PRAXE

Datum: 27.02. 2023	Střední průmyslová škola, Chomutov, Školní 50, příspěvková organizace	Třída: V4
Číslo úlohy: 5.	Trezor	Jméno: Tomáš Bartoš

Zadání:

S využitím IO karty sestavte v jazyku C# program pro ovládání modelu mikrovlnné trouby ve funkci trezorové výstavní skříně. Dle svého uvážení zvolte buď prostředí konzolové aplikace nebo grafické prostředí.

Výsledné řešení musí obsahovat následující funkcionalitu:

- V uzavřeném stavu aktivujte osvětlení na 50% maxima
- Automatické otevření dvířek po zadání správného kódu (ve výchozím stavu 1234) s možností změny a uložení
- tohoto kódu, pokud jsou dveře otevřené. Po otevření rozsviťte žárovku na 100% pro dostatečné osvětlení expozice
- při úpravě.
- Zvukové znamení při otevření dvířek po dobu delší než 1 minuta a při zadání chybného kódu 3x za sebou
- Trvalé uložení nastaveného kódu s jeho obnovou při znovuspuštění obslužného programu

Teorie:

K úspěšnému řešení této úlohy je velmi podstatné chápat, jakým způsobem funguje na modelu trezoru/mikrovlnky ovládání displeje a čtení tlačítek, společně s obecnými informacemi o IO kartě, sloužící pro komunikaci s periferií.

Ovládání displeje stojí primárně na jeho samotné konstrukci. Jedná se totiž o 4-místný 7-segmentový display, který zobrazuje znaky uložené ve své PROM paměti znaků, jež je adresována 5 piny. Každá dostupný znak s jeho patřičnou adresou je konkrétněji popsán v dokumentačních materiálech modelu trezoru/mikrovlnky. Pro zobrazení informace na displeji je nutno nejdříve přivést na PROM adresní piny příslušnou binární kombinaci a následně aktivovat tzv. Multiplex pin segmentu, který slouží v podstatě jako CE, povolující přístup informace z PROM paměti na segment. Z důvodu této architektury je také velmi důležité zvolit vhodnou obnovovací frekvenci tak, aby bylo zobrazení informace co nejvíce uživatelsky přívětivé.

Co se čtení 4 tlačítek (MODE, UP, DOWN a SET) týče, tak ta jso zapojena tak, že je vždy jedno tlačítko připojeno k jednomu ze segmentů. Při aktivaci tohoto segmentu a stisku patřičného tlačítka je aktivován stavový signál na modelu a stisk tak může být lépe rozeznán. Kontrola stavu tohoto signálu se kontroluje po každé aktivaci segmentu.

Pro ovládání žárovky modelu je využita technika PWM neboli pulzně šířková modulace. Ta funguje na principu periodického odesílání digitálního signálu v podobě log. 1 a log. 0 v určitém poměru. Tento poměr pak definuje intenzitu produkovaného světla či rychlost otáčení motoru.

Pomocnou periferií použitou u této úlohy je IO karta, která slouží k následujícím účelům:

- Vysílání ovládacích signálů do trezoru/mikrovlnky (žárovka, aktivace segmentů, buzzer)
- Vysílání 5-bitových adres znaků z PROM paměti mikrovlnky
- Přijímání stavových signálů (stav dvířek a stisk tlačítek)

Abychom mohli s IO kartou komunikovat, je potřeba přidat do projektu referenci na dynamickou knihovnu Automation.BDaq4.dll a založit nový profil ve formátu XML v projektové složce pomocí programu Navigator. Následně musíme založit instance třídy DeviceInformation popisující, s jakým zařízením pracujeme a instance tříd InstantDoControl a InstantDiControl, které slouží k vysílání signálů z IO karty (ovládání mikrovlnky a adresace znaků) a přijímání signálů z externích zařízení (stavové signály mikrovlnky). S těmito instancemi nakonfigurujeme IO kartu následovně:

```
using Automation.BDaq;

DeviceInformation.Description = "PCI-1756,BID#0";
DeviceInformation.DeviceMode = AccessMode.ModeWrite;
InstantDoControl.SelectedDevice = …;
InstantDoControl.LoadProfile("profile.xml");
InstantDiControl.SelectedDevice = …;
InstantDiControl.LoadProfile("profile.xml");
```

Popis programu:

Ihned po spuštění programu se spouští konstruktor třídy Program, který mimo základního setupu konzolového prostředí zinicializuje i používaný globální objekt vaultModel, sloužící pro volání obslužných metod na samotném modelu trezoru.

V moment, kdy je vyvolána instance objektu VaultModel, počne interní inicializace tohoto objektu. V průběhu inicializace se nastaví instance na globální objety BitOperations (třída usnadňující práci s bitovými operacemi), PreciseDelay (třída zajišťující funkcionalitu podobnou Thread.Sleep, ale mnohem přesnější) a Board (objekt reprezentující IO kartu), společně s tvorbou timeru pro signalizaci dlouhého otevření dveří (doorTimer), timeru čtení stavu senzorů (readingTimer) a timeru obsluhující PWM na žárovce (lightTimer). V neposlední řadě se tvoří nové válko (portThread) sloužící k neustálému zapisování stavu globálních proměnných port1 a port2 na výstup IO karty, aby k němu byl zajištěn jednotný přístup.

Jakmile jsou všechny instance globálních tříd inicializovány, přechází program do nekonečné smyčky, ve které se neustále volá metoda displayCode třídy VaultModel. V této metodě je zajištěn převod informace z globální proměnné displayData, jenž pro každý segment uchovává informaci o aktuálním zobrazovaném čísle, na hexadecimální kód, jež po přivedení na PROM paměť znaků zajistí zobrazení příslušného čísla. Převod se provádní pomocí globálního slovníku map. Data jsou pak na PROM piny přivedeny pomocí přímého zápisu do proměnné (nastaví se pouze prvních 5 bitů, zbytek zůstává). Pro zobrazení na displeji je poté ještě nutné zaktivovat příslušný multiplex pin, jež vybere jeden ze 4 segmentových displejů. Z důvodu konstrukce displeje je v moment aktivního multiplex signálu nutné zkontrolovat stav tlačítka připojeného na tento segment displeje (signál KEYBOARD_PIN; metoda CheckButton), pokud je jeho stav v log.0 pokračuje se se zpracováním a je podniknuta příslušná činnost.

