VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

SIMULÁTOR VÝSTUPŮ MINIPOČÍTAČE HP 3000

HP 3000 MINICOMPUTER OUTPUT SIMULATOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB KLÁZAR**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. RICHARD RŮŽIČKA, Ph.D., MBA**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Klázar Jakub**
Program: Informační technologie
Název: **Simulátor výstupů minipočítače HP 3000**
HP 3000 Minicomputer Output Simulator
Kategorie: Vestavěné systémy
Zadání:

1. Seznamte se s uspořádáním a činností minipočítače řady HP 3000, zaměřte se na vzhled uživatelského rozhraní a projevy jeho periferií. Vzorem budiž systém, který je dochován v muzeu výpočetní techniky FIT.
2. Popište a zdokumentujte vhodným způsobem typické projevy jeho činnosti, zejména výstupy jeho uživatelského rozhraní.
3. Zvolte vhodnou desku s mikrokontrolérem a na ní implementujte vestavěný systém, který bude svými výstupy simulovat výstupy minipočítače HP 3000.
4. Výstupy navrženého vestavěného systému zobrazte na připojeném terminálu nebo jiných vhodných indikačních prvcích a na vhodně zvolených aplikacích ověřte jeho funkci.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Růžička Richard, doc. Ing., Ph.D., MBA**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 30. července 2021
Datum schválení: 30. října 2020

Abstrakt

Tato práce ukazuje postup návrhu vestavěného zařízení, které simuluje výstupy minipočítače HP 3000. Simulace spočívá ve výpisu znaků na terminál a ovládání kontrolek minipočítače. Instrukce k průběhu simulace jsou načítány z SD karty se souborovým systémem FAT32. Výsledkem je zařízení, které lze jednoduše použít pro oživení minipočítače HP 3000, nebo minipočítačů s rozhraním jemu podobným.

Abstract

This thesis shows the procedure of designing an embedded device that simulates the outputs of the HP 3000 minicomputer. The simulation is based on writing characters to the terminal and controlling the indicators of the minicomputer. Instructions of the simulation process are read from an SD card with the FAT32 file system. The result is a device that can be easily used to revive HP 3000 minicomputer or minicomputers with an interface similar to it.

Klíčová slova

minipočítač HP 3000, simulátor, ATmega1284P, vestavěné zařízení, deska tištěných spojů, SD karta, souborový systém FAT32.

Keywords

minicomputer HP 3000, simulator, ATmega1284P, embedded device, printed circuit board, SD card, FAT32 filesystem.

Citace

KLÁZAR, Jakub. *Simulátor výstupů minipočítače HP 3000*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Richard Růžička, Ph.D., MBA

Simulátor výstupů minipočítače HP 3000

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením docenta Richarda Růžičky. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jakub Klázar
2. srpna 2021

Poděkování

V těchto pár řádcích textu chci uchovat
poděkování panu doc. Richardu Růžičkovi,
za možnost na tomto projektu pracovat,
za stvoření nápadu na tuto práci,
za pomoc, za přístup, za podporu.

Své místo zde mají i moji rodiče,
protože jen táta a máma,
jsou s náma.

A pár přátel stačí mít,
pak stojí za to žít.
Protože Patrik Šimůnek,
ten umí za to vzít.

Obsah

Kapitola 1

Úvod

Kdekdo by řekl, že historická výpočetní technika nemá v dnešní době žádný přínos. Dnešní počítače jsou rychlejší, výkonnější, menší a celkově ve všech ohledech lepší. Přitom jsou neodmyslitelnou součástí vývoje a pokroku, díky kterému má lidstvo technologie, jaké známe dnes. Tyto velké a těžké stroje určovaly směr, kterým se udávaly další a novější zařízení. Ikdyž se to tak na první pohled nezdá, každé moderní zařízení v sobě nese pozůstatky těchto težkopádných strojů. Podobně tak, jako má dnešní závodní automobil stejně kulatá kola, jako vůz tažený párem koní ve středověku. Je smutné, že se na tyto technologické stařík zapomíná. Ukázka toho, jak vypadaly a fungovaly počítače pár desítek let zpátky, není pouze obecně zajímavá. Vznikne tak i podnět k zamyšlení se nad technologickým pokrokem. A zároveň má tato demonstrace vzdělávací hodnotu, která není úplně zanedbatelná. Máme-li možnost vdechnout těmto železným kostkám opět trochu eletrického proudu do jejich elektronických obvodů, udělejme to.

