VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV MIKROELEKTRONIKY
DEPARTMENT OF MICROELECTRONICS

VÝUKOVÁ DESKA PRO PROCESORY STM
EDUCATIONAL BOARD FOR STM PROCESSORS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Matěj Očenášek**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Pavel Šteffan, Ph.D.**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Mikroelektronika a technologie**

Ústav mikroelektroniky

Student: Matěj Očenášek ***ID:*** 203309
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Výuková deska pro procesory STM

POKYNY PRO VYPRACOVÁNÍ:

V rámci bakalářské práce navrhněte rozšiřující výukovou desku pro procesory STM kompatibilní s platformou Nucleo. Deska musí obsahovat OLED displej, paměť EEPROM, Wi-Fi modul, sadu LED diod, segmentový displej a další komponenty. V rámci bakalářské práce vytvořte ukázkové výukové úlohy a základní manuál pro používání navržené desky a jednotlivé periferie.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: doc. Ing. Pavel Šteffan, Ph.D.

doc. Ing. Jiří Háze, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tato práce se zabývá návrhem výukové desky pro mikrokontrolery STM32. Tato výuková deska obsahuje základní typy periferií jako OLED displej, EEPROM paměť, LED diody, nebo sedmi segmentové displeje. Teoretická část je zaměřená na popis procesoru STM32F303 a vývojové desky Nucleo-64, na které je tento procesor osazen, popisuje pak zejména jeho vlastnosti. Mimo procesor STM32F303 jsou pak v teoretické části blíže popsány jednotlivé periferie výukové desky. Praktická část se zabývá návrhem blokového schématu, schématu a návrhem desky plošných spojů.

KLÍČOVÁ SLOVA

Mikrokontroler, STM32F303, ESP8266, výuková deska, Nucleo-64, PWM, EEPROM, periferie, sběrnice.

ABSTRACT

This work deals with design of educational board for STM32 microcontrollers. This educational board contains basic types of peripherals such as OLED display, EEPROM, LED or seven segment displays. The theoretical part focuses on the description of the STM32F303 processor and the Nucleo-64 development board on which the processor is installed. It describes in particular its properties. In the theoretical part, besides the STM32F303 processor, the individual parts of the educational board are described. The practical part deals with the design of the block diagram, the scheme and the design of the printed circuit board.

KEYWORDS

Microcontroller, STM32F303, ESP8266, educational board, Nucleo-64, PWM, EEPROM, periphery, bus

OČENÁŠEK, M. *Výuková deska pro procesory STM*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav mikroelektroniky, 2020. 139 s. Bakalářská práce. Vedoucí práce: doc. Ing. Pavel Šteffan, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	*„Matěj Očenášek“*
VUT ID studenta:	*„203309“*
Typ práce:	*„Semestrální práce“*
Akademický rok:	2019/20
Téma závěrečné práce:	*„Výuková deska pro procesory STM“*

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 jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.

V Brně dne: ***30. listopadu 2019*** ……

Podpis autora

PODĚKOVÁNÍ

Děkuji vedoucímu semestrální práce doc. Ing. Pavel Šteffan, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé semestrální práce. Také bych chtěl poděkovat panu Martinu Mikulkovi za odbornou pomoc.

OBSAH

ÚVOD

Mikrokontroler, často označovaný zkratkou MCU, je jednočipový počítač, určený převážně pro aplikace vykonávající jednu primární funkci jako například regulace teploty vody v nádrži, kontrola výšky vody v nádrži, nebo automatické ovládání rolet. Mikrokontroler obsahuje všechny základní části počítače, jako jsou centrální výpočetní jednotka, paměť typu RAM a ROM, datové sběrnice, vstupní/výstupní brány, vnitřní hodinový signál nebo také A/D a D/A převodníky. Jedním ze základních druhů dělení mikrokontrolerů je dělení podle šířky slova v bitech, tedy 4, 8,16, 32bitové. Dále je možné mikrokontrolery podle jejich instrukční sady jako RISC, CISC nebo VLIW.

V rámci této práce byl jako primární řídící člen zvolen mikrokontroler STM32F303 od společnosti STMicroelectronics řady STM32 osazený na vývojové DPS Nucleo-64. Nucleo-64 má přístup ke všem periferiím výukové DPS, ke všem využívaným sběrnicím a také výukovou DPS napájí. Jako alternativní ovládací člen lze také využít jednu z periferií, a to modul ESP8266, využívající mikroprocesor Tensilica L106 32-bit, tento modul má přístup pouze k omezenému množství sběrnic a pouze vybraným periferiím výukové DPS. Taktéž je možno modul ESP8266 využít pro napájení výukové DPS. Oba uvedené mikrokontrolery využívají instrukční sadu RISC a jsou postaveny na architektuře typu ARM.

Cílem této práce je seznámení studentů s prací s mikrokontrolery z rodiny STM32. Tohoto cíle bude dosaženo realizací DPS speciálně navržené pro účely výuky. Společně s DPS budou vytvořeny demonstrativní úlohy využívající všechny dostupné periferie navržené DPS. Jako komunikační sběrnice byly pro výuku zvoleny nejčastěji využívané typy sběrnic jak u procesorů s 32bitovým slovem, tak u nižších, a to sériové sběrnice SPI, I^2C a UART. Všem sběrnicím byly na DPS vyvedeny hřebínky, které jsou určeny pro sledování časových průběhů digitálního signálu pomocí osciloskopu. U sběrnice I^2C byly navíc přidány měnitelné pull-up rezistory pro SDA i SCL, aby bylo možné sledovat změnu průběhu digitálního signálu po stránce rychlosti komunikace, nebo tvaru samotného signálu. Navíc je díky digitálnímu izolátoru a optočlenům možné sledovat průběh digitálního signálu po galvanickém oddělení. Stejně tak, jako byly hřebínky na DPS vyvedeny pro sériové sběrnice, byly hřebínky vyvedeny také pro sledování časových průběhů pulzně šířkové modulace a D/A převodníku. Mimo již zmíněné periferie jsou na DPS pro výuku navržené sedmi segmentové displeje řízené převodníkem MAX7219CNG, MEMS senzorem LIS3DH, pamětí EEPROM, OLED displejem a Wi-Fi modulem ESP8266.

Cílem vytvořených úloh je také pozitivní motivace studentů k programování. Vytvořené úlohy byly navrženy s cílem motivace studentů do dalšího osobního rozvoje v oblasti programování. Úlohy mají naučit studenty samostatné práci s katalogovým listem jednotlivých komponent, práci se sběrnicemi, zápisem do datových registrů a zamyšlením se nad rychlostí sběrnic z pohledu elektromagnetického rušení.

1. TEORETICKÁ ČÁST

1.1 STM32

STM32 je rodina 32bitových mikrokontrolerů, postavená na jádře Arm® Cortex®-M od společnosti STMicroelectronics. Tyto procesory jsou uživatelsky velice přívětivé a nabízí velký výpočetní výkon. Jsou dostupné v několika modelových řadách, přičemž každá nabízí upravený výkon na periferie potřebné ve vyžadované finální aplikaci. Jednotlivé typy procesorů jsou uvedeny v tab. 1. [2]

Tab. 1: Typy procesorů STM32 podle parametrů [3]

značka	*Jádro*	Max frek [MHz]	Max FLASH [KB]	MAX SRAM [KB]	cíl
F0	CortexM0	48	256	32	Běžné
F1	CortexM3	72	1024	96	Běžné
F2	CortexM3	120	1024	128	Vysoký výkon
F3	CortexM4	72	512	80	Běžné
F4	CortexM4	180	2048	384	Vysoký výkon
F7	CortexM7	216	2048	512	Vysoký výkon
H7	CortexM7	480	2048	1024	Vysoký výkon
L0	CortexM0+	32	192	20	Nízko odběrové
L1	CortexM3	32	512	80	Nízko odběrové
L4	CortexM4	80	1024	320	Nízko odběrové
L5	CortexM33	110	512	256	Nízko odběrové

STM32 Nucleo-64

V rámci této bakalářské práce byla použita vývojová deska Nucleo-64, osazená procesorem STM32F303RE s instrukční sadou RISC. Jedná se o procesor s jádrem ARM® Cortex®-M4, který využívá vnitřní oscilátor 72 MHz. Pracuje na napětí od 2 V do 3,6 V. Je možné využít až 4 rychlé 12bitové AD převodníky, 7 komparátorů, 4 operační zesilovače, 2 DA převodníky, nízko napěťové RTC, až 5 16bitových časovačů, jeden 32bitový časovač. [2]

Díky velkému množství komunikačních rozhraní lze komunikovat přes sběrnice I^2C, SPI, USART, UART, CAN a USB. Tento procesor dokáže správně pracovat v rozsahu teplot -40 až 85 °C. Tento procesor je díky své stavbě vhodný pro nízkonapěťové aplikace. Tyto low-power módy mají tři varianty a to Sleep, Stop a Standby. 96bitové unikátní ID. Procesor také obsahuje CRC výpočetní jednotku. [2]

Paměti:

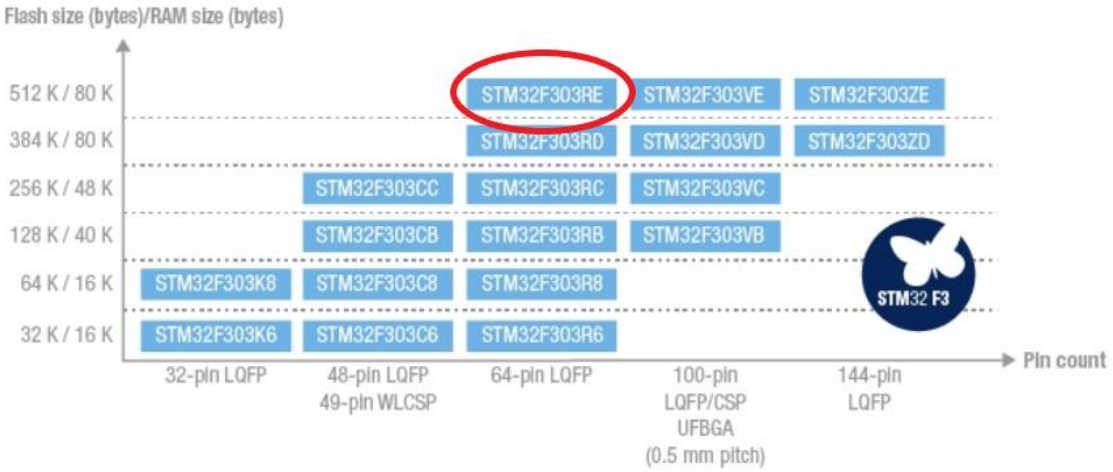
- 512 Kbytes Flash paměť,
- 64 Kbytes SRAM s HW paritní kontrolou na první 32 Kbytes,
- flexibilní paměťový řadič pro statickou paměť s čtyřmi CS. [2]

Řízení hodinového signálu:

- oscilační krystal 4 až 32 MHz,
- 32 kHz oscilátor pro RTC kalibraci,
- vnitřní 8 MHz RC s 16 PLL možnostmi,
- vnitřní 40 kHz oscilátor. [2]

Časovače:

- jeden 32bitový a dva 16bitové časovače s až čtyřmi IC/OC/PWM,
- tři 16bitové šesti kanálové časovače s až 6 PWM kanály,
- jeden 16bitový časovač s dvěma IC/OCs a jedním OCN/PWM,
- dva watchdog časovače,
- dva 16bitové časovače k řízení DA převodníku. [2]



Obr. 1: STM32 F303 v závislosti na velikosti paměti a vývodech [2]

Tento mikrokontroler obsahuje jednotku na ochranu paměti, tato jednotka je schopna zajistit ochranu až 8 oblastem, přičemž tyto oblasti jsou rozděleny do 8 podoblastí. Velikost ochranné oblasti je mezi 32 bytes až celými 4 Gbytes celé adresovatelné paměti. Tato jednotka je užitečná zejména pro aplikace, ve kterých musí dojít k ochraně významného kódu před zneužitím. [4]

Porovnání procesoru STM32 F303 RE s jinými procesory STM32 velikosti paměti a počtu pinů lze vidět na obr. 1. Flash paměť zabudovaná v těchto mikrokontrolerech umožňuje ukládat programy a data až do velikosti 512 Kbytes. Přístup do této paměti je přizpůsoben frekvenci procesoru a to:

- Stav 0 0 až 24 MHz
- Stav 1 24 až 48 MHz
- Stav 2 stav výše

Zdroj napětí je v rozmezí 2 až 3,6 V a je zajištěn externím zdrojem napětí, který je v případě potřeby regulován vnitřním regulátorem napětí. Požadované napětí pro ADC, DAC, komparátory, operační zesilovače a PLL se liší v závislosti na napájení požadované analogové periferie. Tyto údaje je možné sledovat v tab. 2. [4]

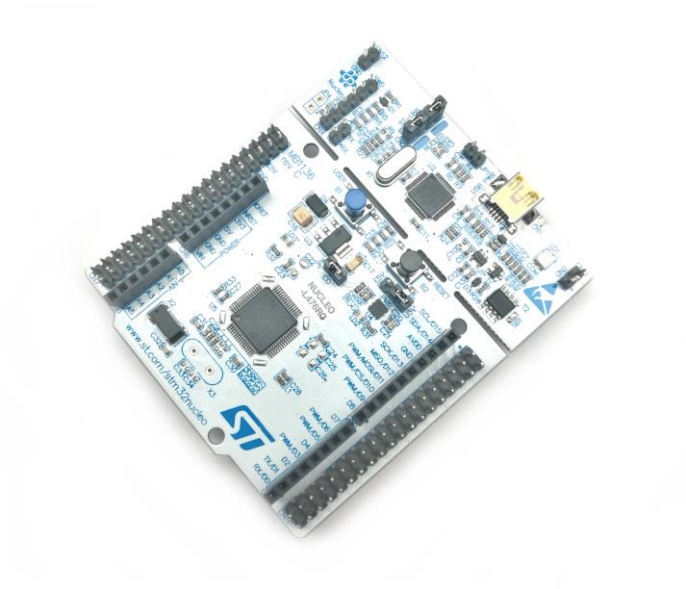
Tab. 2: Maximální a minimální hodnota napětí pro ADC a DAC [3]

Analogová periferie	Minimální napětí	Maximální napětí
ADC/COMP	2,0 V	3,6 V
DAC/OPAMP	2,4 V	3,6 V

STM32F303 podporuje celkem tři nízko napěťové režimy k dosažení nejlepšího požadovaného výsledku mezi nejnižší spotřebou, krátkým nástupním časem a dostupnými budícími zdroji:

- sleep mód: Pouze procesor zastaven. Ostatní periferie pokračují ve vykonávání svých operací a budí procesor v případě přerušení,
- stop mód: Dosahuje nejmenší spotřeby energie. Všechny generátory hodinového signálu fungující na napětí 1,8 V jsou zastaveny. Regulátor napětí může operovat v normálním režimu, nebo může být přepnut do low-power módu,
- standby mód: Tento režim je použit k dosažení nejnižší spotřeby energie. Vnitřní regulátor napětí je vypnutý. Všechny oscilátory jsou také vypnuty. Po spuštění Standby módu je SRAM a registrový obsah ztracen, kromě těch v záložní doméně. [2]

Každý GPIO pin může být nastaven softwarově jako výstup, nebo jako vstup, nebo jako jiná dostupná alternativa. Většina GPIO pinů jsou sdílené analogové nebo digitální funkce. Všechny GPIO jsou schopny vysokého proudu, až na analogové vstupy. [3]



Obr. 2: Nucleo-64 [5]

Součástí toho mikrokontroleru je i přímý přístup k paměti (DMA). Tento přístup je schopný řídit přenosy z paměti do paměti, z periferie do paměti a z paměti do periferie. Podporuje také cyklickou frontu a předchází generování přerušení v momentě, kdy řadič dojde na konec zásobníku. Každý z těchto 12 DMA kanálů je hardwarově připojen k DMA požadavkům se softwarově řízenou spouští pro každý kanál. DMA je použita v hlavních periferiích jako SPI, I^2C, USART, univerzální časovače, DAC a ADC. Procesor STM32 osazený na vývojové DPS Nucleo-64 lze vidět na obr. 2. [4]

Čtyři rychlé A/D převodníky, s možností volby rozlišení mezi 12 a 6 bity jsou součástí všech STM32F303x mikrokontrolerů. A/D převodník má až 40 externích kanálů. Některé z externích kanálů jsou sdíleny mezi ADC1&2 a mezi ADC3&4. A/D převodník nabízí převod mezi single-shot nebo scan módem. Ve scan módu je možné jednotlivé A/D převodníky zahrnout do skupiny která je nepřetržitě sledována. ADC lze použít ve spojení s DMA. Tři analogové watchdog jsou pro ADC dostupné. Samotný procesor STM32 lze vidět na obr. 3. [4]

Dva 12bitové zásobníky DAC kanálů můžou být použity k převedení digitálního signálu na výstupní analogový napěťový signál. Digitální rozhraní podporuje následující funkce:

- dva DAC výstupní kanály,
- 8bitové nebo 10bitové monotické výstupy,
- možnost synchronizované aktualizace,
- dvojitý DAC kanál pro závislý nebo nezávislý převod,
- DMA pro každý kanál,
- vstupní napěťová reference,
- externí spouštění převodu. [3]

Tento procesor také obsahuje sedm ultra rychlých rail-to-rail komparátorů s programovatelným referenčním napětím (vnitřním nebo externím) a volitelnou výstupní polaritu. Referenční napětí může být následující:

- externí I/O,
- výstupní DAC pin. [2]

Všechny komparátory mohou být probuzeny ze STOP módu generováním přerušení pro časovač

Obr. 3: Procesor STM32 [6]

Součástí tohoto procesoru jsou také čtyři operační zesilovače. Pokud je zvolen operační zesilovač, externí ADC kanál je použit ke spuštění výstupního měření. Funkce operačního zesilovače:

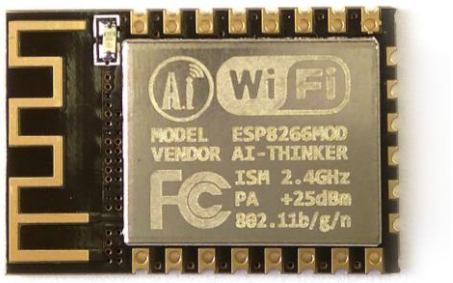
- 8,2 MHz bandwith,
- 0,5 mA výstupní proud,
- rail-to-rail vstup/výstup. [2]

1.2 ESP8266

Jedná se o integrovaný obvod primárně určený k aplikacím s Wi-Fi, kterou tento čip obsahuje. Tento čip se mimo Wi-Fi stará i o výpočetní část, vzhledem k obsahu CPU. Tento čip uvedla na trh společnost Espressif Systems v roce 2013. Jedna z mnoha aplikací toho IO je například jeho osazení na NodeMCU, což je vývojová deska, která umožňuje vytvářet různé aplikace, zejména v oblasti IoT. Tento modul lze vidět na obr. 4. [8]

ESP8266 lze nazývat „System on a chip“, protože pro vývojáře nabízí široké možnosti programování, hardwarové vybavení, a zároveň možnost taktování procesoru. Hlavní výpočetní jednotkou je 32bitový procesor Xtensa LX 106 s instrukční sadou RISC s taktem 80 MHz a jádrem s architekturou ARM. Velikost paměti je pak až 1 MB zkompilovaného kódu. Dokáže efektivně pracovat s elektrickou energií, má kompaktní design a spolehlivý výkon. [8]

Díky samostatným síťovým funkcím Wi-Fi lze s ESP8266 pracovat v samostatných aplikacích tzv. „stand-alone“ zařízení, nebo může být použito jako podřízené zařízení jinému mikrokontroleru. Modul ESP2866 osazený na modulu pro Arduino lze vidět na obr. 5. Integrovaná vysokorychlostní mezipaměť pomáhá zvyšovat výkon systému a optimalizovat systémovou paměť. ESP8266 integruje anténní přepínače, výkonový zesilovač, nízko šumový zesilovač pro přijímač, filtry a moduly pro řízení spotřeby elektrické energie. Díky tomuto návrhu vyžaduje ESP8266 pro svou funkci minimální počet externích obvodů. Hardwarové a softwarové vlastnosti ESP8266 lze sledovat v tab. 3 a tab. 4. [8]



Obr. 4: ESP8266 [7]

Klíčové vlastnosti Wi-Fi:

- 802.11.b/g/n podpora,
- 802.11 podpora 2,4 GHz s přenosem dat až 72.2 Mbps,
- defragmentace,
- 2x virtuální Wi-Fi rozhraní,
- rozmanitost antény. [9]

Tab. 3: Hardwarové vlastnosti ESP8266 [9]

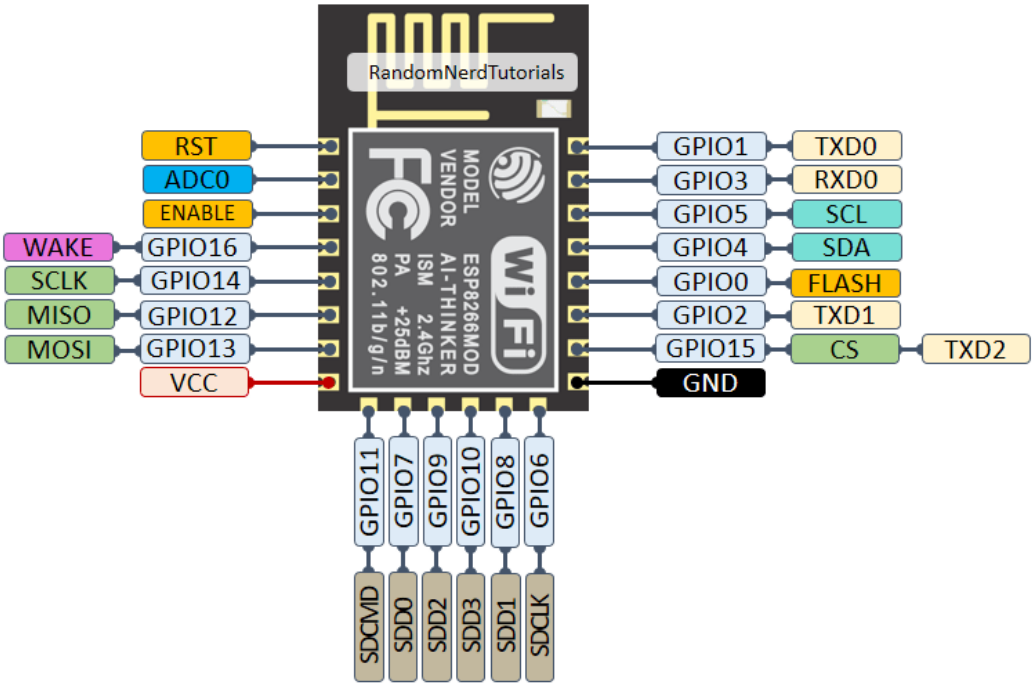
CPU	Tensillica L106 32bitový procesor
Periferní rozhraní	UART/SDIO/SPI/I^2C/I^2S/GPIO/ADC/PWM/LED
Provozní napětí	2,5 V až 3,6 V
Pracovní proud	Průměrná hodnota 80 mA
Provozní teplota	-40 °C až 125 °C
Velikost pouzdra	QFN32-pinů (5 mm x 5 mm)
Externí rozhraní	-

Tab. 4: Softwarové vlastnosti ESP8266 [9]

Wi-Fi mód	Station/SoftAP/SoftAP + Station
Zabezpečení	WPA/WPA2
Šifrování	WEP/TKIP/AES
Firmware aktualizace	UART stažení / OTA
Vývoj softwaru	Podpora Cloud Server vývoj
Síťové protokoly	IPv4, TCP/UHD/HTTP
Uživatelské nastavení	AT instrukční sada, Android/iOS aplikace

Využití v daných aplikacích:

- domácí spotřebiče,
- domácí automatizace,
- chytré zástrčky a světla,
- bezdrátová kontrola v průmyslu,
- monitorování dětí,
- IP kamery,
- síť senzorů,
- nositelná elektronika,
- Wi-Fi zařízení pro rozpoznaní polohy,
- bezpečnostní ID tagy. [9]

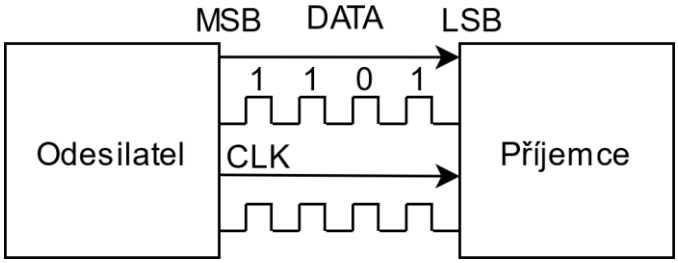


Obr. 5: Arduino modul osazený modulem ESP8266 [10]

1.3 Sériové sběrnice

V sériové komunikaci představují data binární pulzy. V případě, že by došlo ke sledování napětí na sériové sběrnici pomocí osciloskopu, bylo by zřejmé, že napětí 5 V představuje úroveň logické 1 a napětí 0 V představuje úroveň logické 0. Mezi nejvíce používané sběrnice v oblasti mikrokontrolerů patří např. SPI, UART nebo I^2C. V závislosti na vysílacím módu a přenosu dat může mít sériová komunikace více druhů, například polo duplexní, plně duplexní, simplex. Jedná se o techniku tzv. bit po bitu za použití dvou vodičů. [1]

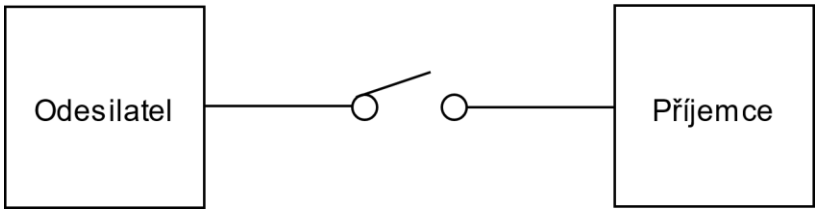
Data jsou odesílána v pořadí, v jakém mají udělenou prioritu, tedy v závislosti na nejvíce významném bitu (MSB) a nejméně významném bitu (LSB). Data jsou odesílána na základě hodinového signálu udávaného hlavní řídící jednotkou obvodu. Komunikace po sériové sběrnici je ideální pro komunikaci na velké vzdálenosti. Příkladem může být komunikace mezi jednotlivými moduly v automobilu, kde se pro rychlou komunikaci na velkou vzdálenost používá CAN sběrnice. Komunikaci lze vidět na obr. 6. [1]



Obr. 6: Základní popis přenášení dat v sériové sběrnici [1]

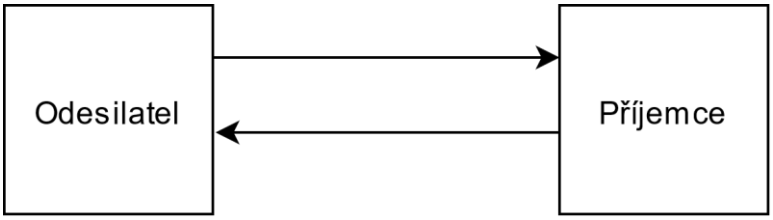
1.4 Vysílací režimy

Polo duplexní metoda - odesilatel i příjemce jsou aktivní, ale ne ve stejný čas. Pokud odesilatel vysílá, příjemce může signál přijmout, ale nemůže ho odesílat a naopak. Jako příklad můžeme uvést internet. Pokud uživatel pošle ze svého zařízení požadavek na server, server musí nejprve zpracovat požadavek a následně odešle uživateli požadovaná data. Komunikaci lze vidět na obr. 7. [1]



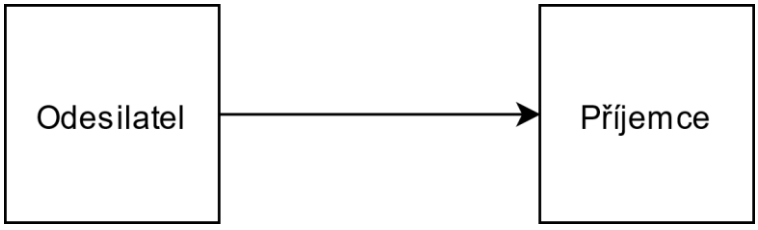
Obr. 7: Polo duplexní metoda [1]

Plně duplexní metoda - nejvíce rozšířený typ sériové komunikace. Odesilatel může přijímat data ve stejný čas v jakém je vysílá a stejně tak druhá strana. Příkladem může být v tomto případě chytrý telefon. Komunikaci lze vidět na obr. 8. [1]



Obr. 8: Plně duplexní metoda [1]

Simplex metoda - jedná se o jednosměrnou komunikační techniku. Umožňuje připojení pouze jednoho uživatele (odesilatel nebo příjemce). Pokud odesilatel data vysílá, příjemce může data pouze přijmout. Příkladem simplexu je rádiové a televizní vysílání. Komunikaci lze vidět na obr. 9. [1]



Obr. 9: Simplex metoda [1]

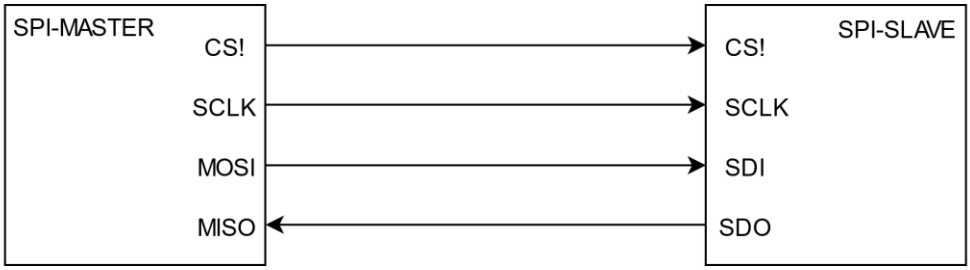
1.5 Druhy sériových sběrnic:

1.5.1 SPI

Sběrnice SPI - byla vyvinuta společností Motorola koncem 20. století. Jedná se o sériovou sběrnici, která využívá plně duplexní metodu přenosu dat. Je to jedna z nejvíce využívaných sběrnic při komunikaci mikrokontroleru s periferiemi jako například senzory, A/D a D/A převodníky, SRAM a mnoho dalších. [11]

Komunikace přes SPI je minimálně 3vodičová, většinou 4vodičová, jedná se o synchronní sběrnici. SPI sběrnice funguje na principu Master a Slave, přičemž Slave zařízení může být více než jedno, ale každý má Chip select (volba Slave zařízení). Master je zařízení, které je zdrojem hodinového signálu. Data posílaná mezi Master a Slave jsou synchronizována podle hodinového signálu Master zařízení. Zařízení využívající pro komunikaci sběrnici SPI umožňují komunikovat na daleko vyšších frekvencích než I^2C sběrnice. [11]

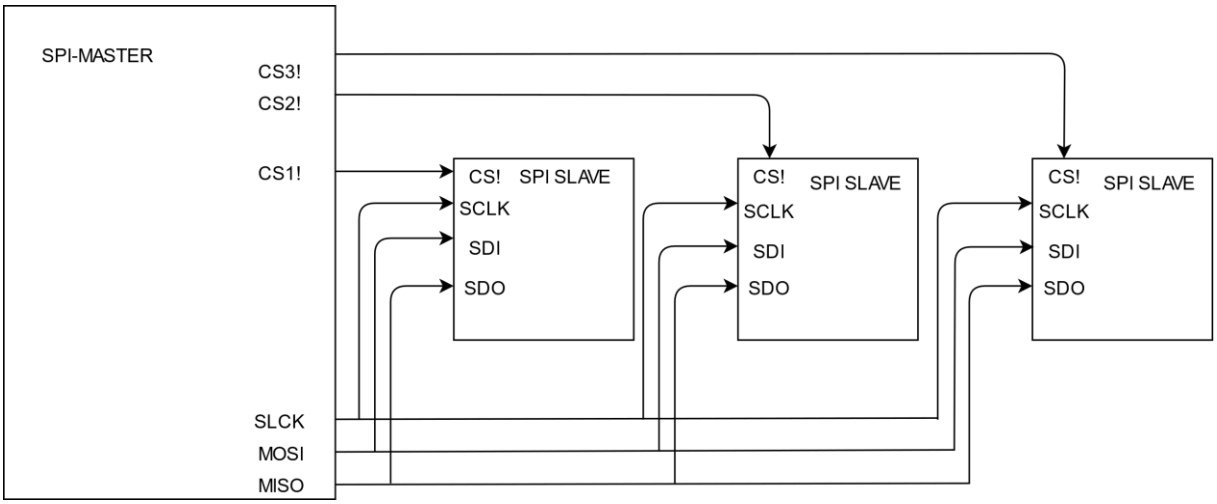
Rozhraní sběrnice SPI umožňuje mít v komunikaci pouze jedno Master zařízení, ale umožňuje mít jednoho, nebo více Slave zařízení. K práci s více Slave zařízeními se využívá pin Chip select, díky kterému je možné určit, kterému zařízení mají data odeslaná z Masteru přijít. MOSI a MISO jsou datové linky. MOSI vysílá data z Master do Slave a MISO vysílá data ze Slave do Master. Princip komunikace lze vidět na obr. 10. [11]



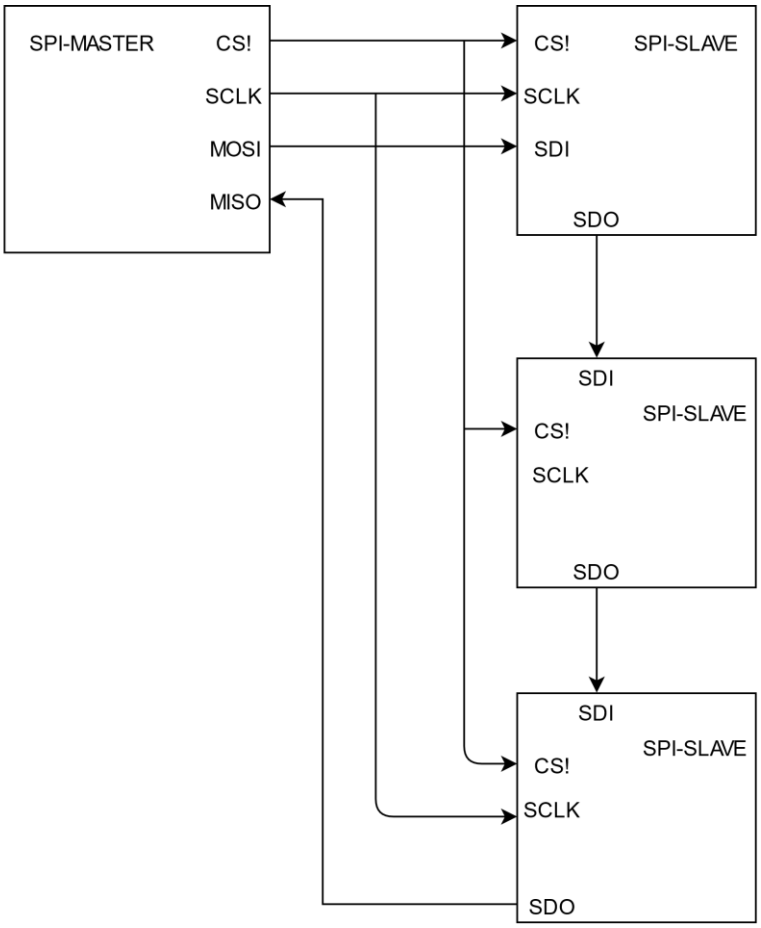
Obr. 10: Základní komunikace mezi Master a Slave [11]

K zahájení komunikace přes SPI je potřeba, aby Master odeslal hodinový signál a za pomoci CS vybral Slave zařízení. Obvykle je CS aktivní nízký signál, proto je nutné, aby Master odeslal logickou 0 a zvolil potřebné Slave zařízení. SPI je plně duplexní komunikace, Master i Slave mohou posílat data ve stejný čas. SPI rozhraní umožňuje uživateli vybrat, jestli chce použít nástupnou, nebo sestupnou hranu hodinového signálu ke vzorkování dat. [11]

Multislave režim, ve kterém je použito více Slave zařízení, může být řízen jedním Master zařízením. Slave může být připojen v klasickém režimu, nebo v tzv. Daisy-chain režimu. Tento režim lze vidět na obr. 12. V Daisy-chain režimu je Slave nastaven tak, že Master využívá pouze jediný Chip select. Signál je totiž v tomto režimu svázán dohromady a data se přesouvají z jednoho Slave zařízení do druhého. V tomto nastavení se všechna Slave zařízení řídí podle stejného SPI hodinového signálu, který přijímají ve stejný okamžik. Multi-Slave režim lze vidět na obr. 11. [11]



Obr. 11: SPI Multi-Slave režim [11]



Obr. 12: SPI Daisy-chain režim [11]

1.5.2 I²C

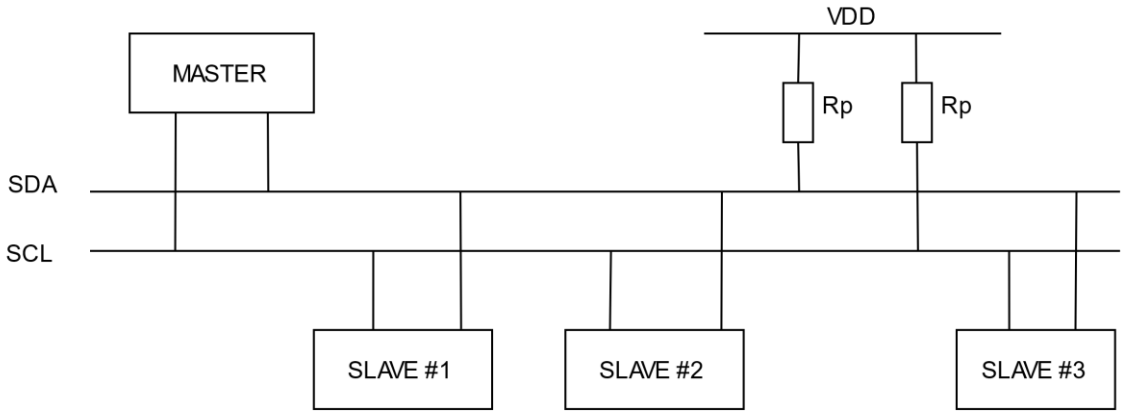
Sběrnice I²C - jedná se o dvou vodičovou sériovou sběrnici, která byla vyvinuta společností Phillips Corporation pro užití ve spotřebitelských produktech. Jedná se o obousměrnou sběrnici. Spojení s ovládaným zařízením probíhá pomocí sériové datové linky SDA a sériovým hodinovým signálem SCL se společnou zemí pro všechny komunikace. [12]

Komunikační protokol sběrnice I²C je postaven na hierarchii Master/Slave, kde Master je definován jako zařízení, které je zdrojem hodinového signálu, adresuje Slave zařízení a zapisuje nebo čte data do nebo z registrů v Slave. Naproti tomu Slave jsou zařízení, která odpovídají, pouze pokud si to Master vyžádá skrz jejich unikátní adresu. Proto je důležité se vyhnout duplicitě adres mezi ostatními Slave. Slave nikdy nevyvolává přenos dat. [12]

I²C sběrnice používá pouze dvě obousměrné linky, sériovou datovou linku SDA a sériovou linku hodinového signálu SCL. Zařízení, připojované ke sběrnici mají výstupy typu otevřený kolektor, díky němuž není nutné, aby na sběrnici bylo připojeno pouze jedno master zařízení a sběrnice tedy může pracovat v multi-master provedení. Jednotlivá zařízení při komunikaci pouze přizemní jednotlivé vodiče sběrnice (SDA

a SCL). Pokud nedochází k přenosu dat, sběrnice I^{2}C je nečinná a linky se pasivně nastaví do HIGH. [12]

Jednotlivé bity reagují na sestupnou hranu hodinového signálu. Standardní rychlost přenosu dat je 100 kbits/s. Rychlý režim má rychlost přenosu dat 400 kbits/s. [12]

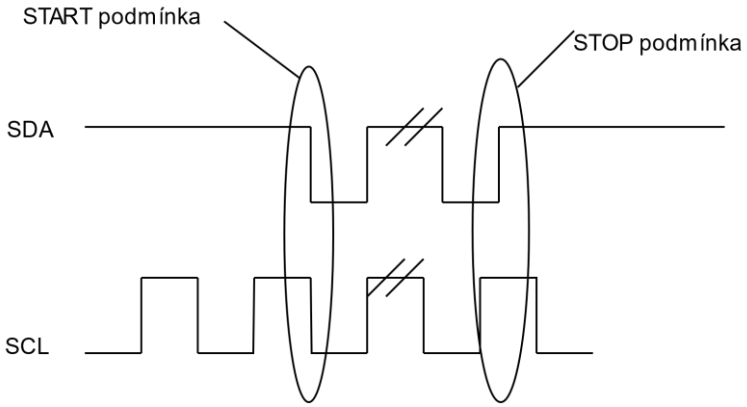


Obr. 13: I^{2}C Připojení více Slave zařízení [8]

I^{2}C sběrnice podporuje připojení vícero zařízení jak Slave, tak Master. Jediným limitem je kapacita sběrnice, která činí 400 pF a šířka adres, která je 128. Toto zapojení lze vidět na obr. 13. [12]

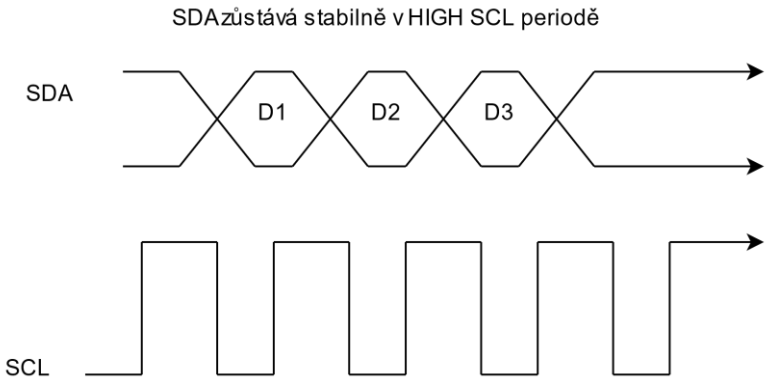
Data, která se přes I^{2}C sběrnici posílají, jsou uspořádána do 8bitů, které zahrnují adresu pro Slave, registrační číslo a data, která mají být přenesena. Data přenášená po sběrnici jsou určena buď ke čtení, nebo k zápisu. Protokol pro čtení a zápis je postaven na sérii pod protokolů, jako jsou start/stop podmínky, opakovaný start bit, adresový bit nebo přenosový data bit. [12]

Start/Stop podmínky - pokaždé, kdy má dojít k přenosu dat inicializovaných Master zařízením, jsou na sběrnici odeslána data, která značí, že začíná komunikace. Tato data probudí všechny Slave zařízení, která jsou na sběrnici připojena a nevykonávají žádnou činnost. Tyto podmínky lze v grafickém provedení vidět na obr. 14. [12]



Obr. 14: Grafické zobrazení start a stop bitu [12]

