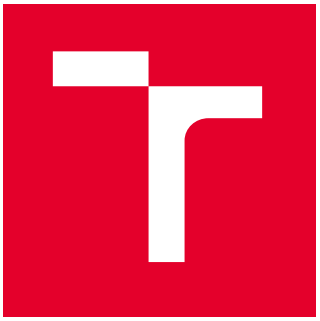


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

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

LABORATORNÍ ÚLOHY PRO MIKROKONTROLÉR RP2040

LABORATORY ASSIGNMENT FOR MICROCONTROLLER RP2040

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Filip Ševčík**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Tomáš Macho, Ph.D.**

SUPERVISOR

BRNO 2024

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: Filip Ševčík ***ID:*** 240440

Ročník: 3 ***Akademický rok:*** 2023/24

NÁZEV TÉMATU:

Laboratorní úlohy pro mikrokontrolér RP2040

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s architekturou procesorů ARMv6-M, ARM Cortex M0+ a mikrokontrolérem RP2040.
2. Připojte k mikrokontroléru RP2040 LCD displej a vyberte vhodnou knihovnu pro práci s displejem. Funkce pro práci s displejem zdokumentujte.
3. Navrhněte alespoň 5 laboratorních úloh pro cvičení do předmětu BPC-MIC demonstrující práci s binárním portem, A/D převodníkem, DMA kanálem a časovači.
4. Pro realizaci laboratorních úloh navrhněte a realizujte potřebné přizpůsobovací obvody.
5. Laboratorní úlohy realizujte a odlaďte.
6. Vytvořte zadání laboratorních úloh a návody pro studenty.

DOPORUČENÁ LITERATURA:

RP2040 Datasheet, Raspberry Pi Ltd, 2023

Termín zadání: 5.2.2024 ***Termín odevzdání:*** 22.5.2024

Vedoucí práce: Ing. Tomáš Macho, Ph.D.

Ing. Miroslav Jirgl, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tato práce se věnuje návrhem laboratorních úloh pro předmět zabývající se embedded systémy a mikrokontroléry. Úlohy byly navrženy pro vývojovou desku Raspberry Pi Pico s 32bitovým mikrokontrolérem od společnosti Raspberry Pi Foundation. V teoretické části je popsaná tato vývojová deska, mikrokontrolér RP2040, na němž je deska postavena a procesor Arm Cortex-M0+, kterým je mikrokontrolér RP2040 vybaven. Pro otestování laboratorních úloh byla navrhnuta deska plošných spojů. Práce také obsahuje zadání laboratorních úloh a návody pro studenty.

KLÍČOVÁ SLOVA

Raspberry Pi Pico, ARM Cortex M0+, mikrokontrolér RP2040, C, laboratorní úlohy

ABSTRACT

This thesis focuses on the design of laboratory exercises for a course dealing with embedded systems and microcontrollers. The exercises were designed for the Raspberry Pi Pico development board with a 32-bit microcontroller from the Raspberry Pi Foundation. The theoretical part describes this development board, the RP2040 microcontroller on which the board is based and the Arm Cortex-M0+ processor with which the RP2040 microcontroller is equipped. A printed circuit board was designed to test the laboratory exercises. The thesis also contains the laboratory assignments and instructions for the students.

KEYWORDS

Raspberry Pi Pico, ARM Cortex M0+, microcontroller RP2040, C, laboratory exercises

ŠEVČÍK, Filip. *LABORATORNÍ ÚLOHY PRO MIKROKONTROLÉR RP2040*. Bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2024. Vedoucí práce: Ing. Tomáš Macho, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Filip Ševčík
VUT ID autora:	240440
Typ práce:	Bakalářská práce
Akademický rok:	2023/24
Téma závěrečné práce:	LABORATORNÍ ÚLOHY PRO MIKRO-KONTROLÉR RP2040

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 ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Tomáši Machovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Bakalářská práce navazuje na semestrální práci a věnuje se návrhem laboratorních úloh pro 32bitový mikrokontrolér RP2040. Mikrokontroléry hrají klíčovou roli v moderní elektronice a jejich pochopení je důležité pro každého, kdo se chce zabývat vývojem nebo programováním embedded systémů nebo hardwaru obecně. Z tohoto důvodu je zařazen předmět Vestavné systémy a mikroprocesory (BPC-MIC) do studijního plánu v bakalářském programu na Fakultě elektrotechniky a komunikačních technologií VUT v Brně, na ústavu automatizace a měřicí techniky.

Pro získání praktických zkušeností se studenti v laboratorních cvičeních tohoto předmětu seznámí s programováním embedded systémů v jazyce C a assembleru. Cvičení stávajícího předmětu jsou postavena na 8bitovém mikrokontroléru MC9S08LH64 z řady HCS08 od firmy NXP (Freescale), avšak společnost Raspberry Pi Foundation v roce 2021 přišla s vlastním mikrokontrolérem RP2040 a vývojovou deskou PR Pi Pico, což nabízí možnost aktualizovat tato laboratorní cvičení.

Teoretická část této práce se zabývá architekturou procesorů ARM Cortex M0+, mikrokontrolérem RP2040 a vývojovou deskou Raspberry Pi Pico.

Dále je v této práci navrženo 6 úloh, ve kterých se studenti seznámí a prakticky si vyzkouší práci s binárními porty, s využitím přerušení, s hodinami reálného času (modul RTC), časovačem, A/D převodníkem, periferií PWM, DMA kanálem a komunikačním protokolem SPI.

Mimo jiné je cílem této práce pro realizaci laboratorních úloh navrhnout potřebné přizpůsobovací obvody. Aby bylo možné zobrazit výstupy z mikrokontroléru, jsou k vývojové desce připojeny led diody a LCD TFT displej a pro umožnění práce s tímto displejem je vybrána vhodná knihovna. V návodech pro vypracování laboratorních úloh je popsáno, jak přidat knihovnu do projektu a v přílohách je dokumentace, kterou studenti mohou využít pro práci s knihovnou.

Pro ovládání vstupně výstupních binárních portů jsou využity tlačítka a pro možnost využití A/D převodníku je k vývojové desce připojen potenciometr.

Jelikož je obsahem této práce i úloha na periferii PWM a pro představení studentům možností této periferie bylo zvoleno měření frekvence, je také k vývojové desce připojen generátor obdélníkového signálu realizovaný integrovaným obvodem TLC555.

K otestování přizpůsobovacích obvodů je součástí práce také návrh desky plošných spojů.

Pro programování mikrokontroléru RP2040 bylo zvoleno vývojové prostředí Visual Studio Code a jelikož tato práce má sloužit pro seznámení budoucích studentů s danou problematikou, text této práce se věnuje i postupu instalace vývojového prostředí a možnosti debugování.

1 Procesor ARM Cortex M0+

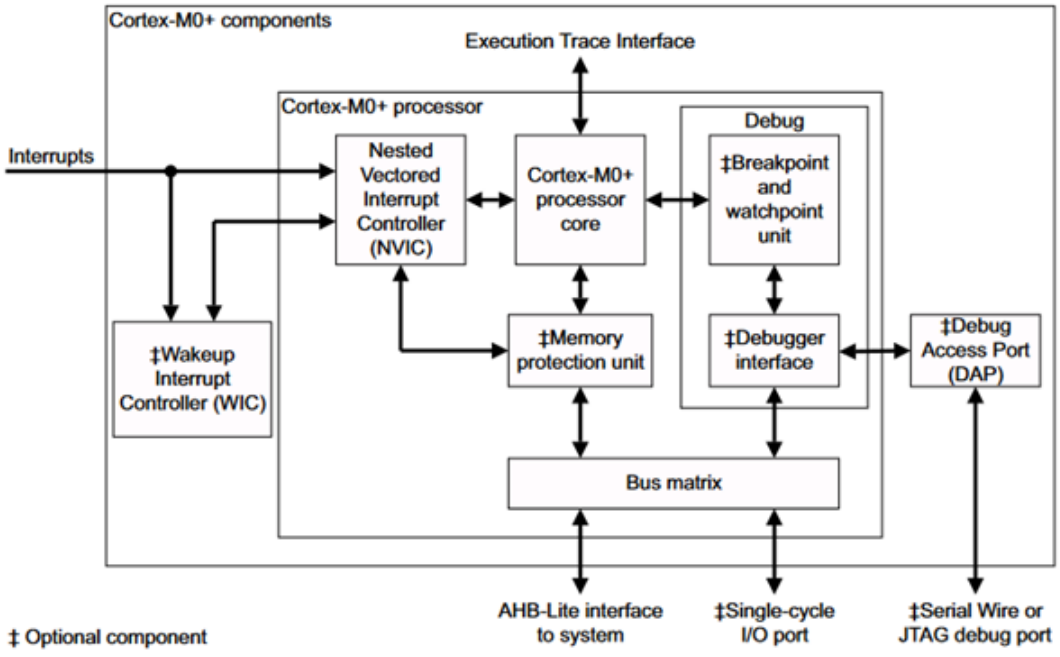
Tato teoretická kapitola popisuje Procesor ARM Cortex M0+, jeho architekturu a zařazení do rodin ARM Cortex.

1.1 Procesory ARM

Existují tři různé ARM Cortex rodiny: Cortex A, Cortex R a Cortex M. Rodina Cortex A má největší výpočetní výkon a je používána pro zařízení s operačním systém. Cortex R procesory slouží pro real-time aplikace, zatímco Cortex M jsou využívány v mikrokontrolérech. [1]

1.2 ARM Cortex M0+

ARM Cortex M0+ má nejmenší rozměry a nejmenší spotřebu z Cortex M procesorů. Je to 32bitový procesor s nízkou spotřebou a s dvoustupňovým řetězením instrukcí (pipelining). Je vhodný pro embedded aplikace.



Obr. 1.1: Blokové schéma procesoru ARM Cortex M0+ [2]

Procesor je implementován ARMv6-M architekturou, většina instrukcí má 16 bitů (Thumb), menší množství instrukcí má velikost 32 bitů (Thumb-2).

Bus matrix na schématu označuje sběrnici, která zajišťuje jádru procesoru a případně DAP (Debug Access Port) přístup ke vnější paměti, k integrovanému řadiči přerušení (NVIC) a k debugovacím nástrojům. Prioritu pro přístup ke sběrnici má ale procesor, aby debugování nezpůsobovalo zpoždění.

Informace v této kapitole jsou převzaty z [2], pokud není uvedeno jinak.

1.3 Výjimky

Slouží pro obsluhu neodkladných událostí. Výjimky naruší sekvenční vykonávání instrukcí a procesor vykoná kód asociovaný s danou výjimkou. Adresy pro vstup do kódů pro obsloužení výjimek jsou umístěny v tzv. *"vector table"* str. 193 [3].

Existují různé typy výjimek, mezi ně patří:

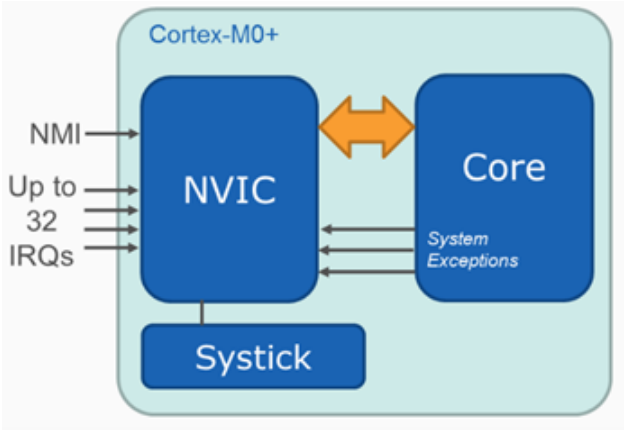
- Reset – Je to speciální typ přerušení, při němž se procesor nastaví do definovaného výchozího stavu. Má nejvyšší prioritu -3.
- Non-Maskable Interrupt (NMI) – nemaskovatelné přerušení slouží pro obsluhu kritických událostí, nelze jej zakázat Má druhou nejvyšší prioritu -2.
- HardFault – jsou to výjimky, které mohou nastat při chybě (dělení nulou). Mají třetí nejvyšší prioritu -1.
- Interrupt (IRQ) – jsou to přerušení, která jsou spojena s periferiemi (UART, GPIO, ...) nebo mohou být nastavená programem. NVIC (Nested Vector Interrupt Controller) vytváří rozhraní pro tato přerušení.

1.3.1 NVIC (Nested Vectored Interrupt Controller)

Řadič přerušení umožňuje 32 různých přerušení. Požadavek na přerušení může být detekován při určité hodnotě signálu nebo při nástupné/sestupné hraně. Přerušení mohou být povolena nebo zakázána zápisem do příslušných konfiguračních registrů. Přerušením lze programově přidělit priority. Počet priorit je určen výrobcem, RP2040 podporuje čtyři priority.

1.4 Model paměti

ARMv6-M je paměťově mapovaná architektura, používá 32bitový adresový prostor. Paměť je rozdělena na sekce které popisuje tabulka 1.1. Pořadí instrukcí v programu nemusí odpovídat pořadí operací přístupu k paměti. Procesor může pro zajištění větší efektivity změnit toto pořadí, například protože přístup k různým částem paměti může mít různé čekací doby nebo proto, že některé přístupy jsou spekulativní. Změna pořadí samozřejmě nesmí změnit celkové chování těchto instrukcí.



Obr. 1.2: Schéma připojení řadiče přerušení k jádru procesoru [4]

- Typ paměti Normal umožňuje procesoru změnit pořadí přístupů k paměti nebo provádět spekulativní čtení.
- Typ paměti Strongly-ordered zachová pořadí instrukcí vzhledem ke všem dalším instrukcím.
- Typ paměti Device zachová pořadí instrukcí vzhledem k instrukcím, které jsou v Device nebo Strongly-ordered části paměti.

