STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 10: Elektrotechnika, elektronika a telekomunikace

Interaktivní světelná tabule

Jan Gebhart
Plzeňský kraj

Plzeň 2019

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 10: Elektrotechnika, elektronika a telekomunikace

Interaktivní světelná tabule

Interactive light board

Autor: Jan Gebhart
Škola: Vyšší odborná škola a Střední průmyslová škola elektrotechnická Plzeň, Koterovská 828/85, 326 00 Plzeň
Kraj: Plzeňský kraj
Konzultant: Ing. Jiří Mazanec

Plzeň 2019

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Plzni dne 13. 2. 2019 ……

Jan Gebhart

Poděkování

V první řadě bych chtěl poděkovat Ing. Karlovi Hajžmanovi a Ing. Jiřímu Mazancovi za prvotní myšlenku, motivaci a vedení práce, všem učitelům za pomoc a odborné rady. A děkuji i svým spolužákům, kteří mi pomáhali s výrobou.

Anotace

Ve své práci SOČ jsem se zabýval návrhem a realizací interaktivní světelné tabule. Jedná se o matici pixelů řízenou mikroprocesorem, kde jednotlivý pixel může zobrazit libovolnou barvu. Dále každý pixel detekuje objekty, které se nacházejí před ním. Díky tomu lze pro interaktivní tabuli naprogramovat nejrůznější aplikace a hry. Součástí práce je popis algoritmu několika jednoduchých her.

Klíčová slova

infračervený; LED; matice; multiplex, interaktivní

Annotation

In my SOČ, I have been working on designing and constructing an interactive light board. It is a pixel matrix controled with a microprocessor, where the pixels are independent on each other. Another feature it has is that each pixel can detect objects in front of it, whitch means that multiple games and applications can be programmed for this board. An example of the algorithm for a few simple games is a part of this work.

Keywords

infra-red; LED; matrix; multiplex, interactive

Obsah

1 Úvod

Když jsem přišel do prvního ročníku na střední škole, oslovil mě Ing. Karel Hajžman s tím, že jeden jeho známý potřebuje něco na měření reakční doby brankářů. Nejprve jsem přišel s myšlenkou tabule, která představuje bránu a na kterou bych umístil matici 8 x 8 žárovek. Každá žárovka by byla zakryta pohyblivým krytem, který by byl napojený na mikrospínač.

Tento panel jsem potřeboval nějak řídit, ale v té době jsem ještě neuměl programovat, tak jsem šel nejjednodušší cestou přes Arduino. Jak jsem se postupně učil další a další věci, přišlo mi škoda mít jen jednobarevné zobrazení, proto jsem přešel k RGB LED namísto žárovek. Později nestačil ani počet pixelů, a tak jsem kvůli potřebě vykreslovat složitější věci přešel na matici 20 x 10 pixelů. Vyrobit 200 spolehlivých průsvitných a robustních tlačítek byl problém. Proto jsem začal testovat jiné alternativy. Ultrazvukové snímání vyžadovalo udělat do panelu otvory, kapacitní snímání vyžadovalo vytvoření průhledné vodivé vrstvy a infračervené snímání nevyžadovalo úpravy žádné. Proto jsem nakonec použil toto infračervené snímání (jak se později ukázalo, jeho nevýhodou je infračervené pozadí).

Začal jsem se stavbou prototypu 4 x 10 pro testování přepínání mezi řádky. Prototyp byl řízen pomocí Arduino UNO (programován v jazyce Wiring). Analogové signály jsem načítal ze 4 sloupečků pomocí ADC převodníku v Arduinu a infračervené pozadí jsem odstraňoval softwarově. Obnovovací frekvence ale byla nedostačující (cca 20 Hz). Od tohoto okamžiku jsem vylepšoval jen řídící jednotku. Zapojení snímačů a zobrazovacích LED zůstalo nezměněné.

V této práci popíšu jednotlivé problematiky a řešení, která jsem použil v konečné finální verzi o rozměrech 20 x 10 pixelů a obnovovací frekvenci 250 Hz.

Protože se tomu věnuji přes 3 roky, přišel jsem na mnoho dalších využití, jako jsou nejrůznější interaktivní aplikace a hry, o kterých se také zmíním. Tabule může být využita při výuce programování, studenti budou pomocí jednoduchého rozhraní přímo ovládat vstupy i výstupy tabule, a to přímo v softwaru tabule (v jazyce C/C++) nebo v jakémkoli jiném programovacím jazyce přes USB. Do budoucna plánuji pro úplné začátečníky zpřístupnit periferie tabule v programu Scratch (jednoduchý grafický programovací jazyk pro výuku programování).

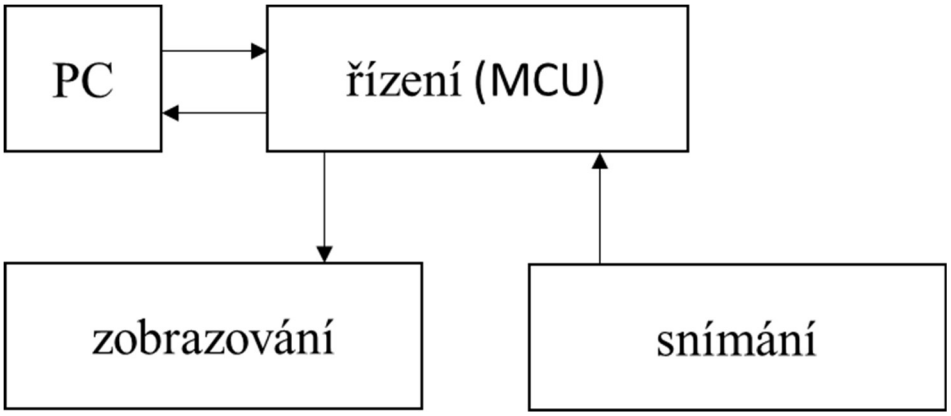
Prakticky vše, co jsem použil při tvorbě interaktivní tabule, pro mě bylo nové a musel jsem se to učit. Sice to zabralo více času, ale to mi nijak nevadilo, protože neznám lepší způsob, jak se seznámit s novými technologiemi a poznatky, než formou experimentů, poznávání, objevování a vymýšlení vymyšleného.

Obrázek 1: Úvodní foto "finální verze při výrobě".

Obrázek 2: Foto "finální verze při testování".

2 PRINCIP ČINNOSTI

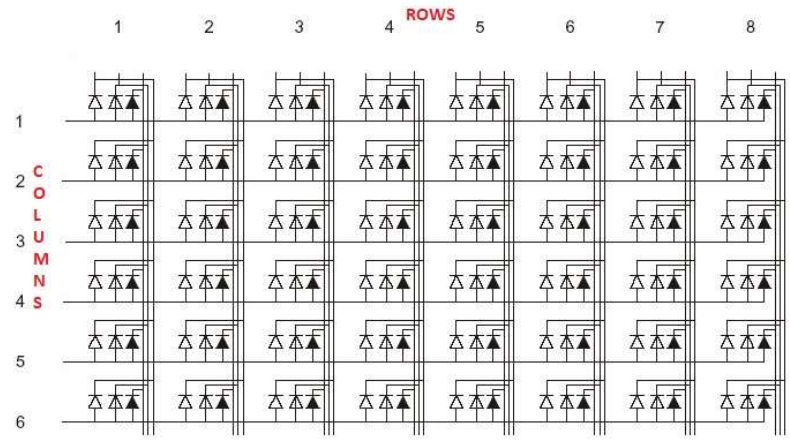
Interaktivní světelná tabule je tvořena pixely, každý pixel snímá objekty ve vzdálenosti 0–5 cm a dokáže zobrazit libovolnou barvu (24bitovou barevnou hloubkou). Samotná aplikace, která řídí chování tabule, může být spuštěna přímo na mikroprocesoru nebo na počítači. V režimu s počítačem si panel můžete představit jako kombinaci klávesnice a monitoru.



Obrázek 3: Principiální blokové schéma.

2.1 Zobrazování

Pro zobrazování jsem potřeboval nějaké vysoce svítivé LED s velkým vyzařovacím úhlem, aby průsvitné plexisklo mohlo být co nejblíže a bylo rovnoměrně osvětlené. A také bylo zapotřebí vyřešit jak takové množství (600 LED) řídit[1].

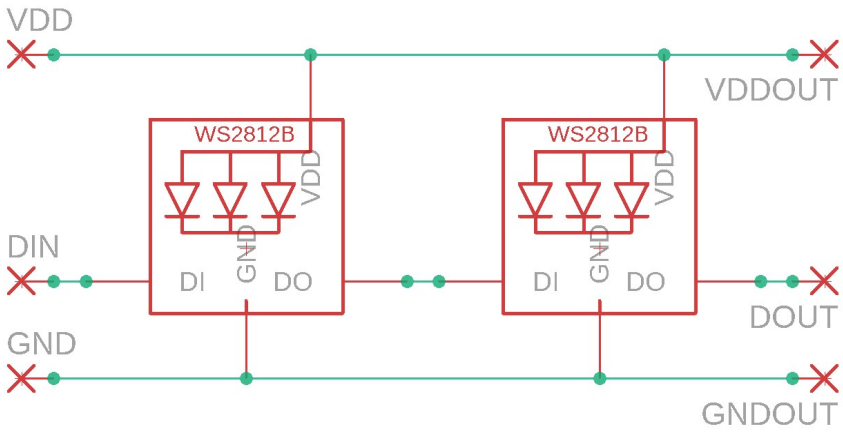


Obrázek 4: Ukázka zapojení RGB matice 8x8. [1]

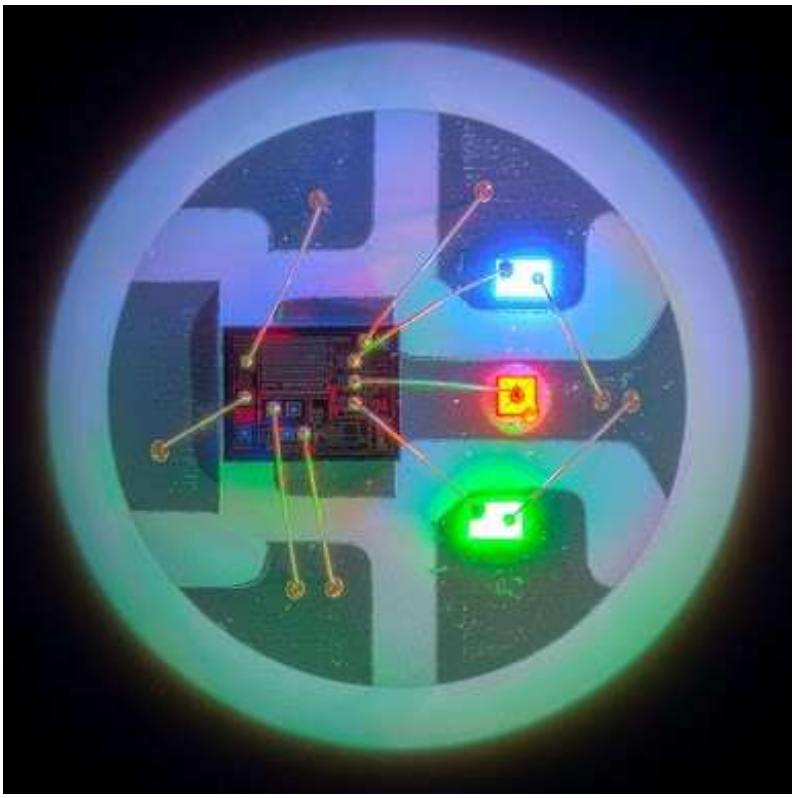
[1] Jednou z testovaných možností bylo zapojení do matice – viz Obrázek 4. Například pro matice o velikosti 8 x 8 RGB LED bych musel použít 32 vodičů. Tento problém je možné vyřešit pomocí posuvných registrů. Použití posuvných registrů sníží pouze počet výstupních pinů z MCU, ale i přesto by se musely všechny LED propojit vodiči. Proto jsem použil adresovatelné LED.

2.1.1 Použité LED

Pro zobrazování jsem použil adresovatelné LED WS2812B. Tyto LED mají čtyři vstupní vodiče, dva pro napájení 5 V (GND, VDD) a dva datové vstupní a výstupní (DI, DO). Pro řízení libovolného počtu LED nám stačí propojit napájecí vodiče všech LED na napájení a datové vodiče vždy tak, že připojíme výstup jedné LED na vstup následující – viz Obrázek 5. Díky tomu by teoreticky stačil pouze jeden datový vodič z MCU pro celou tabuli. WS2812B obsahují řídicí MCU, které zpracovává data a řídí jednotlivé barevné kanály – viz Obrázek 6.



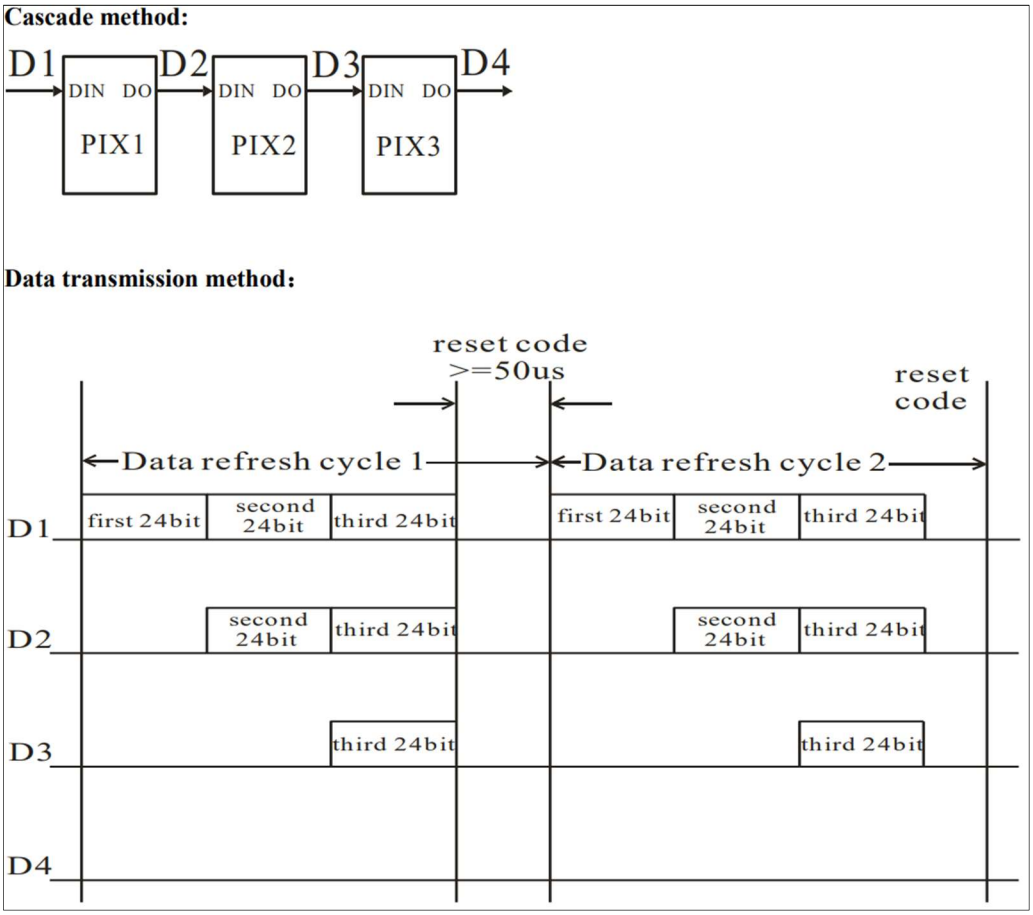
Obrázek 5: Ukázka propojení dvou pixelů. Vytvořeno v programu Eagle.