Datové bity - data jsou po sběrnici přenášena v 8bitovém formátu, který začíná MSB, kde je každý bit synchronizován dle hodinového signálu SCL. Počet přenášených bytů není nějak limitován, ale každý byte musí být následován „Potvrzením“, které je generováno příjemcem dat. Pro bitový přenos dat na SDA lince, musí tato linka zůstat stabilně v HIGH po celou periodu hodinového signálu. Linku SDA i SCL lze vidět na obr. 15. [12]



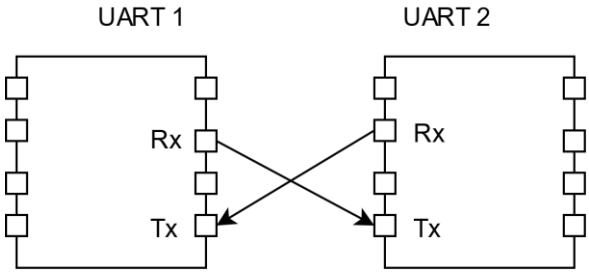
Obr. 15: I²C grafické znázornění SDA linky na SCL lince [12]

1.5.3 UART

Sběrnice UART – na rozdíl od SPI a I²C sběrnic se nejedná o komunikační protokol, ale o fyzický obvod uvnitř mikrokontroleru, nebo samostatný integrovaný obvod. Podobně jako SPI nebo I²C je sériová sběrnice. Hlavním účelem je tedy vysílat a přijímat sériová data. Jedna z největších výhod této sběrnice je využití pouze dvou vodičů pro vysílání a příjem dat mezi zařízeními. [13]

Zařízení využívající pro vzájemnou komunikaci sběrnici UART spolu komunikují napřímo. Vysílající UART, řídící jednotka převádí paralelní data z mikrokontroleru na

sériová, ta odešle druhému zařízení, které je přijme jako sériová a převede je zpátky do paralelní podoby. Data na sběrnici UART se z vysílacího zařízení odesílají z Tx pinu a přijímací zařízení je přijímá na pinu Rx viz obr. 16. [13]



Obr. 16: Komunikace mezi dvě zařízeními pomocí UART [13]

UART vysílá data asynchronně, což znamená, že přenos dat není závislý na hodinovém signálu k synchronizaci výstupních bitů z vysílajícího zařízení, který by vzorkoval bity pro přijímací zařízení. Na místo hodinového signálu je při vysílaní na sběrnici UART přidán do datového balíčku start a stop bit. Tyto bity pak definují začátek a konec příchozích dat, aby bylo pro přijímací stranu možné určit, kdy má začít se čtením bitů. [13]

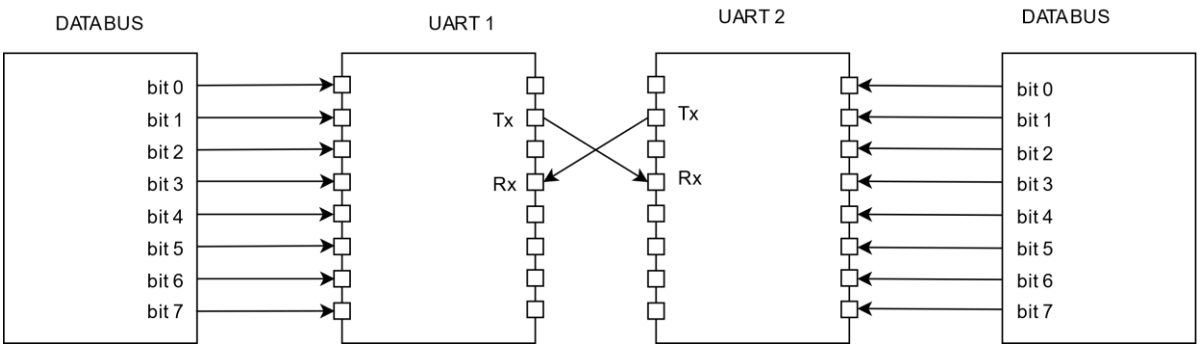
Když přijímací strana pozná, že se jedná o start bit, začne čtení příchozích bitů na specifické frekvenci, která se nazývá „baund rate“. „Baund rate“ slouží k měření rychlosti přenášených dat, vyjádřených v bitech za sekundu (BPS). Obě zařízení na sběrnici UART musí operovat na stejném „baund rate“. „Baund rate“ mezi přijímacím a vysílacím zařízením na sběrnici UART může být rozdílný maximálně o 10 %, aby byla komunikace v pořádku. [13]

Přijímač i vysílač komunikující přes sběrnici UART musí být také nastaveny pro vysílaní a příjem datových balíčků, které mají stejnou strukturu. Základní parametry sběrnice UART lze sledovat v tab. 5 [13]

Tab. 5: Základní parametry sběrnice UART [13]

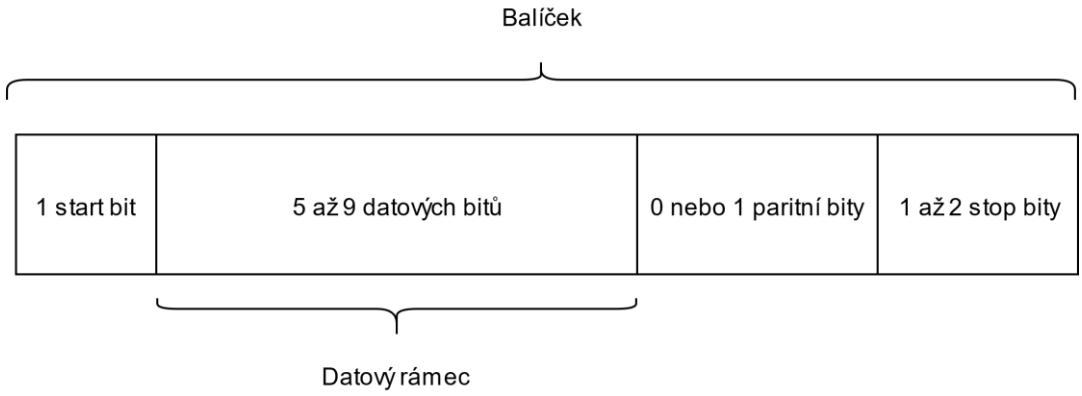
Použitých vodičů	2
Maximální rychlost	Jakákoli rychlost do 115200 baud, obvykle 9600 baud
Synchronní nebo asynchronní?	asynchronní
Sériová nebo paralelní?	sériová
Maximální počet Master	1
Maximální počet Slave	1

Sběrnice UART vysílá a přijímá data z datové sběrnice. Tato datová sběrnice je určena k posílání dat na sběrnici UART z jiného zařízení jako je mikrokontroler, paměť nebo procesor. Data přesunutá z datové sběrnice na vysílající zařízení na sběrnici UART mají paralelní formu. Po přijetí paralelních dat je UART převede na sériová a do datového balíčku přidá start bit, paritní bit a stop bit, čímž vytvoří nový datový balíček, jehož formát lze následně použít pro komunikaci na UART sběrnici. Data jsou následně sériově vysílána bit po bitu z Tx pinu. Přijímající zařízení přečte tento datový balíček bit po bitu na svém Rx pinu. Po přijetí a přečtení jsou data následně převedena zpět do paralelní formy a dojde k odebrání start bitu, paritního bitu a stop bitu. Paralelní data jsou odeslána zpět na datovou sběrnici a následně příjem dat končí. Přenos dat po UART sběrnici lze vidět na obr. 17. [13]



Obr. 17: Přenášení dat z DATA BUS po UART sběrnici [13]

Každý datový balíček, pohybující se na sběrnici UART, je složen z 1 start bitu, 5 až 9 datových bitů (záleží na sběrnici UART), volitelného paritního bitu a 1 nebo 2 stop bity. [13]



Obr. 18: Seřazení jednotlivých bitů v datovém balíčku [13]

Start bit - linka pro vysílání dat je v době, kdy skrz ni neprochází žádná data, nastavena ve vysoko napěťovém stavu. Aby mohlo dojít k přenosu dat, musí vysílací zařízení nastavit vysílací linku z vysokonapěťového stavu do nízkonapěťového po délku jednoho hodinového signálu. V momentě, kdy přijímací zařízení rozpozná, že došlo k nízkonapěťovému stavu na lince, začíná číst bity v datovém rámci na frekvenci, kterou udává „baud rate“. [13]

Datový rámec - obsahuje aktuálně přenášená data, tato data mohou být 5 až 8 bitů dlouhá, pokud je použit paritní bit. Pokud paritní bit použit není, data mohou být až 9 bitů dlouhá. Ve většině případů je nejméně významný bit posílán první. Tvar datového rámce lze vidět na obr. 18. [13]

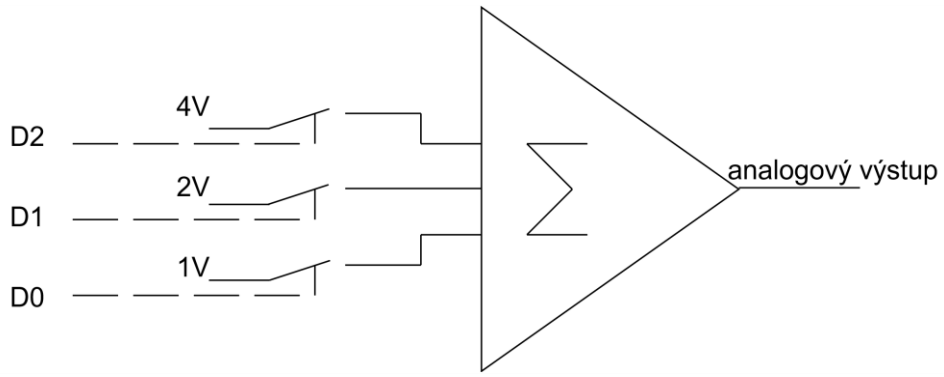
Parita - popisuje sudost nebo lichost čísla. Paritní bit slouží přijímači k určení toho, zda během přenosu došlo ke změně některých dat. Bity v přenosu mohou být změněny vlivem elektromagnetického záření, neodpovídajícím baund ratem, nebo přenosem dat na dlouhou vzdálenost. Po přijetí dojde k přečtení datového rámce. Počítá se počet bitů s hodnotou 1 a kontroluje se, jestli je součet sudý nebo lichý. Pokud je paritní bit 0, první bit v datovém rámci by měl být sudé číslo. Pokud je paritní bit 1, první bit datového rámce by měl být liché číslo. Pokud se paritní bit shoduje s daty, potom je UART schopný určit, že se komunikace obešla bez chyb. Pokud je však paritní bit 0 a součet je lichý, nebo je paritní bit 1 a součet je sudý, UART určí, že během přenosu došlo k chybě a datový rámec se změnil. [13]

Stop bit - oznamuje konec datového balíčku, odesílající zařízení změní napětí linky, na které dochází k přenosu z nízkonapěťové na vysokonapěťovou pro délku alespoň dvou bitů. [13]

1.6 Převod digitálního signálu na analogový

D/A převodník (DAC) je čip nebo obvod, který převádí digitální čísla na analogové hodnoty napětí nebo proudu. D/A převodníky jsou používány pro řízení zařízení, která vyžadují řízení v širokém rozsahu napětí nebo proudu, jako jsou například elektro akustické převodníky (reproduktory), vybrané typy motorů s proměnnými otáčkami a spousta dalších aplikací, u kterých je požadován analogový výstupní signál. Nejvíce se však využívá k tvorbě analogových křivek z digitálního signálu, například u CD přehrávače. [14]

D/A převodníky – si můžeme představit jako obvod, který reguluje úroveň napětí na základě digitálního signálu. Jakékoli napětí může být spuštěno nebo vypnuto elektronickým přepínačem, který je řízen digitálním vstupem. Na obvodu na níže uvedeném obrázku lze vidět 3bitový D/A převodník, složený ze zesilovače napájeného třemi různými napětími, jehož grafickou podobu lze vidět na obr. 19. Záleží na tom, který z přívodů napětí je vypnutý. Výstupní napětí se pohybuje v rozmezí 0 až 7 V. Za použití digitálního signálu, který řídí přepínače, které do obvodu přivádí napětí, je možné postavit obvod, jehož výstupní napětí je úměrné hodnotě digitálního signálu. [15]



Obr. 19: 3-bitový D/A převodník [11]

Nejdůležitější vlastností D/A převodníků je rozlišení a nastavovací doba. V některých aplikacích také ostatní vlastnosti jako doba přeběhu, linearita, monotonicita. Rozlišení (délka výstupního kroku) je dáno rozsahem výstupního napětí, které je rozděleno podle různých úrovní výstupního napětí. N-bit DAC může mít výstupní hodnotu 2^N různých úrovní 2^N-1 kroků. N je v rozsahu 6 až 20 v závislosti na kvalitě převodníku. [15]

Přesnost je maximální rozdíl mezi výstupní hodnotou a teoretickou hodnotou. Typicky je tato hodnota přibližně stejná jako rozlišení.

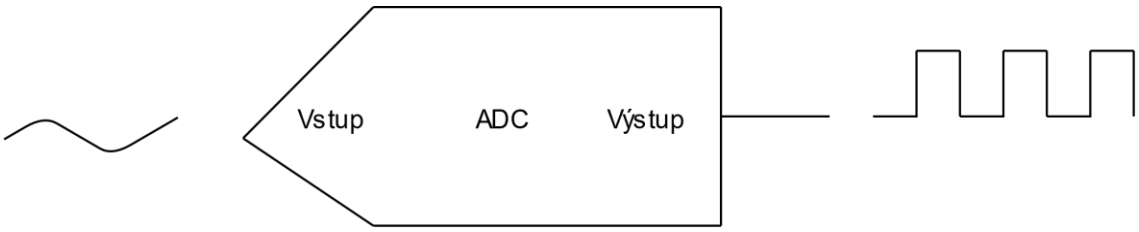
Nastavovací doba je čas, za jaký se výstupní hodnota dostane na 99 % požadované výstupní hodnoty. Hodnoty mohou být v rozsahu od stovek milisekund, pro zařízení s vysokou přesností pak v rozsahu až do nanosekund. Takovéto hodnoty se nejčastěji vyskytují u vysokofrekvenčních generátorů. [14]

Linearita je maximální rozdíl mezi výstupní a lineární interpolací mezi minimální a maximální hodnotou, většinou $1/2^N$. [15]

Ačkoli převodníky nenabízejí vysokou přesnost, jsou často dostatečné pro velké množství aplikací. Proto je vždy potřeba určit, zda je požadovaný výsledek s použitým převodníkem možné dosáhnout. [15]

1.7 Převodníky analogového signálu na digitální

V různých aplikacích je často potřeba měřit určité množství nebo úroveň, jako například teplotu, tlak, sílu apod. Senzory, často nazývané také převodníky, jsou používány k převodu těchto fyzikálních veličin na elektrický signál napětí nebo proud. Tento elektrický signál musí být následně převeden na binární čísla, se kterými může následně řídící jednotka pracovat. A/D převodník je schopný tuto funkci vykonávat. Na obr. 20 lze vidět základní princip fungování A/D převodníku. [15]



Obr. 20: Základní princip ADC

Komparátory - porovnávají dva analogové vstupy (U_{IN} a U_{REF}) a výstupní logický signál, který je velký, pokud je U_{IN} větší než U_{REF} nebo nižší v opačném případě. Komparátory jsou často dostupné jako integrované obvody, většinou v počtu dvou nebo čtyř jednotek v jednom IO. [14]

Rychlejší metoda je použití binárního vyhledávání. Řídící obvod nejprve otestuje, zda je hodnota analogové vstupní hodnoty větší nebo nižší než polovina D/A výstupního rozsahu. Tímto se opraví hodnota nejvýznamnějšího bitu. Potom řídící obvod testuje, jestli je hodnota vyšší nebo nižší než poloviční bod zbylého rozsahu hodnot. Takto se nastaví další nejvýznamnější bit. Tento proces je opakován tak dlouho, dokud nejsou nastaveny hodnoty všech bitů. Délka této metody je N krát D/A nastavovací doby. [15]

Druhy A/D převodníků:

- postupná aproximace ADC – tento převaděč porovnává vstupní signál s výstupním vnitřním DAC převodníkem v každém kroku. Jedná se

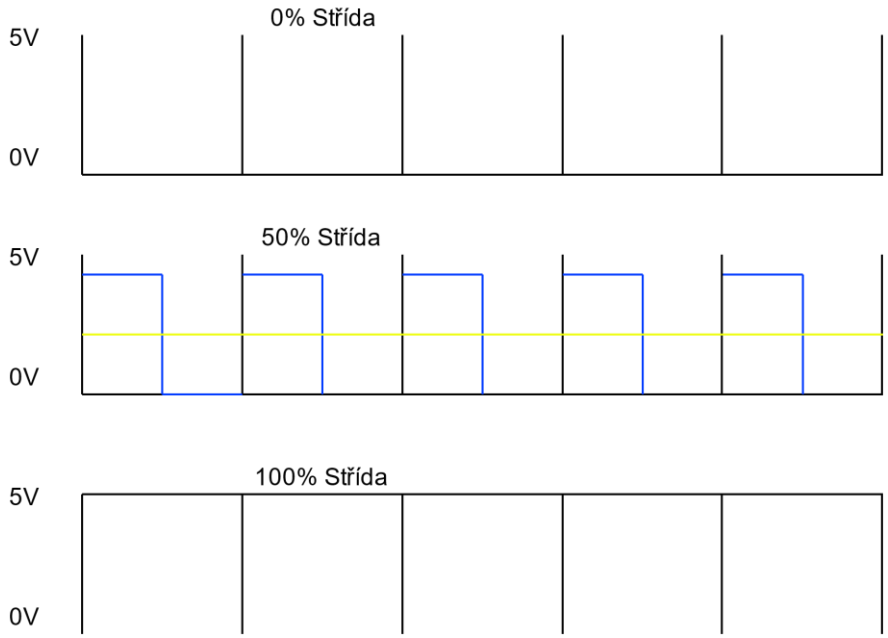
nejdražší A/D převodník,
- Dual Slope ADC – Dosahuje vysoké přesnosti, ale je velice pomalý,
- Delta-Sigma ADC – Má vysoké rozlišení, ale je pomalý kvůli vysokému odběru vzorků,
- Flash ADC – nejrychlejší převodník, ale je velice finančně nákladný. [15]

Využití A/D převodníků:
- v počítačích pro převod analogového signálu na digitální,
- telefony,
- mikrokontrolery,
- reproduktory,
- vědecké nástroje. [15]

1.8 Pulzně šířková modulace

Pulzně šířková modulace (PWM) je způsob, jakým může být řízeno analogové zařízení za pomoci digitálních výstupů. Dalším způsobem, jak lze řídit analogová zařízení je modulovaný digitální signál pocházející z mikrokontroleru. Jedná se o jeden z hlavních způsobů, jakým lze za pomoci mikrokontroleru řídit analogová zařízení jako třeba motory s proměnnými otáčkami, jas světel nebo reproduktory. V případě PWM se nejedná o reálný analogový výstup, PWM se spíše vydává za analogový výstup tím, že například generuje krátké pulzní signály regulovaného napětí. [16]

Signál PWM má tvar čtverce a je vytvářeno několik těchto čtvercových pulzů za sebou. V kterémkoli okamžiku probíhajícího signálu bude mít pulz vysokou nebo nízkou hodnotu. V případě, že napětí PWM signálu bude v rozsahu 0-5 V, vysoká hodnota bude v tomto případě 5 V, a nízká hodnota bude 0 V. Doba, po kterou signál setrvává ve vysoké hodnotě je nazývána „on time“ a doba, po kterou signál setrvává v nízké hodnotě, nazýváme „off time“. Při používání PWM jsou důležité dva parametry, střída PWM a frekvence PWM. [16]



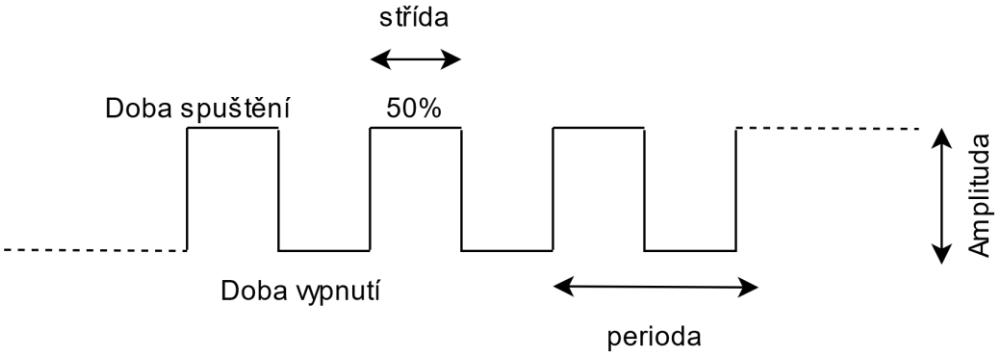
Obr. 21: Grafické znázornění střídy PWM [16]

Střída - PWM signál zůstává po určitou dobu zapnutý a po zbytek periody vypnutý. PWM signál je velice výhodný v tom, že je možné jej nastavit tak, jak dlouho by měl zůstat zapnutý. Tato funkce je řízena pracovním cyklem PWM signálu. Na obr. 21 lze vidět grafické znázornění střídy. [16]

Doba, po kterou zůstává PWM signál ve vysokých hodnotách (on time), se nazývá „duty cycle“ v překladu střída. Pokud signál trvale setrvává ve ON režimu, znamená to, že je střída 100 %, pokud je naopak trvale v OFF režimu, pak je střída 0 %. Střídu lze určit na základě tohoto vzorce. [16]

$$pracovní\ cyklus = \frac{turn\ ON\ time}{turn\ ON\ time + turn\ OFF\ time} \tag{1.1}$$

Na obr. 22 lze vidět PWM signál s 50 % střídou. Jak je možné vidět, po délku jedné periody zůstává signál v ON režimu po dobu 50 % periody. Řízením pracovního cyklu z 0 do 100 % je možné kontrolovat šířku on time pulzu PWM signálu. [17]



Obr. 22: PWM signál se střídou 50 % [17]

Frekvence PWM signálu určuje, jak rychle dojde k opakování jedné periody. Jedna perioda se skládá z času, po který je signál v on time a off time. Obvyklý PWM signál generovaný mikrokontrolerem se pohybuje okolo 500 Hz, takto vysoká frekvence bývá použita ve vysoko rychlostních spínacích zařízeních, jako jsou střídače nebo převodníky. Naopak ne všechny aplikace vyžadují vysoké frekvence. Například k řízení servo motorů je potřeba PWM signál o frekvenci 50 Hz. [17]

1.9 EEPROM

Jedná se o nevolatilní ROM paměť, která je využívána k ukládání malého množství dat v počítačích, nebo jiných elektronických zařízeních. U EEPROM lze za použití elektrického napětí smazat celou paměť, jednotlivé byty smazat nelze. [18]

Tento druh paměti byl vyvinut panem Georgem Perlegosem v roce 1978 ve společnosti Intel. Tato paměť byla vyvinuta jako náhrada PROM a EPROM pamětí. Výhodou těchto pamětí je, že je možné z ní číst data, ale také je mazat a zapisovat do nich nová. K vymazání dat je však potřeba relativně vysoké napětí a jedny z prvních EEPROM pamětí potřebovaly externí zdroj vysokého napětí. Vzhledem k potřebě využívat externí zdroj napětí u mnoha aplikací využívajících paměť EEPROM byl do těchto paměťových čipů začleněn zdroj tohoto napětí, čímž byly výrazně sníženy náklady na celý obvod. [18]

EEPROM paměti mají daleko pomalejší zápis a čtení než RAM paměti. Z tohoto důvodu je vhodné do paměti EEPROM ukládat data, která nejsou nezbytnou součástí pro fungování celého systému. Typicky se do této paměti ukládají data, která mohou být stažena při startu systému. [18]

EEPROM paměť využívá stejné základní principy, které jsou využívány u EPROM pamětí. Paměťové buňky se často skládají ze dvou FET tranzistorů. Jeden z těchto tranzistorů slouží jako úložný prostor a obsahuje tzv. plovoucí bránu. Elektrony jsou zachyceny v této bráně, tyto elektrony následně tvoří data uložená v paměti. [18]

Sériová EEPROM - práce s touto pamětí je složitější, a to z toho důvodu, že je do ní možné zapisovat data pouze sériově, tedy za sebou. Kvůli tomu je zápis a čtení dat ze sériové paměti pomalejší než u paralelní EEPROM. [18]

Existuje několik standardních typů sběrnic, nejběžnějšími jsou: SPI, I^2C, Microwire, UNI/O a 1-Wire. Tyto sběrnice vyžadují 1 až 4 kontrolní signály pro danou operaci. Protokol pro sériovou EEPROM se skládá ze tří částí: OP-code, adresová část a datová část. OP-code je obvykle prvních 8bitů, po kterých následuje 8 až 24 adresních bitů, které závisí na daném zařízení, až potom následuje čtení nebo zápis dat. [18]

Při použití těchto sběrnic lze tuto polovodičovou paměť umístit do pouzdra s osmi vývody. Hlavní výhodou pouzder pro tyto paměti je právě to, že je lze realizovat velmi malé. [18]

Paralelní EEPROM – tyto paměti obvykle mívají paralelní sběrnici o šířce 8 bitů. Díky paralelní sběrnici je možné využít celou paměť na spoustu menších procesorových aplikací. [18]

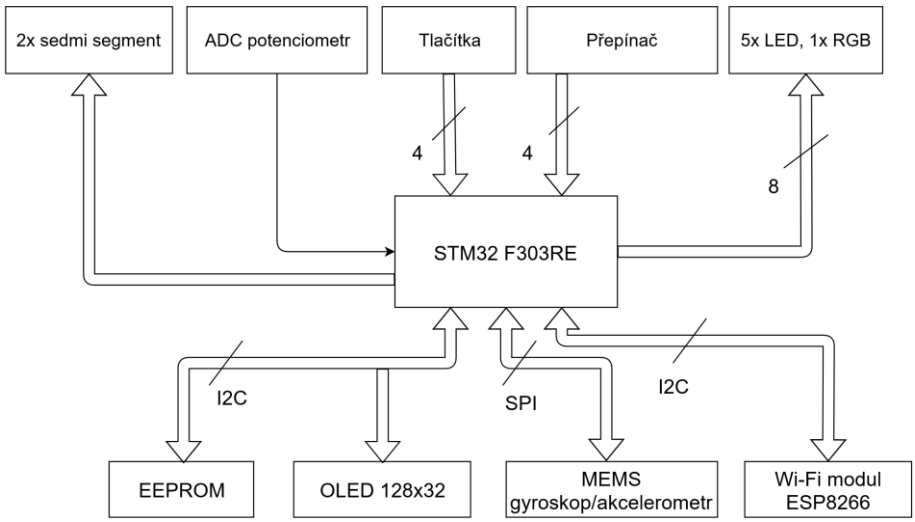
V porovnání se sériovou EEPROM jsou operace na paralelní EEPROM rychlejší a také jednodušší. Nevýhodu je pak velikost v závislosti na vyšším počtu pinů. V důsledku nákladů a pohodlnosti klesá popularita sériových EEPROM pamětí. Dnešní Flash paměti nabízejí lepší výkon za stejnou cenu, kde sériová EEPROM nabízí jako výhodu svou malou velikost. [18]

2. PRAKTICKÁ ČÁST

Tato práce začíná návrhem blokového schématu, které vymezuje zadání této práce, dále pak návrhem elektrického zapojení, návrhem desky plošného spoje a návrhem a zpracováním laboratorních úloh. Blokové schéma je složeno z hlavní řídící jednotky a periferií, které byly zvoleny na základě laboratorních úloh, které budou studenti v rámci výuky zpracovávat, blokové schéma lze vidět na obr. 23. Jako hlavní řídící jednotka bylo zvoleno STM32 F303RE osazené na vývojové desce Nucleo-64. Tento mikrokontroler byl zvolen na základě vyhovujících požadavků týkajících se výpočetního výkonu, 32bitového systému s instrukční sadou RISC na jádře s architekturou typu ARM, dostatečného počtu sériových sběrnic jako I^2C, SPI a UART, vyhovujícím počtem analogových a digitálních vstupů/výstupů, ADC, DAC převodníků a 40 kHz vnitřním oscilátoru.

Z pohledu periferií byly pak v návrhu zahrnuty následující periferie. Dva sedmi segmentové displeje, ADC potenciometr, sada čtyř tlačítek a čtyř přepínačů, pět LED a jedna RGB dioda, paměť typu EEPROM, OLED displej, MEMS akcelerometr a gyroskop a jako periferie, a zároveň také možná řídící jednotka, Wi-Fi modul ESP8266. Tyto periferie byly voleny s ohledem na výuku, tzn. že periferie jsou na desce navrženy tak, aby se studenti v rámci výuky naučili pracovat se základními periferiemi, jako jsou například LED, až po složitější jako například MEMS senzory nebo OLED displej.

Ke všem periferiím obvodu lze přistupovat pomocí hlavní řídící jednotky, tedy STM32, k vybraným periferiím pak pomocí ESP8266. Pro komunikaci s pamětí EEPROM a OLED displejem byla zvolena sériová sběrnice I^2C. Tato sběrnice byla zvolena také pro komunikaci s modulem ESP8266. Pro komunikaci s MEMS byla zvolena sériová sběrnice SPI. Pro výukové účely byly na DPS vyvedeny hřebínky pro sběrnici UART.



Obr. 23: Blokové schéma výukové DPS

2.1 Návrh Schématu

V návrhu schématu byly k hlavní řídící jednotce postupně připojovány jednotlivé periferie. Některé byly připojeny přes sériové sběrnice jako například I^2C nebo SPI, jiné byly připojovány přímo paralelně na vývody mikrokontroleru, jako například LED diody, tlačítka nebo přepínače. Při připojování jednotlivých komponent byla řešena problematika připojování jednotlivých periferií kvůli HW podpoře ze strany mikrokontroleru, jelikož ne každý vývod podporuje připojení každé periferie.

Tlačítka byla k mikrokontroleru připojena na volné GPIO piny, u nichž nezávisí na podpoře z hlediska HW mikrokontroleru. K tlačítkům byly připojeny pull-up rezistory o velikosti 10 kΩ, tyto rezistory byly zapojeny kvůli tomu, aby po sepnutí tlačítka dodaly opačnou polaritu, vzhledem k tomu, že tlačítka jsou připojena k zemi, jsou tedy rezistory připojeny k napájecímu napětí. Tlačítka byla také opatřena sériovými rezistory chránící mikrokontroler před případnými proudovými nárazy nebo vzniklými napěťovými špičkami. Paralelně k tlačítkům byly připojeny kondenzátory o velikosti 100 nF. Tyto kondenzátory slouží k zabránění zákmitům při stisku tlačítka. Dva sedmi segmentové displeje, obsahující každý celkem 4 samostatné sedmi segmenty, jsou řízeny pomocí převodníku MAX7219CNG, tento převodník je výhodný proto, že není potřeba připojovat pro každý segment zvlášť rezistor, ale samotný převodník obsahuje proudový zdroj, který následně dodává proud každému segmentu zvlášť. Sedmi segmentové zobrazovače jsou jím automaticky multiplexovány, a není potřeba žádný zásah z hlediska uživatele, pouze zápis kýžené hodnoty k zobrazení, respektive jeho dat. Tento řadič je ovládán pomocí sběrnice SPI. Díky tomu je možné připojit velké množství segmentů jen za pomoci tří vodičů. Řadič má identifikační vývod, tzv. „ChipSelect“ připojen na vývod GPIO output PB6 a nemá připojený vývod pro čtení dat, jelikož ze sedmi segmentového displeje není, co číst. Zapojení samotného řadiče je dle vzorového zapojení v katalogovém listu.

RGB LED dioda byla připojena na vývody časovače TIM3, aby bylo možné regulovat jas jednotlivých barev a vytvářet i další barvy jejich mixováním. Před RGB LED diodu byly zařazeny patřičné sériové rezistory k omezení proudu, kde každá barva RGB LED diody je definována jiným napěťovým úbytkem a z toho důvodu bylo nutné navrhnout jiný rezistor pro každou barvu. Červené LED diody byly připojovány přímo na vývody mikrokontroleru, jelikož jejich proud propustným směrem nepřekračuje maximální povolený proud mikrokontroleru a je tedy možné, je tímto způsobem připojit. Před LED diody byly připojeny sériové rezistory o velikosti 4,7 kΩ i přes odlišnou vypočtenou hodnotu. Bylo tak stanoveno z praktického hlediska vysoké svítivosti LED diod, aby příliš neoslňovaly.

MEMS senzor LIS3DH byl zapojen dle typického zapojení podle katalogového listu. Tato periferie je řízena po sběrnici SPI. Mezi napájecí a zemnící piny integrovaného obvodu byly připojeny paralelně 3 blokovací kondenzátory, dva menší 100 nF a jeden 10 uF pro zabránění poklesu napětí, což by mohlo způsobit reset obvodu a ochranu proti větších proudových špičkách. Tento integrovaný obvod je schopný pracovat i v případě poklesu napětí, což v rámci výuky při konstantním napájecím napětí nehrozí. Tento obvod slouží k měření vnějších analogových veličin, jako například zrychlení nebo teploty. S naměřenými daty je možné pracovat tak, že se mohou napřímo zobrazit na OLED displeji, dále se mohou ukládat do paměti samotného mikrokontroleru, případně se mohou po přenesení přes I^2C a SPI sběrnici ukládat do EEPROM paměti. Data lze

také ukládat přímo do bufferu samotného integrovaného obvodu, a to do zásobníku nastavených jako LIFO nebo FIFO, případně je zde možnost nastavit buffer jako cyklickou frontu, přičemž každá osa „x, y a z“ má svůj vlastní zásobník na 32 bitů. Vzorkovací frekvence je nastavitelná od 1 Hz až po 5,3 kHz. Vzorkovací frekvence je přímo závislá na odběru proudu, který však díky konstantnímu napájení může být zanedbán. Je možné měřit přetížení až ± 16 g. ADC ani vstupy pro přerušení nebyly z důvodu využitelnosti připojeny.

Přes jednu společnou I^2C sběrnici pak komunikují s mikrokontrolerem OLED displej a paměť EEPROM. Obě tyto periferie byly zapojeny dle katalogového listu. Jako pull-up rezistory byly na vodiče SDA s SCL připojeny rezistory o velikosti 4,7 kΩ. Tyto odpory byly zvoleny na základě katalogového listu, který uvádí, že při těchto velikostech je rychlost sběrnice okolo 400 kHz, toto vychází z maximální povolené kapacity pro správnou funkci tohoto typu sběrnice. Jako paměť byla zvolena sériová EEPROM 24C02C, která umožňuje více než 1 milion zápisů, což je velice výhodné z pohledu délky využitelnosti v rámci výuky. Navíc také obsahuje hardwarovou ochranu. Adresové piny byly uzemněny z toho důvodu, že v obvodu se již nenachází žádná jiná paměť, a tudíž není nutné ji přímo adresovat. Stejně tak byl uzemněn pin pro ochranu zápisu dat, protože se v této aplikaci neočekává žádná manipulace, kvůli které by bylo nezbytné tento pin využívat. Do paměti je možné zapsat až 2048 bitů dat. Displej využívá technologii OLED a dokáže zobrazovat v rozlišení 128x32. Toto rozlišení je z pohledu výuky naprosto dostačující. Jedná se o grafický zobrazovač s uhlopříčkou obrazovky 0,91 palce s dostatečnou svítivostí až 120 cd/m^2. Lze na něm zobrazit vše potřebné od grafů, skrz data z MEMS senzorů až po binární kalkulačku. Napájecí napětí tohoto displeje se pohybuje v rozsahu 2,8 V až 3,6 V. Mezi VCC a VSS musí dojít k zapojení stabilizačního kondenzátoru v případě využití převodníku. Mezi referenčním pinem IREF a zemnícím pinem VSS muselo dojít k připojení rezistoru, což omezí procházející proud, aby nepřekročil hodnotu 12,5 uA Data zobrazená na displeji jsou sériově odesílaná z hlavní řídící jednotky. Pro tuto aplikaci byl zvolen displej s komunikací po I^2C, existují však obdobné modely komunikující přes sběrnici SPI. Kvůli výuce byly na druhou využívanou I^2C sběrnici přidány volitelné pull-up rezistory jak pro datový signál, tak pro hodinový signál. Měnit jejich hodnotu lze pomocí zkratovacích propojek. Při změně je možné na osciloskopu sledovat změnu digitálního signálu, který má demonstrovat vliv pull-up rezistorů na rychlost a celkovou kapacitu sběrnice. Dále pak na také tvar obdélníkového signálu.

V rámci návrhu PWM byl použit operační zesilovač MCP6271, který je zapojen jako napěťový sledovač a slouží k oddělení zátěže od zbytku obvodu. Tento OZ byl vybrán, především díky jeho vlastnostem, má šířku pásma 2 MHz, a to při zesílení 110 dB, což je v rámci této aplikace více než dostačující. Rychlost přeběhu je 900 mV/us a klidový proud je 240 uA. Vstupní napěťová nesymetrie se pohybuje okolo 3 mV a jedná se o OZ tzv. „rail-to-rail“ což znamená, že výstupní hodnota napětí dosahuje téměř hodnotu napětí vstupního. FET tranzistor společně s rezistorem a kondenzátorem tvoří měnič napětí, díky němuž je možné převádět digitální signál, vycházející z mikrokontroleru na analogový signál, který je následně možné sledovat pomocí osciloskopu. Hodnoty rezistoru a kondenzátoru byly voleny tak, aby vhodně plnily podmínku integrace.

V návrhu schématu byla zohledněna potřeba ukázat studentům chování signálu na jednotlivých sběrnicích, za ADC nebo DAC výstupy nebo po galvanickém oddělení.

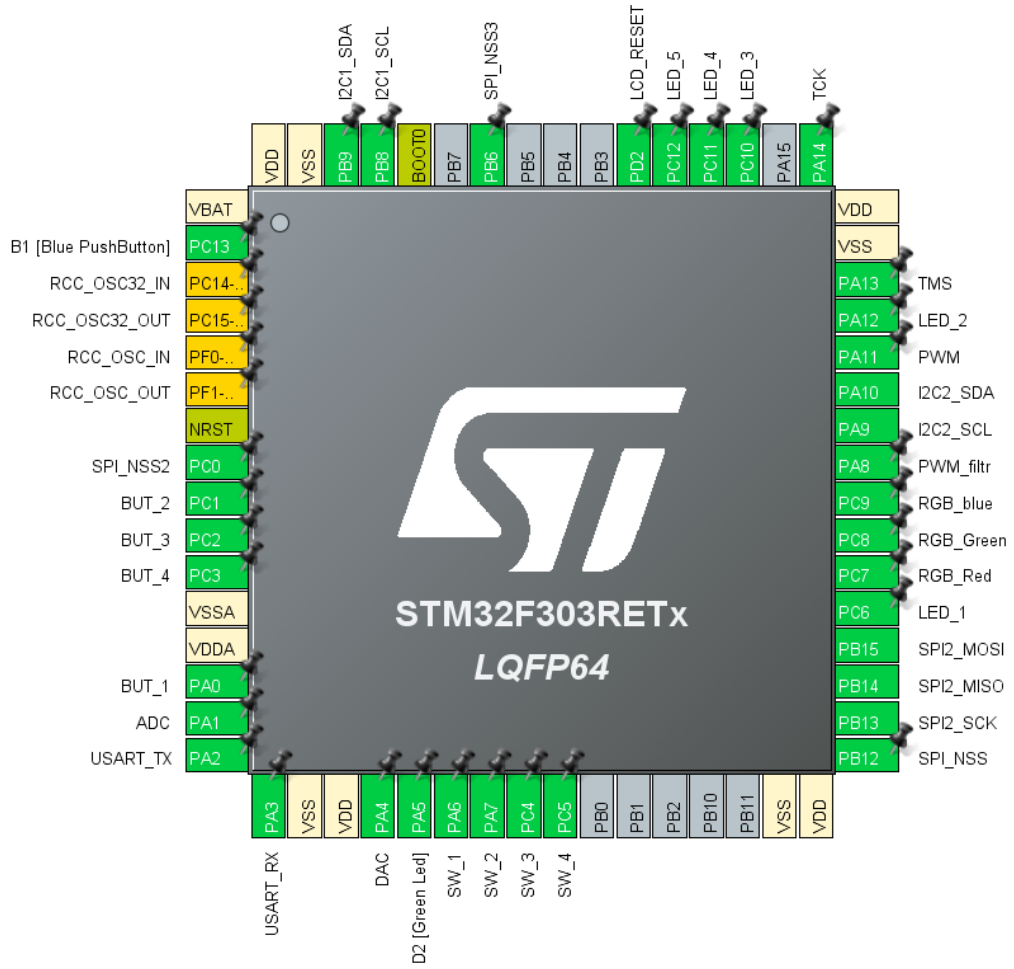
Z tohoto důvodu byly do schématu přidány hřebínky, na které je možné připojit sledovací sondu osciloskopu a pozorovat chování signálu během přenášení datového a hodinového signálu. Chování signálu se totiž může lišit vlivem okolních jevů jako externí rušení datových cest. Okolní teplota může mít vliv na rychlost procesoru. Na osciloskopu je mimo jiné možné sledovat také rychlost samotné sběrnice.

Pro sledování signálu po galvanickém oddělení byly do obvodu zakomponovány dva prvky, digitální izolátor a optočlen. Oba tyto oddělovací prvky komunikují s mikrokontrolerem přes sériovou sběrnici I^2C. Jako optočlen byl zvolen integrovaný obvod ILD205. Optočlen je spínán tranzistorem a základní princip jeho funkce je takový, že v momentě, kdy LED diodou uvnitř pouzdra protéká dostatečně velký proud, který tuto LED diodu rozsvítí, pak tato dioda emitací svého světla sepne fototranzistor umístěný v pouzdře naproti ní, tím dojde k předání informace bez elektrického spojení těchto obvodů a proto lze optočlen využít jako součástku pro galvanicky oddělené aplikace. Optočlen ILD205 byl zvolen z toho důvodu, že jeho parametry ideálně odpovídají požadované aplikaci. Tento optočlen má celkem 2 kanály, jeden využitelný pro data a druhý pro hodinový signál, izolační napětí 4 kV, což je více než dostačující. Před bázi bipolárního tranzistoru byl umístěn ochranný rezistor o velikosti 1 kΩ. Za optočlen byl umístěn hřebínek pro sledování signálu po galvanickém oddělení. Tento signál lze sledovat pomocí osciloskopu. Jako digitální izolátor byl zvolen integrovaný obvod ADUM1250ARZ. Tento digitální izolátor má rychlost přenosu 1 Mbp, napájecí napětí od 3 do 5,5 V DC a izolační napětí 2,5 kV. Digitální izolátor je s mikrokontrolerem propojen přes stejnou I^2C sběrnici. Stejně jako optočlen se využívá v aplikacích, kde je vyžadováno galvanické oddělení a stejně tak i u tohoto izolátoru lze na výstupních hřebíncích sledovat tvar a vlastnosti digitálního signálu po galvanickém oddělení. Základní princip digitálního izolátoru je podobný optočlenu, na primární straně obvodu se digitální signál rozkmitá na vyšší frekvenci, přenese se přes vnitřní transformátor, a na sekundární straně obvodu se opět změní na digitální signál.