Činnosti vykonávané při stisku tlačítka:

- Stisk tlačítka MODE
    - Potvrzení aktuálního kódu -> kontrola správnosti (metoda ValidCode)
        - V moment správného kódu vyvolána metoda UnlockDoors, která vyšle na pin zánku rychlí signál zapnutí a vypnutí, čímž se otevřou dvířka
    - Pokud jsou otevřena dvířka změna aktuálně zobrazovaného kódu na správný otevírací kód
        - Kód ukládán pomocí klasických C# IO operací, konkrétněji jde o metodu WriteAllText
- Stisk tlačítka +:
    - Inkrementace aktuálního zobrazovaného čísla
- Stisk tlačítka -:
    - Dekrementace aktuálního zobrazovaného čísla
- Stisk tlačítka SET:
    - Přepnutí se na nadcházející číslo na displeji

Co se paralelní běžící části řešení (timerů) týče, tak po uplynutí 60s od otevření dvířek nastává přerušení doorTimeru, které jen zvukově signalizuje daný stav, nadále je každých 100ms kontrolován stav dvířek, na základě jejich stavu je lightTimer pro PWM buďto zapnut či vypnut (při vypnutí nastavena 100% intenzita).

Rozbor Proměnných a metod:

Rozbor proměnných:

BoardWiring.cs		
Datový typ	**Název**	**Popis**
int	PORT	Číslo portu.
int	PROM_0	Hodnota pro první PROM pin.
int	PROM_1	Hodnota pro druhý PROM pin.
int	PROM_2	Hodnota pro třetí PROM pin.
int	PROM_3	Hodnota pro čtvrtý PROM pin.
int	PROM_4	Hodnota pro pátý PROM pin.
int	UNLOCK_PIN	Hodnota pro pin odemknutí.
int	MOTOR_PIN	Hodnota pro pin motoru.
int	LIGHT_PIN	Hodnota pro pin světla.
int	SOUND_PIN	Hodnota pro pin zvuku.
int	MULTIPLEX_0	Hodnota pro první pin multiplexeru.
int	MULTIPLEX_1	Hodnota pro druhý pin multiplexeru.
int	MULTIPLEX_2	Hodnota pro třetí pin multiplexeru.
int	MULTIPLEX_3	Hodnota pro čtvrtý pin multiplexeru.
int	PORT	Číslo portu pro vstupy.
int	KEYBOARD_PIN	Hodnota pro pin klávesnice.
int	DOOR_PIN	Hodnota pro pin dveří.

Program.cs		
Datový typ	**Název**	**Popis**
Program	prg	Reference na aktuální program.
VautModel	vaultMode	Reference na třídu modelu trezoru.

VaultModel.cs		
Datový typ	**Název**	**Popis**
BitOperations	bitOperations	Instance třídy BitOperations
PreciseDelay	preciseDelay	Instance třídy PreciseDelay
Board	board	Instance třídy Board
Timer	doorOpenTimer	Timer pro pípání při dlouhém otevření dveří.
Timer	lightTimer	Timer pro PWM světla
Timer	readingTimer	Timer pro čtení senzorů.
Thread	portThread	Thread pro zápis dat z globálních proměnných na port.
Dictionary<int, int>	holdCounterMap	Mapa indexu tlačítka s číslem reprezentujícím dobu stisku.
Dictionary<int, char>	displayData	Mapa indexu tlačítka a zobrazovaného charakteru.
int	selectedDial	Index aktuálně editované hodnoty na displeji.
int	errorCounter	Počítadlo chyb.
bool	editorMode	Určuje, zda-li je zařízení v režimu úprav kódu.
bool	canStart	Indikuje, zda-li je možné spustit zařízení.
bool	lastLight	Poslední stav světla (zapnuto/vypnuto).
int	holdNorm	Norma pro zpoždění při držení tlačítka.
byte	port1	Stav portu 1, který se používá pro digitální výstupy.
byte	port2	Stav portu 2, který se používá pro digitální výstupy.
Dictionary<char, byte>	map	Slovník mapující znaky na jejich binární hodnotu.

Board.cs		
Datový typ	**Název**	**Popis**
DeviceInformation	dev	Informace o zařízení, které se používá pro komunikaci s hardwarovými porty.
InstantDoCtrl	DO	Instance třídy InstantDoCtrl, která slouží k ovládání digitálních výstupů.
InstantDiCtrl	DI	Instance třídy InstantDiCtrl, která slouží k čtení digitálních vstupů.
BitOperations	bitOperations	Instance třídy BitOperations, která poskytuje operace s bity.