Na Fakulně informačních technologií v Brně se dochoval minipočítač HP 3000 z roku 1980, a dokonce ve velmi dobrém stavu. Ačkoliv v jeho názvu stojí "minipočítač", malý rozhodně není. V druhé polovině 20. století mělo slovo "mini", v kontextu výpočetní techniky, trochu jiný rozměr než dnes. Tento stroj se nachází na chodbě fakulty v prosklené vitríně, která je dlouhá asi deset metrů. Zde je přístupný procházejícím studentům, kteří si ho mohou prohlížet. V současném stavu je pouze vypnutý a nic nezobrazuje, protože udržovat tento stroj stále zapnutý má příliš mnoho nevýhod. A to je velká škoda, když je v podstatě funkční. Existuje zde možnost přistoupit k simulaci výstupů minipočítače s využitím pouze periferií zobrazující data. A tato možnost je důvod vzniku celé práce. Proč a jak minipočítač HP 3000 takto simulovat řeší podrobněji kapitola 2.

Ve zkratce, simulátor bude zařízení, které načítá pseudokód ze souboru na SD kartě a podle daných instrukcí rozsvěcí nebo zhasíná kontrolky a vypisuje znaky na terminál. Takto specifické zařízení je potřeba navrhnout od začátku až do konce. Jinak řečeno od výběru součástek, přes desku tištěných spojů a až po interpret pseudokódu instrukcí. Tato bakalářská práce provede čtenáře celým postupem vývoje takového zařízení, včetně popisu jak celý systém funguje uvnitř. Navíc si klade za cíl ukázat, že na řešení není potřeba nijak výkonný procesor, ani operační systém a ani přemnoho velkých knihoven. Ukázat co vše zvládne jeden malý miktrokontrolér a pár integrovaných obvodů kolem. Co vše lze v dnešní době vytvořit, když má člověk přístup k počítači a internetu.

Celý postup vývoje je komplikovaný, jelikož prochází skrz řadu oborů. V základu lze postup rozdělit na 3 části: návrh konceptu zařízení, vytvoření hardware a naprogramování software. Koncept zařízení je podrobně rozebrán v podkapitole 2.3. Hardwarové části projektu se věnuje kapitola 3. Kde je popsán výběr řídícího mikrokontroléru a dalších nutných součástek pro komunikaci s periferiemi minipočítače HP 3000. Především se tato kapitola věnuje zapojení všech součástek do jednoho funkčního celku. Softwarovou část projektu popisuje kapitola 4. Všechna funkcionalita je implementována ručně a tím pádem neexistuje žádná černá skříňka v podobě knihovny, která sice funguje, ale nikdo neví jak. Kapitola řeší podrobně zasílání příkazů pro SD kartu, inicializaci SD karty, načítání bloků dat, rozbor souborového systému FAT32 a čtení souboru. Konec kapitoly se věnuje samotnému interpretování načteného pseudokódu a způsobu vykonávání jednotlivých instrukcí. Kapitola 5 popisuje výsledky této práce a testování vzniklého simulátoru. A v příloze B je sepsán návod jak simulátor obsluhovat.