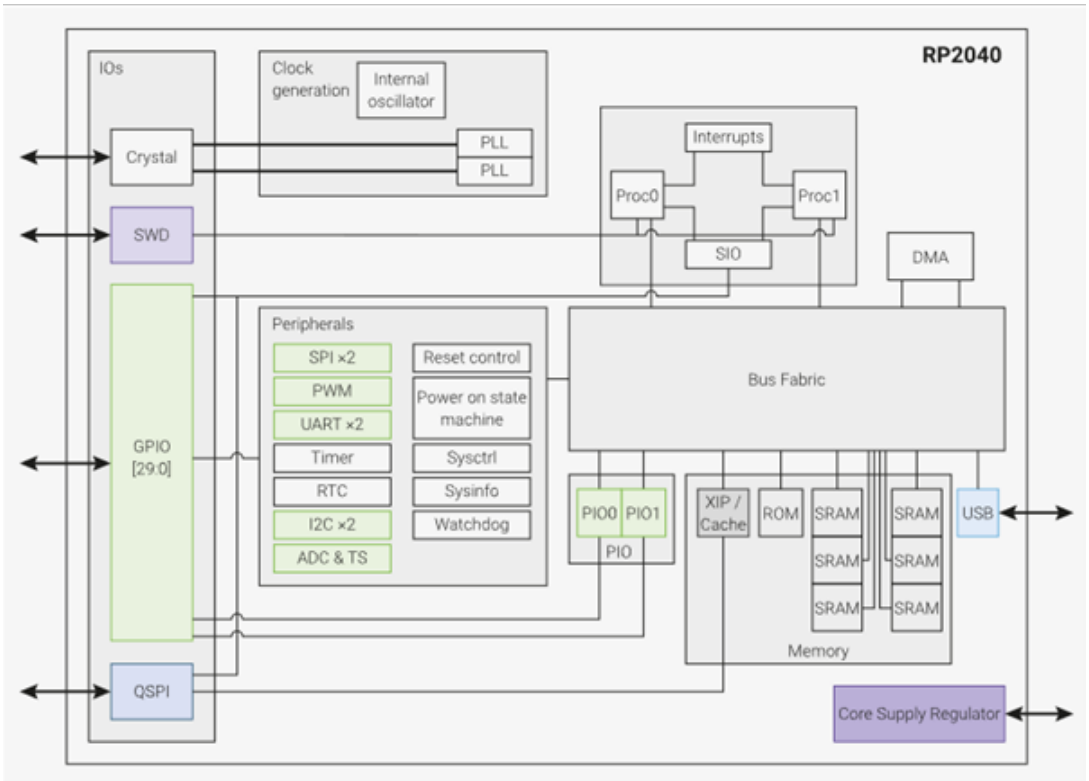
XN znamená, že procesor nemůže vykonávat instrukce v této části paměti. Při nedodržení se nastaví HardFault výjimka.

Tab. 1.1: Rozdělení paměti procesoru ARM cortex-M0+ [2]

Adresa	Název	Typ paměti	XN	Popis paměti
0x00000000-0x1FFFFFFF	Code	Normal	Ne	Typicky je to ROM nebo flash paměť, přímo na čipu. Primárně pro ukládání kódu
0x20000000-0x3FFFFFFF	SRAM	Normal	Ne	Pamět pro data, kód zde také může být
0x40000000-0x5FFFFFFF	Peripheral	Device	Ano	Paměť pro periferie na čipu
0x60000000-0x9FFFFFFF	External RAM	Normal	Ne	Paměť pro data (externí flash)
0xA0000000-0xDFFFFFFF	External device	Device	Ano	Paměť pro externí zařízení
0xE0000000-0xE00FFFFF	Private Peripheral Bus	Strongly-ordered	Ano	Tato oblast obsahuje NVIC, systémový čítač (SysTick)
0xE0100000-0xFFFFFFFF	System	Device	Ano	Tato oblast je určena výrobcem

2 RP 2040

Je to mikrokontrolér od společnosti Raspberry Pi Foundation. Obrázek 2.2 zobrazuje schéma čipu. Vidět jsou zde dva Arm Cortex-M0+ procesory, paměť, sběrnice a další části čipu, které budou popsány v následujícím textu. Informace v této kapitole jsou převzaté z [6], pokud není řečeno jinak.



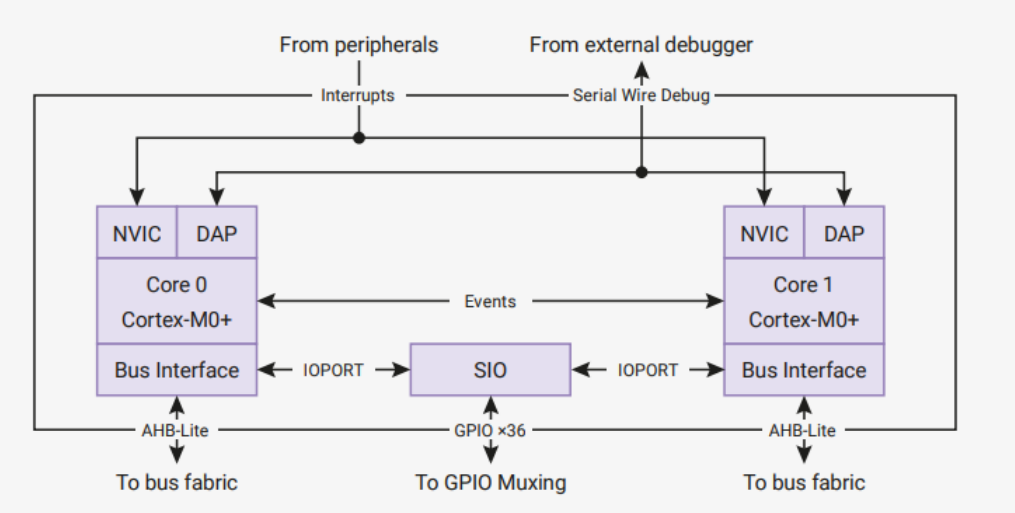
Obr. 2.1: Blokové schéma mikrokontroléru RP2040 [6]

2.1 Paměť

Mikrokontrolér integruje paměť SRAM o velikosti 264 kB, která je rozdělena do šesti bank, což umožňuje více přístupů k paměti zároveň. Do této paměti lze uložit data i kód. Dále mikrokontrolér může přistupovat k 16 MB externí paměti flash přes QSPI rozhraní. Při tomto způsobu přístupu se používá XIP (execute-in-place) hardware. To umožní mikrokontroléru vykonávat kód přímo z flash paměti (přistupovat ke flash paměti se bude, jako by byla přímo na čipu). Dále je na čipu 16 kB integrované paměti ROM. Data na tuto paměť byla zapsána při výrobě čipu a paměť obsahuje například data pro nastavení procesoru do počátečního výchozího stavu.

2.2 Subsystém RP2040

Subsystém se skládá ze dvou Arm Cortex-M0+ procesorů s jejich periferiemi. Každý procesor má svůj řadič přerušení (NVIC) 1.3.1 a je možno jej debugovat přes SWD (Serial WireDebug) sběrnici. Oba procesory jsou připojeny k SIO bloku, se kterým komunikují přes sběrnici IOPORT, která umožňuje rychlé 32bitové operace zápisu a čtení. Doba čtení a zápisu je deterministická a trvá jeden cyklus. Oproti tomu hlavní sběrnice potřebuje dva cykly pro čtení nebo a zápis a také může dojít ke zpoždění kvůli zahlcení sběrnice od jiných master zařízení.



Obr. 2.2: Schéma subsystému RP2040 [6]

2.2.1 SIO (Single-cycle IO)

Je to blok, který zajišťuje přenos dat s velkou rychlostí a malou latencí mezi procesory a některými periferiemi např. GPIO.

Blok obsahuje CPUID registry, ty umožňují jednoduchý způsob, jak softwarově zjistit na kterém jádru se program vykonává. Pokud jádro 0 přečte hodnotu registru, dostane 0 a pokud jádro 1 přečte hodnotu registru, dostane 1.

Dále se zde nachází GPIO registry, které mohou být použity pro ovládání GPIO pinů. Abychom mohli k GPIO přistupovat v jednom cyklu je nutno zvolit, aby byl GPIO ovládaný pomocí SIO (tabulka str. 13 [6]).

Pro komunikaci mezi jádry jsou zde dva 32bitové 8 vzorků hluboké FIFO registry (jedno pro každé jádro). Jádro může zapisovat pouze do jednoho FIFO registru a číst z toho druhého.

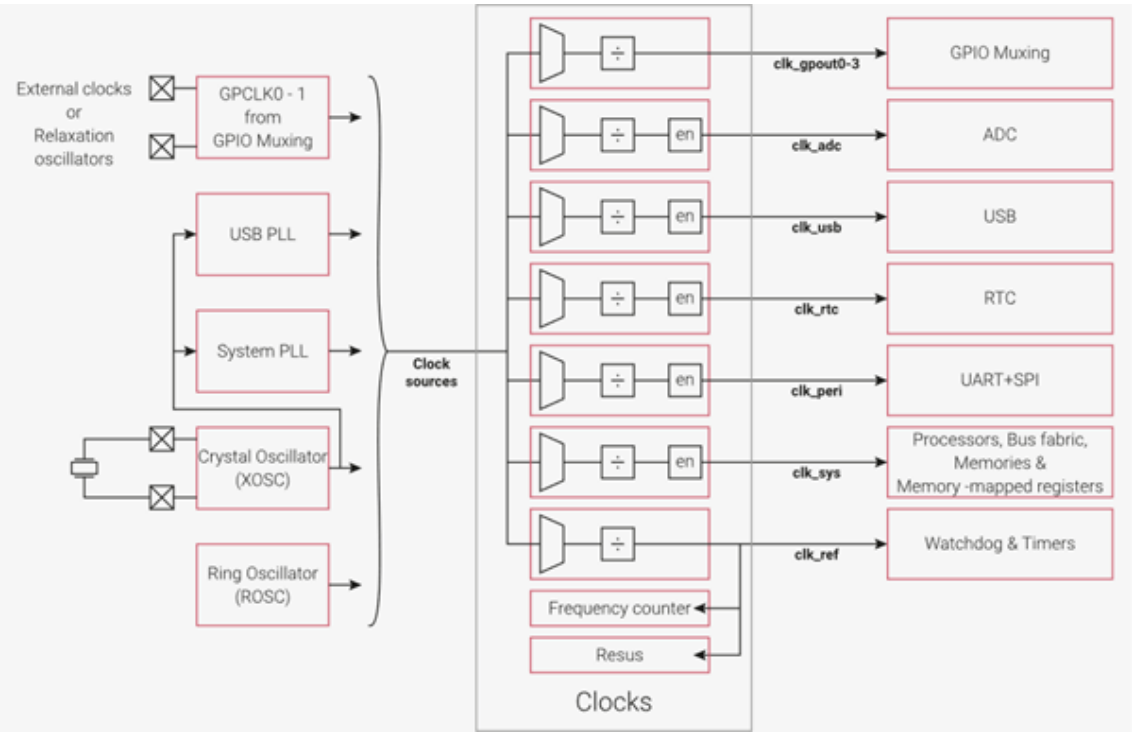
SIO také obsahuje optimalizovanou děličku celých čísel. Dělička vypočítá výsledek celočíselného dělení a jeho zbytek.

2.3 Přerušení

Přerušení slouží pro obsluhu neodkladných událostí. Procesor přeruší aktuálně prováděný kód, provede přerušovací rutinu a následně se vrátí ke přerušenému kódu. Každé jádro je vybaveno řadičem přerušení. Ty sdílí 25 přerušení, plus jedno přerušení ke každému řadiči zvlášť od GPIO pinů. Vnořené přerušení jsou podporovány, pokud dojde při obsluze přerušení s nižší prioritou požadavek na přerušení s vyšší prioritou, bude přerušena přerušovací rutina s nižší prioritou, aby se mohla vykonat ta s vyšší prioritou. Priorita je rozdělena dle nejvyšších dvou bitů v NVIC_IPR0-7 osmibitovém registru. Tedy je rozdělena na čtyři úrovně. Pro přerušení se stejnou prioritou se vybere přerušení s nižším číslem dle tabulky na str. 60 [6].

2.4 Zdroje hodinového signálu

Funkce mikrokontroléru je závislá na zdroji hodinového signálu. Procesor, sběrnice, periferie a další části mikrokontroléru jej potřebují pro synchronizaci operací. Je



Obr. 2.3: Přehled zdrojů hodinového signálu [6]

možné použít hodinový signál od více zdrojů, což umožňuje vyběr mezi stabilitou kmitočtu a spotřebou. Také lze zpomalit hodinový signál děličkou frekvence. Obvody en (enable) na obrázku 2.3 slouží pro automatické vypnutí hodin v režimu spánku. Pro nízkou spotřebu lze jako zdroj hodin použít Ring oscilátor (ROSC), jehož stabilita kmitočtu je nižší s porovnáním oproti krystalovému oscilátoru. Také lze použít externí hodinový signál. Pro stabilní kmitočet se dá použít krystalový oscilátor, který bude vstupem do fázového závěsu (PLL - Phase-locked loop).

2.4.1 Ring oscilátor

Ring oscilátor se skládá z lichého počtu NOT hradel, které jsou zapojeny do kruhu. Výstup tohoto zapojení osciluje mezi dvěma hodnotami napětí. Tento zdroj hodinového signálu je používán při počátečních fázích startu systému, tedy po zapnutí napájecího napětí. Frekvence Ring oscilátoru se pohybuje kolem 6 MHz, ale mění se změnou velikosti napájení nebo se změnou teploty.

2.4.2 Krystalový oscilátor

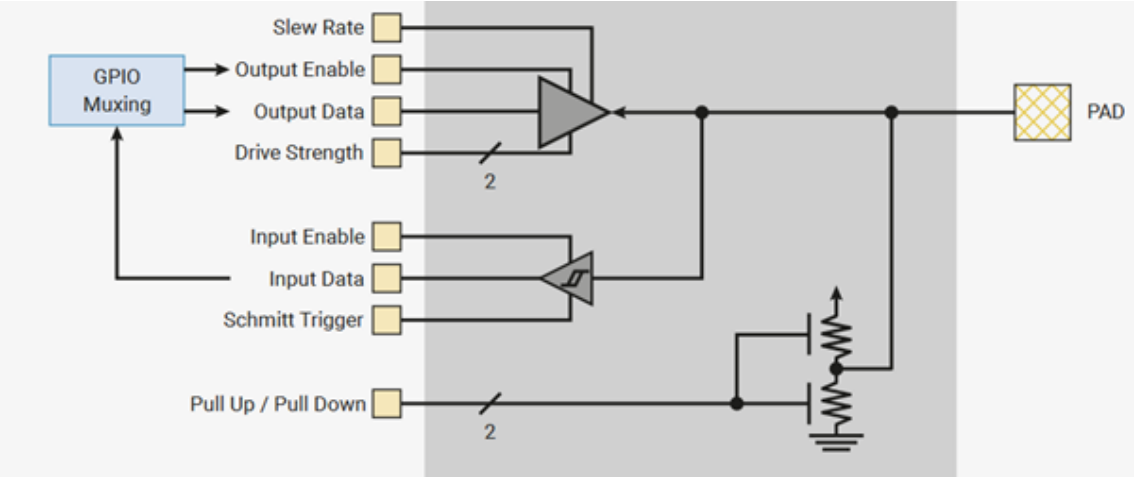
Je to stabilní zdroj kmitočtu. Frekvence je dána externím krystalem. Na výstup krystalu je možné připojit fázový závěs pro zvýšení frekvence. Tento oscilátor není aktivní při počátečních fázích po zapnutí, při použití je potřeba jej aktivovat programově.