Jako sekundární řídící jednotka je v tomto obvodu využit procesor uvnitř integrovaného obvodu ESP8266. Tento Wi-Fi modul obsahuje procesor s jádrem ARM. Je možné ho restartovat pomocí restartovacích hřebínků, které jsou přes zkratovací propojku propojeny digitálním vstupním pinem a ošetřeny 4,7 kΩ rezistorem. Tento modul má vlastní napájením pomocí USB C, které je v rámci tohoto výukového modulu navržen na vstupech, na kterých se nachází také sběrnice UART. Pomocí tohoto procesoru lze ovládat periferie na jedné ze sběrnic I^2C, konkrétně pak digitální izolátor a optočlen. Dále je možné ovládat LED1 a LED2 a sedmi segmentový řadič. Primárně však tento modul slouží jako výukový, a to zejména pro vysvětlení základních principů pro odesílání dat na vzdálený server.

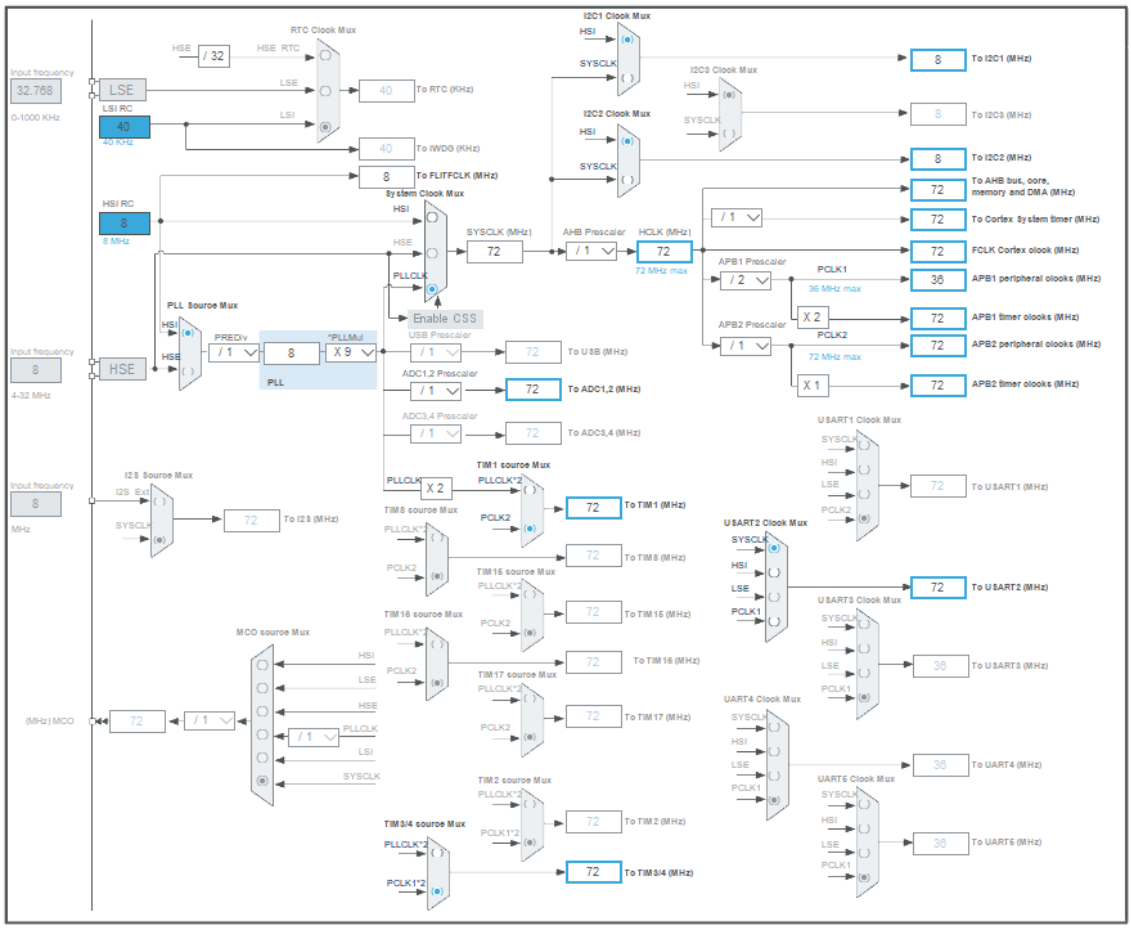
Modul ESP8266 lze k řízení laboratorní DPS využívat samostatně, tzn., že není nutné mít připojené Nucleo-64. Slouží jako řídící jednotka několika periferií, a také jako zdroj elektrické energie pro napájení obvodu.

2.2 Zapojení jednotlivých periferií na mikrokontroleru



Obr. 24: Zapojení jednotlivých periferií k pinům STM32 F303RE

Připojení jednotlivých periferií bylo realizováno v softwaru CubeMX od společnosti STMicroelectronics, připojení jednotlivých pinů lze vidět na obr. 24. Tento software obsahuje rozsáhlou knihovnu velkého množství mikrokontrolerů od společnosti STMicroelectronics. V této knihovně je možné vybrat si mikrokontroler, který je pro daný projekt použit, a díky uživatelsky přívětivému prostředí lze v několika krocích nastavit na výstupní piny mikrokontroleru požadovanou periferii. Tento software ulehčuje práci a šetří čas tím způsobem, že je možné si v rámci nástrojů, které tento software obsahuje, nastavit potřebné parametry týkající se technických požadavků na využívané periferie. Jedná se například o nastavení GPIO jako vstup, nebo výstup. Na které piny je možné připojit jaké sběrnice a jejich základní nastavení, například v rámci sběrnice SPI je možné nastavit, v jakém módu bude tato sběrnice komunikovat, zda se jedná full-duplex, half-duplex, master nebo receive komunikaci. V rámci UART sběrnice je možné nastavit, zda se jedná o synchronní, nebo asynchronní, případně jiné druhy módů. Dále je možné nastavit také časovače, na kterých kanálech bude dané periferie připojena, a dokonce i typ výstupního signálu.



Obr. 25: Zapojení hodin procesoru STM32 F303RE v prostředí CubeMX

Ve stejném softwaru lze dále pracovat také s nastavením jednotlivých zdrojů hodinového signálu pro použité periferie. Tyto hodiny lze vidět na obr. 25. Po nastavení všech potřebných parametrů lze pomocí stejného softwaru vytvořit tzv. knihovny HAL, což jsou knihovny, které jsou generovány právě programem CubeMX. V těchto knihovnách jsou již hotová základní nastavení v jednotlivých funkcích, která byla vygenerována na základě nastavovacích kroků v prostředí CubeMX. Kód, který je za pomoci tohoto softwaru vygenerován lze následně otevřít v prostředí STMCubeIDE, ve kterém byly realizovány jednotlivé ukázkové úlohy, ne však za využití HAL knihoven. HAL knihovny jsou užitečným nástrojem pro urychlení práce. Vzhledem k tomu, že jsou vytvořeny velice univerzálně, pro širokou škálu využití, nemusí mít jejich využívání na danou aplikaci vždy pozitivní vliv a program může být neefektivní. Práci s nimi popisuje jejich referenční manuál od výrobce.

Tab. 6: Nastavení Portu A

PIN	KONFIGURACE	PERIFERIE
PA0	GPIO_Input	BUT_1
PA1	ADC1_IN2	ADC
PA2	USART2_TX	USART_TX
PA3	USART2_RX	USART_RX
PA4	ADC2_IN1/DAC1_OUT1	DAC
PA5	GPIO_Output	LD2 [GreenLed]
PA6	GPIO_Input	SW_1
PA7	GPIO_Input	SW_2
PA8	TIM1_CH1	PWM_filtr
PA9	I2C2_SCL	I2C2_SCL
PA10	I2C2_SDA	I2C2_SDA
PA11	TIM4_CH1	PWM
PA12	GPIO_Output	LED_2
PA13	SYS_JTMS-SWDIO	TMS
PA14	SYS_JTCK-SWCLK	TCK
PA15	Není připojen	Není připojen

Porty A jsou nejvyužívanějšími porty ze všech portů na mikrokontroleru. Výše uvedená tabulka popisuje nastavení jednotlivých pinů a periferii, která je na ni připojena. Na pinech PA13 a PA14 jsou připojeny komunikační sběrnice pro komunikaci s programátorem J-link, který je součástí vývojové DPS Nucleo-64. Pin PA13 slouží k přenášení dat a pin PA14 k přenášení hodinového signálu. Jako jediný nepřipojený pin je pin PA15. Na portech A jsou připojeny celkem 2 komunikační sběrnice, I^2C2 a USART. Písmeno S znamená, že se jedná o asynchronní mód sběrnice. Dále se na Portu A nachází tři vstupní a dva výstupní GPIO piny, dva časovače, a to TIM1 a TIM4, oba komunikující na kanálu 1. Nakonec se na Portech A nachází také dva převodníky z analogového signálu na digitální ADC1 a ADC2. Konfigurace pro jednotlivé piny portu A lze vidět v tab. 6.

Tab. 7: Nastavení Portu B

PIN	KONFIGURACE	PERIFERIE
PB0	Není připojen	Není připojen
PB1	Není připojen	Není připojen
PB2	Není připojen	Není připojen
PB3	Není připojen	Není připojen
PB4	Není připojen	Není připojen
PB5	Není připojen	Není připojen
PB6	GPIO_Output	SPI_NSS3
PB7	Není připojen	Není připojen
PB8	I2C1_SCL	I2C1_SCL
PB9	I2C1_SDA	I2C1_SDA
PB10	Není připojen	Není připojen
PB11	Není připojen	Není připojen
PB12	GPIO_Output	SPI_NSS
PB13	SPI2_SCK	SPI2_SCK
PB14	SPI2_MISO	SPI2_MISO
PB15	SPI2_MOSI	SPI2_MOSI

Port B je na rozdíl od portu A a portu C nejméně využívaným portem. Z Celkem 16 možných pinů je využito pouze 7. Jsou využity Převážně pro sběrnice I^2C1 a SPI2. Sběrnice SPI1 není připojena vůbec, a to kvůli obsazenosti pinů jiným periferiemi. Sběrnice SPI2 je připojena celkem na tři Slave zařízení, a to na řadič MAX7219CNG a na akcelerometr LIS3DH. Třetí Slave zařízení lze připojit externě. Na portech B jsou připojeny jako GPIO výstup právě dvě zmíněné Slave zařízení akcelerometr a řadič. Konfigurace pro jednotlivé piny portu B lze vidět v tab. 7.

Tab. 8: Nastavení portu C

PIN	KONFIGURACE	PERIFERIE
PC0	GPIO_Output	SPI_NSS2
PC1	GPIO_Input	BUT_2
PC2	GPIO_Input	BUT_3
PC3	GPIO_Input	BUT_4
PC4	GPIO_Input	SW_3
PC5	GPIO_Input	SW_4
PC6	GPIO_Output	LED_1
PC7	TIM3_CH2	RGB_Red
PC8	TIM3_CH3	RGB_Green
PC9	TIM3_CH4	RGB_Blue
PC10	GPIO_Output	LED_3
PC11	GPIO_Output	LED_4
PC12	GPIO_Output	LED_5
PC13	GPIO_EXTI13	B1 [Blue PushButton]
PC14	RCC_OSC32_IN	RCC_OSC32_IN
PC15	RCC_OSC32_OUT	RCC_OSC32_OUT

Na portech C jsou připojeny převážně GPIO vstupy a výstupy. Celkem je zde připojeno pět vstupních GPIO a pět výstupních GPIO pinů. Na pinech PC13, PC14 a PC15 jsou připojeny výchozí periferie vývojové DPS Nucleo-64. Na portech PC7, PC8 a PC9 jsou připojeny 3 kanály časovače TIM3 pro ovládání RGB LED diody. Konfigurace pro jednotlivé piny portu C lze vidět v tab. 8.

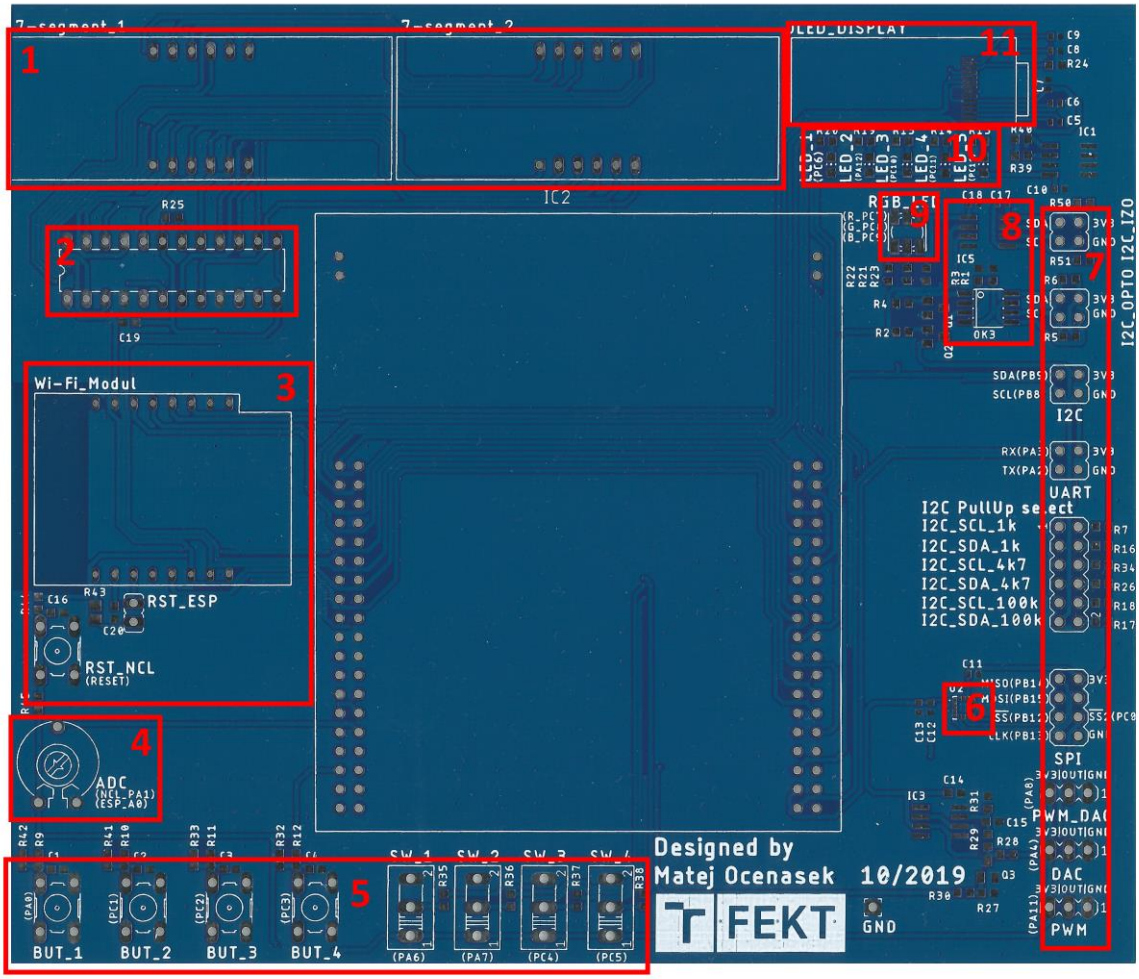
Tab. 9: Nastavení Portu D

PIN	KONFIGURACE	PERIFERIE
PD2	GPIO_Output	LCD_RESET

Port D má pouze jediný port, na kterém je připojený reset pro displej, který je nastaven jako GPIO výstup. Konfigurace pinu na portu D lze vidět v Tab. 9.

2.3 Návrh desky plošného spoje

Tato část se zabývá návrhem desky plošného spoje z pohledu rozmístění jednotlivých periferií, vhodného rozvedení vodičů dle zásad pro návrh plošných spojů.



Obr. 26: rozdělení jednotlivých bloků na DPS

Tab. 10: Tabulka popisující jednotlivé bloky na DPS

1	Dva sedmi segmentové displeje	7	hřebínky pro sledování signálů na osciloskopu
2	Řadič MAX7219CNG	8	Optočlen a digitální izolátor
3	Wi-Fi modul s ESP8266 a reset tlačítko	9	RGB LED dioda
4	ADC potenciometr	10	4x LED diody

5	Ovládací prvky tlačítka a přepínače	11	OLED displej 128x32
6	MEMS akcelerometr a gyroskop		

Rozložení jednotlivých bloků na DPS bylo navrženo tak, aby studentům usnadnilo práci, bylo intuitivní, a do maximální možné míry pohodlné. Popis jednotlivých bloků je možné vidět v tab. 10. Jednotlivé části obvodu byly rozděleny do tří základních bloků, řídící část, zobrazovací část a část pro sledování signálů pomocí osciloskopu. Jak je možné vidět na předchozím obrázku, v dolní části DPS se nachází komponenty, se kterými student nejčastěji přichází fyzicky do styku. Jedná se o 4x mikrospínače využívaná jako tlačítka, 4x přepínače a potenciometr pro ovládání analogového převodníku. Nad tímto blokem se dále nachází modul s ESP8266 a řadič MAX7219CNG pro sedmi segmentové displeje. Zobrazovací část se skládá ze dvou sedmi segmentových displejů, přičemž každý obsahuje 4 sedmi segmentové displeje, OLED displej 128x32 pixelů a 4x LED diody. V bloku pro sledování signálu jsou v řadě pod sebou seřazeny výstupní hřebínky jednotlivých sériových sběrnic a to SPI, I^2C a UART, přičemž I^2C sběrnice má navíc výstupy pro sledování signálu po galvanickém oddělení digitálním izolátorem a optočlenem. Dále jsou zde pak pro I^2C sběrnici výstupní hřebínky, na kterých je možné sledovat signál po změně pull-up rezistorů jak hodinového signálu, tak datové linky. Kromě sériových sběrnic lze také sledovat signál PWM, PWM DAC a DAC. V blízkosti toho bloku se nachází také MEMS senzor. Jednotlivé výstupní hřebínky jsou od sebe vzdáleny tak, aby bylo možné bez mechanických problému připojit vstupy osciloskopu. Uprostřed celé laboratorní DPS se bude na spodní straně nacházet Nucleo-64, které bude sloužit jako hlavní řídící jednotka pro veškeré periferie, dále bude sloužit jako programátor pro procesor STM32 a jako zdroj napětí v obvodu. Rozložení jednotlivých bloků na laboratorní DPS lze vidět na obr. 26.

Z pohledu návrhu vodivých cest byly dodrženy zásady návrhu pro desky plošných spojů. Signály byly na DPS vedeny co možná nejvíce na jedné vrstvě, přičemž byla snaha vést signály na vrstvě 1 horizontálně a na vrstvě 2 vertikálně. Signálové cesty, zejména pak datové vodiče a vodiče hodinových signálů byly vedeny co možná nejkratší cestou a ideálně vedle sebe, aby nedocházelo ke ztrátě dat. Vodiče byly vedeny co nejkratší cestou, aby linka byla zatížena co nejmenší kapacitou. Proto byly komponenty na DPS rozmístěny tak, aby byla tato pravidla co možná nejvíce zachována. Na celé DPS byla následně rozlita zem, aby byla DPS dobře odstíněná, aby byla co nejvíce odrušená a aby byl po celém plošném spoji dobrý rozvod zemnící plochy. Výjimkou na této DPS je plocha pod anténou Wi-Fi modulu, v této oblasti se nenachází žádná vodivá plocha, která by mohla mít negativní vliv na data vysílaná z této antény. Vodiče na DPS byly vedeny pod úhlem 45 °C, a to z toho důvodu, aby při výrobě DPS nedocházelo k podleptání. V rámci vedení signálových cest byla snaha cesty vést mimo integrované obvody a obecně zarušená místa, a to z toho důvodu, aby se rušení anulovalo.

2.4 Výukové materiály

V rámci této bakalářské práce byly vypracovány také výukové materiály, které se

skládají celkem z osmi částí, přičemž tři z nich tvoří doplňující materiály, jako popis pinů k jednotlivým periferiím na výukové DPS, návod na založení projektu a práci s debugovacím prostředím v programu STM32 CubeIDE a průvodce inicializací jednotlivých periferií. V rámci doplňujících materiálů, zejména pak u návodu k založení projektu byl kladen důraz na podrobnost tak, aby byl schopen projekt v prostředí STM32 CubeIDE nastavit i student, který se s daným prostředím do této chvíle nesetkal.

2.4.1 Manuály k nastavení projektu, inicializaci a periferiím

V přílohách lze najít tři dokumenty sloužící k lepší orientaci studentů ve výuce a ke snadnější realizaci úloh v laboratorních cvičeních. V příloze B.1 lze nalézt manuál pro nastavení projektu v prostředí STM32 CubeIDE. Jedná se o prostředí vytvořené společností STMicroelectronics, a je primárně určeno k realizaci projektů právě na mikrokontrolerech této značky. Manuál provádí studenta základními kroky od spuštění programu STM32 CubeIDE, přes založení vlastního projektu, až po manipulaci s nástroji v debugovacím prostředí, které jsou užitečné pro firmwarové úpravy programu v mikrokontroleru v reálném čase. V rámci založení a nastavení projektu se využívají soubory vytvořené v prostředí CubeMX mimo laboratorní cvičení, studenti jsou tak odkazování na Elearning, kde jsou pro ně tyto materiály volně dostupné ke stažení a bez kterých je realizace úloh nemožná. Součástí souborů volně dostupných na Elearningu jsou také knihovny a soubor main.c, který obsahuje základní inicializaci jednotlivých periferií. Manuál je realizován písemnou formou, která obsahuje graficky znázorněné části při zakládání projektu a další manipulaci v prostředí STM32 CubeIDE.

V příloze B.3 jsou uvedeny jednotlivé periferie obvodu. Za pomoci tohoto dokumentu je student schopen rychleji vytvořit zadaný program. Zároveň má možnost vidět, jak jsou jednotlivé periferie na desce připojeny, tudíž je schopen si odvodit, jak má program napsat, aby správně fungoval. Například v rámci úlohy, která obsahuje tlačítka je student nucen vytvořit funkci, která detekuje stisk tlačítka. Aby byl schopen napsat správně, musí vidět elektrické zapojení na přípravku. V této příloze si student najde, jak jsou v tomto případě tlačítka na přípravku zapojena, zjistí že spínají k zemi, protože jsou připojeny přes pull-up rezistor a program tomu přizpůsobí. Stejně tak bude postupovat i u dalších periferií, jako například u LED. Pro znázornění elektrického zapojení jednotlivých periferií na přípravku byly použity části zapojení realizovaného v schématické části softwaru Eagle.

V příloze B.2 se nachází manuál k inicializaci periferií STM32 F303RE. Tento průvodce prezentuje nastavení jednotlivých registrů pro různé periferie mikrokontroleru, použité v rámci laboratorních úloh jako například GPIO, ADC, DAC, sběrnice, nebo například řadič MAX7219CNG. Jednotlivé registry byly nastavovány na základě referenčního manuálu, ve kterém je pro každou periferii uveden návod, co do jakého registru zapsat a proč. Nastavování jednotlivých registrů může být časově náročné, například funkce ADC převodníku jsou tak rozsáhlé, že je ve spoustě aplikací, včetně laboratorních úloh, nutné nastavit jen některé registry tak, aby A/D převodník plnil požadovanou funkci. Každý registr v příloze B.2 obsahuje také krátký komentář, který však v mnohých případech pouze naznačí, co daný registr dělá. Pro lepší porozumění funkce jednotlivých registrů je vhodné využít referenční manuál, ve kterém je popis jednotlivých registrů popsán podrobněji. Tento dokument byl vytvořen primárně za účelem přiblížení funkce mikrokontroleru. Snaží se studentovi přiblížit

problematiku nastavování jednotlivých pinů. Jedná se totiž o první krok při psaní programu v případě, že k tvorbě programu nejsou použity externí knihovny, jako například knihovny HAL, které jsou automaticky generovány při zakládání projektu. Vytváření vlastní inicializační části programu vede k lepší přehlednosti a kontrole nad daným programem, které je užitečné zejména při hledání chyb v nefunkčním programu. Zejména v praxi, kdy je vyvíjen tlak na cenu finálního zařízení, může právě vlastní část inicializace v programu pozitivně přispět k ušetření nákladů, a to z toho důvodu, že je možné volit méně výkonný a tím i levnější mikrokontroler, který však při správně a efektivně napsaném programu dokáže vykonávat požadovanou funkci.

2.4.2 Laboratorní úloha 1

V první laboratorní úloze uvedené v příloze A.1 je student seznámen se základní prací s LED, tlačítky a přepínači. U všech těchto periferií se jedná o GPIO piny, v případě LED o výstupní a v případě tlačítek a přepínačů o vstupní. Student je v začátku laboratorní úlohy pomocí teoretického úvodu seznámen s problematikou dané laboratorní úlohy. Následně je mu představeno elektrické zapojení na DPS laboratorního přípravku a jsou mu zadány úlohy, které má za úkol v rámci laboratorního cvičení zpracovat. Komentáře pod jednotlivými částmi programu vysvětlují studentovi význam daných částí kódu v programu, význam jednotlivých registrů a často také posloupnost prováděných funkcí. Každý řádek kódu je navíc opatřen komentářem, který popisuje funkci daného řádku. Jednotlivé úlohy v rámci cvičení mají tendenci stupňovat míru náročnosti. V rámci této laboratorní úlohy student začíná střídavým rozsvěcením jednotlivých LED dle zadání a končí binárním a dekadickým čítačem, jehož součástí jsou nejen tlačítka, ale i přepínače. V rámci realizace úloh je vhodné využívat referenční manuál, který je dostupný na Elearningu. V rámci nástroje SFR v prostředí STM32 CubeIDE má student možnost přenastavit některé registry v reálném čase, a sledovat tak tuto změnu na LED. Nejjednodušším způsobem, jak sledovat tuto změnu, je měnit nastavení registrů BR (bit reset) nebo BS (bit set). Měněním hodnot v jednom z těchto registrů dochází k zapínání nebo vypínaní jednotlivých LED, a proto je možné takřka okamžitě vidět, jestli LED svítí/nesvítí.

2.4.3 Laboratorní úloha 2

V druhé laboratorní úloze uvedené v příloze A.2 je student seznámen s časovači a využívá znalostí z minulé laboratorní úlohy, a to práce s GPIO vstupy a výstupy. V úvodu úlohy je student opět seznámen pomocí teoretického úvodu s problematikou laboratorního cvičení a pomocí schématu se zapojením dané periferie na laboratorním přípravku. V této úloze je využit časovač TIM3 na kanálech 2, 3 a 4, který pomocí PWM ovládá úroveň intenzity záření RGB diody. Tento časovač je 16bitový. Student musí tedy v rámci úloh nastavit pomocí registru CCR pro patřičný kanál střídu (anglicky duty cycle) na 0 a postupně stiskem tlačítka, která už se naučil používat v předchozí úloze, inkrementovat hodnotu na daném registru do doby, než dosáhne maximální úrovně. V následující úloze je student seznámen s nastavením předděličky, která nám umožní vydělit vstupní hodinový signál časovače. Nastavuje se když potřebujme dosáhnout nižších frekvencí. V případě časovače TIM3 je to 72MHz. PWM,

které je generováno na výstup pinů PC7, PC8 a PC9 k ovládání RGB diod je v průmyslu hojně využíváno například k řízení motorů.

2.4.4 Laboratorní úloha 3

Ve třetí laboratorní úloze uvedené v příloze A.3 je student seznámen s prací se sběrnicí SPI ve režimu Master transmitt, tzn. že zařízení, na které jsou data odeslány neodpovídá zpátky, a s řadičem pro sedmi segmentové displeje MAX7219CNG. Stejně jako v předchozích úlohách, tak i zde jsou využívány dovednosti získané v předchozích úlohách, a to GPIO vstupy, zejména pak tlačítka a přepínače. Novým GPIO vstupem bude v této úloze Slave zařízení, kterým je právě zmíněný řadič MAX7219CNG. Nastavení vysílací funkce MasterTransmitt a základní nastavení řadiče je pro tuto úlohu nachystané v souboru main.c na Elearningu. Studenti mají tedy v rámci této úlohy poslat na adresu uvedenou v katalogovém listu řadiče patřičná data. Tato data pošlou pomocí funkce MasterTransmitt. Dle dokumentu uvedeného v příloze B.2 si studenti nastudují, jak je řadič MAX7219CNG nastaven. Katalogový list je možné stáhnout z Elearningu. Cílem této úlohy je naučit studenty pracovat se sběrnicí SPI a určit výhody řadiče, kterými jsou zejména menší obsazenost pinů na mikrokontroleru. Řadič totiž využívá k rozdělování dat na patřičné digity multiplexování. SPI sběrnice je sériová a je závislá na hodinovém signálu.

2.4.5 Laboratorní úloha 4

Ve čtvrté laboratorní úloze uvedené v příloze A.4 je student seznámen se sběrnicí I^2C, zapisováním dat do paměti EEPROM a zapisováním dat na OLED displej. Knihovny pro realizaci těchto úloh si student stáhne z Elearningu a pomocí návodu v příloze B.1 je nahraje do svého projektu. Tato úloha si klade za cíl naučit studenty práci se sběrnicí I^2C, jejím datovým rámcem a rychlostí na kterých tato sběrnice může komunikovat. Student opět využívá získaných dovednosti s GPIO piny z předchozích úloh. Realizace úloh je prováděna za pomoci funkcí obsažených v knihovnách, které byly zmíněny výše. Zapisování dat do paměti, nebo na displej je v rámci vývojového procesu embedded systémů velice nezbytnou součástí celého procesu ladění daného zařízení.

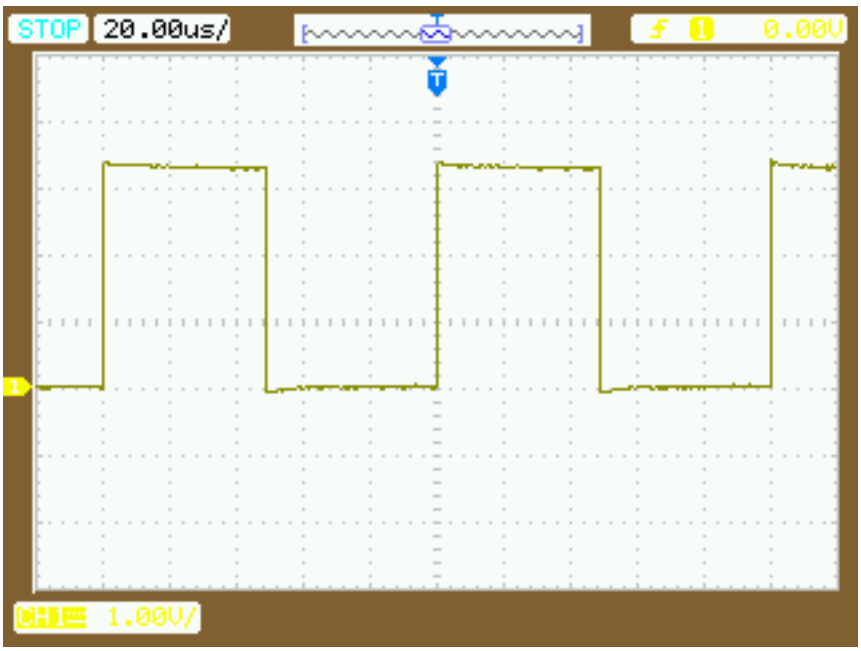
2.4.6 Laboratorní úloha 5

V páté laboratorní úloze uvedené v příloze A.5 je se studenty opakována získaná dovednost z úlohy č. 3, a to konkrétně práce se sběrnicí SPI. V tomto případě se však jedná o pokročilejší využití této sběrnice, a to z toho důvodu, že Slave zařízením je v této úloze akcelerometr LIS3DH, ze kterého je potřeba získat data, která se následně zobrazí na displej formou textu a následně formou grafu jedné z os akcelerometru, tedy X, Y, nebo Z. V této úloze nebude sběrnice provozována pouze v režimu vysílání, tedy Master trasmitt, ale bude data z akcelerometru také přijímat. V této úloze bude tedy využívána funkce Master trasmitt receive. Tato funkce umožňuje také přijímaní dat

z ovládaného zařízení. Základní nastavení používaných sběrnic a zařízení je obsaženo v rámci knihoven dostupných z Elearnignu.

2.4.7 Měření výstupních signálů pomocí osciloskopu

Součástí výukové DPS jsou již výše zmíněny hřebínky, na kterých lze pomocí osciloskopu sledovat signál PWM, DAC, sběrnic SPI a USART. Mimo tyto periferie je na laboratorním přípravku zapojen také optočlen a digitální izolátor. Na tyto periferie jsou data posílána pomocí sběrnice I^2C1, jejíž rychlost byla nastavena na 100 kHz. Na tuto sběrnici byly také připojeny měnitelné pull-up rezistory, které mají vliv na tvar výstupního signálu. Tudíž je možné sledovat, jak se mění tvar signálu v závislosti na pull-up rezistorech a také v závislosti na galvanickém oddělení pomocí optočlenu a digitálního izolátoru. V této kapitole budou výsledky měření jednotlivých kanálu zobrazeny pomocí obrázků získaných z osciloskopu.

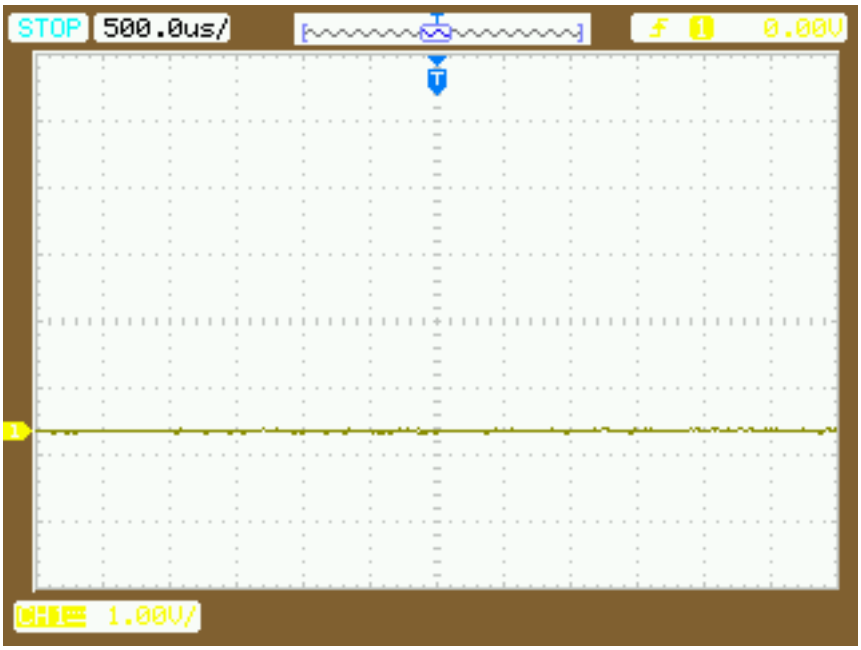


Obr. 27: PWM signál se střídou 50%

Výše uvedený graf popisuje obdélníkový průběh výstupního signálu PWM na pinu PA11 při střídě 50 %. Signál byl generován časovačem TIM4 na kanálu 1. Tento signál lze vidět na obr. 27.

Obr. 28: Výstupní signál DA převodníku

Průběh analogového signálu na DA převodníku, přičemž digitální signál byl získán z pinu PA4. Tento signál lze vidět na obr. 28.

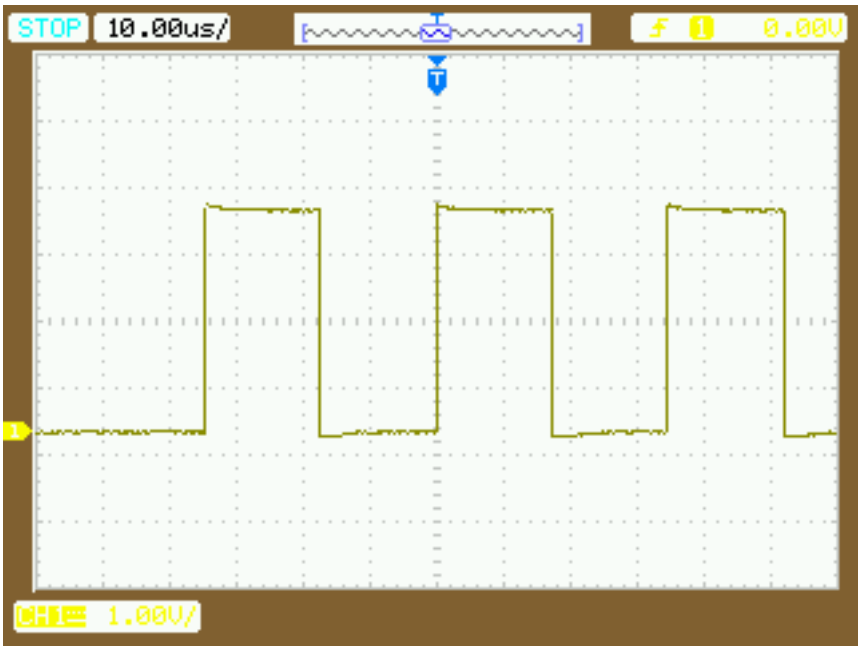


Obr. 29: Výstupní signál PWM na DA převodníku

Tvar výstupního signálu po převodu z digitálního signálu na analogový. PWM signál byl generován na pinu PA8 pomocí časovač TIM1 na kanálu 1. Tento signál lze vidět na obr. 29.

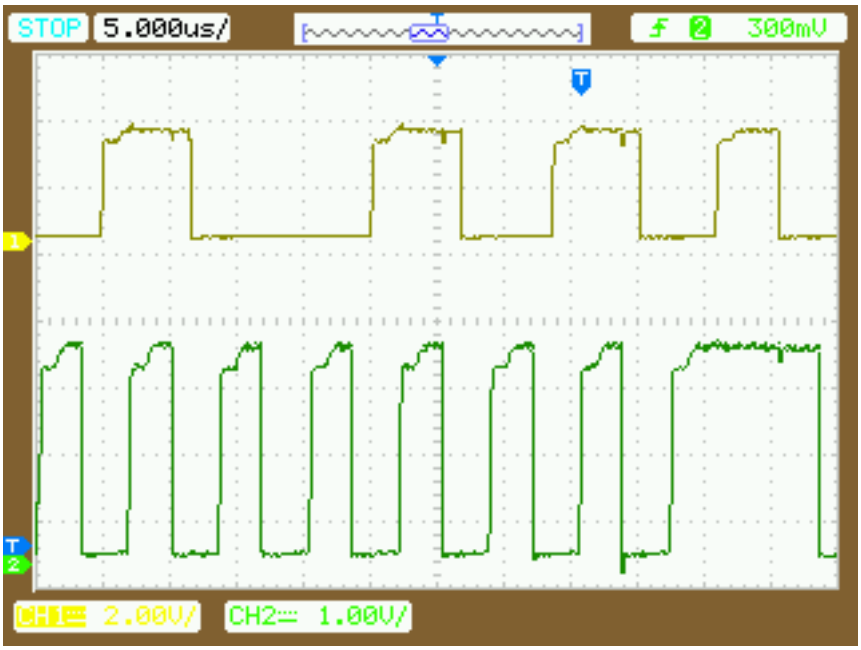
Obr. 30: Datový a hodinový signál na sběrnici SPI

Výstupní signál na SPI sběrnici. Data byla posílána ve tvaru 0xAA. Zelená barva značí hodinový signál, žlutá značí datový signál. Tento signál lze vidět na obr. 30.

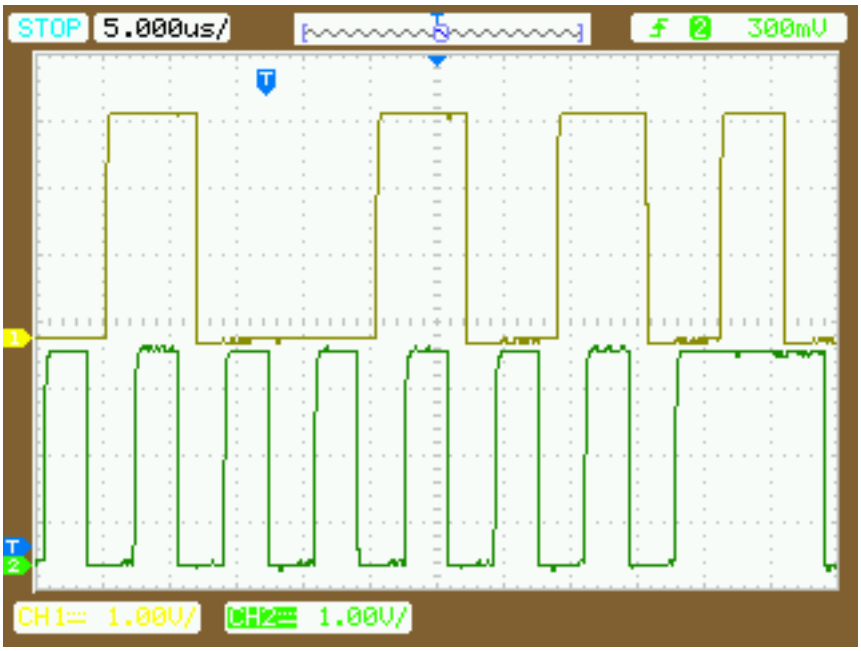


Obr. 31: Vysílání na sběrnici USART - TX linka

Výstupní signál na sběrnici UART. Data odeslaná na sběrnici byla ve tvaru 0xAA. Tento signál lze vidět na obr. 31.



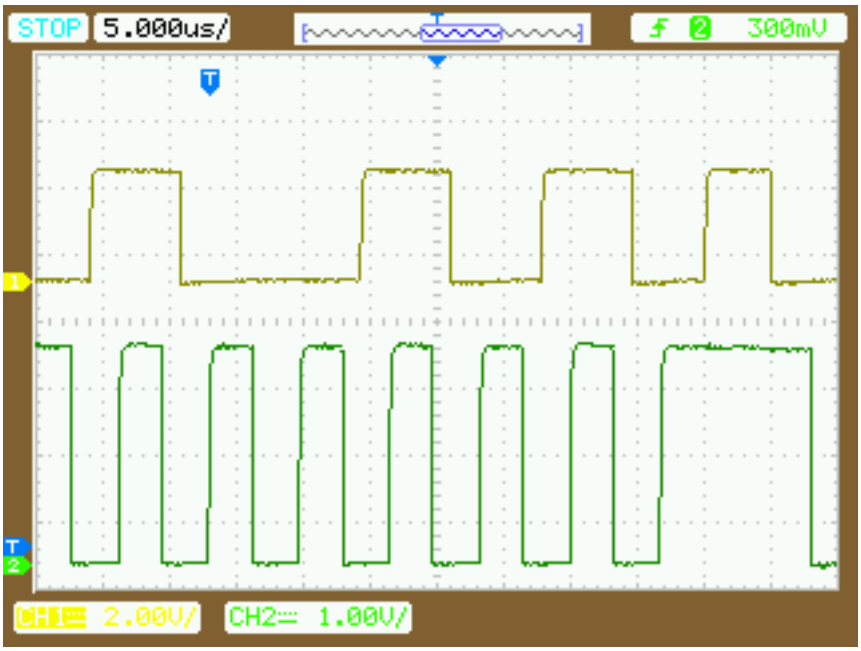
Obr. 32: Výstup sběrnice I2C1 při pull-upu rezistoru 1kΩ



Obr. 33: Výstup sběrnice I2C1 při pull-upu rezistoru 1kΩ za digitálním izolátorem

Výše uvedené obrázky znázorňují výstupní signál na sběrnici I^2C1 při pull-up rezistoru 1 kΩ. Zelený signál značí hodinový signál, žlutý značí datový signál. Na sběrnici byla odeslána data 0xAA při rychlosti 100 kHz. Tento signál lze vidět na obr. 32 a obr. 33.