Obrázek 6: Detailní pohled na WS2812B-B. [2]

2.1.2 Odesílání dat

Data do WS2812B se posílají sériově, každá LED zpracuje 24 bitů a zbytek dat předá do další LED – viz Obrázek 7. Nejprve se odešle reset bit, ten trvá 50 µs, poté data pro všechny LED, protože odeslání jednoho bitu trvá 1,25 µs – viz Obrázek 8. Odeslání dat pro celou tabuli by trvalo cca 6 ms, a protože jsem původně plánoval tyto LED po dobu snímání dat zhasínat, rozdělil jsem tabuli na 5 skupin po 40 LED a zkrátil dobu odesílaní na 1,2 ms (to ale nebyl jediný důvod rozdělení, další důvod popíši v kapitole 2.1.4 Tvorba vlastní knihovny).



Obrázek 7: Způsob, jakým si LED posílají data. [3]

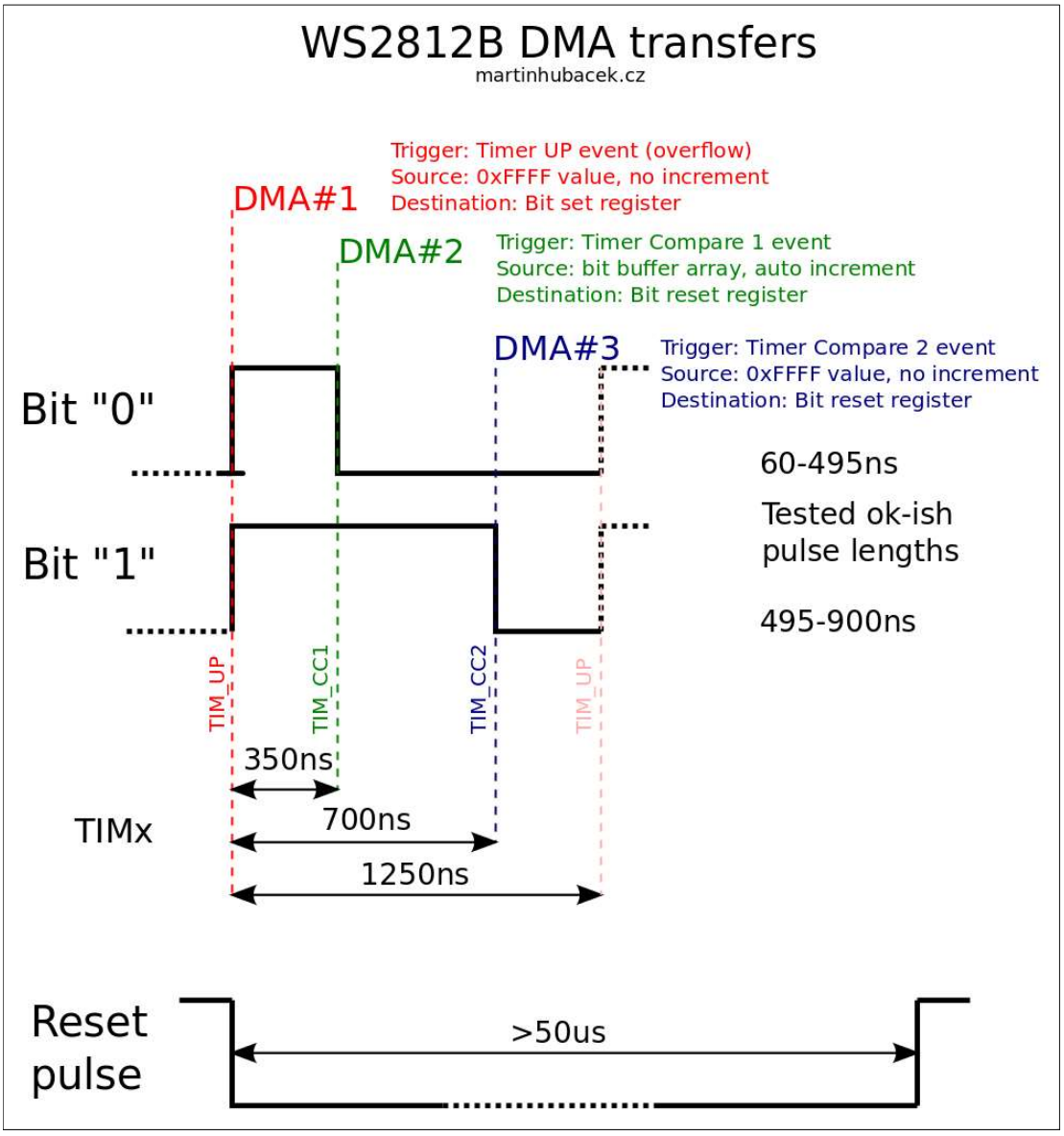
Data transfer time(TH+TL=1.25µs±600ns)

T0H	0 code ,high voltage time	0.4us	±150ns
T1H	1 code ,high voltage time	0.8us	±150ns
T0L	0 code , low voltage time	0.85us	±150ns
T1L	1 code ,low voltage time	0.45us	±150ns
RES	low voltage time	Above 50µs	

Obrázek 8: Tabulka trvání jednotlivých impulsů. [3]

2.1.3 DMA

Jak jsem psal v kapitole 2.1.2 Odesílání dat, odeslání dat trvá 1,2 ms. Abych ušetřil procesorový čas, použil jsem pro řízení WS2812B DMA (Direct Memory Access, tj. přímý přístup do paměti).[2] DMA jsem použil v režimu paměť – periferie. První kanál DMA nastavuje výstupní piny na LOG1, druhý kanál nastavuje výstupní pin na LOG0, ale to jen v případě, že chceme přenést Bit „0“, a třetí kanál nastavuje výstupní pin na LOG0 vždy – viz Obrázek 7.



Obrázek 9: Princip řízení DMA. [4]

[2] „Přímý přístup k paměti (DMA) se používá k zajištění vysokorychlostního přenosu dat mezi periferiemi a pamětí, nebo při přesunu dat paměť – paměť. Data lze rychle přesunout pomocí DMA bez akcí CPU. Tím se zachovávají prostředky CPU pro další operace.“ (přeloženo) [9]

2.1.4 Tvorba vlastní knihovny

Při tvorbě knihovny jsem vycházel z knihovny NeoMaple [5]. Z této knihovny jsem použil část pro řízení HW, kterou jsem upravil tak, aby uměla obsluhovat více výstupních pinů a nezabírala celý port, ale jen požadované piny. Dále jsem si vytvořil vlastní rozhraní knihovny.

Knihovna využívá tři kanály z DMA1 (CH2, CH5 a CH7), tyto kanály spouští časovač TIM2, v původní knihovně se zapisovaly jednotlivé kanály do registru ODR (Output Data Register), ale protože DMA umí přepsat vždy jen celý registr, nešlo by použít zbytek výstupního portu (16 pinů) na nic jiného než řízení LED. Použil jsem pro řízení portu registry BSRR (Bit Set Reset Register) a BRR (Bit Reset Register) - viz Obrázek 10.

```
void DMA_init(void)
{
  dma_init(DMA1);
  // TIM2 Update event
  /* DMA1 Channel2 configuration ----------------------------------------------*/
  dma_setup_transfer(DMA1, DMA_CH2, (volatile void *)&(GPIOB->regs->BSRR), DMA_SIZE_32BITS,
                     (volatile void *)&(WS2812_IO_Mask), DMA_SIZE_32BITS, DMA_FROM_MEM);
  dma_set_priority(DMA1, DMA_CH2, DMA_PRIORITY_HIGH);

  // TIM2 CC1 event
  /* DMA1 Channel5 configuration ----------------------------------------------*/
  dma_setup_transfer(DMA1, DMA_CH5, (volatile void *)&(GPIOB->regs->BRR), DMA_SIZE_16BITS,
                     (volatile void *)WS2812_buffer, DMA_SIZE_16BITS, DMA_FROM_MEM |
DMA_MINC_MODE);
  dma_set_priority(DMA1, DMA_CH5, DMA_PRIORITY_HIGH);

  // TIM2 CC2 event
  /* DMA1 Channel7 configuration ----------------------------------------------*/
  dma_setup_transfer(DMA1, DMA_CH7, (volatile void *)&(GPIOB->regs->BRR), DMA_SIZE_16BITS,
                     (volatile void *)&(WS2812_IO_Mask), DMA_SIZE_16BITS, DMA_FROM_MEM |
DMA_TRNS_CMPLT);
  dma_set_priority(DMA1, DMA_CH7, DMA_PRIORITY_HIGH);

  /* configure DMA1 Channel7 interrupt */
  nvic_irq_set_priority(NVIC_DMA_CH7, 1);
  nvic_irq_enable(NVIC_DMA_CH7);
  dma_attach_interrupt(DMA1, DMA_CH7, DMA1_Channel7_IRQHandler);
  /* enable DMA1 Channel7 transfer complete interrupt */
}
```

Obrázek 10: Ukázka zdrojového kódu, který nastavuje DMA. Vytvořeno v programu Visual Studio Code (rozšíření PIO).

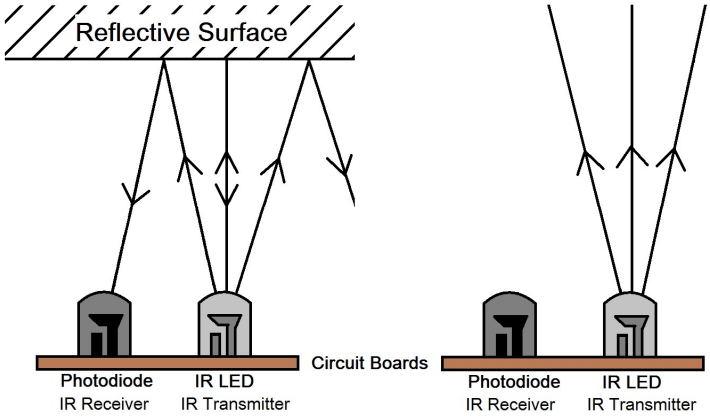
DMA CH5 pracuje v režimu DMA_MINC_MODE (Auto-increment memory address) a data přenáší z bufferu (WS2812_buffer) viz Obrázek 10. V tomto bufferu jsou připravena data k odeslání, DMA tento buffer prochází po celých bajtech a vstupní registr má velikost 2 bajty. Znamená to, že pro každý odeslaný bit je potřeba 2 bajty paměti, tj. 1200 bajtů paměti RAM pro 200 LED. Díky tomu, že DMA dokáže zapsat bity pro celý port v jednom kroku, mohl jsem rozdělit LED do skupin (5 skupin po 40) a zmenšit tak velikost použité paměti na pětinu. To vyžadovalo složitý zapisovací algoritmus – viz Obrázek 11.

```
// Set pixel color
void MyNeo::setPixelColor(
    uint16_t n, uint8_t r, uint8_t g, uint8_t b)
{
  //zjištěni indexu výstupního pinu
  uint8_t pinIndex = 0;
  while (n >= 40)
  {
    n -= 40;
    pinIndex++;
  }
  //zjištěni offsetu
  uint8_t pinOffset = (PIN_MAP[outPins[pinIndex]].gpio_bit);
  if (n < 40)
  {
    uint8_t i;
    for (i = 0; i < 8; i++) //pro všechny bity
    {
      uint32_t offset = (n * 24) + i;
      if ((g & (0x80 >> i)) == 0) //green
      {
        pixels[offset] |= (1 << pinOffset); //set to one
      }
      else
      {
        pixels[offset] &= ~(1 << pinOffset); //reset to zero
      }
      if ((r & (0x80 >> i)) == 0) //red
      {
        pixels[offset + 8] |= (1 << pinOffset); //set to one
      }
      else
      {
        pixels[offset + 8] &= ~(1 << pinOffset); //reset to zero
      }
      if ((b & (0x80 >> i)) == 0) //blue
      {
        pixels[offset + 16] |= (1 << pinOffset); //set to one
      }
      else
      {
        pixels[offset + 16] &= ~(1 << pinOffset); //reset to zero
      }
    }
  }
}
```

Obrázek 11: Ukázka zdrojového kódu „parser“. Vytvořeno v programu Visual Studio Code (rozšíření PIO).

