na tiknutí timeru
        private void nextConfigurationTimer_Tick(object sender, EventArgs e)
        {
            loadConfiguration(Index, rezimListBox.SelectedIndex); //zisk konfigurace
            Index++;
            //vynulování indexu, pokud index přetekl velikost listu
            if(Index >= configurations[rezimListBox.SelectedIndex].Count) {
                Index = 0;
            }
        }
    }
}
```



```
//změna denní/noční
private void rezimListBox_SelectedIndexChanged(object sender, EventArgs e)
{
    loadConfiguration(0, rezimListBox.SelectedIndex);
    Index = 1;
}

//nahrání aktuální konfigurace do křižovatky
private void loadConfiguration(int i, int mode)
{
    IntersectionConfiguration config = configurations[mode][i];
    nextConfigurationTimer.Interval = config.Duration*1000; //nastavení timeru

    intersectionSW.Configuration = config;
    IntersectionHW.Configuration = config;
}

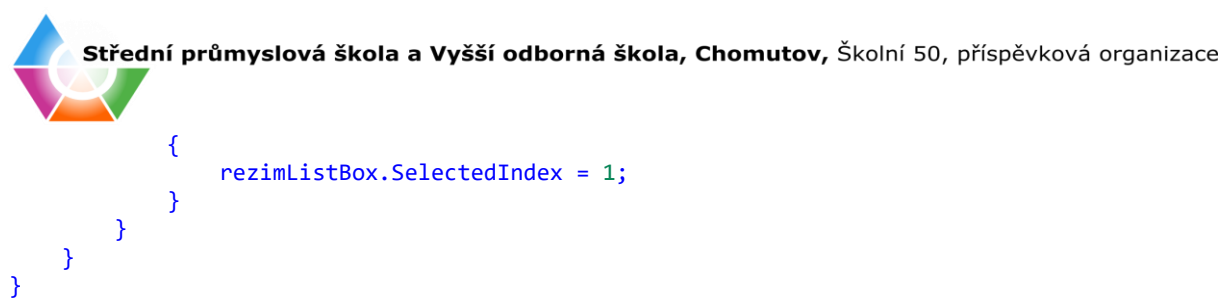
//tlačítko START/STOP
private void stopButton_Click(object sender, EventArgs e)
{
    if(Run)
    {
        //zastavení běhu
        nextConfigurationTimer.Stop();
        Run = false;
        setButtonToStart();
    } else
    {
        //spuštění běhu
        nextConfigurationTimer.Start();
        Run = true;
        setButtonToStop();
    }
}


private void setButtonToStop()
{
    stopButton.BackColor = Color.FromName("ControlDarkDark");
    stopButton.Text = "Stop";
}

private void setButtonToStart()
{
    stopButton.BackColor = Color.FromName("Red");
    stopButton.Text = "Start";
}


//terminace komunikace s deskami při zavření programu
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    IntersectionHW.CloseConnection();
}

//kontrola tlačítek na modelu křižovatky
private void buttonCheckTimer_Tick(object sender, EventArgs e)
{
    Boolean[] result = IntersectionHW.ButtonsPress();
    if(result[0] && !result[1])
    {
        rezimListBox.SelectedIndex = 0;
    } else if (!result[0] && result[1])
```




 **Střední průmyslová škola a Vyšší odborná škola, Chomutov, Školní 50, příspěvková organizace**

```
{  
    rezimListBox.SelectedIndex = 1;  
}  
}  
}
```



Střední průmyslová škola a Vyšší odborná škola, Chomutov, Školní 50, příspěvková organizace

```
{  
    rezimListBox.SelectedIndex = 1;  
}  
}  
}
```



Střední průmyslová škola a Vyšší odborná škola, Chomutov, Školní 50, příspěvková organizace

```

{
    rezimListBox.SelectedIndex = 1;
}
}
}
}

```




Příloha 2

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Library
{
    public class IntersectionConfiguration
    {
        public MainRoad TopRoad { get; set; }
        public MainRoad BottomRoad { get; set; }
        public MinorRoad LeftRoad { get; set; }
        public MinorRoad RightRoad { get; set; }
        public int Duration { get; set; }
    }

    public class MainRoad
    {
        public int ForwardRight { get; set; }
        public int Left { get; set; }
        public int CrossWalk { get; set; }
    }

    public class MinorRoad
    {
        public int Main { get; set; }
        public int Right { get; set; }
        public int CrossWalk { get; set; }
    }
}
```