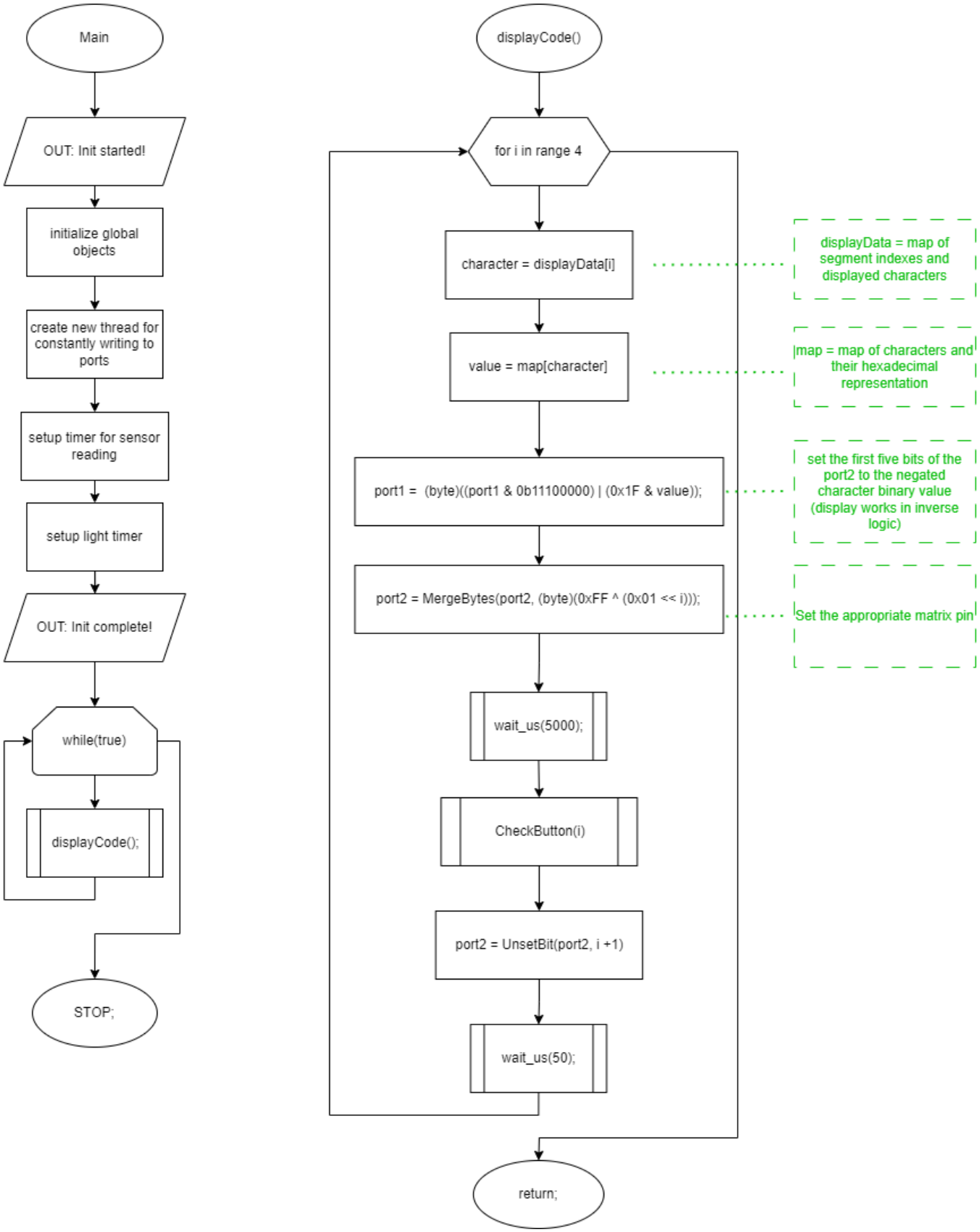
Rozbor metod:

program.cs			
Návrato	**Název**	**Parametry**	**Popis**
void	Initialize		Metoda pro inicializaci.
void	Main	string[] args	Hlavní metoda programu.
void	CurrentDomain_ProcessExit	object sender, EventArgs e	Obsluha události pro ukončení aplikace.

Board.cs			
Návratový typ	**Název**	**Parametry**	**Popis**
void	InitObjects		Inicializuje objekty související s zařízením.
void	PortSetup		Inicializuje výstupní porty na základě nastavení.
void	SetByte	int port, byte data	Nastaví byte na daný port.
void	SetBit	int port, int pinAddr	Nastaví bit na zadaném portu na log. 1.
bool	GetBit	int port, int bitAddress	Vrátí aktuální stav bitu na daném portu.
byte	GetPortState	int port	Vrátí aktuální stav portu.
void	ShowError	string error	Zobrazí chybové hlášení.

VaultModel.cs			
Návratový typ	**Název**	**Parametry**	**Popis**
void	CheckSensors	object? sender, ElapsedEventArgs e	Metoda pro periodickou kontrolu stavu senzoru dveří.
void	DisplayCode		Metoda pro zobrazení kódu.
void	ResetPorts		Resetuje stavy portů na výchozí hodnoty.
void	SwitchLight	object? sender, ElapsedEventArgs e	Metoda pro přepínání světla.
void	Beep	int interval	Metoda pro vygenerování zvukového signálu.
bool	validCode		Metoda pro ověření platnosti kódu.
void	unlockDoors		Metoda pro odemčení dveří.
void	PortThreadInterrupt		Metoda pro pravidelné zápis stavů portů do hardwarových portů.
void	CheckButtton	int btnIndex	Metoda pro kontrolu stisknutí tlačítek.
long	DelayMicroseconds	long microseconds	Metoda pro zpoždění v mikrosekundách.
VaultModel	Konstruktor		Konstruktor třídy.

Vývojový diagram:



