PRAXE

č. 5	STŘEDNÍ PRŮMYSLOVÁ ŠKOLA CHOMUTOV	David Herko
25. 3. 2025	MIKROVLNKA	V4

Zadání

S využitím IO karty sestavte v jazyku C# program pro ovládání modelu mikrovlnné trouby. Dle svého uvážení zvolte buď prostředí konzolové aplikace nebo grafické prostředí. Výsledné řešení musí obsahovat následující funkcionalitu:

- Nastavení času ohřevu (alespoň po desítkách sekund) a výkonu ohřevu ve 2 úrovních

- Manuální spuštění cyklu ohřevu s možností kdykoliv jej přerušit, otevřít dvířka a dále pokračovat nebo zrušit

- Zvukové znamení a zhasnutí po skončení ohřevu, pomalou rotaci talíře během cyklu ohřevu

- Kontrolu bezpečnosti provozu (nemožnost spustit s otevřením dvířek, automatické přerušení ohřevu při porušení bezpečnosti

Teorie

Model mikrovlnné trouby má na sobě 7 segmentový displej se čtyřmi znaky. Pro ovládání pozice, na kterou chceme psát data posíláme na piny hodinový / multiplex signál. Čtyři ovládací tlačítka jsou napojena na segmenty, a když je multiplex v pozici onoho tlačítka, můžeme ho zmáčknout.

Pro připojení na mikrokontroller používáme knihovnu `Automation.Bdaq`. Načteme profil zařízení pomocí příkazů

```
DeviceInformation di = new DeviceInformation();
di.Description = "PCIE-1730,BID#0";
di.DeviceMode = AccessMode.ModeWrite;

InstantDoCtrl ioDO = new InstantDoCtrl();
ioDO.SelectedDevice = di;
ioDO.LoadProfile("profile.xml");

InstantDiCtrl ioDI = new InstantDiCtrl();
ioDI.SelectedDevice = di;
ioDI.LoadProfile("profile.xml");
```

Pro práci s bitovými posuny používám svojí knihovnu `Dejvoss.BitShift`.

Popis řešení

Program se po spouštění připojí na mikrokontroller a nastaví na něm výchozí hodnoty přes `MicrowaveController.setup()`. Dvířka se otevřou `MicrowaveController.unlock()` a režim se nastaví na `Opened`. Uživatel tak nemůže zadávat hodnoty, dokud se zase nezavře. Následně se spustí dva Tasky – `displayTask` pro zobrazování toho, co je na displeji a `logicTask`, ve kterém se děje veškerá logika. To obnáší kontrolu dvířek `MicrowaveController.isDoorOpen()`, odčítání času `MicrowaveController.countDown()` pokud se vaří a spouští se metody dle zmačknutých tlačítek `MicrowaveController.getButtonPress()`. Tyto Tasky běží asynchronně a opakují se do nekonečna (ukončení programu).

Rozbor proměnných a metod

Program.cs

`Controller c`	Odkaz na ovládání IO karty
`MicrowaveController mc`	Odkaz na ovládání mikrovlnky
`int[] buttonIDs`	ID tlačítek

`Int logicPos`	Multiplex
`int sleepTime`	Jak dlouho spát mezi dalším multiplexem
`void Main(string[] args)`	Vstup programu, nastavení IO karty, start
`async Task displayTask()`	Task pro zobrazování na displeji
`async Task logicTask()`	Task pro logickou část programu
`byte switchAll(byte input)`	Přehodí všechny hodnoty v byte (0 → 1 a 1 → 0)
`byte[] numberToByte(int input)`	Hodnota int na byte

MicrowaveController.cs

`Controller c`	Odkaz na Controller
`byte[] chars`	Znaky, které se ukazují na displeji
`byte[] numbers`	Čísla, které se ukazují na displeji při vaření a nastavení
`int caretPos`	Pozice, kterou upravujeme při nastavení
`Enums.MicrowaveMode mode`	Režim mikrovlnky
`void setup()`	Nastavení výchozích hodnot
`int getButtonPress(int currentPos)`	Získá ID zmáčknutého tlačítka
`void startCountDown()`	Nastaví režim do vaření a začne odečítat čas
`void countDown()`	Samotné odečítání času
`void changeCarotPos()`	Změní pozici, kterou upravujeme
`void upButton()`	Zmáčknutí tlačítka pro přičítání času
`void downButton()`	Zmáčknutí tlačítka pro odečítání času
`void unlock()`	Otevře mikrovlnku
`bool isDoorOpen()`	Je mikrovlnka otevřená?

