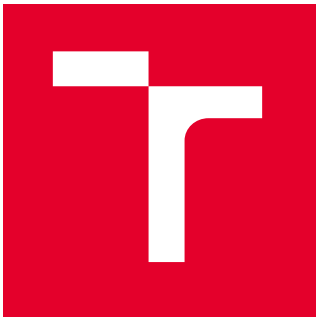


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

GRAFICKÉ UŽIVATELSKÉ ROZHRANÍ PRO EMBEDDED APLIKACE

GRAPHICAL USER INTERFACE FOR EMBEDDED APPLICATIONS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Peter Bránecký**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Petr Petyovský, Ph.D.**

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Peter Bránecký ***ID:*** 230044

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Grafické uživatelské rozhraní pro embedded aplikace

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je navrhnout a realizovat softwarové komponenty umožňující realizaci aplikací s grafickým uživatelským rozhraním (GUI) v jazyce C/C++ pro vývojový kit RPi Pico.

1. Seznamte se s platformou Raspberry Pi Pico a vývojovými prostředky pro tvorbu embedded aplikací.
2. Seznamte se s vývojovým kitem RPi Pico Kit vytvořeným v rámci [2] a tento kit oživte.
3. Nastudujte možnosti generování obrazového výstupu na RPi Pico Kitu. Zvolte knihovnu pro generování obrazového výstupu, popište její vlastnosti a demonstrujte její použití.
4. Zvolte, případně vytvořte, knihovnu pro práci s USB klávesnicí a myší, popište její vlastnosti a demonstrujte její použití.
5. Rozšiřte existující knihovnu pro práci s SD kartou tak, aby ji bolo možné použít s vývojovým kitem.
6. Pro RPi Pico Kit vytvořte aspoň dvě aplikace, které budou obsahovat využívat vytvořené knihovny a budou komunikovat s uživatelem pomocí GUI.
7. Zhodnoťte dosažené výsledky a navrhněte další možné rozšíření.

DOPORUČENÁ LITERATURA:

[1] Raspberry Pi Pico: Technical Specification. Raspberry Pi Foundation. [online]. Nov 30. 2022. Dostupné na WWW: <https://www.raspberrypi.org/documentation/microcontrollers/raspberry-pi-pico.html>.

[2] PONČÁK, M. Inovace laboratorních úloh kurzu Vestavné systémy. Brno: VUT v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2022, 122 s. Diplomová práce. Vedoucí práce: Ing. Petr Petyovský, Ph.D.

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 22.5.2023

Vedoucí práce: Ing. Petr Petyovský, Ph.D.

Konzultant: Ing. Matej Ponček

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

ABSTRAKT

Bakalářská práce se zaobírá návrhem a implementací grafického uživatelského rozhraní pro vývojovou desku Raspberry Pi Pico Kit. Tato vývojová deska je založena na platformě Raspberry Pi Pico. Dále se práce zabývá zpracováním uživatelského vstupu z klávesnice a myši skrze USB rozhraní, generováním obrazového výstupu na VGA rozhraní vývojové desky a využitím microSD karty pro načítaní a ukládaní dat.

KLÍČOVÁ SLOVA

Raspberry Pi Pico, výukový kit RPi Pico Kit, RP2040, embedded systémy, grafické uživatelské rozhraní, klávesnice, myš, microSD karta

ABSTRACT

The bachelor work deals with the design and implementation of a graphical user interface for the Raspberry Pi Pico Kit development board. This board is built around the Raspberry Pi Pico platform. Furthermore, the work deals with the processing of user input from keyboard and mouse via the USB interface, generation of image output for the VGA interface on the development board and usage of microSD card for loading and storing data.

KEYWORDS

Raspberry Pi Pico, education kit RPi Pico Kit, RP2040, embedded systems, Graphic User Interface, keyboard, mouse, microSD card

BRÁNECKÝ, Peter. *Grafické uživatelské rozhraní pro embedded aplikace*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2023, 93 s. Bakalářská práce. Vedoucí práce: Ing. Petr Petyovský, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Peter Bránecký
VUT ID autora:	230044
Typ práce:	Bakalářská práce
Akademický rok:	2022/23
Téma závěrečné práce:	Grafické uživatelské rozhraní pro embedded aplikace

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno 21.5.2023 .....

.....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Petru Petyovskému, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Dále, děkuji panu konzultantovi Ing. Mateji Pončákovi za jeho rady a konzultace k práci.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

Bakalářské práce se věnuje návrhu grafického uživatelského rozhraní pro embedded aplikace, a to konkrétně pro vývojovou desku Raspberry Pi Pico Kit. Tato vývojová deska je založena na platformě Raspberry Pi Pico. Embedded systémy typicky nedisponují dostatečnou rychlostí a velikostí paměti pro zvolení běžné knihovny pro práci z obrazovým výstupem. Některé systémy nepodporují obrazový výstup vůbec, a proto na trhu nelze najít univerzální grafické uživatelské rozhraní pro embedded systémy. Vývojová deska RPi Pico Kit obsahuje všechny potřebné periferie pro generování obrazového signálu.

V práci je obsažen popis platformy Raspberry Pi Pico a také kitu RPi Pico Kit, který je na ní založen. Zde jsou popsány periferie, možnost napájení a prostředky pro vývoj aplikací.

Dále je v práci uvedena praktická realizace zadání. Nachází se zde popis využitých knihoven pro generování VGA videosignálu (PicoQVGA, PicoVGA), zpracování uživatelského vstupu skrze USB rozhraní (TinyUSB) a práci s SD kartou (no-OS-FatFS-SD-SPI-RPi-Pico). U těchto knihoven je vysvětlen princip jejich fungování a ukázány možnosti jejich nastavení. Framework, pro tvorbu grafických uživatelských rozhraní, `RP GUI` využívá tyto knihovny pro svou funkčnost.

Další část práce se zaobírá implementací samotného `RP GUI`. Zde je popsán princip jeho funkčnosti, popsány grafické prvky, které framework implementuje a vysvětlen princip zpracovávání uživatelského vstupu předpokládající využití klávesnice a myši. Nachází se zde jak popis jeho jednotlivých částí, tak i přímo výpisky kódu a obrázky grafických prvků. Framework umožňuje pracovat s obrázky, a to dvěma způsoby. Obrázky je možné načíst staticky při překladu programu, nebo při běhu programu z SD karty.

Následující část práce se zabývá využíváním SD karty. Pro práci s knihovnou, která obsluhuje SD kartu, byla vytvořena knihovna `SDwrapper_lib`. Tato knihovna zabezpečuje inicializaci SD karty, čtení souborů z SD karty, bezpečné ukládaní souborů a ukončování komunikace s SD kartou. Pro zápis dat z programů, byla vytvořena knihovna `rplog_lib`. Využitím této knihovny dokáže programátor jednoduše zapisovat data do souborů umístěných na SD kartě nebo tyto data odesílat pomocí sériové komunikace.

Závěr práce se zaobírá testováním `RP GUI`, jeho rychlosti a funkčnosti. Jako demonstrace použití byla vytvořena hra Pong, která využívá jak prvky grafického uživatelského rozhraní, tak zpracovaná data z klávesnice. Další ukázkou použití je program KitControl. Tento program umožňuje uživateli ovládat intenzitu svitu LED na vývojové desce, nebo měnit číselnou hodnotu počítadla, která je zobrazována prostřednictvím LED na vývojové desce. K tomu jsou využívány prostředky GUI.

Cíle práce

Cílem práce je návrh a realizace softwarového nástroje, který bude umožňovat tvorbu aplikací s grafickým uživatelským rozhraním (GUI) pro vývojovou desku Raspberry Pi Pico Kit. Toto GUI bude obsahovat grafické prvky, kterými bude programátor vytvářet výsledný obraz na monitoru. Je po něm požadováno, aby bylo možné využívat obě jádra mikrokontroléru RP2040. GUI bude napsáno v jazyce C/C++. Pro generování obrazového výstupu bude potřeba zvolit vhodnou knihovnu, kterou vývojový kit podporuje. Dalším cílem práce je zpracování uživatelského vstupu z klávesnice a myši skrze USB rozhraní. Společně s tímto uživatelským vstupem a GUI, bude moct uživatel spouštět libovolné funkce programu a kontrolovat chod programu. Dále je cílem práce využít knihovnu pro práci s SD kartou, a to tak, že programátorovi bude umožněno v programu zapisovat data na SD kartu a načítat z ní obrázky nebo jiné soubory.

1 Popis platforem Raspberry Pi Pico a Raspberry Pi Pico Kit

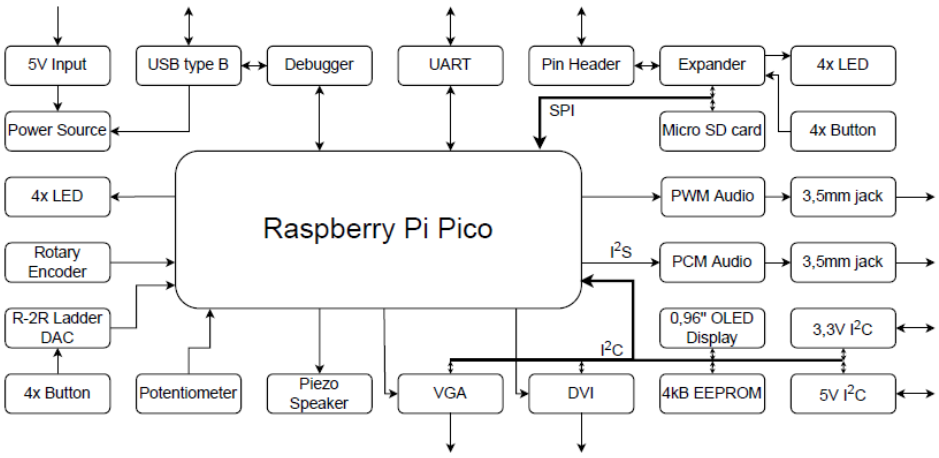
Tato kapitola obsahuje krátké obeznámení s platformami Raspberry Pi Pico a vývojovou deskou Raspberry Pi Pico Kit. Obšírnější popis platforem se nachází v příloze A.

1.1 Platforma Raspberry Pi Pico

Raspberry Pi Pico je malá mikrokontrolérová deska, založena na 32bitovém mikrokontroléru RP2040. RP2040 disponuje maximální jádrovou frekvencí procesoru až 133 MHz a statickou RAM pamětí o velikosti 256 kB. V architektuře RP2040 nalezneme 12 kanálový DMA řadič. Důležitými periferiemi pro tuto práci, kterými RP2040 disponuje, jsou: 2x UART, 2x SPI kontrolér, 1x USB kontrolér, 8x PIO stavových automatů a 30 GPIO pinů. Více informací o této platformě je dostupných v příloze A.1.

1.2 Vývojová deska Raspberry Pi Pico Kit

Platformu Raspberry Pi Pico lze modulárně připojit k vývojové desce RPi Pico Kit a tím k ní připojit řadu periferií. Pro tuto práci je nejdůležitější rozšíření o VGA rozhraní a slot pro microSD kartu. Dále se na desce nachází druhý mikrokontrolér RP2040, skrze který lze nahrávat a odladovat spuštěné programy. Blokové schéma vývojové desky se nachází na obrázku 1.1. Více informací o vývojové desce se nachází v příloze A.3.



Obr. 1.1: Bloková schéma zapojení HW platformy [1]

2 Použité softwarové knihovny

Tato kapitola se zaobírá popisem využitích softwarových knihoven v této práci. Těmito knihovnami jsou PicoVGA[7], PicoQVGA[8], TinyUSB[10] a FatFS-SD-SPI-RPi-Pico[13].

2.1 Knihovny pro generování obrazového signálu

Pro generaci výstupního obrazového signálu byly vybrány softwarové knihovny PicoVGA[7] a PicoQVGA[8]. Knihovna PicoVGA podporuje rozlišení až 1280x960 pixelů. Knihovna PicoQVGA, která je jednodušší verzí knihovny PicoVGA, podporuje rozlišení 320x240 pixelů. Obě knihovny pracují s 8bitovou barevnou hloubkou a barevným formátem R3G3B2. Vývojová deska RPi Pico Kit byla navržena tak, aby splňovala hardwarové požadavky těchto knihoven.

2.1.1 Generace VGA signálu

Vývojový kit má dedikovaných celkově 10 GPIO pinů pro generaci VGA signálu. Prvních 8 pinů je používáno pro generaci barev a zbylé 2 pro generaci synchronizačních signálů VSYNC a HSYNC. Barevné piny musí být seřazeny spojitě za sebou, nesmí být mezi nimi mezera, protože PIO modul neumožňuje měnit pořadí pinů ale pouze jejich začátek. Obdobně to platí i pro synchronizační signály. Signály pro generaci barev využívají síť rezistorů a dokážou zobrazit 8 odstínů červené a zelené, a 4 odstíny modré. Protože je používána 8bitová barevná hloubka, je celkově možné zobrazit 256 různých barev.

Informace je na obrazovce zobrazována postupně, v řádcích zleva doprava a odshora dolů. Obvod pro generaci VGA signálu pracuje na základní pixel frekvenci. Pixel frekvence určuje čas, který je potřeba na vykreslení jednoho pixelu a odvíjí se od této frekvence veškeré časování. Musí také generovat VSYNC a HSYNC signály společně s vystavováním hodnot barev pixelů na výstup. Vykreslování jednotlivých řádků na monitoru se začíná i ukončuje signálem HSYNC. Po detekci tohoto signálů monitor začíná postupně vykreslovat jednotlivé pixely do řádku na monitoru a po jeho opětovné detekci ukončuje vykreslování řádku. Až se vykreslí poslední pixel na monitoru, v pravém dolním rohu, je generován signál VSYNC a celý proces vykreslování se opakuje od začátku. Časovaní generování signálů pro jednotlivé rozlišení monitorů je rozdílné a je definováno a publikováno společností VESA.

2.1.2 Princip a nastavení knihoven

Princip

Princip generování videosignálu je vysvětlen pro knihovnu PicoQVGA. Tato knihovna je oproti PicoVGA jednodušší, avšak využívá stejných principů. Pro generaci VGA signálu, knihovna využívá jeden blok PIO modulu a dva DMA kanály. V PIO modulu jsou nahrány programy zabezpečující generaci synchronizačních signálů a program, který vystavuje jednotlivé hodnoty pixelů na výstup. Pro správné časování jednotlivých signálů, knihovna nastaví systémové hodiny na frekvenci 126 MHz.

Data do PIO modulu jsou odesílaná skrze DMA řadiče. Pro každý vykreslený řádek na obrazovce (Scanline), je potřeba poslat několik datových bufferů. Protože je potřeba posílat několik datových bufferů ne jenom jeden, knihovna využívá dva zřetězené DMA kanály. První kanál přenáší data přímo do PIO modulu a druhý tento přenos řídí a předává data do prvního kanálu. Konfigurace řídícího DMA kanálu a příprava datových bufferů se děje na konci každé scanline, po vygenerování IRQ signálu řídícím DMA kanálem. Data přenášena do PIO bloku jsou uchovávána ve frame bufferu (obrazová mezi-paměť), který v sobě obsahuje informace o všech pixelech, jaké se mají zobrazit na monitoru.

Knihovna je inicializována a pracuje na 1. jádře procesoru. Nejdříve jsou nastaveny systémový hodiny na 126 MHz, následuje inicializace PIO bloku, DMA kanálů, kontrolních bufferů a na konec se povolují IRQ, spouští DMA kanály a PIO modul. Po inicializaci na jádru 1 probíhá nekonečná smyčka a je možné do ní vkládat libovolné funkce, které se mají vykonávat. [8]

Nastavení PicoQVGA

Knihovnu PicoQVGA lze nastavit v hlavičkovém souboru `config.h`, který je umístěn přímo uvnitř knihovny. Nastavuje se tak, že se definují GPIO piny, na kterých mají být generovány výstupní signály. Dále lze definovat, který PIO blok má knihovna využívat a které kanály DMA bloku budou použity. Jak již bylo dříve zmíněno, piny musí být seřazeny bezprostředně za sebou. V hlavičkovém souboru je možné měnit i časování jednotlivých signálů. Toto nastavení se hodí, když je potřeba měnit základní frekvenci procesoru 125 MHz. Ukázka nastavení je na výpisku 2.1.

Nastavení PicoVGA

Knihovna PicoVGA se nastavuje obdobně jako PicoQVGA. Na rozdíl od PicoQVGA, umožňuje knihovna práci s vrstvami, které však framework nevyužívá, a proto jsou zde omezeny pouze na jednu. Nastavení se nachází ve hlavičkovém souboru

```
// QVGA port pins
#define QVGA_GPIO_FIRST 11  // first QVGA GPIO
#define QVGA_GPIO_NUM 8 // number of QVGA color GPIOs
#define QVGA_GPIO_LAST  (QVGA_GPIO_FIRST+QVGA_GPIO_NUM-1)
#define QVGA_GPIO_HSYNC 19  // HSYNC/CSYNC GPIO
#define QVGA_GPIO_VSYNC (QVGA_GPIO_HSYNC+1) // VSYNC GPIO

#define QVGA_PIO  pio0  //  PIO
#define QVGA_SM   0  //  state machine

#define QVGA_DMA_CB 0 // DMA control block of base layer
#define QVGA_DMA_PIO  1
```

Výpis 2.1: Ukázka nastavení PicoQVGA knihovny

`vga_config.h`, který je umístěn uvnitř knihovny PicoVGA. Výstřižek ze souboru je na výpisku 2.2.

2.1.3 Generace DVI signálu

Vývojový kit umožňuje generování i DVI videosignálu. Práce se však tímto typem video výstupu nezabývá. Hlavním důvodem je, že zatím neexistuje knihovna pro generování DVI signálu taková, která by měla implementována funkce pro vykreslování jako knihovny PicoVGA a PicoQVGA. Jediný projekt, který se zabývá generací DVI signálu, je PicoDVI [9]. Tento projekt je oproti VGA knihovnám značně nevyspělejší, složitější na používaní a pracuje až na dvojnásobně výší frekvenci jader (252 MHz), než je základní. Přitom podporuje stejné rozlišení jako knihovna PicoVGA.

2.2 Knihovna pro zpracovávání uživatelského vstupu skrze USB rozhraní

Pro získávání dat od uživatele byla použita softwarová knihovna TinyUSB [10], která podporuje širokou škálu mikrokontrolérů. Mikrokontrolér se dokáže chovat jako USB Device nebo USB Host. Pro tuto práci bude potřeba, aby se mikrokontrolér choval jako USB Host. USB Host zařízení dokáže pracovat jako: HID (Human Interface Device)[1], MSC (Mass Storage Class)[2], nebo jako hub s víceúrovňovou podporou.

[1]Rozhraní pro člověka, například klávesnice, myš nebo jiné.
[2]Zařízení pro ukládaní dat, například USB.

```
// === Configuration
#define LAYERS 1   // total layers
#define SEGMAX 1   // max. number of video segment
#define STRIPMAX 1 // max. number of video strips

#define MAXX 640 // max. resolution in X direction
#define MAXY 480 // max. resolution in Y direction

#define MAXLINE 700 // max. number of scanlines

#define VGA_GPIO_FIRST 11   // first VGA GPIO
#define VGA_GPIO_NUM 10     // number of VGA GPIOs
#define VGA_GPIO_OUTNUM 8   // number of VGA color GPIOs
#define VGA_GPIO_LAST (VGA_GPIO_FIRST + VGA_GPIO_NUM - 1)
#define VGA_GPIO_SYNC 19    // VGA SYNC GPIO
```

Výpis 2.2: Ukázka nastavení PicoVGA knihovny

TinyUSB je volně dostupná multiplatformová USB Host/Device knihovna pro embedded systémy, navržená tak, aby byla paměťově bezpečná, bez dynamických alokací a aby byla thread-safe[3]. Knihovna ukládá přijatá data po USB sběrnici do bufferu, která jsou následné zpracována v implementovaných funkcích [10].

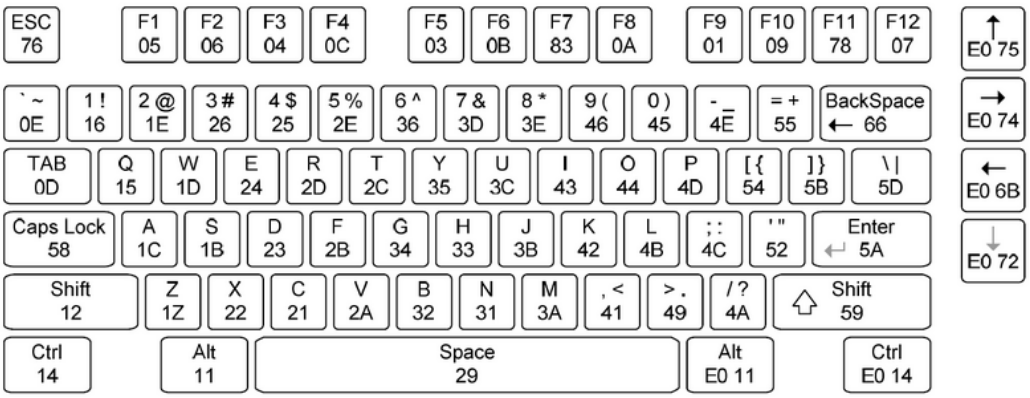
2.2.1 Využití TinyUSB knihovny

V práci je použita jako základ ukázka použití HID rozhraní [11], která byla rozšířena o zpracování puštění kláves na klávesnici a zaznamenávání dat do datových struktur. Po nahrání této ukázky do paměti mikrokontroléru byl objeven problém s napájením periferií. Připojené USB zařízení k mikrokontroléru nebylo napájeno. To bylo způsobeno tím, že vývod mikrokontroléru VBUS nebyl nikde připojen. To se dá vyřešit buď připojením USB zařízení skrze USB hub, který je externě napájený, nebo propojením VBUS vývodu na napětí 5 V. První možnost však přidává nutnost použít další zdroj napájení společně s USB hubem. Druhé řešení je elegantnější, protože napětí 5 V na VBUS vývod mikrokontroléru, je možné dostat propojením výstupu diody D2, která se nachází na přímo vývojové desce. Tato dioda je napájena skrze USB-B konektor, kterým lze desku napájet, nahrávat programy a odladovat spuštěné programy. Proto byly na výstup diody D2 a vývod mikrokontroléru VBUS připájeny piny, které lze následně přepojit kabelovou propojkou. Tímto byl odstraněn problém s napájením USB periferií.