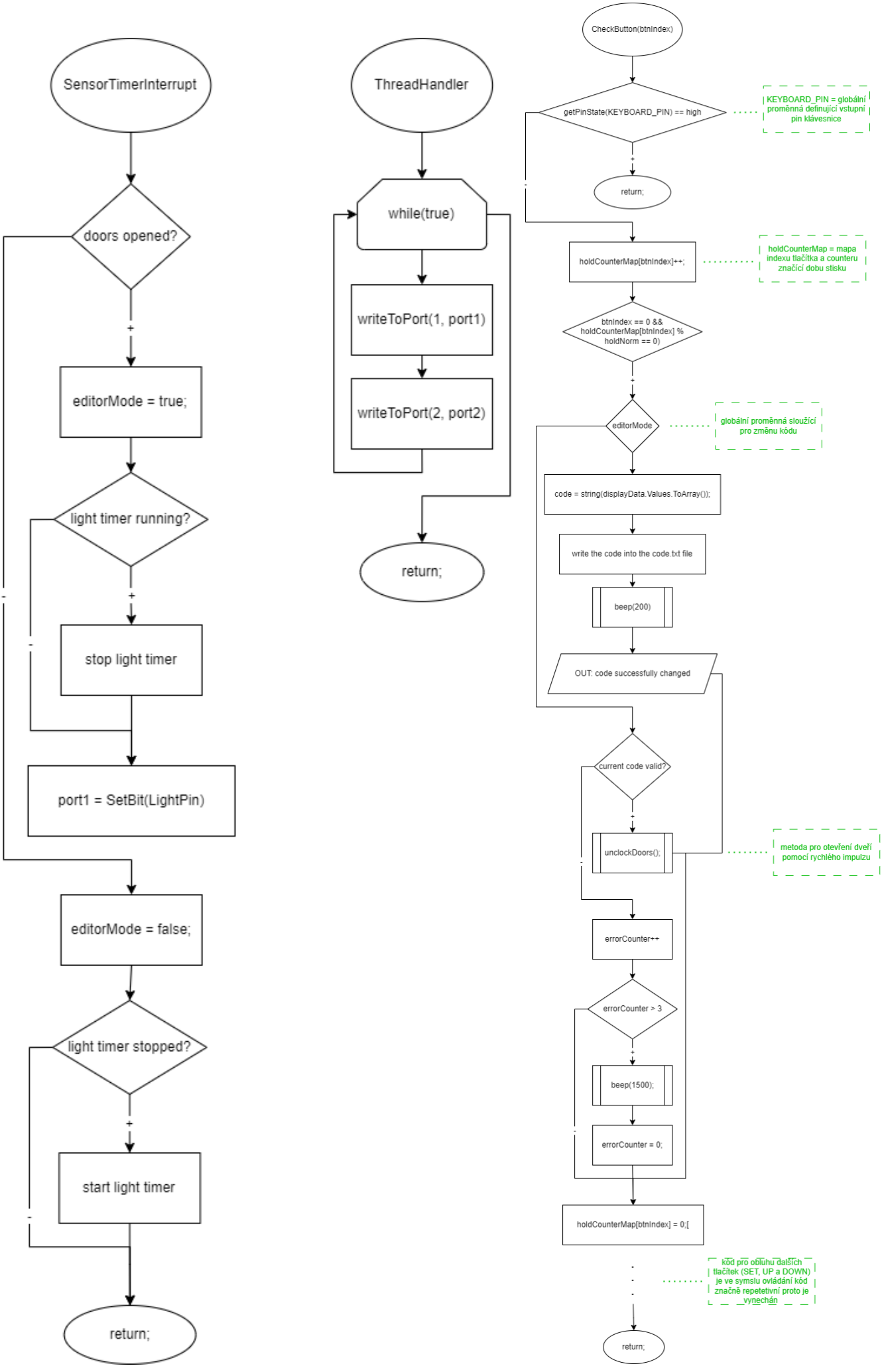
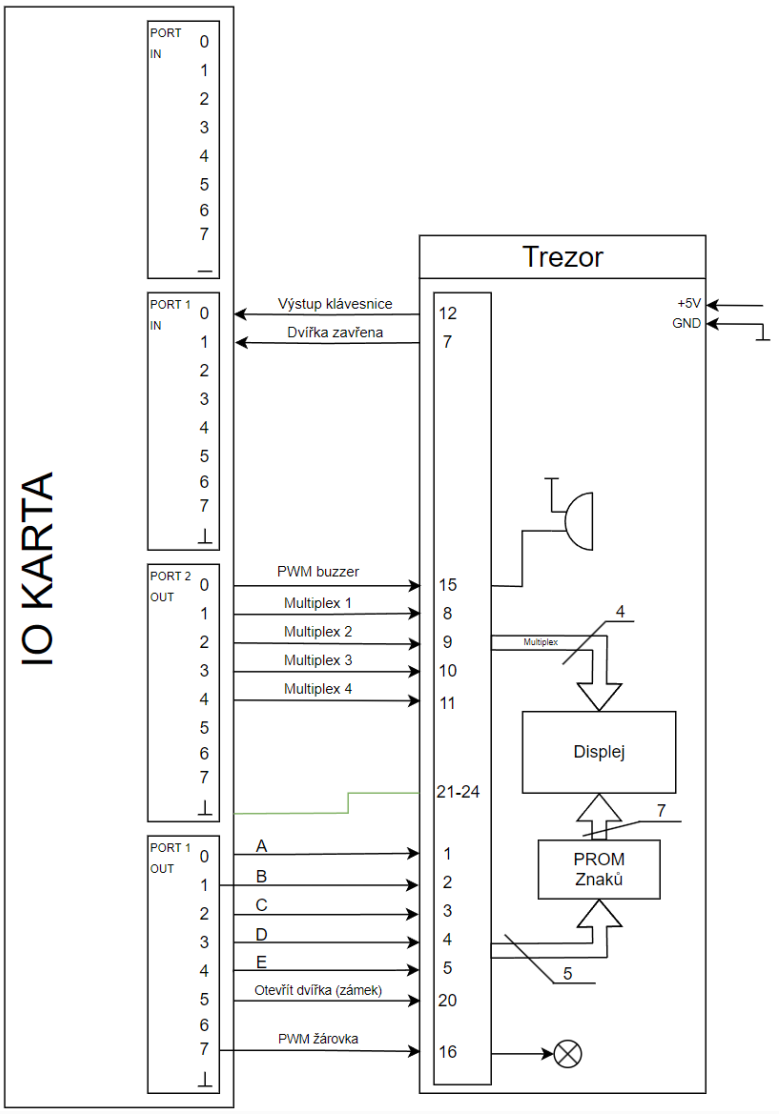