Controller.cs

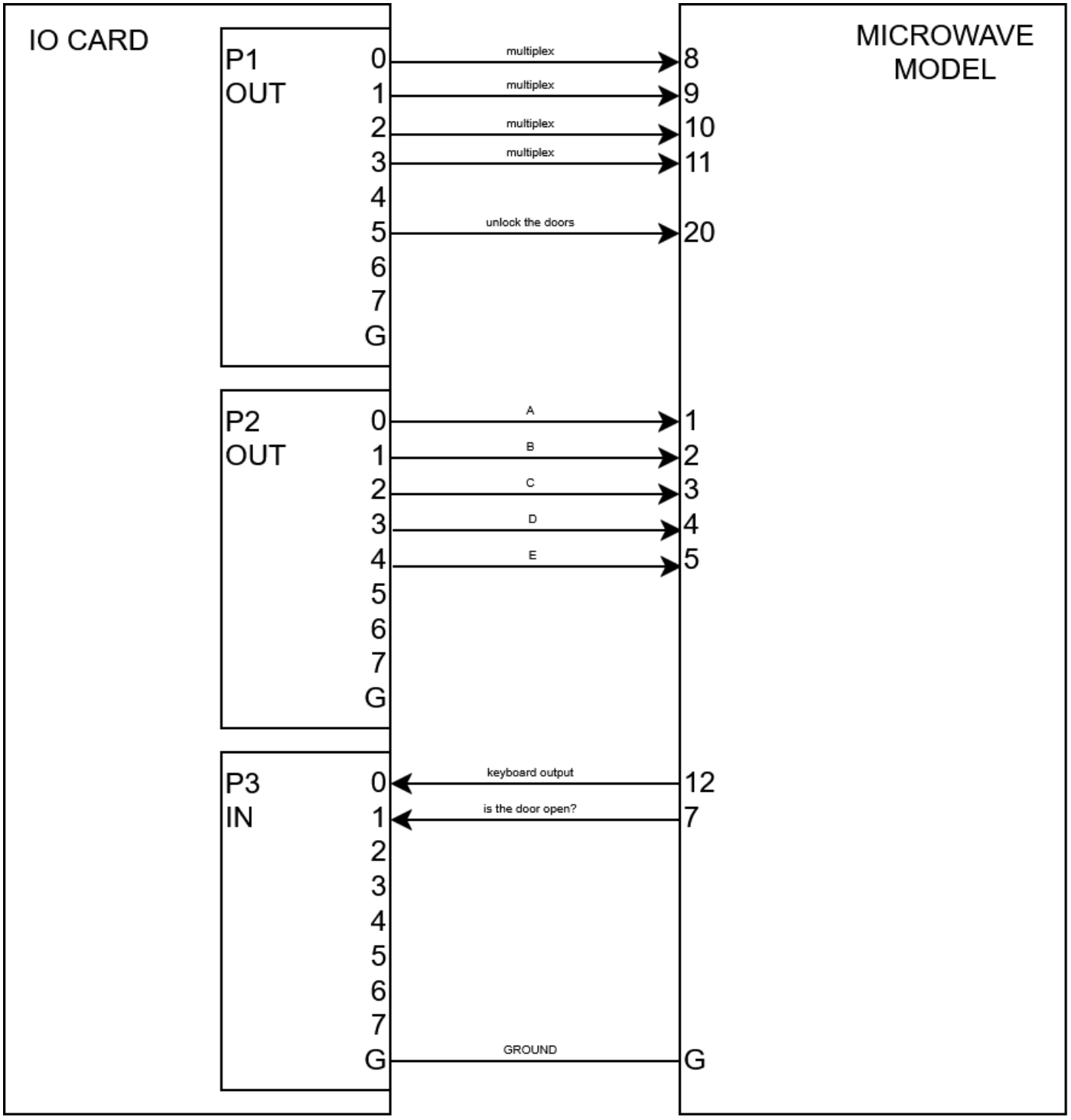
`InstantDoCtrl ioDO, InstantDiCtrl ioDI`	Odkazy na mikrokontroller
`byte[] portData`	Data na portech
`int port1out, port2out, port3in, port4in`	Čísla portů
`void write(int port, byte input)`	Napíše [input] na port [port]