Příloha 3

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;

namespace Library
{
    public class IntersectionHW
    {
        private IntersectionConfiguration _configuration;

        public IntersectionConfiguration Configuration
        {
            get
            {
                return _configuration;
            }

            set
            {
                _configuration = value;
                update();
            }
        }

        private void update()
        {
            OpenDevice(0);
            switch (Configuration.TopRoad.ForwardRight)
            {
                case 0:
                    ClearDigitalChannel(0);
                    ClearDigitalChannel(1);
                    ClearDigitalChannel(2);
                    break;
                case 1:
                    SetDigitalChannel(0);
                    ClearDigitalChannel(1);
                    ClearDigitalChannel(2);
                    break;
                case 2:
                    SetDigitalChannel(0);
                    SetDigitalChannel(1);
                    ClearDigitalChannel(2);
                    break;
                case 3:
                    ClearDigitalChannel(0);
                    ClearDigitalChannel(1);
                    SetDigitalChannel(2);
                    break;
                case 4:
                    ClearDigitalChannel(0);
                    SetDigitalChannel(1);
                    ClearDigitalChannel(2);
                    break;
            }

            switch (Configuration.TopRoad.Left)
            {
                case 0:
```



```
        ClearDigitalChannel(3);
        ClearDigitalChannel(4);
        ClearDigitalChannel(5);
        break;
    case 1:
        SetDigitalChannel(3);
        ClearDigitalChannel(4);
        ClearDigitalChannel(5);
        break;
    case 2:
        SetDigitalChannel(3);
        SetDigitalChannel(4);
        ClearDigitalChannel(5);
        break;
    case 3:
        ClearDigitalChannel(3);
        ClearDigitalChannel(4);
        SetDigitalChannel(5);
        break;
    case 4:
        ClearDigitalChannel(3);
        SetDigitalChannel(4);
        ClearDigitalChannel(5);
        break;
}

switch (Configuration.TopRoad.CrossWalk)
{
    case 0:
        ClearDigitalChannel(6);
        ClearDigitalChannel(7);
        break;
    case 1:
        SetDigitalChannel(6);
        ClearDigitalChannel(7);
        break;
    case 2:
        ClearDigitalChannel(6);
        SetDigitalChannel(7);
        break;
}

OpenDevice(1);

switch (Configuration.LeftRoad.Main)
{
    case 0:
        ClearDigitalChannel(0);
        ClearDigitalChannel(1);
        ClearDigitalChannel(2);
        break;
    case 1:
        SetDigitalChannel(0);
        ClearDigitalChannel(1);
        ClearDigitalChannel(2);
        break;
    case 2:
        SetDigitalChannel(0);
        SetDigitalChannel(1);
        ClearDigitalChannel(2);
        break;
    case 3:
```



```
        ClearDigitalChannel(0);
        ClearDigitalChannel(1);
        SetDigitalChannel(2);
        break;
    case 4:
        ClearDigitalChannel(0);
        SetDigitalChannel(1);
        ClearDigitalChannel(2);
        break;
}

WriteDigitalChannel(3, Configuration.LeftRoad.Right == 1);

switch (Configuration.TopRoad.CrossWalk)
{
    case 0:
        ClearDigitalChannel(4);
        ClearDigitalChannel(5);
        break;
    case 1:
        SetDigitalChannel(4);
        ClearDigitalChannel(5);
        break;
    case 2:
        ClearDigitalChannel(4);
        SetDigitalChannel(5);
        break;
}
}

public Boolean[] ButtonsPress()
{
    OpenDevice(1);
    Boolean[] array = new Boolean[2];
    array[0] = ReadDigitalChannel(6);
    array[1] = ReadDigitalChannel(7);
    return array;
}

private void WriteDigitalChannel(int channel, Boolean value)
{
    if(value)
    {
        SetDigitalChannel(channel);
    }
    else
    {
        ClearDigitalChannel(channel);
    }
}

public void CloseConnection()
{
    CloseDevice();
}

[DllImport("k8055d.dll")]
private static extern int OpenDevice(int CardAddress);
[DllImport("k8055d.dll")]
private static extern void CloseDevice();
[DllImport("k8055d.dll")]
private static extern int ReadAnalogChannel(int Channel);
```