Kapitola 2

Simulace chování minipočítače

2.1 Motivace

Hlavním popudem k vytvoření celé bakalářské práce bylo oživení starého minipočítače HP 3000. Avšak takovým způsobem, aby na první pohled vypadal, že opravdu funguje a zároveň nevyžadoval složitou údržbu a nespotřebovával velké množství energie. Tento starý stroj se nachází, jak již bylo zmíněno v úvodu, na chodbě fakulty za skleněnou vitrínou vedle posezení, kde studenti často tráví čas, viz obrázek 2.1. Nemyslím si ale, že mu někdo věnuje příliš pozornosti i přesto že sedí přímo vedle. To je škoda, protože takový historický kousek si svoji část pozornosti zaslouží. Ovšem co kdyby se kontrolky na discích rozsvěcely a na terminálu se vypisoval výstup libovolného programu. Většina studentů určitě zpozorní a možná zbystří i procházející zamyšlený student. Díky tomu si třeba uvědomí, že mezi jeho mobilním telefonem, do kterého celou cestu chodbou koukal, a tímto starým počítačem je obrovský rozdíl a zároveň jsou si uvnitř stále trochu podobní.

Obrázek 2.1: Kompozice s minipočítačem HP 3000 na chodbě fakulty

2.2 Projevy minipočítače

Cíl je jasný, je potřeba oživit tento kousek historie pro znásobení prezentační hodnoty. Každého napadne prvoplánový způsob, jak tohoto cíle dosáhnout a to spuštěním minipočítače tak jak je. Fakulta vlastní všechny potřebné návody a manuály, které navíc existují také v naskenované formě na stránce [1], takže teoreticky je to možné. Nebyl by to ovšem jednoduchý úkol. Tyto staré stroje jsou komplexní, poruchové a chyby se hledají těžko. I za předpokladu, že by se podařilo minipočítač zprovoznit, přicházejí nové problémy. Energie, tyto stroje spotřebovávají velké množství energie, přece jenom pocházejí z doby kdy jiné možnosti nebyly. Další problém je program. Počítač musí něco vykonávat, aby vytvářel nějaký zajímavý výstup. Minipočítač HP 3000 je za zamčenou skleněnou vitrínou, tudíž s ním nelze nijak interagovat. Bylo by potřeba vytvořit skript, který by spouštěl výpočty, programy, příkazy nebo cokoliv co generuje výstup a zaměstnává počítač. Tento skript vytvořený přímo pro minipočítač HP 3000 by opět vyžadoval značného úsilí. Problémů je více než výsledného užitku.

Avšak pro prezentační účely není nutné minipočítač spouštět celý, stačí zprovoznit pouze ty části, které se nějakým způsobem projevují, když je minipočítač spuštěný. Na obrázcích 2.2 a 2.3 je vidět disk a terminál minipočítače. Tyto části minipočítače obsahují komponenty, které nějakým způsobem interagují s uživatelem tj. něco zobrazují. Aby minipočítač HP 3000 zdánlivě vypadal že běží, stačí když budou fungovat pouze tyto části. Zobrazující část pevného disku je pouze tištěný spoj s množinou kontrolek. Stačí ovládat jednotlivé kontrolky a nebo ještě lépe, na místo původní žárovičky vložit vlastní LED diodu připojenou přímo k simulátoru. Tudíž ani ovládací panel, ani disky ve skutečnosti nemusí být připojené a funkční.

Bohužel terminál takto jednoduše obejít nelze. V terminálu je zobrazujícím prvkem CRT obrazovka, aby pracovala, tak musí fungovat celý terminál. Exituje možnost výměny staré CRT obrazovky uvnitř terminálu za např. LCD display. Tato výměna by ovšem byla vidět a nový display by vypadal umělě, přeci jen stará CRT obrazovka má svůj specifický obraz, který stojí za to ho zachovat. Terminál používá pro komunikaci sériovou linku RS232, na které naslouchá a příchozí znaky vypisuje. Závěr je takový, že funkční a zapnutá bude muset být pouze jediná část minipočítače a to terminál.