`byte read(int port)`	Přečte data z portu [port]

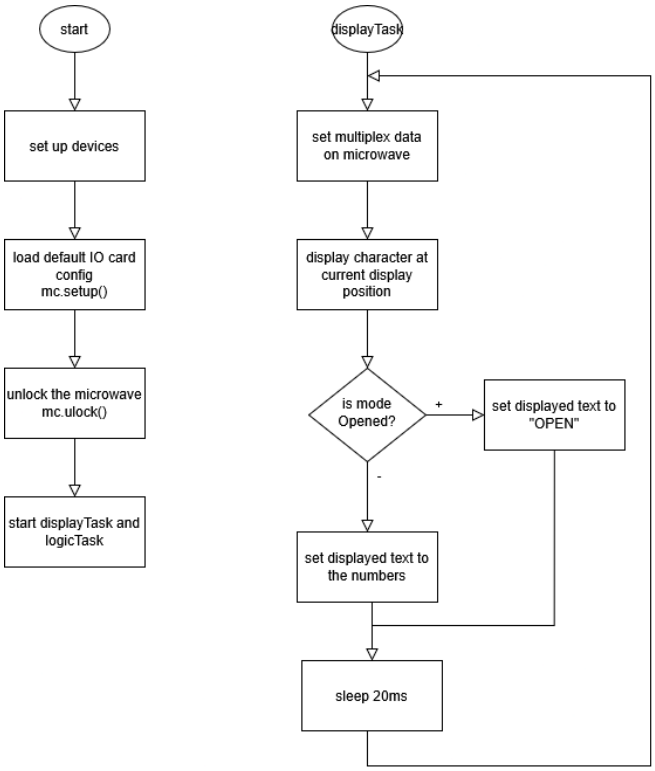
Enums.cs

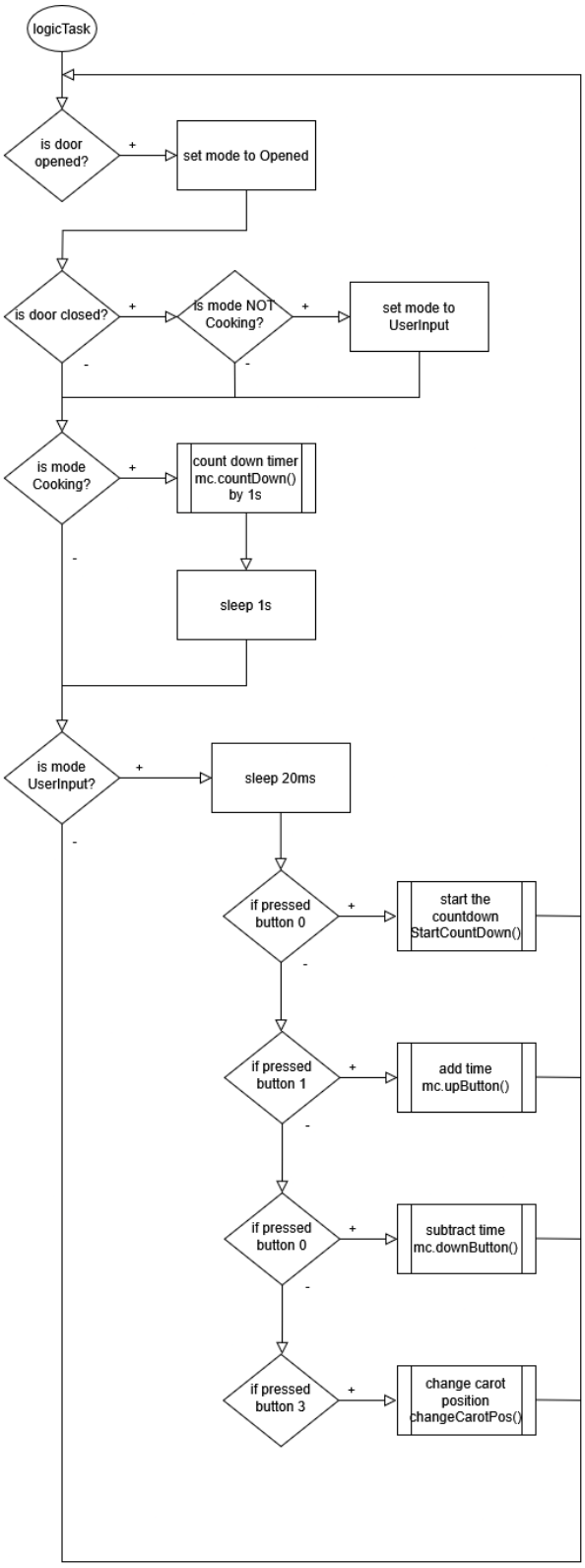
`enum MicrowaveMode`	Enum pro nastavování režimu mikrovlnky

Schéma zapojení



Vývojový diagram





Komentovaný výpis kódu

Program.cs

```
using Automation.BDaq;
using Dejvoss.BitShift;
using System;
using System.Threading;
using System.Threading.Tasks;

namespace Herko.Microwave
{
    internal class Program
    {
        public static Controller c;
        public static MicrowaveController mc;

        public static int[] buttonIDs = { 1, 2, 4, 8 }; //button IDs
        public static int logicPos = 0;

        public static int sleepTime = 20;

        static void Main(string[] args)
        {
            Console.Title = "Microwave v0.1.0 | (c) DEJVOSS Productions 2025";

            //device info
            DeviceInformation di = new DeviceInformation();
            di.Description = "PCIE-1730,BID#0";
            di.DeviceMode = AccessMode.ModeWrite;

            InstantDoCtrl ioDO = new InstantDoCtrl();
            ioDO.SelectedDevice = di;
            ioDO.LoadProfile("profile.xml");

            InstantDiCtrl ioDI = new InstantDiCtrl();
            ioDI.SelectedDevice = di;
            ioDI.LoadProfile("profile.xml");

            //init controller objs
            c = new Controller(ioDO, ioDI);
            mc = new MicrowaveController(c);

            //set defaults
            mc.setup();

            //unlock the door on startup
            mc.unlock();

            //run logic and graphic tasks
            Task.Run(displayTask);
            Task.Run(logicTask);

            Console.ReadLine(); //so the window doesn't close lol
        }

        public static async Task displayTask()
        {
```

```
            while (true)
            {
                for (int i = 0; i < buttonIDs.Length; i++)
                {
                    logicPos = i;

                    //which button was pressed?
                    int btn = buttonIDs[logicPos];

                    //convert number to binary but everything is switched
                    byte test = switchAll((byte)btn);

                    //set btn position byte to port
                    c.write(c.port1out, test);
                    c.write(c.port2out, mc.chars[logicPos]);

                    //if opened, display "open" message
                    if (mc.mode == Enums.MicrowaveMode.Opened)
                    {
                        mc.chars = new byte[] { 0x00, 0x16, 0x0E, 0x15 };
                    }