2.5 GPIO

RP2040 má 36 multifunkčních GPIO (General Purpose Input / Output) pinů. Rozdělených do dvou bank. První banka obsahuje GPIO0 až GPIO29, kterým lze přiřadit různé funkce (SPI, UART, I2C, SIO, ...). Druhá banka (QSPI banka) slouží pro vykonávání kódu z flash paměti mimo čip. Všechny GPIO z první banky se dají využít jako digitální vstupy nebo výstupy, navíc piny GPIO26 až GPIO29 mohou být využity jako analogové vstupy do AD převodníku. Funkce se zvolí zápisem do GPIO Control registru. GPIO piny jsou připojeny na vývody na desce RP PICO.

Tyto vývody slouží mimo jiné i jako ochrana, před elektrostatickým nábojem. Vývodům lze nastavit např. velikost výstupního proudu (2 mA, 4 mA, 8 mA nebo 12 mA), hysterezi, strmost hran, pull up nebo pull down rezistor.

Požadavek na přerušení pro GPIO může být vystaven při následujících událostech, a to, když na vstupu GPIO je logická 1, logická 0 nebo při nástupné/sestupné hraně. Na vývod může být připojeno 2,3 V až 3,3 V, při jiném nastavení pin může být ovládán 1,8 V.



Obr. 2.4: Schéma vývodu RP2040 [6]

2.6 DMA (Direct Memory Access)

DMA je sběrnicový master, který vydává příkazy na sběrnici. DMA se používá pro přenos dat s velkou rychlostí, nezávisle na procesoru. Procesor není zatěžován jednoduchými úkoly a jeho zdroje mohou být využity jiným způsobem nebo lze vstoupit do režimu spánku s nízkou spotřebou. Procesor nastaví počáteční a cílovou adresu v paměti, velikost přenášených dat, jestli se bude cílová, nebo počáteční adresa inkrementovat a po kolika bitech se budou data přenášet. Přenosy obvykle probíhají následujícími způsoby. Přenos mezi adresou v paměti a registrem periferie. Přenos mezi registrem periferie a adresou v paměti. Nebo přenos mezi dvěma místy v paměti.

Kdyby se pomocí DMA přenášely data do registru nebo z registru periferie maximální rychlostí mohlo by dojít ke ztrátě dat. Pro dosažení rychlosti přenosu požadované periferií existuje DREQ (Data request). DMA zahájí přenos vzorku dat při vystavení DREQ periferií. Maximální rychlost je využita pro přenos mezi dvěma místy v paměti.

DMA má velkou datovou propustnost, každý hodinový impulz lze provést operaci čtení a zápisu o velikosti 8, 16 nebo 32 bitů a mikrokontrolér obsahuje 12 nezávislých DMA.

Přenos dat může být zahájen zápisem do příslušného registru nebo pomocí tzv. chain trigger, kdy přenos je zahájen po dokončení jiného přenosu.[1]

[1]Control registry DMA jsou paměťově mapovány, tzn. že DMA kanál může zapsat do Control registru jiného DMA kanálu.

2.7 SWD

SWD (Serial Wire Debug) slouží pro debugování. Umožňuje nahrát firmware do SRAM nebo flash paměti, ovládat běh procesoru (run, halt mód, pro debugování umožňuje přidat tzv. breakpoints). Také umožňuje přistupovat do paměti. SWD má dva signály, a to SWDIO a SWCLK.

2.8 Periferie RP2040

V této kapitole budou popsány vybrané periferie mikrokontroléru RP2040.

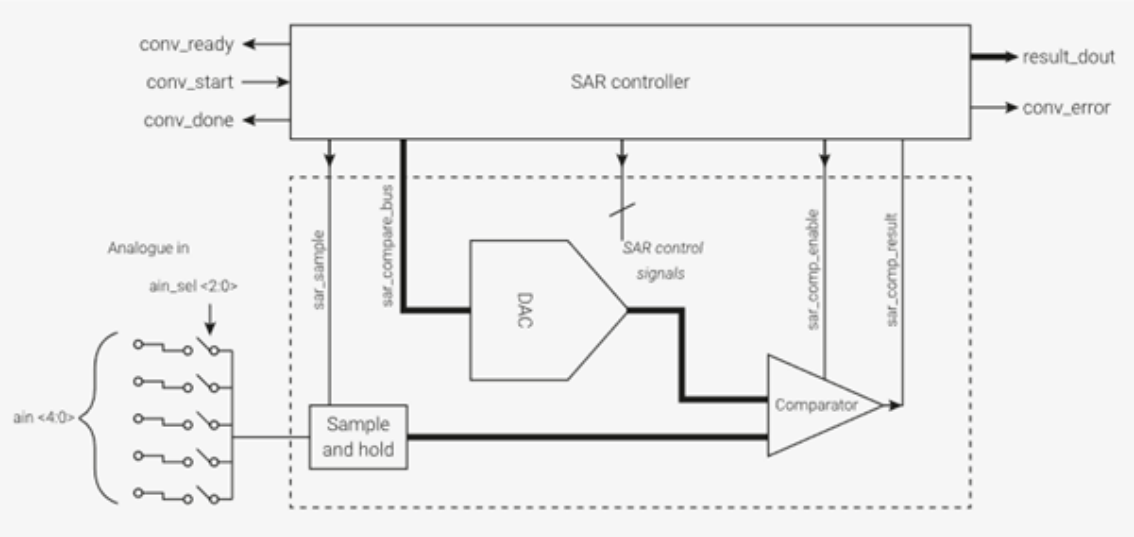
2.8.1 A/D převodník

RP2040 má integrovaný 12bitový A/D převodník (efektivní počet bitů je 8,7). Převod analogové hodnoty na digitální vrátí číslo od 0 do 4095, což znamená celkově 4096 možných hodnot. Při napájení mikrokontroléru 3,3 V bude kvantizační krok 3,3/4096 což odpovídá přibližně 0,8 mV.

ADC potřebuje hodinový signál o frekvenci 48 MHz. Jako zdroj tohoto signálu může být použit signál z krystalového oscilátoru, který je přiveden na fázový závěs pro dosažení požadované frekvence. Pro převedení jednoho vzorku potřebuje ADC 96 hodinových impulzů, což při frekvenci 48 MHz znamená 500 ksps. Zápisem do DIV registru lze nastavit děličku a A/D převodník bude vzorkovat pomaleji. Také lze zvolit Round Robin mód, v tomto případě se bude vzorkovat z více vstupů. Při dokončení převodu, se vybere automaticky další vstup.

Na obrázku 2.5 je znázorněno schéma ADC. V levém dolním rohu je pětivstupový multiplexer, ten určuje, který z analogových vstupů bude zvolen. Čtyři z těchto vstupů mohou být vyvedeny na piny na vývojové desce, pátý je připojený na interní teplotní senzor. Multiplexer je připojený na Sample and hold registr, což je kondenzátor s malou kapacitou. Pokud připojíme některý ze vstupů mulitplexeru, kondenzátor se nabije na napětí na vstupu. Postupný aproximační registr si nejprve zvolí nejvyšší bit, který je reprezentován napětím z D/A převodníku a porovná toto napětí s napětím na kondenzátoru. Dle výsledku porovnání se zapíše hodnota bitu do postupného aproximačního registru (SAR). Následně se určí druhý nejvyšší bit. Tímto algoritmem se postupuje až do dosažení přesnosti odpovídající 12 bitů.

AD převodník je možné nastavit do jednorázového režimu. Napětí na vstupu AD převodníku bude převedeno na digitální hodnotu, a tato hodnota se zapíše do registru. Také lze nastavit AD převodník do kontinuálním režimu. V tomto režimu je možno nastavit, aby se vzorky ukládaly do FIFO registru, ze kterého mohou být například kopírovat pomocí DMA do pole.



Obr. 2.5: Schéma AD převodníku [6]

2.8.2 Timer

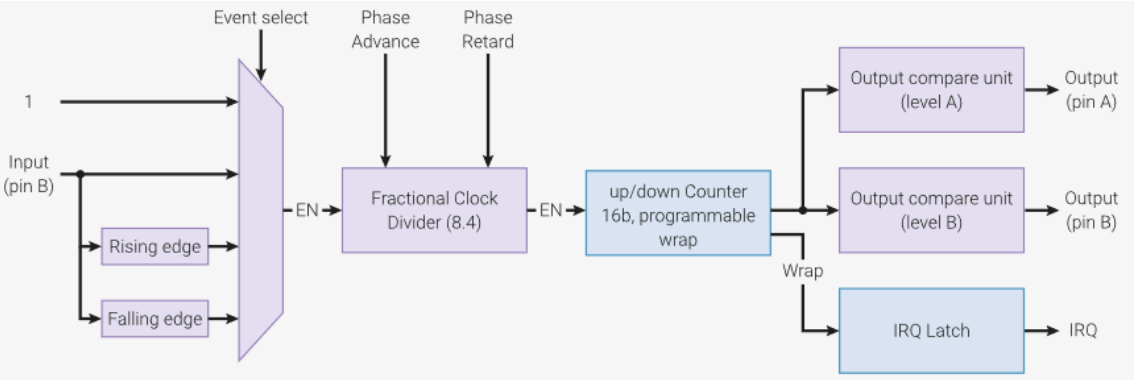
Tato periferie poskytuje 64bitový čítač, který čítá s periodou 1 µs. Čítač v podstatě nemůže přetéct, při frekvenci 1 MHz by to trvalo tisíce let. Čítač je 64bitový, ale na RP2040 je ale 32bitová sběrnice, to znamená, že s hodnotou čítače se musí pracovat pomocí páru registrů.

Tato periferie také obsahuje čtyři 32bitové alarmy. Každý alarm může vyvolat požadavek na přerušení. Alarm vyvolá požadavek na přerušení, pokud se nastavená hodnota shoduje s hodnotou nižších 32 bitů čítače. To znamená že alarm může být nastaven maximálně 2^{32} mikrosekund dopředu (72 minut).

2.8.3 PWM (Pulse-width modulation)

Pomocí pulzně šířkové modulace je možno vytvářet obdélníkový signál s danou střídou a periodou. PWM modul obsahuje 16bitový volně běžící čítač. Na jeho vstup může být připojený signál *system clock* nebo signál z ADC pinu B (viz níže). Čítač je možno zpomalit děličkou frekvence. Hodnota čítače je porovnávána s hodnotou v *cc* registru. Jestliže je hodnota v čítači menší, než hodnota v *cc* registru výstup je v logické 0, jestliže je větší, výstup je v log. 1.

RP2040 integruje 8 PWM modulů. Každý modul obsahuje dva kanály, které lze nastavit pro generování dvou signálů nebo jeden z těchto kanálů může být nastaven jako vstupní a lze jej použít pro měření frekvence nebo střídy. Čítač se v tomto případě bude inkrementovat při nástupné/sestupné hraně nebo při logické 1.

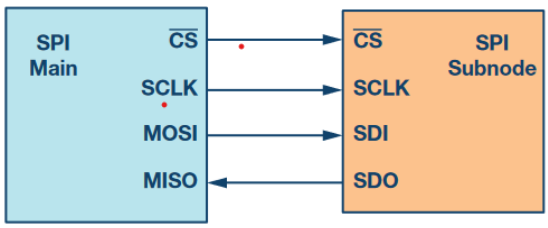


Obr. 2.6: Schéma PWM modulu [6]

2.8.4 SPI (Serial peripheral interface)

SPI je jeden z nejpoužívanějších protokolů používaných mezi mikrokontrolerem a periferiemi jako jsou sensory, AD převodníky, DA převodníky a další. Používá master slave architekturu (main-subnode). Je to synchronní, full duplex (obě zařízení mohou posílat data zároveň) komunikace. Původně byl protokol navržený pro čtyři vodiče, ale lze použít tří vodičovou variantu, která je ovšem half duplex. Což např. pro komunikaci s displejem je dostačující. Plná komunikace používá následující čtyři signály:

- Chip select (CS) – signál aktivní v 0, nastavuje jej master zařízení, aby adresoval slave zařízení a zahájil komunikaci
- SCLK – synchronní hodinový signál
- MISO – data od slave zařízení do master zařízení
- MOSI – data od master zařízení do slave zařízení



Obr. 2.7: Schéma SPI komunikace [7]

Hodinový signál je generován master zařízením a synchronizuje komunikaci, Při porovnání SPI a I2C, SPI podporuje mnohem vyšší frekvenci hodinového signálu než I2C. Protokol není striktně definovaný, takže si lze zvolit z několika konfigurací. Data mohou být čtena a zapisována na nástupnou nebo sestupnou hranu. Také si lze

zvolit jaká bude klidová hodnota pro hodinový signál. RP2040 obsahuje dvě stejné SPI periferie. [7].

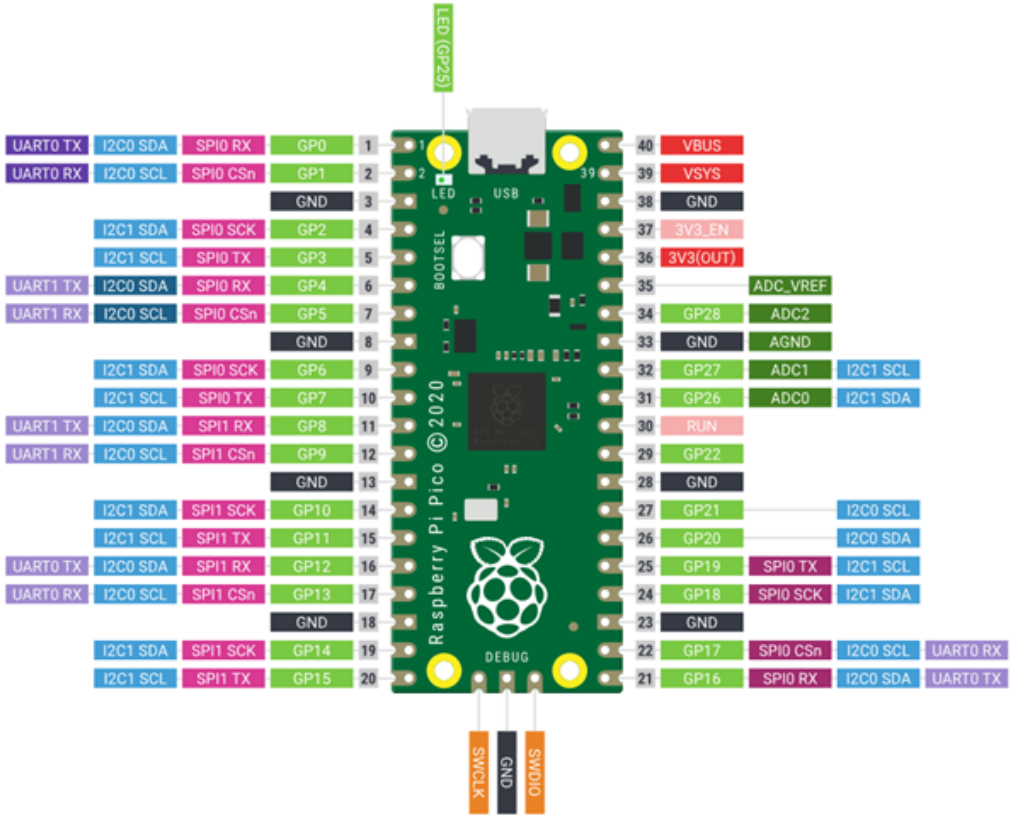
2.8.5 RTC - Real-time Clock

Hodiny reálného času udržují údaj o aktuálním čase v čitelném formátu pro člověka. Hodnota je rozdělena na sedm polí (rok, měsíc, den, den v týdnu, hodina, minuta, sekunda), přičemž den v týdnu je celé číslo, které reprezentuje string.