SensorTimerInterrupt
doors opened?
editorMode = true;
light timer running?
stop light timer
port1 = SetBit(LightPin)
editorMode = false;
light timer stopped?
start light timer
return;
ThreadHandler
while(true)
writeToPort(1, port1)
writeToPort(2, port2)
return;
CheckButton(btnIndex)
getPinState(KEYBOARD_PIN) == high
KEYBOARD_PIN = globální proměnná definující vstupní pin klávesnice
return;
holdCounterMap[btnIndex]++;
holdCounterMap = mapa indexu tlačítka a counteru značící dobu stisku
btnIndex == 0 && holdCounterMap[btnIndex] % holdNorm == 0)
editorMode
globální proměnná sloužící pro změnu kódu
code = string(displayData.Values.ToArray());
write the code into the code.txt file
beep(200)
OUT: code successfully changed
current code valid?
unclockDoors();
metoda pro otevření dveří pomocí rychlého impulzu
errorCounter++
errorCounter > 3
beep(1500);
errorCounter = 0;
holdCounterMap[btnIndex] = 0;[
kód pro obsluhu dalších tlačítek (SET, UP a DOWN) je ve symslu ovládání kód značně repetetivní proto je vynechán
return;

Schéma zapojení:



Komentovaný výpis programu:

Program.cs:

```
using System;
using Trezor.Essentials;

namespace Trezor
{
    public class Program
    {
        public static Program prg = new Program();
        public VaultModel vaultModel { get; set; }

        public void Initialize()
        {
            // initialize the vault model
            vaultModel = new VaultModel();
        }
        public static void Main(string[] args)
        {
            AppDomain.CurrentDomain.ProcessExit += new
EventHandler(CurrentDomain_ProcessExit);
            Console.WriteLine("Init start");
```

```
            // run the init
            prg.Initialize();

            Console.WriteLine("Init complete");

            while (true)
            {
                prg.vaultModel.DisplayCode();
            }
        }


        static void CurrentDomain_ProcessExit(object sender, EventArgs e)
        {
            prg.vaultModel.ResetPorts();
            Console.WriteLine("exit");
        }
    }
}
```

BitOperations.cs:

```
/*
 * File: BitOperations.cs
 * Author: Tomáš Bartoš
 * Date: November 3, 2023
 * Description: This file contains the BitOperations class, which provides functions for
bitwise manipulation
 *              with the support for inverted logic.
 */

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Trezor.Essentials
{
    public class BitOperations
    {
        /// <summary>
        /// Method for setting a specific bit in the given byte.
        /// </summary>
        public byte SetBit(byte value, int bit, bool inversedLogic = false)
        {
            if (inversedLogic)
            {
                return (byte)(value & ~(0x01 << bit));
            }
            return (byte)(value | (0x01 << bit));
        }

        /// <summary>
        /// Method for nulling (setting to 0) a specific bit in the given byte.
```

```
        /// </summary>
        public byte NullBit(byte value, int bit, bool inversedLogic = false)
        {
            if (inversedLogic)
            {
                return (byte)(value | (0x01 << bit));
            }
            return (byte)(value & ~(0x01 << bit));
        }

        /// <summary>
        /// Method for inverting all bits in the given byte.
        /// </summary>
        public byte InvertByte(byte value)
        {
            return (byte)~value;
        }

        /// <summary>
        /// Method for toggling a specific bit in the given byte.
        /// </summary>
        public byte ChangeBit(byte value, int bit, bool inversedLogic = false)
        {
            if (inversedLogic)
            {
                return (byte)(value ^ (0x01 << bit));
            }
            return (byte)(value ^ (0x01 << bit));
        }

        /// <summary>
        /// Method for merging two bytes together.
        /// </summary>
        public byte MergeBytes(byte byte0, byte byte1)
        {
            return (byte)(byte0 & byte1);
        }

        /// <summary>
        /// Method for checking if a specific bit of the given byte is high (1).
        /// </summary>
        public bool IsHigh(byte value, int bit, bool inverseLogic = false)
        {
            if (inverseLogic)
            {
                return ((value & (0x01 << bit)) == 0);
            }
            return ((value & (0x01 << bit)) != 0);
        }

        /// <summary>
        /// Method for checking if a specific bit in a given byte is low (0)
        /// </summary>
        public bool IsLow(byte value, int bit, bool inverseLogic = false)
```

```
        {
            if (inverseLogic)
            {
                return ((value & (0x01 << bit)) != 0);
            }
            return ((value & (0x01 << bit)) == 0);
        }
    }
}
```

Board.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Automation.BDaq;
using Trezor.Essentials;

namespace Trezor.Essentials
{
    public partial class Board
    {
        public DeviceInformation dev { get; set; }
        public InstantDoCtrl DO { get; set; }
        public InstantDiCtrl DI { get; set; }

        public BitOperations bitOperations { get; set; }

        public Board()
        {
            InitObjects();
            PortSetup();
        }

        /// <summary>
        /// Method for initializing the device related objects
        /// </summary>
        public void InitObjects()
        {
            // basic device information
            dev = new DeviceInformation
            {
                Description = "PCIE-1730,BID#0",
                DeviceMode = AccessMode.ModeWrite
            };

            // Digitial output information
            DO = new InstantDoCtrl();
            DO.SelectedDevice = dev;
            DO.LoadProfile("CardProfile.xml");

            // Digital input information
            DI = new InstantDiCtrl();
            DI.SelectedDevice = dev;
            DI.LoadProfile("CardProfile.xml");

            // init of external objects
            bitOperations = new BitOperations();
        }

        /// <summary>
        /// Init method for setting up the ouput ports
        /// </summary>
        public void PortSetup()
```

```
        {
            if(VaultWiring.General.INVERSE_LOGIC)
            {
                // set the default state of the ports to 0xFF
                SetByte(0, 0xFF);
                SetByte(1, 0xFF);
            }
            else
            {
                // set the default state of the ports to 0x00
                SetByte(0, 0x00);
                SetByte(1, 0x00);
            }
        }

        /// <summary>
        /// Method for setting a byte to a desired port
        /// </summary>
        public void SetByte(int port, byte data)
        {
            // fucntion for setting a byte to a desired port
            DO.Write(port, data);
        }

        /// <summary>
        /// Method that sets a bit on a given port to a log. 1
        /// </summary>
        public void SetBit(int port, int pinAddr)
        {
            // get the port stratus
            byte status = GetPortState(port);

            // get the bite index
            int index = VaultWiring.GetPinID((byte)pinAddr);