2.2 Snímání

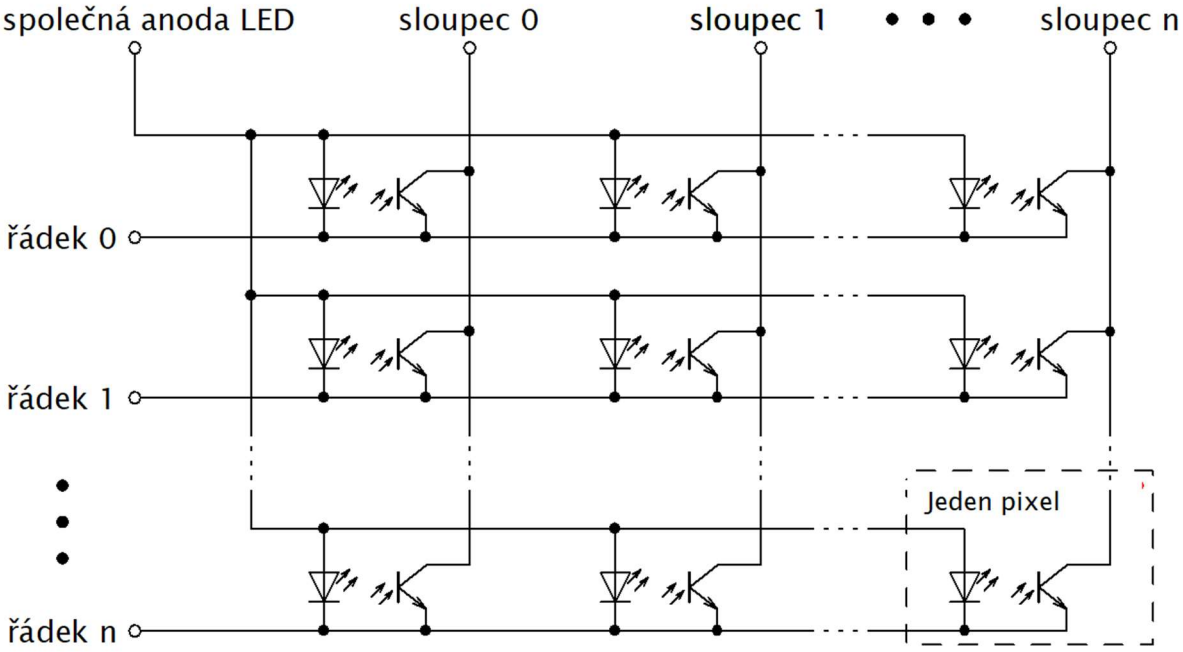
O snímání se stará infračervený vysílač (IR LED) a infračervený snímač (IR phototransistor), které pracují mimo viditelné spektrum. Pokud je před pixelem překážka, infračervený paprsek z vysílače se odrazí zpět do přijímače, jak můžete vidět na následujícím obrázku.



Obrázek 12: Princip snímání objektů. [6]

2.2.1 Zapojení

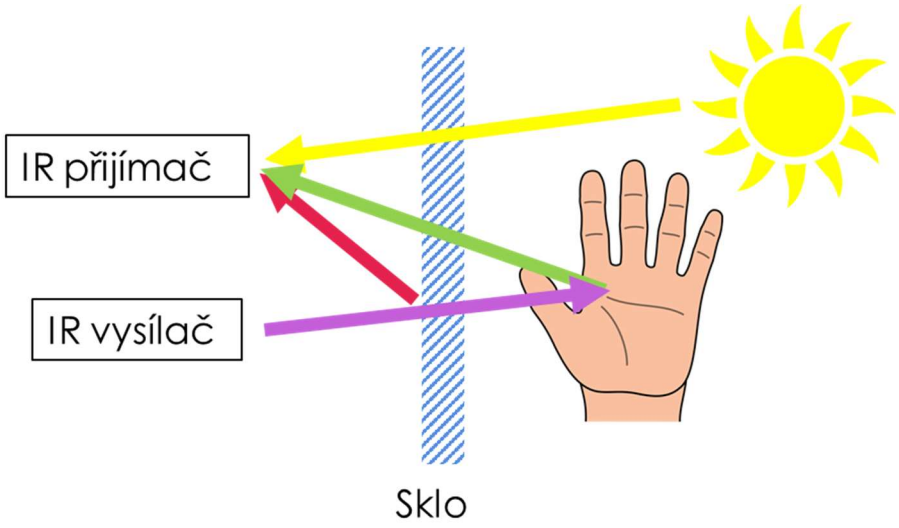
Kvůli úspoře vodičů, analogových obvodů a pinů MCU jsou IR LED a IR fototranzistory zapojeny do matice – viz Obrázek 13. Na aktivní řádek je přivedeno záporné napětí, pomocí společné anody LED mohu řídit svit vysílačů (IR LED), více v kapitole 2.2.2 Odfiltrování IR pozadí. Proud tekoucí do sloupce je přímo úměrný množství přijatého světla v aktivním řádku.



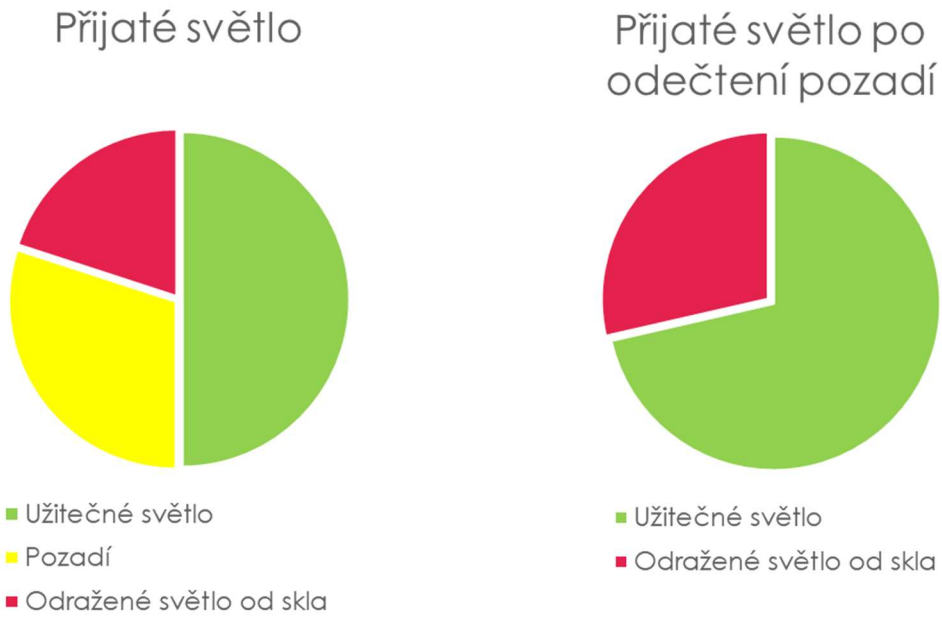
Obrázek 13: Schéma zapojení matice IR LED a IR fototranzistorů.

2.2.2 Odfiltrování IR pozadí

Intenzita přijatého světla (dále jen „světla“) se skládá ze tří složek – viz Obrázek 14, z odraženého světla od skla (červeně), pozadí (žlutě) a užitečného světla (zeleně). Když chceme zjistit množství užitečného světla, musíme znát druhé dvě složky. Množství odraženého světla jednoduše odečteme, protože je to konstanta. Pozadí zjistíme tak, že vypneme IR vysílač a změříme pozadí. Když následně zapneme IR vysílač, množství přijatého světla naroste o užitečnou a odraženou složku.



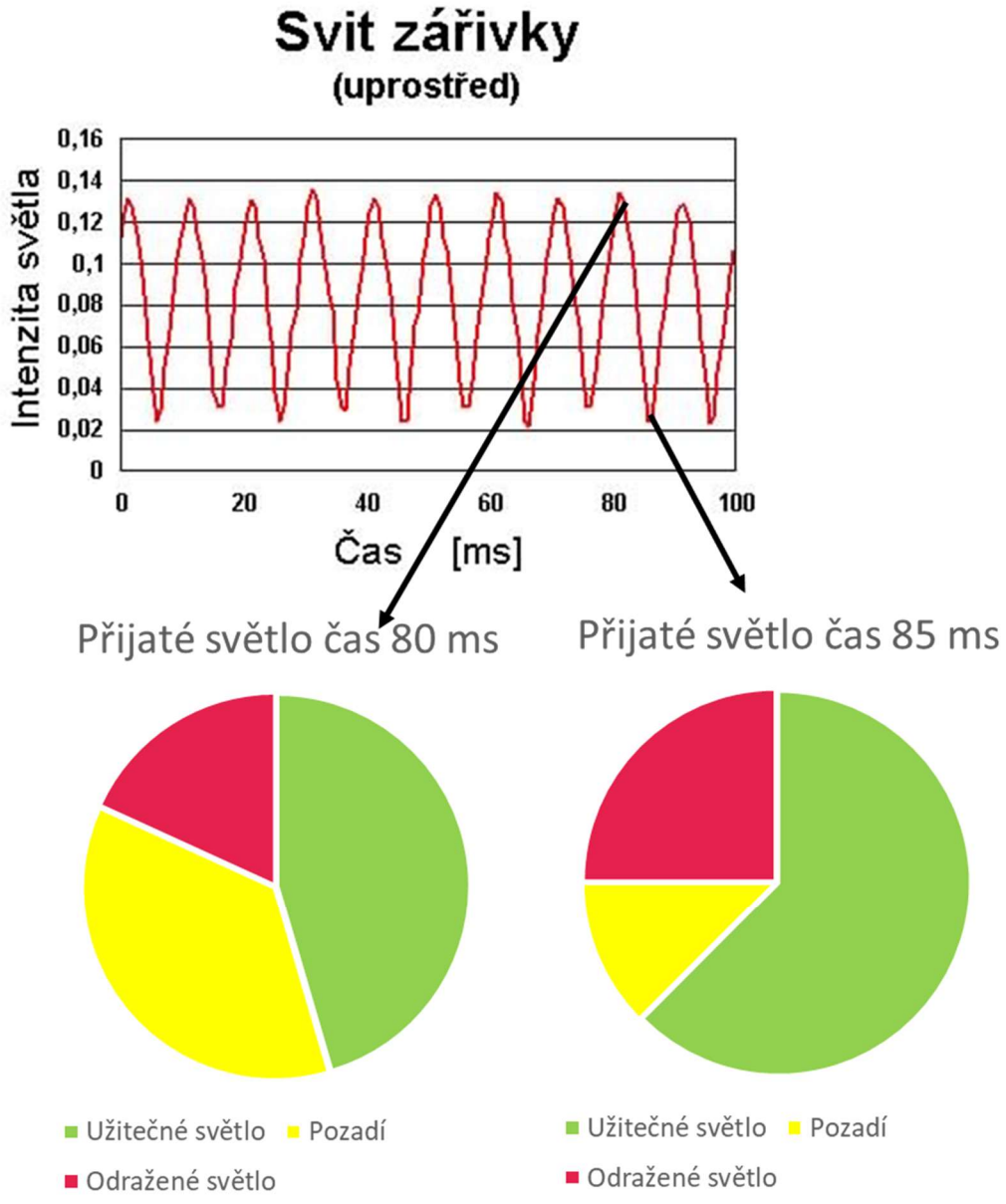
Obrázek 14: Schematický obrázek jednotlivých složek.



Obrázek 15: Grafické zobrazení zastoupení jednotlivých složek.

2.2.3 Odstranění stroboskopického jevu

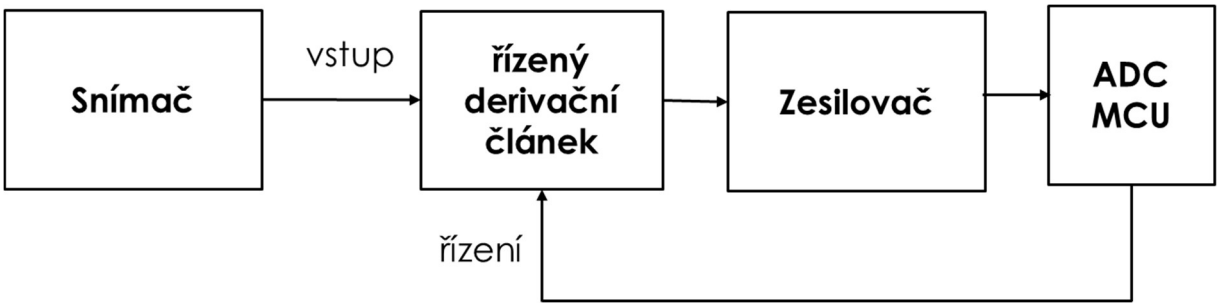
Protože se pozadí v čase velmi rychle mění, například v blízkosti zářivek, které blikají s frekvencí 100 Hz, vznikala při odečtení pozadí chyba způsobená časovou prodlevou mezi měřením bez a se zapnutým vysílačem. Na obrázku níže je znázorněno v grafech zastoupení jednotlivých složek ve dvou různých časech. Poměr intenzity odraženého a užitečného světla zůstává stejný, mění se pouze pozadí.



Obrázek 16: Zastoupení složek přijatého světla v časech 80 ms a 85 ms.

2.2.4 Princip vyhodnocení dat

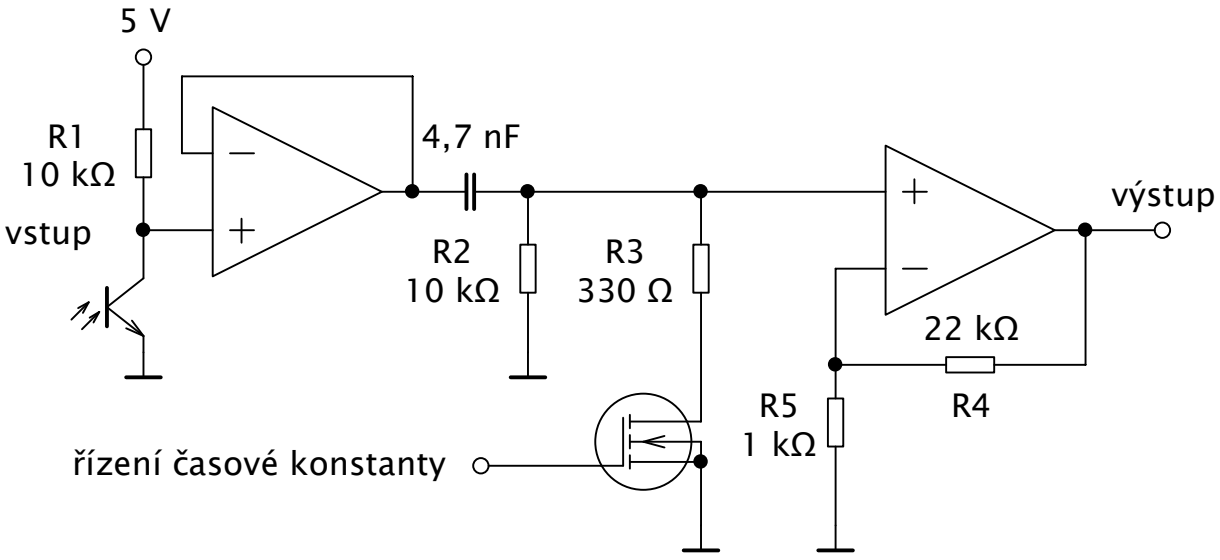
Proud do sloupce se měří pomocí bočníku. Úbytek napětí bočníku je přiveden do vyhodnocovacího obvodu (analogový obvod s operačními zesilovači). Úkolem tohoto obvodu je signál impedančně přizpůsobit, vyhodnotit a odstranit jeho stejnosměrnou složku. Následně signál putuje přes zesilovač do ADC (Analog Digital Conventer), který je součástí MCU. Z MCU mohu zároveň řídit (přepínat) časovou konstantu derivačního článku, který je součástí vyhodnocovacího obvodu. Viz blokové schéma níže.



Obrázek 17: Blokové schéma vyhodnocení dat.