[3]Bezpečné manipulovaní se sdílenými daty mezi jádry procesoru.

Zpracování vstupu z klávesnice

Z klávesnice lze najednou zpracovávat data o 6 právě stisknutých klávesách. Klávesnice tyto data generuje ve formě scancode[4] čísla. Omezení na 6 kláves vzniká přímo v protokolu *USB Boot Keyboard*, který využívá klávesnice pro komunikaci s deskou.



Obr. 2.1: Scancodes klávesnice [12]

Data musí být nejdříve převedena ze scancode do ASCII znaku, až pak s nimi může pracovat v kódu. Tento převod se děje za pomoci look up table (LUT)[5], která je definována v TinyUSB knihovně. Tato LUT však nezahrnuje všechny kódy, které lze z klávesnice získat, pouze alfanumerické znaky.

Při zpracovávaní dat z klávesnice se můžou vyskytnout situace: klávesa byla právě stisknuta, klávesa byla právě puštěna, nebo je stále stisknuta. Proto funkce, která data zpracovává, je rozdělená do tří částí. Tato funkce se nachází na výpisku C.4.

Zpracování dat z myši

Z myši lze dostávat informace o stisknutých tlačítkách, kterých může být až pět, a také informaci o relativní změně polohy myši v osách X a Y. Zpracování těchto dat je oproti klávesnici výrazně jednodušší a je vidět na výpisku C.5. Nastavují se zde také příznaky, zda bylo myší pohnuto, anebo bylo stisknuto tlačítko.

[4]Skenovací kód přiřazen každému tlačítku.

[5]Pole, ve kterém jsou uloženy hodnoty takovým způsobem, aby je bylo rychlejší vyhledat pro určité argumenty.

2.2.2 Nastavení TinyUSB knihovny

Nastavení knihovny se provádí v hlavičce `tusb_config.h`, která musí být dostupná při překladu programu. Zde lze nastavovat, zda budou dostupné funkce pro HID, MSC, CDC (Comunication Device Class)[6]. Dále, lze nastavit maximální množství zařízení, které dokáže knihovna podporovat a velikosti datových bufferů. Příklad nastavení se nachází na výpisku 2.3.

```
1  // CONFIGURATION
2  // Size of buffer to hold descriptors
3  // and other data used for enumeration
4  #define CFG_TUH_ENUMERATION_BUFSIZE 256
5
6  #define CFG_TUH_HUB 1
7  #define CFG_TUH_CDC 0
8  #define CFG_TUH_HID 4
9  // typical keyboard+mouse device can have 3-4 HID interfaces
10 #define CFG_TUH_MSC 0
11 #define CFG_TUH_VENDOR 0
12
13 // max device support (excluding hub device)
14 #define CFG_TUH_DEVICE_MAX (CFG_TUH_HUB ? 4 : 1)
15 // hub typically has 4 ports
16
17 //-------------- HID --------------//
18 #define CFG_TUH_HID_EPIN_BUFSIZE 64
19 #define CFG_TUH_HID_EPOUT_BUFSIZE 64
```

Výpis 2.3: Nastavení TinyUSB knihovny

2.2.3 Datový výstup z TinyUSB knihovny

Aby byla zpracovaná data z uživatelského vstupu přístupná v celém programu, byl vytvořen hlavičkový soubor `tusb_ data.hpp`. Tento soubor definuje datové struktury obsahující zpracované data z klávesnice a myši. Jsou zde také počítadla, která zaznamenávají, jak dlouho trvalo zpracování dat. Tato počítadla jsou kumulativní, proto je jich třeba nulovat manuálně po přečtení zaznamenaného časového intervalu. Dále jsou přidány indikátory toho, zda bylo s myší pohnuto nebo zda došlo ke kliknutí. Tohoto je využito při zpracování pohybu kurzoru na monitoru. Definice této struktury se nachází na výpisku 2.4.

[6]Využívá se při komunikaci mezi počítači

```
namespace HID {
    struct kbd {    // Max 6 pressed keys
        inline static uint8_t pressedKeys[6] = {};
        inline static uint64_t processTime = 0;
    };

    struct mouse {  // X,Y movement
        inline static int16_t mousePos[2] = {};
        inline static uint8_t mouseKeys[5] = {};
        inline static uint64_t processTime = 0;
        inline static bool moved = false;
        inline static bool clicked = false;
    };
}
```

Výpis 2.4: Datové struktury pro práci s TinyUSB

2.3 Knihovna pro práci s SD kartou

Pro práci z SD kartou pro platformu Raspberry Pi Pico existuje knihovna no-OS-FatFS-SD-SPI-RPi-Pico [13]. Tato softwarová knihovna poskytuje programátorovi přistup do FAT (File Allocation Table) složkového systému na SD kartě a to skrze SPI rozhraní. Pro své funkce knihovna využívá knihovnu FatFS, která je generickou FAT systémovou knihovnou pro embedded systémy. Knihovna podporuje:

- Vícero SD karet
- Využití vícero SPI rozhraní
- Použití vícero SD karet na jednom SPI rozhraní
- Použití hodin reálného času pro uchovávaní časových známek souborů
- Cyklický redundantní součet (CRC) pro kontrolu chyb
- Všechny funkce FatFS rozhraní

Knihovna, pro svou práci využívá 2 DMA kanály, které generují systémové požadavky na přerušení a aspoň jeden SPI kontrolér. V knihovně bylo nastaveno, na jakých GPIO pinech je na desce připojena SD karta, jaký IRQ kanál má být použít a jaké DMA kanály mají být využívány. Bylo zvoleno používání DMA kanálů 8 a 9. Knihovna v základu využívala funkci `dma_claim_unused_channel`, která však nepřináší správné výsledky, když jsou používány VGA knihovny. PicoVGA knihovna může pro svou funkci potřebovat až 8 DMA kanálů, právě z toho důvodu jsou nastaveny kanály na 8 a 9. Taktéž bylo nastaveno, aby byl generován IRQ signál na kanálu 1, jelikož VGA knihovna využívá pro svou funkci kanál 0.

2.3.1 Princip komunikace s SD kartou

Komunikace s SD kartou probíhá tak, že SPI kontrolér vybere kartu pomocí pinu CS, a pošle jí po MOSI (Master Out - Slave In) lince příkaz. Celý příkaz se skládá z 8bitového typu příkazu, 32bitového argumentu a 8bitového CRC (Cyclic Redundancy Check) kódu. SD karta následně pošle odpověď zpátky po MISO (Master In - Slave Out) lince. Formát odpovědi se odlišuje podle typu příkazu, někdy nemusí odpověď přijít žádná.

3 Framework pro tvorbu grafického uživatelského rozhraní - RP GUI

V této kapitole je popsán implementovaný framework RP GUI pro tvorbu grafických uživatelských rozhraní na vývojové desce, to jak funguje a jaké funkce jsou v něm implementovány.

3.1 Jádro RP GUI

V jádru framework vykonává funkce pro vykreslování obsahu na obrazovku, zpracovává data z klávesnice a myši, obsluhuje pohyb kurzoru a umožňuje přidávání stránek, které se mají vykreslovat. Framework podporuje uložení vícero stránek, přičemž jenom jedna může být aktivně vykreslována na obrazovku. Mezi těmito stránkami je možné v programu přepínat.

Pro vykreslení obsahu na obrazovku a obsluhu pohybu kurzoru, musí programátor volat funkci `Update`. V této funkci jsou také volány funkce na zpracování surových dat z klávesnice a myši. Všechny funkce, které implementuje jádro jsou dostupné programátorovi v třídě `MainApp` a jsou implementovány jako statické funkce. Programátor musí zavolat funkci `Init`, která inicializuje video mód pro generování obrazu v knihovně PicoVGA. Tato funkce se nachází v hlavní třídě `MainApp`. Pozor, funkce `Init` mění jádrovou frekvenci procesoru. Proto musí být volána před inicializací vstupně/výstupních rozhraní, aby mohli pracovat se správnou časovou základnou.

3.1.1 Vykreslování obrazu a zpracování dat

Vykreslování prvků, aktualizace pozice kurzoru a zpracování vzniklých událostí, probíhá ve funkci `Update`. Vykreslení prvků proběhne po zpracování vstupů z myši a klávesnice. Probíhá tak, že se vykreslí kurzor, všechny grafické prvky na aktuální stránce a pak znovu kurzor. Kurzor je vykreslován dvakrát, na začátku vykreslování a na konci. Je to z toho důvodu, že může nastat situace, kdy vrchní část monitoru je již vykreslena VGA knihovnou a kurzor se tak zde nestihne vložit do frame bufferu.

Jestli grafický prvek mění svoji pozici na monitoru, na místě jeho staré pozice vznikne černý obdélník a tím se efektivně vymaže z frame bufferu. Tento přístup byl zvolen proto, aby nebylo nutné při každém vykreslování vymazávat celý frame buffer a přepisovat ho úplně od začátku. Kdyby byl mazán kompletně celý, při větším množství prvků na monitoru, by se prvky nestíhali zapisovat do frame bufferu a docházelo by ke ztrátě dat v horní části monitoru.

V nastaveních `settings.hpp` lze specifikovat, na jakém jádru má vykreslování probíhat a zda se má po vykreslení čekat na VSYNC signál. Funkce pro zpracovávání dat a vykreslování, se nachází na výpiscích C.1, C.2, C.3.

3.1.2 Rozhraní `IVGA` pro vykreslování obrazu

Pro usnadnění práce s knihovnami PicoVGA a PicoQVGA, bylo vytvořeno rozhraní `IVGA`. Toto rozhraní zaobaluje buďto knihovnu PicoVGA nebo PicoQVGA, a to dle toho, jaká je použita při překladu programu. Jednotlivé funkce pro vykreslování knihoven byly zapouzdřeny do funkcí rozhraní `IVGA`. Toto rozhraní umožňuje používat obě knihovny bez toho, aby to, jakkoliv ovlivňovalo funkce, které toto rozhraní definuje. Výběr knihovny, která se má při překladu použít, se provádí v souboru `CMakeLists.txt`, který je umístěn ve složce `gui_lib`. Ukázka přepínaní knihoven se nachází na výpisku C.10.

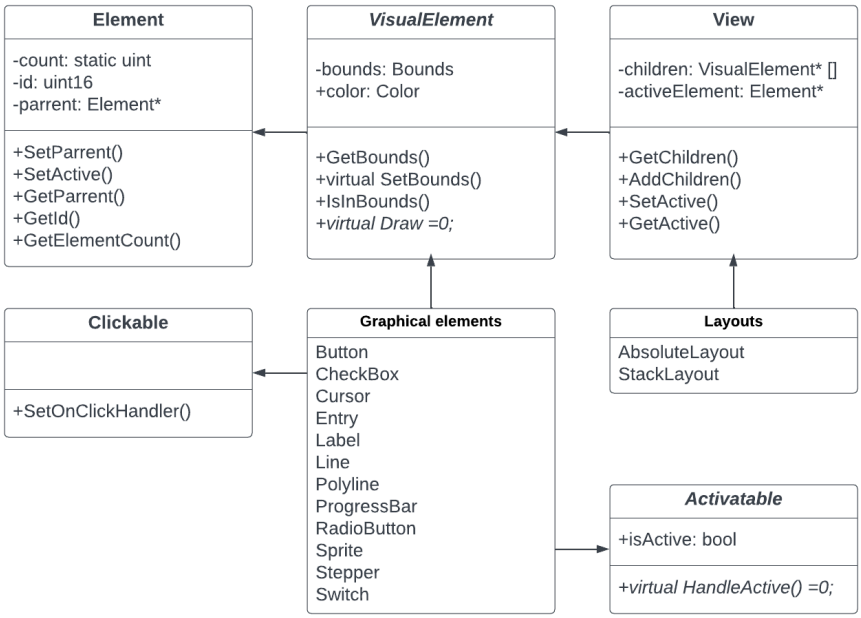
Dále rozhraní využívá datové struktury namísto primitivních datových typů, jako tomu je u PicoVGA knihoven. Využitím těchto datových struktur, rozhraní do jisté míry zabraňuje tvoření logických chyb v kódu, jako například prohození šířky a výšky v argumentech funkcí. Tyto vlastní datové struktury jsou `Point, Width, Height a Bounds`. Tyto struktury definuje framework RP GUI, ne přímo rozhraní. Je to z toho důvodu, že framework je na nich závislý a nechceme, aby je bylo možně měnit v tomto rozhraní.

Pro přehlednější práci s barvami, byl vytvořen výčtový typ `Color`. Uživatel tak může vybírat z předdefinovaných barev, které knihovny podporují. Není zde definovaných všech 256 podporovaných barev, programátor si však může jednoduše do tohoto výčtového typu implementovat barvy, jaké právě potřebuje.

Na výpisku 3.1, lze vidět všechny funkce, které rozhraní definuje. Jednotlivé implementace těchto funkcí závisí na použitém typu VGA knihovny.

```
1  void init();
2  void IDrawRectangle(...);
3  void IDrawFrame(...);
4  void IDrawPoint(...);
5  void IDrawClear();
6  void IDrawLine(...);
7  void IDrawCircle(...);
8  void IDrawText(...);
9  void IDrawImage(...);
10 void ICore1Exec(...));
11 void IWaitVSync();
```

Výpis 3.1: Funkce definované rozhraním IVGA



Obr. 3.1: Diagram dědičností prvků RP GUI

3.1.3 Běžné datové typy a struktury RP GUI

Framework a jeho části, včetně rozhraní `IVGA`, využívá pro své funkce společné základní a běžné datové typy. Tyto typy v sobě mají definována data nebo funkce, které jsou v RP GUI využívány. Na obrázku 3.1, lze vidět dědičnosti mezi běžnými datovými typy a grafickými prvkami RP GUI.

Definice základních datových typů RP GUI

V hlavičkovém souboru `Types.hpp` se nachází definice zapouzdřených základních datových typů. Jsou nimi: šířka, výška, poloměr, bod, kruh a obdélník. Je zde také výčtový typ, který definuje tři typy událostí, které mohou nastat při stisknutí myši: `Pressed`, `Released` a `Clicked`. Tyto datové typy se nacházejí uvnitř jmenného prostoru `rpgui::type`. Na výstřižku kódu 3.2 hlavičkového souboru `Types.hpp` jsou definice těchto základních datových typů.

Třída `Element`

Třída `Element` je bázovou třídou pro všechny prvky frameworku a musí její vlastnosti v nějaké podobě dědit každý prvek. Nachází se v ní informace o aktuálním počtu vytvořených prvků, identifikační číslo prvku a ukazatel na rodiče prvku. Rodič prvku je nadřazený element v procesu vykreslování obrazovky. Na výpisku 3.3 lze vidět výstřižek definice této třídy. Datový typ `ID` je definován jako `uint16_t`.

```
namespace rpgui::type {
    typedef uint16_t ID;
    struct Width{ uint16_t v; };
    struct Height { uint16_t v; };
    struct Radius { uint16_t v; };
    struct Point { int16_t x, y; };
    struct Rect {
        Point start;
        Width w;
        Height h;
    };
    struct sCircle {
        Point start;
        Radius r;
    };
    enum class MouseEventType {
        Clicked,
        Released,
        Pressed,
    };
} // namespace rpgui::type
```

Výpis 3.2: Výstřižek hlavičkového souboru `Types.hpp`

Třída `VisualElement`

V třídě `VisualElement` je definována oblast `Bounds`, kterou prvek na obrazovce zabírá a kde se vykresluje. Je zde také definována čistě virtuální funkce `Draw`, kterou musí implementovat každý prvek, který dědí třídu `VisualElement`. Protože je funkce čistě virtuální lze ji při její implementaci označit klíčovým slovem `final` a tím pádem, kompilátor nevygeneruje tabulku virtuálních funkcí pro tuto funkci. Tímto přístupem je zajištěno, že každý prvek má funkci `Draw` implementovanou, zatím co se nijak nezvětšuje výpočetní náročnost přeloženého programu. K datům o oblasti prvku na obrazovce nelze přistupovat přímo, ale přes `get` a `set` funkce, přičemž `set` funkce je označena jako virtuální a může být přepsána. Při nastavování nové oblasti, jakou prvek na obrazovce zabírá, jsou jeho stará data vymazána z frame bufferu. Tato třída dědí vlastnosti základní třídy `Element`. Na výpisku 3.4, lze vidět část kódu třídy `VisualElement`.

```
private:
    inline static uint16_t _count = 0;
    rpgui::type::ID _id;
    Element *_parrent;
public:
    /* rule of 3 ... */
    void SetParrent(Element *parrent);
    virtual void SetActive(Element* element);
    const Element *GetParrent() const;
    const uint16_t GetId() const;
    static uint16_t GetElementCount();
```

Výpis 3.3: Výstřižek kódu třídy `Element`

```
private:
    Bounds _bounds;
public:
    Color color;
    /* rule of 3 ... */
    VisualElement(const Bounds &bounds, const Color color);
    const Bounds& GetBounds() const;
    virtual void SetBounds(const Bounds &bounds);
    virtual void Draw() const = 0;
```

Výpis 3.4: Výstřižek kódu třídy `VisualElement`

Datová struktura `Bounds`

Již zmiňovaná datová struktura `Bounds`, uchovává data o mezích elementu na obrazovce. Ty jsou uchovávána v podobě startovacího bodu, šířky a výšky. Dále se zde nachází přetížení některých operátorů, které zmenšují nebo zvětšují velikost jednotlivé meze. Na výpisku 3.5 je zobrazen výstřižek datové struktury `Bounds`.

Datová struktura `Margin`

Datová struktura `Margin` uchovává informaci o tom, jak velké okraje má mít grafický prvek. Tuto strukturu využívá rozložení `StackLayout` při určování pozic prvků. Na výpisku 3.6 je zobrazen výstřižek datové struktury `Margin`.

Datová struktura `SpriteData`

V datové struktuře `SpriteData` jsou uchovávaná data o velikosti obrázku (spritu) a ukazovateli na místo v paměti, kde je sprite uložen. Tuto datovou strukturu využívá

```
struct Bounds {
    int16_t x, y;
    uint16_t w, h;
    /* rule of 3 ... */
    Bounds(int16_t x, int16_t y, uint16_t w, uint32_t h);
    Bounds operator-(const uint8_t rhs);
    Bounds operator+(const Bounds rhs);
};
```

Výpis 3.5: Výstřižek kódu datové struktury `Bounds`

```
    struct Margin {
        uint8_t l, t, r, b;
        /* Rule of 3 ... */
    };
```

Výpis 3.6: Výstřižek kódu datové struktury `Margin`

třída `Sprite`. Struktura, přesto že obsahuje ukazatel na místo v paměti, se nestará o uvolňování paměti při svém zániku. Předpokládá se, že se obrázky načtou na začátku programu a jsou ním nadále využívány. Na výpisku 3.7 je zobrazen výstřižek datové struktury `SpriteData`.

Třída `Activatable`

Děděním vlastností této třídy definujeme, že prvek dokáže být na obrazovce aktivní. To znamená, že s ním uživatel momentálně přímo interaguje, po nějaký spojitý časový úsek. V této třídě je definována čistě virtuální funkce `HandleActive` a příznak zda je prvek aktivní `isActive`.

Třída `View`

Třída `View` je základní třídou pro prvky, které budou při svém vykreslování vykreslovat i vícero jiných grafických prvků. To jsou v tomto případě prvky rozložení. V třídě je definován seznam "dětí", které prvek obsahuje, a které se mají při zavolání funkce `Draw` vykreslit. Nachází se zde také reference na aktivní prvek v daném `View`. Tato třída dědí vlastnosti třídy `VisualElement`. Na výpisku 3.8 je zobrazen výstřižek třídy `View`.

```
struct SpriteData {
        uint8_t *image;
        const int16_t size;
        const int16_t width;
        const int16_t height;
        const int16_t pitch;
        /* Rule of 5 ... */
};
```

Výpis 3.7: Výstřižek kódu datové struktury `SpriteData`

```
class View : public VisualElement {
    std::vector<VisualElement *> _children;
    VisualElement* _activeElement = nullptr;
public:
    /* Rule of 3 ... */
    View(const Bounds &coords, const Color color);

    const std::vector<VisualElement *> &GetChildren() const;
    ID AddChildren(VisualElement* child);

    ID SetActive(VisualElement* element);
    VisualElement* GetActive() const;
};
```

Výpis 3.8: Výstřižek kódu třídy `View`

Třída `Clickable`

Děděním vlastností třídy `Clickable` je přidána možnost přiřadit grafickému prvku funkci, která se bude po stisknutí na něj spouštět. Pro přidání funkce, která bude volána po stisknutí, programátor musí použít funkci `SetOnClickHandler`. Tato funkce požaduje jako parametr typ stisknutí jaké nastalo a funkci jaká se má spustit. Funkce která se má spouštět, musí mít jako parametry referenci na `MouseEvent`, ukazatel na `Clickable` prvek a návratovou hodnotu typu `void`. Použitím této funkce se zapíše do jádra RP GUI, jaká funkce se má spouštět po kliknutí na daný prvek.