                    else // or just display numbers
                    {
                        //display numbers (this could be prettier but shhh)
                        byte[] toDisplayFirst = numberToByte(mc.numbers[0]);
                        byte[] toDisplaySecond = numberToByte(mc.numbers[1]);

                        mc.chars = new byte[] { toDisplayFirst[0], toDisplayFirst[1],
toDisplaySecond[0], toDisplaySecond[1] };
                    }

                    Thread.Sleep(sleepTime);
                }
            }

            displayTask();
        }

        public static async Task logicTask()
        {
            while (true)
            {
                //here check if the microwave is opened
                if (mc.isDoorOpen())
                    mc.mode = Enums.MicrowaveMode.Opened;
                else if (!mc.isDoorOpen() && mc.mode != Enums.MicrowaveMode.Cooking)
                    mc.mode = Enums.MicrowaveMode.UserInput;

                if (mc.mode == Enums.MicrowaveMode.Cooking)
                {
                    //count down
                    mc.countDown();
                    Thread.Sleep(1000);
                }
                else if (mc.mode == Enums.MicrowaveMode.UserInput)
                {
                    Thread.Sleep(sleepTime);
```

```
                    //get button press
                    int btnPress = mc.getButtonPress(logicPos);

                    if (btnPress == 0) //mode button
                        mc.startCountDown();
                    else if (btnPress == 1) //up button
                        mc.upButton();
                    else if (btnPress == 2) //down button
                        mc.downButton();
                    else if (btnPress == 3) //set button
                        mc.changeCarotPos();
                }
            }

            logicTask();
        }

        static byte switchAll(byte input)
        {
            byte output = input;
            for (int i = 0; i < 8; i++)
            {
                output = EightBit.switchAt(output, i);
            }

            return output;
        }

        static byte[] numberToByte(int input)
        {
            String inputStr = input.ToString();

            if (inputStr.Length == 2)
            {
                return new byte[] { (byte)int.Parse(inputStr[0].ToString()),
(byte)int.Parse(inputStr[1].ToString()) };
            }
            else
            {
                return new byte[] { 0, (byte)int.Parse(inputStr[0].ToString()) };
            }
        }
    }
}
```