Také lze pomocí této periferii nastavit přerušení do kterého se vstoupí po uplynutí určitého času.

3 Vývojová deska Raspberry Pi Pico

Je to vývojová deska od společnosti Raspberry Pi Foundation. Tato společnost se spíše specializuje na jednodeskové počítače, které mají operační systém a mají vetší rozměry. Standardně tato deska operační systém nepoužívá. Vývojová deska je poměrně levná, její cena se pohybuje do 180 Kč. Prodává se ve čtyřech variantách [5] a to standardní s označením Pico, Pico H s napájenými pinovými headery, Pico W s bezdrátovým rozhraním, desku si lze také pořídit ve variantě WH, která kombinuje varianty W a H. Pro tuto vývojovou desku si společnost vytvořila vlastní čip RP2040.



Obr. 3.1: Rozložení pinů na RP Pico [5]

3.1 Parametry Vývojové desky

- Dual-core Arm Cortex M0+ procesor, flexibilní hodiny až do frekvence 133MHz
- 264kB paměti SRAM, a 2 MB flash mimo čip
- USB
- Režimy spánku s nízkou spotřebou

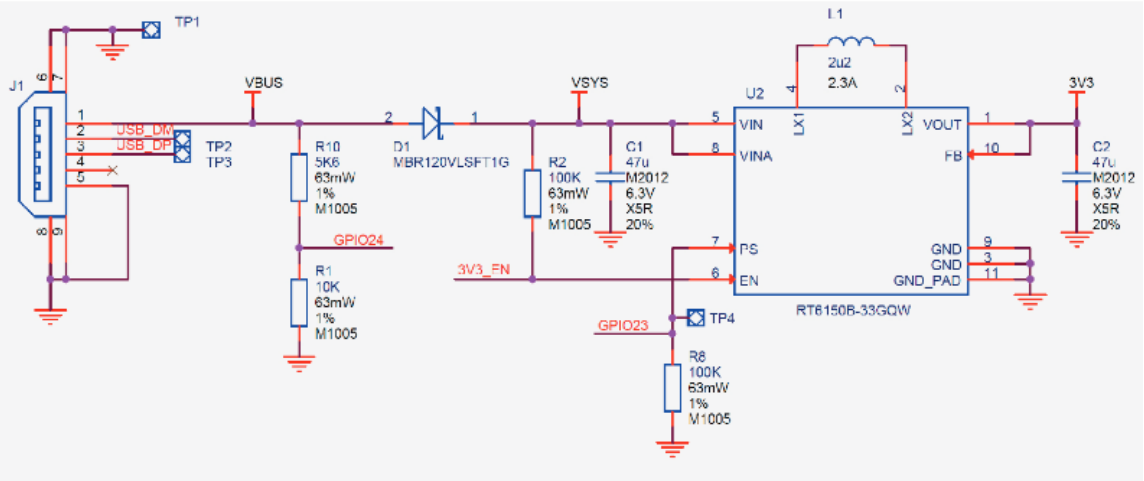
- 2 fázové závěsy pro generování hodinového signálu
- 26 GPIO pinů, 3 z nich mohou být použity jako analogové vstupy
- 2 × SPI, 2 × I2C, 2 × UART, 12bitový AD převodník, 16 × ovladatelných PWM kanálů, tepelný senzor
- 8 Programovatelných I/O PIO stavových automatů, 12 DMA kanálů

Funkci pinů je možno nastavit dle obrázku 3.1. Každý GPIO pin může být připojen na různé periferie, některé periferie mohou být připojeny na jeden z více pinů, pro větší flexibilitu. Všechny piny lze také připojit k SIO 2.2.1 a k PIO0 a PIO1. PIO jsou bloky, které mohou nezávisle vykonávat program, jsou časově deterministické, takže se dají použít například pro komunikace nebo operace s GPIO. Dále se těmito PIO nebudeme zabývat.[6].

3.2 Napájení

Napájet desku je možno 5 V přes micro USB kabel. VBUS je přímo připojen na micro USB port. Není vhodné napájet desku přes VBUS, protože by také mohla být připojena přes micro USB a mohlo by dojít k opačnému proudu elektrické energie, než bylo zamýšleno. Pro zamezení tohoto jevu je mezi VSYS a VBUS Schottkyho dioda. Vývojovou desku lze také napájet z pinu VSYS napětím 1,8 V až 5,5 V. VSYS je připojen na tzv. BUCK-BOOST[2], který generuje napětí 3,3 V pro RP2040 a obvody připojené na pin 3V3.

Spotřeba desky je od 18 mA do 95 mA při maximálním zatížení. Informace v této podkapitole jsou přebrány z [10].



Obr. 3.2: Schéma napájení vývojové desky [10]

[2]BUCK-BOOST je měnič typu step up a zároveň step down [9]

4 Práce s vývojovou deskou RP Pico

V této kapitole bude popsáno vybrané vývojové prostředí pro programování mikrokontroléru RP2040, jeho instalace, založení projektu a práce s ním.

4.1 Programování vývojové desky

Desku lze programovat v jazycích C/C++ nebo také v jazyku MicroPython což je implementace programovacího jazyka Python 3. MicroPython se skládá z kompilátoru, který kompiluje python kód na tzv. bytecode a interpreteru, který vykonává tento bytecode. Tento způsob programování je jednodušší, ale dále v této práci nebude využíván.[8]

4.2 C/C++ SDK

Programovat vývojovou desku je možné zápisem do registrů, ale při složitějších programech je tento způsob obtížný. Proto existuje SDK (Software Development Kit), který vytváří vyšší úroveň abstrakce a místo zápisu do jednotlivých registrů stačí zavolat funkci. SDK obsahuje hlavičkové soubory, knihovny a systém který je potřeba k sestavení programu [11].

4.3 CMake

CMake je multiplatformový opensource software, který je určený pro automatizaci překladu, testování a vytváření softwarových balíků [12]. CMake používá soubory CMakeLists.txt, které obsahují instrukce pro proces sestavení projektu. CMake generuje podle CMakeLists.txt soubory, podle kterých se bude řídit překlad.

Zde je příklad souboru CMakeLists.txt 4.3. Jako první funkci je potřeba zavolat *cmake_minimum_required()*, tím se nastaví minimální verze cmake pro daný projekt. Druhá funkce slouží pro přidání cmake souboru, který obsahuje předepsané funkce pro lokalizaci pico SDK. Funkce *project()* nastaví jméno projektu, funkce *set()* určí verzi C/C++, dále následuje inicializace SDK. Funkce *add_executable()* určuje, že program s názvem "jména projektu" bude sestaven z main.c souboru. Dále se nastaví, aby byly vytvořeny další soubory s příponou například .uf2 (ten slouží pro jednoduché nahrání programu na Pico pomocí boot tlačítka), .bin nebo .hex. Funkce *target_link_libraries()* specifikuje, které soubory použije linker, definují se zde knihovny, které jsou potřeba pro daný projekt. Poslední funkce určuje, na která rozhraní má být stdout zpráva poslána (např. funkcí *printf()*).

Pro tisk na konzoli při nahrání programu přes masové uložiště je potřeba mít povolený výstup na USB a při debugování je potřeba mít povolený výstup na UART, protože debugovaný mikrokontroler komunikuje s debuggerem přes UART. Informace v tomto odstavci jsou převzaty z [15].

Výpis 4.1: Příklad souboru CMakeLists.txt [15].

```
#Set minimum required version of CMake
cmake_minimum_required(VERSION 3.12)

#include build functions from Pico SDK
include($ENV{PICO_SDK_PATH}/external/pico_sdk_import.cmake)

#set the name of the project and C/C++ standarts
project(blink C CXX ASM)
set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)

#Creates a pico-sdk subdirectory
#in our project for the libreries
pico_sdk_init()

#Tell CMake where to find the executable source file
add_executable(${PROJECT_NAME}
    main.c
)

# Create map/bin/hex/uf2 files
pico_add_extra_outputs(${PROJECT_NAME})

# Link to pico_stdlib (gpio, time, etc. functions)
target_link_libraries(${PROJECT_NAME}
pico_stdlib)

#enable USB output, disable UART output
pico_enable_stdio_usb(${PROJECT_NAME} 1)
pico_enable_stdio_uart(${PROJECT_NAME} 0)
```

4.4 Vývojové prostředí

Jako vývojové prostředí bylo zvoleno Visual Studio Code. Nejprve bylo postupováno dle návodu [13], ale ukázalo se, že tento způsob je spíše vhodný pro Raspberry Pi

OS. Návod také popisuje, jak nainstalovat toolchain na Microsoft Windows a Apple macOS. Při instalaci na macOS není postup příliš rozdílný od instalace na Linux. Avšak při použití Windows je postup složitější. Proto byl zvolen jiný postup, a to dle [16], z něhož jsou čerpány informace v této podkapitole. Na této webové stránce se nachází instalační balíček, který obsahuje vše potřebné včetně editoru kódu Visual Studio Code. Verze SDK (software development kit), se kterou bylo pracováno je v1.5.1, pro jiné verze následující postup nemusí platit.

4.5 Postup instalace vývojového prostředí

Na webové stránce [16] stačí kliknout na tlačítko *Download Windows Pico Installer* a stáhne se instalační soubor (*pico-setup-windows-x64-standalone.exe*). Dále je potřeba pokračovat v instalaci. Tento balík obsahuje všechno potřebné včetně Visual Studio Code a softwaru pro debugování. Dále je nutno najít ve Start nabídce *Pico - Visual Studio Code* a spouštět program tímto způsobem, aby se nastavili potřebné proměnné prostředí (environment variables).

4.6 Založení nového projektu

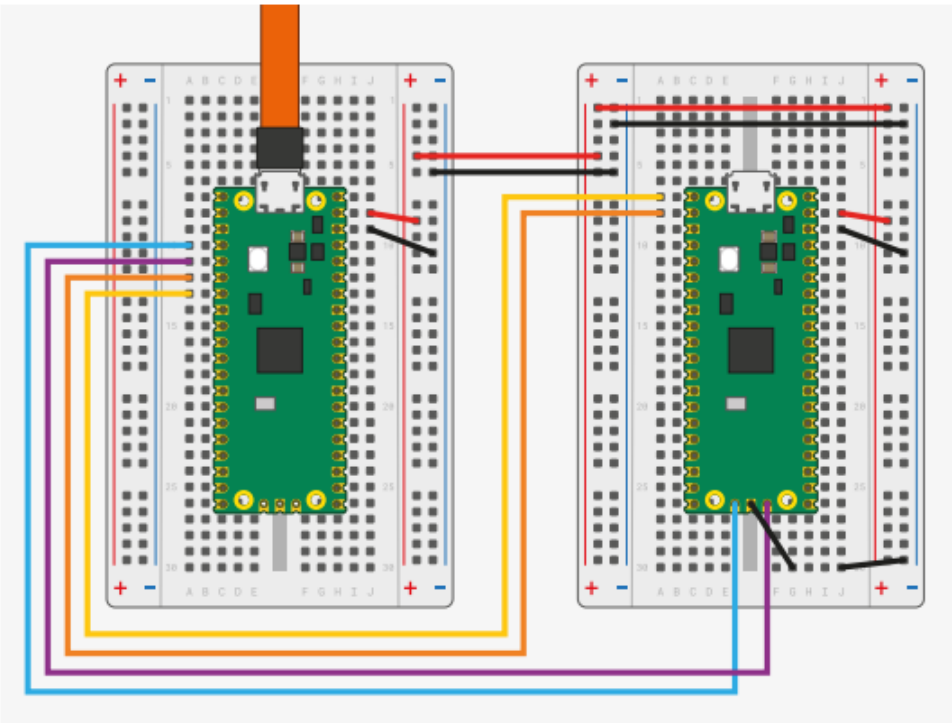
Při vytvoření nového projektu je nutno vždy (pro možnost debugování) zkopírovat do adresáře projektu adresář s názvem *.vscode*, viz. příloha E.3. Dále je nutno v projektu vytvořit soubor *CMakeLists.txt* 4.3. Jako další krok je nutno program zkonfigurovat kliknutím vlevo na záložku *CMake*, poté se se klikne na *Delete Cache and Reconfigure* a zvolí se Pico ARM GCC.

4.7 Nahrání programu na RP Pico

Program lze nahrát dvěma způsoby, a to podržením BOOTSEL tlačítka a zastrčením USB konektoru do portu na desce. Pico se bude poté chovat jako masové uložiště a je možno do něj zkopírovat sestavený soubor (zmáčknutím Build) s příponou .uf2 z adresáře *build* v projektu.

Druhá možnost je při použití druhé desky RP Pico, které slouží jako debugger. Vývojové desky se zapojí dle 4.1, kdy levá deska bude připojena k počítači přes USB[13]. Na tuto desku se nahraje firmware s názvem *picoprobe*. Debugger bude přeposílat SWD a UART komunikaci na USB z desky, na které se vykonává program. Debugger bude zároveň poskytovat napájení desce vpravo propojením pinů VSYS a taktéž propojením zemí obou desek.

Pro debugování se ve vývojovém prostředí Visual Studio Code klikne na tlačítko *Debug C/C++ File* vpravo nahoře a zvolí se Pico *Debug (C ++ Debuger)*. Poté se zvolí cíl spuštění (launch target), ten bude mít jméno, které bylo nastaveno v CMakeLists.txt. Tímto se program se zkompiluje a nahraje na mikrokontrolér. Výhoda této možnosti je, že není nutno při každém nahrání programu vysouvat USB konektor, a tím opotřebovávat port na počítači nebo na RP Picu.