Na výstřižku kódu 3.9 se nachází definice třídy `Clickable` společně s ukázkou použití. Na řádku 16, dochází k přiřazení funkce `onClicked`, ke grafickému prvku `btn`, definovaném na řádku 15.

```
1     using HandleFunc = std::function<void(MouseEvent<
      MouseEventType> &, Clickable *)>;
2  class Clickable {
3  public:
4     /* rule of 3 ... */
5     void SetOnClickHandler(const MouseEventType type, const
      HandleFunc &function);
6  };
7  // example function
8  void onClicked(MouseEvent<MouseEventType> &event, Clickable *
      sender) {
9     auto btn = (ui::Button *)sender;
10    if (btn)
11        btn->color = Color::DarkRed;
12    event.SetHandled();
13 }
14 // assign function to button
15 auto btn = new ui::Button(Width(30), Height(30), Color::
      SemiCyan);
16 btn->SetOnClickHandler(MouseEventType::Clicked, onClicked);
```

Výpis 3.9: Výstřižek kódu třídy `Clickable` s ukázkou použití

3.2 Systém na zpracovávání událostí

Pro obsluhu kliknutí myši, byl vytvořen systém na zpracovávání událostí, který jednoduše umožňuje přiřazovat `Clickable` prvkům funkce, jenž mají vykonávat po kliknutí na ně. Funkcionalitu systému zabezpečují dvě třídy `MouseEvent` a `MouseDispatcher` v spolupráci s hlavní třídou `MainApp`, která jednotlivé události iniciuje.

Funkce, která zpracovává události vytvořené myší, musí mít jako parametry: `MouseEvent<MouseEventType> &` a `Clickable *`. Ukazatel na `Clickable` prvek, lze následovně přetypovat na libovolný grafický prvek. Tímto je umožněno ho libovolně upravovat. Když si je programátor jistý, že událost je zcela zpracována, může ji nastavit jako zpracovanou a tím ukončit její další zpracovávání.

Příklad upravování prvku a nastavení události jako ukončené, je zobrazen na výpisku kódu 3.9. K přetypování prvku a jeho úpravě, dochází na řádcích 9 a 11. Nastavení události jako ukončené, probíhá na řádku 12.

```
private:
    T _type;
    bool _handled = false;
public:
    const Point pos;
    /* rule of 3 ... */
    MouseEvent(T type, const Point& position);
    inline const T Type() const;
    virtual bool IsHandled();
    void SetHandled();
```

Výpis 3.10: Výstřižek kódu třídy `MouseEvent`

3.2.1 Třída `MouseEvent`

Třída `MouseEvent` reprezentuje vzniklou událost kliknutím myši. Je zde uchovávána informace o tom, kde došlo ke kliknutí, jaký typ kliknutí vznikl (`Pressed`, `Released`, `Clicked`) a zda je událost dokončena. Na výstřižku kódu 3.10, lze vidět definici třídy `MouseEvent`.

3.2.2 Třída `MouseDispatcher`

Hlavní logika systému událostí se nachází v třídě `MouseDispatcher`. Zde se přihlašují a uchovávají informace o posluchačích událostí (listeners), který čekají na vyskytnutí se události.

Informaci o funkci, která se má po stisknutí vykonat, je uložená v datové struktuře `Handler`. Ta kromě toho že nese informaci o funkci, ukládá také informaci o tom, jaký prvek byl stisknut. Jednotlivý posluchač události dokáže mít přiřazenou prioritu, dle které bude zpracován při vyskytnutí se události. Základní priorita je nastavena na hodnotu 0. Grafické prvky v RP GUI, které v základu po kliknutí vykonávají nějakou funkci, mají nastavenou prioritu na hodnotu 1.

Třída definuje dvě metody, a to `Subscribe` a `Post`. Těmito funkcemi, se přihlašuje nový posluchač událostí, nebo zahajuje událost. Na výstřižku kódu 3.11 se nachází definice datové struktury `Handler` a třídy `MouseDispatcher`. Každému posluchači událostí je při jeho vytváření přiděleno ID. To se primárně využívá při odladňování programů.

Funkce `Subscribe`

Tato funkce přijímá jako parametry: výčtový typ události, na který má `Handler` reagovat, `Handler` strukturu a prioritu. Funkce přidá tyto parametry do mapy, mezi

```
1  struct Handler {
2      inline static uint16_t counter = 0;
3      const uint16_t id;
4      HandleFunc handler;
5      Clickable *sender;
6
7      Handler(const HandleFunc& func, Clickable* sender);
8  };
9
10 class MouseDispatcher {
11 private:
12     using PriorityListener = std::map<uint8_t,std::map<
       MouseEventType, std::vector<Handler>>>;
13     PriorityListener _listeners;
14 public:
15         /* rule of 3 ... */
16         void Post(MouseEvent<MouseEventType> &event);
17         void Subscribe(MouseEventType type, const Handler &
       handler, const uint8_t priority = 0);
18 };
```

Výpis 3.11: Výstřižek kódu datové struktury `Handler` a třídy `MouseDispatcher`

ostatní prioritní posluchače událostí.

Funkce `Post`

Funkce `Post` dle poskytnuté události, která právě nastala, zkontroluje, zda se událost stala nad některým s uložených posluchačů událostí. Jestliže ano, zavolá jeho obslužnou funkci. Implementace této funkce se nachází na výstřižku kódu 3.12. Funkce nejdříve najde, zda se v seznamu posluchačů, nachází posluchači, kteří obsluhují daný typ události (řádek 5). Jestli ano, funkce těmito posluchači iteruje a zjišťuje, zda se událost stala nad grafickým prvkem přiřazeným k posluchači (řádek 16). Pokud je tomu skutečně tak, spouští se funkce přiřazena grafickému prvku (řádek 18).

3.2.3 Ukázka použití

Na výpisku 3.13 se nachází kód, který ukazuje jak lze využít vytvořený systém událostí. Nejdříve se do vytvořené proměnné `dispatcher`, registruje tlačítko pomocí funkce `Subscribe`. V tomto kroku se definuje, na jaký typ kliknutí se má reagovat, jaká funkce bude volána a který prvek byl stisknut. To se děje na řádku 6. Na řádcích

```
1  void Post(MouseEvent<MouseEventType> &event) {
2      for (auto const &priority : _listeners)
3      {
4          // No listeners
5          if (priority.second.find(event.Type()) == priority.
       second.end())
6              continue;
7
8          for (auto const &handler : priority.second.at(event.
       Type()))
9          {
10             if (event.IsHandled())
11                 return;
12             // Posts event to handling function of listener,
13             // with information of sender objs
14             auto view = (VisualElement *)handler.sender;
15             view->GetBounds();
16             if (view->IsInBounds(event.pos))
17             {
18                 handler.handler(event, handler.sender);
19             }
20         }
21     }
22 }
```

Výpis 3.12: Implementace metody `Post`

8 a 9, jsou vytvořeny dvě události. První je typu `Clicked` a druhá `Pressed`. Obě tyto události mají společný bod kliknutí uvnitř mezí tlačítka. Na řádcích 11 a 12, jsou tyto události nastartovány. Jelikož první událost nastala nad tlačítkem `btn` a je typu `Clicked`, bude spuštěna funkce `onClicked`. Druhá událost je typu `Pressed`, tudíž, i když došlo k události nad prvkem `btn`, funkce nebude spuštěna.

3.2.4 Obsluha pohybu kurzoru myši

Obsluha pohybu kurzoru je rozdělena do dvou funkcí v třídě `MainApp`.

První funkce `processMouseInput` testuje, zdali došlo ke kliknutí tlačítka myši. Jestliže je detekováno kliknutí, vygeneruje událost s příslušným typem kliknutí, a pozicí, kde se právě nachází kurzor na obrazovce. Zde se také mění momentální stav tlačítka myši, jestli je stisknuté nebo v klidové poloze. Tato informace je potřeba k tomu, aby bylo možné rozlišit události typu `Pressed`, `Released` a `Clicked`.

```
1  void onClicked(MouseEvent<MouseEventType> &event, Clickable *
       sender);
2
3  MouseDispatcher dispatcher;
4  auto btn = new Button(Width(64), Height(8), Color::Blue);
5
6  dispatcher.Subscribe(MouseEventType::Clicked, Handler(
       onClicked, btn));
7
8  auto event1 = MouseEvent<MouseEventType>(MouseEventType::
       Clicked, Point(30,4));
9  auto event2 = MouseEvent<MouseEventType>(MouseEventType::
       Pressed, Point(30,4));
10
11 dispatcher.Post(event1);
12 dispatcher.Post(event2);
```

Výpis 3.13: Ukázka použití systému událostí

Zpracováváno je pouze stisknutí levého tlačítka myši.

Následující funkce `processMouseMovement` detekuje, zda bylo myší pohnuto, a na základě toho posouvá pozici kurzoru na obrazovce. Pozice kurzoru je limitována maximálním rozlišením monitoru a rychlost jeho pohybu závisí na citlivosti definované v hlavičkovém souboru `settings.hpp`. Funkce pro obsluhu pohyby kurzoru myši se nachází na výpiscích C.2, C.3.

Třída `Cursor`

Tato třída ukládá informace o pozici kurzoru a implementuje funkci pro jeho vykreslování. Kurzor je vykreslován jako bílý kříž. Jeho velikost a tloušťku, lze nastavit v hlavičkovém souboru `settings.hpp`, umístěném ve složce `config`.

3.3 Třídy rozložení

Framework podporuje dva typy rozložení. Jsou nimi `AbsoluteLayout` a `StackLayout`. Rozložení se používají pro shlukovaní jednotlivých grafických prvků do celků a umožňuje se tak jejich jednoduchému vykreslování v programu. Dále rozložení definují absolutní pozici prvků vzhledem k obrazovce.

3.3.1 Pomocná třída `Page`

Tato třída je určená pro shlukovaní rozložení na jednu stránku. Následně lze tyto stránky přidat do hlavní třídy `MainApp`. V hlavní třídě potom lze vybírat jednotlivé `Page`, které se budou zobrazovat na obrazovce.

3.3.2 Rozložení `AbsoluteLayout`

Rozložení typu `AbsoluteLayout` definuje absolutní pozici prvku na obrazovce, který byl do něj přidán. Toto rozvržení je primitivní, má za úkol jenom shlukovat prvky a rozmisťovat je na obrazovce. Programátor pro přidání prvku do rozložení využívá funkci `AddElement`, v které definuje absolutní pozici prvku a poskytuje ukazatel na nově přidávaný prvek.

3.3.3 Rozložení `StackLayout`

Rozložení `StackLayout` přináší oproti primitivnímu rozložení `AbsoluteLayout` funkci automatického vkládání přidaných prvků pod sebe. Zde musí programátor nastavit šířku samotného rozložení, podle kterého se budou vložené prvky zarovnávat.

Pro vložení prvku do rozložení programátor opět využívá funkci `AddElement`, která jako parametr požaduje ukazatel na přidávaný element. Ten se po přidání roztáhne na šířku rozložení, do kterého je vkládán a je mu upravena jeho absolutní poloha. Absolutní poloha je mu upravena tak, aby se vyskytoval v oblasti rozložení pod předešlým prvkem v tomto rozložení. U vkládaného elementu záleží na jeho definované výšce, která zůstane v rozložení zachována. Dalším parametrem funkce `AddElement` může být hodnota okrajů vkládaného prvku. Velikost vkládaného prvku se následně upraví dle zadaných hodnot okrajů. Mění se pozice prvku vůči okrajům rozložení a jeho šířka. Výška prvku zůstává nezměněna.

Rozložení uchovává informaci o svojí aktuální výšce, aby bylo schopné přidávat nové prvky a po každém přidaném prvku se tato hodnota navyšuje. Momentálně není nijak implementované omezení toho, kolik prvků je možné do rozložení vložit. Programátor nese zodpovědnost za to, že všechny vložené prvky do rozložení se vejdou na obrazovku.

3.4 Grafické prvky

Programátor má k dispozici řadu grafických prvků, které mu umožňují prezentovat uživateli obsah na obrazovce. Nejenže může programátor prezentovat informace uživateli, ale může mu dát možnost, interagovat s chodem programu pomocí tlačítek

nebo jiných interaktivních grafických prvků.

Veškeré grafické prvky dědí vlastnosti třídy `VisualElement`, aby je bylo možné přidávat do rozložení a následně vykreslovat na obrazovce. Celkově má programátor k dispozici 13 grafických prvků. Tyto prvky se dají rozdělit do dvou kategorií: zobrazovací a interaktivní. Zobrazovací prvky se používají pouze na zobrazovaní informací na obrazovce. S interaktivními prvky, může uživatel interagovat použitím myši nebo klávesnice. K dispozici je 7 zobrazovacích prvků:

- `Label`
- `Line`
- `Polyline`
- `ProgressBar`
- `Sprite`
- `Circle`
- `Rectangle`

A 6 interaktivních prvků:

- `Button`
- `CheckBox`
- `Entry`
- `RadioButton`
- `Stepper`
- `Switch`

3.4.1 Interaktivní prvek `Button`

Prvek `Button` umožňuje programátorovi vytvořit tlačítko, které lze umístit na monitoru. Programátor dokáže tlačítku přiřadit funkci, kterou bude vykonávat po tom, co na něj bude kliknuto myší. K tomu je dostupná funkce `SetOnClickHandler`, která je definována v bázové třídě `Clickable`. Tato funkce požaduje jako parametr typ události kliknutí myší a funkci, jaká má být po kliknutí spuštěna.

Dále je možné do tlačítka vložit text, který se bude zobrazovat uprostřed tohoto tlačítka. U textu je možné nastavovat jeho velikost, barvu a font.

Tlačítko je vykreslováno na obrazovku jako obdélník a je přes něj překreslený `Label`. Jak vypadá `Button` na obrazovce, lze vidět na obrázku 3.2.

3.4.2 Interaktivní prvek `CheckBox`

Třída `CheckBox` implementuje prvek, který se po kliknutí na něj zaškrtne a změtí tak svou interní hodnotu `checked` typu `bool`. Programátor má opět možnost přiřadit boxu funkci, kterou vykoná po kliknutí na něj. Mimo to, `CheckBox` sobě automaticky přiřazuje funkci, která po kliknutí změní vnitřní hodnotu `checked`.

Obr. 3.2: Fotografie prvku `Button` na obrazovce

`CheckBox` má definovanou minimální výšku a šířku na 5 pixelů. Na obrazovku je vykreslován jako obdélník z okraji, přes který je v závislosti na vnitřní hodnotě `checked` vykreslen křížek, skládající se ze dvou čar. Jak vypadá `CheckBox` na obrazovce, lze vidět na obrázku 3.3.

3.4.3 Interaktivní prvek `Entry`

Prostřednictvím interaktivního prvku `Entry`, je schopen uživatel zadávat data do programu, skrze připojenou klávesnici. Po kliknutí na grafický prvek `Entry`, začíná do něj uživatel psát. Po opětovném kliknutí na tento prvek, nebo stisknutí klávesy Enter, je zadaná hodnota zpracována a je uložena do paměti programu. To tohoto prvku může uživatel zadávat jak číselné hodnoty, tak i celá slova. Programátor může nastavit, s jakým datovým typem bude prvek `Entry` pracovat a tudíž, na jaký datový typ se má uživatelem zadaná hodnota převést. Ke kontrole, zda lze zadanou hodnotu převést na číselnou hodnotu, je využit regex[1], který kontroluje, zda je zadané číslo validní. Datové typy, s kterýma dokáže prvek `Entry` pracovat, mají omezení na to, že musí implementovat vstupně výstupní operátory « a ». Tento prvek jako jediný deďí vlastnosti třídy `Activatable`.

[1]Speciální řetězec znaků používaný pro vyhledávání a manipulaci s textovými daty v rámci zpracování textu

Obr. 3.3: Fotografie prvku `CheckBox` na obrazovce

`Entry` se vykresluje stejným způsobem jako prvek `Label` - přímo voláním funkcí v rozhraní `IVGA`. Jak vypadá `Entry` na obrazovce, lze vidět na obrázku 3.4.

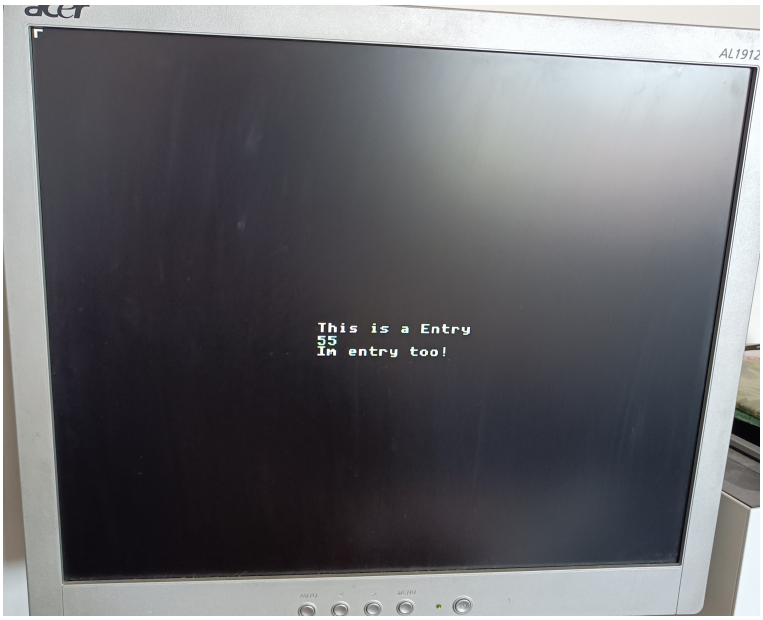
3.4.4 Zobrazovací prvek `Label`

Grafický prvek `Label` byl vytvořen, aby bylo možné vykreslovat text na monitor. Programátor při jeho tvorbě může definovat text jaký bude zobrazován, barvu textu, barvu pozadí textu, font textu a výšku textu. Na rozdíl od ostatních prvků, nelze při vytváření přímo definovat hranice jaké bude prvek zabírat na obrazovce. Ty jsou automaticky vypočítány při vytváření prvku nebo při každé změně textu funkcí `SetText` z velikosti fontu a počtu písmen. Proto když bude programátor měnit text, měl by tak dělat za použití této funkce.

Prvek je na obrazovku vykreslován přímo funkcí pro vykreslování textu z rozhraní `IVGA`. Ta zajišťuje vykreslení textu buďto z pozadím nebo bez, v závislosti na tom, jestli je barva pozadí nastavena jako transparentní. Jak vypadá `Label` na obrazovce, lze vidět na obrázku 3.5.

3.4.5 Zobrazovací prvky `Line` a `Polyline`

Tyto dvě třídy, `Line` a `Polyline`, umožňují vykreslování čar na monitor. Prvek `Line` vykreslí pouze jednu čáru. Prvek `Polyline` může vykreslovat řadu čar, které na



Obr. 3.4: Fotografie prvku `Entry` na obrazovce

sebe navazují. Prvky jsou do této třídy vkládány jako body a je mezi nimi následně vykreslena čára. Jak vypadá `Line` a `Polyline` na obrazovce, lze vidět na obrázku 3.6.

3.4.6 Zobrazovací prvek `ProgressBar`

Prvek `ProgressBar` je určen na zobrazování postupu hodnoty, v určitém definovaném rozsahu. Programátor určuje hodnotu postupu, která je typu `double` a také spodní a vrchní mez této hodnoty, která je typu `uint16_t`. Překročení těchto hodnot je hlídáno prvkem a tímto je zabezpečeno, že se vykreslovaný pás postupu bude vždy zobrazovat uvnitř prvku.

Vykreslování prvku probíhá tak, že se vykreslí obdélník na pozadí, vypočítá se aktuální šířka pásu postupu a obdélník s touto šířkou je překreslen na pozadí. Nakonec je nakreslen rámeček. Jak vypadá `ProgressBar` na obrazovce, lze vidět na obrázku 3.7.

3.4.7 Interaktivní prvek `RadioButton`

Třída `RadioButton` umožňuje programátorovi definovat tabulku položek, z které je možné kliknutím vybrat právě jednu položku. Zvolená položka je dostupná jako index do vektoru, v kterém je uložen její název a je v tomto pořadí vykreslována.