2.2.5 Zapojení vyhodnocovacího obvodu

Vyhodnocovací obvod je tvořen dvěma operačními zesilovači. Oba jsou zapojeny v neinvertujícím zapojení, první je zapojen jako sledovač napětí, druhý jako zesilovač. Na vstup sledovače je přivedeno napětí, které je tvořeno pomocí bočníku a je závislé na intenzitě světla. Za sledovačem je derivační článek s přepínatelnou časovou konstantou. O změnu časové konstanty se stará N-Mosfet. Výstup z derivačního článku je přiveden na výstupní zesilovač.



Obrázek 18: Schéma zapojení vyhodnocovacího obvodu.

Výpočet 1. časové konstanty (otevřený mosfet):

$$\tau_1 = R_2 * C\ [s]$$

$$\tau_1 = 10 * 10^3 * 4{,}7 * 10^{-9} = 47\ us$$

Výpočet 2. časové konstanty (sepnutý mosfet):

$$\tau_2 = \frac{R_2 * R_3}{R_2 + R_3} * C\ [s]$$

$$\tau_2 = \frac{10000 * 330}{10000 + 330} * 4{,}7 * 10^{-9} = 1{,}5\ \mu s$$

Výpočet zesílení výstupního zesilovače:

$$A_u = 1 + \frac{R_4}{R_5}[-]$$

$$A_u = 1 + \frac{22}{1} = 23$$

2.2.6 Knihovna myAdc

Protože jsem nenašel vyhovující knihovnu pro rychlou obsluhu ADC, napsal jsem si vlastní. Tato knihovna využívá oba ADC převodníky. Při zavolání funkce „read()“ se provede převod pro všech 10 analogových vstupů. Jelikož knihovna používá krátký vzorkovací čas, je potřeba zajistit nízký vstupní odpor do ADC. Více v „How to get the best ADC accuracy in STM32 microcontrollers“. [7] Knihovna se skládá ze dvou souborů níže.

```
#ifndef myAdc_h
#define myAdc_h

#include <arduino.h>
#include <libmaple/adc.h>

class myAdc
{
  private:
    //ukazatele na registry
    adc_reg_map *adc1Regs = ADC1->regs;
    adc_reg_map *adc2Regs = ADC2->regs;

  public:
    myAdc(); //konstruktor
    void read();
    int values[10]; //output values
};
#endif
```

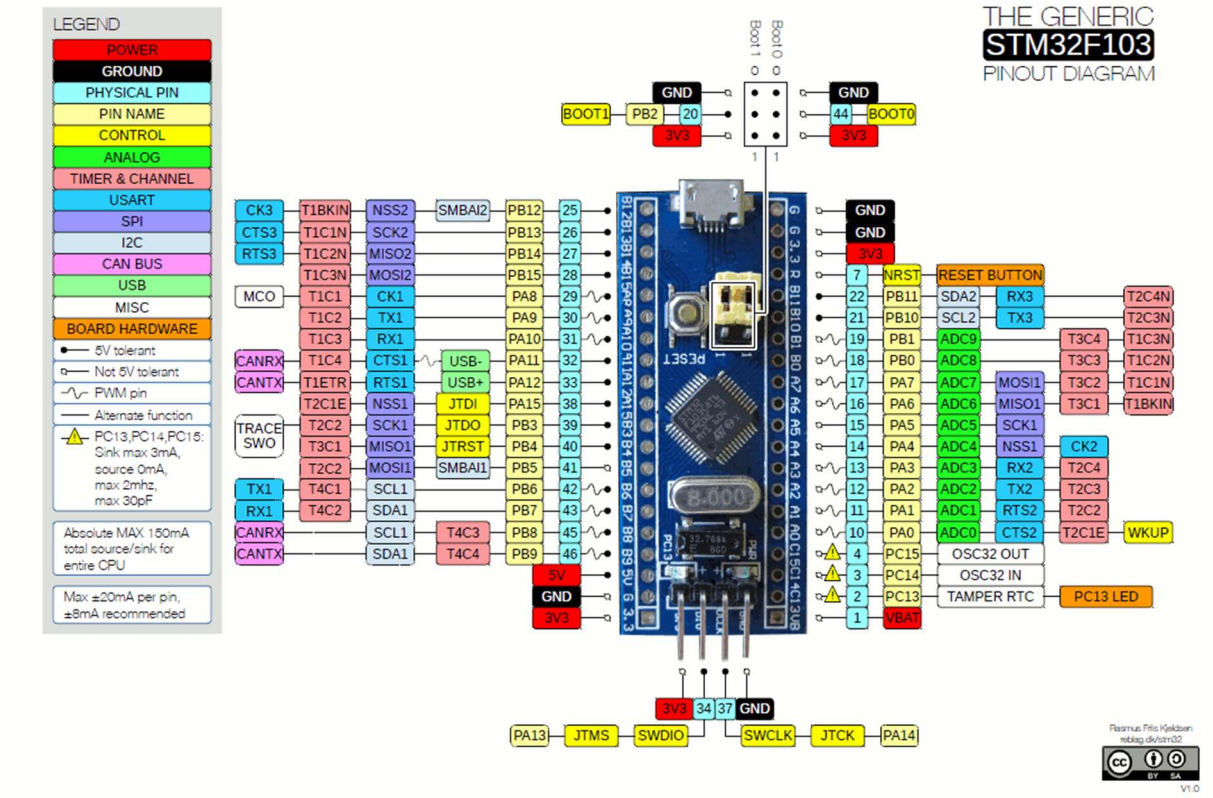
Obrázek 19: Zdrojový kód myAdc.h.

```
#include <arduino.h>
#include <libmaple/adc.h>
#include <myAdc.h>

myAdc::myAdc() //konstruktor
{
  //nastaveni doby vzorkovaní
  adc_set_sample_rate(ADC1, ADC_SMPR_1_5);
  adc_set_sample_rate(ADC2, ADC_SMPR_1_5);
  //nastaveni délky sekvence
  adc_set_reg_seqlen(ADC1, 1);
  adc_set_reg_seqlen(ADC2, 1);
}

void myAdc::read() //čtení
{

  for (byte i = 0; i < 10; i += 2)
  {
    adc1Regs->SQR3 = i;
    adc2Regs->SQR3 = i + 1;

    //začni převádětdet
    adc1Regs->CR2 |= ADC_CR2_SWSTART;
    adc2Regs->CR2 |= ADC_CR2_SWSTART;

    while (!((adc1Regs->SR & ADC_SR_EOC) &
    (adc2Regs->SR & ADC_SR_EOC))) //čeká, dokud není dokončen převod
      ;

    //uloží data
    values[i] = (adc1Regs->DR & ADC_DR_DATA);
    values[i + 1] = (adc2Regs->DR & ADC_DR_DATA);
  }
}
```

Obrázek 20: Zdrojový kód myAdc.cpp.

2.3 Řízení

Pro řízení jsem použil mikroprocesor STM32F103, který je dostatečně rychlý a má všechny potřebné periferie, 10 analogových vstupů, dva ADC s rozlišením 12bit a DMA. Pro snazší práci s tímto MCU jsem použil vývojovou desku „Blue Pill“ - viz obrázek níže.



Obrázek 21: Vývojová deska Blue Pill a rozložení pinů. [8]

2.3.1 Zapojení periferií

Na piny ADC0 až ADC9 jsou zapojeny výstupy z analogových obvodů. PB10 až PB15 využívá multiplexer pro přepínání řádků. Ten je schopen řídit až 32 řádků, proto je elektrické zapojení matice 20 řádků a 10 sloupců (fyzicky má tabule 10 řádků a 20 sloupců). Piny PB3 až PB7 využívá WS2812B. Pin PC13 se používá pro přepínání časových konstant derivačních článků. Ostatní piny jsou nevyužity (PC14, PC15, PB8, PB9 a PA8 až PA15).[3]

[3] Některé piny používám pro pomocná ovládací tlačítka nebo pro spouštění časové základny osciloskopu, případně logického analyzátoru; v budoucnu zde může být připojena další periferie, např. display, Wifi modul, ultrazvukový senzor vzdálenost, Bluetooth atd.

2.3.2 Algoritmus řízení

Po startu programu se nastaví všechny piny a inicializují se knihovny. Poté se nastaví přerušení a spustí se časovač, který snímá pomocí přerušení hodnoty ze všech pixelů nezávisle na chodu programu. Následně se změří referenční hodnoty pro všechny pixely. Nakonec program cyklicky porovnává referenční a naměřené hodnoty a podle toho ukládá do proměnné „tlacitka[10][20]“ LOG1 (stisknuto) nebo LOG0. Zároveň aktualizuje buffer pro DMA a obsluhuje spuštěnou aplikaci.

2.3.3 Cyklická přerušení

Cyklická přerušení spouští TIMER2, tato přerušení se spouští v časech 0 µs, 65 µs a 100 µs. Čítač časovače přeteče každých 400 µs. Obrázek níže ukazuje zdrojový kód konfigurace časovače.

```
void timerSetup(void)
{
  timer.pause();
  // Set up period
  timer.setPrescaleFactor(72); //1MHz//      1us
  timer.setOverflow(400);
  // Set up an interrupt on channel 1
  timer.setChannel1Mode(TIMER_OUTPUT_COMPARE);
  timer.setCompare(TIMER_CH1, 0);
  timer.attachCompare1Interrupt(handler1);

  timer.setChannel2Mode(TIMER_OUTPUT_COMPARE);
  timer.setCompare(TIMER_CH2, 65);
  timer.attachCompare2Interrupt(handler2);

  timer.setChannel3Mode(TIMER_OUTPUT_COMPARE);
  timer.setCompare(TIMER_CH3, 100);
  timer.attachCompare3Interrupt(handler3);

  // Refresh the timer's count, prescale, and overflow
  timer.refresh();
  // Start the timer counting
  timer.resume();
}
```

Obrázek 22: Zdrojový kód konfigurace časovače.

- **Cyklické přerušení v čase 0 µs**

Po přetečení časovače se vypnou vysílače v aktuálním řádku. Časová konstanta vyhodnocovacího obvodu se nastaví $\tau_1 = 47\ us$.

```
void handler1(void)
{
  gpio_write_bit(GPIOC, 13, LOW); //nesviti
}
```

Obrázek 23: zdrojový kód přerušení v čase 0 µs.

- **Cyklické přerušení v čase 65 µs**

Po uplynutí 65 µs je na výstupu vyhodnocovacího obvodu napětí, které je úměrné velikosti změny proudu do sloupce po vypnutí vysílače. Toto napětí změřím ADC. Výsledek se přičte do proměnné „values[row][i]“. Součástí přerušení je i počítadlo měření a počítadlo řádků. Proměnná měření slouží průměrování libovolného počtu výsledků a nulování proměnné „values[row][i]“ - viz zdrojový kód níže.

```
void handler2(void)
{
  gpio_write_bit(GPIOC, 14, HIGH); //debug
  ADC.read();
  gpio_write_bit(GPIOC, 14, LOW); //debug
  for (int i = 0; i < 10; i++)
  {
    if (mereni == 1)
    {
      values[row][i] = 0;
    }
    values[row][i] += ADC.values[i];
  }
  if (row == (pocetradku - 1))
  {
    row = 0;
    mereni++;
  }
  else
  {
    row++;
  }
}
```

Obrázek 24: Ukázka zdrojového kódu v čase 65 µs.

Ve zdrojovém kódu výše si můžete všimnout dvou řádek komentovaných „//debug“ - tyto řádky mi pomáhají pomocí osciloskopu učit dobu, kdy pracuje ADC. Více v kapitole **2.3.4 Časové průběhy signálů.**

- **Cyklické přerušení v čase 100 µs**

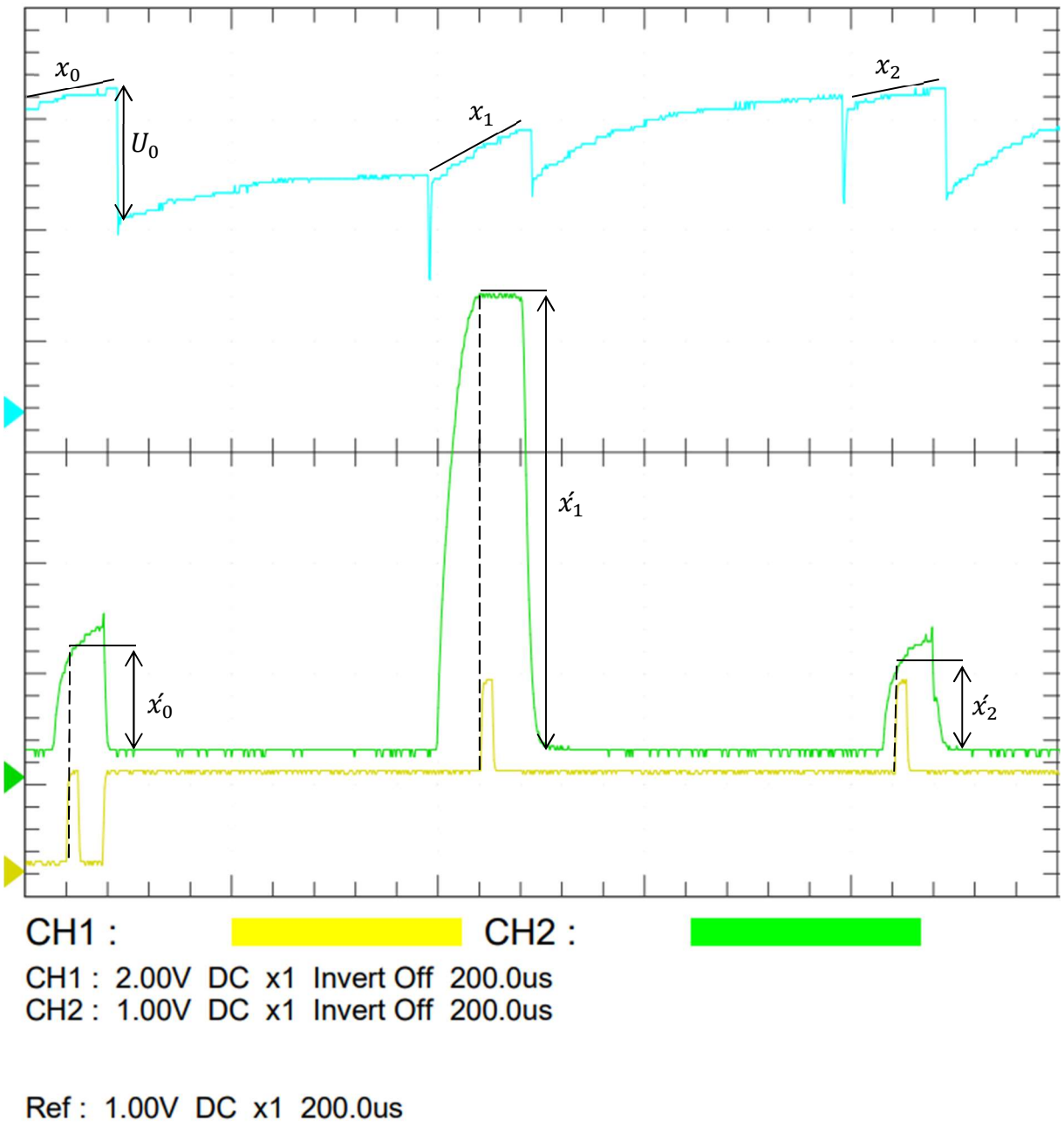
Toto přerušení se spouští po dokončení převodu ADC. Aktivuje další řádek, zapne vysílače a časovou konstantu nastaví na $\tau_2 = 1{,}5\,\mu s$ tak, aby se výstup vyhodnocovacího obvodu nastavil co nejrychleji na hodnotu 0. Po tomto přerušení trvá 300 µs, než přeteče čítač časovač a začne měření dalšího řádku (dokud se kondenzátor v derivačním článku nenabije na stejnosměrnou složku aktuálního řádku). Toto přerušení také obsahuje podmínku, která zastaví všechna přerušení pozastavením časovače, pokud se provedla všechna měření. Přerušení se neprovádějí, dokud je hlavní program znovu nezapne. Hlavní program následně vydělí hodnoty ze snímačů počtem měření a získá aritmetický průměr.

```
void handler3(void)
{
  //prepne radek
  GPIOB->regs->BSRR = (0xf800 << 16) | ((mux[row] << 11) & 0xf800);
  gpio_write_bit(GPIOC, 13, HIGH); //sviti
  if (mereni >= pocetmereni)
  {
    timer.pause();
    mereni = 0;
  }
}
```

Obrázek 25: Ukázka zdrojového kódu v čase 100 µs.