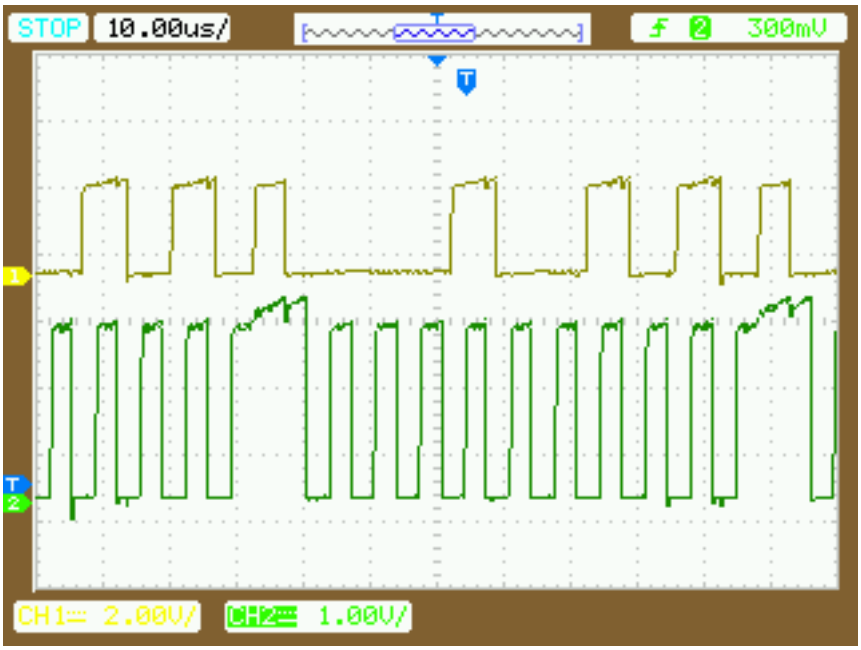
Obr. 34: Výstup sběrnice I2C1 při pull-upu rezistoru 4,7kΩ



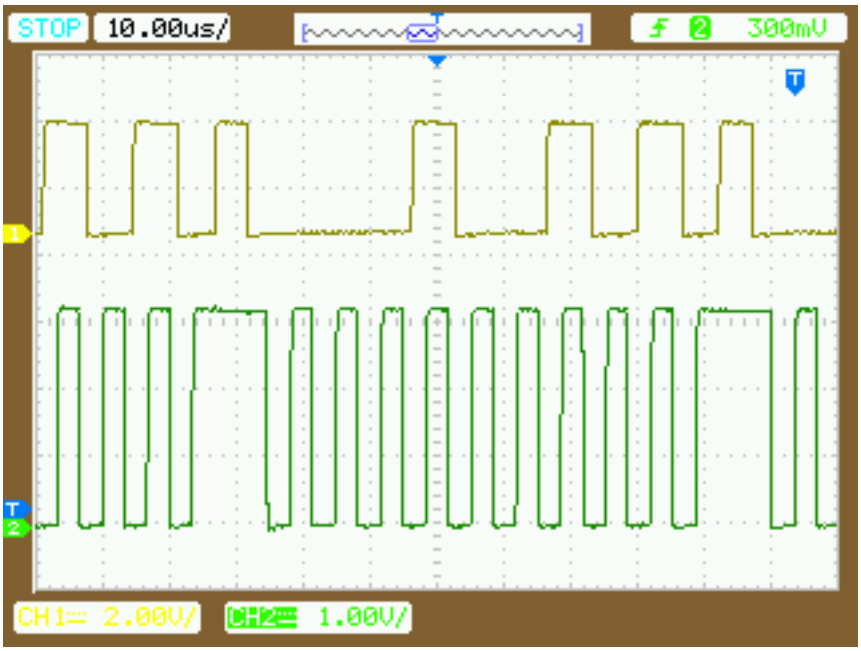
Obr. 35: Výstup sběrnice I2C1 při pull-upu rezistoru 4,7kΩ za digitálním izolátorem

Ještě optočlen

Výše uvedené obrázky znázorňují výstupní signál na sběrnici I^2C1 při pull-up rezistoru 4,7 kΩ. Zelený signál značí hodinový signál, žlutý značí datový signál. Na sběrnici byla odeslána data 0xAA při rychlosti 100 kHz. Tento signál lze vidět na obr. 34 a obr. 35.



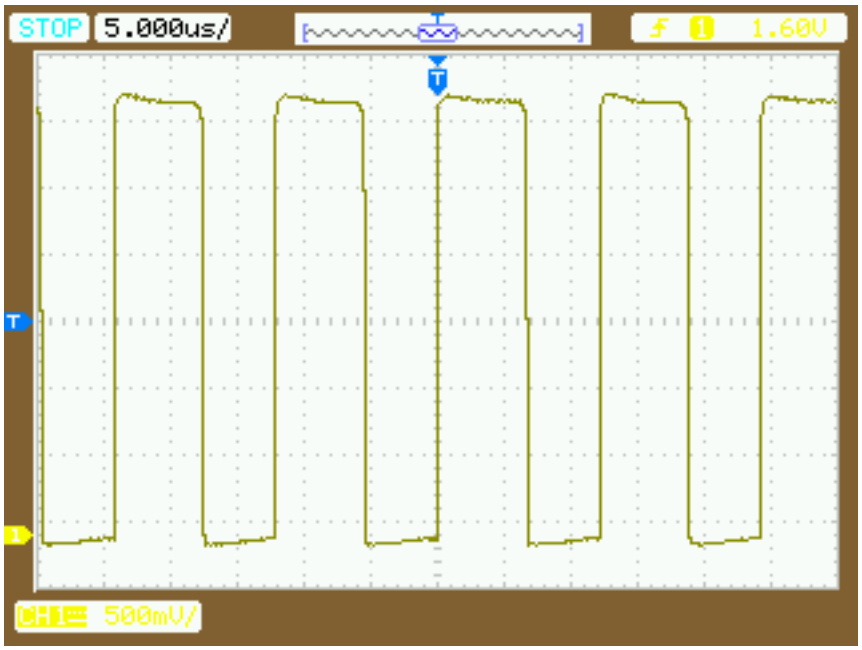
Obr. 36: Výstup sběrnice I2C1 při pull-upu rezistoru 100kΩ



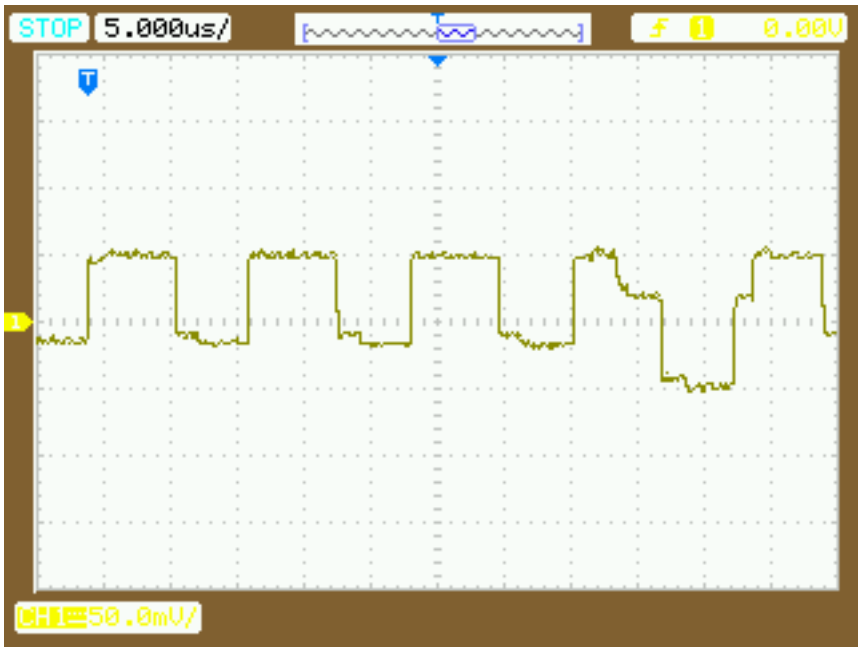
Obr. 37: Výstup sběrnice I2C1 při pull-upu rezistoru 100 kΩ za digitálním izolátorem

Tady bude jeset optočlen

Výše uvedené obrázky znázorňují výstupní signál na sběrnici I^2C1 při pull-up rezistoru 100 kΩ. Zelený signál značí hodinový signál, žlutý značí datový signál. Na sběrnici byla odeslána data 0xAA při rychlosti 100 kHz. Tento signál lze vidět na obr. 36 a obr. 37.



Obr. 38: Hodinový signál I2C při pull-up rezistoru 100 kΩ před optočlenem



Obr. 39: Hodinový signál I2C při pull-up rezistoru 100 kΩ za optočlenem

Výše uvedený obr. 38 a obr. 39 znázorňuje průběh hodinového signálu při komunikaci na sběrnici I^2C před a za optočlenem. Na sběrnici byl použit pull-up 100 kΩ.

3. ZÁVĚR

Cílem této práce bylo uvést do problematiky procesorů využívajících jádro ARM, a to přesně procesorů STM32 a Tensillica L106, popsání jednotlivých periferií laboratorní DPS a přiblížit problematiku návrhu schématu a samotného návrhu DPS z pohledu rozmístění a propojení signálovými cestami. Oba již zmíněné procesory pracují na 32bitech a používají instrukční sadu RICS. Byla teoreticky přiblížena problematika sériových sběrnic, jak z pohledu programové obsluhy, tak z pohledu přenosu samotného datového a hodinového signálu, byla blíže specifikována role hlavní řídící jednotky a jí podřízené periferie. Dále byl přiblížen proces změny digitálního signálu na analogový a obráceně, základní principy paměti EEPROM a pulzně šířkové modulace PWM.

Z pohledu návrhu schématu pak došlo k objasnění výběru jednotlivých periferií, jejich významu v rámci obvodu, funkci a postup výběru jednotlivých RC článku. Z pohledu schématu byly mimo jiné popsány funkce jednotlivých periferií, jejich přednosti apod. Dále bylo vysvětleno rozložení jednotlivých periferií v rámci DPS, kde byl záměr vytvořit uživatelsky co nepřívětivější ovládání, které bude také intuitivní. Popis jednotlivých komponentů na desce tak, jak jsou připojeny na procesor, aby byla ulehčena práce při tvorbě programu. Byl vysvětlen postup rozvedení jednotlivých signálových cest na DPS z pohledu návrhových zásad tak, aby nedocházelo k chybám při přenosu dat po datových linkách.

Po realizaci laboratorní DPS byly pomocí mikrokontroleru STM32F303RE realizovány také laboratorní úlohy. Těchto úloh bylo realizováno celkem pět. Každá z úloh obsahuje teoretický úvod, ve kterém je student seznámen s problematikou, dále každá úloha obsahuje schéma zapojení na přípravku a zadání pro samostatné vypracování. Jednotlivé úlohy jsou zároveň také vypracovány. Podstatné části kódu obsahují krátký komentář, v případě, že se části kódu opakují, pak komentář neobsahují. Části, které je potřeba podrobněji vysvětlit, obsahují rozsáhlejší komentář v rozsahu jednoho několikařádkového odstavce. Pro výuku byly nad rámec této bakalářské práce vytvořeny materiály ve formě manuálu, které mají studentům pomoct nastavit projekt, usnadnit orientaci při nastavování jednotlivých pinů, nebo nastínit způsob vytváření inicializace procesoru bez použití knihoven.

POUŽITÉ ZDROJE:

[1] *What is Serial Communication and How it works?* [online]. Codrey elektronic. [Cit. 11.12.2019]. Dostupné z: https://www.codrey.com/embedded-systems/serial-communication-basics/

[2] *STM32F303* [online]. STMicroelectronic. [Cit. 11.12.2019]. Dostupné z: https://www.st.com/en/microcontrollers-microprocessors/stm32f303.html#overview

[3] *STM32F303RE: datasheet. STMicroelectronic* [online]. [Cit. 11.12.2019]. Dostupné z: https://www.st.com/resource/en/datasheet/stm32f303re.pdf

[4] *Nucle-64: dataseet. STMicroelectronic* [online]. [Cit. 11.12.2019]. Dostupné z: https://www.st.com/resource/en/data_brief/nucleo-f303re.pdf

[5] *Nucleo-64 STM32L476, Funduino* [online.] [Cit. 28. 5. 2020]. Dostupné z: https://www.funduinoshop.com/epages/78096195.sf/en_GB/?ObjectPath=/Shops/78096195/Products/03-133

[6] *STM32F107RTC6 – ARM MCU, Farnell* [online]. [Cit. 28. 5. 2020]. Dostupné z: https://cz.farnell.com/stmicroelectronics/stm32f107rct6/mcu-32bit-cortex-m3-72mhz-lqfp/dp/1737139

[7] *ESP-8266-12 WiFi module with 9 GPIO, Nettigo* [online]. [Cit. 28. 5. 2020]. Dostupné z: https://nettigo.eu/products/esp-8266-12-wifi-module-with-9-gpio--2

[8] *NodeMCU a jeho verzie: doska s Wi-Fi čipom ESP8266*. [online]. ROOT.cz [Cit. 11.12.2019]. Dostupné z: https://www.root.cz/clanky/nodemcu-a-jeho-verzie-doska-s-wi-fi-cipom-esp8266/

[9] *ESP8266: datasheet. Espressif* [online]. [Cit. 11.12.2019]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

[10] *ESP8266 12-E Chip pinout, Random Nerds Tutorials* [online]. [Cit. 28. 5. 2020]. Dostupné z: https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/

[11] DHAKER, Piyu. *Introduction to SPI interface*. Analog Devices [online]. [Cit 11.12.2019]. Dostupné z: https://www.analog.com/media/en/analog-dialogue/volume-52/number-3/introduction-to-spi-interface.pdf

[12] AFZAL, Sal. *I^2C Primer: What is I^2C*. Analog Devices [online]. [Cit. 11.12.2019]. Dostupné z: https://www.analog.com/en/technical-articles/i2c-primer-what-is-i2c-part-1.html

[13] *Basics of UART communication*. [online]. Circuit basics. [Cit. 11.12.2019]. Dostupné z: http://www.circuitbasics.com/basics-uart-communication/

[14] *Analog to digital converter* [online]. Electricalu 4U. [Cit. 11.12.2019]. Dostupné z: https://www.electrical4u.com/analog-to-digital-converter/

[15] *How to convert the analog signal to digital signal by ADC converter*. [online]. Electronic projects focus. [Cit. 11.12.2019]. Dostupné z: https://www.elprocus.com/analog-to-digital-adc-converter/

[16] *PWM: Pulse Width Modulation: What is it and how does it work?* [online]. Analog IC tips. [Cit. 11.12.2019]. Dostupné z: https://www.analogictips.com/pulse-width-modulation-pwm/

[17] RAJ, Aswinth, *What is PWM: Pulse Width Modulation.* Circuit digest [online]. [Cit. 11.12.2019]. Dostupné z: https://circuitdigest.com/tutorial/what-is-pwm-pulse-width-modulation

[18] *What is EEPROM Memory Technology* [online]. Electronics notes. [Cit.12.12.2019]. Dostupné z: https://www.electronics-notes.com/articles/electronic_components/semiconductor-ic-memory/eeprom-e2prom-technology.php

SEZNAM ZKRATEK:

V	Volt (jednotka napětí)
SPI	Serial Peripheral Interface (sériové periferní rozhraní)
UART	Universal asynchronous receiver-transmitter
I^2C	Integrated circuit (dvou vodičová sběrnice)
MSB	Most significant bit (nejvýznamější bit)
LSB	Least significant bit (nejméně významný bit)
CAN	Controller Area Network
A/D	analog to digitál
D/A	digitál to analog
SRAM	Static Random Acces Memory (statická paměť)
MOSI	Master výstup Slave vstup
MISO	Master vstup Slave výstup
SCLK	hodinový signál
NSS	Slave select
CS	Chip select
SDA	Synchronous data
SCL	Synchronous clock
pF	piko Farad (jednotka kapacity)
Tx	vysílání
Rx	přijímání
BPS	bit per second (bit za sekundu)
DAC	digital to analog converter
ADC	analog to digital converter
CD	Compact Disc
U	fyzikální veličina napětí
U_{IN}	vstupní napětí
U_{REF}	referenční napětí
IO	integrovaný obvod
PWM	Pulse Width Modulation
Hz	Hertz (jednotka frekvence)
ROM	Read-Only Memory
EEPROM	Electrically Erasable Programmable Read-Only Memory
PROM	Programmable Read Only Memory
RAM	Random acces memory
CPU	Hlavní řídící jednotka (central processor unit)

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

A - LABORATORNÍ ÚLOHY

A.1 Laboratorní úloha č. 1

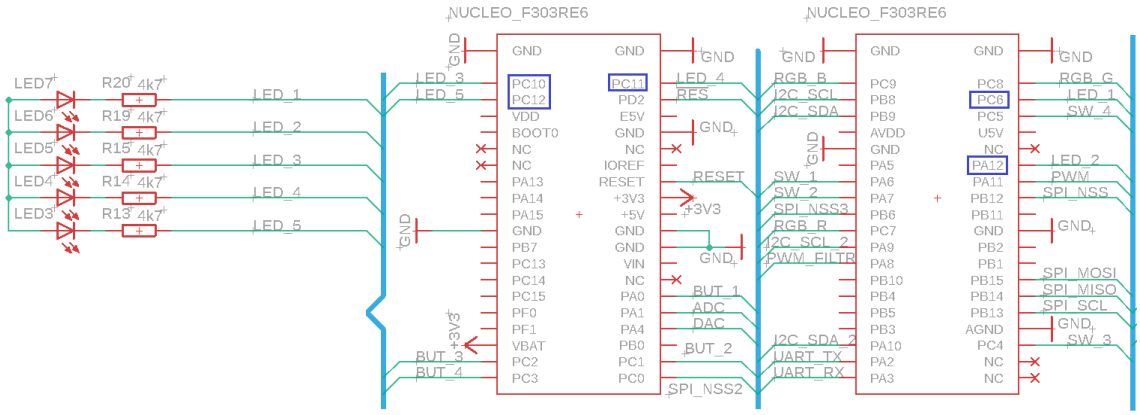
Základní práce s GPIO piny, využití5x LED

Teoretický úvod:

GPIO piny jsou vývody, které lze za pomoci procesoru volně ovládat. Lze je nastavit jako digitální vstup, digitální výstup nebo po odpojení digitálního bufferu lze získat analogovou funkci. Typickým zapojením digitálního vstupu jsou například tlačítka, nebo přepínače. Typickým zapojením digitálního výstupu jsou například LED, nebo motory.

Úlohy budou realizovány v prostředí STM32 CubeIDE za využití jazyka C. K realizaci kódu bude využit již hotový init (základní nastavení) GPIO pinů, který je obsažen v souboru main.c. LED jsou připojeny k pinům PC6, PC10, PC11, PC12, PA12, tlačítka pak PA0, PC1, PC2, PC3, přepínače jsou připojeny na piny PA6, PA7, PC4, PC5. V debugovacím prostředí můžeme program pro názornost krokovat.

Schéma zapojení laboratorního přípravku:



Obrázek 1. Schéma zapojení LED a procesoru

Zadání úloh:

1) Rozsviťte všech 5 LED střídavě, tedy „10101“ nebo „01010“
2) Vytvořte sledovač signálu, tedy stisk tlačítka bude přímo ovlivňovat patřičnou LED diodu (TL1 – LED1, … TL4-LED4)
3) Jednoduchý čítač – TL1 a TL2 bude přičítat a odčítat „1“ a celkový počet bude zobrazen na LED diodách. Přepínač SW1 bude měnit způsob čítání – BIN/DEC
4) Vytvořte program čítající LED zleva doprava, jehož směr se otočí stiskem tlačítka

Úloha 1) Rozsviťte všech 5 LED střídavě, tedy „10101“ nebo „01010“

Jako první krok je dle dokumentu v Elearningu nutné založit nový projekt v prostředí STM32 CubeIDE. Všechny kroky, nutné k založení projektu, jsou v něm podrobně popsány.

```
#define STM32F303xE                          //Definování typu MCU
```

Definování použitého typu procesoru je nutné kvůli HAL knihovnám.

```
#include "stm32f3xx.h"                       //knihovny pro STM32 řady F
#include "core_cm4.h"                        //core knihovna CMSIS
#include "stm32f303xe.h"                     //knihovny STM32f303
```

Na začátku každého programu je potřeba inkludovat knihovny pro použitý mikrokontroler. V programovacím jazyce C direktiva #include říká preprocesoru aby vložil obsah jiného souboru do zdrojového kódu na místě kde je tato direktiva #include napsána.

```
void init (void);                            //Deklarace funkce pro init MCU
void GPIO_init(void);                  //Deklarace funkce pro init GPIO pinů
```

Před funkcí „**int main (void)**“, ve které se budou jednotlivé programy psát, je nutné deklarovat funkce, ve kterých je vytvořeno základní nastavení používaných periferií. Pro úlohy v tomto cvičení jsou to funkce **init** (která vytváří základní nastavení celého mikrokontroléru) a funkce **GPIO_init** (která vytváří základní nastavení používaných GPIO pinů).

```
int main(void)
{
  init();                              //Zavolání funkce init
  GPIO_init();                         //Zavolání funkce GPIO_init

  GPIOC->BSRR = GPIO_BSRR_BR_6 | GPIO_BSRR_BR_10 | GPIO_BSRR_BS_11 |
                GPIO_BSRR_BR_12;
  //Rozsvícení LED 1 (PC6), LED 3 (PC10) LED5 (PC12)
  //Zhasnutí LED 4 (PC11)
  GPIOA->BSRR = GPIO_BSRR_BS_12;  //Zhasnutí LED2 (PA12)
}
```

V hlavním funkci **main** je potřeba zavolat funkce **init** a **GPIO_init**. V registru **BSRR** se pomocí bitů BR a BS rozsvěcuje, nebo zhasíná LED. **BR** znamená „bit reset“ tudíž nastaví hodnotu na výstupu na logickou 0 a BS znamená „bit set“ a ten nastaví hodnotu na výstupu na logickou 1. LED mají anodu připojenou na kladné napětí, procesorem potřebujeme sepnout katodu k zemi, aby LED svítila. Tímto vzniká negativní logika, tudíž zapsáním do bitu **BR** v registru **BSRR** dochází k rozsvícení LED a zapsaní do bitu **BS** v registru **BSRR** dochází k zhasnutí LED.

Úloha 2) Vytvořte sledovač signálu, tedy stisk tlačítka bude přímo ovlivňovat patřičnou LED diodu (TL1 – LED1, … TL4-LED4)

V již vytvořeném projektu můžeme úlohu 1 buď smazat, nebo zakomentovat. Komentář lze vytvořit buď zapsáním „//“ na začátek řádku, nebo „/*“ před první komentovaný znak a „*/“ za poslední komentovaný znak.

V následující úloze zachováme tedy vše, kromě původního programu v **main** funkci.

```
int main(void)
{
  for(;;)            //nekonečný cyklus
  {
            if (!(GPIOA->IDR & GPIO_IDR_0))
        //Pokud je tlačítko BUT_1 stisknuté
        {
              GPIOC->BSRR |= GPIO_BSRR_BR_6; //Rozsvítit LED_1 (PC6)
              //Pokud je LED_1 už rozsvícená, zůstane rozsvícená
        }
        else         //Pokud tlačítko BUT_1 není stisknuté
        {
              GPIOC->BSRR |= GPIO_BSRR_BS_6; //Zhasnout LED_1 (PC6)
              //Pokud je LED_1 už zhasnutá, zůstane zhasnutá
        }

        if (!(GPIOC->IDR & GPIO_IDR_1))
        {
              GPIOA->BSRR |= GPIO_BSRR_BR_12;
        }
        else
        {
              GPIOA->BSRR |= GPIO_BSRR_BS_12;
        }

        if (!(GPIOC->IDR & GPIO_IDR_2))
        {
              GPIOC->BSRR |= GPIO_BSRR_BR_10;
        }
        else
        {
              GPIOC->BSRR |= GPIO_BSRR_BS_10;
        }

        if (!(GPIOC->IDR & GPIO_IDR_3))
        {
              GPIOC->BSRR |= GPIO_BSRR_BR_11;
        }
        else
        {
              GPIOC->BSRR |= GPIO_BSRR_BS_11;
        }
  }
}
```

Tento krátký program využívá k rozsvícení LED zmáčknutí tlačítka odpovídajícího poloze LED. K dosažení informace o zmáčknutí tlačítka se využívá bit **IDRy** v registru **IDR** (input data register). Tlačítko spíná k zemi, tudíž je potřeba podmínku **if** negovat za pomoci „!“. Podmínka říká, že pokud dojde ke stisku tlačítka, rozsvítí se příslušná led. Pokud ne, LED zůstane zhasnutá. **IDR** registr je „read-only“, což ve volném překladu znamená že je určen pouze ke čtení, nelze do něj zapisovat.

Úloha 3) Jednoduchý čítač – TL1 a TL2 bude přičítat a odčítat „1“ a celkový počet bude zobrazen na LED diodách. Přepínač SW1 bude měnit způsob čítání – BIN/DEC

Podobně jako v předchozím kroku je potřeba zakomentovat, nebo vymazat původní program. Celý níže uvedený program se nachází v **main** funkci.

```
uint8_t stav [2] = {0};
//Proměnná stav uchovává poslední známý stav tlačítek
uint8_t counter = 0;  //Proměnná counter uchovává napočítanou hodnotu

for(;;)              //Nekonečný cyklus
{
  if (!(GPIOA->IDR & GPIO_IDR_0))
  //Pokud je tlačítko BUT_1 stisknuto
  {
            if (stav [0] == 0)
            //Pokud nebylo tlačítko při poslední kontrole stisknuto
        {
                  stav [0] = 1;
                   //Zapsaní do stavu že tlačítko bylo stisknuto
                  ++counter;
                   //Inkrementuj hodnotu proměnné counter

        }

  }
  else                                    //Tlačítko BUT_1 není stisknuto
  {
            stav [0] = 0;
            //Zapsání stavu že tlačítko nebylo stisknuto
  }

  if (!(GPIOC->IDR & GPIO_IDR_1))
  {
        if (stav [1] == 0)
        {
              stav [1] = 1;
              --counter;

        }

  }
  else
  {
        stav [1] = 0;
  }
```

Výše uvedená část kódu sleduje stisknutí tlačítek BUT_1 nebo BUT_2. Po stisku tlačítka BUT_1 se hodnota proměnné counter zvyšuje a po stisku tlačítka BUT_2 se hodnota counter snižuje. Proměnná stav slouží k zapamatování posledního známého stavu tlačítek.

```
if (GPIOA->IDR & GPIO_IDR_6)
//Přepínač SW_1 je v poloze pro binární čítání
  {
        if (counter & 1) //Pokud je nultý bit proměnné counter 1
        {
              GPIOC->BSRR |= GPIO_BSRR_BR_6; //Rozsviť LED_1
        }
        else  //Pokud je nultý bit proměnné counter 0
        {
              GPIOC->BSRR |= GPIO_BSRR_BS_6; //Zhasni LED_1
        }

        if (counter & 2) //Pokud je první bit proměnné counter 1
        {
              GPIOA->BSRR |= GPIO_BSRR_BR_12;
        }
        else
        {
              GPIOA->BSRR |= GPIO_BSRR_BS_12;
        }

        if (counter & 4) //Pokud je druhý bit proměnné counter 1
        {
              GPIOC->BSRR |= GPIO_BSRR_BR_10;
        }
        else
        {
              GPIOC->BSRR |= GPIO_BSRR_BS_10;
        }

        if (counter & 8) //Pokud je třetí bit proměnné counter 1
        {
              GPIOC->BSRR |= GPIO_BSRR_BR_11;
        }
        else
        {
              GPIOC->BSRR |= GPIO_BSRR_BS_11;
        }

        if (counter & 16) //Pokud je čtvrtý bit proměnné counter 1
        {
              GPIOC->BSRR |= GPIO_BSRR_BR_12;
        }
        else
        {
              GPIOC->BSRR |= GPIO_BSRR_BS_12;
        }
  }

  else
  //Přepínač SW_1 je v poloze pro dekadické čítání
  {
        if (counter >= 1) //Pokud je proměnná counter větší nebo rovna 1
        {
```

```
                GPIOC->BSRR |= GPIO_BSRR_BR_6; //Rozsviť LED_1
            }
            else //Pokud není proměnná counter větší nebo rovna 1
            {
                GPIOC->BSRR |= GPIO_BSRR_BS_6; //Zhasni LED_1
            }

            if (counter >= 2)
            {
                GPIOA->BSRR |= GPIO_BSRR_BR_12;
            }
            else
            {
                GPIOA->BSRR |= GPIO_BSRR_BS_12;
            }

            if (counter >= 3)
            {
                GPIOC->BSRR |= GPIO_BSRR_BR_10;
            }
            else
            {
                GPIOC->BSRR |= GPIO_BSRR_BS_10;
            }

            if (counter >= 4)
            {
                GPIOC->BSRR |= GPIO_BSRR_BR_11;
            }
            else
            {
                GPIOC->BSRR |= GPIO_BSRR_BS_11;
            }

            if (counter >= 5)
            {
                GPIOC->BSRR |= GPIO_BSRR_BR_12;
            }
            else
            {
                GPIOC->BSRR |= GPIO_BSRR_BS_12;
            }
        }
    }
```

V druhé části programu se sleduje, v jaké poloze se nachází přepínač a dle toho se následně volí binární, nebo dekadické zobrazení. Vzhledem k tomu, že se jedná pouze o dva stavy, je možné použít pouze podmínku **if**, kdy v hlavní části dochází k zapisování na LED podle hodnoty v proměnné counter. V binárním režimu (stejně jako v minulých úlohách používáme k rozsvícení LED register **BSRR**). A v druhé části, resp. v **else** dochází opět podle hodnoty v proměnné counter k dekadickému zobrazení.

Úloha 4) Vytvořte program rotování LED zleva doprava, jehož směr se otočí stiskem tlačítka

V poslední úloze znovu vytvoříme, již výše zmíněnými způsoby, prostor pro nový program ve funkci **main.**

```
uint8_t counter = 0;
int8_t smer = 1;
uint8_t stav = 0;
uint32_t delicka = 1;
```

Rozdíl mezi **uint8_t** a **int8_t** je ve znaménku, „**u**“ na začátku znamená **unsigned**, se dá přeložit jako bez znaménka. Tzn. že je možné do něj zapisovat pouze kladné hodnoty od 0 do 255. Na rozdíl **int8_t** znaménko mít může, tzn. že je možné do něj zapsat hodnoty od -128 do 127. Číslo 8 znamená, že proměnná má velikost 8 bitů.

```
for (;;)              //Nekonečný cyklus
{
  if (!(GPIOA->IDR & GPIO_IDR_0)) //Pokud je tlačítko BUT_1 stisknuto
  {
            if (stav == 0)
            //Pokud nebylo tlačítko při poslední kontrole stisknuto
        {
                  stav = 1;
              //Zapsaní do stavu že tlačítko bylo stisknuto
              smer *= -1;
              //Obrací směr počítání

        }

  }
  else //Pokud tlačítko není stisknuto
  {
        stav = 0; //Zapsaní do stavu že tlačítko nebylo stisknuto
  }
```

Proměnná delicka nám umožní změnit hodnotu v proměnné counter pouze jednou za 1 500 000 průchodů nekonečným cyklem, díky čemuž můžeme v tomto cyklu účinně kontrolovat i stisk tlačítka. Tato doba odpovídá zhruba 1 sekundě.

```
--delicka; //Dekrementuje 1 od delicky
if (delicka == 0) //Pokud hodnota proměnné delicka došla k 0
{
            delicka = 1500000;
            //Nastavení hodnoty proměnné delicka na původní násobek
      counter += smer;
      //Přičtení hodnoty proměnné smer k hodnotě proměnné counter
      if (counter > 5) //Pokud je hodnota proměnné counter větší než 5
      {
            counter = 1; //Hodnota proměnné counter se nastaví na 1
      }

      if (counter < 1) //Pokud je hodnota proměnné counter menší než 1
      {
```

```
                counter = 5; //Hodnota proměnné counter se nastaví na 5
        }
```

Výše uvedené dvě podmínky zajišťují, že pokud program dojde k poslední LED, začne počítat od začátku.

```
        if (counter == 1) //Pokud je hodnota proměnné counter rovna 1
        {
                GPIOC->BSRR |= GPIO_BSRR_BR_6; //Rozsvítí se LED_1
        }
        else //Pokud hodnota proměnné counter není rovna 1
        {
                GPIOC->BSRR |= GPIO_BSRR_BS_6; //Zhasne se LED_1
        }

        if (counter == 2)
        {
                GPIOA->BSRR |= GPIO_BSRR_BR_12;
        }
        else
        {
                GPIOA->BSRR |= GPIO_BSRR_BS_12;
        }

        if (counter == 3)
        {
                GPIOC->BSRR |= GPIO_BSRR_BR_10;
        }
        else
        {
                GPIOC->BSRR |= GPIO_BSRR_BS_10;
        }

        if (counter == 4)
        {
                GPIOC->BSRR |= GPIO_BSRR_BR_11;
        }
        else
        {
                GPIOC->BSRR |= GPIO_BSRR_BS_11;
        }

        if (counter == 5)
        {
                GPIOC->BSRR |= GPIO_BSRR_BR_12;
        }
        else
        {
                GPIOC->BSRR |= GPIO_BSRR_BS_12;
        }
    }
}
```

Výše uvedená část kódu zajišťuje zobrazování hodnoty v proměnné counter na LED diodách.

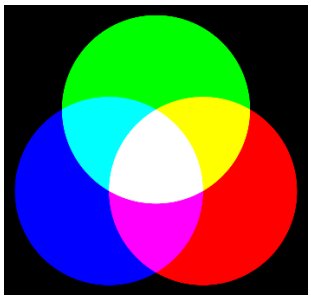
A.2 Laboratorní úloha č. 2

Základní práce s časovači, využití RGB LED řízeného PWM

Teoretický úvod:

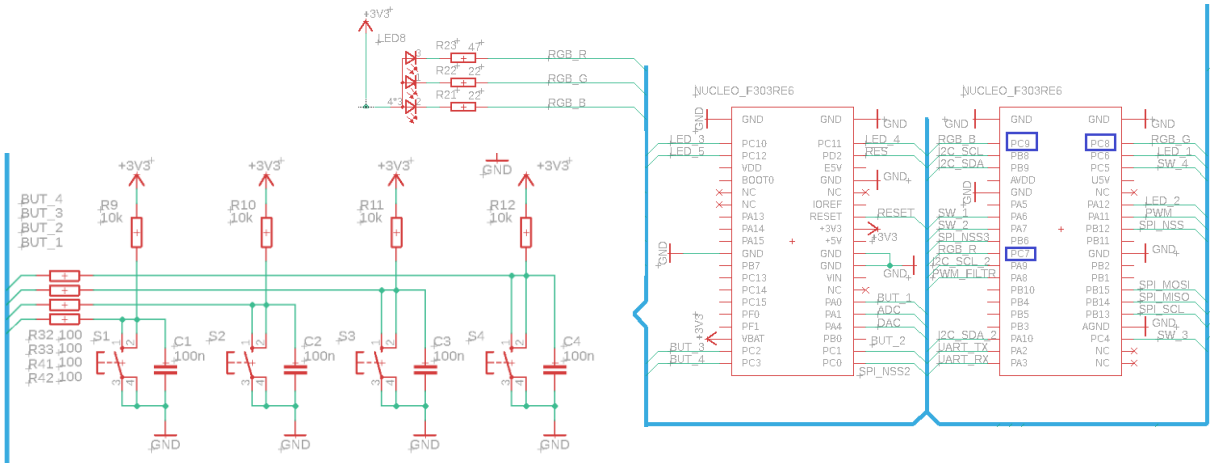
Časovače se skládají z 16-bitového nebo 32-bitového čítače, který se sám opětovně načítá a který má softwarově řízenou předdělička. V rámci této úlohy bude časovač používán ke generování výstupního **PWM** signálu, kterým bude řízena **RGB LED**. Jas **RGB LED** bude regulován poměrem střídy **PWM** signálu. Časovače je také ale možno použít například k měření délky vstupního signálu. V rámci této úlohy bude využit **časovač 3 (TIM3)** a to na výstupních pinech pro **RGB**. Jsou to piny **PC7, PC8, PC9** a k nim kanály 2, 3 a 4. Každý časovač totiž také obsahuje daný počet kanálů. Časovač **TIM3** je 16-bitový.

RGB LED je součástka složená ze tří LED diod a to červené, zelené a modré. Snižováním a zvyšováním intenzity svítivosti jednotlivých LED dochází ke změně výsledné barvy, kterou lze na RGB sledovat. Aditivní míchání barev lze sledovat na přiloženém obrázku.



Obrázek 2. Aditivní michání barev

Schéma zapojení laboratorního přípravu:



Obrázek 3. Schéma zapojení RGB diody a tlačítek

Zadání úloh:

1) Zprovozněte PWM regulaci jasu RGB LED diody, jejíž barvy budou nastavované pomocí prvních tří tlačítek v 15 ti krocích na barvu (stisk tlačítka inkrementuje hodnotu jasu, při přesažení hodnoty 15 je jas resetován na hodnotu 0
2) Pomocí časovače čítejte od 0 do 25 při frekvenci 1 Hz, jehož aktuální počet bude zobrazen binárně na LED diodách

Úloha 1) *Zprovozněte PWM regulaci jasu RGB LED diody, jejíž barvy*

budou nastavované pomocí prvních tří tlačítek v 15 ti krocích na barvu (stisk tlačítka inkrementuje hodnotu jasu, při přesažení hodnoty 15 je jas resetován na počáteční hodnotu 0

Jako první krok je dle dokumentu v Elearningu nutné založit nový projekt v prostředí STM32 CubeIDE. Všechny kroky, nutné k založení projektu, jsou v něm podrobně popsány.

```
#define STM32F303xE                    //Definování typu MCU
```

Definování použitého typu procesoru je nutné kvůli HAL knihovnám.

```
#include "stm32f3xx.h"                 //knihovny pro STM32 řady F
#include "core_cm4.h"                  //core knihovna CMSIS
#include "stm32f303xe.h"               //knihovny STM32f303
```

Na začátku každého programu je potřeba inkludovat knihovny pro použitý mikrokontroler. V programovacím jazyce C direktiva #include říká preprocesoru aby vložil obsah jiného souboru do zdrojového kódu na místě kde je tato direktiva #include napsána.

```
void init (void);                       //Deklarace funkce pro init MCU
void GPIO_init(void);                 //Deklarace funkce pro init GPIO pinů
void TIMER3_init (void);              //Deklarace funkce pro init
TIMER3
```

Před funkcí „**int main** (**void**)“, ve které se budou jednotlivé programy psát, je nutné deklarovat funkce, ve kterých je vytvořeno základní nastavení používaných periferií. Pro úlohy v tomto cvičení jsou to funkce **init** (která vytváří základní nastavení celého mikrokontroleru), funkce **GPIO_init** (která vytváří základní nastavení používaných GPIO pinů) a funkce **TIMER3_init** (která vytváří základní nastavení pro časovač 3 a použité kanály pro RGB diodu).

```
int main(void)
{
  init();                    //zavolání funkce init
  GPIO_init();               //zavolání funkce GPIO_init
  TIMER3_init();             //zavolání funkce TIMER3_init

uint8_t stav [3] = {0};
//Proměnná stav uchovává poslední známý stav tlačítek
TIM3->CCR2 = 0;              //Nastavení DCL na 0% kanál 2
TIM3->CCR3 = 0;              //Nastavení DCL na 0% kanál 3
TIM3->CCR4 = 0;              //Nastavení DCL na 0% kanál 4
TIM3->CCER |= TIM_CCER_CC2E | TIM_CCER_CC3E | TIM_CCER_CC4E;
//zapnutí kanálů 2, 3 a 4
TIM3->CR1 |= TIM_CR1_CEN;          //Povolení čítače
for (;;)
{
  if (!(GPIOA->IDR & GPIO_IDR_0)) //Tlačítko
  {
        if (stav [0] == 0)         //Předchozí stav tlačítka
        {
              stav [0] = 1;
              ++TIM3->CCR2;        //navýšení DCL kanál 2
              if              (TIM3->CCR2              >              15)
```

```
                //Pokud kanál 2 dojde na maximální intenzitu
                {
                        TIM3->CCR2 = 0; //Začne od nuly
                }

        }
    }
    else //Pokud se nestiskne tlačítko
    {
        stav [0] = 0;       //Zapsání posledního stavu tlačítka
    }

    if (!(GPIOC->IDR & GPIO_IDR_1))
    {
        if (stav [1] == 0)
        {
                stav [1] = 1;
                ++TIM3->CCR3;
                if (TIM3->CCR3 > 15)
                {
                        TIM3->CCR3 = 0;
                }

        }

    }
    else
    {
        stav [1] = 0;
    }

    if (!(GPIOC->IDR & GPIO_IDR_2))
    {
        if (stav [2] == 0)
        {
                stav [2] = 1;
                ++TIM3->CCR4;
                if (TIM3->CCR4 > 15)
                {
                        TIM3->CCR4 = 0;
                }

        }

    }
    else
    {
        stav [2] = 0;
    }

}
```

Ve výše uvedeném kódu jsou podmínky, které zajišťují postupné zvyšování intenzity jednotlivých barev RGB diody při stisku příslušných tlačítek.

Úloha 2) *Pomocí časovače čítejte od 0 do 25 při frekvenci 1 Hz, jehož aktuální počet bude zobrazen binárně na LED diodách*

V již vytvořeném projektu můžeme úlohu 1 buď smazat, nebo zakomentovat. Komentář lze vytvořit buď zapsáním „//“ na začátek řádku, nebo „/*“ před první komentovaný znak a „*/“ za poslední komentovaný znak.