Programátor může přidávat položky do prvku funkcí `AddItem`, která požaduje jako parametr název položky. Při každém přidání položky se zvětšuje hranice, kterou

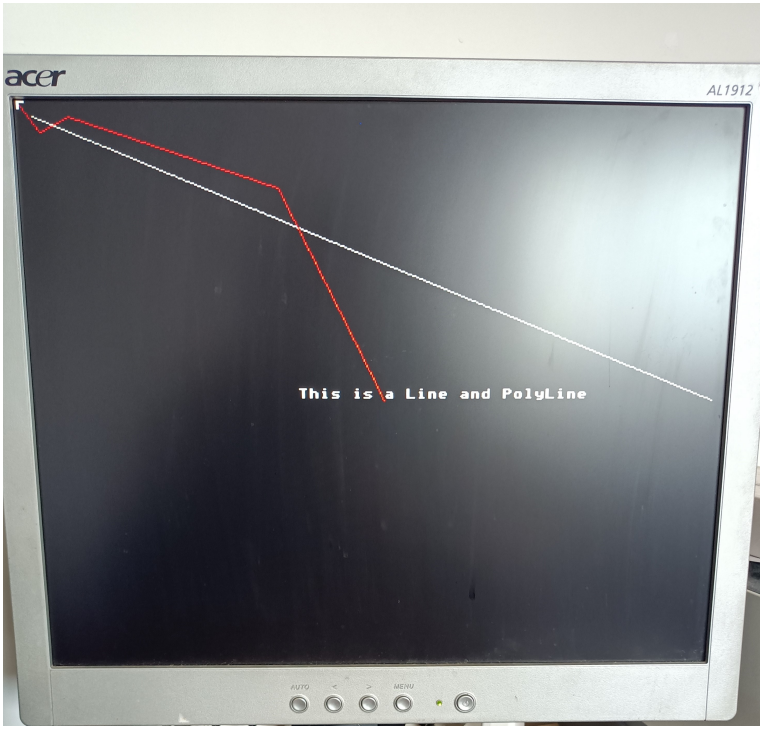
Obr. 3.5: Fotografie prvku `Label` na obrazovce

prvek zabírá na obrazovce. Proto není nutné nastavovat výšku tohoto prvku při jeho tvorbě, bude se automaticky zvětšovat podle definované výšky jednoho prvku. Při vytváření prvku je automaticky vytvořen posluchač událostí, který čeká až bude na prvek kliknuto myší. Po kliknutí se zjistí, na jakou položku bylo právě kliknuto. Index této položky v tabulce se pak nastaví do hodnoty `checkedItem`.

Vykreslování tohoto prvku probíhá tak, že se každá položka v prvku vykreslí postupně pod sebe. Ve vykreslené položce zabírá pětinu její velikosti oblast, v které je kruh znázorňující, zda je položka vybrána. Vybrání položky je znázorněno vyplněním kruhu. Velikost tohoto kruhu lze nastavit hodnotou `circleRadius` a jeho barva je stejná jako textu. Text v položce je zobrazován ve zbylých čtyřech pětinách položky. Jak vypadá `RadioButton` na obrazovce, lze vidět na obrázku 3.8.

3.4.8 Interaktivní prvek `Stepper`

Grafický prvkem `Stepper` dokáže uživatel postupně inkrementovat počítadlo. Prvek požaduje po programátorovi, aby mu byl při vytváření poskytnut datový typ, který umožňuje jak inkrementaci, tak dekrementaci. Proto lze inkrementovat jak primitivní datové typy, tak i jakékoliv datový typ, který má implementován operátory prefix `++` a prefix `--`. Při vytváření grafického prvku bez definované datové vazby na počítadlo, musí být poskytnutý datový typ implementován také základní konstruktor.



Obr. 3.6: Fotografie prvků `Line` a `Polyline` na obrazovce

`Stepper` se skládá ze dvou tlačítek, kterým je při vytváření přiřazen text - a +. Při inicializaci jsou také vytvořeny posluchače událostí, které zajišťují to, že po kliknutí na tlačítka je počítadlo buď inkrementováno nebo dekrementováno.

Vykreslení probíhá tak, že se nejdříve vykreslí okraj prvku, následuje oddělovací pás mezi tlačítky, a nakonec jsou vykresleny tlačítka samotná. Jak vypadá `Stepper` na obrazovce, lze vidět na obrázku 3.9.

3.4.9 Interaktivní prvek `Switch`

V třídě `Switch` je implementován přepínač, který přepíná mezi boolovskými hodnotami `true` nebo `false`. Tato proměnná je dostupná programátorovi pod názvem `isSwitched`. Přepínači je při jeho inicializaci přidělen posluchač událostí s funkcí, která při kliknutí na něj přepíná hodnotu `isSwitched`.

Přepínač je vykreslován jako obdélník, přes který je vykreslen další obdélník. Druhý obdélník se nachází buďto vlevo, pokud je hodnota `isSwitched false`, nebo vpravo, pokud je hodnota `isSwitched true`. Barvy obdélníků v jednotlivých pozicích může programátor definovat při vytváření přepínače. V základě jsou červená a zelená. Jak vypadá `Switch` na obrazovce, lze vidět na obrázku 3.10.

Obr. 3.7: Fotografie prvku `ProgressBar` na obrazovce

3.4.10 Zobrazovací prvky `Circle` a `Rectangle`

Grafické prvky `Circle` a `Rectangle` jsou primitivní zobrazovací prvky. Programátor může nastavovat pouze jejich pozici, velikost a barvu na obrazovce.

3.5 Datové vazby

Datové vázaní (Data binding) propojuje konzumenta dat - grafický prvek, z poskytovatelem dat - datová struktura v programu. Toto propojení umožňuje programátorovi nadefinovat data v kódu, které následně propojí z grafickými prvky. Využitím tohoto propojení pak nemusí uchovávat reference na vytvořené grafické prvky a pak v nich měnit hodnoty proměnných, ale může je měnit přímo ve svých datových strukturách. Jakékoliv změna dat je následovně propagována do grafického prvku, který je s daty svázán. Datové vazby fungují obousměrně, to znamená, že změna dat v grafickém elementu se propaguje do datových struktur a naopak.

Generická třída `BindableProperty` implementuje datovou vazbu. Při vytváření této třídy, je dle parametru konstruktoru vytvořena buď to datová vazba, nebo se vytvoří implicitní vnitřní proměnná definovaného typu. Tímto dvojím způsobem vytváření třídy, je umožněno grafickým prvkům mít buďto svou interní proměnou, nebo být datově svázán s proměnnou.

Obr. 3.8: Fotografie prvku `RadioButton` na obrazovce

Datové svázání využívají grafické prvky, u kterých se očekává, že programátor bude chtít mít přístup k datové proměnné, kterou používají. Jsou nimi:

- `CheckBox` - checked
- `Entry` - value
- `Label` - text
- `ProgressBar` - progress
- `Stepper` - counter
- `Switch` - isSwitched

3.5.1 Ukázka použití

Na výpisku 3.14 se nachází ukázka použití datových vazeb. Je zde definována datová struktura `data`, která obsahuje ukázková data (řádek 3). Na řádku 5, je vytvořeno rozložení, do kterého boudou přidávány grafické prvky. První prvek typu `Switch`, je vytvořen na řádku 7, s interní proměnou nastavenou na hodnotu `false`. Na řádku 9 je vytvořen další prvek `Switch`, a je přidán do rozložení. Na rozdíl od prvního prvku `switch1`, je tento `Switch` datově svázán s proměnnou `data.led2`.

Na řádcích 13 a 14 jsou nastaveny výstupní hodnoty, které rozsvěcuji nebo zhasínají LED. Předtím než je možné nastavit požadovanou hodnotu pro LED1, musí se nejdříve přistupovat přes prvek `switch1`, na jeho interní proměnnou (řádek 11). Využitím datových vazeb, lze nastavit hodnotu LED2 přímo, bez nutnosti přístupu

Obr. 3.9: Fotografie prvku `Stepper` na obrazovce

do jakéhokoliv prvku. A to proto, že hodnota `data.led2` je datově svázaná s prvkem v rozložení a její hodnotu nastavuje prvek po kliknutí na něj.

3.6 Práce s obrázky - zobrazovací prvek `Sprite`

Pro práci s obrázky byla vytvořena třída `Sprite`. Ta dokáže pracovat s obrázky, které jsou v datovém formátu typu `.bmp`. Programátor dokáže využívat dva typy obrázků: staticky vygenerované při kompilaci nebo dynamicky získané při běhu programu z microSD karty.

Na statické vygenerování obrázku je využíván nástroj `RaspPicoImg.cpp`. Tento nástroj vygeneruje definici obrázku do hlavičkového souboru. Jaké obrázky má tento nástroj vygenerovat musí být definováno v `CMakeLists.txt`. K tomu je dostupná funkce `qvga_generate_img_header`, která požaduje jako parametry: cestu k obrázku, jméno vygenerované struktury. Tento nástroj byl upraven, aby generoval data obrázku do datové struktury, v které budou definována jak data obrázku samotného, tak i jeho parametry (výška, šířka). Následně, při vytváření grafického prvku, jsou tyto data použita. Ukázka použití tohoto nástroje se nachází na výpisku C.11.

O dynamické načítaní prvku se stará třída samotná. Programátor při vytváření prvku zadá název obrázku a cestu k němu na SD kartě. Následně se, inicializuje SD karta, otevře se soubor s obrázkem a jsou z něj načtena data. Načítaní dat probíhá

Obr. 3.10: Fotografie prvku `Switch` na obrazovce

obdobně jako v nástroji `RaspPicoImg.cpp`. Po načtení obrázku je soubor zavřen. K práci s SD kartou je použita třída `SDWrapper`, která zabezpečuje správnou interakci s SD kartou.

Načítaní dat probíhá tak, že se načte celý `*.bmp` soubor do bufferu. Z tohoto bufferu jsou následně vytvořeny datové struktury, které v sobě obsahují parametry obrázku. Pokračuje se zkontrolováním formátu obrázku a finální načtení dat do nového, dynamicky alokovaného pole. Následně je toto pole s obrázkem uloženo do mapy instancí třídy `SpriteData`, aby ho bylo možné znovu používat ostatními instancemi třídy `Sprite`. Tímto je dosaženo toho, že obrázek je načten pouze jednou a je uložena pouze jedna jeho instance v paměti RAM.

Vykreslování obrázku probíhá přímo voláním funkce z `IVGA` rozhraní. To v závislosti na použité knihovně použije přímo data obrázku (PicoQVGA), nebo převádí tyto data do datové struktury `sCanvas`, kterou využívá knihovna PicoVGA.

3.6.1 Ukázka použití

Výpisek kódu 3.15 ukazuje, jakými způsoby lze inicializovat obrázky. Na řádku 1, je obrázek inicializován z datové struktury, která byla vygenerována nástrojem `RaspPicoImg`. Na řádku 2, je obrázek načítán z SD karty, umístěné na vývojové desce. Oproti první metodě, programátorovi stačí určit cestu a jméno souboru, z kterého bude obrázek načten.

```
struct example_data_t {
    bool led1, led2;
} data;

auto layout = new StackLayout(Point(10, 10), Width(64),
   Height(100));
auto switch1 = new Switch(Width(64), Height(8), false, Color
   ::Blue);

layout->AddElement(switch1);
layout->AddElement(new Switch(Width(64), Height(8), data.led2
   , Color::Blue));

data.led1 = switch1->isSwitched.GetValue();

gpio_put(LED1, data.led1);  //< needs to access data from
   switch1 to update
gpio_put(LED2, data.led2);  //< databound to second switch in
    layout
```

Výpis 3.14: Ukázka použití datových vazeb

Jak vypadá `Sprite` na obrazovce, lze vidět na obrázku 3.11.

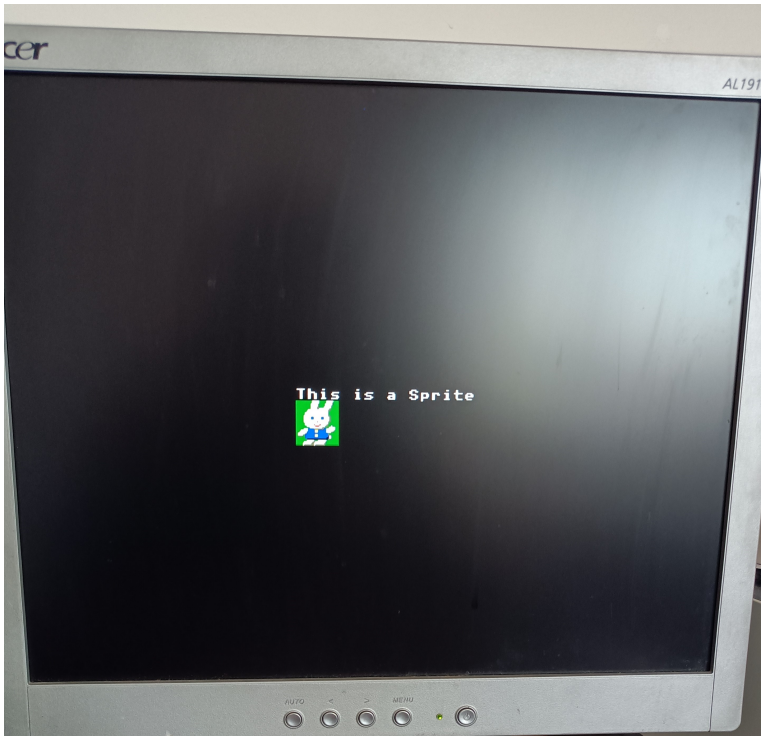
3.7 Shrnutí

V této kapitole je ukázána implementace a princip funkčnosti nástroje pro tvorbu grafických uživatelských rozhraní RP GUI. Programátor má k dispozici řadu grafických prvků, které rozmisťuje na obrazovku za použití rozložení - layouts. Tyto rozložení lze dále shlukovat do stránek, které jsou následně uloženy do hlavní třídy. Framework, jako svou jádrovou část, implementuje třídu `MainApp`. Tato třída obsahuje pouze statické funkce a atributy, proto není umožněno vytvářet její instance. V třídě jsou implementovány funkce pro aktualizaci GUI, přidávání stránek, vybíraní aktivní stránky a přidáváni posluchačů událostí. Programátor musí volat funkci `Init`, která inicializuje video mód pro funkčnost knihovny PicoVGA. Pozor, tato funkce mění hodinovou frekvenci procesoru, a proto musí být volána před inicializací vstupně-výstupních rozhraní.

Po přidání prvků do rozložení a stránek do hlavní třídy, programátor musí v kódu volat funkci `Update`. Tato funkce se zabezpečuje: zpracování surových dat z klávesnice a myši, aktualizaci polohy kurzoru, zpracovaní událostí v případě že došle

```
1     auto baloon = Sprite(Blue8Img.image, sizeof(Blue8Img.
    image), Blue8Img.width, Blue8Img.height, Blue8Img.pitch);
2     auto peter = Sprite("peter8.bmp");
3
4     peter.Draw();
5     baloon.Draw();
```

Výpis 3.15: Ukázka načítaní obrázků



Obr. 3.11: Fotografie prvku `Sprite` na obrazovce

ke kliknutí a vykreslení všech prvků na monitor. V nastaveních lze určit, jestli se bude funkce pro vykreslení spouštět na prvním nebo druhém jádru procesoru a také zda se má po vykreslení čekat na VSYNC signál.

Všechny třídy implementované v RP GUI dodržují zásadu tří. To znamená, že mají vždy nadefinování default counstructor, copy constructor a destructor. Ve většině případů je default counstructor a copy constructor vymazán. Pro vytváření prvků se využívají konstruktory, které obsahují parametry jako jsou: šířka, výška, barva a jiné parametry specifické pro daný prvek. Jelikož tyto konstruktory mohou požadovat velké množství parametrů, část z nich je vždy nastavena na předvolené hodnoty.

Framework umožňuje používaní datových vazeb, proto jsou u prvků, které podporují tyto vazby dostupné dva konstruktory. Pro hodnotu, která se má vytvořit

jako vnitřní (definována přímo v prvku), nebo pro hodnotu, z kterou má být prvek datově svázán.

Další funkce, které by mohl framework implementovat jsou: rolování obrazu - přidání rolovacího rozložení, možnost vytváření šablon pro grafické prvky, další grafické prvky např. Slider a optimalizaci vykrelovacího procesu. Dalším možným rozšířením je využití pravého tlačítka myši pro interakce s grafickými prvky.

```
int main() {
    rpgui::core::MainApp::Init();
    stdio_init_all();
    LED_init();
    struct {
        bool led[4] {false};
    }leds;

    auto layout = new StackLayout(Point(WIDTH/2-32, HEIGHT/2)
   ,Width(64), Height(128));
    layout->AddElement(new Switch(Width(0), Height(16),leds.
   led[0]));
    layout->AddElement(new Switch(Width(0), Height(16),leds.
   led[1]));
    layout->AddElement(new Switch(Width(0), Height(16),leds.
   led[2]));
    layout->AddElement(new Switch(Width(0), Height(16),leds.
   led[3]));
    MainApp::AddPage(new Page(layout));

    while (1) {
        MainApp::Update();
        gpio_put(LED1, leds.led[0]);
        gpio_put(LED2, leds.led[1]);
        gpio_put(LED3, leds.led[2]);
        gpio_put(LED4, leds.led[3]);
    }
}
```

Výpis 3.16: Ukázka použití RP GUI

3.7.1 Ukázka použití RP GUI jako celku

Na výpisku 3.16 se nachází kód, kterým je vytvořeno grafické uživatelské rozhraní a jsou skrze něj ovládány LED na vývojové desce. Na řádcích 3 až 5 jsou inicializovány periferie a knihovny. Na řádcích 11 až 16, je postupně vytvářeno celé grafické uživatelské rozhraní. To je následně ve smyčce aktualizováno a jsou přiřazeny hodnoty LED na výstup GPIO pinů. Takovýmto poměrně krátkým kódem, bylo možné vytvořit GUI, které ovládá LED na vývojové desce. K tomu stačilo napsat 15 řádků kódu, z čehož 8 řádků zahrnuje obsluhu a tvorbu GUI.

4 Práce s SD kartou

V této kapitole je shrnuto, jaký způsobem je využita knihovna FatFS-SD-SPI-RPi-Pico a k ní vytvořená mezivrstva `SDWrapper`.

4.1 Balíček pro práci s SD kartou

Třída `SDWrapper` je využívána jako mezivrstva pro práci s knihovnou FatFS-SD-SPI-RPi-Pico. Tento balíček (wrapper) má za úkol zjednodušit práci se soubory uloženými na SD kartě, otevírat a zavírat je. Wrapper se stará o inicializovaní složkového systému na SD kartě, udržuje přehled o otevřených souborech, bezpečně zavírá soubory a ukončuje komunikaci s SD kartou.

Jsou zde k nalezení funkce `Init`, `Dispose`, `OpenFile`, `CloseFile`, a `CloseAllFiles`. Všechny funkce a členské proměnné třídy jsou implementovány jako statické.

Funkcí `Init` se inicializuje FAT složkový systém na SD kartě. Funkcí `OpenFile` může programátor otevřít soubor s definovaným jménem a cestou k němu na SD kartě. Zde lze také definovat v jakém módu soubor otevřít. V základu se soubor otevírá na čtení a zápis, který začíná na konci složky. Funkcemi `CloseFile` a `CloseAllFiles` může programátor bezpečně zavírat složky. Nakonec funkce `Dispose` automaticky zavře otevřené složky a ukončuje komunikaci s SD kartou.

Funkce mají jako návratový typ výčtový typ `FRESULT`, který je definován v hlavičkovém souboru `ff.h`. Tento výčtový typ reprezentuje, jakým způsobem dopadly operace se složkovým systémem. Co znamenají jednotlivé návratové kódy lze nalézt v dokumentaci knihovny FatFS [14].

Tento wrapper je vytvořen jako vlastní knihovna a nese název `SDwrapper_lib`. Výstřižek kódu obsahující definice funkcí této knihovny se nachází na výpisku 4.1.

4.2 Datový logger

SD karta najde své využití mimo toho, že se z ní načítají obrázky, také při ukládaní důležitých informací z programu. K tomu byla vytvořena knihovna `rplog_lib`, která implementuje třídu `Logger`. Využíváním této knihovny může programátor ulehčit svou práci při logovaní dat. Programátor může využívat funkce třídy `Logger` pro zaznamenávání dat. Tyto funkce požadují jako parametr správu, která se má zaznamenat a místo kde se má správa logovat. Místo pro logování, takzvaný `sink`, může být typu buďto `FIL` (složka) nebo `std::ostream` (výstupní tok).

Dále lze vytvářet instance této třídy. To těchto instancí lze přidávat vícero `sink`, které reprezentují místo, do kterých budou data logována. Přidáním vícerých `sink`

```
class SDWrapper
{
private:
    inline static bool _sdInit = false;
    inline static std::vector<FIL *> _openedFiles;

public:
    SDWrapper() = delete;
    SDWrapper(const SDWrapper &) = delete;
    ~SDWrapper() = delete;

    static FRESULT Init();
    static void Dispose();

    static std::tuple<FRESULT, FIL *> OpenFile(const std::
   string &name, const std::string &path, const char mode =
   FA_READ | FA_WRITE | FA_OPEN_APPEND);
    static FRESULT CloseAllFiles();
    static FRESULT CloseFile(FIL *file);
};
```

Výpis 4.1: Definice funkcí třídy `SDWrapper`

do instance třídy `Logger` a voláním logovací funkce, programátor zaznamenává data na vícero míst voláním jedné logovací funkce.

Je zde také definován výčtový typ `Level`, který reprezentuje závažnost správ. Ve vytvořeném logovacím objektu, lze nastavit úroveň logování v proměnné `logLevel`, která definuje úroveň závažnosti, od které se budou data zaznamenávat. U statických metod třídy je tato hodnota definovaná taktéž, jako proměnná `globalLogLevel`. Význam nastavování logovací úrovní spočívá v tom, že pokud zaznamenávána správa má menší logovací úroveň, než je ta nastavená, nebude správa nikde zaznamenána.

Mimo toho, že lze nastavovat logovací úrovně jednotlivých instancí třídy `Logger`, je ji možné nastavit také vloženým `sink`. Logovací úroveň instance je této úrovni nadřazena, pokud je vyšší než úroveň jednotlivého `sink`.