            // edit the desired bit
            byte data;
            if(VaultWiring.General.INVERSE_LOGIC)
            {
                // bring the particicuar bit down
                data = bitOperations.SetBit(status, index, false);
            }
            else
            {
                // bring the particular bit low
                data = bitOperations.SetBit(status, index, true);
            }
            SetByte(port, data);
        }

        /// <summary>
        /// Method for returning the current state of the bit on a given port
        /// </summary>
        /// <returns>
        /// A boolean value representing the state of the bit
        /// </returns>
        public bool GetBit(int port, int bitAddress)
        {
            byte data;
            DI.Read(port, out data);

            if (bitOperations.IsHigh(data, bitAddress))
            {
                return true;
            }

            return false;
        }
```

```
        /// <summary>
        /// Method for returning the current state of the port
        /// </summary>
        /// <returns>
        /// A byte representation of the port state
        /// </returns>
        public byte GetPortState(int port)
        {
            byte state;
            DO.Read(port, out state);
            return state;
        }

        /// <summary>
        /// Method for showing an error message
        /// </summary>
        public void ShowError(string error)
        {
            Console.WriteLine(error);
        }
    }
}
```

PreciseDelay.cs:

```
/*
 * File: PreciseDelay.cs
 * Author: Tomáš Bartoš
 * Date: November 3, 2023
 * Description: This file contains the PreciseDelay method, which provides functions connected
with precise timind and delays.
 */

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Trezor.Essentials
{
    public static class StopwatchExtension
    {
        // extend the functionality of System.Diagnostics stopwatch by adding function that
gets elapsed microseconds
        public static long ElapsedMicroseconds(this Stopwatch stopwatch)
        {
            return stopwatch.ElapsedTicks / (Stopwatch.Frequency / (1_000_000L));
        }
    }

    public class PreciseDelay
    {
        /// <summary>
        /// Method for delaying the program for a specific amount of microseconds
        /// </summary>
        public long DelayMicroseconds(long microseconds)
        {
            Stopwatch delayWatch = Stopwatch.StartNew();

            while (delayWatch.ElapsedMicroseconds() < microseconds)
            {
                // Do nothing, just wait
            }