V následující úloze vytvoříme funkci **TIM3_IRQHandler**. Tato funkce bude vytvořena mimo funkci **main**. Nastavení časovače 3 provedeme ve funkci **main**.

```
uint8_t glcounter = 0;
//Vytvoření proměnné glcounter s hodnotou 0
void TIM3_IRQHandler (void)
{
      TIM3->SR &= ~((uint32_t)TIM_SR_UIF);
      //Restartování bitu UIF (update interrupt flag)
  ++glcounter;                    //Inkrementace proměnné glcounter
```

K přerušení dochází jednou za sekundu, tudíž v proměnné glcounter počítáme do 25 s frekvenci 1 Hz.

```
  if (glcounter>25)  //Pokud hodnota proměnné glcounter je větší než 25
  {
        glcounter = 0; //Nastavíme hodnotu proměnné glcounter na 0
  }
```

Dále zobrazíme hodnotu glcounter na LED:

```
  if (glcounter & 1)
  {
        GPIOC->BSRR |= GPIO_BSRR_BR_6;
  }
  else
  {
        GPIOC->BSRR |= GPIO_BSRR_BS_6;
  }

  if (glcounter & 2)
  {
        GPIOA->BSRR |= GPIO_BSRR_BR_12;
  }
  else
  {
        GPIOA->BSRR |= GPIO_BSRR_BS_12;
  }

  if (glcounter & 4)
  {
        GPIOC->BSRR |= GPIO_BSRR_BR_10;
  }
  else
  {
        GPIOC->BSRR |= GPIO_BSRR_BS_10;
  }

  if (glcounter & 8)
```

```
  {
        GPIOC->BSRR |= GPIO_BSRR_BR_11;
  }
  else
  {
        GPIOC->BSRR |= GPIO_BSRR_BS_11;
  }

  if (glcounter & 16)
  {
        GPIOC->BSRR |= GPIO_BSRR_BR_12;
  }
  else
  {
        GPIOC->BSRR |= GPIO_BSRR_BS_12;
  }
}
```

V následující části kódu ve funkci main nastavíme časovač 3.

```
int main(void)
{
  init();                   //Zavolání funkce init
  GPIO_init();              //Zavolání funkce GPIO_init
  TIMER3_init();            //Zavolání funkce TIMER3_init

#define MNVIC_BASE    0xe000e100
//Definice kořenové adresy registru NVIC
typedef struct              //Definice typu pro registr NVIC
{
  uint32_t ISER0;           //0x0
  //Definice pro registr NVIC_ISER0, ostatní registry nepotřebujeme
} MNVIC_typedef;
#define MNVIC ((MNVIC_typedef *) MNVIC_BASE)    //Definice registru NVIC

  MNVIC->ISER0 |= (1<<29);  //povolení přerušení 29 -> TIM3 Interrupt
  TIM3->PSC = 7200-1;       //Nastavení předděličky
  TIM3->ARR = 10000-1;      //Nastavení maxima čítače pro TIM3
  TIM3->DIER |= TIM_DIER_UIE; //Povolení přerušení při přetečení čítače
  TIM3->SR &= ~((uint32_t)TIM_SR_UIF); //Resetování bitu UIF
  TIM3->CR1 |= TIM_CR1_CEN; //Povolení časovače

            for (;;);

}
```

Nastavení předděličky vychází z hodinového signálu procesoru pro časovač 3, který je 72MHz. Předdělička se nastaví na frekvenci 7,2 kHz. Registerem ARR dojde k nastavení horní hranice čítaní v časovači 3.

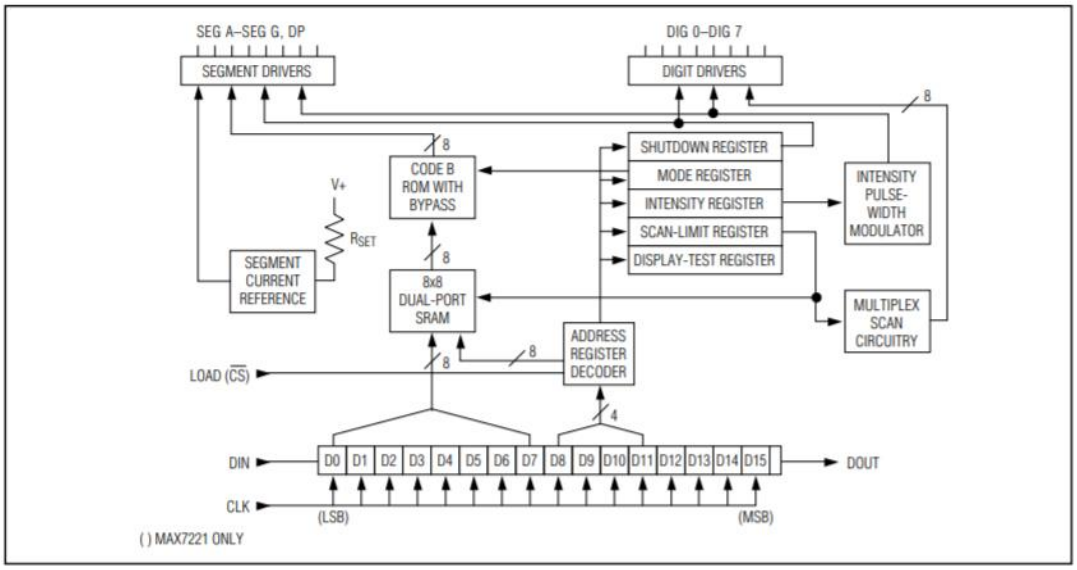
$$Frekvence\ přerušení = \frac{PCLK1 * 2}{(PSC + 1) * (ARR + 1)} = \frac{72\ MHz}{7200 * 10000} = 1\ Hz$$

A.3 Laboratorní úloha č. 3

Základní práce s SPI sběrnicí a řadičem MAX7219CNG

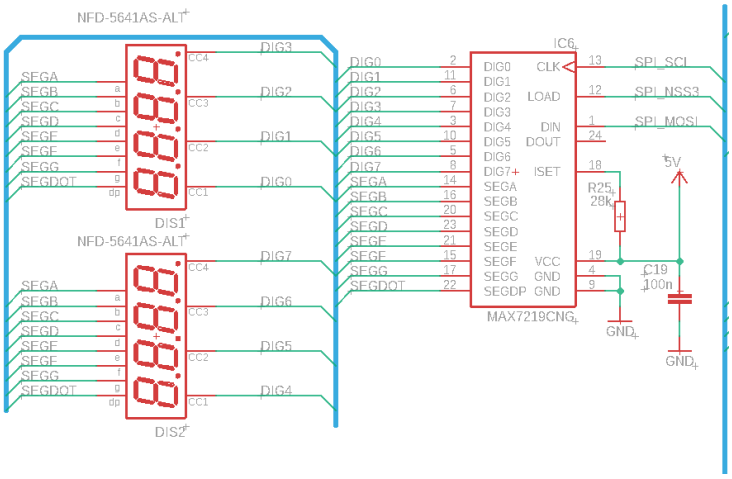
Teoretický úvod:

Sběrnice SPI je sériová sběrnice s volitelným počtem Slave zařízení. Tři hlavní vodiče se dělí na MOSI (Master output Slave input), MISO (Master input Slave output) a SCK (hodinový signál). Slave zařízení volíme pomocí vodiče SS nebo NSS. Zkratka SS znamená Slave select, písmeno N pak značí negative, při obrácené logice. V rámci této úlohy bude použito SPI2, které je připojeno na pinech PB13 (SCK), PB14 (MISO), PB15 (MOSI) a pro řízení Slave zařízení, které v této úloze tvoří řadič MAX7219CNG pin PB6 (NSS3). Protože řadič jako Slave zařízení neodpovídá a chová se jen jako výstup, budeme sběrnice SPI využívat v Master režimu a využívat tak sběrnice pouze k odesílání dat na příslušnou adresu v řadiči. Data budeme do řadiče posílat v 8-bitové velikosti. Řadič v rámci zapojení na laboratorní DPS ovládá 8 sedmi segmentů. Řadič je využit aby obsloužil multiplexování displejů, a tím nám ušetřil práci a výkon procesoru. Blokový diagram vnitřního zapojení řadiče je možné sledovat na obrázku č.1.



Obrázek 4. Blokové schéma řadiče MAX7219CNG

Schéma zapojení laboratorního přípravu:



Obrázek 5. Zapojení řadiče MAX7219XNG na laboratorním přípravku

Zadání úloh:

1) Pomocí funkce SPI_MasterTransmit posílejte adresu a data na řadič 7segmentového zobrazovače MAX7219CNG, zobrazte na patřičných displejích čísla 0-7 dle patřičného pořadí displeje
2) Za pomocí čítače a SPI zobrazte na 7segmentových zobrazovačích jednoduchého hada, tedy postupně rozsvěcející se segmenty tvořící kruh. Spínačem SW1 měňte směr „rotace“ hada a spínačem SW2 měňte způsob:
   - Poloha 1 – rotace na samostatném jednom 7segmentu
   - Poloha 2 – rotace přes všechny 7segmenty

Úloha 1) Pomocí funkce SPI_MasterTransmit posílejte adresu a data na řadič 7segmentového zobrazovače MAX7219CNG, zobrazte na patřičných displejích čísla 0-7 dle patřičného pořadí displeje

Jako první krok je dle dokumentu v Elearningu nutné založit nový projekt v prostředí STM32 CubeIDE. Všechny kroky, nutné k založení projektu, jsou v něm podrobně popsány.

```
#define STM32F303xE                          //Definování typu MCU
```

Definování použitého typu procesoru je nutné kvůli HAL knihovnám.

```
#include "stm32f3xx.h"                       //knihovny pro STM32 řady F
#include "core_cm4.h"                        //core knihovna CMSIS
#include "stm32f303xe.h"                     //knihovny STM32f303
```

Na začátku každého programu je potřeba inkludovat knihovny pro použitý mikrokontroler. V programovacím jazyce C direktiva #include říká preprocesoru aby vložil obsah jiného souboru do zdrojového kódu na místě kde je tato direktiva #include napsána.

```
void init (void);                            //Deklarace funkce pro init MCU
void SPI_init(void);
//Deklarace funkce pro init SPI sběrnice
void SPI_MasterTransmit (uint8_t Command, uint8_t Data);
//Deklarace funkce pro odesílání adresy a dat přes SPI sběrnici
void MAX_driver(void);
//Deklarace funkce pro init řadiče
```

Před funkcí „**int main (void)**“, ve které se budou jednotlivé programy psát, je nutné deklarovat funkce, ve kterých je vytvořeno základní nastavení používaných periferií. Pro úlohy v tomto cvičení jsou to funkce **init** (která vytváří základní nastavení celého mikrokontroléru), funkce **SPI_init** (která vytváří základní init pro SPI piny a v níž se nachází pro tyto piny i GPIO init), funkce **SPI_MasterTransmit** (která zajišťuje odesílání adresy a dat) a funkci **MAX_driver** (která vytváří základní nastavení řadiče pro realizaci požadovaných úloh)

```
int main(void)
{
  init();                            //Zavolání funkce init
  SPI_init();                        //Zavolání funkce SPI_init
  MAX_driver();                      //Zavolání funkce nastavení pro řadič

  GPIOB->BSRR = GPIO_BSRR_BR_6;      //Zapnutí Slave zařízení na pinu PB6
  SPI_MasterTransmit(0x01, 0x00);    //Odeslání adresy a dat do řadiče
  GPIOB->BSRR = GPIO_BSRR_BS_6;      //Vypnutí Slave zařízení na pinu PB6
  GPIOB->BSRR = GPIO_BSRR_BR_6;
  SPI_MasterTransmit(0x02, 0x01);
  GPIOB->BSRR = GPIO_BSRR_BS_6;
  GPIOB->BSRR = GPIO_BSRR_BR_6;
  SPI_MasterTransmit(0x03, 0x02);
  GPIOB->BSRR = GPIO_BSRR_BS_6;
```

```
GPIOB->BSRR = GPIO_BSRR_BR_6;
SPI_MasterTransmit(0x04, 0x03);
GPIOB->BSRR = GPIO_BSRR_BS_6;
GPIOB->BSRR = GPIO_BSRR_BR_6;
SPI_MasterTransmit(0x05, 0x04);
GPIOB->BSRR = GPIO_BSRR_BS_6;
GPIOB->BSRR = GPIO_BSRR_BR_6;
SPI_MasterTransmit(0x06, 0x05);
GPIOB->BSRR = GPIO_BSRR_BS_6;
GPIOB->BSRR = GPIO_BSRR_BR_6;
SPI_MasterTransmit(0x07, 0x06);
GPIOB->BSRR = GPIO_BSRR_BS_6;
GPIOB->BSRR = GPIO_BSRR_BR_6;
SPI_MasterTransmit(0x08, 0x07);
GPIOB->BSRR = GPIO_BSRR_BS_6;
```

V hlavním funkci **main** je potřeba zavolat **init**, **SPI_init** a **SPI_MasterTransmit**. Následně potom vytvořit program, který rozsvítí jednotlivá čísla na segmentech. K tomu je potřeba zapsat do registru **BSRR** na bit **BR**. Bit **BR** slouží jako reset, tzn. že nastaví na svoji pozici v registru logickou 0. Vzhledem k tomu, že je však řadič zapojen jako **NSS**, to znamená s negativní logikou, otočí se tím význam bitu **BR**, tzn. bitem **BR** aktivujeme Slave zařízení. Funkcí **SPI_MasterTransmit** dochází k odeslání adresy a následně i dat do řadiče. Dle registrové mapy adres v katalogového listu při zapsání na adresu 0x01 volíme odesílání dat na nultý digit. Druhá pozice ve funkci **SPI_MasterTransmit** udává odesílaná data. Odeslaná data se na displeji zobrazí jako znaky, tzn. že po odeslání dat 0x00 se na displeji zobrazí 0, resp. rozsvítí se segmenty a, b, c, d, e, f. Po zapsání je nutné restartovat pin **PB6**, tudíž se do registru **BSRR** zapíše bit **BS**, který zruší výběr Slave zařízení

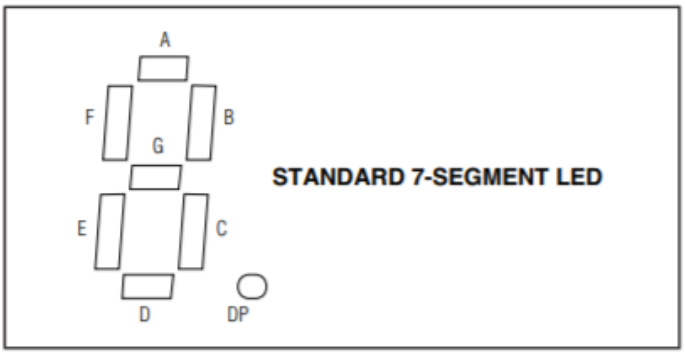
Úloha 2) *Za pomocí čítače a SPI zobrazte na 7segmentových zobrazovačích jednoduchého hada, tedy postupně rozsvěcející se segmenty tvořící kruh. Spínačem SW1 měňte směr „rotace“ hada a spínačem SW2 měňte způsob:*

- *Poloha 1 – rotace na samostatném jednom 7segmentu*
- *Poloha 2 – rotace přes všechny 7segmenty*

V již vytvořeném projektu můžeme úlohu 1 buď smazat, nebo zakomentovat. Komentář lze vytvořit buď zapsáním „//“ na začátek řádku, nebo „/*“ před první komentovaný znak a „*/“ za poslední komentovaný znak.

V následující úloze zachováme tedy vše, kromě původního programu v **main** funkci.

V rámci nastavení řadiče MAX7219CNG lze zapsáním do patřičného registru zvolit mód dekódování. Díky tomu je možné ovládat jednotlivé segmenty.



	REGISTER DATA							
	D7	D6	D5	D4	D3	D2	D1	D0
Corresponding Segment Line	DP	A	B	C	D	E	F	G

Obrázek 6. Data bity a odpovídající segmenty v režimu bez dekodéru

```
int main(void)
{
#define SEG_A 64
#define SEG_B 32
#define SEG_C 16
#define SEG_D 8
#define SEG_E 4
#define SEG_F 2
#define SEG_G 1

uint8_t seg[2][20] =   {{SEG_A, SEG_A, SEG_A, SEG_A, SEG_A, SEG_A, SEG_A,
                    SEG_A, SEG_B, SEG_C, SEG_D, SEG_D, SEG_D, SEG_D,
                    SEG_D, SEG_D, SEG_D, SEG_D, SEG_E, SEG_F}, {SEG_A,
                    SEG_B, SEG_C,SEG_D, SEG_E, SEG_F}};
//Proměnná seg uchovává informaci o tom, které segmenty mají být
  rozsvíceny v závislosti na poloze přepínače a na čase
```

```
uint8_t digit[2][20] = {{1, 2, 3, 4, 5, 6, 7, 8, 8, 8, 8, 7, 6, 5, 4, 3,
                         2, 1, 1, 1}, {1, 1, 1, 1, 1, 1}};
//Proměnná digit uchovává informaci o tom, na kterých digitech se mají
  výše uvedené segmenty rozsvítit v závislosti na poloze přepínače a
  čase
uint8_t funcPointer = 0;                 //Uchovávání stavu přepínače
uint8_t upperLimit[2] = {19, 5};
 //Proměnná upperLimit udává délky sekvencí v závislosti na poloze
  přepínače

int8_t digitPointer = 0;
//Uchovává informaci o tom, který digit právě svítí
int8_t dir = 1;            //Uchovává informaci o požadovaném směru
GPIOB->BSRR = GPIO_BSRR_BR_6;  //Zapnutí Slave zařízení na pinu PB6
SPI_MasterTransmit (0x09, 0);  //Nastavení na nedekódovací režim
GPIOB->BSRR = GPIO_BSRR_BS_6;  //Vypnutí Slave zařízení na pinu PB6
for (;;)
{
  if (GPIOA->IDR & GPIO_IDR_6)    //Přepínač
  {
        dir = 1;
        //Směr efektu po směru hodinových ručiček
  }
  else
  {
        dir = -1;
        //Směr efektu proti směru hodinových ručiček
  }

  if (GPIOA->IDR & GPIO_IDR_7)    //Přepínač
  {
        funcPointer = 1;          //Efekt hada na jednom digitu
  }
  else
  {
        funcPointer = 0;          //Efekt hada přes všechny digity
  }

      uint8_t lastPointer = digitPointer;
        //Vytvoření proměnné lastPointer pro uchování posledního
        zapsaného digitu, abysme jej následně mohli vypnout
  digitPointer += dir;
  //Přičte, nebo odečte 1 v závislosti na přepínači
```

V níže uvedených podmínkách dochází k omezování délky sekvencí podle proměnné upperLimit v závislosti na přepínači a také opakování efektů, pokud dojdeme k jejich konci.

```
  if (digitPointer > upperLimit [funcPointer])
  {
        digitPointer = 0;
  }
  if (digitPointer < 0)
  {
        digitPointer = upperLimit [funcPointer];
  }
```

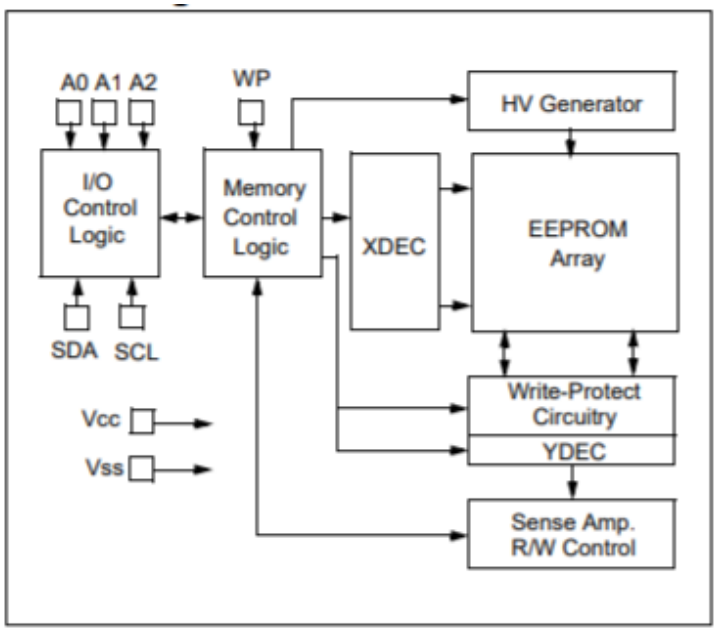
```
        GPIOB->BSRR = GPIO_BSRR_BR_6;
        SPI_MasterTransmit (digit [funcPointer] [lastPointer], 0);
        //Nulování předchozího digitu
        GPIOB->BSRR = GPIO_BSRR_BS_6;
        GPIOB->BSRR = GPIO_BSRR_BR_6;
        SPI_MasterTransmit (digit [funcPointer] [digitPointer],
                            seg[funcPointer] [digitPointer]);
        //Rozsvícení nového digitu
        GPIOB->BSRR = GPIO_BSRR_BS_6;
        HAL_Delay(400);                         //Zpoždění
    }
}
```

A.4 Laboratorní úloha č. 4

Základní práce se sběrnicí I²C, pamětí EEPROM a OLED displejem

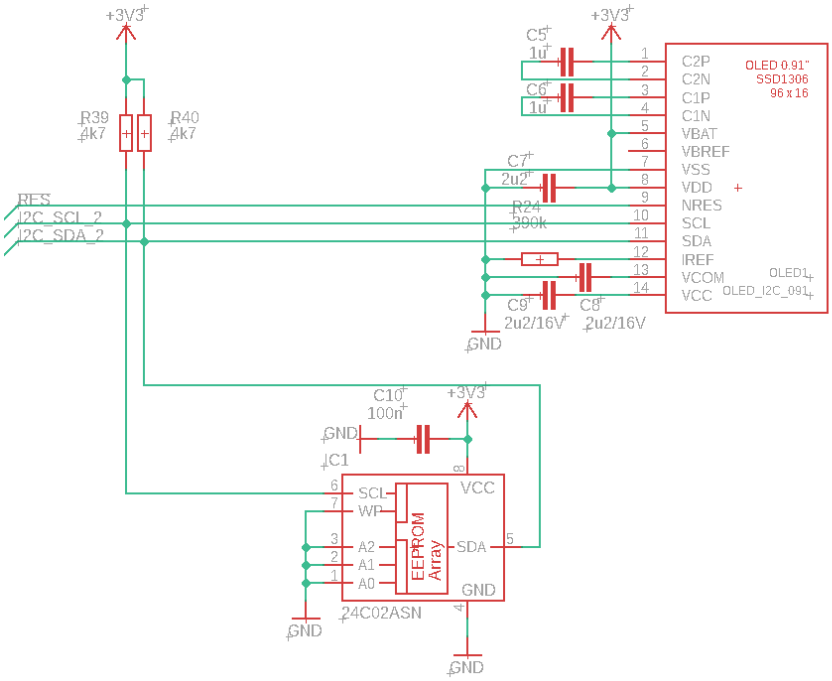
Teoretický úvod:

I²C je sériová sběrnice, která komunikuje přes dva vodiče, a to datový SDA, skrz který se posílají data a tzv. hodinový SCL, který udává hodinový signál. Rychlost sběrnice je závislá na pull-up rezistorech. Nejpoužívanější velikosti pull-up rezistorů jsou 1 kΩ, 4,7 kΩ a 10 kΩ. Velikost 1 kΩ se používá pro rychlost 400kHz a více při napětí 3,3 V, nebo pro méně než 400kHz při napětí 1,8 V. Velikost 4,7 kΩ udává rychlost sběrnice 400kHz při napětí 5 V, nebo pro rychlost nižší než 400kHz při napětí 3,3V. Velikost odporu 10 kΩ se pak používá pro dosažení rychlosti menší než 100kHz při napětí 3,3 V nebo 5 V. V laboratorním přípravku jsou na sběrnici I²C, která bude sloužit k přenášení dat u EEPROM paměti a OLED displeje použity pull-upy o velikosti 4,7 kΩ. Paměť EEPROM navržená na laboratorním přípravku je organizovaná jako jednoduchý blok o 256x8-bitů. Je možné k ní přistupovat po sériové lince, v případě této úlohy přes sběrnici I²C. Zapisovat je možné rychlostí 100 kHz, nebo 400 kHz. OLED displej je taktéž připojen na sběrnici I²C a bodově se skládá z 128 sloupců a 32 řádků.



Obrázek 7. Blokové schéma paměti EEPROM

Schéma zapojení laboratorního přípravu:



Obrázek 8. Schéma zapojení paměti EEPROM a OLED displeje na laboratorním přípravku

Zadání úloh:

1) Zprovozněte komunikaci pomocí I2C rozhraní a pro ověření její funkce vytvořte následující program:
   a) Přepínače SW1-SW4 slouží jako binární zadání hodnoty, která se zobrazuje v reálném čase na 7segmentovém zobrazovači
   b) Tlačítkem TL1 aktuální hodnotu zapište do paměti 24C02 připojené na I2C periferii
   c) Tlačítkem TL2 vyčtete zapsanou hodnotu z paměti 24C02 a zobrazíte ji na posledním 7segmentovém zobrazovači
   d) Po vypnutí napájení a posunutí spínačů SW1-SW4 do jiné polohy musí být přečtená hodnota z paměti stejná jako před resetem napájení
   e) Operace musí být opakovatelná (přepis čísla v paměti)
2) Zprovozněte OLED zobrazovač připojený na I2C rozhraní a zobrazte nápis „Hello world“

Úloha 1) *Zprovozněte komunikaci pomocí I2C rozhraní a pro ověření její funkce dle zadání uvedené výše.*

V rámci této úlohy je nutné přidat si do projektu knihovny, které naleznete na Elearningu a vložit je do příslušné složky ve vašem projektu. Jsou to soubory I2C_Lib.c tyto soubory vložte do složky Src ve vašem projektu. A dále soubory I2C_Lib.h tyto soubory vložte do složky Inc ve vašem projektu. Váš projekt v prostředí CubeIDE obnovte přes file -> refresh.

```
#define STM32F303xE                 //Definování typu MCU
```

Definování použitého typu procesoru je nutné kvůli HAL knihovnám.

```
#include "stm32f3xx.h"              //knihovny pro STM32 řady F
#include "core_cm4.h"               //core knihovna CMSIS
#include "stm32f303xe.h"            //knihovny STM32f303
#include "I2C_Lib.h"                //knihovna pro I2C
```

Na začátku každého programu je potřeba inkludovat knihovny pro použitý mikrokontroler. V programovacím jazyce C direktiva #include říká preprocesoru aby vložil obsah jiného souboru do zdrojového kódu na místě kde je tato direktiva #include napsána.

```
void init (void);           //deklarace funkce
void SPI_init(void);
void GPIO_init(void);
void SPI_MasterTransmit (uint8_t Command, uint8_t Data);
void MAX_driver(void);

int main(void)
{
  init();                   //zavolání knihovny
  GPIO_init();
  SPI_init();
  MAX_driver();
  I2CInit(I2C2);

  uint8_t InValue = 0;              //Vytvoření proměnné
  uint8_t OutValue = 0;
  uint8_t dekoder [16] = {0x7E, 0x30, 0x6D, 0x79, 0x33, 0x5B, 0x5F,
                          0x70, 0x7F, 0x7B, 0x77, 0x1F, 0x4E, 0x3D, 0x4F,
                          0x47};
```

Výše uvedená proměnná dekoder simuluje funkci dekodéru v procesoru z důvodu, že řadič MAX7219CNG převádí binární hodnoty vyšší než 9 na nechtěné číslice zobrazené na sedmi segmentovém displeji. Pomocí uměle vytvořeného dekodéru je možné zobrazit číslice 0 až 9 a písmena A až F.

```
  GPIOB->BSRR = GPIO_BSRR_BR_6;
  SPI_MasterTransmit(0x09, 0);      //Vypnutí interního dekodéru řadiče
  GPIOB->BSRR = GPIO_BSRR_BS_6;
  for(uint8_t i = 1; i <= 8; ++i)
  //Tento cyklus zhasíná všechny displeje
  {
        GPIOB->BSRR = GPIO_BSRR_BR_6;
```

```
        SPI_MasterTransmit(i, 0);
        GPIOB->BSRR = GPIO_BSRR_BS_6;
}
while (1)
{
        InValue = 0;


        if(!(GPIOA->IDR & GPIO_IDR_6))  //Detekce stavu přepínače
        {
                InValue += 1;
        }
        if(!(GPIOA->IDR & GPIO_IDR_7))
        {
                InValue += 2;
        }
        if(!(GPIOC->IDR & GPIO_IDR_4))
        {
                InValue += 4;
        }
        if(!(GPIOC->IDR & GPIO_IDR_5))
        {
                InValue += 8;
        }
        GPIOB->BSRR = GPIO_BSRR_BR_6;
        SPI_MasterTransmit(0x01, dekoder [InValue]);
    //Zápis binární hodnoty v závislosti na stavu přepínačů na digit 1
        GPIOB->BSRR = GPIO_BSRR_BS_6;

        if(!(GPIOA->IDR & GPIO_IDR_0)) //Detekce stisku tlačítka
        {
                I2CMasterTransmit(I2C2, 0xA0, &InValue, 1);
                //Zapsání hodnoty do paměti
                while(!(GPIOA->IDR & GPIO_IDR_0));
        }
        if(!(GPIOC->IDR & GPIO_IDR_1)) //Detekce stisku tlačítka
        {
                I2CMasterReceive(I2C2, 0xA0, &OutValue, 1);
                //Načtení hodnoty z paměti
                GPIOB->BSRR = GPIO_BSRR_BR_6;
                SPI_MasterTransmit(0x08, dekoder [OutValue]);
                //Zobrazení hodnoty na sedmi segmentu
                GPIOB->BSRR = GPIO_BSRR_BS_6;
                while(!(GPIOC->IDR & GPIO_IDR_1));
        }
}
```

Úloha 2) *Zprovozněte OLED zobrazovač připojený na I2C rozhraní a zobrazte nápis „Hello world“*

V rámci této úlohy je nutné přidat si do projektu knihovny, které naleznete na Elearningu a vložit je do příslušné složky ve vašem projektu. Jsou to soubory fonts.c, I2C_Lib.c, SSD1306.c, tyto soubory vložte do složky Src ve vašem projektu. A dále soubory fonts.h, I2C_Lib.h, SSD1306.h, tyto soubory vložte do složky Inc ve vašem projektu. Váš projekt v prostředí CubeIDE obnovte přes file -> refresh.

```
#define STM32F303xE                 //Definování typu MCU
```

Definování použitého typu procesoru je nutné kvůli HAL knihovnám.

```
#include "stm32f3xx.h"              //knihovny pro STM32 řady F
#include "core_cm4.h"               //core knihovna CMSIS
#include "stm32f303xe.h"            //knihovny STM32f303
#include "I2C_Lib.h"                //knihovna pro I2C
#include "SSD1306.h"                //knihovna pro OLED displej
#include "fonts.h"                  //knihovna fontů
```

Na začátku každého programu je potřeba inkludovat knihovny pro použitý mikrokontroler. V programovacím jazyce C direktiva #include říká preprocesoru, aby vložil obsah jiného souboru do zdrojového kódu na místě kde je tato direktiva #include napsána.

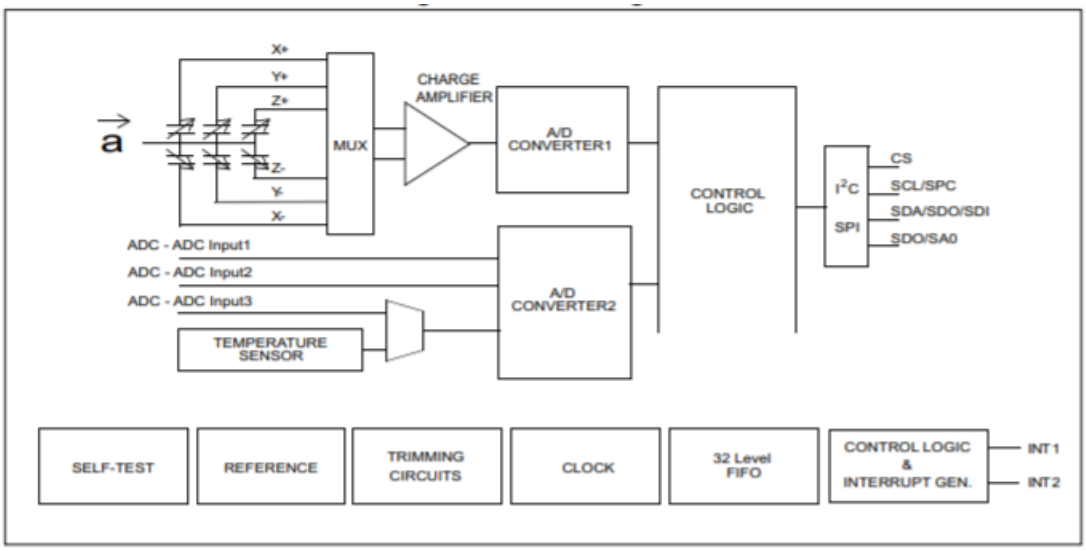
```
void init (void);                   //Deklarace funkce pro init MCU
int main(void)
{
  init();                           //Zavolání funkce init
  I2CInit(I2C2);                    //Zavolání funkce I2CInit


  ssd1306_Init();                   //Zavolání funkce pro init dipleje
  ssd1306_SetCursor(0,0);           //Nastavení kurzoru na pozici 0,0
  char* text = "Hello world";       //Vytvoření ukazatele na char s textem
  ssd1306_WriteString(text, Font_7x10, White);
  //Vypsání řetězce z proměnné text, výběr fontu a barvy
  ssd1306_UpdateScreen();           //Přenesení dat do displeje
  for(;;);
}
```

A.5 Laboratorní úloha č. 5

Základní práce s MEMS akcelerometrem LIS3DH s komunikací přes SPI sběrnici

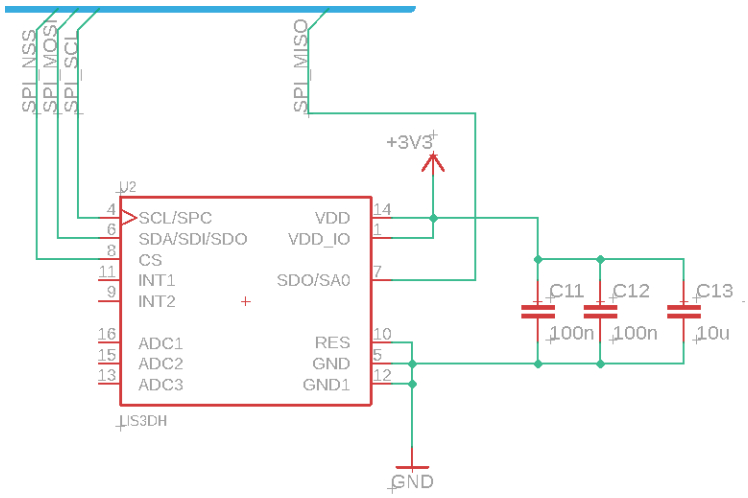
Teoretický úvod:

Zkratka MEMS znamená MicroElectroMechanical system, což ve volném překladu znamená mikroelektromechanická součástka. Akcelerometr je zařízení, které měří zrychlení ve třech osách X, Y a Z. V dnešní době mají široké spektrum využití v nejrůznějších oblastech, a to od automobilového průmyslu, až po průmysl spotřební elektroniky, jako například různá herní zařízení, nebo telefony. Součástí MEMS senzoru LIS3DH je kromě akcelerometru také gyroskop a část obvodu pro měření teploty. MEMS senzor, který je použit v zapojení na laboratorním přípravku (LIS3DH), zahrnuje akcelerometr i gyroskop. Akcelerometr je lineární, tří osy a umožňuje vzorkovat data s frekvencí až 5,3 kHz. Pro každou osu má integrovanou 32-bitovou FIFO paměť. Je možné s ním komunikovat pomocí sběrnice I^2C nebo SPI. V rámci této laboratorní úlohy bude pro komunikaci použita sběrnice SPI.



Obrázek 9. Blokový diagram vnitřního zapojení součástky LIS3DH

Schéma zapojení laboratorního přípravu:



Obrázek 10. Schéma zapojení součástky LIS3DH v laboratorním přípravku

Zadání úloh:

1) Zprovozněte akcelerometr a na OLED displeji zobrazte aktuální hodnoty v osách „X, Y a Z“ jako desetinná čísla.
2) Na OLED display zobrazte jednoduchý graf ukazující vychýlení v ose X nebo Y nebo Z

Úloha 1) *Zprovozněte akcelerometr a na OLED displeji zobrazte aktuální hodnoty v osách „X, Y a Z“ jako desetinná čísla.*

V rámci této úlohy je nutné přidat si do projektu knihovny, které naleznete na Elearningu a vložit je do příslušné složky ve vašem projektu. Jsou to soubory fonts.c, I2C_Lib.c, SSD1306.c, SPI_Lib.c, LIS3DH_Lib.c tyto soubory vložte do složky Src ve vašem projektu. A dále soubory fonts.h, I2C_Lib.h, SSD1306.h, SPI_Lib.h, LIS3DH_Lib.h tyto soubory vložte do složky Inc ve vašem projektu. Váš projekt v prostředí CubeIDE obnovte přes file -> refresh.

```
#define STM32F303xE                 //Definování typu MCU
```

Definování použitého typu procesoru je nutné kvůli HAL knihovnám.

```
#include "stm32f3xx.h"              //knihovny pro STM32 řady F
#include "core_cm4.h"               //core knihovna CMSIS
#include "stm32f303xe.h"            //knihovny STM32f303
#include "SSD1306.h"                //knihovna pro OLED displej
#include "fonts.h"                  //knihovna fontů
#include "SPI_Lib.h"                //knihovna SPI
#include "math.h“                   //knihovna matematiky
#include "LIS3DH_Lib.h“             //knihovna LIS3DH
```

Na začátku každého programu je potřeba inkludovat knihovny pro použitý mikrokontroler. V programovacím jazyce C direktiva #include říká preprocesoru, aby vložil obsah jiného souboru do zdrojového kódu na místě, kde je tato direktiva #include napsána.

```
int main(void)
{
  init();                   // Zavolání funkce init
  GPIO_init();              // Zavolání funkce GPIO_init
  SPIInit();                // Zavolání funkce SPIInit
  ssd1306_Init();           // Zavolání funkce ssd1306_Init
  uint8_t s = LIS3DH_Init();
  //Zavolání funkce LIS3DH_Init s navracenou hodnotou

float X,Y,Z;
// Vytvoření proměnný X,Y,Z pro data z akcelerometru
char text [20];       // Vytvoření pole o 20 znacích
for(;;)
{
  ssd1306_Fill(Black);      //Nastavení barvy OLED displeje na černo
  LIS3DH_ReadXYZ(&X, &Y, &Z);     //Čte hodnoty z akcelerometru
  sprintf(text, "x:%.2f y:%.2f", X, Y);
  //Vypíše hodnoty z akcelerometru do pole text
  ssd1306_SetCursor(0,0);         //Nastavení kurzoru na pozici 0,0
  ssd1306_WriteString(text, Font_7x10, White);
  //Zapíše řetězec na displej
  sprintf(text, "z:%.2f", Z);
  //Vypíše hodnoty z akcelerometru do pole text
  ssd1306_SetCursor(0,11); //Nastavení kurzoru na pozici 0,11
  ssd1306_WriteString(text, Font_7x10, White);
  //Zapíše řetězec na displej
  ssd1306_UpdateScreen();
  //Přenesení dat do displeje
}
}
```

Úloha 2) *Na OLED display zobrazte jednoduchý graf ukazující vychýlení v ose X nebo Y nebo Z*

V již vytvořeném projektu můžeme úlohu 1 buď smazat, nebo zakomentovat. Komentář lze vytvořit buď zapsáním „//“ na začátek řádku, nebo „/*“ před první komentovaný znak a „*/“ za poslední komentovaný znak.