```
[DllImport("k8055d.dll")]
private static extern void ReadAllAnalog(ref int Data1, ref int Data2);
[DllImport("k8055d.dll")]
private static extern void OutputAnalogChannel(int Channel, int Data);
[DllImport("k8055d.dll")]
private static extern void OutputAllAnalog(int Data1, int Data2);
[DllImport("k8055d.dll")]
private static extern void ClearAnalogChannel(int Channel);
[DllImport("k8055d.dll")]
private static extern void SetAllAnalog();
[DllImport("k8055d.dll")]
private static extern void ClearAllAnalog();
[DllImport("k8055d.dll")]
private static extern void SetAnalogChannel(int Channel);
[DllImport("k8055d.dll")]
private static extern void WriteAllDigital(int Data);
[DllImport("k8055d.dll")]
private static extern void ClearDigitalChannel(int Channel);
[DllImport("k8055d.dll")]
private static extern void ClearAllDigital();
[DllImport("k8055d.dll")]
private static extern void SetDigitalChannel(int Channel);
[DllImport("k8055d.dll")]
private static extern void SetAllDigital();
[DllImport("k8055d.dll")]
private static extern bool ReadDigitalChannel(int Channel);
[DllImport("k8055d.dll")]
private static extern int ReadAllDigital();
[DllImport("k8055d.dll")]
private static extern int ReadCounter(int CounterNr);
[DllImport("k8055d.dll")]
private static extern void ResetCounter(int CounterNr);
[DllImport("k8055d.dll")]
private static extern void SetCounterDebounceTime(int CounterNr, int DebounceTime);
[DllImport("k8055d.dll")]
private static extern int Version();
[DllImport("k8055d.dll")]
private static extern int SearchDevices();
[DllImport("k8055d.dll")]
private static extern int SetCurrentDevice(int lngCardAddress);
[DllImport("k8055d.dll")]
private static extern int ReadBackDigitalOut();
[DllImport("k8055d.dll")]
private static extern void ReadBackAnalogOut(int[] Buffer);
}
}
```



Příloha 4

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace winFormUI
{
    public partial class signallight : UserControl
    {
        private String _color;
        private Boolean _enabled;

        public String SignalColor
        {
            get
            {
                return _color;
            }
            set
            {
                _color = value;
                drawSignal(drawBox.CreateGraphics(), GetColor());
            }
        }

        public Boolean Status
        {
            get
            {
                return _enabled;
            }
            set
            {
                _enabled = value;
                drawSignal(drawBox.CreateGraphics(), GetColor());
            }
        }

        public signallight()
        {
            InitializeComponent();
        }

        private void drawSignal(Graphics g, Color c)
        {
            SolidBrush brush = new SolidBrush(c);
            g.FillEllipse(brush, 0, 0, 20, 20);
        }

        private void drawBox_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.Clear(Color.Black);
            if(SignalColor != null)
            {

```



```
        drawSignal(g, GetColor());
    }
}

private Color GetColor()
{
    if(Status)
    {
        switch(SignalColor)
        {
            case "red":
                return ColorTranslator.FromHtml("#ff0000");
            case "yellow":
                return ColorTranslator.FromHtml("#ffff00");
            case "green":
                return ColorTranslator.FromHtml("#00ff00");
        }
    }
    else
    {
        switch (SignalColor)
        {
            case "red":
                return ColorTranslator.FromHtml("#600000");
            case "yellow":
                return ColorTranslator.FromHtml("#606000");
            case "green":
                return ColorTranslator.FromHtml("#006000");
        }
    }
    return Color.Black;
}
}
```



Příloha 5

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace winFormUI
{
    public partial class semafor_Main : UserControl
    {
        private int _status;

        public int Status
        {
            get
            {
                return _status;
            }
            set
            {
                _status = value;
                update();
            }
        }

        public semafor_Main()
        {
            InitializeComponent();
        }

        private void update()
        {
            switch(Status)
            {
                case 0:
                    red.Status = false;
                    yellow.Status = false;
                    green.Status = false;
                    break;
                case 1:
                    red.Status = true;
                    yellow.Status = false;
                    green.Status = false;
                    break;
                case 2:
                    red.Status = true;
                    yellow.Status = true;
                    green.Status = false;
                    break;
                case 3:
                    red.Status = false;
                    yellow.Status = false;
                    green.Status = true;
                    break;
                case 4:
                    red.Status = false;
                    yellow.Status = true;
            }
        }
    }
}
```




```
        green.Status = false;  
        break;  
    }  
}  
}
```



Příloha 6

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Library;

namespace winFormUI
{
    public partial class intersectionSW : UserControl
    {
        private IntersectionConfiguration _configuration;

        public IntersectionConfiguration Configuration
        {
            get
            {
                return _configuration;
            }
            set
            {
                _configuration = value;
                update();
            }
        }

        public intersectionSW()
        {
            InitializeComponent();
        }

        private void update()
        {
            if(Configuration != null)
            {
                TopRoad_ForwardRight.Status = Configuration.TopRoad.ForwardRight;
                TopRoad_Left.Status = Configuration.TopRoad.Left;
                TopRoad_CrossWalk.Status = Configuration.TopRoad.CrossWalk;

                BottomRoad_ForwardRight.Status = Configuration.BottomRoad.ForwardRight;
                BottomRoad_Left.Status = Configuration.BottomRoad.Left;
                BottomRoad_CrossWalk.Status = Configuration.BottomRoad.CrossWalk;

                LeftRoad_Main.Status = Configuration.LeftRoad.Main;
                LeftRoad_Arrow.Status = Configuration.LeftRoad.Right;
                LeftRoad_CrossWalk.Status = Configuration.LeftRoad.CrossWalk;

                RightRoad_Main.Status = Configuration.RightRoad.Main;
                RightRoad_Arrow.Status = Configuration.RightRoad.Right;
                RightRoad_CrossWalk.Status = Configuration.RightRoad.CrossWalk;
            }
        }
    }
}
```