MicrowaveController.cs

```
using Dejvoss.BitShift;
using System;

namespace Herko.Microwave
{
    public class MicrowaveController
    {
        private Controller c;

        public byte[] chars = { 0b00000000, 0b00000000, 0b00000000, 0b00000000 }; //the
"memory" of what characters are gonna be displayed on screen
        public byte[] numbers = { 0, 0 }; //the two numbers on the display
```

```
        public int caretPos = 0; //position of the caret, 0 or 1

        public Enums.MicrowaveMode mode;

        public MicrowaveController(Controller c)
        {
            this.c = c;
        }

        //setup with default values
        public void setup()
        {
            c.write(c.port1out, 0b11111111);
            c.write(c.port2out, 0b11111111);
        }

        //returns ID of pressed button (0 - 3); -1 means no button pressed
        public int getButtonPress(int currentPos)
        {
            if (EightBit.getAt(c.read(c.port3in), 0) == 0)
            {
                return currentPos;
            }

            return -1;
        }

        //starts to count the time down
        public void startCountDown()
        {
            Console.WriteLine("cooking!!");
            mode = Enums.MicrowaveMode.Cooking;
        }

        //counts the time down
        public void countDown()
        {
            //for now just start the countdown
            numbers[1]--;
            if (numbers[1] > 60)
            {
                //remove one minute
                numbers[1] = 59;
                numbers[0]--;

                if (numbers[0] > 60)
                {
                    //end countdown ig
                    numbers[0] = 0;
                    numbers[1] = 0;
                    mode = Enums.MicrowaveMode.UserInput;
                    unlock();
                }
            }
        }

        //changes carot position
        public void changeCarotPos()
```

```
        {
            if (caretPos == 0)
                caretPos = 1;
            else
                caretPos = 0;

            Console.WriteLine("caretPos " + caretPos);
        }

        //moves the number at [caretPos] up
        public void upButton()
        {
            numbers[caretPos]++;
            if (numbers[caretPos] > 60)
                numbers[caretPos] = 0;

            Console.WriteLine($"numbers[{caretPos}] " + numbers[caretPos]);
        }

        //moves the number at [caretPos] down
        public void downButton()
        {
            numbers[caretPos]--;
            if (numbers[caretPos] > 60)
                numbers[caretPos] = 60;

            Console.WriteLine($"numbers[{caretPos}] " + numbers[caretPos]);
        }

        //unlocks the lock
        public void unlock()
        {
            mode = Enums.MicrowaveMode.Opened;
            c.write(c.port1out, EightBit.setAt(c.read(c.port1out), 5, 0));
            Console.WriteLine($"unlocked");
        }

        //returns true if door is opened
        public bool isDoorOpen()
        {
            if (EightBit.getAt(c.read(c.port3in), 1) == 0)
            {
                return false;
            }

            return true;
        }
    }
}
```