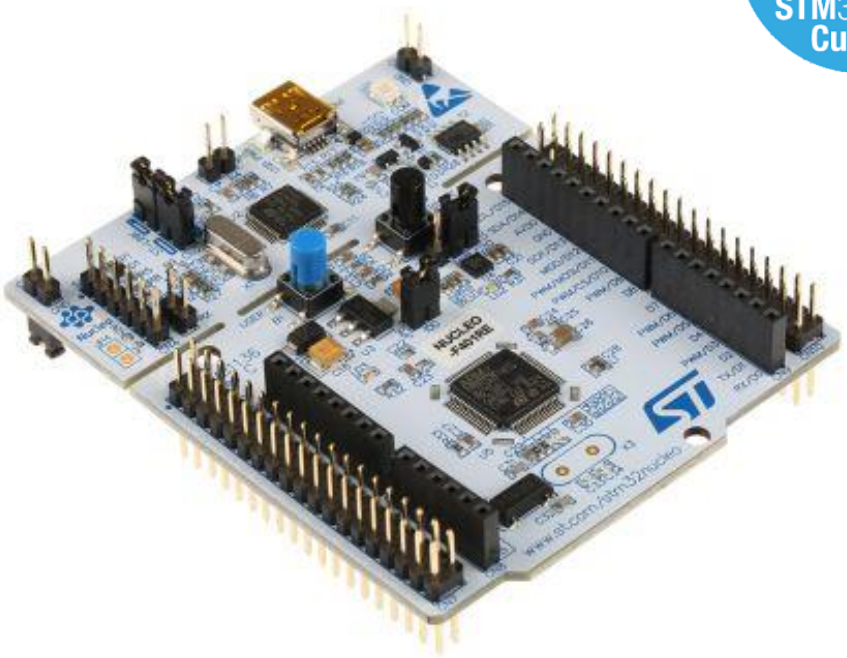
V následující úloze zachováme tedy vše, kromě původního programu v **main** funkci.

```
int main(void)
{
  init();                   // Zavolání funkce init
  GPIO_init();              // Zavolání funkce GPIO_init
  SPIInit();                // Zavolání funkce SPIInit
  ssd1306_Init();           // Zavolání funkce ssd1306_Init
      uint8_t s = LIS3DH_Init();
      //Zavolání funkce LIS3DH_Init s navracenou hodnotou
float X,Y,Z;
//Vytvoření proměnný X,Y,Z pro data z akcelerometru
#define sloupce 128               // Definice počtu sloupců
int8_t hodnoty [sloupce] = {0};
//Proměnná hodnoty pole uchovávající hodnoty grafu
#define radky 32;                 // Definice počtu řádků
for(;;)
{
  ssd1306_Fill(Black);         //Nastavení barvy OLED displeje na černo
  LIS3DH_ReadXYZ(&X, &Y, &Z);     //Čte hodnoty z akcelerometru
  for(uint8_t i = sloupce-2; i>=0; --i)
  //Posunutí hodnot v poli hodnoty aby se vytvořilo místo pro nový údaj
  {
        hodnoty [i+1] = hodnoty [i];
  }
  hodnoty [0] = (int8_t)math.round(X);   //Zapsání nového údaje
  for(uint8_t i = 0; i<sloupce; ++i)     //Vykreslování celého grafu
  {
        if(hodnoty [i]>0)         //Vykreslování jediného sloupce nad 0
        {
              for(int8_t j = 0; j>-hodnoty[i]; --j)
              //Vykreslí nad sebou tolik pixelů, jaké je číslo hodnoty
              //pro daný sloupec
              {
                    ssd1306_DrawPixel (i, 15+j, White);
              }
        }
        else if(hodnoty [i]<0) // Vykreslování jediného sloupce pod 0
        {
              for(int8_t j = 0; j<-hodnoty[i]; ++j)
              //Vykreslí pod sebou tolik pixelů, jaké je číslo hodnoty
              //pro daný sloupec
              {
                    ssd1306_DrawPixel (i, 16+j, White);
              }
        }
  }
  ssd1306_UpdateScreen();         //Přenesení dat do displeje
  HAL_Delay(50);                  //Zpoždění
}
```

B - LABORATORNÍ MANUÁLY

B.1 Manuál pro práci v prostředí STM32 CubeIDE

Manuál pro práci v prostřední STM32 CubeIDE



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

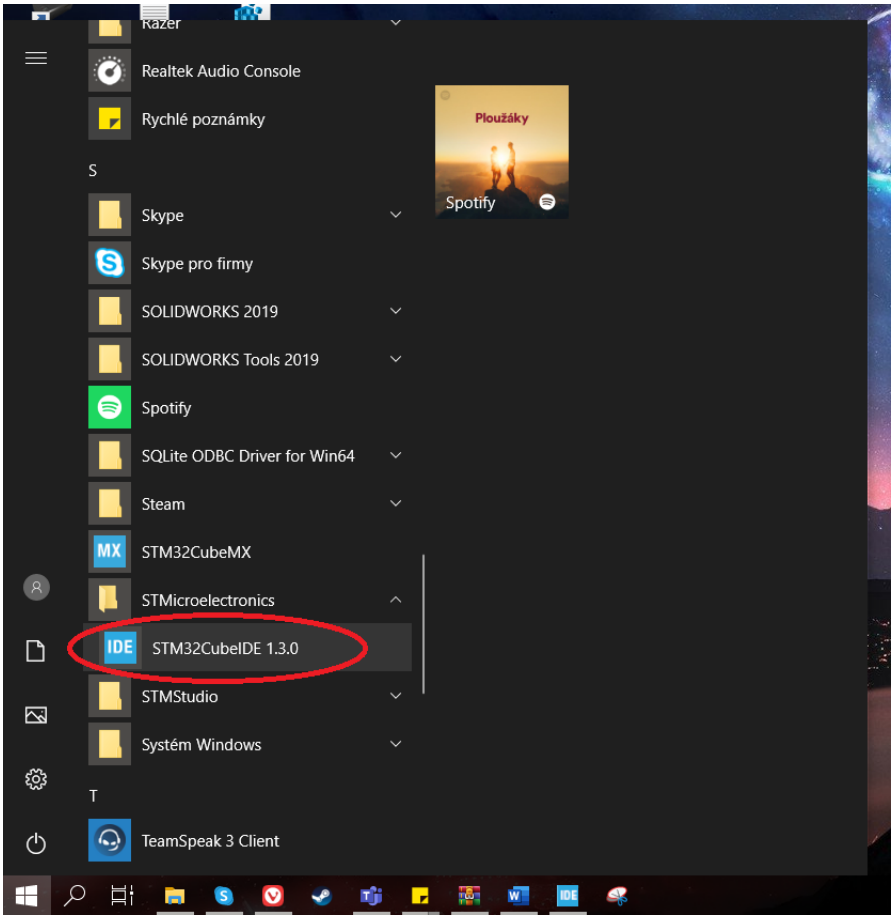
Autor: Matěj Očenášek

Obsah

Seznam Obrázků

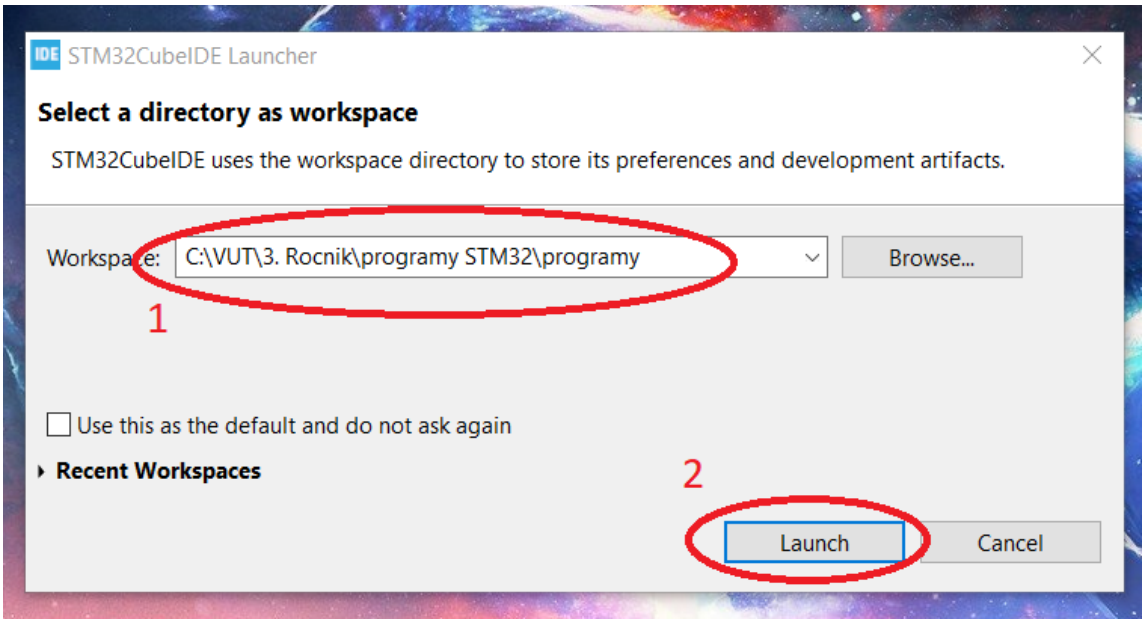
1. Manuál pro nastavení projektu

1.1. Spuštění STM32 CubeIDE



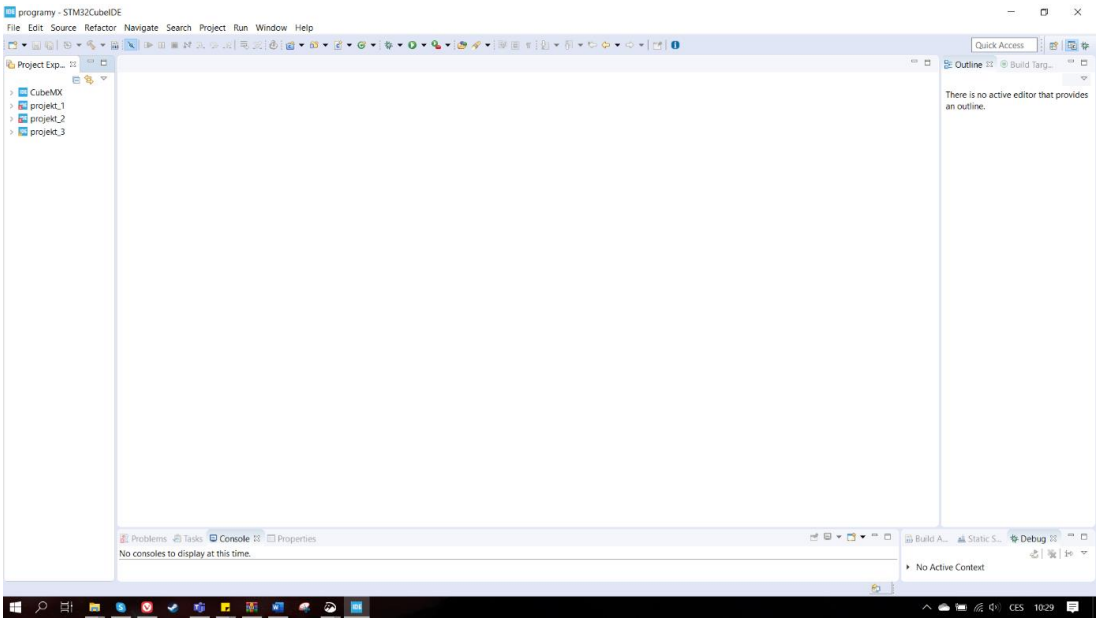
Obrázek 11. Spouštěcí soubor STM32CubeIDE

Pro spuštění programovacího prostřední je potřeba kliknout myší na **start** -> následně v seznamu otevřít složku **STMicroelectronics** -> a poté kliknout na soubor **STM32CubeIDE** s danou verzí.



Obrázek 12. STM32 CubeIDE Launcher

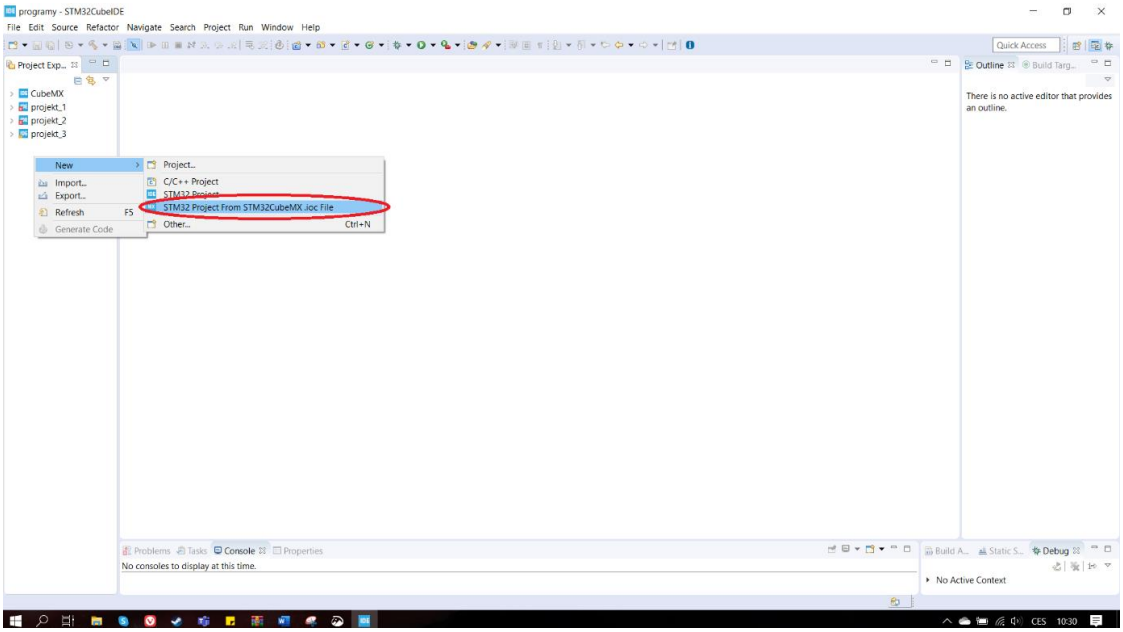
Následně se Vám zobrazí okno STM32CubeIDE Launcher. V prvním kroku je nutné nastavit místo, do kterého se budou vytvořené soubory ukládat. Místo, kam se budou programy ukládat je libovolné. V druhém kroku, poté co vyberete místo uložení, klikněte na tlačítko **Launch.**



Obrázek 13. Vývojové prostředí STM32 CubeIDE

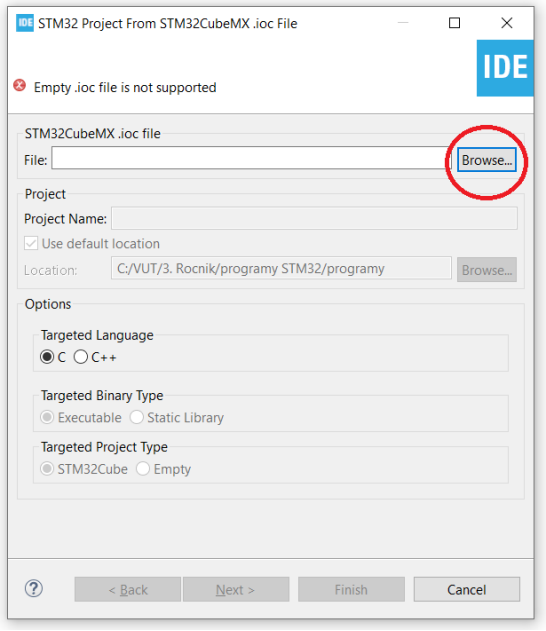
Po spuštění se Vám zobrazí prostředí STM32CubeIDE, které v levém sloupci obsahuje Vámi založené projekty, v horní části obrazovky pak panel nástrojů, ve spodní části konzoly a v pravém sloupci debug funkce

1.2. Založení projektu



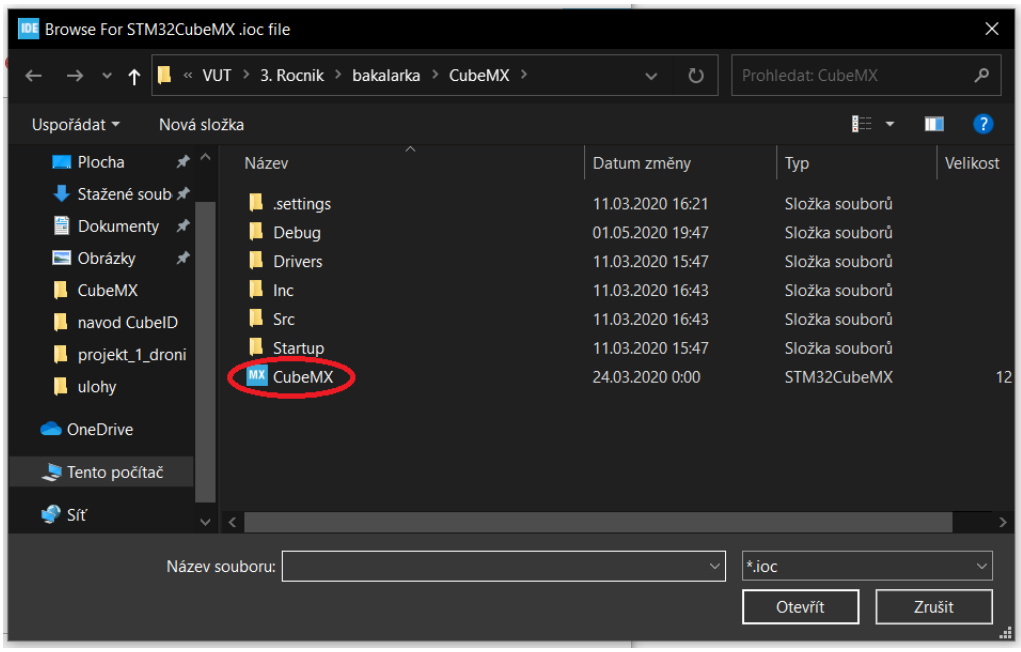
Obrázek 14. Založení nového projektu

Pro založení projektu je nutné kliknout **pravým** tlačítkem myši do **levého** sloupce. Následně se vám otevře seznam, v tomto seznamu označíte kurzorem myši na záložku **New** -> a následně levým tlačítkem myši kliknete na záložku **STM32 Project From STM32CubeMC.ioc File** .



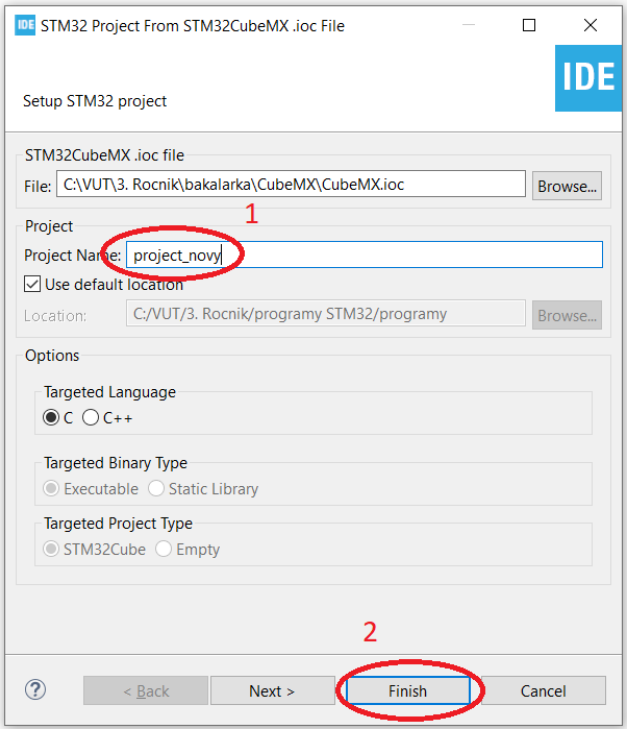
Obrázek 15. Druhá část založení nového projektu

Poté se Vám zobrazí okno na obrázku č. 5. Levým tlačítkem myši kliknete na **Browse.**



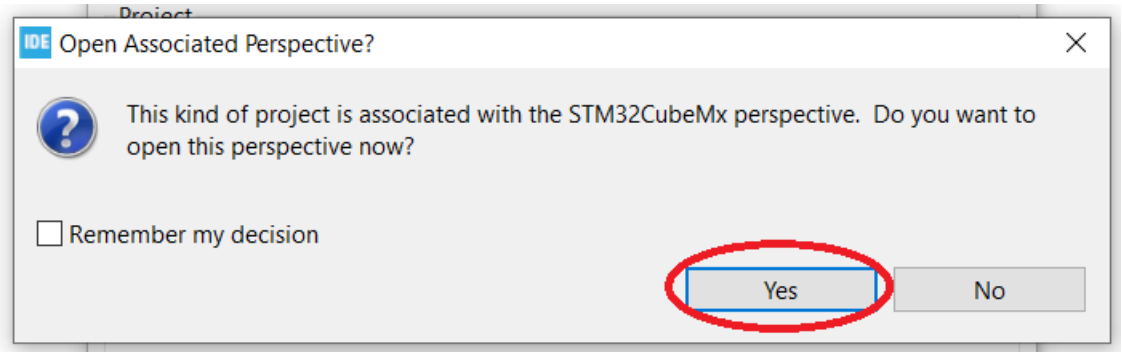
Obrázek 16. Výběr souboru CubeMX

Následně se Vám otevře okno, do kterého máte zadat umístění souboru s názvem CubeMX. Tento soubor si stáhněte z Elearningu, naleznete ho v kartě předmětu. Soubor si uložte do Vámi vytvořené složky na disku v počítači. Po uložení ve vyhledávacím okně klikněte levým tlačítkem myši na soubor CubeMX a klikněte na otevřít.



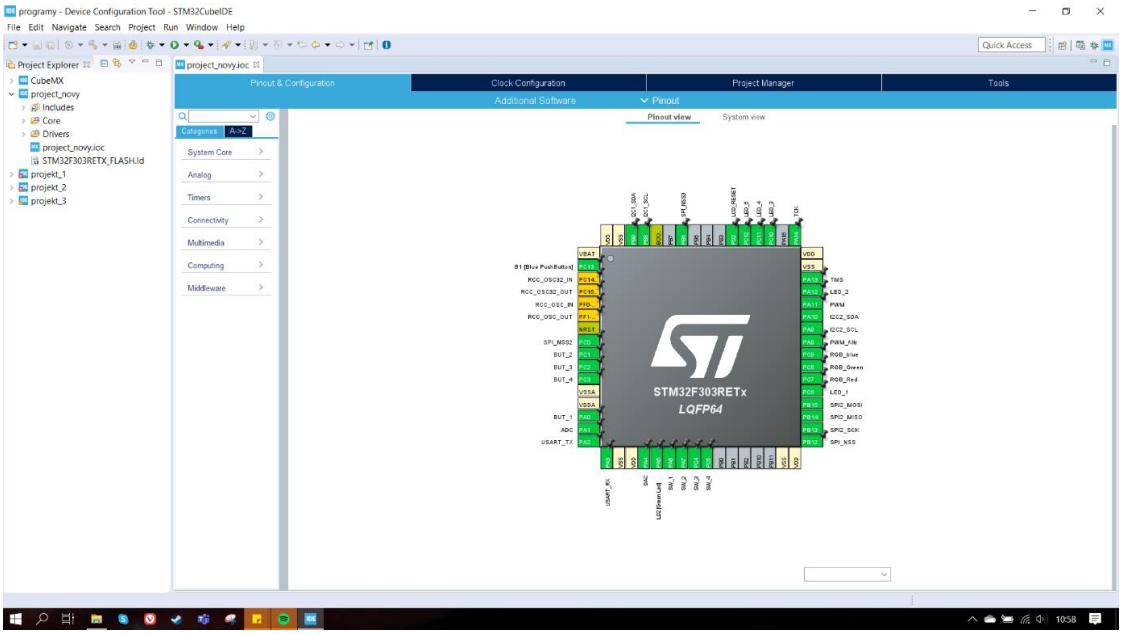
Obrázek 17. Třetí část založení nového projektu

Poté do kolonky „**Project Name**“ napište název Vámi nazvaného projektu a poté klikněte levým tlačítkem myši na **Finish**.



Obrázek 18. Okno "Open Associated Perspective"

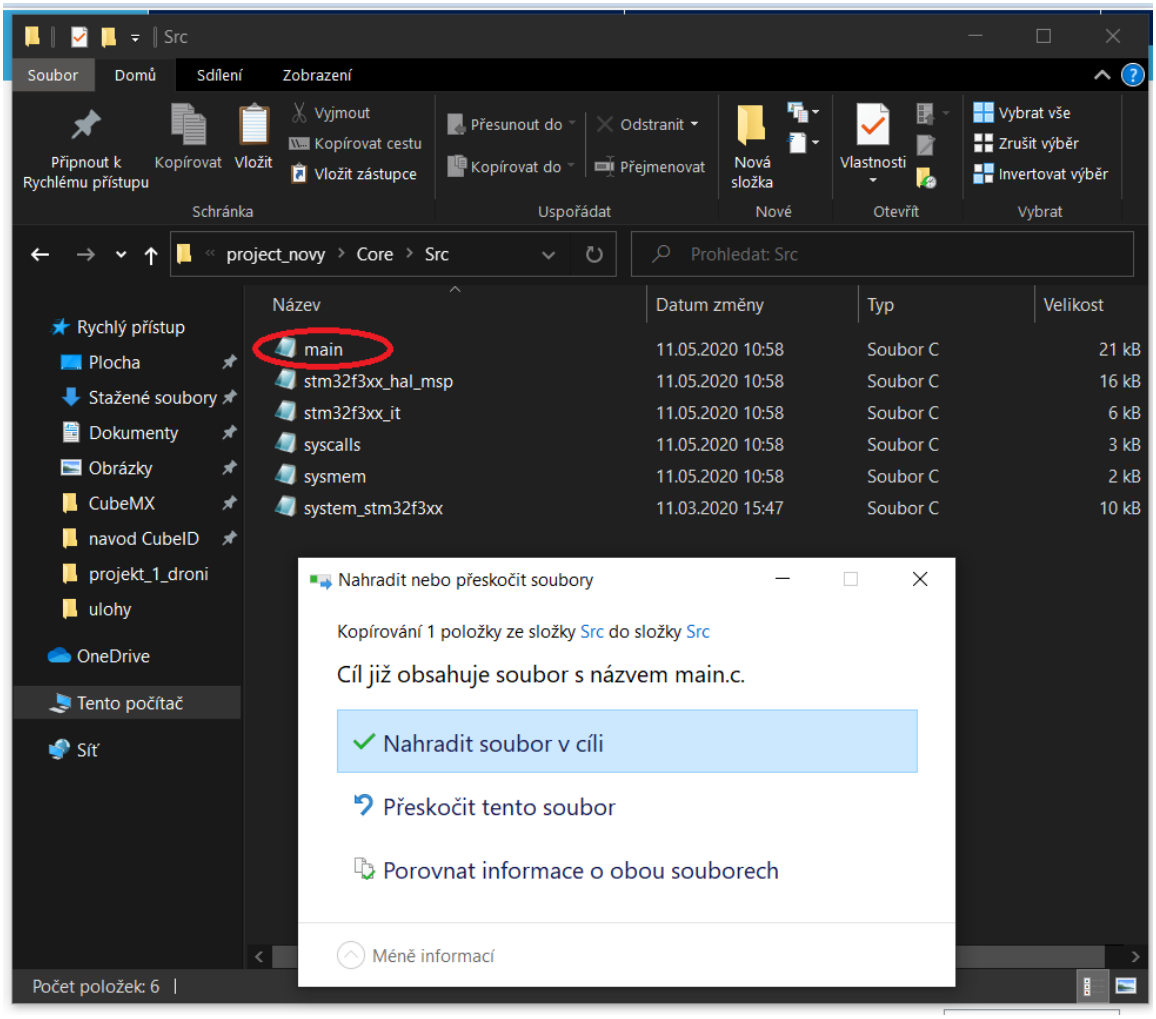
V tomto okně klikněte na „**Yes**“



Obrázek 19. Prostředí STM32 CubeIDE po založení nového projektu

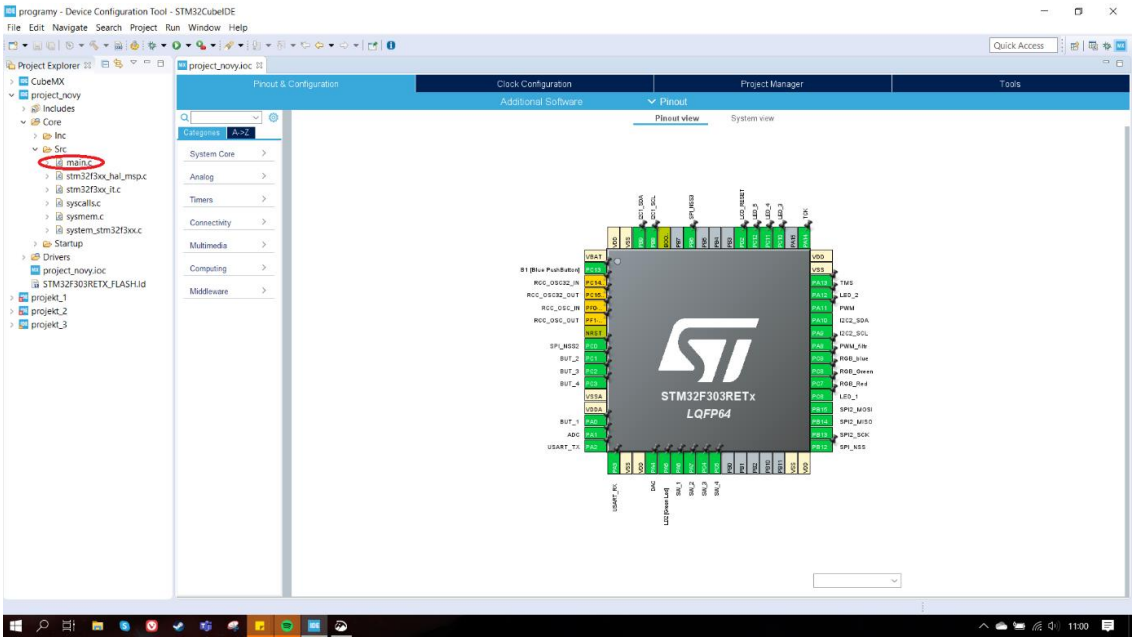
Následně se Vám otevře záložka s názvem Vašeho projektu. V horní části klikněte na křížek a tuto záložku vypněte.

1.3. Vytvoření souboru main.c



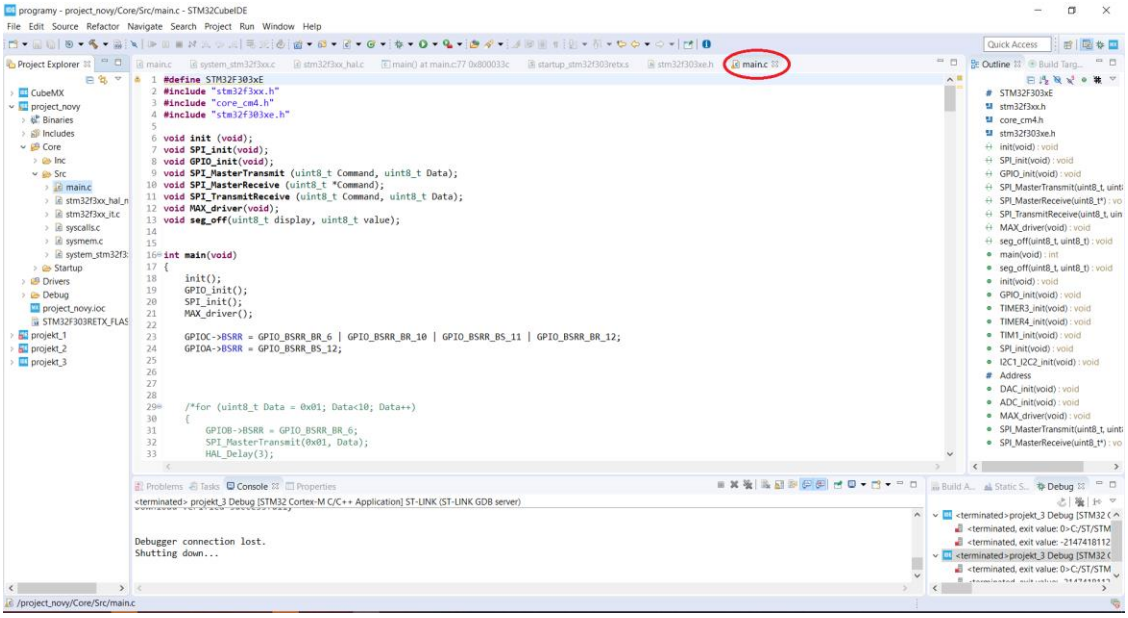
Obrázek 20. Přepis souboru "main.c" v nově založeném projektu

Ze složky předmětu na Elearningu si stáhněte soubor s názvem main. Poté si otevřete složku s Vaším projektem -> otevřete složku „**core**“ -> poté otevřete složku „**src**“ a v této složce přepište soubor „main“ za ten, který jste si stáhli z Elearningu.



Obrázek 21. Stav projektu po přepsání souboru "main.c"

Následně se vraťte zpátky do prostředí STM32 cubeIDE, otevřete si Váš projekt -> otevřete si složku „**core**“ -> následně si otevřete složku „**src**“ a dvojklikem levým tlačítkem myši otevřete soubor „**main.c**“.

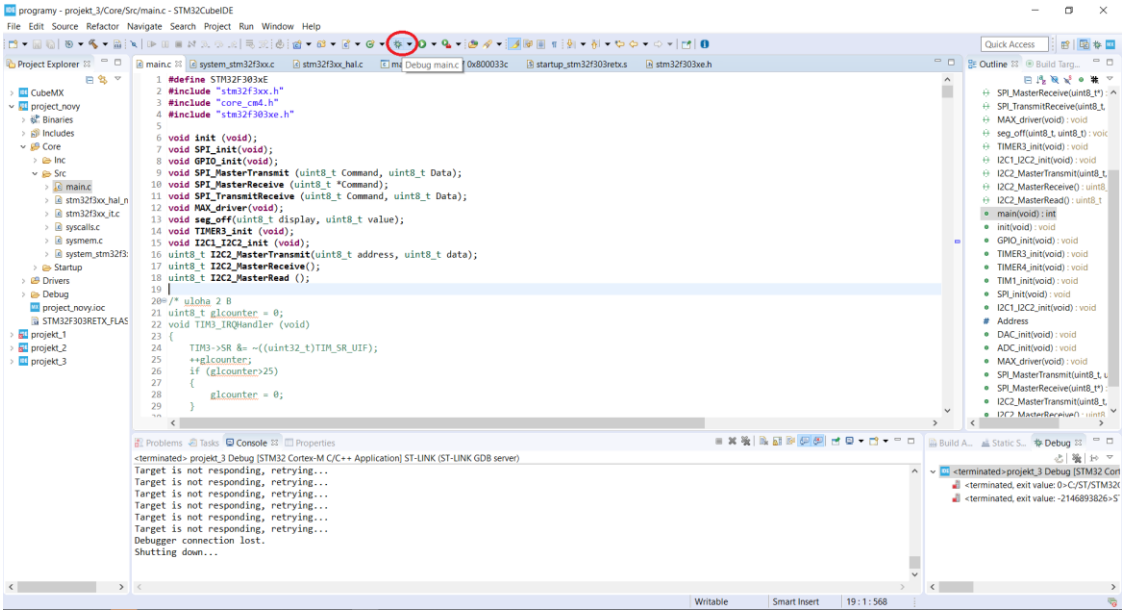


Obrázek 22. Soubor main.c otevřený v prostředí STM32 CubeIDE po přepsání souboru "main.c"

Poté uvidíte okno, které vypadá takto.

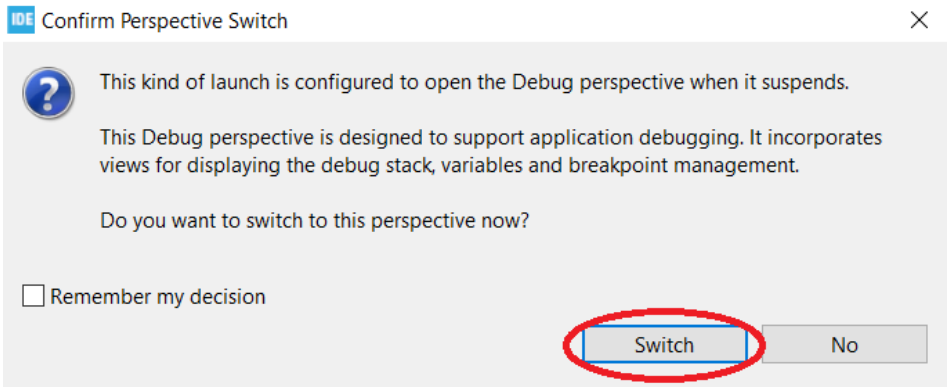
2. Manuál pro debug programu

2.1. Postup pro nahrávání programu do mikrokontroleru



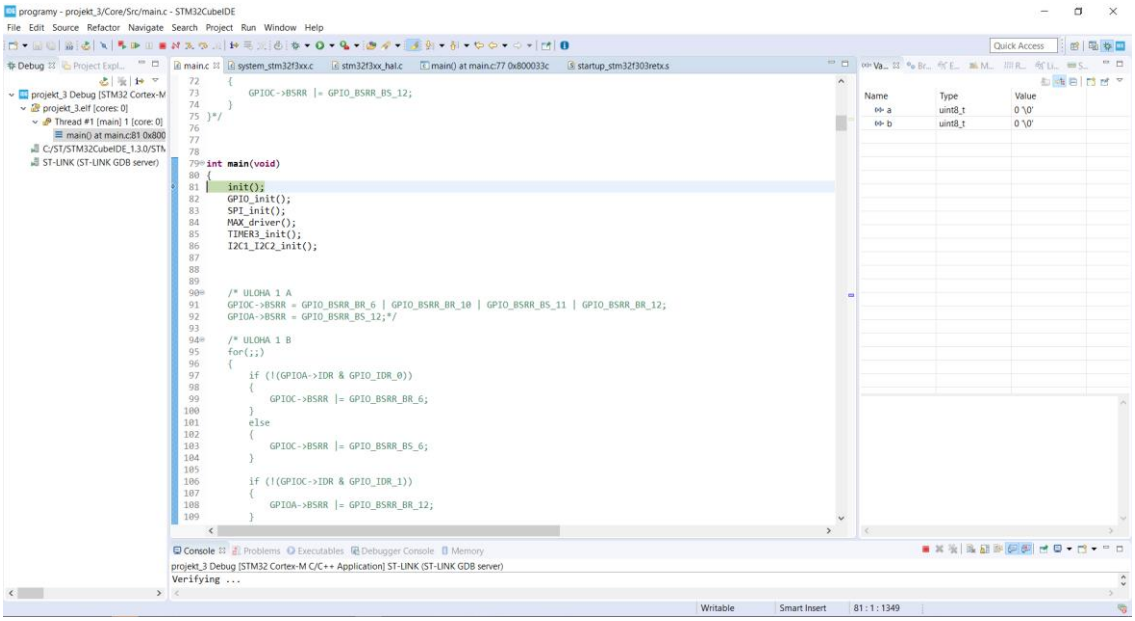
Obrázek 23. Přechod do debugovacího prostředí

Pro zprovoznění Vašeho programu je potřeba program nejprve nahrát do mikrokontroleru. Po vytvoření vašeho kódu kliknete levým tlačítkem myši na ikonku brouka s názvem „**Debug main.c**“



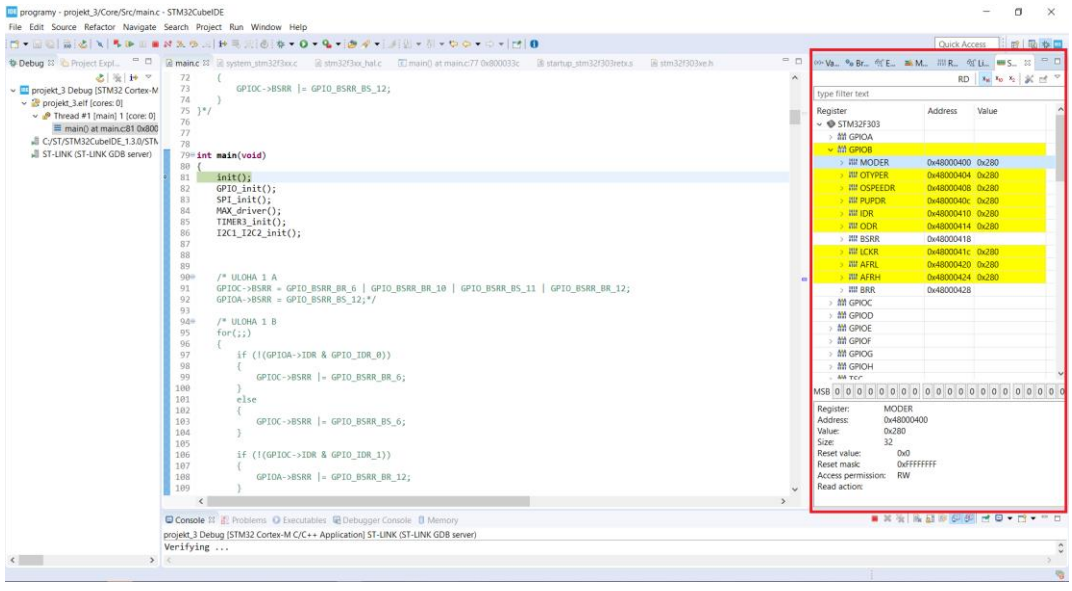
Obrázek 24. Následující část při přechodu do debugovacího prostředí

Po kliknutí na ikonku brouka Vás program vyzve k přesměrování do debugovacího prostředí, proto klikněte na tlačítko „**Switch**“



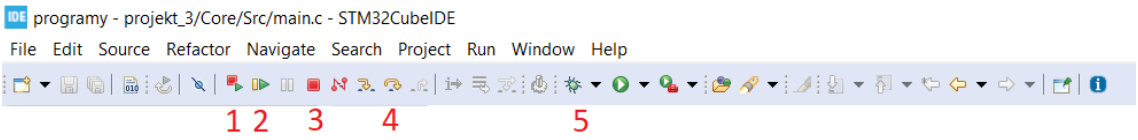
Obrázek 25. Debugovací prostředí

Poté Vás program přesměruje do debugovacího prostředí, které vypadá podobně jako na obrázku č. 15.



Obrázek 26. Ukázka pravého okna pro práci s SFR

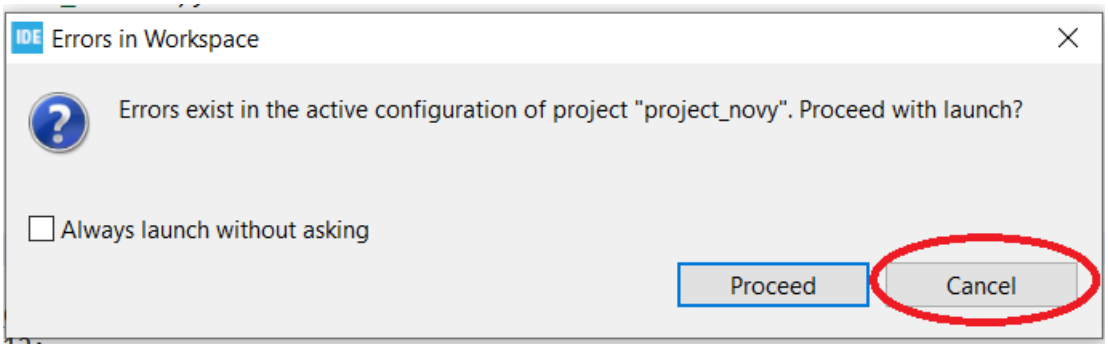
Pro rychlou úpravu jednotlivých registrů lze použít okno na pravé straně obrazovky a záložku „**SFR**“. V té je možné v reálném čase upravovat jednotlivé registry. V kódu se Vám tyto úpravy neprojeví, je proto nezbytné je následně vepsat správně do kódu.



Obrázek 27. Nástrojová lišta

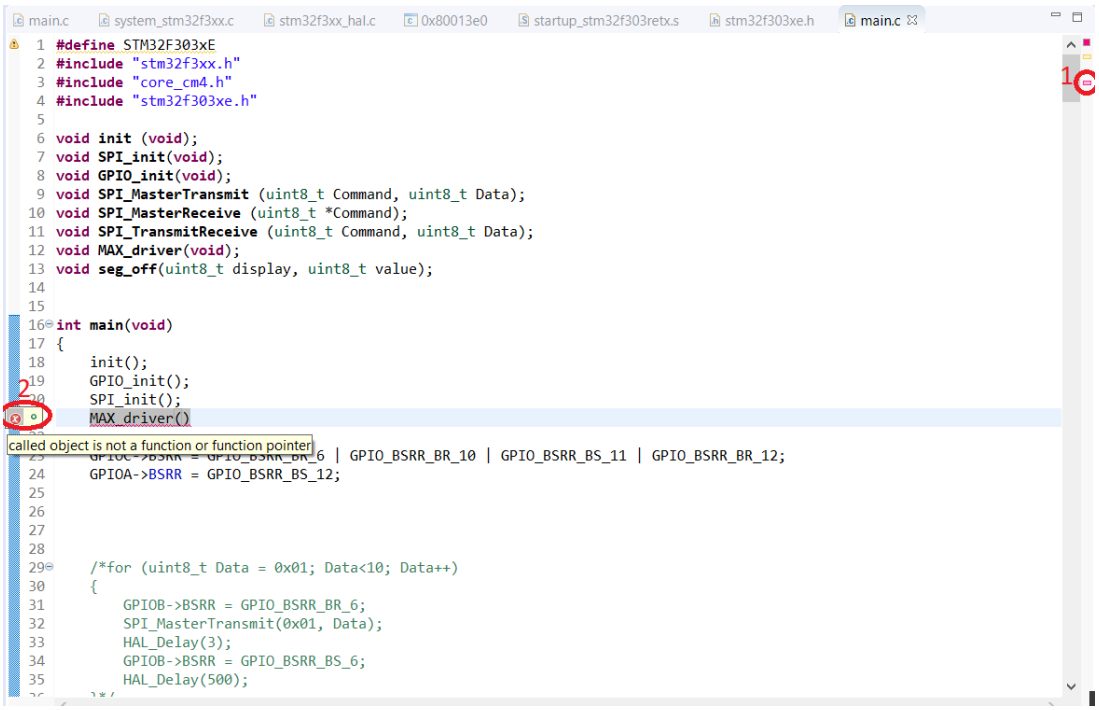
V rámci práce v debug prostředí je vhodné využívat některé z výše zvýrazněných nástrojů. Nástroj označený číslem 1 Vám kód zastaví a spustí od začátku. Nástroj dva Vám umožní po přechodu do debugovacího prostředí program spustit, případně je možné program zastavit „**pause**“ tlačítkem po pravé straně tlačítka 2. Tlačítko 3 Vás vrátí zpět do pre-debugovacího prostředí. Tlačítkem 4. je možné program krokovat. Tlačítkem 5 je nutné kód po každé úpravě zkompilovat.