Programátorovi jsou dostupné tyto logovací úrovně:

- TRACE
- DEBUG
- INFORMATION
- WARNING
- ERROR

```
1  class Logger {
2     private:
3          std::vector<Sink> _sinks;
4     public:
5          Level logLevel = Level::TRACE;
6          inline static Level globalLogLevel = Level::TRACE;
7          /* rule of 3 ... */
8
9          bool AddFile(const std::string& name, const std::
     string& path, Level logLevel);
10         void CloseFiles();
11         static void DisposeSD();
12
13         void AddSink(FIL *const file, Level logLevel));
14         void AddSink(std::ostream &stream, Level logLevel));
15         void Log(const std::string &message, const Level &
     severity);
16
17         static void log(FIL* const file, const std::string&
     message, const Level severity);
18         static void logTrace(FIL *const file, const std::
     string &message);
19         /* other static methods for logging ... */
20     };
```

Výpis 4.2: Výstřižek kódu třídy `Logger`

- CRITICAL
- None

Skrze logger dokáže programátor přímo otevírat složky a přidávat je tak do `sinks`, zavírat všechny složky a také, ukončovat komunikaci s SD kartou.

Na výpisku 4.2 se nachází výstřižek kódu třídy `Logger`.

4.2.1 Příklad použití

Jako příklad použití, byl vytvořen program, který zaznamenává data do souborů na SD kartě, a také, je odesílá do standardních výstupních toků. Tento program se nachází na výpisku 4.3.

Na tomto výpisku kódu je vytvořena instance třídy `Logger`. Je jí nastavena logovací úroveň `TRACE` na řádku 2. Na řádcích 3 až 6, jsou do instance přidána místa, kde se mají zaznamenávat data. K těmto místům jsou přiřazeny různé logovací úrovně.

```
1  auto logger = new Logger();
2  logger->logLevel = Level::TRACE;
3  logger->AddSink(std::cout, Level::DEBUG);
4  logger->AddFile("logs_verbose.txt","", Level::TRACE);
5  logger->AddSink(std::cerr, Level::WARNING);
6  logger->AddFile("logs_error.txt","", Level::ERROR);
7
8  logger->Log("LOGGER TRACE MESSAGE", Level::TRACE);
9  logger->Log("LOGGER DEBUG MESSAGE", Level::DEBUG);
10 logger->Log("LOGGER INFO MESSAGE", Level::INFORMATION);
11 logger->Log("LOGGER WARNING MESSAGE", Level::WARNING);
12 logger->Log("LOGGER ERROR MESSAGE", Level::ERROR);
13 logger->Log("LOGGER CRITICAL MESSAGE", Level::CRITICAL);
14
15 Logger::globalLogLevel = Level::WARNING;
16 Logger::logTrace(std::cout, "LOGGER TRACE MESSAGE");
17 Logger::logDebug(std::cout, "LOGGER DEBUG MESSAGE");
18 Logger::logInfo(std::cout, "LOGGE INFO MESSAGE");
19 Logger::logWarning(std::cout, "LOGGER WARNING MESSAGE");
20 Logger::logError(std::cout, "LOGGER ERROR MESSAGE");
21 Logger::logCritical(std::cout, "LOGGER CRITICAL MESSAGE");
22
23 Logger::DisposeSD();
```

Výpis 4.3: Příklad použití třídy `Logger`

Zpráva na řádku 8, bude zaznamenána pouze do souboru `logs_verbose.txt`. Zpráva na řádku 9, bude zaznamenána jak do `logs_verbose.txt`, tak i na výstup `std::cout`.

Podobně je tomu tak i u ostatních zpráv, přičemž do souboru `logs_error.txt`, budou zaznamenány zprávy s úrovní `ERROR` a `CRITICAL`.

Na řádcích 16 až 21, jsou využity statické funkce pro zaznamenávaní dat. Globální úroveň logovaní je nastavena na úroveň `WARNING`, proto budou zaznamenávány správy z důležitostí `WARNING` a vyšší.

Na řádku 23 je ukončena komunikace s SD kartou a zavřeny otevřené soubory.

5 Testování a ukázkové aplikace

Tato kapitola je zaměřena na testování rychlosti RP GUI a ukázku aplikací, které používají GUI, vytvořeno pomocí RP GUI nástroje.

5.1 Testování rychlosti

Rychlost RP GUI byla testována tak, že byli vykreslovány jednotlivé grafické prvky, které RP GUI implementuje a měřena doba trvání vykreslování. V testech byl měřen průměrný čas vykreslení 50 prvků, průměrný čas vykreslí jednoho prvku, maximální a minimální čas vykreslení jednoho prvku. Průměrné časy vykreslování byli vypočteny ze vzorku o velikosti 100 hodnot. Funkce, která byla pro toto měření využita, se nachází na výpisku kódu C.6. Naměřené výsledky jsou uvedeny v tabulce 5.1

Tab. 5.1: Tabulka naměřených dob trvání vykreslování prvků RP GUI

Prvek	50 Prvků [µs]	1 Prvek [µs]	Max. [µs]	Min. [µs]
Button	55 527	1 110	1 118	1 110
CheckBox	53 265	1 065	1 070	1 063
Entry	19 462	389	395	380
Label	45 882	917	922	917
Line	5 304	106	118	105
PolyLine	5 322	106	120	105
ProgressBar	54 373	1 087	1 091	1 086
RadioButton	73 768	1 475	1480	1474
Stepper	57 492	1 149	1 154	1 147
Switch	78 780	1 575	1 580	1 573
Sprite	3 433	68	70	68

Hlavním faktorem rychlosti vykreslování prvků byla jejich celková velikost, jakou zabírali ve frame bufferu. Prvky `Button`, `CheckBox`, `ProgressBar`, `RadioButton`, `Stepper` a `Switch` byly vykreslovány s velikostí 128x128 pixelů (16384 pixelů). `Label` byl vykreslován s 40 písmeny to jest 40*8*8 pixelů (2560 pixelů). Prvek `Entry` byl vykreslován s maximální hodnotou proměnné typu `int` 2147483647, což představuje 10*8*8 pixelů (640 pixelů). Tato hodnota je převáděna na proměnnou typu `std::string`. Proto bude vykreslování trvat déle než kdyby je použita hodnota typu `std::string`. `Line` a `Polyline` byly vykreslovány z rohu do rohu obrazovky, to je přibližně 452 pixelů. Vykreslovaný `Sprite` měl rozměry 32x32 pixelů (1024 pixelů).

Když porovnáme prvky se stejnou zobrazovací plochou, je vidět, že nejpomalejší na vykreslování jsou prvky `Switch` a `RadioButton`. Volání funkcí pro zaznamenávání času, měli na výsledek měření minimální dopad, protože spolu trvají 2 µs.

5.2 Ukázková hra Pong

Jako ukázka funkcionality, kterou framework RP GUI poskytuje, byla vytvořena hra Pong. Pong je klasická 2D arkádová hra z roku 1972, která simuluje stolní tenis. Je to jedna z prvních úspěšných video her na světě. Hra obsahuje dvě pálky, které se pohybují vertikálně na stranách obrazovky a kuličku, která se odráží mezi nimi. Cílem hráče je kuličku odrážet pomocí pálek a dostat ji na okraj obrazovky protihráče. Hráči, mohou pomocí kláves `w`, `s`, `i` a `k` měnit pozici pálek. Po odražení kuličky od pálek, se postupně zvětšuje její rychlost. Směr odražení kuličky závisí na tom, z jakého segmentu pálky byla odražena. Z vrchního segmentu se kulička odráží nahoru, ze spodního dolu a z prostředního se odráží kulička pod stejným uhlem jako dopadajícím.

Hra využívá pro své zobrazování prvky: `Label` jako počítadla skóre, `Circle` jako odráženou kuličku a `Rectangle` jako pádla. Rychlost aktualizace logiky hry dosahovala časového intervalu 133 µs, aktualizace grafického rozhraní zabírala 228 µs. Tudíž se většina času strávila čekáním na VSync signál. Zpracování dat z klávesnice trvalo přibližně 45 µs.

Vyfocená obrazovka s hrou Pong se nachází na obrázku 5.1. Dále se v elektronické příloze nachází video, kde je hrána hra Pong. Toto video je umístěno ve složce `video_examples`.

5.3 Ukázkový program KitControl

Jako druhá ukázka použití nástroje RP GUI, byl vytvořen program, skrze který, dokáže uživatel rozsvěcovat LED na vývojové desce. K tomu je potřeba mít připojenou klávesnici a myš. GUI aplikace se skládá ze dvou stránek, na první se nachází ovládání čtyř LED pomocí `Switch` a `ProgressBar` grafických prvků. Uživatel může volit, které LED jsou zapnuté a kterým právě nastavuje intenzitu svitu pomocí potenciometru na desce. Fotografie této stránky se nachází na obrázku 5.2.

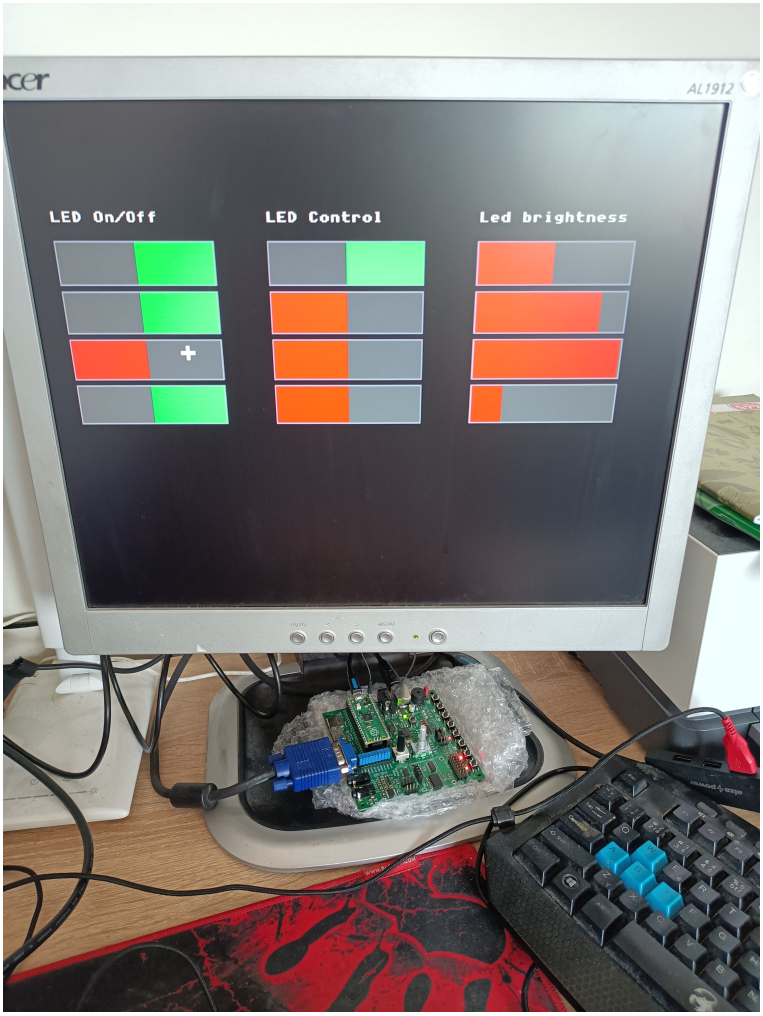
Další stránkou uživatel zobrazuje číselné hodnoty 0 až 15 pomocí čtyř LED na desce. Zobrazovanou hodnotu může uživatel měnit pomocí enkodéru na desce a využitím grafických prvků `Stepper` nebo `Entry`. Na zobrazování aktuální hodnoty počítadla je použit prvek `Label`. Hodnotu počítadla lze zadávat přímo z klávesnice prostřednictvím prvku `Entry`. Fotografie této stránky se nachází na obrázku 5.3.

Obr. 5.1: Fotografie hry Pong na obrazovce

Uživatel může přepínat mezi jednotlivými stránkami využitím kláves `a` a `s` na klávesnici. Video kde je tento program používán, se nachází v elektronické příloze ve složce `video_examples`.

5.3.1 Popis kódu aplikace

Aplikace je rozdělena do dvou stránek. Každá stránka pracuje s datovou strukturou, funkcí pro vytvoření GUI a kontrolérem, který aktualizuje a zpracovává data. Na výpisku kódu C.7 se nachází hlavní funkce programu. Zde jsou vytvořeny jednotlivé stránky s GUI, inicializovány potřebné periferie a volány aktualizační funkce programu.



Obr. 5.2: Fotografie první stránky programu KitControl

Stránka 1

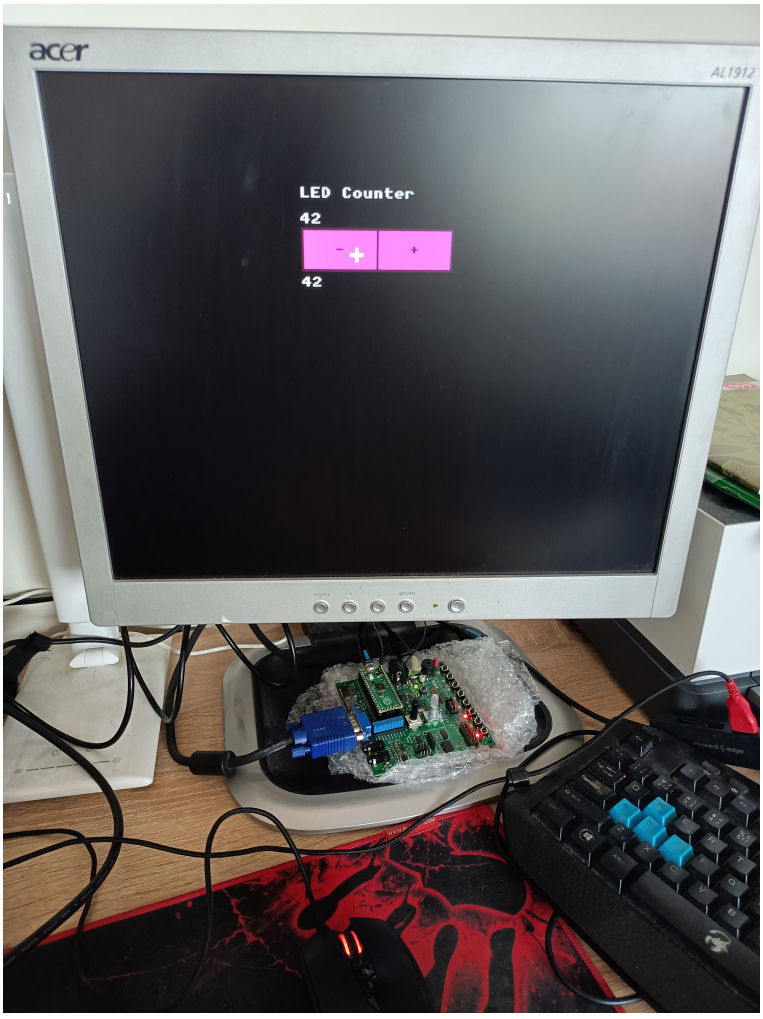
V první stránce je využita pro datové vazby datová struktura `data_s`, která obsahuje informace o tom, zda je LED zapnuta, vybrána pro nastavování svítivosti a její momentální nastavená svítivost. Tyto data jsou následně svázána s grafickými prvky, které jsou vytvořené ve funkci `CreatePage`. Grafické prvky v této stránce jsou rozloženy do tří `StackLayout` rozložení. V levém se nachází přepínače pro zapínaní nebo vypínaní jednotlivých LED. V prostředním rozložení se nacházejí přepínače, které vybírají jaké LED bude nastavována intenzita svitu. V posledním pravém rozložení se nachází `ProgressBar` prvky, které zobrazují aktuální svítivost jednotlivých LED. Inicializační a aktualizační funkce této stránky zabezpečují převod napětí z potenciometru na desce, které je poté převedeno vybraným LED jako intenzita jejich svitu.

Implementace této stránky se nachází na výpisku kódu C.8. Stránka byla rozdě-

lena na tři jmenné prostory: `model`, `view` a `controller`. Fotografie této stránky se nachází na obrázku 5.2.

Stránka 2

Druhá stránka obsahuje ve svém datovém modelu data o momentálních hodnotách počítadel z rotačního enkodéru a prvku `Stepper` na obrazovce. Kombinace hodnot těchto počítadel je datově svázána s grafickým prvkem `Label`, který zobrazuje tuto hodnotu uživateli. Uživatel dokáže inkrementovat nebo dekrementovat hodnotu počítadel, buď to pomocí rotačního enkodéru přímo na vývojové desce, nebo použitím grafického prvku `Stepper`. Uživateli je také umožněno zadat hodnotu počítadla přímo, využitím grafického prvku `Entry`.



Obr. 5.3: Fotografie druhé stránky programu KitControl

Kontrolní část této stránky inicializuje a aktualizuje hodnotu počítadla z rotačního enkodéru a rozsvěcuje LED podle aktuální hodnoty počítadel.

Implementace této stránky se nachází na výpisku kódu C.9. Stránka byla opět rozdělena na tři jmenné prostory: `model`, `view` a `controller`. Fotografie této stránky se nachází na obrázku 5.3.

6 Výsledky studentské práce

Výsledkem práce je funkční framework RP GUI. Tento framework lze použít jako nástroj, pro návrh aplikací s grafickým uživatelským rozhraním, pro vývojovou desku Rpi Pico Kit. Framework spolu s deskou podporují generaci 8bitového barevného VGA video signálu, zpracování uživatelského vstupu z myši a klávesnice skrze USB rozhraní, zápis a čtení dat z SD karty. Pro generování obrazového signálu, framework využívá knihovnu PicoVGA [7] nebo PicoQVGA [8], která obstarává veškeré časovaní a operace s daty, potřebné pro generování obrazu na monitoru. Tyto knihovny k tomu využívají PIO blok pro výstup dat z mikrokontroléru do monitoru a DMA řadič pro přesun dat z frame bufferu do PIO bloku. K práci s USB rozhraním byla zvolena knihovna TinyUSB [10], která umožňuje zpracovávat uživatelský vstup z klávesnice a myši. Z klávesnice je možné zpracovávat 6 stisknutých kláves najednou. Myš generuje údaje o 5 stisknutých tlačítkách a relativní změně polohy. Data získaná z klávesnice přichází ve formě scancode kódů, které jsou následně pomocí LUT převáděny na ASCII znaky.

Pro usnadnění práce s knihovnami PicoQVGA a PicoVGA, které jsou základem pro RP GUI framework, bylo vytvořeno rozhraní `IVGA`. Toto rozhraní nejenže práci usnadňuje, ale také předchází vytváření logických chyb při programování. Na to využívá datové struktury namísto primitivních datových typů při volání funkcí. Rozhraní umožňuje vykreslování geometrických objektů a textu s volitelnými barvami. Barvy lze volit pomocí výčtového typu `Color`, který je oproti makrům používaným v knihovnách výrazně přehlednější. Definice funkcí, které rozhraní implementuje, se nemění v závislosti na použité VGA knihovně. Proto mezi těmito knihovnami je možné libovolně přepínat bez nutnosti úprav kódu.

Pro zpracování uživatelského vstupu byla vybrána knihovna TinuUSB, kterou platforma Raspberry Pi Pico podporuje. Knihovna podporuje veškeré typy USB zařízení, pro tuto práci je využíváno její možnosti zpracovávat data z HID zařízení. Pro zpracování dat z myši a klávesnice jsou vytvořeny dvě funkce ve zdrojovém souboru `hid_handler.cpp`, který se nachází v knihovně `gui_lib`, ve složce `util`. Funkce pro zpracování vstupu z klávesnice dokáže detekovat, zda bylo tlačítko právě stisknuto, nebo uvolněno, případě zda jeho stisknutí stále trvá . Podle toho ukládá data do struktury `kbd`, která je definována v hlavičkovém souboru `tusb_ data.hpp`. Data v tomto hlavičkovém souboru, jsou definována jako statická, tudíž jsou inicializována pouze jednou za běh programu. Programátor k nim může přistupovat na libovolném místě v jeho kódu, po zahrnutí hlavičkového souboru `tusb_data.hpp`. Celkově lze zpracovávat šest stisknutých kláves najednou. Funkce, pro zpracování dat z myši, zpracovává data o pěti stisknutých tlačítkách a relativním pohybu myši na osách X a Y. Tyto data jsou umístěny v struktuře `mouse`. Knihovna podporuje také použití

USB hubu. Díky němu lze mít připojenou zároveň klávesnici a myš. Tím se rozšiřují možnosti využití RP GUI.

Framework, pro vytváření grafických uživatelských rozhraní, s názvem RP GUI, umožňuje programátorovi dynamicky vytvářet grafické prvky a rozmisťovat je na monitoru. Grafických prvků přístupných programátorovi je celkově 13. Momentálně jsou implementovány tyto prvky:

- `Button` dokáže po jeho stisknutí vykonávat jakoukoliv programátorem vytvořenou funkci.
- `CheckBox` je zaškrtávací políčko, které zobrazuje svůj stav typu `bool`.
- `Circle` je kruh na obrazovce monitoru.
- `Entry` umožňuje uživateli zadávat hodnoty skrze klávesnici.
- `Label` dokáže zobrazovat libovolný text na obrazovce.
- `Line` vykresluje čáru na monitoru.
- `Polyline` vykresluje spojitě čáry na monitoru.
- `ProgressBar` znázorňuje úroveň postupu mezi dvěma zvolenými hodnotami.
- `RadioButton` dává uživateli možnost zvolit z možných variant pouze jednu.
- `Rectangle` je obdélník na obrazovce.
- `Sprite` třída určena na vykreslování obrázků.
- `Stepper` umožňuje uživateli inkrementovat nebo dekrementovat hodnotu proměnné.
- `Switch` umožňuje uživateli přepínat proměnnou typu `bool`.