            delayWatch.Stop();
            return delayWatch.ElapsedMicroseconds();
        }
    }
```

}

VaultModel.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Timers;

namespace Trezor.Essentials
{
    public partial class VaultModel
    {
        public BitOperations bitOperations { get; set; }
        public  PreciseDelay preciseDelay { get; set; }
        public Board board { get; set; }
        public System.Timers.Timer doorOpenTimer { get; set; }
        public System.Timers.Timer lightTimer { get; set; }

        public System.Timers.Timer readingTimer { get; set; }

        public Thread portThread { get; set; }

        public Dictionary<int, int> holdCounterMap = new Dictionary<int, int>()
        {
            {0, 0},
            {1, 0},
            {2, 0},
            {3, 0}
        };

        public Dictionary<int, char> displayData = new Dictionary<int, char>()
        {
            {0, '0'},
            {1, '0'},
            {2, '0'},
            {3, '0'}
        };

        int selectedDial = 0;

        int errorCounter = 0;

        bool editorMode = false;
        public bool canStart = false;

        public bool lastLight = false;

        public int holdNorm = 10;
        public byte port1 = 0xFF;
        public byte port2 = 0xFF;

        public Dictionary<char, byte> map = new Dictionary<char, byte>(){
```

```
            {'0', 0x00},
            {'1', 0x01},
            {'2', 0x02},
            {'3', 0x03},
            {'4', 0x04},
            {'5', 0x05},
            {'6', 0x06},
            {'7', 0x07},
            {'8', 0x08},
            {'9', 0x09},
            {'E',0x0E},
            {'R', 0x17 },
            {' ', 0x1D },
            {'-', 0x1E }
        };

        /// <summary>
        /// Constructor of the VaultModel object.
        /// </summary>
        public VaultModel()
        {
            // iniciatlie the global objects
            bitOperations = new BitOperations();
            preciseDelay = new PreciseDelay();
            board = new Board();

            doorOpenTimer = new System.Timers.Timer();
            doorOpenTimer.Interval = 60_000;
            doorOpenTimer.Elapsed += (sender, e) => Beep(300);

            readingTimer = new System.Timers.Timer();
            readingTimer.Interval = 100;
            readingTimer.Elapsed += CheckSensors;
            readingTimer.AutoReset = true;
            readingTimer.Start();

            lightTimer = new System.Timers.Timer();
            readingTimer.Interval = 1;
            readingTimer.Elapsed += SwitchLight;
            readingTimer.AutoReset = true;


            // initiate the port thread
            portThread = new Thread(PortThreadInterrupt);
            portThread.Priority = ThreadPriority.Highest;
            portThread.Start();
        }

        /// <summary>
        /// Interrupt handler for the light timer. Its main function is to switch the
light on and off.
        /// </summary>
        private void SwitchLight(object? sender, ElapsedEventArgs e)
        {
```

```
            lastLight = !lastLight;
            port1 = bitOperations.SetBit(port1,
VaultWiring.GetPinID(VaultWiring.Out.PORT1.LIGHT_PIN), lastLight);
        }

        /// <summary>
        /// Method for resetting the ports to their default state.
        /// </summary>
        public void ResetPorts()
        {
            board.SetByte(VaultWiring.Out.PORT1.PORT, 0xFF);
            board.SetByte(VaultWiring.Out.PORT2.PORT, 0xFF);
        }

        /// <summary>
        /// Timer interrupt handler for checking the door sensor status every N seconds.
        /// </summary>
        private void CheckSensors(object? sender, ElapsedEventArgs e)
        {
            if (board.GetBit(VaultWiring.In.PORT,
VaultWiring.GetPinID(VaultWiring.In.DOOR_PIN)))
            {
                // door is opened -> light full brightness
                // ideally I will have a pwm thread which will be set to full brightness
                editorMode = true;

                Console.WriteLine("Doors opened");

                Console.WriteLine("#0 " + lightTimer.Enabled);
                if (lightTimer.Enabled)
                {
                    Console.WriteLine("Stopping timer");
                    lightTimer.Enabled = false;
                }
                Console.WriteLine("#1 " + lightTimer.Enabled);

                // set the pin high -> light full brightness
                port1 = bitOperations.SetBit(port1,
VaultWiring.GetPinID(VaultWiring.Out.PORT1.LIGHT_PIN), false);
            }
            else
            {
                // door is closed
                editorMode = false;

                if(lightTimer.Enabled == false)
                {
                    // PERFORM PWM
                    Console.WriteLine("Starting PWM");
                    lightTimer.Enabled = true;
                }
                //port1 = bitOperations.SetBit(port1,
VaultWiring.GetPinID(VaultWiring.Out.PORT1.LIGHT_PIN), true);
```

```
            }
        }

        /// <summary>
        /// A method for displaying the code on the vault display by multiplexing the 7-
segment displays.
        /// </summary>
        public void DisplayCode()
        {
            // method for displaying the code on the vault
            for (int i = 0; i < displayData.Count; i++)
            {
                // get the current character
                char character = displayData[i];
                byte value = map[character];

                // set PROM data

                port1 = (byte)((port1 & 0b11100000) | ((0x1F & value)));

                port2 = bitOperations.MergeBytes(port2, (byte)(0xFF ^
(VaultWiring.Out.PORT2.MULTIPLEX_0 << i)));

                preciseDelay.DelayMicroseconds(5000);

                // check button
                CheckButttton(i);

                // unset the port2 state to hight
                port2 = bitOperations.SetBit(port2, i+1);
                preciseDelay.DelayMicroseconds(50);
            }
        }

        /// <summary>
        /// A method for checking the button status and performing the corresponding
action.
        /// </summary>
        public void CheckButttton(int btnIndex)
        {
            if(board.GetBit(VaultWiring.In.PORT,
VaultWiring.GetPinID(VaultWiring.In.KEYBOARD_PIN)))
            {
                // log. 1 on port -> nothing to perform -> return
                return;
            }

            holdCounterMap[btnIndex]++;

            if(btnIndex == 0 && holdCounterMap[btnIndex] % holdNorm == 0)
            {
                // mode button is pressed -> check the code
                Console.WriteLine("MODE");
```

```
            if(editorMode)
            {
                // if the door is opened -> change the code
                // write the code to the file
                string code = new string(displayData.Values.ToArray());
                File.WriteAllText(VaultWiring.General.CODE_FILE, code);
                Console.WriteLine("Code changed to: " + code);
                Beep(200);
            }
            else
            {
                // if the door is closed -> check the code
                if (validCode())
                {
                    // open the door
                    unlockDoors();
                }
                else
                {
                    errorCounter++;
                    if (errorCounter > VaultWiring.General.MAX_ERROR - 1)
                    {
                        errorCounter = 0;
                        Beep(1500);
                    }
                }
            }

            holdCounterMap[btnIndex] = 0;
        }
        if (btnIndex == 1 && holdCounterMap[btnIndex] % holdNorm == 0)
        {
            // button up is pressed -> increment the current value
            int newValue = Int32.Parse(displayData[selectedDial].ToString()) + 1;
            if(newValue > 9)
            {
                newValue = 0;
            }
            displayData[selectedDial] = Char.Parse(newValue.ToString());
            Console.WriteLine("UP");

            holdCounterMap[btnIndex] = 0;
        }
        if (btnIndex == 2 && holdCounterMap[btnIndex] % holdNorm == 0)
        {
            // button down is pressed -> decrement the current value
            int newValue = Int32.Parse(displayData[selectedDial].ToString()) - 1;
            if (newValue < 0)
            {
                newValue = 9;
            }
            displayData[selectedDial] = Char.Parse(newValue.ToString());

            Console.WriteLine("DOWN");
```

```
                holdCounterMap[btnIndex] = 0;
            }
            if(btnIndex == 3 && holdCounterMap[btnIndex] % holdNorm == 0)
            {
                // set button is pressed -> select the next dial and beep
                Console.WriteLine("SET");
                selectedDial++;
                if(selectedDial > displayData.Count - 1)
                {
                    selectedDial = 0;
                }
                holdCounterMap[btnIndex] = 0;

                Beep(100);
            }
        }
        /// <summary>
        /// A mtethod for unlocking the doors and starting the door open timer to
signalize the doors being opened for too long.
        /// </summary>
        private void unlockDoors()
        {
            port1 = bitOperations.SetBit(port1, VaultWiring.GetPinID(0x20));
            preciseDelay.DelayMicroseconds(100000);
            port1 = bitOperations.SetBit(port1, VaultWiring.GetPinID(0x20), true);
        }

        /// <summary>
        /// A method for checking the current code with the code stored in the code.txt
file.
        /// </summary>
        private bool validCode()
        {
            int code;
            // read code.txt file and compare the value with the current code
            try
            {
                string[] lines = File.ReadAllLines(VaultWiring.General.CODE_FILE);
                code = Int32.Parse(lines[0]);
            }
            catch
            {
                // if file doesa not exist
                File.WriteAllText(VaultWiring.General.CODE_FILE,
VaultWiring.General.DEFAULT_CODE.ToString());
                code = VaultWiring.General.DEFAULT_CODE;
            }

            int currentCode = Int32.Parse(new string(displayData.Values.ToArray()));
            return code == currentCode;
        }

        /// <summary>
        /// A method for beeping the sound for a given interval.
```

```
        /// </summary>
        public void Beep(int interval)
        {
            // set beep port high wait and set beep low and wait
            for(int i=0; i < interval; i++)
            {
                port2 = bitOperations.SetBit(port2,
VaultWiring.GetPinID(VaultWiring.Out.PORT2.SOUND_PIN), true);
                preciseDelay.DelayMicroseconds(1000);
                port2 = bitOperations.SetBit(port2,
VaultWiring.GetPinID(VaultWiring.Out.PORT2.SOUND_PIN));
                preciseDelay.DelayMicroseconds(250);
            }
        }

        /// <summary>
        /// A method for asigning the current state of the ports to the HW ports.
        /// Helps with the access management to physical ports.
        /// </summary>
        public void PortThreadInterrupt()
        {
            Console.WriteLine("Thread interrupt started");

            // method solely for writing the curent state of the ports to the HW ports
            while (true)
            {
                // assign the current state of the ports to the HW port
                board.SetByte(VaultWiring.Out.PORT1.PORT, port1);
                board.SetByte(VaultWiring.Out.PORT2.PORT, port2);
            }
        }
    }
}
```