2.3.4 Časové průběhy signálů

Pro naměření časových průběhů jsem použil dvoukanálový osciloskop v režimu se zapnutou časovou základnou, protože se impulzy z jednotlivých pixelů opakují a časová základna by se spouštěla při měření náhodného pixelu. Připojil jsem na sondu kanálu CH1 pomocí dvou 10 kΩ rezistorů dva signály (signál pro aktivaci 0. řádku a signál detekující činnost DAC). Kanál CH1 jsem poté použil pro synchronizaci časové základny. Pomocí kanálu CH2 jsem nejprve změřil signál na vstupu vyhodnocovacího obvodu a uložil ho do paměti. Poté jsem znovu pomocí kanálu CH2 změřil signál na výstupu vyhodnocovacího obvodu. Tyto tři průběhy můžete vidět na obrázku – viz další strana.

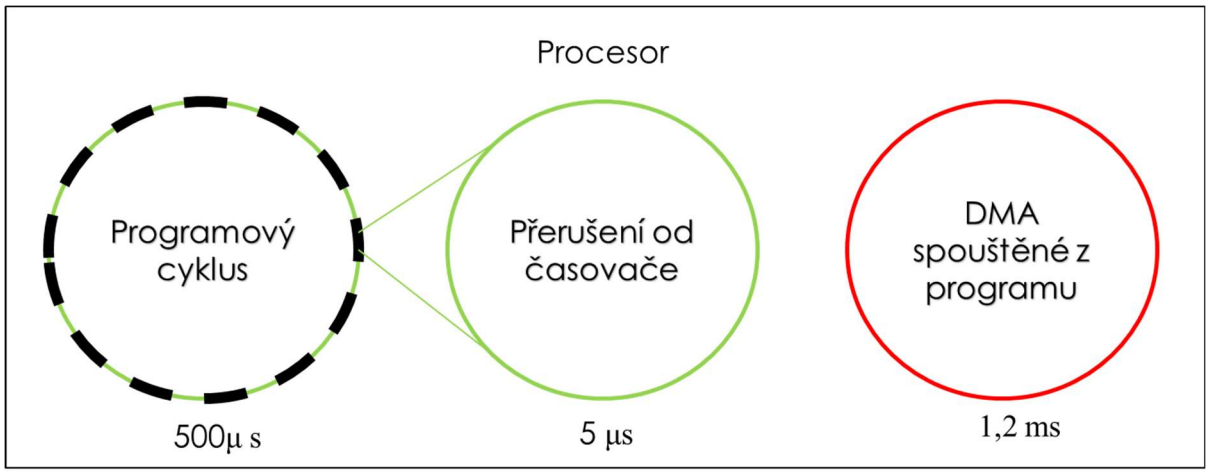


Obrázek 26: Obrázek časového průběhu. Kanál CH1 žlutě, kanál CH2 zeleně a průběh uložený v paměti modře. Pixel x_1 je zcela zakrytý a pixely x_0 a x_2 jsou odkyté.

Na obrázku můžete vidět průběh snímání z prvních třech pixelů 0. sloupce. Signál ze vstupu vyhodnocovacího obvodu je vyznačen modře. Výstupní signál je vyznačen zeleně, jeho amplituda v okamžiku měření odpovídá derivaci vstupního signálu, jinými slovy odpovídá velikosti změny vstupního signálu (při zhasnutí vysílače). Jak si můžete všimnout, výstupní signál nemá na rozdíl od vstupního žádný ofset. Také není ovlivněn změnou stejnosměrného ofsetu mezi pixely (v obrázku je vyznačen jako U_0).

Režie procesorového času

Jak jsem popsal v předešlých kapitolách, v MCU se provádějí tři nezávislé cykly. Hlavní program provádí v nekonečné smyčce aktualizaci tabule. Když jsou připravena data z měření, byl dokončen předchozí přenos a jsou k dispozici nová data k zobrazení, aktivuje DMA. Cyklická přerušení pro načítání se spouštějí na úkor hlavního programu, ale již zmíněné DMA funguje zcela samostatně. Viz obrázek níže.



Obrázek 27: V mikroprocesoru jsou spuštěny tři nezávislé cykly, časové údaje vespodu udávají dobu trvání jednoho cyklu.

3 HARDWARE

3.1 Moduly

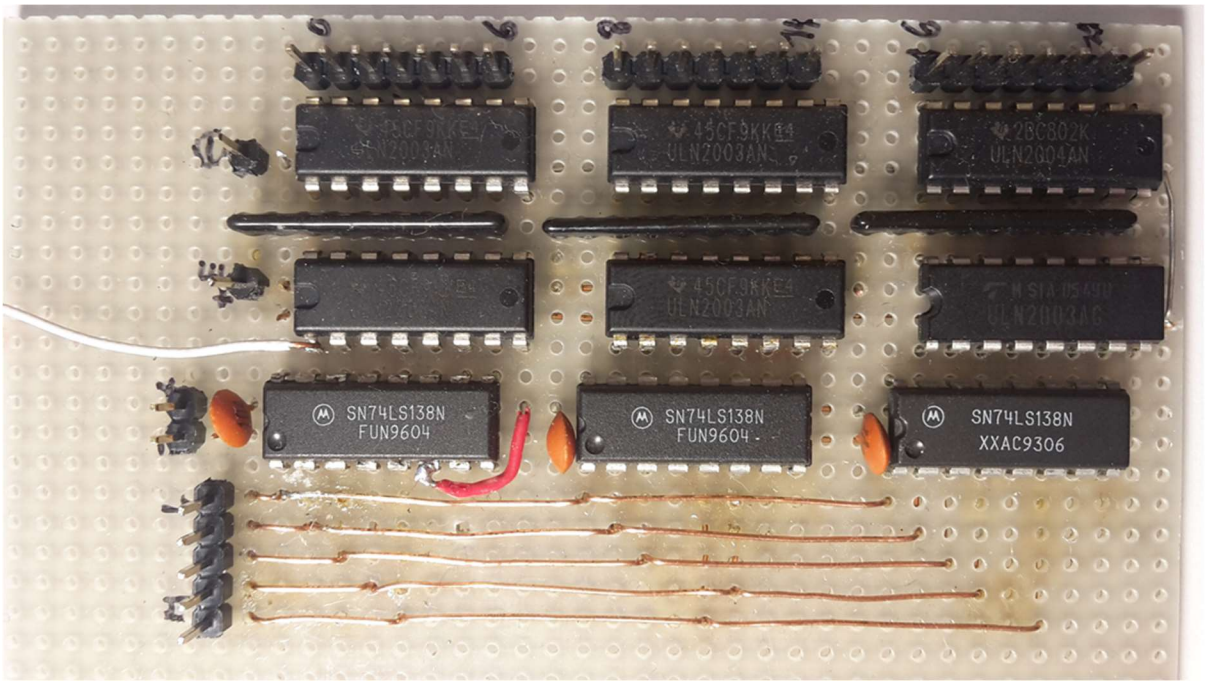
Modul je základní stavební jednotka tabule, obsahuje 4 pixely. Propojuje snímací i zobrazovací část s výstupními ploškami pro připájení vodičů. Jeden modul má rozměry 32 x 2 cm. Po snazší výrobu jsem modul rozdělil na dvě části a poskládal do pětic. Tím jsem získal foto předlohu o rozměrech 10 x 16 cm – viz Obrázek 38 v bodu 8 Příloha 2: Technické dokumenty. Použité cuprexitové desky s fotocitlivou vrstvou měly rozměry 16,5 x 10,5 cm, díky přesahům nebylo pozicování průsvitné folie tak důležité. Nerovnosti jsem srovnal při stříhání na pákových nůžkách. Jednotlivé moduly jsem odstřihával podle vodicích linek.

3.2 Řídicí jednotka

Řídící jednotka se skládá ze dvou částí. První část obsahuje vyhodnocovací obvody, pomocné napájení a vývojovou desku „Blue pill“[4] – viz obrázek níže. Druhá část obsahuje multiplexery a tranzistorové sítě pro přepínání řádků – viz Obrázek 29.

Obrázek 28: Prototyp řídicí jednotky s vyhodnocovacími obvody.

[4] Ve fázi vývoje často měním zapojení, proto je na univerzálním DPS vynecháno místo pro vývojovou desku „Blue pill“, kterou mám zapojenou do nepájivého kontaktního pole pomocí kolíkové lišty. V příloze na obrázkuObrázek 40 můžete vidět předlohu pro výrobu řídící jednoty s implementovanou deskou Blue pill.



Obrázek 29: Prototyp modulu pro přepínání řádků.

3.3 Napájení

Každá zobrazovací LED odebírá proud přibližně 50 mA, pro celou tabuli je to 10 A. Proto jsem k napájení použil modulový spínaný napájecí zdroj 14 A @ 4,5 až 5,5 V výrobní označení LRS-75-5. Tento síťový zdroj je určen pro vestavbu, ale musí být zajištěn odvod tepla a také vhodné jištění.

3.4 Mechanické provedení

Tělo interaktivní světelné tabule je zhotoveno ze staré kolečkové tabule. Na přední straně jsou připevněny moduly s pixely, nad nimi je mřížka z polystyrenu[5] pro vnitřní zateplení (tloušťka 7 mm). Přes polystyrénovou mřížku na pření straně je krycí průsvitné plexisklo. Na zadní straně je tabule dutá. Do této dutiny jsem umístil řídicí jednotku a napájení.

[5] Mřížka slouží ke směrování světla. Tuto mřížku jsem vyrobil pomocí řezačky polystyrenu. Nákres šablony a foto řezačky naleznete v přílohách.

4 SOFTWARE

4.1 Ukázková aplikace Malování

Při spuštění aplikace malování v prvním sloupci tabule vykreslí paletu barev – viz následující obrázek.

```
void setupMalovani()
{
  for (int i = 0; i < pocetradku; i++)
  {
    barva(i);
    LED.setPixelColor((i * pocetsloupcu), cervena, zelena, modra);
  }
}

void barva(int a)
{
  switch (a)
  {
  case 0: //červená
  {
    zelena = 255;
    cervena = 0;
    modra = 0;
    break;
  }
  case 1: //oranžová
  {
    zelena = 255;
    cervena = 255;
    modra = 0;
    break;
  }

.
. //další barvy
.

}
```

Obrázek 30: Ukázka kódu (jsou zobrazeny jen první dvě barvy).

Z této palety si můžete vybrat barvu pro kreslení nebo dlouhým stiskem (déle než 2 s) překreslit celou tabuli na vybranou barvu. Když máte vybranou barvu, může zakrytím libovolného pixelu změnit jeho barvu na barvu „štětce“. O tyto záležitosti se stará nekonečný cyklus – viz obrázek na další straně.

```
void loopMalovani()
{
  for (int i = 0; i < pocetradku; i++)
  { // cte radky
    for (int j = 1; j < pocetsloupcu; j++)
    { // cte sloupce
      if (tlacitka[i][j] == 1)
      {
        LED.setPixelColor((i * pocetsloupcu + j), cervena, zelena, modra);
      }
    }
  }
  for (int i = 0; i < pocetradku; i++)
  {
    if (tlacitka[i][0] == 1)
    {
      if (x[i] == 0)
      {
        cas = millis();
        x[i] = 1;
      }
      if (millis() - cas > 2000)
      {
        clearMalovani();
      }
      if (millis() - cas > 100)
      {
        barva(i);
      }
    }
    else
    {
      x[i] = 0;
    }
  }
}

void clearMalovani()
{
  for (int i = 0; i < pocetradku; i++)
  { // cte radky
    for (int j = 1; j < pocetsloupcu; j++)
    { // cte sloupce
      LED.setPixelColor((i * pocetsloupcu + j), cervena, zelena, modra);
    }
  }
}
```

Obrázek 31: Zdrojový kód nekonečného cyklu programu malování.

4.2 Další aplikace

Program malování není jediná aplikace, kterou lze pro takovou tabuli naprogramovat. Možností je neomezeně. Hry jako je tetris, pong, snake, formule, hru pro brankáře, postřehové hry, color puzzle, piškvorky, 2D Rubikova kostka a spousta dalších...

5 ZÁVĚR

Tabule je plně funkční, ale v tuto chvíli bohužel ještě není plně dokončena. Nachází se ještě ve fázi prototypu. To znamená, že postupně nahrazuji funkční a otestované prototypy jejich finálními verzemi.

Tabule funguje spolehlivě, nevadí jí ani přímé světlo ze zářivek. Na přímém slunci má tabule problémy, protože se fototranzistory dostávají do saturace. Hlavním problémem ale zůstává to, že na přímém slunci je tabule špatně čitelná. Proto je použití omezeno na vnitřní použití v prostorách s umělým osvětlením.