2.2. Situace chyby kódu



Obrázek 28. Okno s chybovou hláškou

V případě, že je v kódu chyba, Vás program vyzve k přerušení operace, nebo k přeskočení chyby. Doporučuji kliknout na tlačítko „**cancel**“ a chybu opravit.

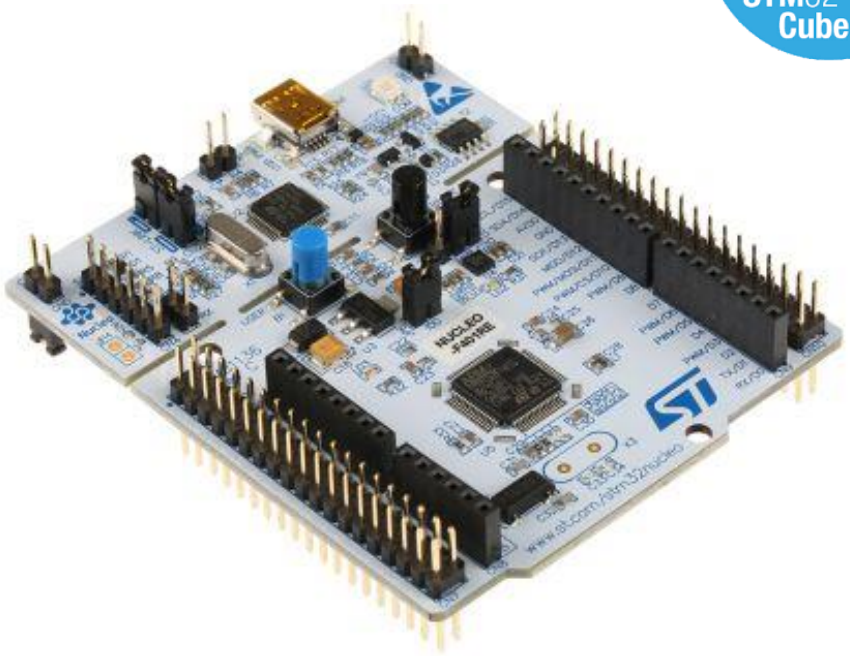


Obrázek 29. Označení chyb v rámci vývojového prostředí

Chyba se Vám zobrazí na dvou místech. Nejjednodušší je podívat so do posuvníku po pravé straně, vedle posuvníku se Vám ukáže červená odrážka označená číslem 1, při kliknutí na ni Vás program přesměruje na část kódu, která je špatně. Po označení křížku v bodě 2 kurzorem se Vám ukáže hláška, která upozorní na vaši chybu v kódu, po opravení chyby soubor uložte a znovu klikněte na ikonu brouka pro zkompilování kódu.

B.2 Manuál k inicializaci procesoru STM32 F303RE

Manuál k inicializaci procesoru STM32 F303RE



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Autor: Matěj Očenášek

Úvod

Tento dokument popisuje jednotlivé inicializační části kódu, které slouží k základnímu nastavení jednotlivých periferií obvodu. Jedná se zejména o ty periferie, které jsou využívány v jednotlivých laboratorních úlohách. Vytvoření vlastní inicializační části vede k lepšímu pochopení problematiky a vnitřnímu fungování mikroprocesoru. Programátor je schopen efektivněji psát svůj kód a šetřit tím místo, počet operací, tedy rychlost, a paměť procesoru.

Nastavení základního systémové inicializace

```
void init (void)
{

  RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
  //NASTAVENI "SYSTEM CONFIGURATIONCONTROLLER"
  RCC->APB1ENR |= RCC_APB1ENR_PWREN;
  //NASTAVENI "POWER CONTROL"
  RCC->CR |= RCC_CR_HSION;
  //SPUSTENI HSI
  while ((RCC->CR & RCC_CR_HSIRDY) != RCC_CR_HSIRDY ){};
  //CEKA NEZ SE VYSTUP STABILIZUJE
  RCC->CR &= ~RCC_CR_HSITRIM;
  // ZAKLADNI NASTAVENI PO KALIBRACI [0:7]
  RCC->CR |= 16U<<RCC_CR_HSITRIM_Pos;
  // KALIBRACE INTERNÍHO OSCILÁTORU NA PŘESNOU FREKVENCI
  RCC->CR &= ~RCC_CR_PLLON;
  // VYPNUTÍ PLL
  while ((RCC->CR & RCC_CR_PLLRDY) == RCC_CR_PLLRDY){};
  // ČEKANÍ DOKUD PLL NEBUDE VYPNUTO
  RCC->CFGR2 &= ~RCC_CFGR2_PREDIV;
  // NASTAVENÍ PŘEDDĚLIČKY NA HODNOTU 0
  RCC->CFGR &= ~(RCC_CFGR_PLLMUL | RCC_CFGR_PLLSRC);
  // NASTAVENÍ HODNOTY 0 PRO NASOBIČKU A ZDROJ HODIN
  RCC->CFGR |= RCC_CFGR_PLLMUL9 | RCC_CFGR_PLLSRC_HSI_PREDIV;
  // NASTAVENÍ NÁSBIČKY NA DANOU HODNOTU A VÝBĚR HODIN HSI
  RCC->CR |= RCC_CR_PLLON;
  // SPUŠTĚNÍ PLL
  while ((RCC->CR & RCC_CR_PLLRDY) != RCC_CR_PLLRDY){};
  // ČEKÁNÍ DOKUD NEBUDE PLL SPUŠTĚNO
  FLASH->ACR = FLASH_ACR_LATENCY_1;
  //NASTAVENÍ ZPOŽDĚNÍ NA TWO WAIT STATE
  RCC->CFGR &= ~RCC_CFGR_SW;
  // NASTAVENÍ HSI JAKO SYSTÉMOVÝCH HODIN
  RCC->CFGR |= RCC_CFGR_SW_PLL;
  // NASTAVENÍ PLL JAKO SYSTÉMOVÝCH HODIN
  while ((RCC->CFGR & RCC_CFGR_SWS_PLL) != RCC_CFGR_SWS_PLL){};
  // SPUŠTĚNÍ PLL JAKO SYSTÉMOVÝCH HODIN
  RCC->CFGR &= ~RCC_CFGR_HPRE;
  // NASTAVENÍ DĚLIČKY HODIN HLCK DO 0
  RCC->CFGR &= ~(RCC_CFGR_PPRE1 | RCC_CFGR_PPRE2);
  // VYPNUTÍ DĚLENÍ PRO APB1 A APB2
  RCC->CFGR |= RCC_CFGR_PPRE1_DIV2;
  // NASTAVENÍ DĚLIČKY PRO APB1 NA DĚLENÍ DVĚMA
  RCC->CFGR |= RCC_CFGR_PPRE2_DIV1;
  // NASTAVENÍ DĚLIČKY APB2 NA DĚLENÍ JEDNOU
  SystemCoreClockUpdate();
  //DO SYSTEMOVE PROMENNE SE ZAPISE RYCHLOST HODIN,...
  RCC->CFGR3 &= ~(RCC_CFGR3_I2C1SW | RCC_CFGR3_I2C2SW | RCC_CFGR3_TIM1SW
                  | RCC_CFGR3_TIM34SW | RCC_CFGR3_USART2SW);
  // VYPNUTÍ VŠECH HODIN PRO VYBRANÉ PERIFERIE
  RCC->CFGR2 &= ~RCC_CFGR2_ADCPRE12;
  // VYPNUTÍ HODIN PRO AD PŘEVODNÍK
  RCC->CFGR3 |= RCC_CFGR3_I2C1SW_HSI | RCC_CFGR3_I2C2SW_HSI |
```

```
                RCC_CFGR3_TIM1SW_HCLK | RCC_CFGR3_TIM34SW_HCLK |
                RCC_CFGR3_USART2SW_PCLK;
    // ZAPNUTÍ VŠECH HODIN PRO VYBRANÉ PERIFERIE
    RCC->CFGR2 |= RCC_CFGR2_ADCPRE12_DIV1;
    // ZAPNUTÍ HODIN PRO AD PŘEVODNÍK
    NVIC_SetPriorityGrouping(0x03);
    // NASTAVENÍ TŘÍD PŘERUŠENÍ NA HODNOTU 3
    SysTick_Config(SystemCoreClock/1000);
    // NASTAVENÍ SYSTICKU
    SysTick->CTRL |= 0x00000004U;
    // NASTAVENÍ SYSTICKU
    NVIC_SetPriority(SysTick_IRQn,
    NVIC_EncodePriority(NVIC_GetPriorityGrouping(),0,0));
    //NASTAVENÍ PŘERUŠENÍ PRO SYSTICK

}
```

Základní nastavení Arm jádra pro požadovanou rychlost včetně základního nastavení hodinových obvodů.

Vzor nastavení GPIO výstupu

```
void GPIO_init (void)
{

  RCC->AHBENR |= RCC_AHBENR_GPIOAEN | RCC_AHBENR_GPIOBEN |
              RCC_AHBENR_GPIOCEN | RCC_AHBENR_GPIODEN;
  //POVOLENÍ HODIN PRO VÝSTUPY

  //INIT LEDKY
  GPIOC->BSRR = GPIO_BSRR_BS_6 | GPIO_BSRR_BS_10 | GPIO_BSRR_BS_11 |
               GPIO_BSRR_BS_12;
  //NASTAVENÍ JEDNOTLIVÝCH LEDEK DO 1
  GPIOC->MODER &= ~(0x03U<<12 | 0x03U<<20 | 0x03U<<22 | 0x03U<<24);
  // NASTAVENÍ MÓDU GPIO DO HODNOTY 0
  GPIOC->MODER |= GPIO_MODER_MODER6_0 | GPIO_MODER_MODER10_0 |
               GPIO_MODER_MODER11_0 | GPIO_MODER_MODER12_0;
  //NASTAVENÍ GPIO JAKO VÝSTUPU
  GPIOC->OTYPER &= ~(0x01U<<6 | 0x01U<<10 | 0x01U<<11 | 0x01U<<12);
  // NASTAVENÍ VÝSTUPU JAKO PUSH-PULL
  GPIOC->OSPEEDR &= ~(0x03U<<12 | 0x03U<<20 | 0x03U<<22 | 0x03U<<24);
  // NASTAVENÍ RYCHLOSTI JAKO LOW-SPEED
  GPIOC->PUPDR &= ~(0x03U<<12 | 0x03U<<20 | 0x03U<<22 | 0x03U<<24);
  // NASTAVENÍ REGISTRU PUPDR DO 0
  GPIOC->PUPDR |= 0x02U<<12 | 0x02U<<20 | 0x02U<<22 | 0x02U<<24;
  // NASTAVENÍ NA PULL-DOWN
  GPIOA->BSRR = GPIO_BSRR_BS_12;
  GPIOA->MODER |= GPIO_MODER_MODER12_0;
  GPIOA->PUPDR |= 2<<24;

}
```

GPIO piny můžeme vnímat jako volně užitné piny. Nejčastěji se používají pro ovládání LED, tlačítek nebo přepínačů, mohou být nastaveny jako vstupní nebo výstupní piny. GPIO init je nezbytnou součástí všech GPIO pinů. V GPIO initu je na začátku nutné zapnout hodiny pro všechny využité GPIO piny. Hodiny, které jsou přiřazené k jednotlivým periferiím jako časovače, sběrnice nebo právě GPIO piny, lze nalézt v příslušném katalogovém listu daného procesoru. Registry, které je potřeba pro dané GPIO piny nastavit se mohou lehce lišit v závislosti na využití, podrobný postup nastavování jednotlivých registrů je vždy přesně popsán v referenčním manuálu. Ve výše uvedeném kódu došlo k nastavení registru bit set/reset (BSRR) k nastavení výstupních pinů, na kterých jsou připojeny jednotlivé LED do logické 1. V port mode registru (MODER) dochází nejprve k nulování (vypnutí) registru a následného nastavení GPIO pinu jako výstupního. Dále je pak nastaven output type register do push-pull stavu (reset state). Pomocí nulování registru output speed register (OSPEEDER) je nastavena nejnižší úroveň rychlosti. Nakonec dochází k nastavení PUPDR registru, kde se výstup nastavuje jako pull-down.

Vzor nastavení časovače výstupu

```
    void TIMER3_init (void)
    {
RCC->APB1ENR |= RCC_APB1ENR_TIM3EN; //POVOLENI HODIN
    TIM3->CR1 &= ~(TIM_CR1_CKD);// NASTAVENÍ DĚLENÍ HODIN DO 0
    TIM3->CR1 &= ~(TIM_CR1_CMS);// NASTAVENÍ REGISTRU CMS DO 0
    TIM3->CR1 |= (TIM_CR1_ARPE);// NASTAVENÍ BUFFROVÁNÍ
    TIM3->CR1 &= ~(TIM_CR1_DIR);// NASTAVENÍ SMĚRU ČÍTÁNÍ DOLŮ
    TIM3->PSC = 48000-1; //NASTAVENÍ DELIČKY
    TIM3->ARR = 15-1; //NASTAVENÍ KROKU
    TIM3->EGR = TIM_EGR_UG; //NASTAVENÍ UPDATU PRO REGISTER
    TIM3->SMCR &= ~(TIM_SMCR_MSM); // VYPNUTÍ MASTER SLAVE MODU
    TIM3->CR2 &= ~(TIM_CR2_MMS);// VYPNUTÍ MASTER MODE

    TIM3->CCER &= ~(TIM_CCER_CC2E | TIM_CCER_CC3E | TIM_CCER_CC4E);
    //VYPNUTI JEDNOTLIVYCH KANALU
    TIM3->CCMR1 &= ~(TIM_CCMR1_OC2M | TIM_CCMR1_CC2S);
    // VYPNUTÍ OUTPUT COMPARE MODU 2 I CAPTURE/COMPARE VÝBĚRU PRO KANÁL 2
    TIM3->CCMR2 &= ~(TIM_CCMR2_OC3M | TIM_CCMR2_CC3S | TIM_CCMR2_OC4M |
                     TIM_CCMR2_CC4S);
    // VYPNUTÍ OUTPUT COMPARE MODU 2 I CAPTURE/COMPARE VÝBĚRU PRO KANÁL 3 A 4
    TIM3->CCMR1 |= (TIM_CCMR1_OC2M_1 | TIM_CCMR1_OC2M_2);
    //ZAPNUTÍ OUTPUT COMPARE MODU PRO KANÁL 2
    TIM3->CCMR2 |= (TIM_CCMR2_OC3M_1 | TIM_CCMR2_OC3M_2 | TIM_CCMR2_OC4M_1 |
                    TIM_CCMR2_OC4M_2);
    // ZAPNUTÍ OUTPUT COMPARE MODU PRO KANÁL 2 A 3
    TIM3->CCER |= (TIM_CCER_CC2P | TIM_CCER_CC3P | TIM_CCER_CC4P);
    //NASTAVENI POLARITY VYSTUPU
    TIM3->CCMR1 |= (TIM_CCMR1_OC2PE);
    // NASTAVENÍ HODNOTY NA KTERÉ ZAČNE ČÍTAT KANÁL 2
    TIM3->CCMR2 |= (TIM_CCMR2_OC3PE | TIM_CCMR2_OC4PE);
    // NASTAVENÍ HODNOTY NA KTERÉ ZAČNE ČÍTAT KANÁL 3 A 4
    TIM3->CCMR1 &= ~(TIM_CCMR1_OC2FE);
    // VYPNUTÍ FAST ENABLE PRO KANÁL 2
    TIM3->CCMR2 &= ~(TIM_CCMR2_OC3FE | TIM_CCMR2_OC4FE);
     // VYPNUTÍ FAST ENABLE PRO KANÁL 3 A 4

    }
```

V této části je ukázka kódu pro inicializaci časovače 3. Ne všechny časovače jsou stejné, a tudíž je při nastavování jednotlivých registrů potřeba postupovat dle návodu v referenčním manuálu daného procesoru, nebo skupiny procesoru. V rámci procesoru STM32 F303RE jsou časovače rozděleny do skupin TIM1/TIM8/TIM20, TIM2/TIM3/TIM4, TIM6/TIM7 a TIM15/TIM16/TIM17. Jednotlivé skupiny se od sebe liší zejména komplexností jednotlivých časovačů, skupina časovačů TIM6/TIM7 se v referenčním manuálu nazývá „basic“, což v překladu do češtiny znamená základní. Častým využitím časovačů je pak pulzně šířková modulace (PWM – pulse width modulation). Každý samotný časovač obsahuje dané množství kanálů. Například časovač 3, jehož inicializace je uvedena výše, obsahuje celkem 4 kanály. Na výukové DPS, která je řízena procesorem STM32 F303RE je kanál 2,3 a 4 vyveden na jednotlivé piny RGB led. Zde je možné pomocí změny střídy u PWM regulovat svítivost jednotlivých barev.

Vzor nastavení SPI komunikace

```
void SPI_init (void)
{
   RCC->APB1ENR |= RCC_APB1ENR_SPI2EN;
   //POVOLENÍ HODIN APB1 PRO SPI2
   RCC->APB1RSTR |= (RCC_APB1RSTR_SPI2RST);
   //RESTART HODIN APB1 PRO PERIFERII SPI2
   RCC->APB1RSTR &= ~(RCC_APB1RSTR_SPI2RST);
   // NASTAVENÍ RESETOVACÍ REGISTRU DO 0
   SPI2->CR1 |= (SPI_CR1_SSM);
   //POVOLENÍ SOFTWAROVÉHO ŘÍZENÍ SLAVE
   SPI2->CR1 |= (SPI_CR1_SSI);
   //NASTAVENÍ IGNORACE NSS PINU
   SPI2->CR1 |= (SPI_CR1_MSTR);
   //KONFIGURACE SPI JAKO MASTER
   SPI2->CR2 |= (SPI_CR2_DS_0 | SPI_CR2_DS_1 | SPI_CR2_DS_2);
   //NASTAVENÍ DELKY DAT PTO SPI PŘENOS
   SPI2->CR2 |= (SPI_CR2_FRXTH);
   //NASTAVUJE V PAMETI FIFO DELKU PRIJATYCH DAT NA 8-BITŮ
   SPI2->I2SCFGR &= ~(SPI_I2SCFGR_I2SMOD);
   //NASTAVENÍ SPI MODU V I2SCFGR REGISTRU

   GPIOB->AFR [1] &= ~(0x0FU<<20 | 0x0FU<<24 | 0x0FU<<28);
   //NASTAVENÍ VÝSTUPNÍCH PINŮ MOSI,MISO,SCK
   GPIOB->AFR [1] |= (0x05U<<20 | 0x05U<<24 | 0x05U<<28);
   GPIOB->MODER &= ~(0x03U<<26 | 0x03U<<28 | 0x03U<<30);
   GPIOB->MODER |= (GPIO_MODER_MODER13_1 | GPIO_MODER_MODER14_1 |
            GPIO_MODER_MODER15_1);
   GPIOB->OTYPER &= ~(0x01U<<13 | 0x01U<<14 | 0x01U<<15);
   GPIOB->OSPEEDR |= (0x03U<<26 | 0x03U<<28 | 0x03U<<30);
   GPIOB->PUPDR &= ~(0x03U<<26 | 0x03U<<28 | 0x03U<<30);

   GPIOB->BSRR = (GPIO_BSRR_BS_6 | GPIO_BSRR_BS_12); //NSS1 A NSS3
   GPIOB->MODER &= ~(0x03U<<12 | 0x03U<<24);
   GPIOB->MODER |= (GPIO_MODER_MODER6_0 | GPIO_MODER_MODER12_0);
   GPIOB->OTYPER &= ~(0x01U<<6 | 0x01U<<12);
   GPIOB->OSPEEDR |= (0x03U<<12 | 0x03U<<24);
   GPIOB->PUPDR &= ~(0x03U<<12 | 0x03U<<24);
   GPIOC->BSRR =  (GPIO_BSRR_BS_0); //NSS2
   GPIOC->MODER &= ~(0x03U);
   GPIOC->MODER |= GPIO_MODER_MODER0_0;
   GPIOC->OTYPER &= ~(0x01U);
   GPIOC->OSPEEDR &= ~(0x03U);
   GPIOC->PUPDR &= ~(0x03U);

   SPI2->CR1 |= (SPI_CR1_SPE);
   // POVOLENÍ SPI

}
```

Výše uvedený kód popisuje inicializaci sériové sběrnice SPI. Popis registrů, které je nutné v rámci inicializace použít, je chronologicky popsán v referenčním manuálu. Na rozdíl od inicializaci jiných periferií, se inicializace GPIO pinů, využívaných SPI sběrnicí provádí přímo v inializační funkci SPI. Jsou to zejména GPIO inity podřízených zařízení, datových pinů MOSI, MISO a pinů pro hodinový signál SCK. GPIO inicializace jiných periferií se provádí ve společné funkci „void GPIO_init (void)“. Nastavení pinů podřízených zařízení lze nastavit s negativní logikou (NSS), nebo s kladnou logikou (SS).

Vzor nastavení I²C komunikace

```
void I2C1_I2C2_init (void)
{
   RCC->APB1ENR |= (RCC_APB1ENR_I2C1EN | RCC_APB1ENR_I2C2EN);
   //POVOLENÍ HODIN APB1 PRO I2C1 A I2C2
   RCC->APB1RSTR |= (RCC_APB1RSTR_I2C1RST | RCC_APB1RSTR_I2C2RST);
   //RESTART HODIN APB1 PRO PERIFERII SPI2
   RCC->APB1RSTR &= ~(RCC_APB1RSTR_I2C1RST | RCC_APB1RSTR_I2C2RST);
   //NASTAVENÍ RESETOVACÍ REGISTRU DO 0

   #define Address 0 //DEFINOVANÍ PROMĚNNÉ ADDRESS


   I2C1->CR1 &= ~(I2C_CR1_PE);
   //NULOVÁNÍ REGISTRU PRO SPUŠTĚNÍ PERIFERIE I2C
   I2C1->CR1 |= (0x05U<<I2C_CR1_DNF_Pos);
   //SPUŠTĚNÍ A NASTAVENÍ FILTROVACÍ KAPACITY DO 5 HODINOVÝCH SIGNÁLŮ
   I2C1->TIMINGR |= (0x03U<<I2C_TIMINGR_SCLDEL_Pos);
   //NASTAVENÍ ZPOŽDĚNÍ MEZI SDA A SCL NASTUPNOUT HRANOU
   I2C1->TIMINGR |= (0x01U<<I2C_TIMINGR_SDADEL_Pos);
   //NASTAVENÍ ZPOŽDĚNÍ MEZI SDA A SCL SESTUPNOU HRANOU
   I2C1->TIMINGR |= (0x09U<<I2C_TIMINGR_SCLL_Pos);
   //NASTAVENÍ NÍZKÉ PERIODY PRO SCL V MASTER MODU
   I2C1->TIMINGR |= (0x03U<<I2C_TIMINGR_SCLH_Pos);
   //NASTAVENÍ VYSOKÉ PERIODY PRO SCL V MASTER MODU
   I2C1->CR2 |= (I2C_CR2_NACK);
   //NASTAVUJE POSÍLÁNÍ NACK PO PŘIJATÉM BYTU
   I2C1->CR2 |= (I2C_CR2_AUTOEND);
   //STOP PODMÍNKA SE POŠLE AUTOMATICKY PO PŘENESENÍ NBYTES
   I2C1->OAR1 &= ~(I2C_OAR1_OA1EN);
   //NASTAVENÍ BITU OA1EN DO 0
   I2C1->OAR1 |= (Address | I2C_OAR1_OA1EN);
   //POVOLENÍ BITU OA1EN S PŘIŘAZENÍM ADRESY
   I2C1->OAR2 &= ~(I2C_OAR2_OA2EN);
   //NASTAVENÍ BITU OA2EN DO 0
   I2C1->CR1 |= (I2C_CR1_PE);
   //POVOLENÍ I2C1

   I2C2->CR1 &= ~(I2C_CR1_PE);
   I2C2->CR1 |= (0x05U<<I2C_CR1_DNF_Pos);
   I2C2->TIMINGR |= (0x03U<<I2C_TIMINGR_SCLDEL_Pos);
   I2C2->TIMINGR |= (0x01U<<I2C_TIMINGR_SDADEL_Pos);
   I2C2->TIMINGR |= (0x09U<<I2C_TIMINGR_SCLL_Pos);
   I2C2->TIMINGR |= (0x03U<<I2C_TIMINGR_SCLH_Pos);
   I2C2->CR2 |= (I2C_CR2_NACK);
   I2C2->CR2 |= (I2C_CR2_AUTOEND);
   I2C2->OAR1 &= ~(I2C_OAR1_OA1EN);
   I2C2->OAR1 |= (Address | I2C_OAR1_OA1EN);
   I2C2->OAR2 &= ~(I2C_OAR2_OA2EN);
   I2C2->CR1 |= (I2C_CR1_PE);

}
```

Výše je uveden kód pro inicializaci I²C1 a I²C2. Vzhledem k tomu, že se obě sběrnice nastavují stejně, je komentář uveden pouze k té první. I2C je sériová sběrnice a postup pro nastavení jednotlivých registrů je popsán v referenčním manuálu. Od sběrnice SPI se liší zejména nastavením registru pro nastavení reakce sběrnice na nástupnou a sestupnou hranu

Vzor nastavení D/A převodníku

```
{
  RCC->APB1ENR |= (RCC_APB1ENR_DAC1EN);
  //POVOLENÍ HODIN APB1 PRO DAC1
  RCC->APB1RSTR |= (RCC_APB1RSTR_DAC1RST);
  //RESTART HODIN APB1 PRO PERIFERII DAC1
  RCC->APB1RSTR &= ~(RCC_APB1RSTR_DAC1RST);
  // NASTAVENÍ RESETOVACÍ REGISTRU DO 0

  DAC->CR |= (DAC_CR_EN1);
  //POVOLENÍ DAC

}
```

Výše uvedený kód představuje základní nastavení A/D převodníku. Vzhledem k tomu, že D/A převodník je na výukové DPS realizován přes tranzistor MOSFET, je v základním nastavení nutné zapnout pro tuto periferii pouze hodinový signál, nastavit ji do základního (restartovaného) stavu a spustit samotnou periferii.

Vzor nastavení A/D převodníku

```
void ADC_init (void)
{
  RCC->AHBENR |= (RCC_AHBENR_ADC12EN);
  //POVOLENÍ HODIN AHB PRO ADC12
  RCC->AHBRSTR |= (RCC_AHBRSTR_ADC12RST);
  //RESTART HODIN AHB PRO PERIFERII ADC12
  RCC->AHBRSTR &= ~(RCC_AHBRSTR_ADC12RST);
  //NASTAVENÍ RESETOVACÍ REGISTRU DO 0

  ADC1->CFGR |= (ADC_CFGR_OVRMOD);
  //NASTAVENÍ MÓDU PŘETEČENÍ -> REGISTER JE PŘEPSÁN POSLEDNÍM VÝSLEDKEM
  ADC1->CFGR |= (ADC_CFGR_CONT);
  //NASTAVENÍ KONVERZNÍHO MÓDU JAKO POKRAČUJÍCÍHO
  ADC12_COMMON->CCR &= ~(ADC_CCR_CKMODE);
  //NASTAVENÍ HODINOVÉHO MÓDU PRO ADC DO 0 (ASYNCHRONÍ MÓD)
  ADC1->CR &= ~(ADC_CR_ADVREGEN_1);
  // NULOVÁNÍ BITU PRO REGULACI NAPĚTÍ
  ADC1->CR |= (ADC_CR_ADVREGEN_0);
  //NASTAVOVÁNÍ REGULACE NAPĚTÍ U ADC
  ADC1->SMPR1 |= (ADC_SMPR1_SMP2 | ADC_SMPR1_SMP0) << 6;
  //NASTAVENÍ KANÁLU A VZORKOVACÍHO ČASU
  ADC1->SQR1 |= (ADC_SQR1_SQ1_1);
  //BIT NASTAVUJÍCÍ KONVERZI V REGULÉRNÍ SEKVENCI

  ADC1->CR |= (ADC_CR_ADCAL);
  //KALIBRACE ADC
  while ((ADC1->CR & ADC_CR_ADCAL) == (ADC_CR_ADCAL)) {};
  //ČEKÁNÍ NEŽ DOJDE KE ZKALIBROVÁNÍ
  ADC1->CR |= ADC_CR_ADEN;
  //POVOLENÍ ADC

}
```

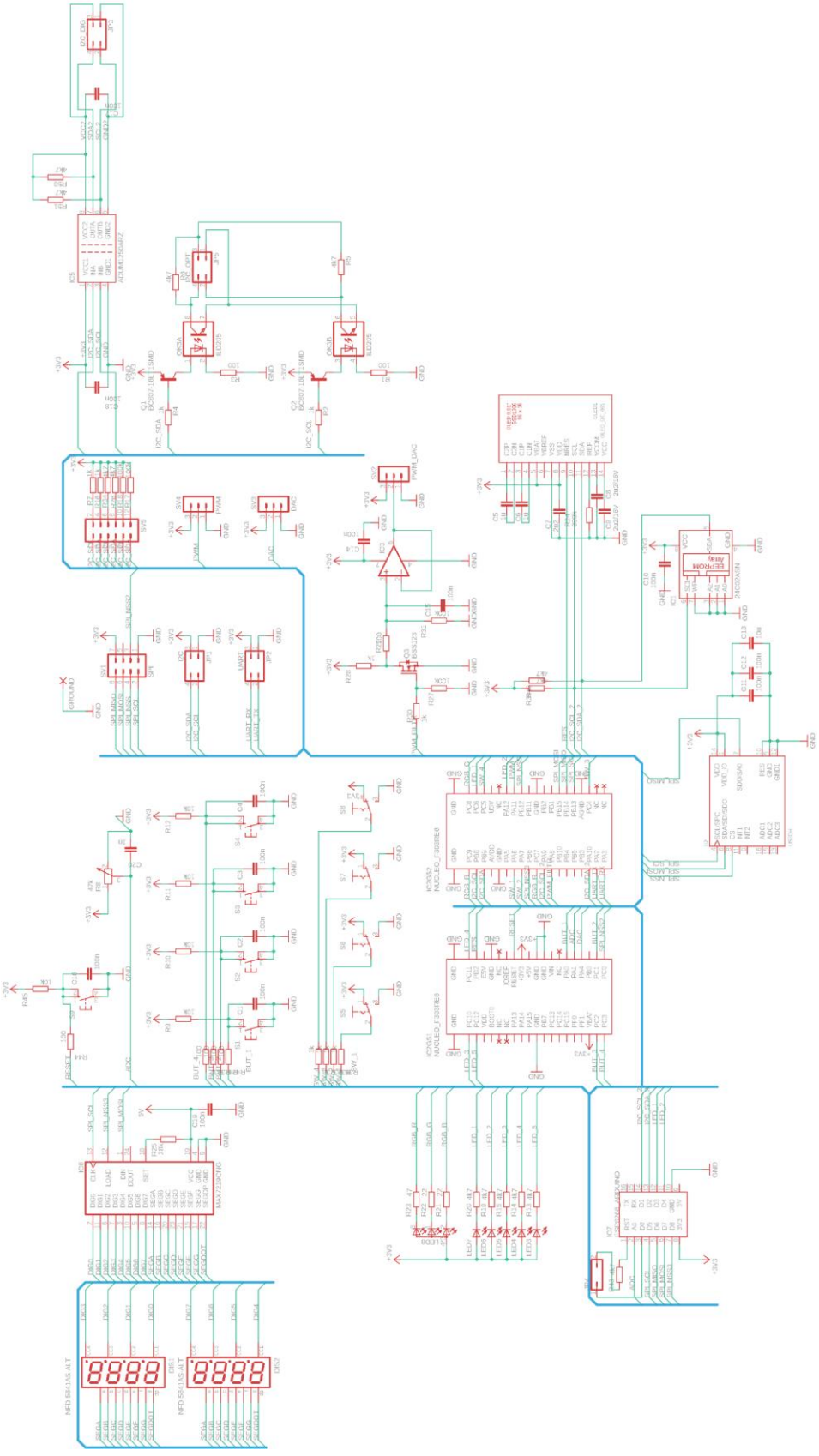
Výše uvedený kód představuje inicializaci A/D převodníku. A/D převodník už dle názvu převádí analogový signál na vstupu na digitální. Je možné ho využívat například k ovládání jasu LED. Na výukové DPS je A/D převodník realizován pomocí potenciometru. Vzhledem k širokým možnostem A/D převodníku je nutné velice podrobně nastudovat referenční manuál. A/D převodník, jako i zbylé periferie, se vždy nastavuje dle potřeb pro dané použití.

Vzor nastavení řadiče MAX7219CNG

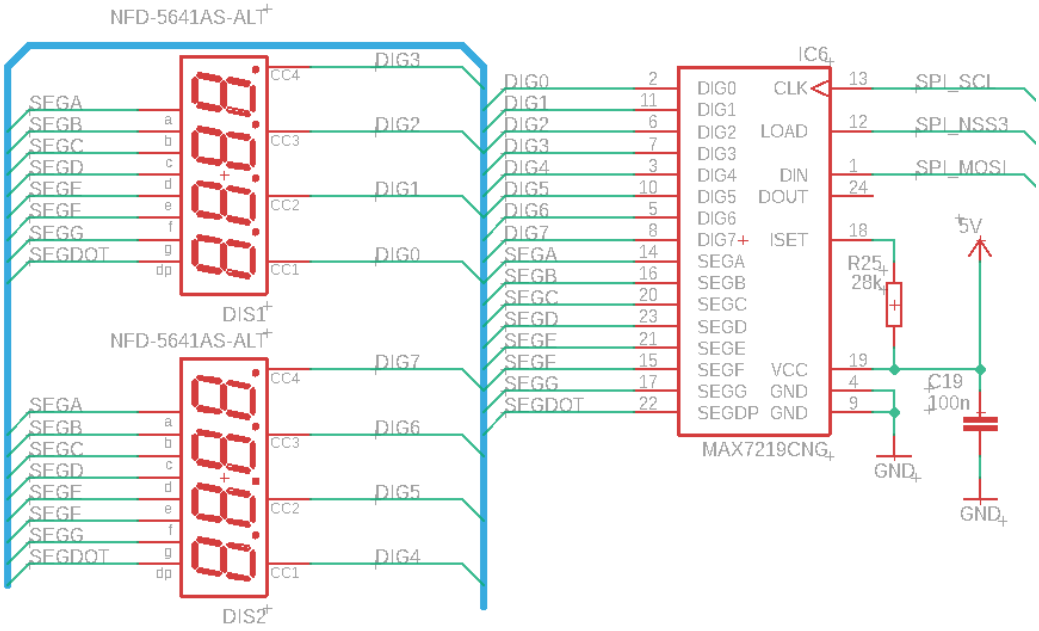
```
void MAX_driver(void)
{
  GPIOB->BSRR = GPIO_BSRR_BR_6;
  SPI_MasterTransmit(0x0B, 0x07);
  //NASTAVENÍ SCAN LIMITU PRO VŠECH 8 DIGITŮ
  GPIOB->BSRR = GPIO_BSRR_BS_6;
  HAL_Delay(3);
  GPIOB->BSRR = GPIO_BSRR_BR_6;
  SPI_MasterTransmit(0x09, 0xFF);//NASTAVENÍ DECODOVANI PRO DIGITY 7-0
  GPIOB->BSRR = GPIO_BSRR_BS_6;
  HAL_Delay(3);
  GPIOB->BSRR = GPIO_BSRR_BR_6;
  SPI_MasterTransmit(0x0F, 0x00);
  //NASTAVENI DISPLAY TESTU DO NORMÁLNÍHO OPERAČNÍHO MODU
  GPIOB->BSRR = GPIO_BSRR_BS_6;
  HAL_Delay(3);
  GPIOB->BSRR = GPIO_BSRR_BR_6;
  SPI_MasterTransmit(0x0A, 0x0F);//NASTAVENÍ NEJVĚTŠÍ INTENZITY
  GPIOB->BSRR = GPIO_BSRR_BS_6;
  HAL_Delay(3);
  GPIOB->BSRR = GPIO_BSRR_BR_6;
  SPI_MasterTransmit(0x00, 0x00);//ŽÁDNÁ OPERACE
  GPIOB->BSRR = GPIO_BSRR_BS_6;
  HAL_Delay(3);
  GPIOB->BSRR = GPIO_BSRR_BR_6;
  SPI_MasterTransmit(0x0C, 0x01);//POVOLENÍ VÝSTUPŮ NORMÁLNÍCH OPERACÍ
  GPIOB->BSRR = GPIO_BSRR_BS_6;
  HAL_Delay(3);
}
```

Základní nastavení řadiče MAX7219CNG osazeného na výukové desce pro ovládání sedmisegmentového pole. Samotný řadič je řízený přes sběrnice SPI přes NSS3. Jednotlivé nastavení registrů v řadiči vychází z požadavků na programy v úloze. První proměnná v závorce značí adresu, na kterou je potřeba zapsat a druhá proměnná značí hodnotu. Při nastavování jednotlivých registrů je vždy nutné postupovat podle informací v katalogovém listu.