Obr. 4.1: Zapojení vývojové desky (vpravo) ke 2. desce, která slouží jako debugger (vlevo) [13]

4.8 Tisk na konzoli

Aby funkce *printf()* mohla tisknout na terminál, je třeba inicializovat UART funkcí *stdio_init_all()*. Také je třeba umístit do CMakeLists.txt příkaz *pico_enable_stdio_uart(${PROJECT_NAME} 1)*, který zajistí přesměrování stdout na UART. Toto je potřeba, jelikož RP Pico, na kterém je vykonávaný program je připojena k desce, která slouží jako debugger přes UART. Dále je potřeba ve spodní liště v editoru kódu VS Code rozkliknout *SERIAL MONITOR* a nastavit *Baud rate* na 115200, nastavit správný port a začíst čtení sériové komunikace tlačítkem *Start Monitoring*.

5 Návrh přizpůsobovacích obvodů

Pro realizaci laboratorních úloh jsou zapotřebí dvě vývojové desky RP Pico, přičemž na jedné z nich se bude vykonávat program a druhá bude sloužit jako debugger. Dále se budou v laboratorních úlohách využívat následující komponenty: Dvě led diody s odpory pro omezení proudu, dvě tlačítka, potenciometr, TFT LCD displej a obvod TLC555 s příslušnými rezistory a kondenzátory.

Nejprve byly výše uvedené komponenty propojeny na nepájivém poli a následně byla vytvořena deska plošných spojů.

5.1 Popis jednotlivých částí desky plošných spojů

V této kapitole budou popsány jednotlivé části desky plošných spojů. Celé schéma zapojení a taktéž návrh DPS je uveden v příloze B. V příloze C je také seznam použitých komponent.

5.1.1 Generátor obdélníkového signálu

Pro generování signálu v "Laboratorní úloze - ADC a DMA, PWM" je využit integrovaný obvod TLC555, pro jehož funkci stačí napájecí napětí 2 V (na rozdíl od standardní NE555), proto lze s výhodou použít přímo pin 3V3 z RP Pico. Integrovaný obvod používá CMOS technologii.

Časovač pracuje v astabilním režimu. Integrovaný obvod je zapojen dle katalogového listu [14]. Pin DIS (Discharge) je interně připojen ke tranzistoru, který je ovládán výstupem z klopného obvodu. Negace výstupu klopného obvodu je zároveň výstupem celého integrovaného obvodu (pin Q). Jakmile se tento tranzistor sepne, pin DIS bude připojen k zemi, a tím se začne vybíjet kondenzátor C1. Po dosažení určitého napětí na pinu THR (threshold) a zároveň na pinu TR (trigger), se klopný obvod přepne a tranzistor připojený k pinu DIS se zavře a kondenzátor C1 se začne nabíjet až do dosažní určitého napětí na pinech THR a TR.

Průběh napětí na pinu Q byl zaznamenán pomocí osciloskopu, viz. příloha D.

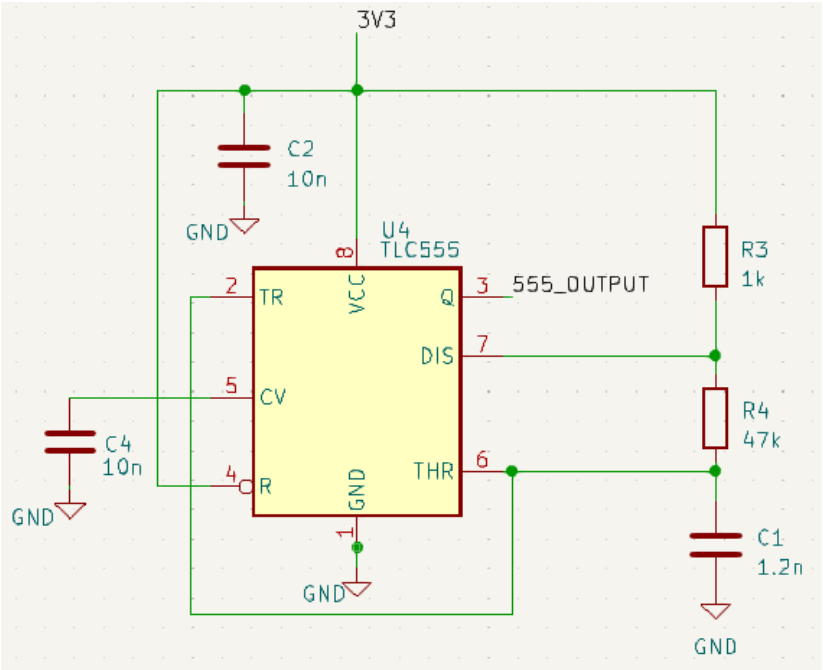
C2 je blokovací kondenzátor, který zajišťuje, aby se šum nedostal do integrovaného obvodu, ale byl uzemněn. C4 je také blokovací kondenzátor. V katalogovém listu [14] jsou vztahy, podle kterých byly určeny hodnoty rezistorů a kondenzátoru.

Veličina t_h reprezentuje dobu vysoké úrovně signálu a t_l dobu nízké úrovně signálu.

$$\begin{aligned} t_h &= 0,693 \cdot (R3 + R4) \cdot C \\ t_h &= 0,693 \cdot (1000 + 47000) \cdot 1,2 \cdot 10^{-9} = 3,99 \cdot 10^{-5} \ \ s \\ t_l &= 0,693 \cdot R4 \cdot C \\ t_l &= 0,693 \cdot 47000 \cdot 1.2 \cdot 10^{-9} = 3.91s \cdot 10^{-5} \ \ s \end{aligned} \tag{5.1}$$

Perioda je pak dána součtem t_h a t_l.

$$\begin{aligned} T &= t_h + t_l = 3.99 \cdot 10^{-5} + 3,91 \cdot 10^{-5} = 7,9 \cdot 10^{-5} \ \ s \\ f &= \frac{1}{T} = \frac{1}{7,9 \cdot 10^{-5}} = 12,66 \ \ kHz. \end{aligned} \tag{5.2}$$



Obr. 5.1: Schéma zapojení generátoru s integrovaným obvodem TLC555

5.1.2 TFT LCD Displej

Pro zobrazování výstupů z laboratorních úloh byl vybrán ST7789 TFT LCD (Thin-Film-Transistor Liquid Crystal Display) displej, s rozlišením 240 x 240 pixelů. Displej používá ovladač ST7789VW [17]. Na rozdíl od OLED displeje, je na LCD TFT displeji nutné podsvícení. [18]

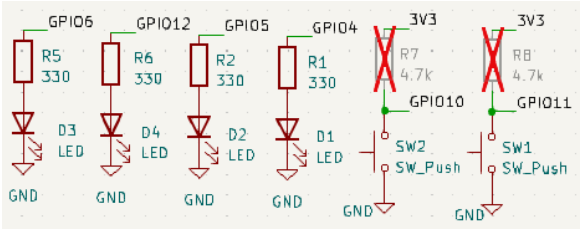
Pro ovládání displeje je využita knihovna *pico-displayDrivs* [19], její dokumentace je příloze A. Tato knihovna je z velké části postavena na GFX knihovně [20] od společnosti Adafruit, která poskytuje sadu funkcí pro více typů displejů.

Obraz je vytvořen z pixelů, které mají v původním nastavení souřadnicový systém s počátkem v levém horním rohu. Pixel je reprezentován třemi složkami, a to RGB (Red, Green, Blue), a to 16bitovou hodnotou ve formátu, kde nejnižších 5 bitů reprezentuje modrou barvu, následujících 6 bitů barvu zelenou a nejvyšších 5 bitů červenou. [20]

5.1.3 Led diody a tlačítka

Pro základní interakci s RP Pico slouží led diody a tlačítka. Proud led diodami je omezen rezistory.

Tlačítka nemají obvody pro odstranění zákmitů, jelikož je toto cílem jedné z laboratorních úloh. Také jsou zde neosazené pullup rezistory, které lze však připojit sofwarově v mikrokontroléru.



Obr. 5.2: Zapojení tlačítek a led diod

5.1.4 Zapojení desky RP Pico

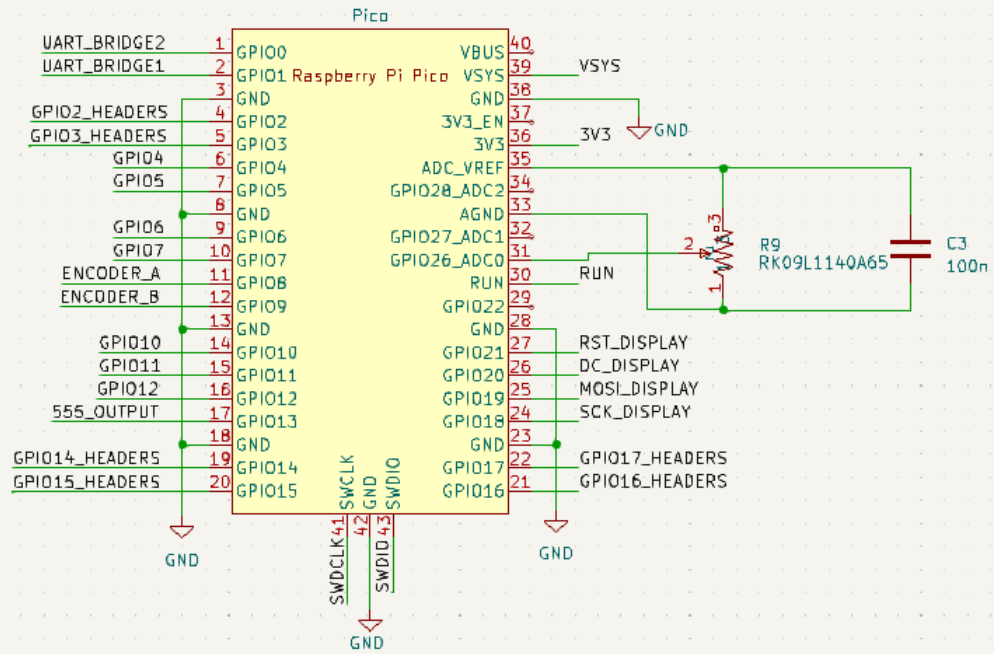
Na obrázku 5.3 je znázorněno připojení podpůrných obvodů k mikrokontroléru RP Pico. Debugger je připojen přes SWD interface viz. 4.1 a přes UART pomocí GPIO0 a GPIO1.

Je zde také vidět potenciometr, ten je napájený z pinu ADC_VREF, toto napětí je generováno z pinu 3V3, které je filtrováno RC filtrem. Potenciometr je připojen na analogovou zem RP Pica.

5.1.5 Návrh a výroba desky plošných spojů

Celý návrh byl vytvořen v programu KiCad ve verzi 7.0. Navrhnuta byla dvouvrstvá DPS s rozměry 97 x 84 mm z důvodu nejnižší cenové kategorie u výrobce JLCPCB.

Nejprve byly určeny minimální parametry pro výrobu dle požadavků výrobce (JLCPCB). Dále byly zvoleny rozměry desky a proběhlo rozmístění obou RP Pico desek tak, aby oba micro USB konektory byly na okrajích DPS. Následně byly rozmístěny další komponenty pro jednoduchý přístup. Blokovací kondenzátory byly



Obr. 5.3: Schématické zapojení RP Pica

umístěny blízko pinů, ke kterým jsou připojeny. Následně byly spojeny všechny cesty. Jelikož UART ani SWD nedosahují frekvencí takových, aby vlnová délka signálu byla porovnatelná s délkou trasy, tyto trasy nebyly vedeny jako diferenční páry. Šířka signálových cest byla zvolena 0,3 mm, což výrazně převyšuje minimální přípustnou šířku pro proud, který těmito cestami poteče. Tento proud je zanedbatelný. Šířka napájecích cest byla zvolena 0,5 mm. DPS je napájena z micro USB a maximální odběr proudu všech komponent na DPS taktéž výrazně převyšuje přípustný proud pro danou šířku cest.

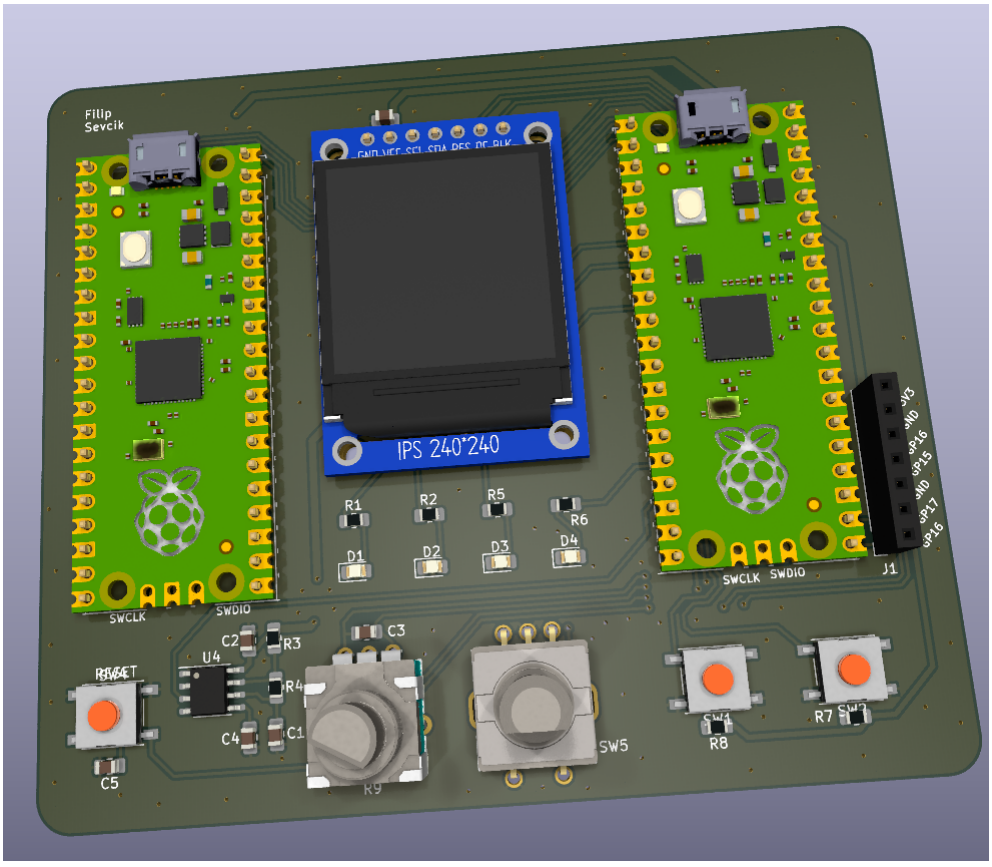
Pro výrobu byly vygenerovány Gerber, Drill a Placement soubory. Také byl vygenerován BOM (Bill of Materials). Tyto soubory jsou v příloze E

5.1.6 Osazení a otestování DPS

SMD komponenty byly osazeny přímo ve výrobě, THT komponenty byly ručně napájeny. Na DPS byly navrhnuty díry pro uchycení displeje šrouby s distančními sloupky, avšak při osazování byl displej pouze napájen. Distanční sloupky by případně mohly být vytisknuty na 3D tiskárně, avšak to už není náplní této práce. Krabička pro celou DPS by také mohla být vytisknuta na 3D tiskárně. Při pájení nevznikly žádné problémy. Byly vyzkoušeny všechny navržené laboratorní úlohy a všechny části DPS byly funkční.