Do budoucna bych rád pokračoval na vylepšování softwarové stánky. Také bych chtěl celou konstrukci přehodnotit, poučit se ze zjištěných nedostatků, využít nových znalostí a technologií. Například 3D tisku, o kterém jsem před třemi roky neměl ani tušení (teď mám doma 2 tiskárny), nebo nových kapacitních snímačů komunikujících přes sběrnici I2C. Mohl bych si na multi-materiálové 3D tiskárně vytisknout průhledné krytky a pomocí vodivého filamentu v nich vytvořit mřížku, na kterou bych připojil již zmíněné kapacitní snímače.

V průběhu stavby jsem našel na internetu i velmi podobné nápady, většina jich vnikla v době, kdy jsem pracoval na vlastním řešení. Mrzí mě, že nejsem první, ale jsem rád, protože žádný z těchto nápadů nepoužívá stejný princip snímání jako ten, který jsem vymyslel. Lidem se má interaktivní tabule líbí a mám i dva zájemce, kteří jsou za interaktivní světelnou tabuli ochotni zaplatit, ale pro mě je to hlavně koníček, ve kterém budu pokračovat i při studiu na vysoké škole.

POUŽITÁ LITERATURA

[1] „DIY RGB LED Matrix,“ 11 Leden 2018. [Online]. Available: https://www.electronicshub.org/diy-rgb-led-matrix/.

[2] o. a. N. a. s. l. C. BY-SA-NC. [Online]. Available: https://www.thitiblog.com/wp-content/uploads/2015/06/image.jpg.

[3] WorldSemi, „ DIGITAL RGB LED,“ [Online]. Available: http://www.world-semi.com/solution/list-4-1.html#117.

[4] M. Hubáček, „Improved STM32 WS2812B DMA library,“ [Online]. Available: http://www.martinhubacek.cz.

[5] E. Ritterbusch, „NeoMaple,“ 2014. [Online]. Available: https://github.com/fergul/NeoMaple/blob/master/neomaple_hardware.c.

[6] P. embedded, „Detecting obstacle with IR Sensor and Arduino,“ [Online]. Available: https://www.playembedded.org.

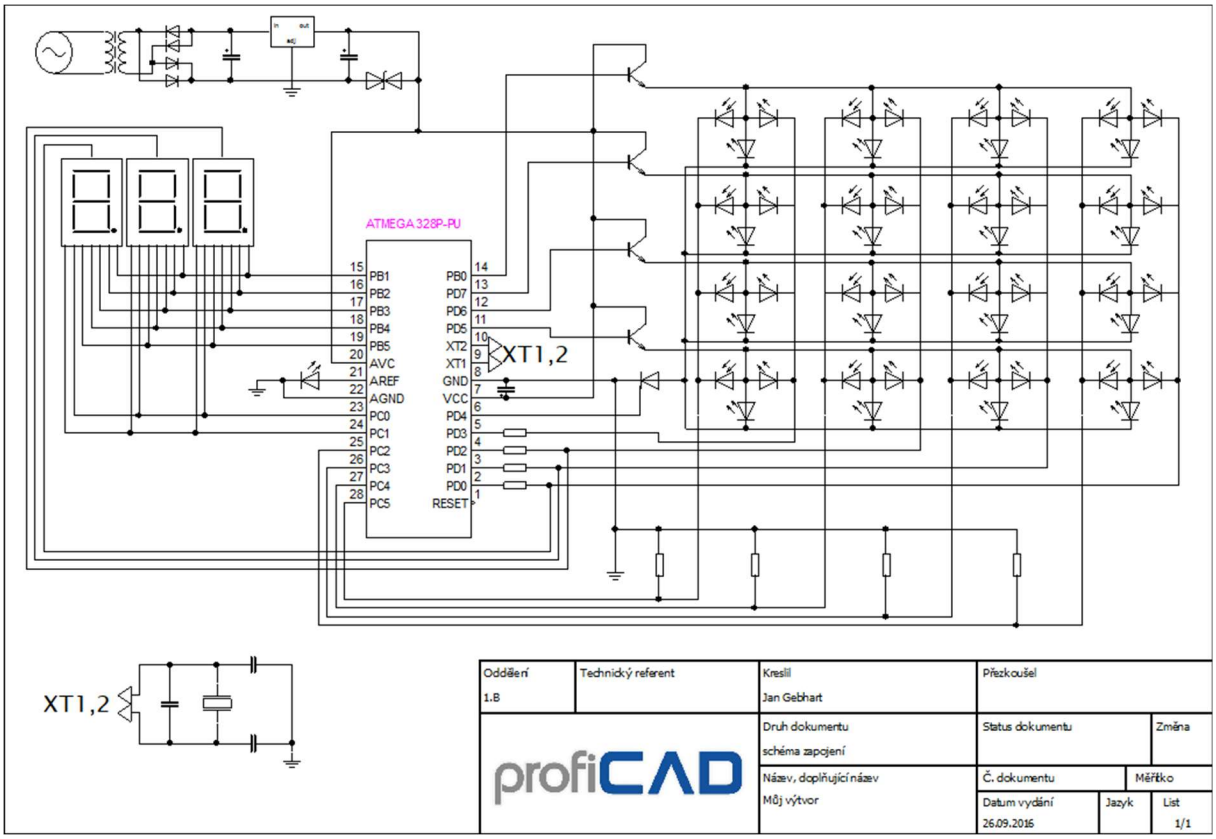
[7] ST, „AN2834,“ [Online]. Available: https://www.st.com/.

[8] stm32duino, „File:Bluepillpinout.gif,“ [Online]. Available: https://wiki.stm32duino.com.

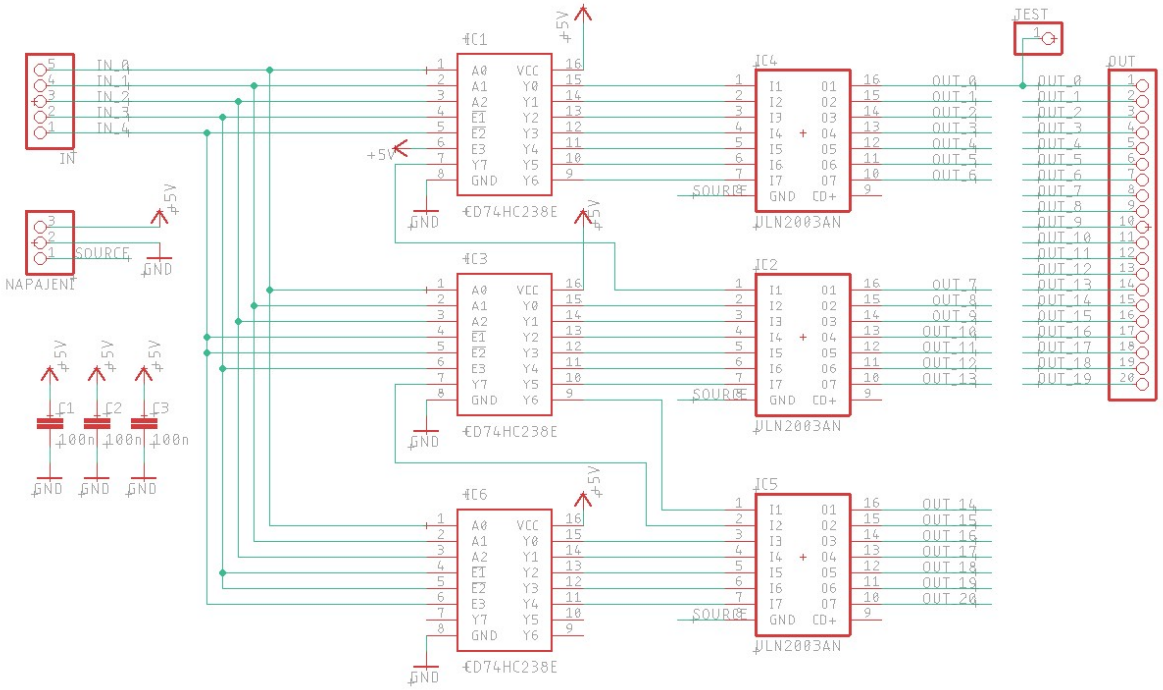
[9] ST, „RM0008,“ [Online]. Available: www.st.com.

6 SEZNAM OBRÁZKŮ

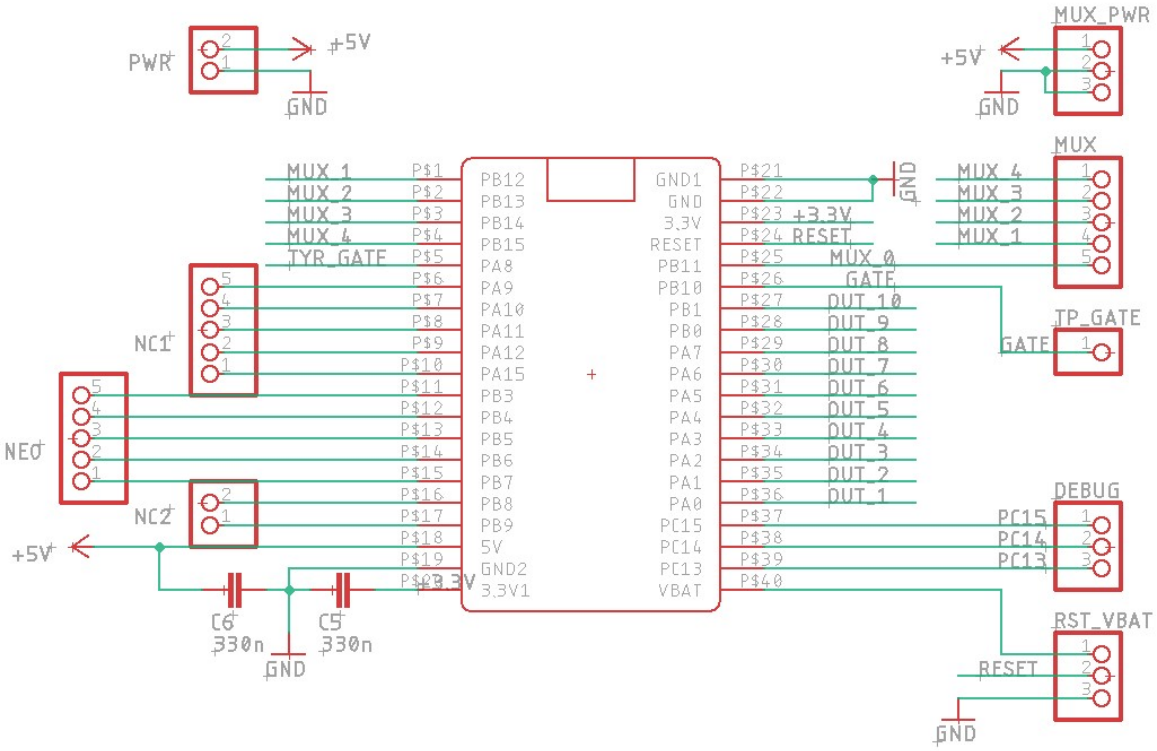
7 PŘÍLOHA 1: SCHÉMA ZAPOJENÍ



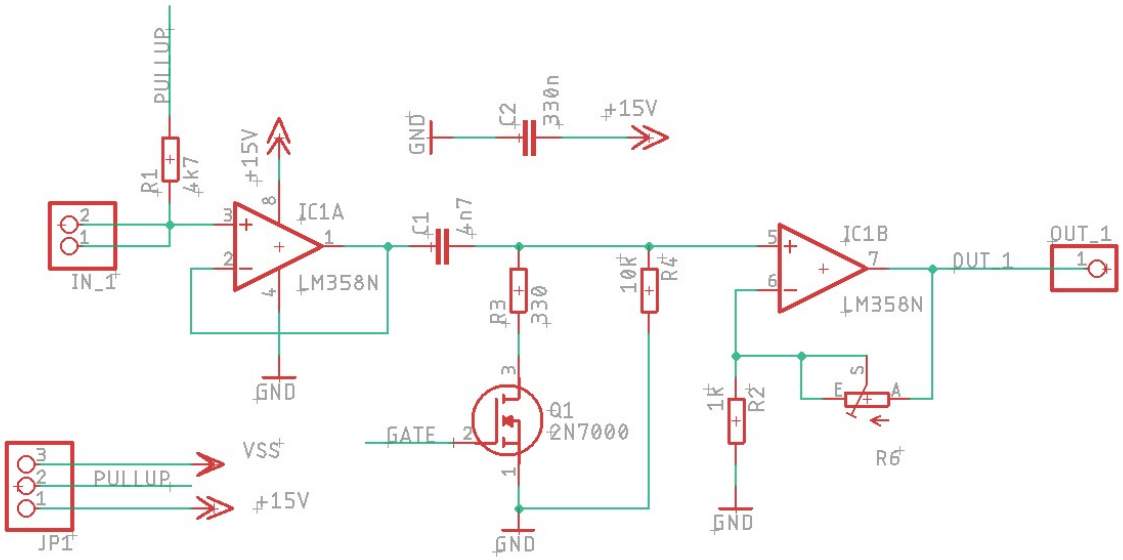
Obrázek 32: Celkové schéma zapojení původního návrhu zapojení tabule 4x4 (ještě bez WS2812B a bez vyhodnocovacích obvodů). Vytvořeno v programu ProfiCad.



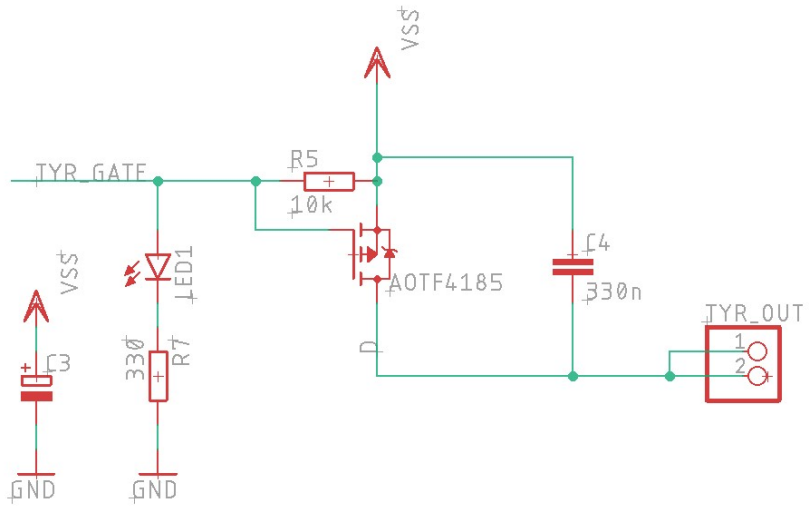
Obrázek 33: Schéma zapojení modulu pro spínání řádků. Vytvořeno v programu Eagle.