B.3 Manuál k zapojení jednotlivých periferií

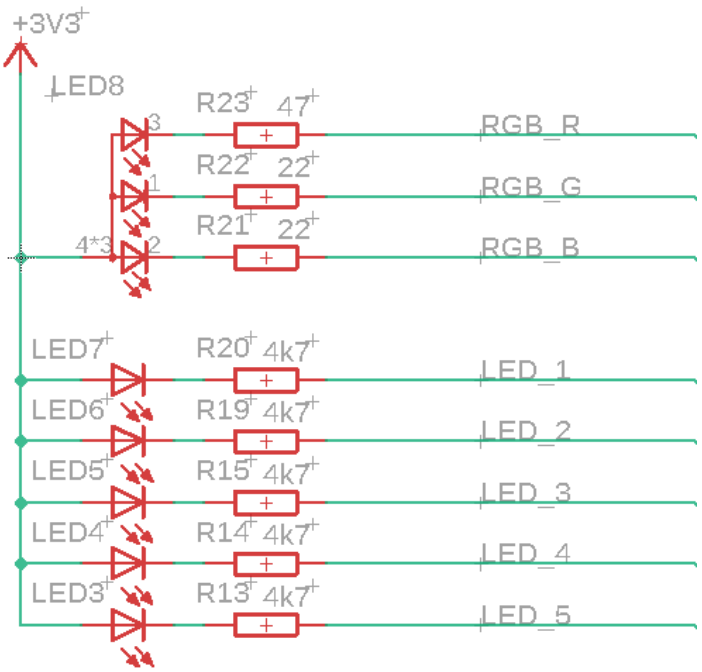


8x 7-segmenty řízený řadičem MAX7219CNG



MAX7219CNG	PIN	PORT
CLK – hodinový signál	13	PB13
LOAD	12	PB6
DIN – digitální vstup	1	PB15
DOUT – digitální výstup	24	NC
ISET	18	+5V
VCC – napájení	19	+5V
GND – zem	4	GND
GND - zem	9	GND

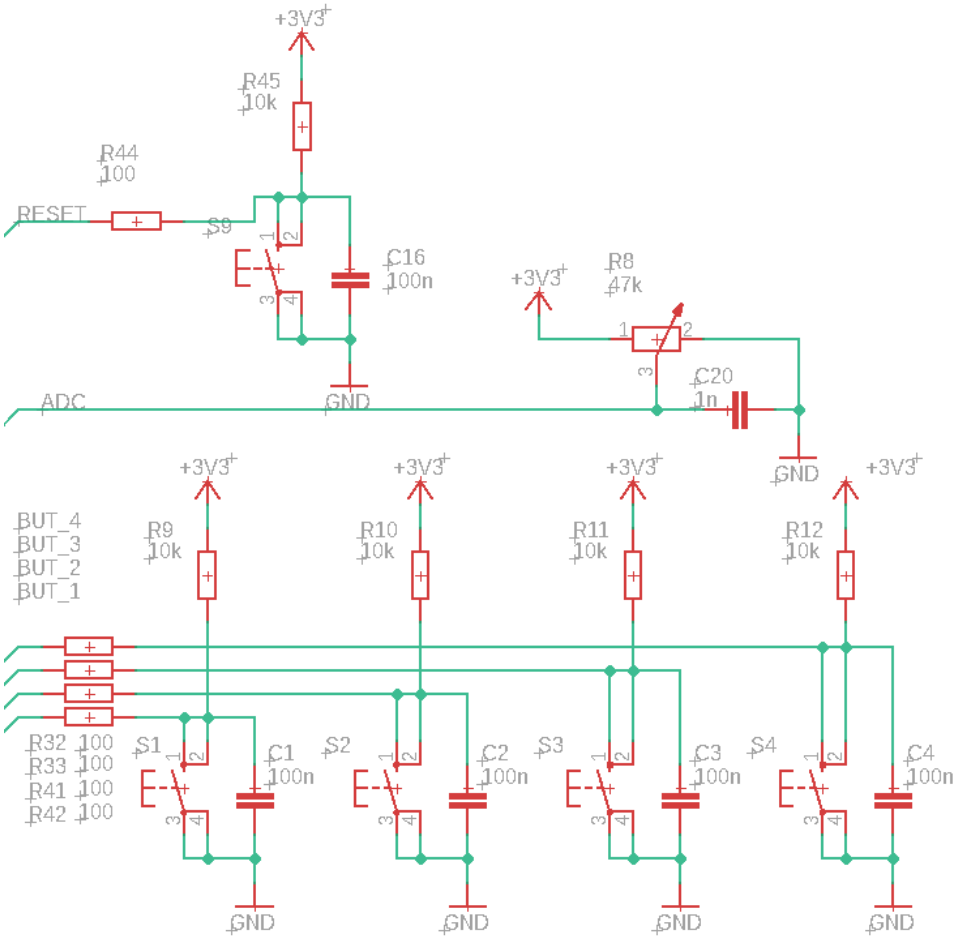
RGB a 5x LED



RGB LED	PIN	PORT
RED	x	PC9
GREEN	x	PC8
BLUE	x	PC7

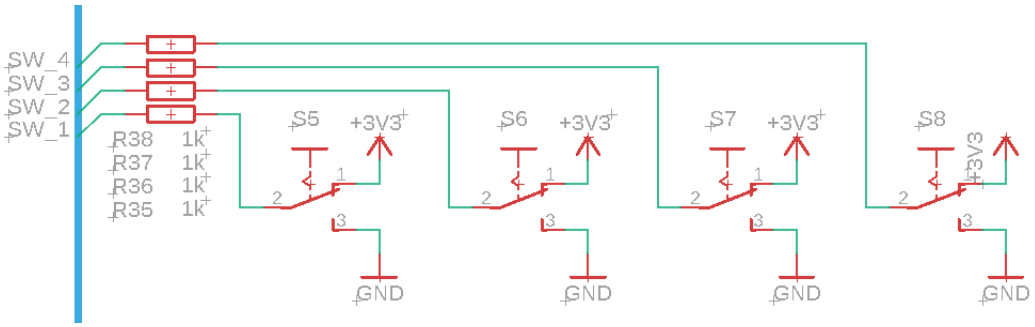
5x LED	PIN	PORT
LED_1	x	PC6
LED_2	x	PA12
LED_3	x	PC10
LED_4	x	PC11
LED_5	x	PC12

Tlačítka



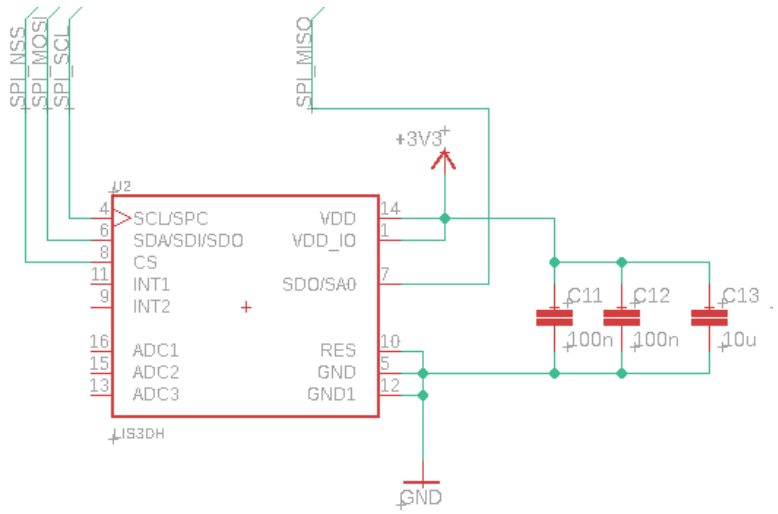
Tlačítka	PIN	PORT
BUT_1	x	PA0
BUT_2	x	PC1
BUT_3	x	PC2
BUT_4	x	PC3
RESET	x	NRST

Přepínače



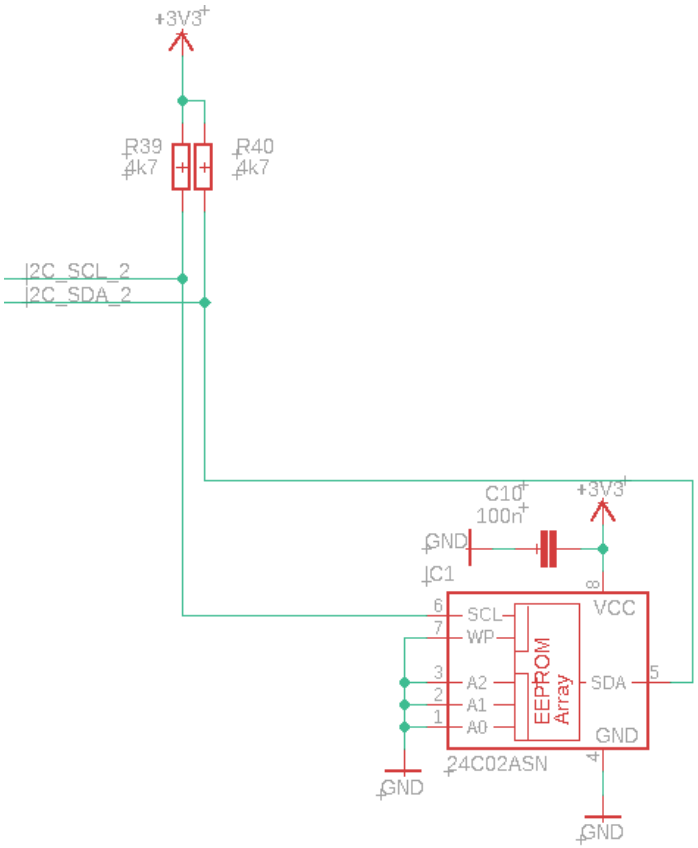
Přepínač	PIN	PORT
SW_1	x	PA6
SW_2	x	PA7
SW_3	x	PC4
SW_4	x	PC5

Akcelerometr/gyroskop LIS3DH STMicroelectronics – SPI BUS



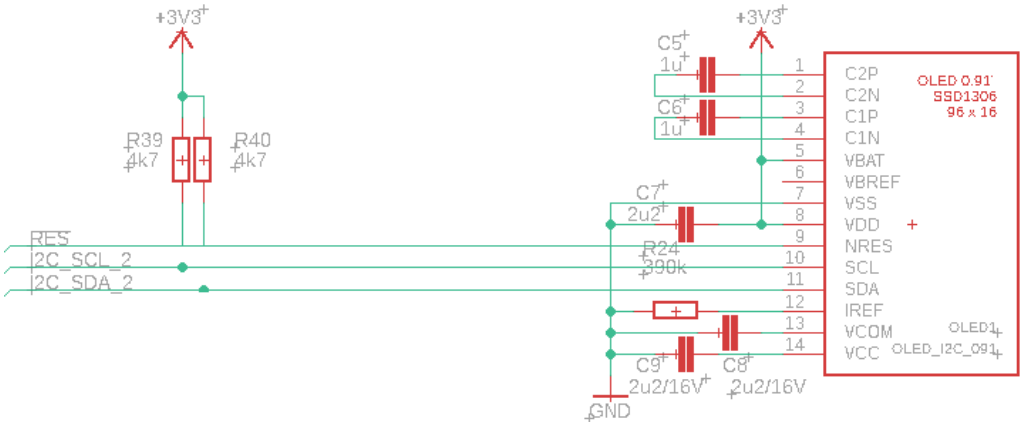
Akce/Gyro LIS3DH	PIN	PORT
SPI_SCL (SCL/SPC)	4	PB13
SPI_MOSI (SDA/SDI/SDO)	6	PB15
SPI_NSS (CS)	8	PB12
INT1	11	NC
INT2	9	NC
ADC1	16	NC
ADC2	15	NC
ADC3	13	NC
VDD	14	+ 3,3V
VDD_IO	1	+ 3,3V
SPI_MISO (SDO/SA0)	7	PB14
RES	10	GND
GND	5	GND
GND1	12	GND

Paměť EEPROM 24C02ASN – I2C BUS



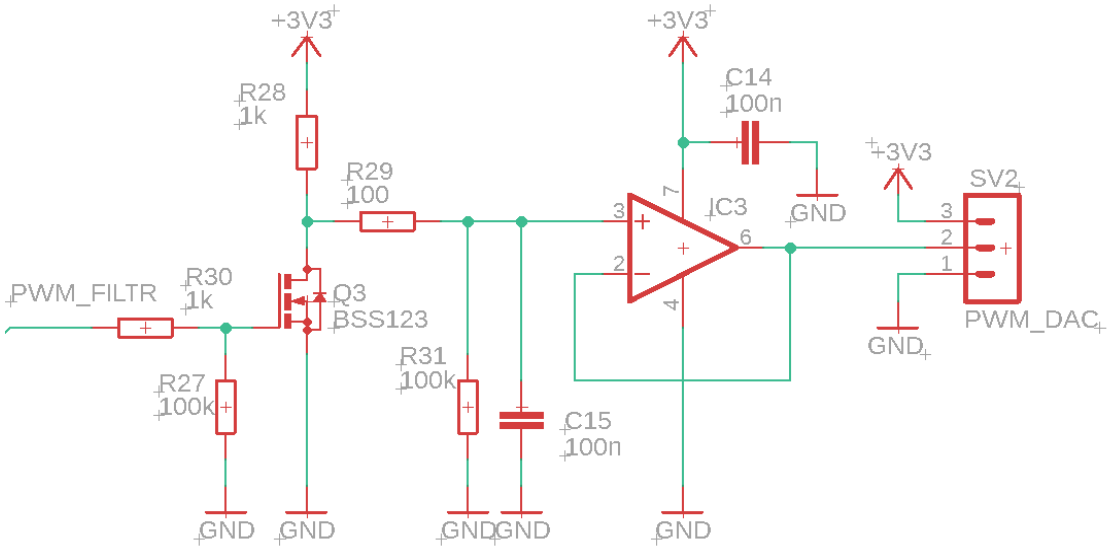
Paměť EEPROM	PIN	PORT
I2C_SCL_2	6	PA9
I2C_SDA_2	5	PA10
WP	7	GND
A2	3	GND
A1	2	GND
A0	1	GND
VCC	8	+3,3V
GND	4	GND

OLED Displej – I2C BUS



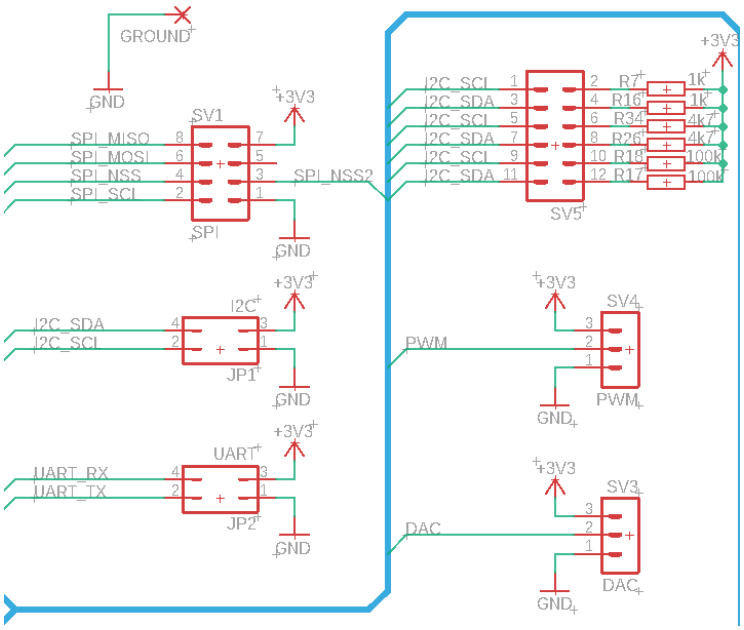
Displej	PIN	PORT
C2P	1	C2N
C2N	2	C2P
C1P	3	C1N
C1N	4	C2P
VBAT	5	+3,3V
VBREF	6	NC
VSS	7	GND
VDD	8	GND
RESET (NRES)	9	PD2
I2C_SCL_2 (SCL)	10	PA9
I2C_SDA_2 (SDA)	11	PA10
IREF	12	GND
VCOM	13	GND
VCC	14	GND

PWM Filtr



PWM filter	PIN	PORT
PWM_FILTR	x	PA8

Výstupní hřebínky pro sledování signálu přes osciloskop



SV1 – SPI sběrnice	PIN	PORT
SPI_MISO	8	PB14
SPI_MOSI	6	PB15
SPI_NSS	4	PB12
SPI_SCL	2	PB13
VCC	7	+3,3V
Volný pin	5	NC
SPI_NSS2	3	PC0
GND	1	GND

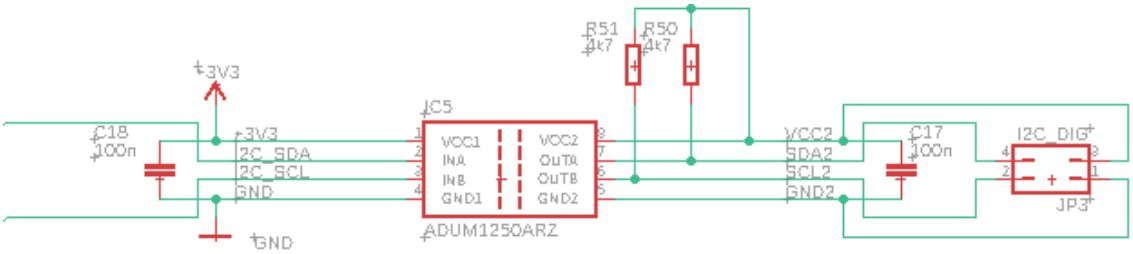
JP1/SV5 – I2C sběrnice	PIN	PORT
I2C_SDA	4	PB9
I2C_SCL	2	PB8
VCC	3	+3,3V
GND	1	GND

JP2 – UART sběrnice	PIN	PORT
UART_RX	4	PA3
UART_TX	2	PA2
VCC	3	+3,3V
GND	1	GND

SV4 – PWM	PIN	PORT
PWM	2	PA11
VCC	3	+3,3V
GND	1	GND

SV3 - DAC	PIN	PORT
DAC	2	PA4
VCC	3	+3,3V
GND	1	GND

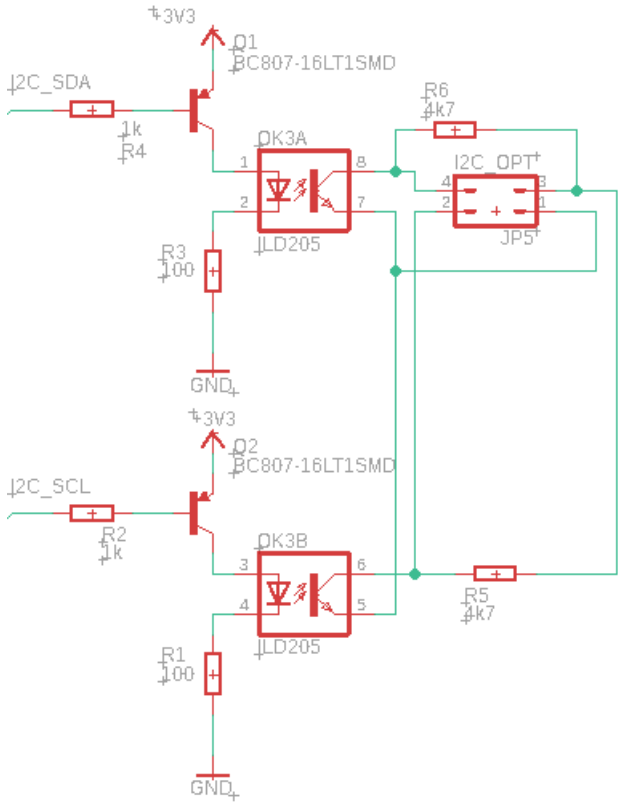
Sledování signálu na sběrici I2C za digitálním izolátorem



IC5 – digitální izolátor - IN	PIN	PORT
VCC	1	+3,3V
I2C_SDA	2	PB9
I2C_SCL	3	PB8
GND	4	GND

JP3 – digitální izolátor - OUT	PIN	PORT
VCC	3	+3,3V
I2C_SDA	4	SDA
I2C_SCL	2	SCL
GND	1	GND

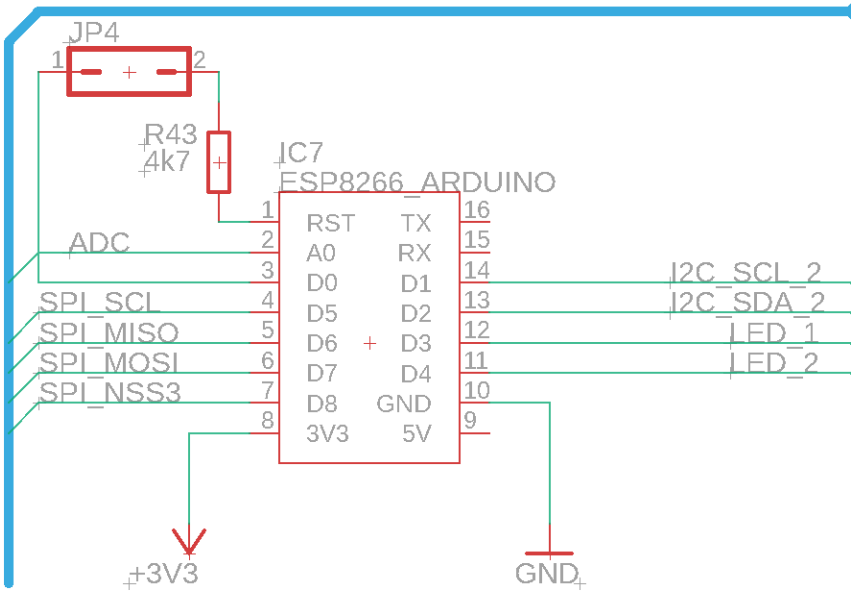
Sledování signálu na sběrici I2C za optočlenem



OK3 A/B - IN	PIN	PORT
I2C_SDA	1	PB9
I2C_SCL	3	PB8

JP5 – optočlen - OUT	PIN	PORT
VCC	3	+3,3V
I2C_SDA	4	I2C_SDA
I2C_SCL	2	I2C_SCL
GND	1	GND

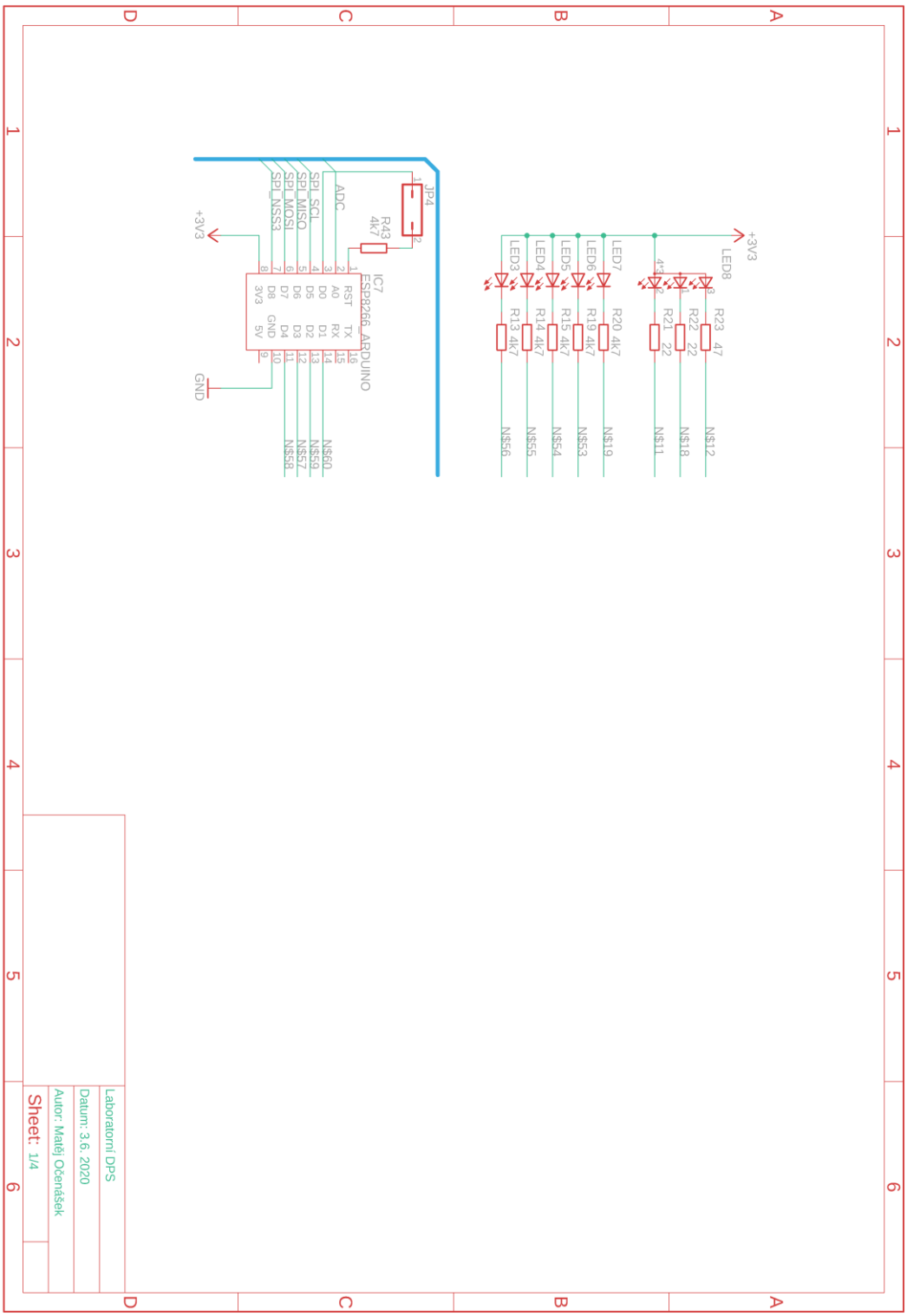
Wi-Fi modul ESP8266 Arduino



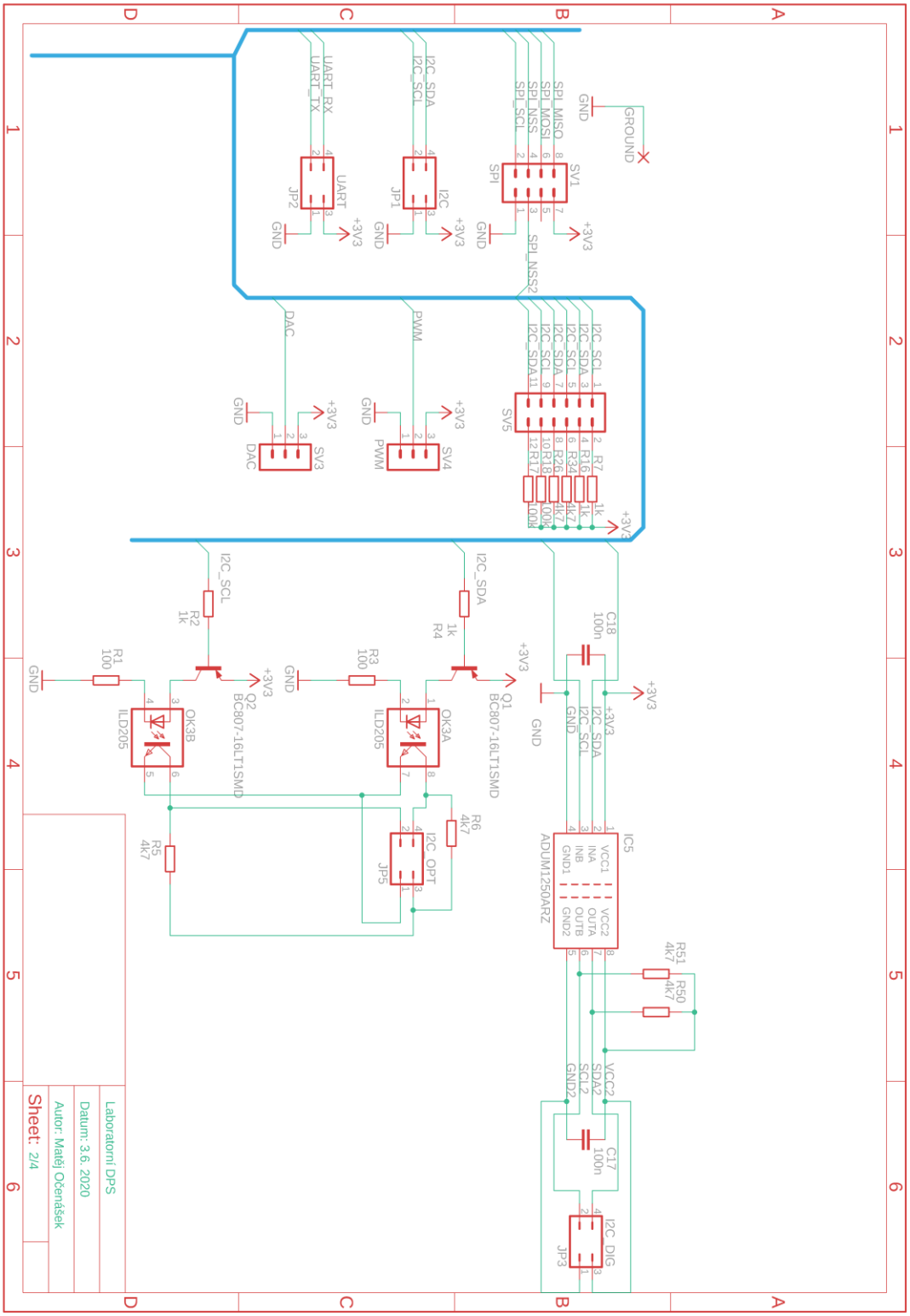
ESP8266 Arduino	PIN	PORT
Reset	1	D0
ADC (A0)	2	PA1
D0	3	RST
SPI_SCL (D5)	4	PB13
SPI_MISO (D6)	5	PB14
SPI_MOSI (D7)	6	PB15
SPI_NSS3 (D8)	7	PB6
VCC_1	8	+3,3V
VCC_2	9	NC
GND	10	GND
LED_2 (D4)	11	PA12
LED_1 (D3)	12	PC6
I2C_SDA_2 (D2)	13	PA10
I2C_SCL_2 (D1)	14	PA9
RX	15	NC
TX	16	NC

C - SCHÉMA A LAYOUT

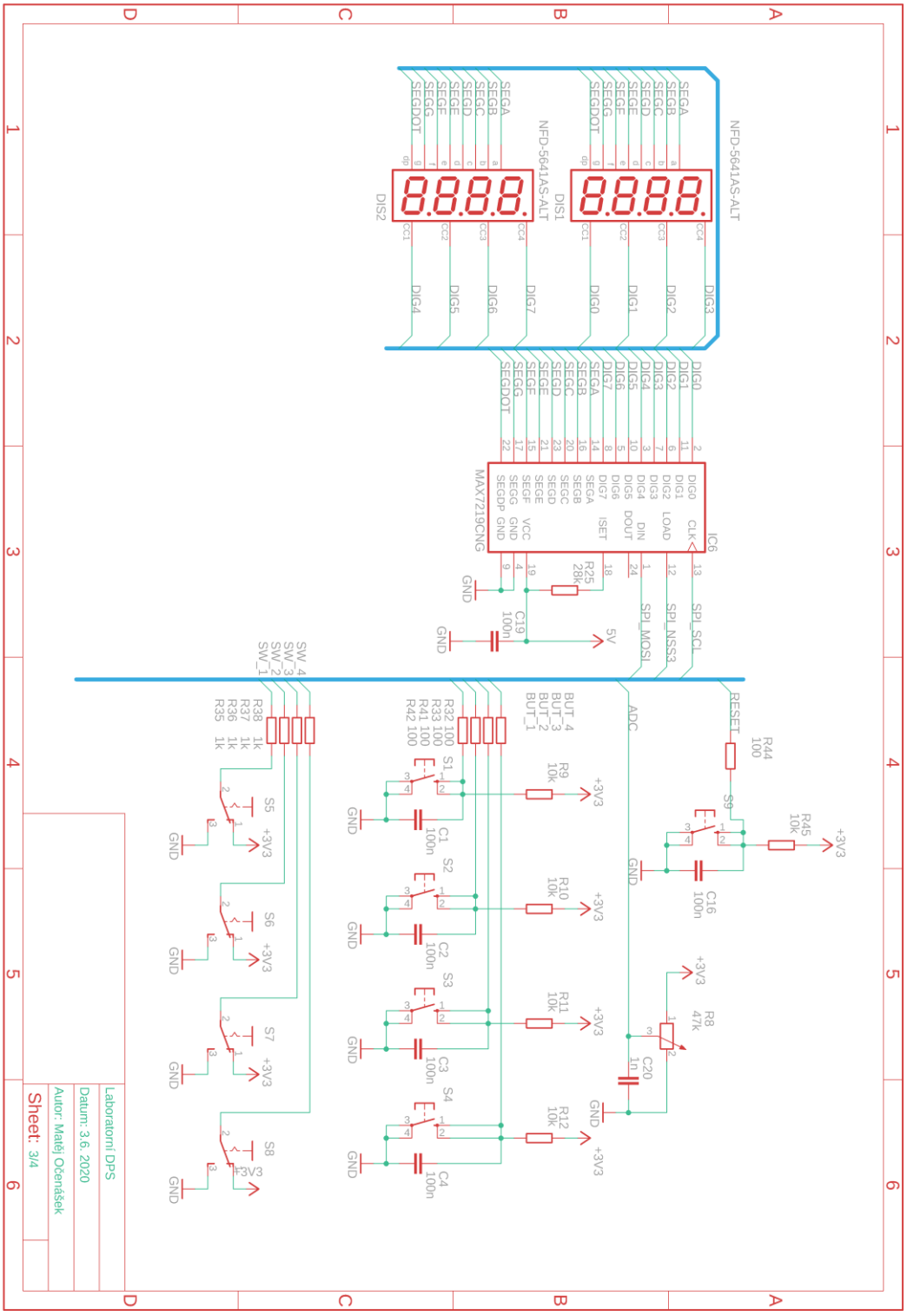
C.1 Schéma část 1/4



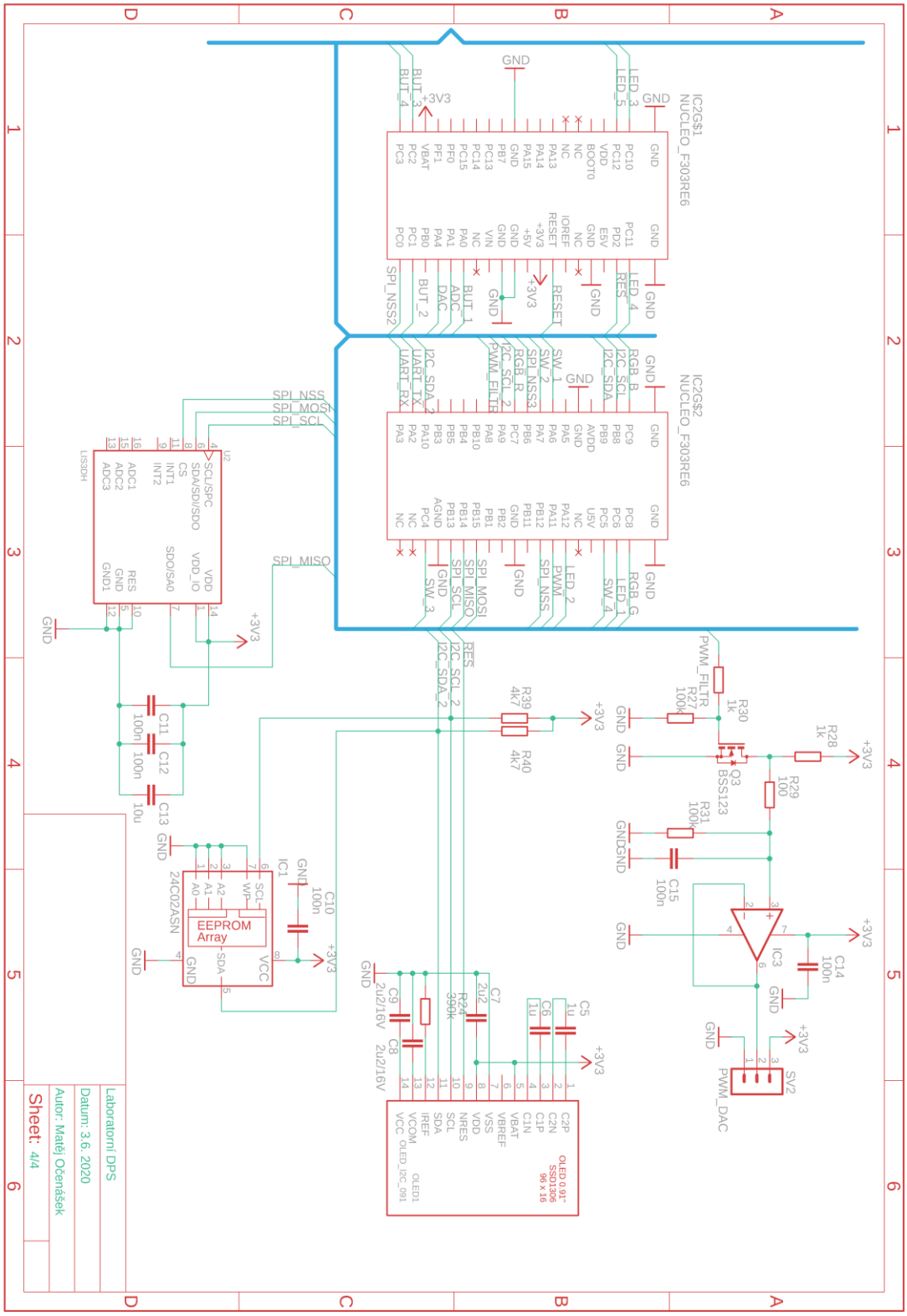
C.2 Schéma část 2/4



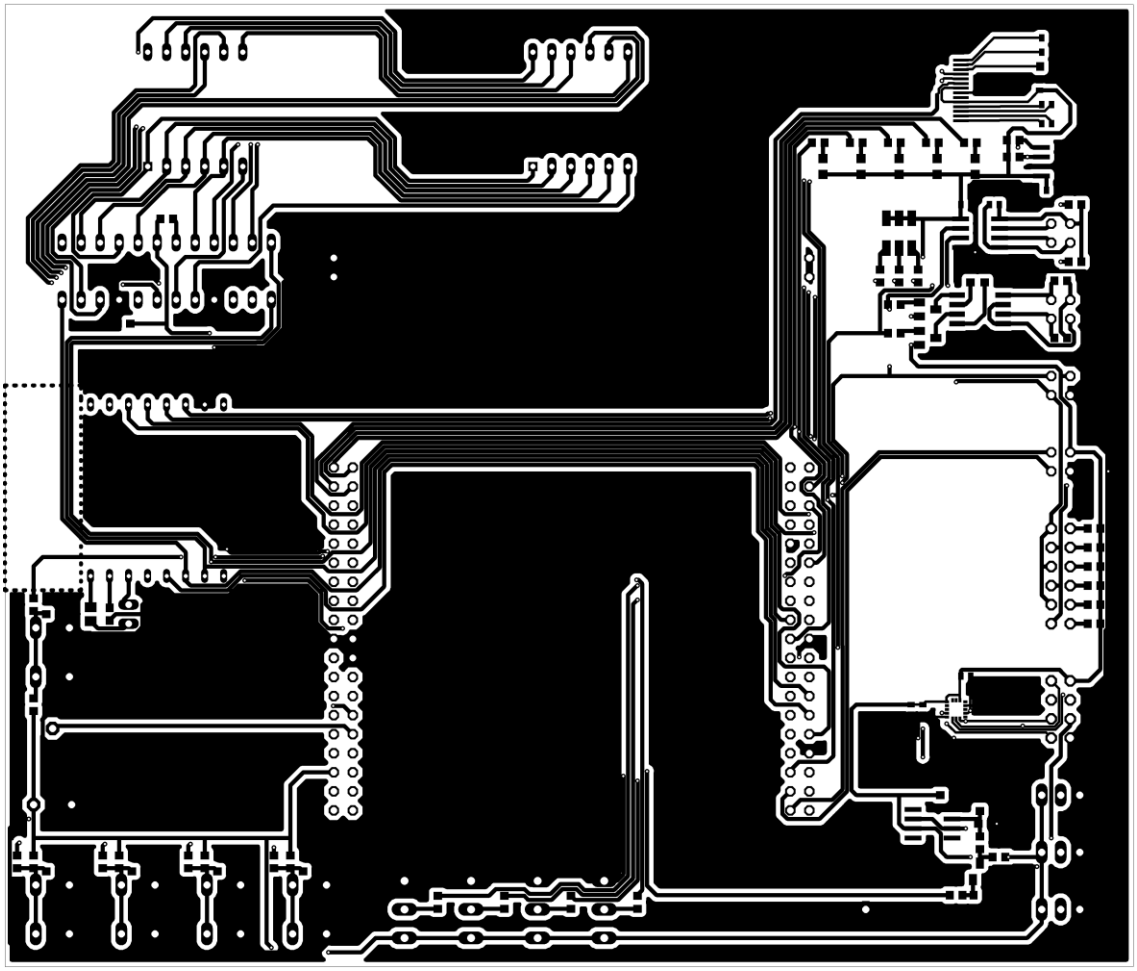
C.3 Schéma část 3/4



C.4 Schéma část 4/4

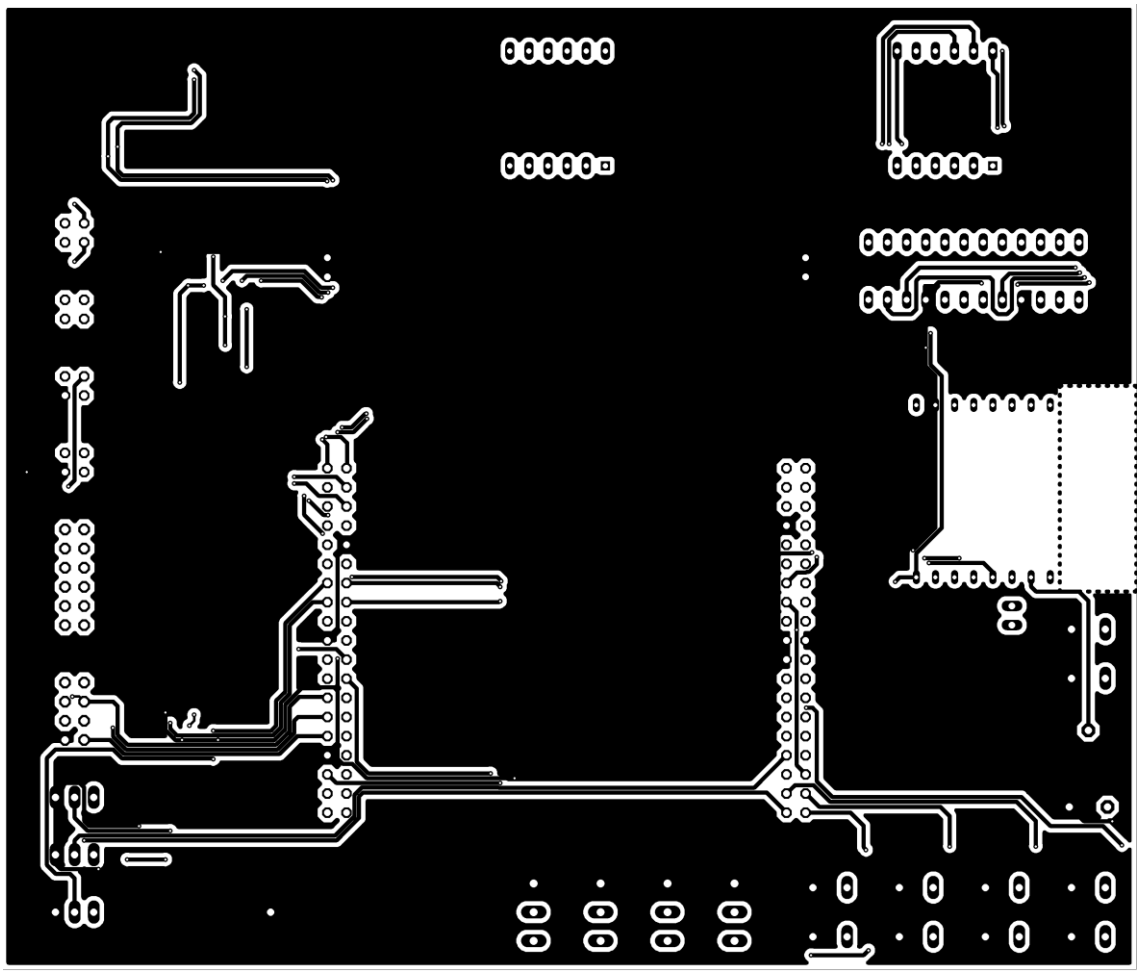


C.5 DPS motiv ze shora



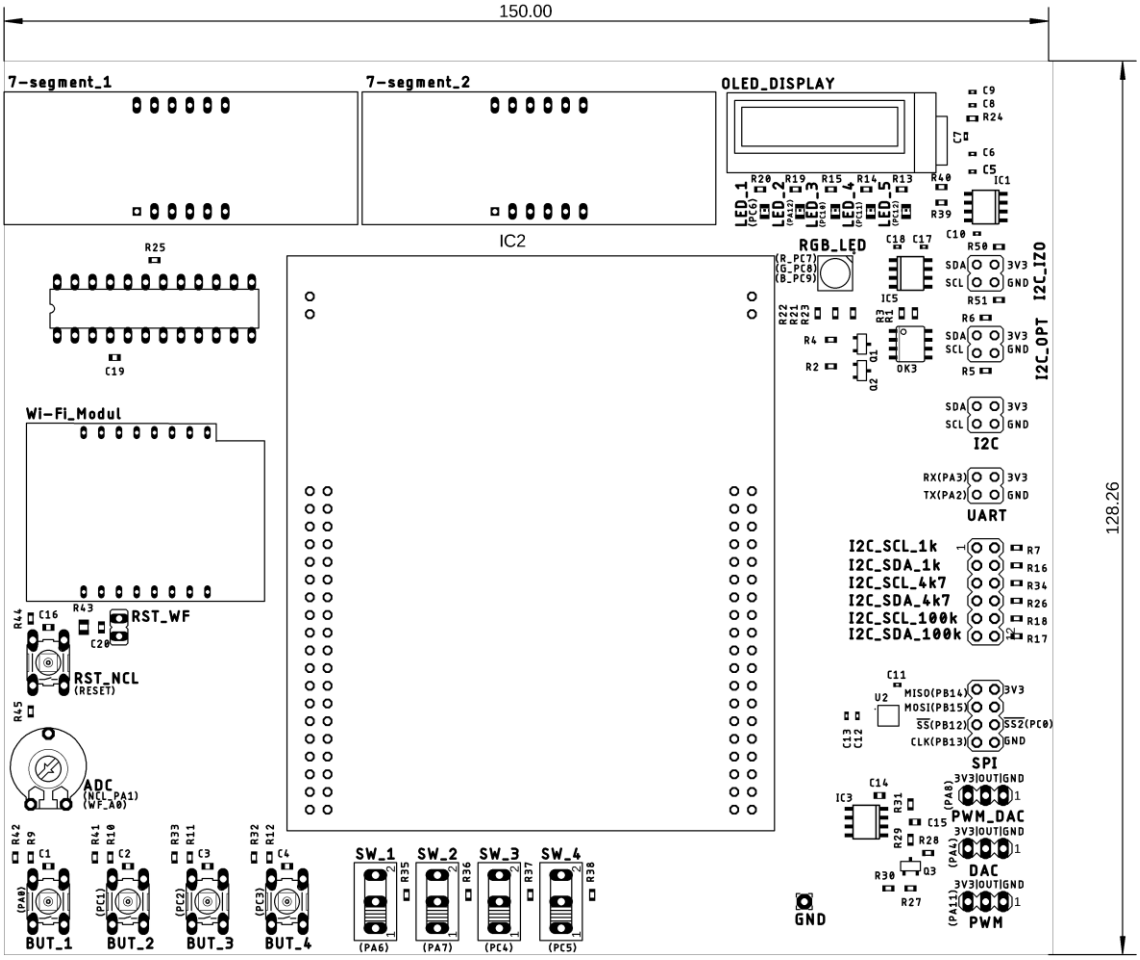
Měřítko 1:1, Rozměry 150 x 128,26 mm

C.6 DPS motiv ze spoda



Měřítko 1:1, Rozměry 150 x 128,26 mm

C.7 DPS osazovací plán



Měřítko 1:1, Rozměry 150 x 128,26 mm