Tab. 5.1: Tabulka parametrů použitých k výrobě DPS

Tloušťka DPS	1,6 mm	Průměr díry prokovů signálových tras	0,8 mm
Základní materiál DPS	FR-4	Průměr díry prokovů napájecích tras	0,8 mm
Minimální vzdálenost dvou cest	0.127 mm	Vnější průměr prokovů signálových tras	1,6 mm
Šířka vnějších měděných vrstev	1 oz/ft^2	Vnější průměr prokovů napájecích tras	1,6 mm



Obr. 5.4: 3D model navrhnuté desky plošných spojů

6 Laboratorní úlohy

V této kapitole jsou zadání jednotlivých laboratorních úloh. V některých laboratorních úlohách v návodu k vypracování je řečeno, že si studenti mají do projektu přidat například adresář pico-*displayDrivs* nebo adresář *.vscode*, tyto adresáře jsou v přílohách E.

6.1 Laboratorní úloha - GPIO

Cíle úlohy:

Cílem laboratorní úlohy je, aby se studenti naučili konfigurovat a ovládat binární vstupně/výstupní porty a využívat přerušení.

Zadání:

1. Vytvořte program, který bude pomocí tlačítka ovládat led diodu. Tlačítko je připojeno na pinu 10. Led dioda je připojena na pin 5.
2. Vytvořte program, který se bude chovat jako RS klopný obvod. Led dioda na pinu 4 bude reprezentovat signál Q a led dioda na pinu 5 bude reprezentovat signál $\overline{Q}$. Tlačítko na pinu 10 bude reprezentovat signál SET a tlačítko na pinu 11 RESET. Při sepnutí jednoho z tlačítek se vstoupí do přerušovací rutiny, kde se vykoná program reprezentující RS klopný obvod.

Návody k vypracování:

1. **Ovládání led diody tlačítkem**
   - Vytvořte adresář pro projekt ve vývojovém prostředí Visual Studio Code, nakopírujte do něj adresář s názvem *.vscode* a vytvořte soubory *main.c* a *CmakeLists.txt*.
   - Do souboru *CMakeLists.txt* nakopírujte následující kód.

```
cmake_minimum_required(VERSION 3.12)

include($ENV{PICO_SDK_PATH}/external/pico_sdk_import.cmake)

project(blink C CXX ASM)
set(CMAKE_C_STANDARD 11)
set(CMAKE_CXX_STANDARD 17)

pico_sdk_init()
add_executable(${PROJECT_NAME} main.c)
pico_add_extra_outputs(${PROJECT_NAME})
```

- V souboru *CMakeLists.txt* zavolejte funkci s parametry: *target_link_libraries(${PROJECT_NAME} pico_stdlib)*, která specifikuje soubory, které budou použity při linkování.
- Do souboru *main.c* přidejte knihovnu *#include "pico/stdlib.h"*.
- Zapište do příslušného konfiguračního registru, aby byl pin 5 ovládán pomocí sio.
- Nakonfigurujte pin 5 jako výstupní (využijte sio registry).
- Zapište do příslušného konfiguračního registru, aby pin 10 byl ovládán pomocí sio.
- Nastavte v pad registrech pin 10 jako vstupní.
- Nastavte, aby k pinu 10 byl připojen pull up rezistor.
- Doplňte kód, který zajistí, aby při sepnutí tlačítka led dioda svítila, při uvolnění tlačítka led dioda zhasla.

2. **RS klopný obvod**
   - Do souboru *main.c* přidejte knihovnu *#include "pico/stdlib.h"*. V souboru *CmakeLists.txt* zavolejte funkci s argumenty: *target_link_libraries(${PROJECT_NAME} pico_stdlib)*, která specifikuje soubory pro linker.
   - Zapište do příslušných konfiguračních registrů, aby piny 4, 5, 10 a 11 byly ovládány pomocí sio.
   - Nastavte piny 10 a 11 jako vstupní, piny 4 a 5 jako výstupní. Ke vstupním pinům připojte pull up rezistory.
   - Najděte v katalogovém listu registr VTOR, v němž je uložena adresa tabulky výjimek. Adresu (hodnotu v registru) přetypujte na ukazatel na ukazatel na funkci *(irq_handler_t*)*, který si nadefinujte následovně *typedef void (*irq_handler)(void);*. Tento ukazatel už lze indexovat. Pozn.: Prvních 16 výjimek v tabulce slouží pro procesor ARM Cortex-M0+ z toho plyne, že při indexaci je potřeba přičíst hodnotu 16. Pozn.: Přerušení v RP2040 jsou číslovány dle tabulky v katalogovém listu v kapitole 2. 3. 2. Pozn.: Ve výchozím nastavení adresa v registru VTOR ukazuje na začátek paměti RAM.
   - Na adresu, kde je uložena adresa přerušovací rutiny obsluhující přerušení od GPIO (správný index v tabulce výjimek), zapište adresu přerušovací rutiny, kterou si nadefinujte jako *void gpio_irq_callback()*.
   - V této přerušovací rutině zjistěte, které tlačítko způsobilo přerušení a nastavte na piny Q a $\overline{Q}$ správné hodnoty.
   - V přerušovací rutině je také nutno potvrdit (acknowledge), že přerušení bylo zpracováno, a to zápisem 1 do INTR1 registru (pro správný pin a typ přerušení).

- Povolte přerušení od GPIO 10 a 11 v PROC0_INTE1 registru.
- Povolte přerušení od GPIO v NVIC_ISER registru.

6.2 Laboratorní úloha - Displej, modul RTC

Cíle úlohy: Cílem této laboratorní úlohy je, aby se studenti naučili využívat periferií RTC (Real-time Clock) a ovládat displej. Pro realizaci úlohy bude využito SDK.

Zadání:

1. Seznamte se s dokumentací A ST7789 displeje a vytvořte program, který na displeji zobrazí text.
2. Vytvořte program, který bude na displeji zobrazovat aktuální datum a čas ve tvaru viz. obrázek 6.1.

Návody k vypracování:

1. **Seznámení se z displejem**
   - Založte si nový projekt, viz. Laboratorní úloha - GPIO.
   - Přidejte do projektu adresář *pico-displayDrivs* E.2, který obsahuje adresář s upravenou GFX knihovnu a také přizpůsobovací program pro daný typ displeje (st7789).
   - Do souboru přidejte funkci s argumenty *target_link_libraries(${PROJECT_NAME} gfx st7789 hardware_rtc*), aby linker mohl připojit dané knihovny.
   - V souboru *CMakeLists.txt* zavolejte funkci *add_subdirectory(pico-displayDrivs)*, jejíž parametr definuje relativní cestu k daným knihovnám, tímto se také vykonají příkazy v souboru CMakeLists.txt ve složce s danou relativní cestou.
   - Do souboru *main.c* přidejte následující hlavičkové soubory *#include "gfx.h"* a *#include "st7789.h"*.
   - Inicializujte komunikaci s displejem.
   - Vytvořte Frame buffer (oblast v paměti ve které jsou uložené data, která budou zobrazena na displeji).
   - Vytisknete naformátovaný text.
   - Funkcí *GFX_flush()* zobrazte na displeji data z tzv. Frame buffer.
   - Vytvořte program, ve kterém se bude kontinuálně tisknout text na stejnou pozici na displeji.
2. **RTC modul**
   - Vyjděte z předchozí úlohy.
   - Do souboru *main.c* přidejte hlavičkový soubor *#include "hardware/rtc.h"*. Do souboru CmakeLists.txt přidejte parametr *hardware_rtc* funkci

target_link_libraries().

- Vytvořte proměnnou typu *datetime_t* a do její členských proměnných zapište aktuální datum a čas.
- Inicializujte periferii RTC funkcí *rtc_init()*.
- Nastavte aktuální čas a datum uložené v proměnné typu *datetime_t* pomocí vhodné funkce, kterou najdete v katalogovém listu Pico SDK.
- Vytvořte funkci s parametrem typu *datetime_t*, ve které tiskněte čas a datum ve tvaru viz. 6.1.
- Ve smyčce vyčítejte čas a datum a volejte funkci pro tisk na displej.

Obr. 6.1: Příklad zobrazení aktuálního data a času na displeji

6.3 Laboratorní úloha - Timer a využití přerušení

Cíle úlohy:

Cílem této laboratorní úlohy je seznámit studenty s periferiemi typu časovač a využívat přerušení. Pro realizaci úlohy bude využito SDK.

Zadání úlohy:

1. Napište program, který bude generovat periodický obdélníkový signál s periodou 1 s a střídou 50 % s využitím periferie Timer.
2. Napište program, který bude zobrazovat na displeji počet sepnutí tlačítka.
3. Upravte program z předchozí úlohy, tak aby se na displeji zobrazoval skutečný počet stisknutí tlačítka.

Návody k vypracování:

1. **Program generující periodický obdélníkový signál**
   - Založte si nový projekt, viz. Laboratorní úloha - GPIO.
   - Do souboru *main.c* přidejte hlavičkový soubor *#include "pico/stdlib.h"*. V souboru *CmakeLists.txt* zavolejte následující funkci s argumenty *target_link_libraries(${PROJECT_NAME} pico_stdlib)*, která specifikuje soubory pro linker.
   - Napište callback funkci, která bude invertovat stav led diody, připojené na pinu 5.
   - Nastavte pin 5 jako výstupní.
   - Pomocí funkce *add_repeating_timer_ms()* nakonfigurujte časovač a zaregistrujte callback funkci.
2. **Program zobrazující počet sepnutí tlačítka**
   - Napište callback funkci, ve které se bude počítat počet sepnutí tlačítka.
   - Nastavte pin 10 jako vstupní, připojte pull up rezistor.
   - Pomocí funkce *gpio_set_irq_enabled_with_callback()*, nastavte přerušení a zaregistrujte callback funkci. Do přerušení se vstoupí při sepnutí tlačítka. V přerušovací rutině inkrementuje globální proměnnou indikující počet sepnutí tlačítka.
   - V hlavní smyčce kontrolujte, zda došlo ke změně počtu sepnutí tlačítka, jestliže ano, zobrazte novou hodnotu na displeji. Pro zobrazování na displeji vyjděte z předchozí úlohy.
   - Odůvodněte proč zobrazený počet sepnutí tlačítka neodpovídá skutečnému počtu sepnutí.
3. **Eliminace zákmitů při sepnutí tlačítka**
   - Vyjděte z předchozí úlohy.

- Po sepnutí tlačítka v přerušení s využitím funkce *add_alarm_in_ms()* po zadanou dobu ignorujte další změny na pinu.
- Obdobně při uvolnění tlačítka po danou dobu ignorujte další změny tlačítka.

6.4 Laboratorní úloha - ADC

Cíle úlohy:

Cílem této laboratorní úlohy je, aby se studenti naučili pracovat s A/D převodníkem.

Zadání úlohy:

1. Napište program, který bude na displeji zobrazovat aktuální napětí na pinu 26. Využijte AD převodníku pracujícího v jednorázovém režimu.
2. Napište program, který bude na displeji zobrazovat aktuální napětí na pinu 26. Využijte AD převodníku pracujícího v jednorázovém režimu, avšak místo čekání, než se dokončí převod, nakonfigurujte ADC, aby se po dokončení jednoho převodu vyvolal požadavek na přerušení.

Návody k vypracování:

1. **Zobrazování napětí na analogovém pinu**
   - Založte si nový projekt, viz. Laboratorní úloha - GPIO.
   - Zapište do příslušného konfiguračního registru, aby ADC zahájil startovací sekvenci.
   - Zapište do příslušného konfiguračního registru, aby ADC převáděl napětí na pinu 26 na digitální hodnotu.
   - Ve smyčce čekejte než bit *CS.READY* bude nastaven na hodnotu 1, ADC bude tehdy připraven na zahájení převodu.
   - Napište program, který ve smyčce zahájí jednorázový převod, počká, jakmile se převod dokončí, vyčte výsledek a zobrazí napětí ve [V] na displej.
   - Při zobrazování na displej vyjděte z předchozích úloh.
2. **Úprava předchozí úlohy, po dokončení převodu se vstoupí do přerušení**
   - Vyjděte z předchozí úlohy.
   - Povolte přerušení v INTE registru.
   - Zapište do správného konfiguračního registru, aby se každý výsledek převodu zapsal do FIFO registru. Nastavte, aby se vystavil požadavek na

přerušení při jednom vzorku v tomto registru.

- Definujte přerušovací rutinu: *void adc_irq_callback()* a zaregistrujte ji do tabulky výjimek, vyjděte z prvního cvičení. V této přerušovací rutině vyčtěte výsledek z FIFO registru do globální proměnné. Vyčtením se zároveň zruší požadavek na přerušeni.
- Nakonec povolte přerušení od ADC v NVIC_ISER registru.
- Ve smyčce zahajujte jednorázový převod a zobrazujte na displeji napětí na pinu v rozsahu 0 až 3,3 V.

6.5 Laboratorní úloha - ADC a DMA, PWM

Cíle úlohy: Cílem této laboratorní úlohy je seznámit studenty s využíváním ADC s kombinací s DMA a v druhé části naučit studenty využívat periferii PWM. Pro realizaci úlohy bude využito SDK.

Zadání úlohy:

1. Napište program, který převede 100 vzorků vstupního analogového napětí na digitální hodnoty a pomocí DMA tyto hodnoty bude ukládat do pole. ADC bude pracovat v kontinuálním režimu. Po dokončení 100 převodů, hodnoty napětí zobrazte na displeji. Využijte Pico SDK.
2. Vytvořte program, který bude na displeji zobrazovat aktuální frekvenci obdélníkového periodického signálu na pinu 13.