VaultWiring.cs:

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Trezor.Essentials
{
    public static class VaultWiring
    {
        public class Out
        {
            public class PORT1
            {
                public const int PORT = 0;
                public const int PROM_0 = 0x01;
                public const int PROM_1 = 0x02;
                public const int PROM_2 = 0x04;
                public const int PROM_3 = 0x08;
                public const int PROM_4 = 0x10;

                public const int UNLOCK_PIN = 0x20;
```

```
                public const int MOTOR_PIN = 0x40;
                public const int LIGHT_PIN = 0x80;
            }
            public class PORT2
            {
                public const int PORT = 1;

                public const int SOUND_PIN = 0x01;

                public const int MULTIPLEX_0 = 0x02; // 0b0000 0010
                public const int MULTIPLEX_1 = 0x04; // 0b0000 0100
                public const int MULTIPLEX_2 = 0x08; // 0b0000 1000
                public const int MULTIPLEX_3 = 0x10; // 0b0001 0000
            }
        }

        public static class In
        {
            public const int PORT = 0;

            public const int KEYBOARD_PIN = 0x01;
            public const int DOOR_PIN = 0x02;
        }

        public class General
        {
            public const bool INVERSE_LOGIC = true;
            public const int LIGHT_FREQ = 50000; // max PWM frequency

            public const long LIGHT_MODULO = 1_000_000 / LIGHT_FREQ;
            public const int DEFAULT_CODE = 1234; // default vault code
            public const int MAX_ERROR = 3; // max error count
            public const string CODE_FILE = "code.txt"; // file with the code
        }

        /// <summary>
        /// A method to get the pin index on a port from its byte address.
        /// </summary>
        public static int GetPinID(byte pinAddress)
        {
            return (int)Math.Log(pinAddress, 2);
        }
    }
}
```

Odpovědi na otázky:

1. Pokud budeme zvažovat zařízení s „odemčením činnosti“ (např. samoobslužná tankovací stanice), můžete narazit na identifikaci pomocí iButtonu. Pojmenujte a stručně popište protokol pro obsluhu tohoto typu identifikačních prvků.[1]

   iButtony používají pro komunikaci se čtečkou tzv. 1 – Wire protokol. Jedná se o half-duplex protokol komunikující pomocí jedné datové linky s poměrně nízkým ratem datového toku (16.3 kbit/s). Byl vyvinut společností Dallas Semiconductor. Jeho princip funkce spočívá v tom, že by-default je datová linka ve stavu logické 1 (3.3 nebo 5V), což krom jednodušší komunikace pro slave zařízení znamená, že tato zařízení nepotřebují externí zdroj napětí.

   Konkrétní komunikace mezi iButtonem a čtečkou funguje takto:
   - Inicializace:
     - Čtečka vysílá „reset“ impuls na datovou linku
   - Identifikace:
     - Čtečka vysílá „presence“ impuls

[1] https://en.wikipedia.org/wiki/1-Wire

        - iButton odpoví svým „presence“ impulsem
        - iButton odesílá své 64 bitové unique ID
    - Výměna dat:
        - Čtečka odesílá „Match ROM“ společně s unique ID tak, aby odpověděl pouze konkrétní iButton
        - Čtečka si odesílá čtecí requesty a iButton odesílá svá interní data sloužící k indetifikaci

2. V případě „connected“ zařízení s možností obsluhy např. z mobilní aplikace vám bude jistě záležet na zabezpečení důvěrnosti a integrity přenášených dat. Pokuste se definovat tyto dva pojmy svými slovy.

- Důvěrnost = Data, která jsou přijímána serverem či libovolným zařízením ke zpracování opravdu pocházejí od platného zdroje a nejsou odesílány pomocí cizích zařízení/aplikací vydávajících se za originály. Příkladem útoku na důvěrnost příchozích dat je Cross-Site Request Forgery, kdy se data do validního cíle odesílají z cizích webových stránek.
- Integrita = Data nejsou při přenosu, jakkoliv změněna a upravena. Příkladem napadení tohoto aspektu je odeslání platby 100$ na účet XY, kdy útočník odchytí komunikaci a packet pro banku přepíše tak, aby peníze přišli na účet ZX.

3. V rámci řešení „chytré domácnosti“ můžeme najít řadu prvků, jejichž stav je nutné zaznamenat (otevření okna, teplota, změna ovládacího prvku apod.). Pokuste se najít a stručně popsat některý ze standardních protokolů určených pro přenos tohoto typu informací.[2]

- MQTT:
    - Jedná se o protokol pro publikování a odběr zpráv
    - Je light-weight a relativně účinný, což ho právě dělá ideálním pro Smart Home
    - Pracuje na principu client-server
        - Klient (senzor) publikuje na server data pod určitá témata (topics)
        - Ostatní klienti mohou dané zprávy odebírat
    - Komunikace je asynchronní (není vyžadován ACK po obdržení)

Závěr:

Výsledkem řešení této úlohy je plně funkční implementace trezoru, která splňuje všechny parametry ze zadání. Pro produkční verzi produktu by bylo dobré naimplementovat zvukové signály a delší dobu prodlev při špatném zadání kódu a případně integraci modulu WiFi společně s access management systémem tak, aby se produkt stal lákavým pro zákazníka a jednoduše magnetovatelným odkudkoliv na světě.

[2] https://en.wikipedia.org/wiki/MQTT, https://mqtt.org/