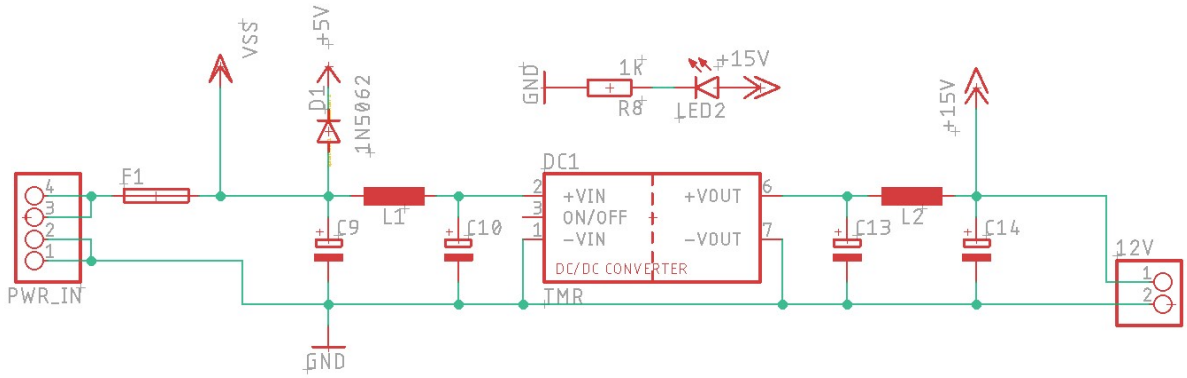
Obrázek 34: Schéma zapojení řídící jednotky část 1. Mikroprocesor a konektory. Vytvořeno v programu Eagle.



Obrázek 35: Schéma zapojení řídící jednotky část 2. Jeden z 10 vyhodnocovacích obvodů. Vytvořeno v programu Eagle.

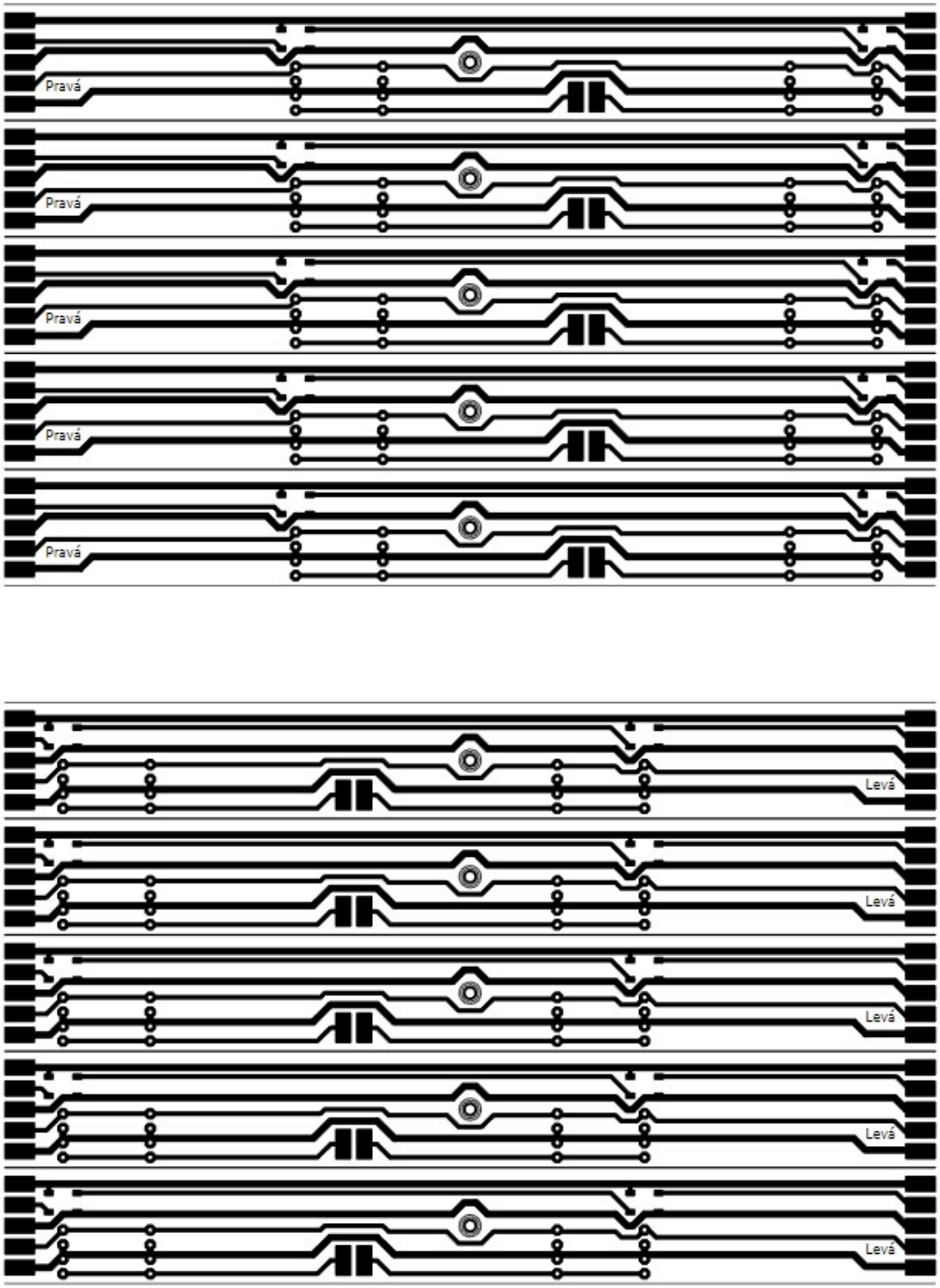


Obrázek 36: Schéma zapojení řídící jednotky část 3. Obvod pro spínání vysílačů. Vytvořeno v programu Eagle.

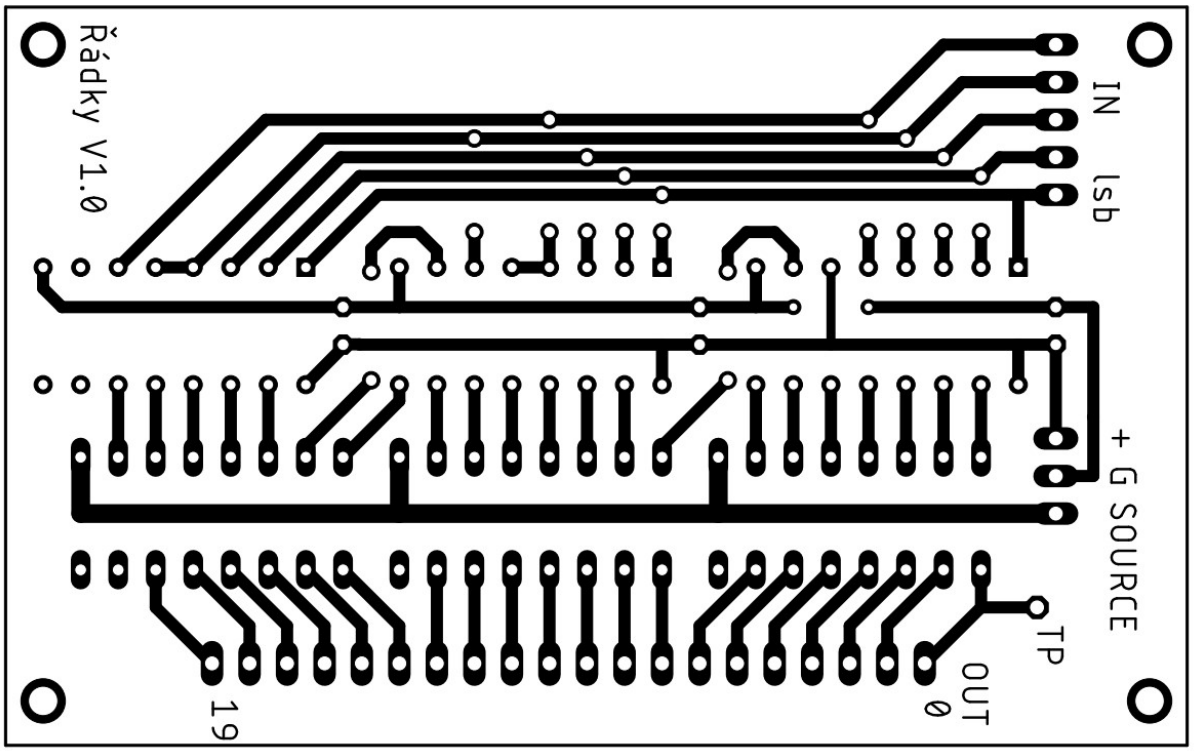


Obrázek 37: Schéma zapojení řídící jednotky část 4. Napájecí obvod. Vytvořeno v programu Eagle.

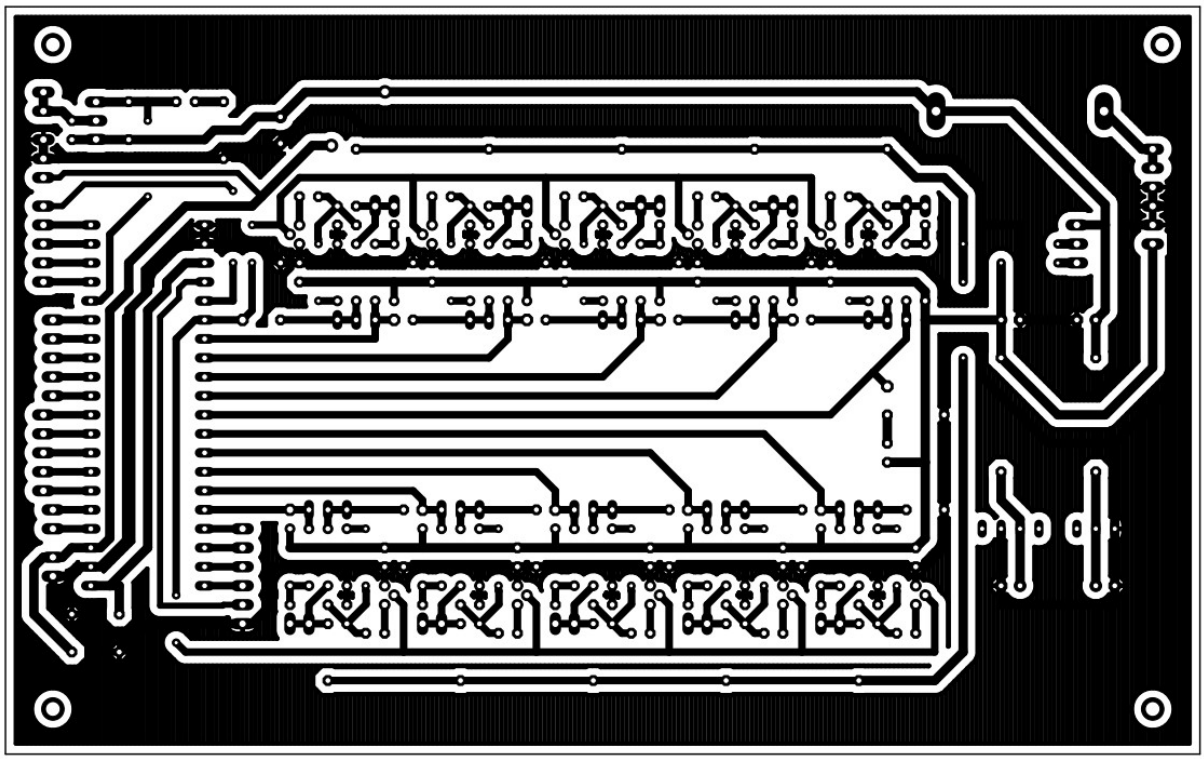
8 PŘÍLOHA 2: TECHNICKÉ DOKUMENTY



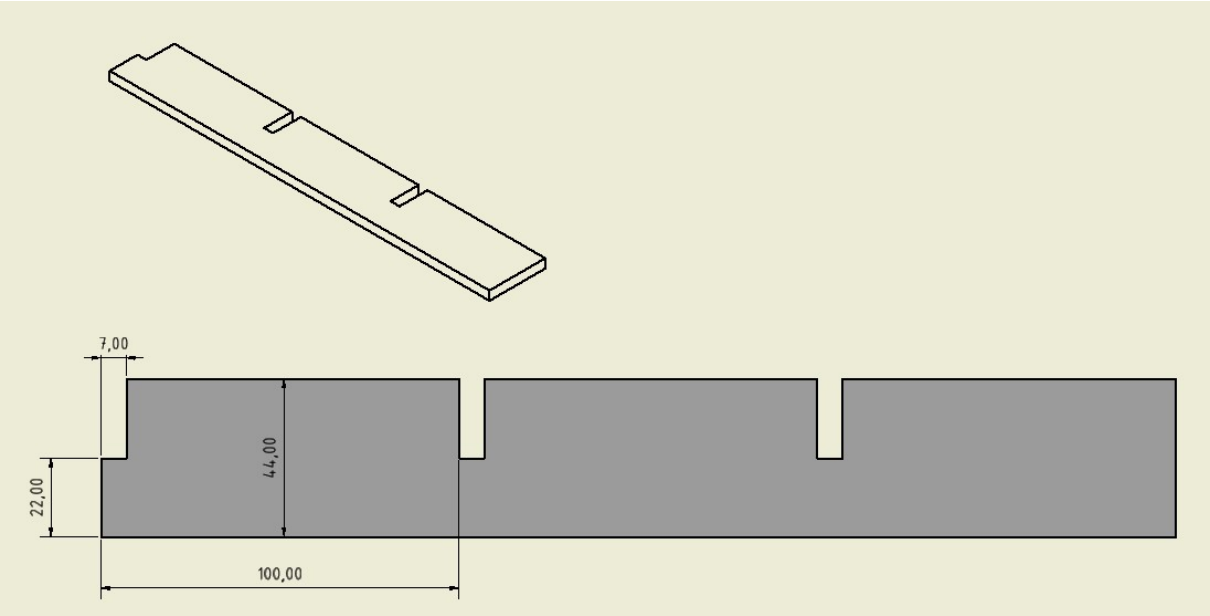
Obrázek 38: Ukázka foto předloh pro výrobu modulů pro pixely. Vytvořeno v programu Eagle.



Obrázek 39: Předloha modulu pro spínání řádků. Vytvořeno v programu Eagle.