Pro interakci s prvky, na které lze klikat, byl implementován kurzor. Kurzor, když se nachází nad grafickým prvkem, který má přiraženou funkci, tuto funkci spustí po kliknutí levého tlačítka myši. Kurzor má momentálně implementován reakci pouze na kliknutí levého tlačítka myši. Proces kliknutí je rozdělen do 3 stádií: `Pressed`, `Released` a `Clicked`. Grafické prvky na které lze klikat kurzorem, využívají pro svou funkcionalitu systém událostí.

Aby bylo možné slučovat jednotlivé grafické prvky dohromady, a tak je jednoduše v programu vykreslovat, byly vytvořeny rozložení `AbsoluteLayout` a `StackLayout`. Do těchto rozložení lze umístit prvky a nastavit tak jejich absolutní pozici a velikost vzhledem k obrazovce. Tyto rozložení lze následovně přidávat do pomocné třídy `Page`. Tato třída je určena pro shlukovaní rozložení a jako jedinou ji lze přidat do hlavní třídy `MainApp`.

Třída `MainApp` obsahuje všechny přidané instance třídy `Page`, mezi kterými je možné přepínat a tím určovat jaká stránka se má vykreslovat. Dále obsahuje funkci `Update`, která vykreslí obsah stránky na obrazovku, aktualizuje pozici kurzoru a zpracovává události spojené s kliknutím myši. Pro správné fungování RP GUI je po programátorovi požadováno, aby periodicky volal funkci `Update`. Framework dokáže tuto aktualizaci vykonávat na zvoleném jádru procesoru. Je doporučováno,

aby volání funkce `Update` probíhalo na prvním jádru procesoru, jelikož druhé jádro využívají VGA knihovny na generování video signálu. Toto lze nastavit ve hlavičkovém souboru `settings.hpp`. Zde lze také nastavovat velikost kurzoru, rozlišení obrazovky a volbu, zda se má při vykreslování čekat na VSYNC signál.

Aby bylo možné jednotlivé grafické prvky vykreslovat na monitor, musí všechny dědit vlastnosti ze třídy `VisualElement`. Tato třída, kromě toho že uchovává informace o poloze prvku a jeho hranicích, obsahuje čistě virtuální funkci `Draw`. Tuto funkci musí jednotlivé grafické prvky vždy implementovat. Hranice prvků jsou nutné pro jednoduché určení zda se kurzor nachází právě nad tímto grafickým prvkem. Dále jsou potřebné, aby bylo možné, při změně jeho polohy zmazat jeho starou polohu z frame bufferu.

Framework dokáže pracovat i s obrázky, které jsou v souborovém formátu `.bmp`. Tyto obrázky lze načíst a uložit do grafického prvku `Sprite`. Načíst tyto obrázky dokáže programátor dvěma způsoby. Prvním způsob spočívá v tom, že se vygeneruje hlavičkový souboru s definicí obrázku. Tato generace se děje během překladu programu a provádí se pomocí nástroje `RaspPicoImg`, který byl upravený, aby vložil všechny data a parametry obrázku do datové struktury. Z této datové struktury lze pak vytvořit instanci třídy `Sprite`. Druhý způsob je dynamický, to znamená, že načítání obrázků probíhá za běhu programu. Programátor dokáže určit jaký obrázek se má načíst z SD karty.

Přímou práci s SD kartou zabezpečuje knihovna no-OS-FatFS-SD-SPI-RPi-Pico. Tato knihovna k tomu využívá SPI rozhraní a dva DMA kanály. Aby byla usnadněna práce s touto knihovnou a zabezpečeno správné zacházení s SD kartou, byla vytvořena knihovna `SDwrapper_lib`. Tato knihovna zodpovídá za inicializaci komunikace s SD kartou a také její ukončování. Programátor dokáže otevírat soubory na SD kartě po poskytnutí jejich jména a cesty k nim. Při otevírání si může volit módy, v jakém budou soubory otevírány, např. jenom pro čtení. Aby bylo zajištěno zapsání dat do složek, jsou všechny otevřené složky automaticky zavřeny při ukončovaní komunikace s SD kartou.

Za účelem demonstrace využití SD karty byla vytvořena knihovna `rplog_lib`, která nejenže bude užitečná pro účely demonstrace, ale je také užitečným nástrojem pro programátora. Tato knihovna má za úkol zapisování dat buďto do souborů na SD kartě, nebo jiného výstupního komunikačního rozhraní. Programátorovi umožňuje volit důležitost zpráv, které budou logovány. Celkově jsou k dispozici tyto úrovně důležitosti: `Trace`, `Debug`, `Information`, `Warning`, `Error`, `Critical` a `None`. Programátor může vytvářet instance třídy `Logger`. Do těchto instancí lze přidávat vícero výstupních lokalit, do kterých budou zvolena data logována společně, využitím logovací funkce třídy. Logovat data lze i bez nutnosti vytváření instance třídy `Logger` a to využitím členských funkcí.

Na ukázku celkové funkcionality RP GUI, byly vytvořeny dva programy. Prvním je hra Pong. Tato klasická 2D arkádová hra simuluje stolní tenis. Hráči hýbou pomocí klávesnice pálky, kterými se usilují odrážet kuličku zpátky na protivníka. Tato hra pro své vykreslování na obrazovce využívá grafické prvky z RP GUI. Dále hra využívá zpracovaná data z klávesnice pro změnu polohy pálek na obrazovce.

Druhým ukázkovým programem je program KitControl, určen na ovládání vývojové desky RPi Pico Kit. Zde uživatel dokáže prostřednictvím klávesnice a myši ovládat základní uživatelské rozhraní nacházející se na desce. Tento program je rozdělen na dvě stránky GUI. Na první stránce dokáže uživatel skrze GUI zapínat LED na desce a nastavovat jim intenzitu svitu pomocí potenciometru na desce. Na druhé stránce GUI dokáže uživatel inkrementovat nebo dekrementovat hodnotu počítadla. Tuto inkrementaci lze provádět třemi způsoby: přímím zadáním hodnoty skrze klávesnici použitím prvku `Entry`, použitím grafického prvku `Stepper` nebo rotačním enkodérem na desce. Hodnota počítadla je zobrazována pomocí LED na vývojové desce.

Závěr

Cílem této bakalářské práce bylo vytvořit rozšíření, ve kterém bude možné vyvářet grafické uživatelské rozhraní pro Raspberry Pi Pico Kit. Toto rozšíření je založeno na knihovnách PicoQVGA a PicoVGA, které generují obrazový signál. Dále knihovně TinyUSB, která je využívána pro zpracovávaní uživatelského vstupu skrze USB rozhraní a knihovně no-OS-FatFS-SD-SPI-RPi-Pico, která obsluhuje SD kartu skrze SPI rozhraní. Toto rozšíření bylo nazváno RP GUI.

Zdrojový kód RP GUI je nahrán v GitHub repositáři a je dostupný Raspberry komunitě na odkazu <`https://github.com/K3fas/rp-pi-pico-GUI`>.

První část práce se věnuje obeznámení se s platformami, na kterých je tato práce založena, s jejich hardwarovými a softwarovými možnostmi. Těmito platformami jsou Raspberry Pi Pico [2] a vývojová deska RPi Pico Kit [1].

Následující část se věnuje popisu softwarových knihoven, které jsou v této práci využité. Zde jsou popsány knihovny PicoQVGA [8], PicoVGA [7], TinyUSB [10] a knihovna no-OS-FatFS-SD-SPI-RPi-Pico [13]. Je zde také vysvětlen princip jejich funkčnosti a jaké části platformy Raspberry Pi Pico potřebují pro svou práci.

Práce, nejdříve využívala pouze knihovnu PicoQVGA, protože byla jednodušší oproti knihovně PicoVGA. Na ní bylo odzkoušeno, jakými způsoby bude možné uživateli zobrazovat informace na obrazovce monitoru a prozkoumáno jakým způsobem funguje. Později se přešlo na knihovnu PicoVGA, která podporuje vyšší rozlišení a má širší možnosti práce s textem. Obě tyto knihovny bylo potřeba přepsat do CMake projektové struktury. Při přepisování knihovny PicoVGA vznikli menší problémy s kompatibilitou knihoven, které tato knihovna využívala. Tyto problémy byly vyřešeny a nyní programátor může používat libovolnou VGA knihovnu pro vykreslování.

Pro zpracování dat z klávesnice a myši byla využita knihovna TinyUSB. Zde byla rozšířena implementace funkcí na zpracování dat z klávesnice a myši. Tato zpracovaná data jsou ukládána do datových struktur, které jsou následně dostupné v celém kódu. Integrace TinyUSB knihovny do práce se neobešla bez svých problémů. Nejdříve se objevil problém, že připojené USB zařízení nebyly napájeny. To bylo vyřešeno přivedením napětí 5 V na VBUS pin mikrokontroléru z diody D2, pomocí kabelové propojky. Dále byl řešen problém, že při připojení klávesnice nebo myši do USB hubu, program havaroval. Tento problém se nakonec odstranil sám s aktualizací SDK pro Raspberry. Po vyřešení těchto problémů lze zpracovávat uživatelská data, získávána z klávesnice a myši skrze USB rozhraní.

Knihovna no-OS-FatFS-SD-SPI-RPi-Pico umožňuje využití microSD karty pro načítání nebo zapisování dat. Při využívání této knihovny nenastaly žádné problémy. Po nastavení správných GPIO a přiřazení nevyužitých DMA kanálů, se podařilo zprovoznit komunikaci s SD kartou.

Další částí práce byl návrh a implementace nástroje, který umožňuje tvorbu aplikací s grafickým uživatelským prostředím. Tento nástroj (framework) byl nazván RP GUI. Framework využívá pro svou činnost `IVGA` rozhraní, které zaobaluje knihovny PicoQVGA a PicoVGA. Využívá také knihovnu TinyUSB pro získávání vstupu od uživatele. Základem RP GUI jsou jeho grafické prvky. Momentálně jsou implementovány prvky: `Button`, `CheckBox`, `Circle`, `Entry`, `Label`, `Line`, `Polyline`, `ProgressBar`, `RadioButton`, `Rectangle`, `Sprite`, `Stepper`, `Switch`. Tyto prvky lze dále umisťovat do rozložení, které přesně definují jejich pozici na obrazovce. K dispozici jsou rozložení typu: `AbsoluteLayout` a `StackLayout`. Tyto rozložení lze následně přidávat do třídy `Page`, kterou je následně potřeba přidat do hlavní třídy `MainApp`.

V třídě `MainApp` je implementována funkce `Update`, která vykreslí obsah na monitor a zpracuje data získaná z pohybu myši. Po programátorovi je požadováno, aby periodicky volal funkci `Update`. Obsluha vykreslování pro framework dokáže probíhat na libovolném jádru mikrokontroléru. Je však doporučováno vykreslování na prvním jádru, jelikož na druhém pracuje obsluha generování VGA signálu. Interakci s prvky provádí uživatel skrze kurzor. Po stisknutí levého tlačítka myši, kurzor spouští funkci grafického elementu, který je právě umístěn pod ním.

Programátor dokáže pracovat i s obrázky a to využitím třídy `Sprite`. Zde může definovat jaký obrázek chce použít, a to dvěma způsoby. První statický způsob, využívá nástroje `RaspPicoImg`, který vygeneruje hlavičkový soubor obsahující definici obrázku. Druhý dynamický způsob, načte obrázek přímo z microSD karty za běhu programu.

Pro zpřehlednění práce s daty, které GUI prezentuje uživateli, byly implementovány datové vazby. Pomocí těchto datových vazeb bylo zmenšeno množství kódu, který je potřebný pro aktualizaci dat. Díky nim můžou být datové proměnné přímo svázány s grafickými prvky, a tak vzájemně zajišťovat modifikaci způsobu vykreslování nebo hodnotu dat samotných.

Implementace RP GUI zabrala nejvíce času a to přibližně 150 hodin. Design, který framework implementuje, se postupně vyvíjel v průběhu implementace grafických prvků a to tak, že byly postupně přidávány nové prvky a funkcionality do RP GUI. Finálním designem, jakým RP GUI disponuje, lze vytvářet aplikace, které lze svou architekturou přirovnat k architektuře MVC (Model-View-Controller). Tj. rozdělení aplikace na 3 části: pohledy (UI), kontroléry (zpracování dat) a datové struktury.

Následující část práce se zaobírá obsluhou a prací s SD kartou. K tomu byla zvolena knihovna no-OS-FatFS-SD-SPI-RPi-Pico. Pro zjednodušení práce s touto knihovnou, byla vytvořena knihovna `SDwrapper_lib`. Tato knihovna se stará o navazování komunikace s SD kartou, správné otevíraní souborů, správné uzavíraní souborů a ukončování komunikace s SD kartou. Vytvořením této mezivrstvy bylo

ulehčeno programátorovi přistupování k datům na SD kartě a je zabráněno nesprávnému zacházení s SD kartou, které může vést ke ztrátě dát nebo až k poškození souborového systému.

Další vytvořený nástroj `rplog_lib` tuto mezivrstvu používá při své činnosti. Tento nástroj je datový logger, který umožňuje programátorovi zaznamenávat data jak na SD kartu, tak i na ostatní komunikační rozhraní. K tomuto může programátor využívat funkce definované v třídě `Logger`. Dále dokáže vytvářet instance třídy `Logger`, do kterých lze přidávat vícero míst pro zaznamenávání dat (soubory nebo výstupní proudy dat). Tímto lze jednoduše zaznamenávat data na vícero míst jedním voláním logovací funkce.

Jakožto ukázku celkové funkcionality nástroje RP GUI, byla vytvořena 2D arkádová hra Pong. Tato hra využívá pro vykreslování svých součástí prvky definované v RP GUI a využívá zpracovaná data z klávesnice pro hraní hry. Další ukázkou je program KitControl, kterým lze ovládat LED na vývojové desce a to pomocí GUI. Uživatel pomocí této aplikace dokáže nastavovat intenzitu svitu LED na desce a tyto hodnoty zobrazovat na obrazovce. Dále lze inkrementovat hodnotu počítadla použitím myši, klávesnice nebo rotačního enkodéru na desce. Hodnota tohoto počítadla je znázorněna pomocí LED na desce.

Budoucím možným rozšířením práce by mohlo být například: implementování nových grafických prvků jako je Slider - pro změnu hodnoty proměnné posunováním kurzoru, přidání rozložení, které umožňuje rolování prvků v něm uložených. Tímto by bylo možné přidávat neomezené množství prvků na obrazovku a stále by existovala možnost jak k jednotlivým prvkům přistupovat v GUI. K tomu by bylo vhodné implementovat datové šablony, které umožní programátorovi definovat, jak se budou poskytnutá data zobrazovat na obrazovce. Dalším rozšířením je umožnění dekódování speciálních znaků na klávesnici nebo využití stlačeni pravého tlačítka myši.

Literatura

[1] PONČÁK, M., *Inovace laboratorních úloh kurzu Vestavné systémy.* VUT FEKT Brno: 2022 [cit. 27. 12. 2022], Dostupné z URL: <http://hdl.handle.net/11012/204867>

[2] Raspberry Pi, [Online katalogový list], *Raspnerry Pi Pico product brief,* Rev. 7/2022 [cit. 27. 12. 2022] Dostupné z URL: <https://datasheets.raspberrypi.com/pico/pico-product-brief.pdf>

[3] Raspberry Pi, [Online katalogový list], *Getting started with Raspberry Pi Pico :C/C++ development with Raspberry Pi Pico and other RP2040-based microcontroller boards,* Rev. 11/2022 [cit. 27. 12. 2022] Dostupné z URL: <https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf>

[4] Raspberry Pi, [Online katalogový list], *RP2040 Datasheet: A microcontroller by Raspberry Pi,* Rev. 11/2022 [cit. 27. 12. 2022] Dostupné z URL: <https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf>

[5] Raspberry Pi, [Online katalogový list], *Raspberry Pi Pico Datasheet: An RP2040-based microcontroller board,* Rev. 6/2022[cit. 27. 12. 2022] Dostupné z URL: <https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf>

[6] Raspberry Pi, [Online katalogový list], *Raspberry Pi Pico C/C++ SDK: Libraries and tools for C/C++ development on RP2040 microcontrollers,* Rev. 11/2022 [cit. 27. 12. 2022] Dostupné z URL: <https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf>

[7] Němeček M. *PicoVGA - displej VGA/TV na Raspberry Pi Pico,* Rev. 1.0, 6/2021 [cit. 27. 12. 2022]. Dostupná z URL <http://www.breatharian.eu/hw/picovga/index_en.html#property>

[8] Němeček M. *PicoQVGA - minimalistický displej QVGA na Raspberry Pi Pico,* Rev. 1.0, 9/2021 [cit. 27. 12. 2022]. Dostupná z URL <http://www.breatharian.eu/hw/picoqvga/index_en.html#property>

[9] Wren6991 *PicoDVI - projekt na generováni DVI signálu pro RP2040* [cit. 23. 4. 2023]. Dostupná z URL <https://github.com/Wren6991/PicoDVI>

[10] TinyUSB *TinyUSB: open-source cross-platform USB Host/Device stack for embedded systems* [cit. 27. 12. 2022]. Dostupná z URL <https://docs.tinyusb.org/en/latest/>

[11] Raspberry Pi, *Ukázka kódu pro práci s HID zařízeními* [cit. 27. 12. 2022]. Dostupné z URL <https://github.com/raspberrypi/pico-examples/tree/master/usb/host/host_cdc_msc_hid>

[12] WikiLabs *Kódování klávesnice* [cit. 27. 12. 2022] Dostupné z URL <https://wiki.dcae.pub.ro/images/thumb/0/0c/Scancodes.png/800px-Scancodes.png>

[13] carlk3 *Knihovna pro práci s SD kartou na platformě Raspberry Pi Pico* [cit. 23. 4. 2023]. Dostupné z URL <https://github.com/carlk3/no-OS-FatFS-SD-SPI-RPi-Pico>

[14] ELM-ChaN *Návratové hodnoty API funkcí* [cit. 23. 4. 2023]. Dostupné z URL <http://elm-chan.org/fsw/ff/doc/rc.html>

Seznam symbolů a zkratek

GUI	Graphical user interface - Grafické uživatelské rozhraní
SDK	Software developement kit - Systémový vývojový nástroj
API	Aplplication programming interface - Rozhraní pro programovaní aplikací
SRAM	Static random access memory - Statická RAM paměť
FLASH	Zkratka pro elektricky programovatelnou nevolatilní paměť
SPI	Serial peripheral interface - Sériovo periferní rozhraní
MOSI	Master Out - Slave In
MISO	Master In - Slave Out
CRC	Cyclic Redundancy Check - Cyklická redundantní kontrola
QSPI	Quad serial peripheral interface - SPI protokol se čtyřmi komunikačními linkami
UART	Universal asynchronous receiver-transmitter - Univerzální asynchornní přímač-vysílač
I2C	Inter-integrated circuit - Multi-master počítačová sériová sběrnice
USB	Universal serial bus - Univerzální sériová sběrnice
PWM	Pulse width modulation - Pulzně šířková modulace
PCM	Pulse-code modulation - Pulzně kódová modulace
PLL	Phase-locked loop - Fázový závěs
GPIO	General-purpose input/output - Univerzální vstupno/výstupní pin
PIO	Programmable I/O - Programovatelné vstupno-výstupní rozhraní
HID	Human interface device - Třída zařízení lidského rozhraní
CDC	Communication device class - Třída zařízení pro komunikaci
MSC	Mass storage device class - Třída velkokapacitních paměťových zařízení

Seznam příloh

A Platforma Raspberry Pi Pico a vývojová deska RPi Pico Kit

V této kapitole přílohy se nachází obeznámení s platformou Raspberry Pi Pico a k ní vytvořené vývojové desce RPi Pico Kit.

A.1 Popis platformy Raspberry Pi Pico

Série Raspberry Pi Pico je rodina rozměrově malých, výkonných a všestranných mikrokontrolérových desek, které se dostaly na trh začátkem roku 2021. Platforma staví na použití 32bitového RP2040 mikrokontroléru, který navrhla firma Raspberry Pi Foundation samotná. Architektura RP2040 je postavena kolem procesoru Arm Cortex-M0+, který je vysoce výkonný, zatímco má velmi nízkou spotřebu. Při úsporném režimu se spotřeba pohybuje méně jako 1 mW. Procesor obsahuje 264 kB interní RAM paměti a podporuje externí úložiště paměti o velikosti až 16 MB. Dále podporuje širokou škálu vstupně/výstupních (I/O) možností jako například I2C, SPI a jedinečně i programovatelnou I/O jednotku (PIO). Vzhledem jeho poměrně malé ceně, která se pohybuje kolem 150 Kč, je jeho potenciální využití obrovské. Verze Raspberry Pi Pico W přináší navíc i bezdrátový LAN modul (2.4 GHZ 802.11n), tudíž jej dělá výborného kandidáta na použití v IoT aplikacích a v aplikacích, které vyžadují bezdrátovou komunikaci.