1. **Program využívající ADC a DMA**
   - Založte si nový projekt, viz. Laboratorní úloha - GPIO.
   - Do souboru *main.c* přidejte následující hlavičkové soubory *#include "hardware/adc.h"* a *#include "hardware/dma.h"*. V souboru *CmakeLists.txt* přidejte funkci *target_link_libraries()* následující argumenty: *hardware_adc* a *hardware_dma*.
   - Zavolejte funkci *adc_init()* pro inicializaci ADC.
   - Zavolejte funkci, která nastaví pin 26 jako vstup do ADC.
   - Vhodně nastavte děličku frekvence pro ADC.
   - Pomocí funkce *adc_fifo_setup()* zajistěte, aby se každá hodnota zapsala do FIFO registru, povolte DREQ, proveďte operaci posunu, aby vzorky měly 8 bitů.
   - Zavolejte funkci *dma_channel_get_default_config()*, která má jako návratovou hodnotu strukturu s výchozími konfiguracemi DMA kanálu. Zavolejte vhodné funkce, které zajistí následující nastavení: DMA přenos po 8 bitech, inkrementace cílové adresy, startovací adresa zůstane stejná, zahájení DMA přenosu při nastavení příznaku *DREQ_ADC*.

- Zavolejte funkci *dma_channel_configure()* s vhodnými parametry. Počet přenosů zvolte 100.
- Zahajte ADC v kontinuálním režimu.
- Zavolejte funkci *dma_channel_wait_for_finish_blocking()*, která zajistí dokončení DMA přenosu.
- Ukončete činnost ADC.
- Zobrazte posupně na displeji přepočítané hodnoty z pole na napětí v rozsahu 0 až 3,3 V.

2. **Měření frekvence**
   - Do souboru *main.c* přidejte hlavičkový soubor *#include "hardware/pwm.h"*. V souboru *CmakeLists.txt* přidejte funkci *target_link_libraries()* argument *hardware_pwm*.
   - Zavolejte funkci *gpio_set_function()*, která zajistí, aby byl pin 13 ovládán pomocí periferie PWM.
   - Zavolejte funkci *pwm_get_default_config()*, která má jako návratovou hodnotu strukturu s výchozími konfiguracemi periferie PWM.
   - Zavolejte vhodnou funkci, která zajistí, že čítač periferie PWM bude inkrementován při nástupné hraně na pinu 13.
   - Zavolejte funkci *pwm_init()*, s argumenty příslušného PWM modulu (*slice*), strukturou typu *pwm_config*, která obsahuje vytvořená nastavení a s hodnotou *false*.
   - Vytvořte funkci *get_freq()* s jedním parametrem typu *uint16_t*. Funkce bude vracet počet nástupných hran, které byly detekovány během specifikovaného intervalu (parametr funkce).
   - Funkci volejte ve smyčce. Její návratovou hodnotu přepočtete, tak aby reprezentovala frekvenci měřeného signálu.
   - Frekvenci zobrazte na displeji. Pro zobrazování na displeji vyjděte z předchozích úloh.

6.6 Laboratorní úloha - SPI

Cíle úlohy: Cílem této laboratorní úlohy je, seznámit studenty s komunikačním protokolem SPI. Pro realizaci úlohy bude využito SDK.

Zadání úlohy:

1. Napište program, který rozsvítí všechny pixely na displeji s ovladačem ST7789VW. Displej komunikuje s RP Pico pomocí SPI protokolu. Zapojení signálů je následující: SCK - GIPO 18, MOSI - GIPO 19, DC (Data Command) - GIPO

20, RST (Reset) - GIPO 21. Displej má rozlišení 240 x 240 pixelů.

Návody k vypracování:

1. **Rozsvícení všech pixelů na displeji**
   - Založte si nový projekt, viz. Laboratorní úloha - GPIO,
   - Přidejte do souboru *main.c* hlavičkové soubory *#include "hardware/spi.h"* a *#include "pico/stdlib.h"*. V souboru *CmakeLists.txt* zavolejte následující funkci s argumenty
     target_link_libraries(${PROJECT_NAME} pico_stdlib hardware_spi), která specifikuje soubory pro linker.
   - Vytvořte pole hodnot *start_sequence* typu *uint8_t* ve kterém bude sekvence příkazů s případnými parametry pro inicializaci displeje. Pole bude mít následující strukturu: Příkaz, počet parametrů + hodnota indikující že před posláním dalšího příkazu je nutno počkat, argumenty, doba před posláním dalšího příkazu, další příkaz, ...
   - Pole bude obsahovat následující příkazy s parametry. V katalogovém listu je specifikováno, jestli před posláním dalšího příkazu se má počkat a případně jakou dobu.
     - SWRESET - bez parametrů
     - SLPOUT - bez parametrů
     - COLMOD - s 1 parametrem který nastavý 16 bitů pro pixel a 65K hodnot pro pixel.
     - CASET - se 4 parametry, XS (XSTART) nastavte na 0, XE (XEND) nastavte na 239.
     - RASET - se 4 parametry, YS (YSTART) nastavte na 0, YE (YEND) nastavte na 239.
     - INVON - bez parametrů, tento příkaz slouží k tomu, aby byl použit RGB barevný model (černá barva bude mít hodnotu 0), místo CMY modelu. Po tomto příkazu počkejte 10 ms před poslání dalšího příkazu.
     - DISPON - bez parametrů.
   - Inicializujte periferii SPI s přenosovou rychlostí 4 MBd.
   - Nastavte, aby GPIO 18 a 19 byly ovládány periferií SPI.
   - Pomocí funkce *spi_set_format()* nastavte 8 bitové přenosy, klidovou úroveň hodinového signálu do log. 1, čtení hodnot při přechodu hodinového signálu z aktivní do klidové úrovně a MSB první.
   - Inicializujte piny 20 (DC) a 21 (RST) a nastavte je jako výstupní.
   - Proveďte hardwarový reset dle katalogového listu (signál RESX).
   - Napište kód, který bude procházet pole *start_sequence* a bude posílat jed-

notlivé příkazy a jejich argumenty pomocí SPI komunikačního protokolu, případně zavolá funkci *sleep_ms()*, pokud se má počkat před posláním dalšího příkazu. Při posílaní příkazu je nutné mít signál DC v log. 0, při posílání argumentů v log. 1.

- Pošlete příkaz *RAMWR* pomocí SPI komunikačního protokolu (DC signál musí být v log. 0) a následně pošlete jeho argumenty (DC signál musí být v log. 1).

Závěr

Cílem této bakalářské práce se bylo seznámit s architekturou ARMv6-M, procesorem ARM Cortex M0+ a mikrokontrolérem RP2040. Dále vytvořit laboratorní úlohy do cvičení pro předmět BPC-MIC a navrhnout a realizovat pro ně potřebné přizpůsobovací obvody.

Na začátku teoretické části se čtenář seznámí s procesorem ARM Cortex M0+, na němž je postaven mikrokontrolér RP2040. Procesor je zde zařazen do rodiny ARM Cortex M procesorů, je porovnán s ostatními procesory této rodiny a je popsána jeho architektura.

Ve čtvrté kapitole je po teoretické části v textu uveden postup instalace celého toolchain včetně vývojového prostředí Visual Studio Code. Při instalaci toolchain bylo nejprve postupováno podle oficiálního návodu *Getting started with Raspberry Pi Pico*, ale podle tohoto návodu se nepodařilo zprovoznit debugování, které vyžadovalo instalaci dalšího software. Nakonec byl zvolen jiný postup, a to podle článku na webových stránkách Raspberry Pi Pico [16]. Tento článek poskytuje jednoduchý způsob instalace celého toolchain na operační systém Windows.

Hlavní částí této práce byl návrh laboratorních úloh. Úlohy mají danou strukturu, kdy na začátku je vždy popsán cíl dané úlohy, následuje stručné zadání a detailní návod k vypracování, podle kterého budou studenti postupovat na cvičeních. V první úloze (v pořadí, jak je uvedeno v práci) se studenti seznámí se založením projektu a s využitím GPIO. V druhé části této úlohy se také naučí využívat přerušení od GPIO. Cílem druhé úlohy je, aby se studenti seznámili s využíváním LCD displeje za pomoci knihovny, kterou si přidají do projektu a pro usnadnění práce s touto knihovnou mohou využít dokumentaci, která je přílohách této práce. Displej je také použit v následujících úlohách pro zobrazování výstupů. Ve druhé části této úlohy se studenti seznámí s hodinami reálného času. Ve třetí laboratorní úloze se studenti seznámí s periferií Timer a zjistí, že při sepnutí tlačítka dojde k zákmitům a tyto zákmity softwarově odstraní. Následující úloha seznámí studenty s využitím A/D převodníku v jednorázovém režimu. V páté úloze se využije nabytých znalostí z předchozí úlohy a s A/D převodníkem bude pracováno v kontinuálním režimu, přičemž se studenti zároveň seznámí s možností přenosu dat pomocí DMA kanálu. Ve druhé části bude využita periferie PWM pro změření frekvence periodického obdélníkového signálu. Poslední úloha v pořadí uvedeném v textu se zaměřuje na komunikaci SPI. Studenti se už naučili pracovat s knihovnou pro ovládání displeje, a v této úloze budou posílat příkazy displeji přímo s využitím periferie SPI. Cílem je pouze rozsvítit všechny pixely danou barvou.

Nejprve byly úlohy realizovány na nepájivém poli, ale poté byla vytvořena deska plošných spojů, na které byly úlohy otestovány. Všechny části DPS se ukázaly být

funkční. Výstup z generátoru obdélníkového signálu je v přílohách.

Pro využití navržené DPS ve výuce by bylo vhodné přidat bezpečnostní opatření, protože například při nastavení GPIO jako výstupu by při zmáčknutí tlačítka mohlo dojít ke zkratu se zemí. Také by pro navrženou DPS pro využití ve výuce měla být vytvořena krabička, například na 3D tiskárně, nakonec by samotná DPS mohla být rozšířena o další komponenty.

Literatura

[1] Arm CPU Architecture – Arm®. Online. Building the Future of Computing – Arm®. C1995-2024. Dostupné z:https://www.arm.com/architecture/cpu. [cit. 2024-04-02].

[2] Documentation – Arm Developer. Online. Arm Developer. 2009. Dostupné z: https://developer.arm.com/documentation/dui0662/latest/. [cit. 2023-12-25].

[3] Armv6-M Architecture Reference Manual. Online. Arm Developer. 2007. Dostupné z: https://developer.arm.com/documentation/ddi0419/latest/. [cit. 2024-02-13].

[4] The Arm® Cortex®-M0+ Nested Vector Interrupt Controller diagram. Online. In: Home - Developer Help. C2023. Dostupné z: https://microchipdeveloper.com/xwiki/bin/view/products/mcu-mpu/32bit-mcu/sam/arm-cortex-m0/nvic/. [cit. 2023-12-25].

[5] Raspberry Pi Documentation - Raspberry Pi Pico and Pico W. Online. RASPBERRY PI. Raspberry Pi. C2012-2023. Dostupné z: https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html. [cit. 2023-12-22].

[6] RP2040 Datasheet A microcontroller by Raspberry Pi. Online. Raspberry Pi. C2020-2023. Dostupné z:https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf. [cit. 2023-12-25].

[7] Introduction to SPI Interface. Online. Mixed-signal and digital signal processing ICs | Analog Devices. 2018. Dostupné z: https://www.analog.com/media/en/analog-dialogue/volume-52/number-3/introduction-to-spi-interface.pdf. [cit. 2024-01-02].

[8] MicroPython - Python for microcontrollers. Online. 2014. Dostupné z: https://micropython.org/. [cit. 2023-12-25].

[9] Definition of Buck-Boost Converter | Analog Devices. Online. Mixed-signal and digital signal processing ICs | Analog Devices. 2020. Dostupné z: https://www.analog.com/en/design-center/glossary/buck-boost.html. [cit. 2023-12-27].

[10] Raspberry Pi Pico Datasheet - pico-datasheet.pdf. Online. Raspberry Pi. C2020-2023. Dostupné z: https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf. [cit. 2023-12-27].

[11] Raspberry Pi Pico SDK: Raspberry Pi Pico SDK. Online. C2020. Dostupné z: https://cec-code-lab.aps.edu/robotics/resources/pico-c-api/index.html. [cit. 2023-12-25].

[12] About CMake. Online. CMake - Upgrade Your Software Build System. 2015. Dostupné z: https://cmake.org/about/. [cit. 2023-12-25].

[13] Getting started with Raspberry Pi Pico. Online. Raspberry Pi. C2020-2023. Dostupné z: https://datasheets.raspberrypi.com/pico/getting-started-with-pico.pdf?_gl=1*1eva3lh*_ga*MjA3NTYyODUwNS4xNzAyMTk5MTM0*_ga_22FD70LWDS*MTcwMjE5OTEzMy4xLjAuMTcwMjE5OTEzMy4wLjAuMA. [cit. 2023-12-25].

[14] TLC555 LinCMOS™ Technology Timer datasheet (Rev. J) - tlc555.pdf. Online. Analog | Embedded processing | Semiconductor company | TI.com. C2024. Dostupné z: https://www.ti.com/lit/ds/symlink/tlc555.pdf?ts=1715711893759&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252Fde-de%252FTLC555. [cit. 2024-05-15].

[15] Raspberry Pi Pico C/C++ SDK. Online. Raspberry Pi. C2020-2023. Dostupné z: https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf?_gl=1*60fjha*_ga*MTQ2ODkwNDkzMi4xNzAzNjcxMTk5*_ga_22FD70LWDS*MTcwNDIwNzE4NS4zMi4wLjE3MDQyMDcxODkuMC4wLjA.. [cit. 2024-01-02].

[16] Raspberry Pi Pico Windows Installer - Raspberry Pi. Online. In: Raspberry Pi. 2023. Dostupné z: https://www.raspberrypi.com/news/raspberry-pi-pico-windows-installer/. [cit. 2023-12-25].

[17] ST7789VW. Online. Waveshare Wiki. 2017. Dostupné z: https://www.waveshare.com/w/upload/a/ad/ST7789VW.pdf. [cit. 2024-02-01].

[18] 13-inch-colorful-tft-module-spec. Online. LaskaKit. 2017. Dostupné z: https://www.laskakit.cz/user/related_files/13-inch-colorful-tft-module-spec.pdf. [cit. 2024-02-01].

[19] GitHub - tvlad1234/pico-displayDrivs: Display drivers for pico-sdk. Online. GitHub: Let's build from here · GitHub. 2019. Dostupné z:https://github.com/tvlad1234/pico-displayDrivs. [cit. 2024-02-01].

[20] GitHub - adafruit/Adafruit-GFX-Library. Online. GitHub: Let's build from here · GitHub. 2012. Dostupné z: `https://github.com/adafruit/Adafruit-GFX-Library`. [cit. 2024-02-01].