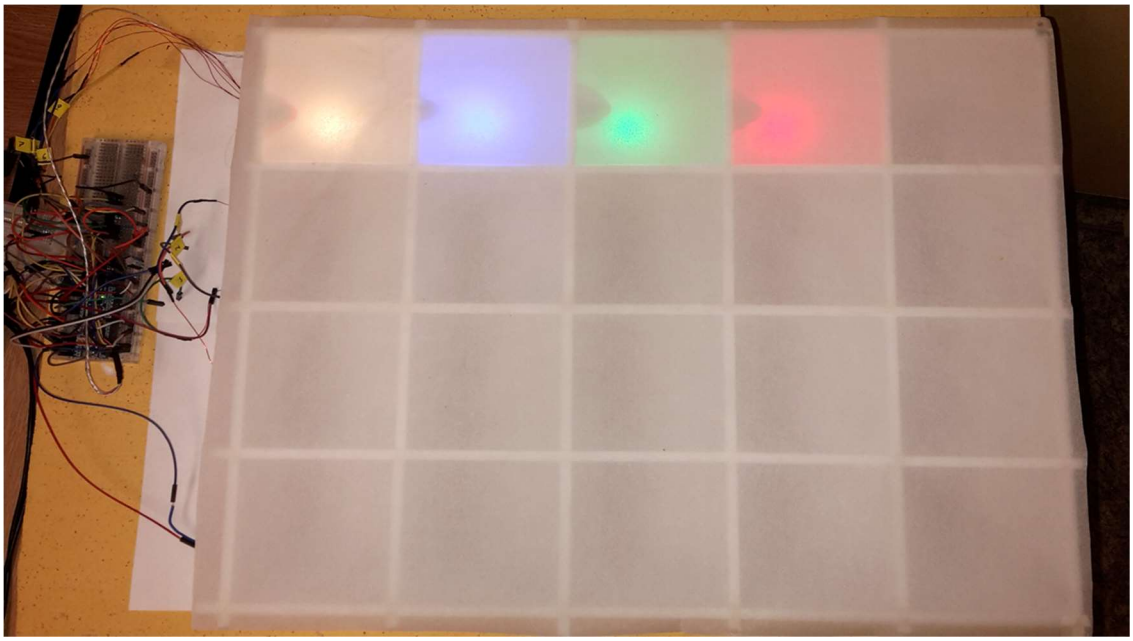
Obrázek 40: Předloha řídící jednotky. Vytvořeno v programu Eagle.



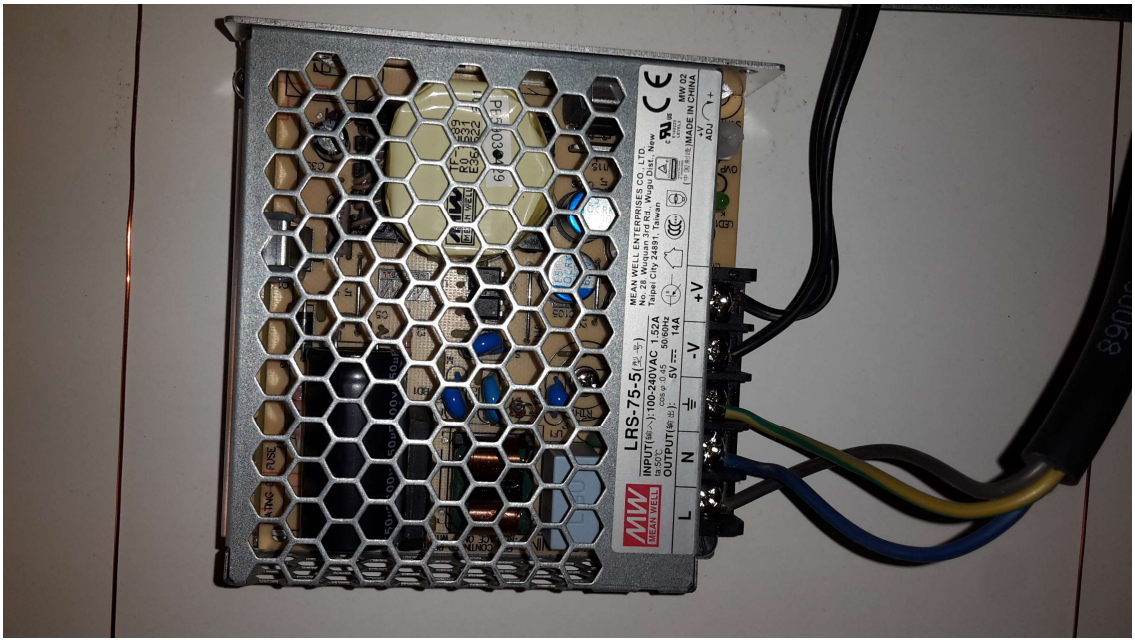
Obrázek 41: Výkres šablony pro vyřezávání polystyrenu. Vytvořeno v programu Autodesk Inventor.

9 PŘÍLOHA 3: FOTODOKUMENTACE

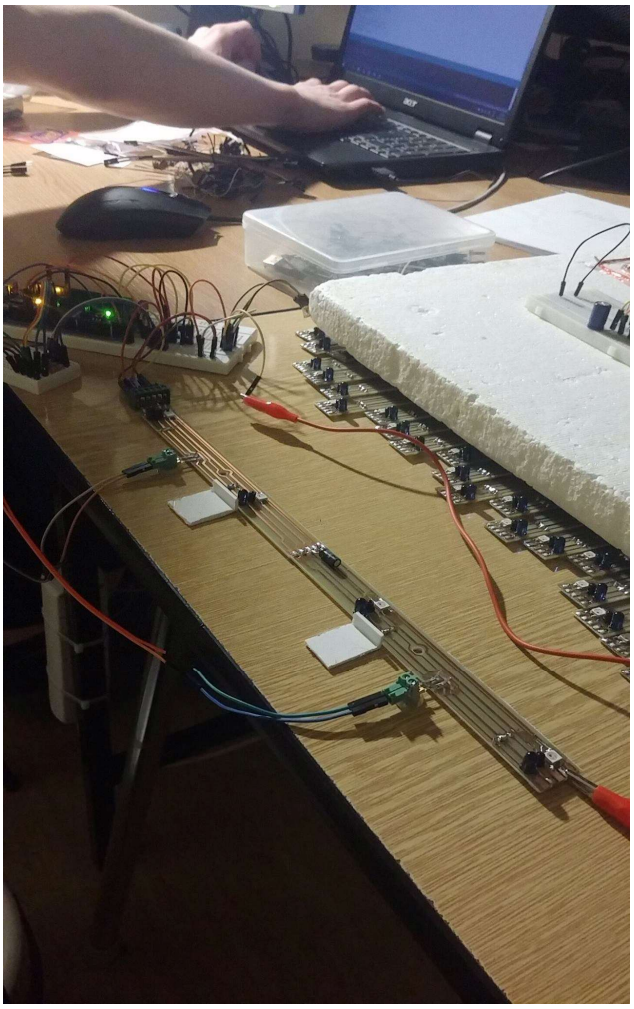
Obrázek 42: Řezačka polystyrenu.



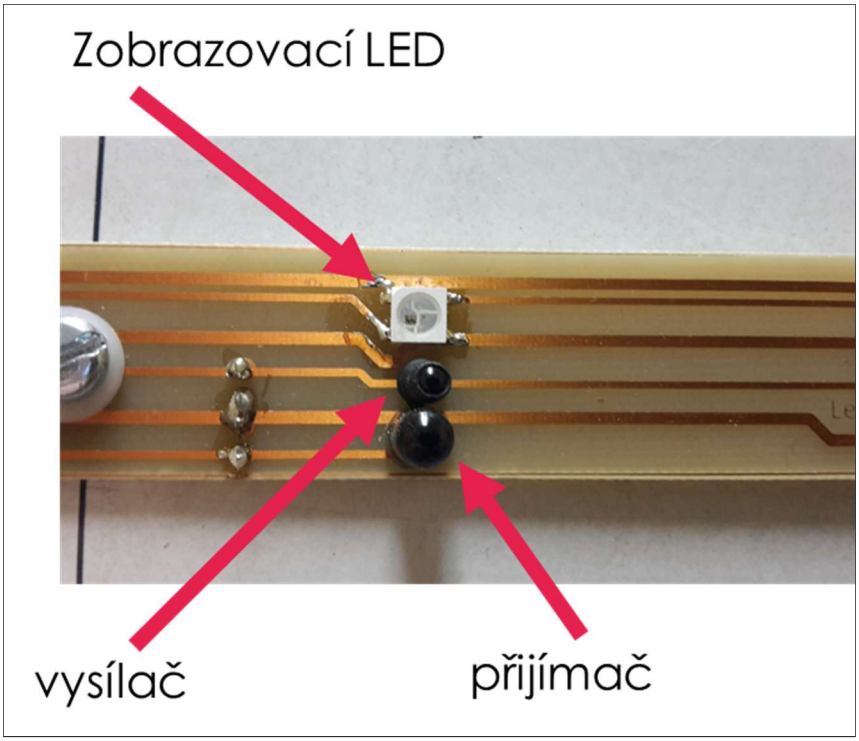
Obrázek 43: Malá tabule 4x5. Aplikace malování.



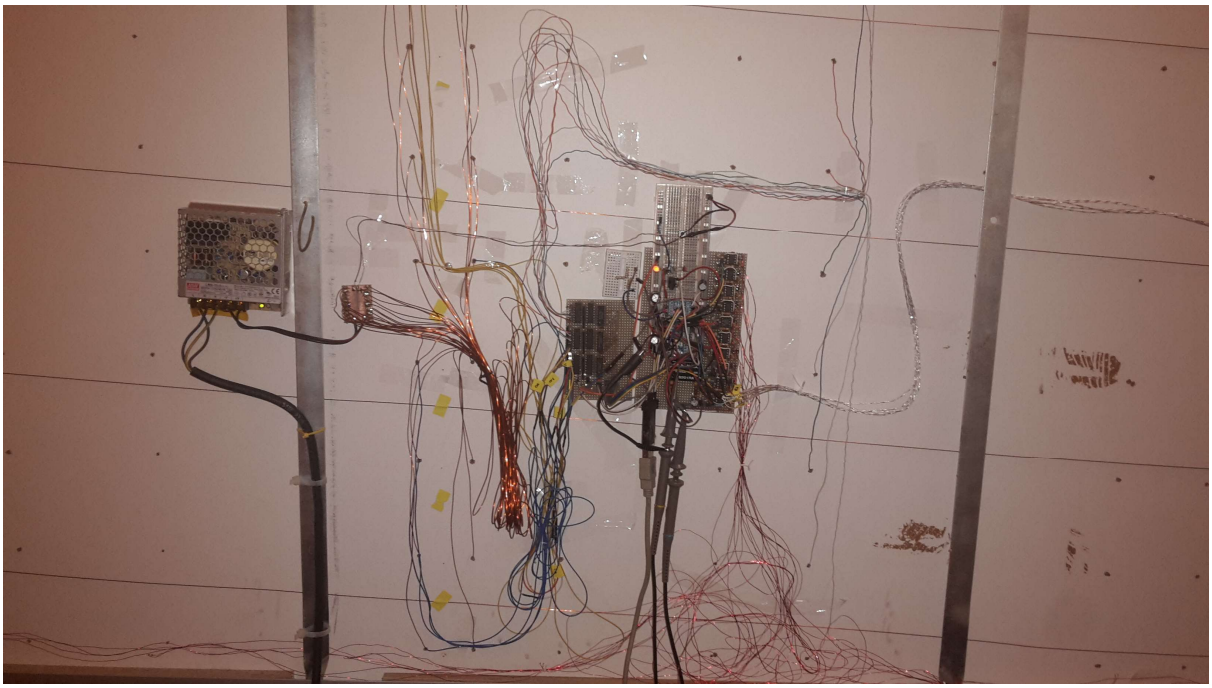
Obrázek 44: Napájecí zdroj na zadní straně tabule.



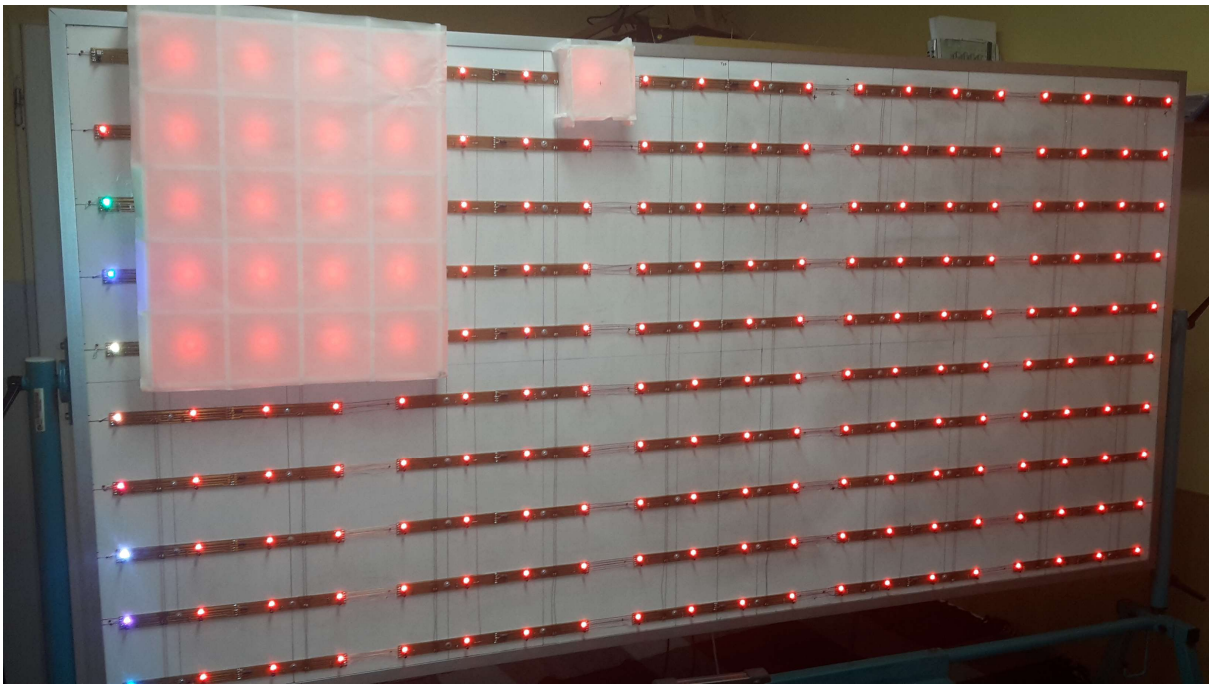
Obrázek 45: Přípravek pro testování modulů.



Obrázek 46: Detailní pohled na jeden pixel.



Obrázek 47: Zadní strana prototypu filiální verze.



Obrázek 48: Přední strana prototypu filiální verze.