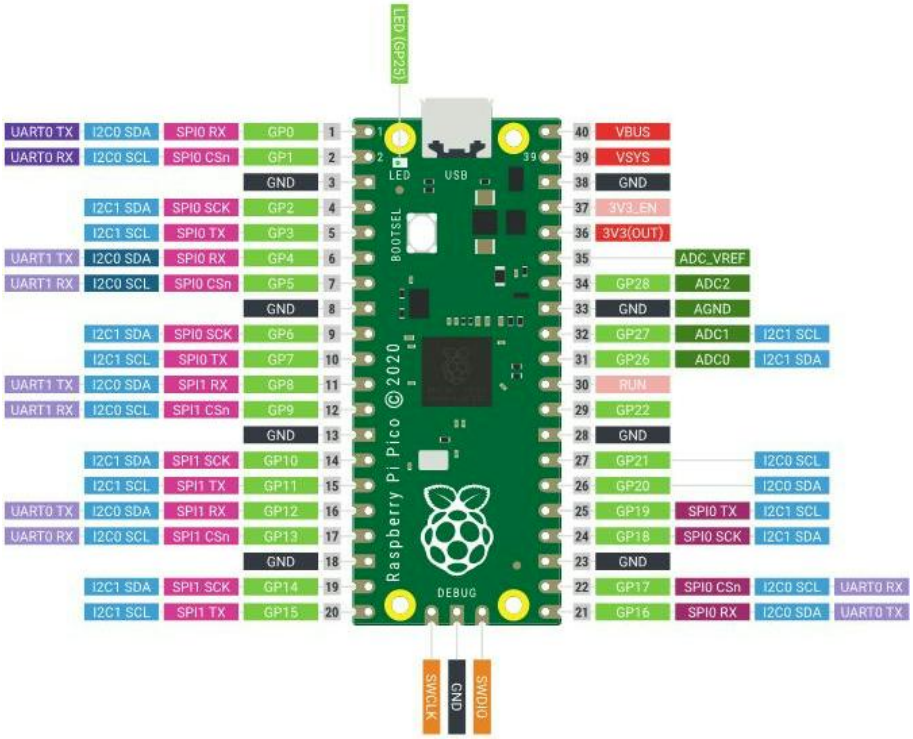
Platforma pracuje se dvěma SDK (Software developement kit). Jedním pro MicroPython a druhým pro C/C++, které bude pro účely této práce využíváno. SDK je pravidelně aktualizováno a je kvalitně dokumentováno. Dále obsahuje velikou řadu demonstračních projektů od jednoduchého rozsvícení ledky až po komunikaci po některé ze sběrnic platformy.

Komunita kolem platformy Raspberry je ohromně velká a publikuje různé projekty, blogy, a dokonce organizuje i semináře, kde si lidé můžou vyzkoušet pracovat s výrobky Raspberry.

A.1.1 Popis mikrokontroléru

Jak bylo výše popsáno, architektura mikrokontroléru je postavené kolem 32bitového procesoru Arm CortexM0+. Procesor je dual-core s flexibilní frekvencí až do 133 MHz a zároveň nízkou spotřebou v spánkovém a nečinném režimu. Disponuje velkou 256 kB statickou RAM (SRAM) pamětí přímo v čipu a podporuje až 16 MB externí FLASH[1] paměti pro kterou je dedikována nástavba sériově periferní sběrnice

[1]Název pro nevolatilní paměť, která dokáže být elektronicky přepisována.



Obr. A.1: Rozložení vývodů desky Raspberry Pi Pico [5]

QSPI. Architektura pozůstává ze:

- Řadiče pro přímý přístup do paměti (DMA)
- Interpolátoru a celočíselné děličky
- Programovatelného regulátoru napětí s nízkým úbytkem napětí (LDO)
- Dvou generátorů s fázovým závěsem (PLL) pro generování frekvencí jádra procesoru

Periferie mikrokontroléru:

- 2 x UART
- 2 x kontrolér pro sériově periferní rozhraní (SPI)
- 2 x kontrolér pro I2C
- 16 x kanál s pulzně šířkovou modulací (PWM)
- 1 x kontrolér pro univerzální sériovou sběrnici (USB) verze 1.1
- 8 x programovatelných stavových automatů pro vstupně/výstupní operace

Rozhraním mikrokontroléru je 30 univerzálních vstupně/výstupních (GPIO) pinů z kterých 4 dokážou být použity jako analogový vstup. Díky svému velkému počtu periferií může být čip použit na řízení motorů, zvukový výstup a pro tuto práci důležitý obrazový výstup [1].

A.1.2 Paměť mikrokontroléru

Mikrokontrolér RP2040 má vestavěnou ROM a SRAM paměť a je schopný komunikovat s externí FLASH pamětí skrze QSPI rozhraní.

Paměť ROM

Obsah ROM paměti o velikosti 16 kB je dán výrobcem při výrobě a nelze jej měnit. Tato paměť obsahuje: startovací sekvenci mikrokontroléru, bootloader[2] pro nahrávaní programu skrze USB rozhraní do RAM nebo FLASH paměti, rutiny pro obsluhu FLASH paměti, užitkové knihovny jako například knihovna pro rychlé operace z desetinnými čísly - fast floating point. Adresa začátku paměti je 0x00000000. Boot sekvence mikrokontroléru je blíže popsána zde [4].

Paměť SRAM

Na čipu je celkově dostupných 264 kB SRAM paměti, která je fyzicky rozdělena do šesti bank. Toto fyzické rozdělení paměti do bank zvyšuje propustnost paměti tím, že umožňuje sekvenční čtení z vícero bank najednou. Propustnost paměti je parametr, který určuje počet bitů, jaký lze přečíst/zapsat do paměti za jednu sekundu. Nejsou zde žádné restrikce o tom, jaké data můžou být uložena v jednotlivých bankách. Pro software se paměť jeví jako kontinuální paměť o velikosti 264 kB. Adresovací prostor paměti začíná na adrese 0x2000000 [1].

Paměť FLASH

Mikrokontrolér dokáže přistupovat k externí FLASH paměti skrze QSPI rozhraní. Paměť FLASH se pak mikrokontroléru jeví, jako jeho interní paměť. Jednou z výhod FLASH paměti je to, že umožňuje procesoru vykonávat kód bez toho, aby kopíroval kód z FLASH paměti do jeho interní RAM paměti, z které by byl program následně vykonáván. Tudíž umožňuje vykonávat kód přímo z FLASH paměti bez nutnosti kopírováni. Tento princip se odborně nazývá execute in place (XIP) [1].

A.1.3 Periferie mikrokontroléru

V této části budou blíže popsány periferie, jimiž disponuje mikrokontrolér a jsou podstatné pro tuto práci.

[2]Malý program, který se zapne při startu.

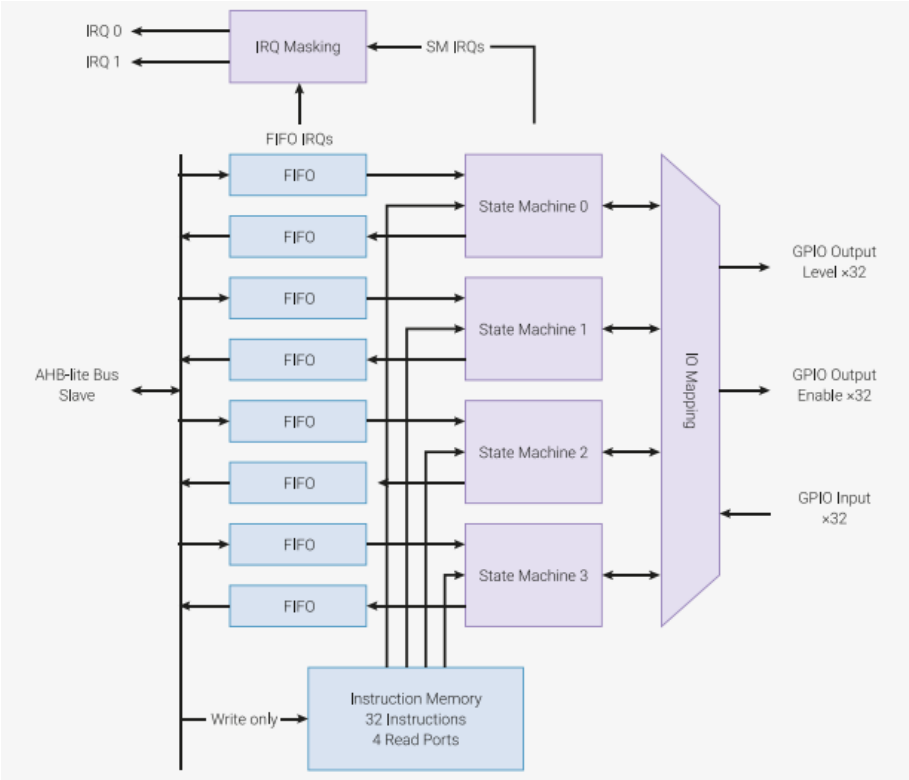
USB

Mikrokontrolér RP2040 obsahuje USB 2.0 řadič a fyzickou vrstvu verze 1.1. Řadič dokáže pracovat jako:

- Full Speed device s přenosovou rychlostí 12 MB za sekundu
- Host, který dokáže komunikovat jako Low (1,5 MBps) nebo Full Speed device se zařízeními připojenými k USB hubu.

Hardware USB kontroléru řeší nízkoúrovňový USB protokol, což znamená, že hlavním úkolem programátora je správně nakonfigurovat kontrolér samotný a zabezpečit místo v paměti pro výměnu dat mezi kontrolérem a procesorem přes vyrovnávací paměti [1].

Programmable I/O



Obr. A.2: Diagram PIO bloku [4]

Programovatelná vstupně/výstupní periferie (PIO) je programována obdobně jako procesor, avšak má svou vlastní malou instrukční sadu. V mikrokontroléru se nacházejí dva PIO bloky, které dokážou nezávisle na sobě vykonávat kód. Tudíž umožňují pracovat se vstupně/výstupními daty bez přímé účasti procesoru. Na rozdíl od generického procesoru jsou PIO stavové automaty specificky navrhnuty pro práci se vstupy a výstupy. Každý blok je připojen na datovou sběrnici a dokáže

generovat systémové přerušení. Jak je vidět na obrázku A.2, uvnitř PIO bloku má každý stavový automat k dispozici dvě FIFO, a celkově jsou v jednom bloku čtyři stavové automaty. Architektura PIO je zaměřena na precizní časování a determinismus. Každý PIO blok zabírá na mikrokontroléru místo přibližně stejné jako standardní IO blok. Výhoda PIO je v tom, že se dá různě konfigurovat. Kupříkladu, aby vykonávala činnost onoho standardního vstupně/výstupního bloku. Výstupy PIO dokážou ovládat všechny GPIO mikrokontroléru [1].

DMA Modul

Direct Memory Access (DMA) modul umožňuje přenos dat v systémové paměti bez účasti procesoru a tím zvyšuje jeho efektivitu. Efektivita je zvýšena tím, že procesor nemusí zabezpečovat a čekat na přenos dat, ale může dále pokračovat ve vykonávání kódu. Kontrolér má dvě separátní připojení na datovou sběrnici, jednu pro čtení dat a druhou pro zápis dat. To umožňuje zápis a čtení 32 bitů z paměti každý hodinový cyklus DMA modulu. Generátor adres zabezpečuje správné spárování adres pro čtení a zápis. Modul obsahuje 12 nezávislých kanálů pro přenos dat. To znamená, že může probíhat 12 operací přenosů dat. Existují 3 typy přenosu dat:

- Z paměti do paměti - DMA modul co nejrychleji přenese data mezi dvěma oblastmi v paměti RAM.
- Z periferie do paměti - periferie signalizuje DMA modulu, že má nová data. DMA modul tyto data přečte a uloží je do paměti RAM.
- Z paměti do periferie - periferie signalizuje DMA modulu, že potřebuje přenést data. DMA přečte data z paměti RAM nebo FLASH a zapíše je do periferie.

Všech 12 přenosových kanálů je kontrolováno řídícími a stavovými registry. Datová velikost přenosu může být 32, 16 anebo 8 bitů a je nastavitelná pro každý kanál nezávisle [1].

A.2 Vývojové prostředky

Aby bylo možné platformu programovat musí být nainstalováno několik programů, a to, překladač pro architekturu Arm (Arm GNU Toolchain) a CMake spolu s Visual Studio Code. Desku lze programovat v operačních systémech Linux, MacOS, a také Windows. Postup instalace je pro každý operační systém trochu odlišný a je detailně popsaný v katalogovém listu [3]. Ze zmíněného dokumentu budou dále čerpány informace pro tuto sekci. Raspberry Pi Pico lze programovat v jazyce C/C++, jazyku symbolický instrukcí (Arm assembly) a také v jazyku Python.

A.2.1 SDK pro C/C++

Sada vývojových nástrojů (SDK) umožňuje tvoření aplikací pro určité softwary. Pro platformu Raspberry Pi Pico existuje tato sada nástrojů v jazyce C/C++. SDK je navrženo tak, aby bylo programovací prostředí familiární jak pro programátory znalé v embedded oblasti, tak pro programátory, kteří jsou v této oblasti nováčky. Jsou podporovány standardní C/C++ knihovny, ale také API[3] pro přístup k hardwaru uvnitř mikrokontroléru RP2040. SDK dále podporuje práci z knihovnami vyšší úrovně, které se vypořádávají s nastavováním časovačů, nebo komunikací přes USB rozhraní. Pro přímý přístup k hardwaru, programátor nalezne plně zadokumentované hlavičkové definice pro ovládaní registrů v mikrokontroléru. Podrobná dokumentace tohoto SDK je k nalezení v katalogovém listu [6]. V čase psaní práce byly použity verze SDK 1.4.0 a později 1.5.0.

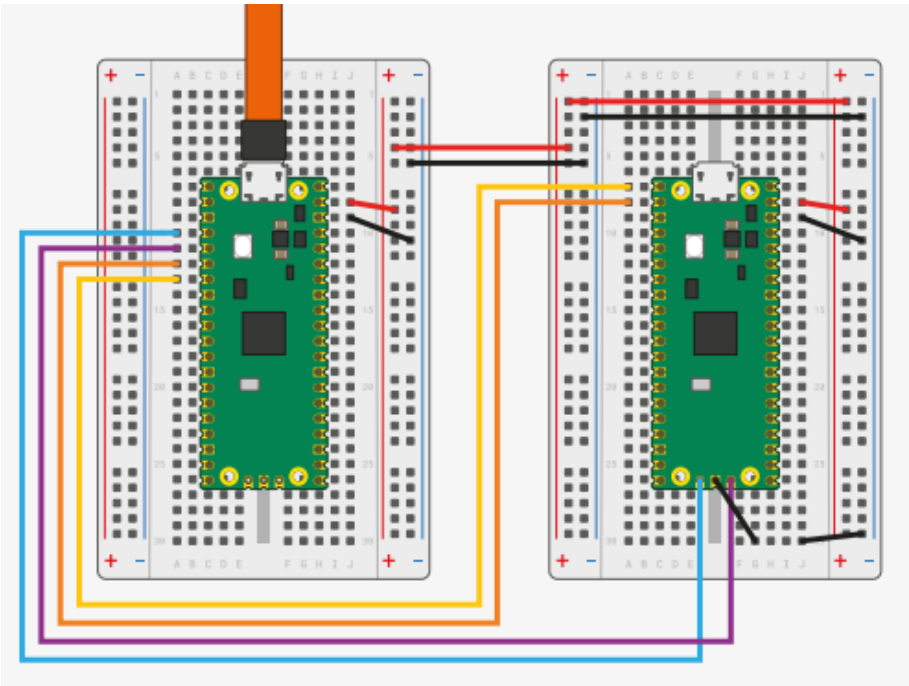
A.2.2 Vývojové prostředí

Pro vývoj a odlaďování programů je použito vývojové prostředí Visual Studio Code. VS Code je volně přístupný program vydaný společností Microsoft, lze používat na všech operačních systémech, které jsou kompatibilní s SDK pro Raspberry Pi. V prostředí VS Code musí být nainstalovány doplňky, které umožní práci s platformou Raspberry Pi Pico. Jsou nimi: CMake, CMake Tools, C/C++ a Cortex-Debug. Dále v nastaveních CMake Tools je potřeba pro správnou funkci nastavit proměnou *CMake: Generator* na hodnotu *Unix Makefiles*. Nesmí se zapomenout nakonfigurovat prostředí samotné a to tak, že se nastaví cesta k programům OpenOCD, Arm GDB toolchain a také cesta k C/C++ SDK. Postup je detailně popsán v [3].

Pro nahrávaní a ladění programů na desku bude použito SWD (Serial Wire Debug) rozhraní a na jeho obsluhu je potřeba nainstalovat program OpenOCD. Jak ho nainstalovat, je k nalezení v katalogovém listu [3]. Pro převod dat mezi počítačem a deskou Raspberry Pi Pico musí být použita další deska, ve které musí být nahrán program `picoprobe.utf2`.

Na obrázku A.3 lze vidět zapojení, potřebné pro ladění programu na Raspberry Pi desce. Jedna deska bude využívaná pro ladění programu a na druhé desce bude spuštěn program. Pro ladění programu lze použít program OpenOCD, k tomuto programu se dá připojit program GDB skrze který, bude možné ladit program běžící na desce. GDB umožňuje programátorovi ladit pomocí bodů přerušení (breakpoint), krokováním kódu po řádcích a čtením dat z registrů a paměti.

[3] Rozhraní pro tvoření aplikací



Obr. A.3: Propojení dvou Raspberry pro ladění SW [3]

A.3 Popis vývojové desky RPi Pico Kit

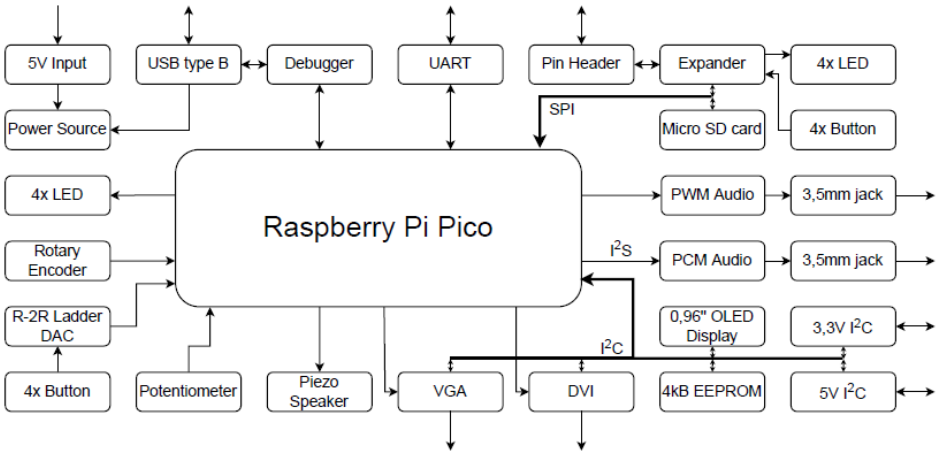
Vývojová deska RPi Pico Kit byla navržena pro modernizaci laboratorních úloh v předmětu BPC-MIC. K platformě Raspberry Pi Pico proto byli připojeny komponenty, které jsou zapotřebí v laboratorních úlohách a umožňují tak využití funkcí platformy Raspberry Pi Pico. Informace v této sekci jsou čerpány převážně z dokumentace pro tuto desku [1].

Architektura desky je založena na platformě Raspberry Pi Pico, která je modulárně připojena k desce skrze dutinové lišty. Zapojení platformy je rozděleno do bloků a je vidět na obrázku A.4.

Pro interakci s uživatelem je na desce celkem: 8x LED, 8x tlačítko, rotační enkodér a rotační potenciometr. Dále periferie pro vstup a výstup z desky jsou:

- USB typu B
- dva 3,5 mm konektory pro PWM a PCM[4] audio
- piezoelektrický měnič
- UART
- 3,3 V a 5 V I2C sběrnice
- DVI port
- VGA port
- OLED Display 0,96"

[4]Pulzná kódová modulace



Obr. A.4: Bloková schéma zapojení HW platformy [1]

Nachází se zde také paměť 4 kB EEPROM se kterou lze komunikovat skrze sběrnici I2C. Také existuje možnost na desku připojit microSD kartu, která komunikuje po sběrnici SPI.

Na desce je dále osazen druhý RP2040 čip, který je přímo spojený s platformou Raspberry Pi Pico a umožňuje tak nahrávání kódu a ladění (debugger) bez potřeby druhé Pico desky [1].

A.3.1 Napájení desky

Vývojová deska pro svou funkci potřebuje napájení 5 V, které je možné realizovat třemi různými způsoby. Pro tuto práci je nejčastěji využíváno napájení pomocí USB konektoru typu B, který se nachází přímo na desce. Toto zapojení umožňuje využívat SWD rozhraní zabudované v desce a tím je možné odlaďovat běžící programy nebo je nahrávat do mikrokontroléru.

Další možnost napájení je připojení desky Raspberry Pi Pico, která je osazena na desce, přímo k počítači skrze USB Mini-A konektor. U tohoto zapojení však není možné využívat SWD rozhraní. Poslední možností je napájení dedikovaným DC konektorem s napětím 5 V.

Všechny možnosti napájení jsou ochráněný proti nadproudu PTC pojistkami F1 a F2 se jmenovitou hodnotou 500 mA. Dále jako ochrana proti přepólování plní funkci Schottkyho dioda, přes kterou je každý zdroj napětí přivedený na +5 V bod. Zapojení zároveň slouží jako ochrana proti protečení zpětného proudu při zapojení více než jedné z možnosti napájení.

Komponenty na desce jsou napájeny ze stabilizátoru napětí 3,3 V. Integrované obvody pro generování zvuku jsou napájeny z vlastního stabilizátoru 3,3 V. Mikrokontrolér na vývojové desce obsahuje vlastní napájecí zdroj, takže je připojen přímo

na 5 V. Napájení je signalizováno zelenou LED, která má označení LED9 [1].

A.3.2 Odlaďování programů

Jako nástroj pro odlaďovaní je použitý mikrokontrolér RP2040, který je osazen přímo na vývojové desce. Mikrokontrolér je napojený na desku skrze SWD rozhraní umožňující ladění a nahrávání kódu.