Seznam příloh

A Dokumentace základních funkcí pro ovládání displeje

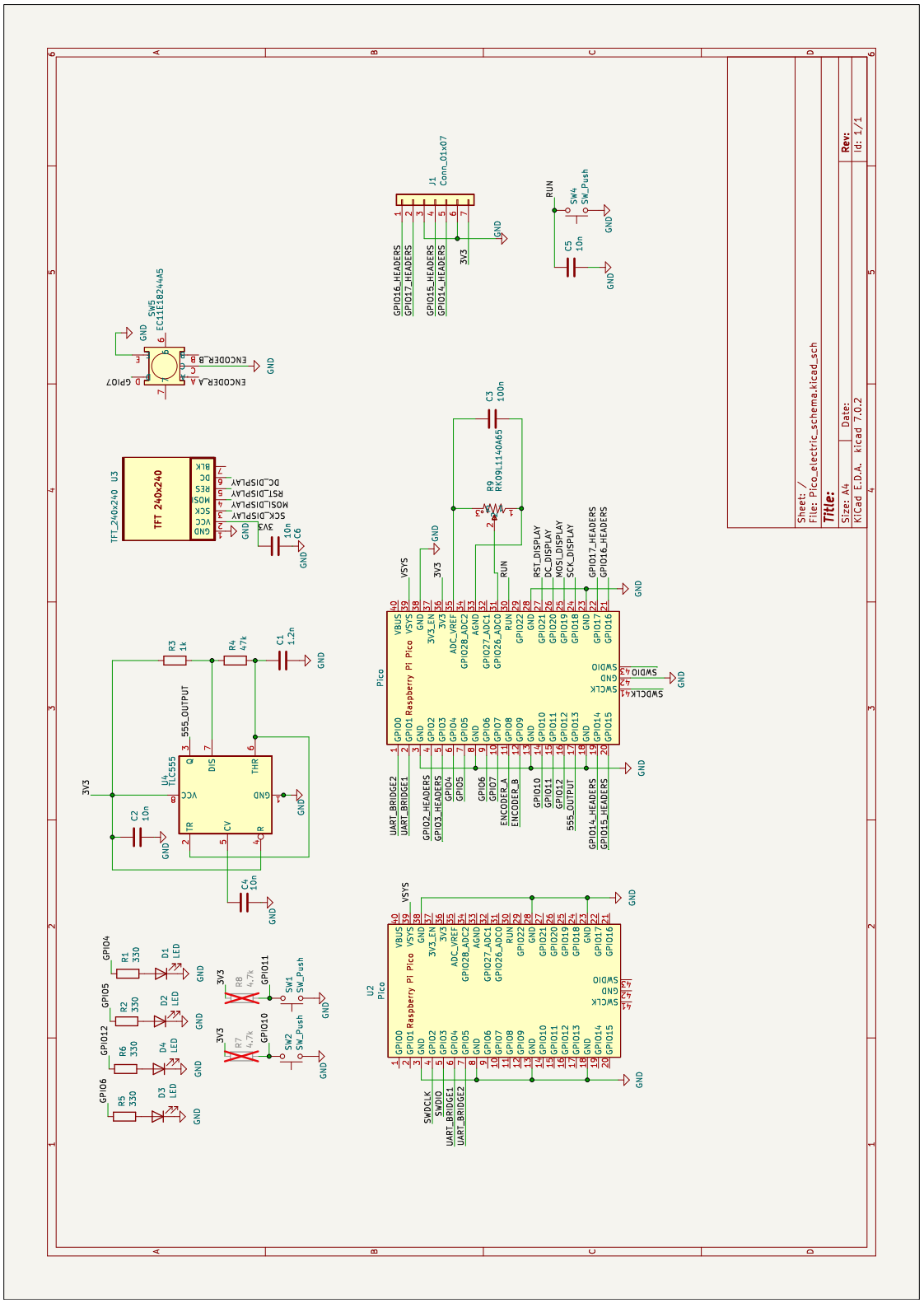
- `LCD_initDisplay(uint16_t width, uint16_t height)` - slouží pro inicializaci komunikace a specifikování velikosti displeje.
- `LCD_setRotation(uint8_t m)` - nastaví orientaci displeje rotací počátku, parametr m může nabývat hodnot 1 - 4 pro rotaci o 0°, 90°, 180°, nebo 270°.
- `GFX_fillScreen(uint16_t color)` - vyplní celou obrazovku danou barvou, smaže co je právě zobrazeno.
- `GFX_setClearColor(uint16_t color)` - nastaví barvu pro funkci *GFX_clearScreen()*.
- `GFX_clearScreen()` - vyplní obrazovku výchozí černou barvou, nebo barvou danou *GFX_setClearColor(uint16_t color)*, smaže co je právě zobrazeno.
- `GFX_drawPixel(uint16_t x, uint16_t y, uint16_t color)` - vykreslí jeden pixel s danou polohou a barvou.
- `GFX_drawRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color)` vykreslí danou barvou obdélník s počátkem v (x, y) šířkou w a výškou h.
- `GFX_drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1, uint16_t color)` vykreslí danou barvou úsečku mezi body (x0, y0) a (x1, y1).
- `GFX_drawChar(int16_t x, int16_t y, unsigned char c, uint16_t color, uint16_t bg, uint8_t size_x, uint8_t size_y)` - zobrazí jeden unsigned char s danou polohou, barvou, barvou pozadí (backgroundcolor) a s velikostí v ose x a y.

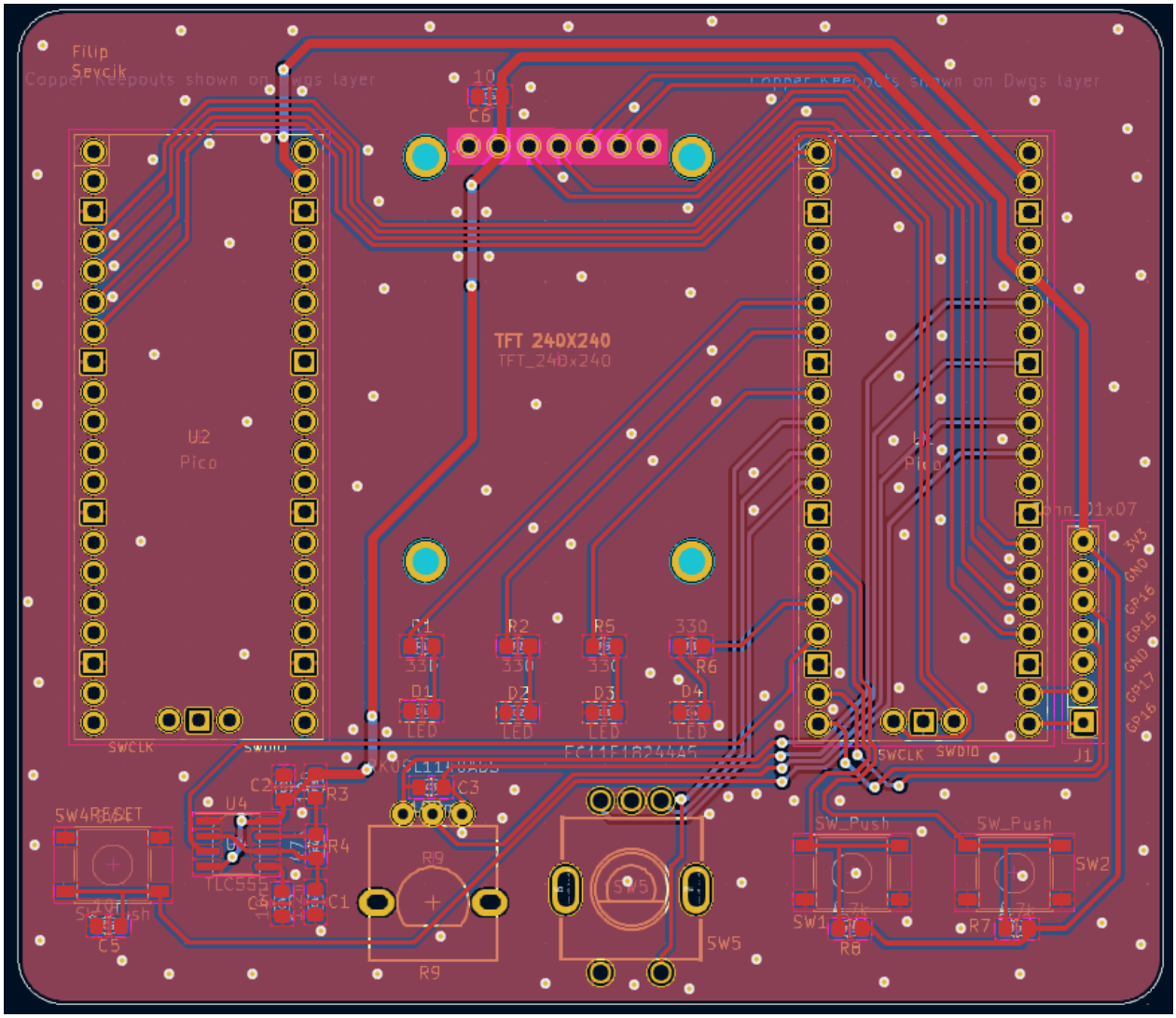
Pro tisk textu, místo jedné funkce s více parametry je barvu, polohu a další vlastnosti možné nastavit separátními funkcemi. Text je poté zobrazen funkcí *GFX_printf()*.

- `GFX_setTextColor(uint16_t color)` - nastaví barvu textu.
- `GFX_setCursor(int16_t x, int16_t y)` - nastaví kurzor pro tisk na specifikované souřadnice, přičemž počátek souřadnicového systému (0, 0) je umístěn do levého horního rohu a displej má velikost 240x240 pixelů.
- `GFX_setTextBack(uint16_t color)` - nastaví barvu pozadí textu.
- `GFX_write(uint8_t c)` - zobrazí unsigned char v daném formátu.
- `GFX_printf()` - zobrazí text v daném formátu.

Ve výchozím stavu tato knihovna zapisuje pixely přímo na obrazovku. Pro aplikace, kde je požadována rychlost lze vytvořit tzv. frame buffer funkcí *GFX_createFramebuf()*, do kterého se budou zapisovat pixely, místo toho aby se přímo zobrazovali na displeji. Buffer lze poté zobrazit na displeji funkcí *GFX_flush()*. Zavoláním funkce *GFX_destroyFramebuf()* se ukončí zápis pixelů na frame buffer. [20]

B Schémata k DPS

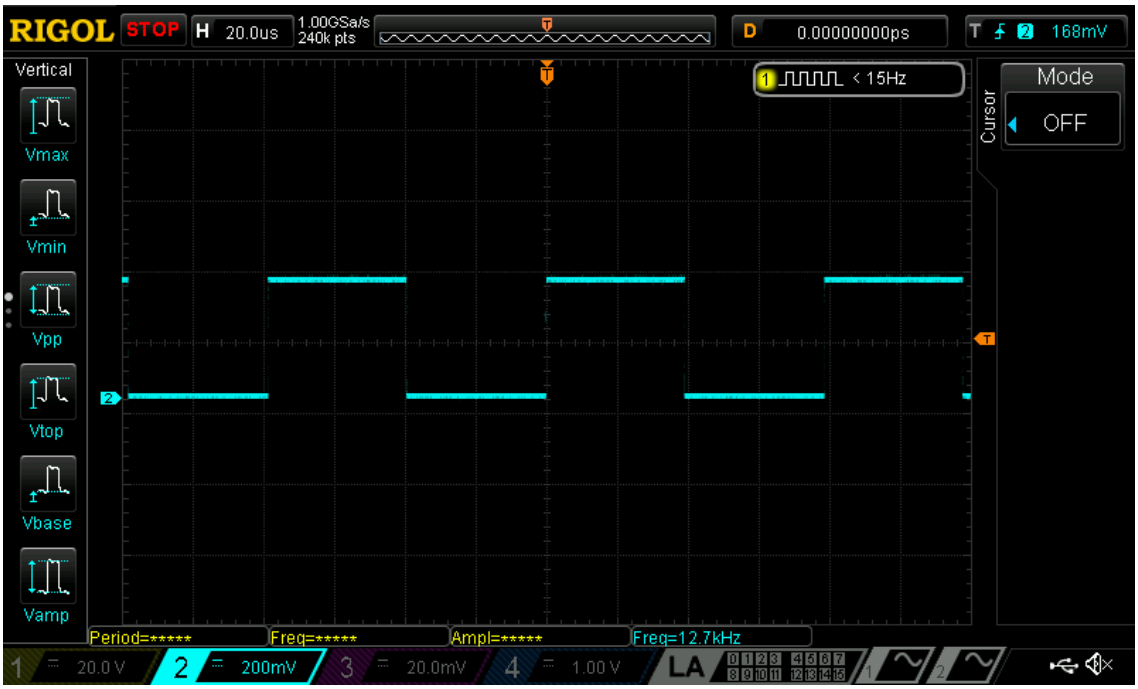



Filip
Sevcik
TFT 240X240
TFT_240x240
U2
Pico
SWCLK
SWDIO
R1
R2
R5
R6
330
D1
D2
D3
D4
LED
SW_Push
SW1
SW2
SW5
R7
R8
R9
C1
C2
C3
C5
C6
U4
TLC555
J1
3V3
GND
GP16
GP15
GP17

C Tabulka použitých komponent

Označení	Footprint	Počet	Hodnota	Číslo souč.
C1	0805	1	1.2n	FM21X122K251PXG
C2, C4, C5, C6	0805	4	10n	CL21B103KBANNNC
C3	0805	1	100n	CC0805KRX7R9BB104
D1, D2, D3, D4	0805	4	LED	KT-0805Y
J1	PinSocket_1x07_P2.54mm_Vertical	1	Conn_01x07	KLS S1G20C
R1, R2, R5, R6	0805	4	330	0805W8F3300T5E
R3	0805	1	1k	0805W8F1001T5E
R4	0805	1	47k	0805W8F4702T5E
R9	RES-ADJ-TH_RK09L1140A65	1	RK09L1140A65	RK09L1140A65
SW1, SW2, SW4	SW_PushK2-1157SP-I4SW-01	3	SW_Push	K2-1157SP-I4SW-01
SW5	SW-TH_EC11E18244A5	1	EC11E18244A5	EC11E18244A5
U4	SOIC-8_3.9x4.9mm_P1.27mm	1	TLC555	TLC555IDR
U1, U2	Pico_Female_Headers	2	Pico	Raspberry Pi Pico
U3	st7789_display	1	TFT_240x240	1.3"240x240 TFT IPS Barevný displej ST7789, SPI

D Výstup z generátoru obdélníkového signálu



E Elektronické přílohy

E.1 Vzorové řešení laboratorních úloh uložené na přiloženém USB flash disku

E.2 Knihovna pro st7789 displej uložená na přiloženém USB flash disku

E.3 Adresář .vscode s nastaveními pro založení nového projektu uložená na přiloženém USB flash disku

E.4 Soubory nutné pro výrobu DPS (Gerber, Drill, Placement a BOM) uložené na přiloženém USB flash disku

E.5 Projekt v programu KiCad 7.0 uložený na přiloženém USB flash disku