Controller.cs

```
using Automation.BDaq;

namespace Herko.Microwave
{
    public class Controller
    {
        //controllers
```

```
        public InstantDoCtrl ioDO;
        public InstantDiCtrl ioDI;

        //bytes to work with
        public byte[] portData = new byte[4];

        //ports
        public int port1out = 0;
        public int port2out = 1;
        public int port3in = 0;
        public int port4in = 1;

        //construct
        public Controller(InstantDoCtrl ioDO, InstantDiCtrl ioDI)
        {
            this.ioDO = ioDO;
            this.ioDI = ioDI;
        }

        //write [input] to [port]
        public void write(int port, byte input)
        {
            ioDO.Write(port, input);
            portData[port] = input;
        }

        //reads byte at [port]
        public byte read(int port)
        {
            byte data;
            ioDI.Read(port, out data);

            return data;
        }
    }
}
```

Enums.cs

```
namespace Herko.Microwave
{
    public class Enums
    {
        public enum MicrowaveMode
        {
            Opened,
            UserInput,
            Cooking
        }
    }
}
```

Odpovědi na otázky

1) Na jakém principu je založen ohřev jídla v reálné mikrovlnné troubě?

> Magnetron generuje elektromagnetické záření o frekvenci 2,4GHz, které je posíláno na jídlo v mikrovlnce a hýbe v něm molekulami vody. Tím vzniká teplo a jídlo se ohřívá.[1]

2) Stále rostoucím trendem v zařízeních spotřební elektroniky je modulárnost. Co by bylo třeba zajistit, aby např. Ovládací rozhraní MVT mohl dodávat výrobci externí dodavatel, příp. aby stejný model zařízení mohl existovat ve verzi s tlačítkovým nebo dotykovým panelem při zachování zbytku výkonové elektroniky?

> Nějak nerozumím, co je onou „modulárností“ myšleno. Nestačí prostě a jednoduše přidělat jiný druh displeje a vyřešit tím problém jiného rozložení tlačítek? A když chci, aby mi ovládací rozhraní vyráběl někdo jiný, pošlu mu schéma, jak to má vypadat.

3. Reálná mikrovlnná trouba má možnost nastavit několik výkonových stupňů ohřevu (např. rozmrazování, 250W – 600W a plný výkon). Vysvětlete jak je toho dosaženo.

> Upravuje se výkon magnetronu, ten vydává více či méně záření. Magnetron se ale více zahřívá a je proto třeba nainstalovat ventilační systém.[2]

Závěr

Mikrovlnka se mi, troufnu si říct, poměrně povedla. K dokonalosti chybělo ovládání světla a motoru pomocí PWM k simulování vaření. Z důvodu méně času na debugging se mi také nepodařilo vyřešit občasné přeskočení multiplexu v jednom Tasku, čímž se spustí následující funkce při zmáčknutí tlačítka.

1 https://en.wikipedia.org/wiki/Microwave_oven

2 https://itechnic.techinfus.com/cs/dlja-kuhni/mikrovolnovka/princip-raboty-mikro/