Aby bylo možné napájet, nahrávat, odlaďovat a komunikovat s deskou skrze USB port, je k mikrokontroléru připojený UART port vývojové desky. Pro jeho činnost, je nutné, aby v něm byl nahrán program PicoProbe. Ten se do mikrokontroléru nahraje skrze USB rozhraní. Je zde implementována ochrana proti neúmyslnému přepsání dat tak, že je nutné před nahráváním do mikrokontroléru, zkratovat propojku JP1 v čase restartu mikrokontroléru.

Mikrokontrolér, určen pro odlaďování, generuje při svojí činnosti impulzy, které jsou znázorněny oranžovou LED10 na desce. Generované impulzy jsou příliš krátké na to, aby je bylo možné vidět pouhým okem, a proto jsou přiváděny na monostabilní klopný obvod, který prodlužuje dobu jejich trvání [1].

A.3.3 Základní uživatelské rozhraní

Pro základní interakci s uživatelem jsou na desce čtyři LED, čtyři tlačítka, rotační enkodér a potenciometr. Celkově je na desce až osm LED a osm tlačítek, ale zbylé čtyři tlačítka a LED jsou k vývojové desce připojeny skrze expander, který komunikuje po sběrnici SPI [1].

LED

Čtyři červené LED jsou připojeny k vývojové desce na pinech GPIO6 až GPIO9. Na těchto pinech jsou připojeny i jiné součásti desky. Diody jsou ovládané skrze MOSFET-N tranzistory, aby se předešlo nadměrnému zatěžování výstupů procesoru [1].

Rotační enkodér

Rotační enkodér má celkově 3 výstupy A, B a tlačítko. Výstupy A a B jsou připojeny na piny GPIO21 a GPIO22. Tlačítko je připojeno na pin GPIO11. Tyto vývody je nutno v softwaru připojit na pull-down rezistory pro správnou funkci [1].

Rotační potenciometr

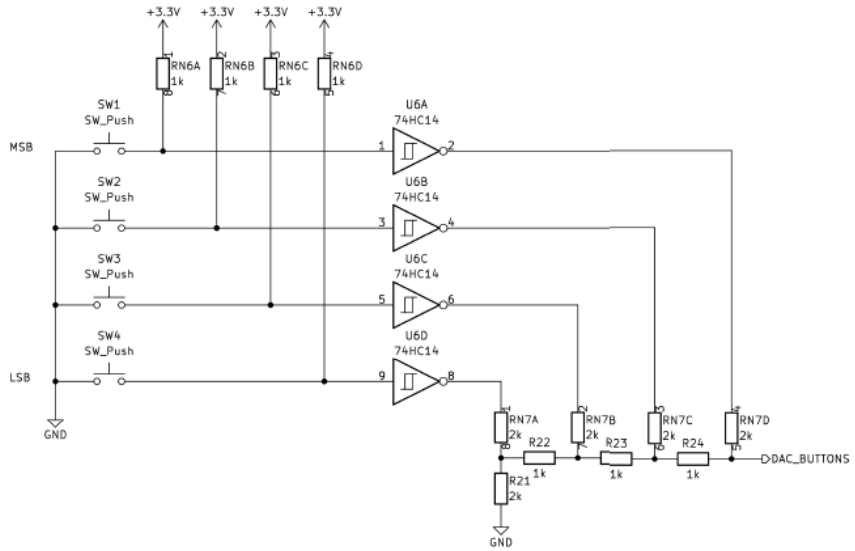
Potenciometr je připojen k pinu GPIO27, který je jedním z analogových vstupů na vývojové desce. Vstup potenciometru je filtrován filtrem 1. řádu [1].

Tlačítka na analogovém vstupu

Připojením každého tlačítka zvlášť na digitální vstup, by byli značně omezeny ostatní části desky, kterými by byli obsazeny potřebné piny. Proto jsou všechny čtyři tlačítka připojena na jeden analogový vstup skrze R-2R rezistorovou síť A.5. Tato rezistorová síť umožňuje detekci jakékoliv kombinace zmáčknutí tlačítek. Jednotlivé napěťové příspěvky tlačítek se sčítají na vstup analogového pinu GPIO28. Napětí na tomto pinu pak lze dopočítat z rovnice:

$$U_{OUT} = U_{REF} \frac{x}{2^N} \tag{A.1}$$

Kde referenční napětí $U_{REF} = 3,3V$, počet bitů D/A převodníku $N = 4$ a x je binární kódovaní stavu tlačítek [1].



Obr. A.5: Připojení tlačítek na rezistorovou síť R-2R [1]

A.3.4 UART rozhraní

Komunikace přes UART rozhraní na desce je realizována dvěma způsoby, kolíkovými propojkami, JP2 a JP3, lze nastavit, zda budou UART porty vývojové desky komunikovat s debuggerem nebo s externím zařízením na portu J3.

Signalizace komunikace je provedena stejně jako u debuggeru a signalizují ji dvě oranžové LED, LED10 a LED11 [1].

A.3.5 SPI sběrnice

Na SPI sběrnici jsou připojeny dvě zařízení, a to expander bitových vstupů a výstupů a slot pro microSD kartu. SPI sběrnice má připojený vstup MISO (Master In - Slave Out) na pinu GPIO4, výstup MOSI (Master Out - Slave In) na pinu GPIO7 a hodinový signál na pinu GPIO6 [1].

Expander

Expander použitý na desce obsahuje osm vstupně/výstupních portů, které mají každý interní pull-up rezistor. Na expander jsou připojeny čtyři diody na výstupech GP0 až GP3 a čtyři tlačítka na vstupech GP4 až GP7. Povolovací pin CS je připojen k vývojové desce na pinu GPIO26 [1].

SD karta

Možnost přístupu k SD kartě je v 1bitovém režimu. Karta samotná se chová jako SPI zařízení a má povolovací pin připojený k desce na pinu GPIO5 [1].

A.3.6 Obrazový výstup

Na desce se nachází dvě rozhraní pro obrazový výstup, jsou nimi VGA a DVI.

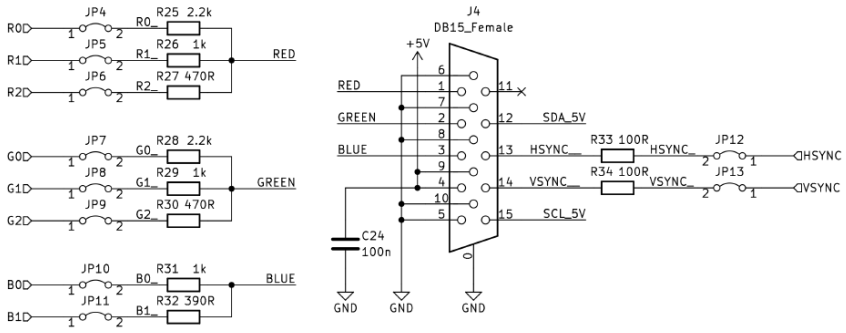
VGA rozhraní

Vývojová deska je navržena s podporou 8bitového obrazového výstupu, který je připojen na piny GPIO11 až GPIO18. Synchronizační signály HSYNC a VSYNC jsou připojeny ke konektoru na pinech GPIO19 a GPIO20. Piny obrazového výstupu musí být po sobě jdoucí, protože je to tak implementováno v použité softwarové knihovně [8]. Rozlišení jednotlivých barev je následující:

- 3bitová červená složka
- 3bitová zelená složka
- 2bitová modrá složka

Připojení jednotlivých barevných výstupů k monitoru je vidět na obrázku A.6.

VGA rozhraní je taktéž připojeno na I^2C sběrnici proto, aby bylo možné číst EDID (Extended Display Information Data) metadata monitoru. Ke sběrnici je rozhraní připojeno na adresách 0x37, 0x49, 0x50, 0x59. Relevantní pro metatada je



Obr. A.6: Zapojení obrazového VGA výstupu [1]

jenom adresa 0x50. Čtení metadat však nemusí fungovat, kvůli zkratu, který vznikal připojením monitoru. Tato závada je blíže popsána v práci [1].

DVI rozhraní

DVI rozhraní na desce umožňuje realizaci digitálního obrazového výstupu. Výstupní digitální signál se skládá ze tří diferenciálních datových výstupů D0 až D2 a signálem CLK, který je také diferenciální. Tyto signály jsou připojeny na pinech GPIO12 až GPIO19. Z DVI rozhraní je možné číst EDID metadata stejně jako u VGA rozhraní [1].

B Obsah elektronické přílohy

Elektronická příloha obsahuje framework v adresáři `lib/gui_lib` a ostatní podpůrné knihovny v adresáři `lib`. Ukázky se nacházejí ve složce `examples`. V adresářích nejsou uváděny soubory `CMakeList.txt`. Práce je k nalezení také v github repositáři na adrese <`https://github.com/K3fas/rp-pi-pico-GUI`>.

- `/ rp-pi-pico-GUI` ..... kořenový adresář přiloženého archivu
  - `examples` ..... adresář obsahující demo programy
    - `Pong` ..... demo hra využívající framework RP GUI
    - `KitControl` ..... demo aplikace na interakci s Rpi Pico Kit
  - `lib` ..... adresář obsahující knihovny
    - `gui_lib` ..... framework pro tvorbu aplikací s GUI
      - `config` ..... soubory obsahující nastavení knihovny
      - `include` ..... hlavičkové soubory knihovny
        - `common` ..... běžné soubory pro funkci RP GUI
        - `elements` ..... třídy grafických prvků RP GUI
        - `event` ..... třídy event systému RP GUI
        - `layouts` ..... třídy rozložení RP GUI
        - `pages` ..... třídy stránek RP GUI
        - `include.hpp` .....
        - `rp_core.hpp` ..... jádro RP GUI
        - `rpgui.hpp` ..... hlavní hlavičkový soubor knihovny
      - `src` ..... zdrojové soubory knihovny
      - `util` ..... pomocné nástroje knihovny
        - `hid_handler.cpp` ..... zdrojový soubor pro zpracovávaní USB vstupu
        - `IVGA.hpp` ..... interface pro knihovny PicoQVGA a PicoVGA
        - `Timers` ..... třída pro zaznamenávaní časových intervalů
        - `tusb_data.hpp` .. hlavičkový souboru obsahující zpracovaná USB data
    - `qvga_lib` ..... knihovna pro generování obrazu
    - `rplog_lib` ..... knihovna pro logování dat
      - `include` ..... hlavičkové soubory knihovny
        - `rp_logger.hpp` ..... datový logger pro kit
      - `src` ..... zdrojové soubory knihovny
    - `SDwrapper_lib` ..... knihovna pro komunikaci s SD kartou
      - `FatFs_SPI` ..... knihovna pro komunikaci s SD kartou
      - `include` ..... hlavičkové soubory knihovny
        - `SDWraper.hpp` ..... wrapper pro knihovnu FatFs_SPI
      - `src` ..... zdrojové soubory knihovny
    - `vga_lib` ..... knihovna pro generování obrazu
  - `tools` ..... adresář obsahující pomocné nástroje VGA knihoven
  - `video_exmaples` ..... videa používání demo aplikací
  - `pico_extras_import.cmake` ..... skript pro vyhledání pico extras modulu
  - `pico_sdk_import.cmake` ..... skript pro vyhledání C/C++ SDK

C Výpisky kódu

Tato příloha obsahuje výpisky kódu, které se nevešly nebo nebylo vhodné dát do hlavní části práce.

```
void Update()
{
    tuh_task();
    hid_app_task();
    if (drawOnCore1)
    {
        IVGA::ICore1Exec(updateOnCoreX);
    }
    else
    {
        updateOnCoreX();
    }
}

void updateOnCoreX()
{
    processMouseInput();
    processMouseMovement();
    drawCursor();
    drawPage();
    drawCursor();

    if (waitVSync)
    {
        IVGA::IWaitVSync();
    }
}
```

Výpis C.1: `Update` funkce RP GUI

```
void processMouseMovement()
{
    if (!HID::mouse::moved)
        return;
    MainApp::_cursor.pos.x += HID::mouse::mousePos[0] *
   sensitivity * 0.020;
    MainApp::_cursor.pos.y += HID::mouse::mousePos[1] *
   sensitivity * 0.020;

    if (MainApp::_cursor.pos.x > WIDTH)
        MainApp::_cursor.pos.x = WIDTH;
    if (MainApp::_cursor.pos.y > HEIGHT)
        MainApp::_cursor.pos.y = HEIGHT;
    if (MainApp::_cursor.pos.x < 0)
        MainApp::_cursor.pos.x = 0;
    if (MainApp::_cursor.pos.y < 0)
        MainApp::_cursor.pos.y = 0;

    MainApp::_cursor.SetBounds(Bounds(MainApp::_cursor.pos.x,
    MainApp::_cursor.pos.y, cursorSize * 2 + 1, cursorSize *
   2 + 1));

    HID::mouse::mousePos[0] = 0;
    HID::mouse::mousePos[1] = 0;

    HID::mouse::moved = false;
}
```

Výpis C.2: Funkce pro aktualizace pozice kurzoru

```
1  void processMouseInput()
2  {
3      // lmb only
4      // handle new pressed key
5      if (HID::mouse::mouseKeys[0] == true && MainApp::
       _clickState == MainApp::clickState::none)
6      {
7          MainApp::_clickState = MainApp::clickState::pressed;
8
9          auto pos = _cursor.GetBounds();
10         auto event = MouseEvent<MouseEventType>(
       MouseEventType::Pressed, Point(pos.x, pos.y));
11         MainApp::_mouseHandler.Post(event);
12     }
13
14     // handle released key
15     if (HID::mouse::mouseKeys[0] == false && MainApp::
       _clickState == MainApp::clickState::pressed)
16     {
17         MainApp::_clickState = MainApp::clickState::none;
18
19         auto pos = _cursor.GetBounds();
20         auto eReleased = MouseEvent<MouseEventType>(
       MouseEventType::Released, Point(pos.x, pos.y));
21         auto eClicked = MouseEvent<MouseEventType>(
       MouseEventType::Clicked, Point(pos.x, pos.y));
22         MainApp::_mouseHandler.Post(eReleased);
23         MainApp::_mouseHandler.Post(eClicked);
24     }
25 }
```

Výpis C.3: Funkce pro vytváření událostí kliknutí

```
if (report->keycode[i])
  { /*  exist in previous report means
        the current key is holding */
    if (find_key_in_report(&prev_report,report->keycode[i]))
        {}
        else {
    /* not existed in previous report means
             the current key is pressed */
          bool const is_shift = report->modifier
            & (KEYBOARD_MODIFIER_LEFTSHIFT
            | KEYBOARD_MODIFIER_RIGHTSHIFT);
          uint8_t ch = keycode2ascii[report->keycode[i]]
                    [is_shift ? 1 : 0];
          HID::kbd.pressedKeys[i] = ch;
        continue;
      }}
    else {
      // Key released handler
      if (!prev_report.keycode[i])
        break;
      for (uint8_t j = 0; j < 5; j++) {
        if (report->keycode[j] != prev_report.keycode[j]) {
          // j == key released pos
          for (uint8_t k = j; k < 5 - j; k++) {
            HID::kbd.pressedKeys[i] =
                        HID::kbd.pressedKeys[k + 1];
          }
          break;
}}}}
  prev_report = *report;
```

Výpis C.4: Zpracování vstupu z klávesnice

```
static hid_mouse_report_t prev_report = {0};
//------------- button state  -------------//
uint8_t button_changed_mask =
    report->buttons ^ prev_report.buttons;
if (button_changed_mask) {
  report->buttons &MOUSE_BUTTON_LEFT ?
    HID::mouse::mouseKeys[0] =1 : HID::mouse::mouseKeys[0]
   =0;
  report->buttons &MOUSE_BUTTON_MIDDLE ?
    HID::mouse::mouseKeys[1] =1 : HID::mouse::mouseKeys[1]
   =0;
  report->buttons &MOUSE_BUTTON_RIGHT ?
    HID::mouse::mouseKeys[2] =1 : HID::mouse::mouseKeys[2]
   =0;
  report->buttons &MOUSE_BUTTON_BACKWARD ?
    HID::mouse::mouseKeys[3] =1 : HID::mouse::mouseKeys[3]
   =0;
  report->buttons &MOUSE_BUTTON_FORWARD ?
    HID::mouse::mouseKeys[4] =1 : HID::mouse::mouseKeys[4]
   =0;
  HID::mouse::clicked = true;
}

// Process moving cursor
HID::mouse::mousePos[0] += report->x;
HID::mouse::mousePos[1] += report->y;
HID::mouse::moved = true;

prev_report = *report;
```

Výpis C.5: Zpracování vstupu myši

```
PerformTest(View *elements[], uint16_t count, uint16_t runs,
    const std::string& elementName)
{
    speedResults results;

    results.total = time_us_64();
    // Counts total time
    for (size_t j = 0; j < runs; j++)
    {
        for (size_t i = 0; i < count; i++)
        {
            elements[i]->Draw();
        }
    }

    results.total = time_us_64() - results.total;
    results.average = results.total / runs;
    results.averageElement = results.average / count;
    // Run again for other calculations
    for (size_t j = 0; j < runs; j++)
    {
        auto start = time_us_64();
        for (size_t i = 0; i < count; i++)
        {
            elements[i]->Draw();
        }
        auto iteration = time_us_64() - start;

        if (iteration > results.max)
        {
            results.max = iteration;
        }
        if (iteration < results.min)
        {
            results.min = iteration;
        }
    }

    results.midrange = (results.max - results.min) / 2;
    LogResults(results, elementName);
    return results;
}
```

Výpis C.6: Funkce na testování rychlosti RP GUI

```
int main()
{
    rpgui::core::MainApp::Init();
    stdio_init_all();

    page1Id = page1::view::CreatePage();
    page2Id = page2::view::CreatePage();

    page2::controller::Init();
    core::MainApp::SelectPageAt(1);0

    while (1)
    {
        switchPages();
        if (core::MainApp::GetSelectedPageId() == page1Id)
        {
            page1::controller::Update();
        }
        if (core::MainApp::GetSelectedPageId() == page2Id)
        {
            page2::controller::Update();
        }
        rpgui::core::MainApp::Update();
    }
}
```

Výpis C.7: Hlavní funkce aplikace `KitControl`

```
namespace page1::model {
    inline static struct kit_s {
        bool led[4];
        bool state[4];
        double progress[4];
    } data;
} // namespace page1::model
namespace page1::view {
    type::ID CreatePage() {
        auto left = new layout::StackLayout(Point(20, 60),
   Width(120), Height(400));
        auto middle = new layout::StackLayout(Point(180, 60),
    Width(120), Height(400));
        auto right = new layout::StackLayout(Point(340, 60),
   Width(120), Height(400));

        left->AddElement(new ui::Label("LED On/Off"));
        left->AddElement(new ui::Switch(Width(0), Height(32),
    data.state[0]));
        /* Other state switches ... */
        middle->AddElement(new ui::Label("LED Control"));
        middle->AddElement(new ui::Switch(Width(0), Height
   (32), data.led[0]);
        /* Other control switches ... */
        right->AddElement(new ui::Label("Led brightness"));
        right->AddElement(new ui::ProgressBar(Width(0),
   Height(32), data.progress[0], 0, 255));
        /* Other brightness progress bars ... */
        auto page = new Page();
        page->AddLayout(left);
        page->AddLayout(middle);
        page->AddLayout(right);
        return core::MainApp::AddPage(page);
    }
} // namespace page1::view
namespace page1::controller {
    void Init() {
        /* Init gpio ... */ }
    void Update() {
       /* Update LED output ... */ }
} // namespace page1::controller
```

Výpis C.8: První stránka aplikace `KitControl`

```
namespace page2::model {
    inline static struct data_s {
        int encoder = 0;
        int counter = 0;
        ui::Label *counter_lbl;
        data_s &operator++() {
            ++counter;
            return *this;
        }
        data_s &operator--() {
            --counter;
            return *this;
        }
    } data;
} // namespace page2::model
namespace page2::view {
    type::ID CreatePage()
    {
        auto middle = new layout::StackLayout(Point(180, 60),
    Width(120), Height(400));
        middle->AddElement(new ui::Label("LED Counter"));
        auto lbl = new ui::Label("0");
        data.counter_lbl = lbl;
        middle->AddElement(lbl), Margin(0, 10, 0, 10);
        middle->AddElement(new ui::Stepper(Width(120), Height
   (32), data, Color::DarkGray, Color::Magenta));
        middle->AddElement(new ui::Entry(data.counter));

        auto page = new Page();
        page->AddLayout(middle);
        return core::MainApp::AddPage(page);
    }
} // namespace page2::view
namespace page2::controller {
    void Init() {
        /* Init Encoder PIO and GPIO ... */
    }
    void Update() {
        /* Update LED output ... */
    }
} // namespace page2::controller
```

Výpis C.9: Druhá stránka aplikace `KitControl`

```
add_library(gui_lib INTERFACE
        ${CMAKE_CURRENT_LIST_DIR}/include/rpgui.hpp)

set(use_picoqvga false)

... ... ...

if(use_picoqvga)
        target_link_libraries(gui_lib INTERFACE
        qvga_lib)
else()
        target_compile_definitions(gui_lib INTERFACE PICOVGA)
        target_link_libraries(gui_lib INTERFACE
                                vga_lib)
endif()
```

Výpis C.10: Ukázka přepínaní knihoven PicoVGA a PicoQVGA

```
cmake_minimum_required(VERSION 3.13)

add_executable(program)

qvga_generate_img_header(program ${CMAKE_CURRENT_LIST_DIR}/
   img/Peter8.bmp Peter8Img)

... ... ...
```

Výpis C.11: Ukázka využití nástroje `RaspPicoImg`