Obrázek 2.2: Pevný disk HP 7925

Obrázek 2.3: Terminál HP 2645A

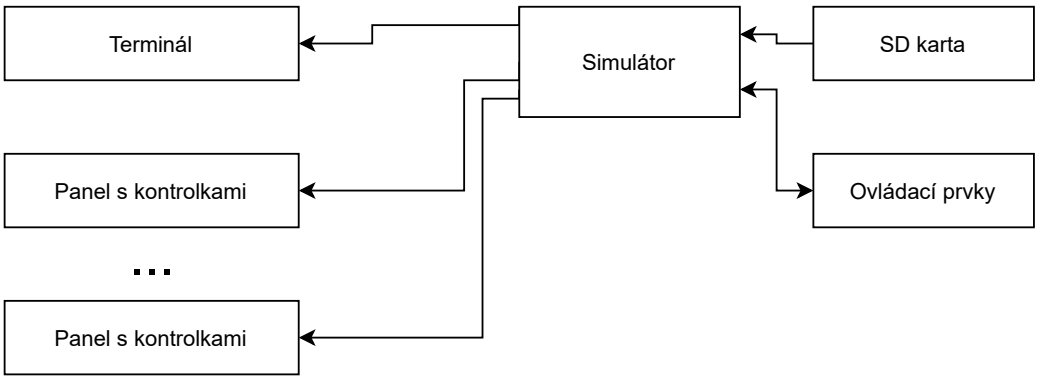
2.3 Koncept simulátoru

Hlavnímy výstupy simulátoru budou konektory pro terminál a připojení LED diod. Pro připojení terminálu stačí pouze připravit na simulátoru konektor DB-25, ke kterému se připojí terminál k tomu určeným kabelem. Jakým způsobem terminál komunikuje podrobněji popisuje kapitola 3. Pro kontext konceptu simulátoru nyní stačí pouze informace o konektoru. Připojení LED diod z logiky věci vede ke konektoru pro připojení množiny párů vodičů. Každý pár povede od simulátoru k jedné LED diodě na místě kontrolky v minipočítači.

Mimo to simulátor potřebuje vědět, dle jakého vzoru je ovládat. Jaká data odesílat na terminál, jak a kdy jakou kontrolku rozsvítit nebo zhasnout. Tyto instrukce lze naprogramovat přímo do kódu simulátoru. Tím se zařízení simulátoru stane použitelné pouze na tento specifický minipočítač a zobrazovaná data nepůjdou změnit jinak než přeprogramováním celého zařízení. To by byla škoda, lepší způsob je vytvořit univerzálnější zařízení. Takové v kterém lze jednoduše měnit zobrazovaná data, případně simulovat i jiný stroj podobného typu. Instrukce k tomu jaká data zobrazovat, musí simulátor logicky načítat z externího média. Jako médium se nabízí USB flash klíčenka nebo SD karta. Rozhodl jsem se pro SD kartu, protože pro komunikaci používá sběrnici SPI, která je příjemnější na implementaci než USB protokol. Navíc SD kartu je možné zasunout do zařízení na rozdíl od USB flash klíčenky, která bude vždy vyčnívat. Ohledně souborového systému je pro podobné aplikace vhodné použít souborový systém FAT32. Ten je podporován většinou operačních systémů a zároveň není komplikovaný pro čtení i obyčejným mikrokontrolérem.

Stávající koncept umí načíst instrukce a podle nich simulovat minipočítač. Zbývá k vyřešení poslední záležitost, konkrétně ovládání samotného simulátoru. Není nutné aby simulátor bežel neustále. Stačí když se spustí jednou za definovaný čas a poté se uspí a odpojí terminál od napájení. Za daný čas se opět spustí, zapne terminál a takto stále dokola. Tento čas lze definovat v souboru s pseudokódem. Jako ovládací prvky pro zobrazení stavu zařízení, ruční spouštění simulace, ruční uspání zařízení atd. skvěle poslouží tlačítka a LED diody. Ovšem jako spouštěcí prvek může také posloužit např. mincovník a tak spouštět simulaci vhozením mince či žetonu. Díky tomu může být simulátor nasazen i v aplikacích pro veřejnost.

Celý koncept shrnuje obrázek 2.4. Simulátor načítá instrukce pro simulaci z SD karty. Podle těchto instrukcí ovládá LED diody v panelech minipočítače a odesílá znaky na terminál. Simulace probíhá jednou za čas definovaný v souboru s instrukcemi pro simulaci. A celý simulátor je ovládaný pomocí ovládacích prvků.

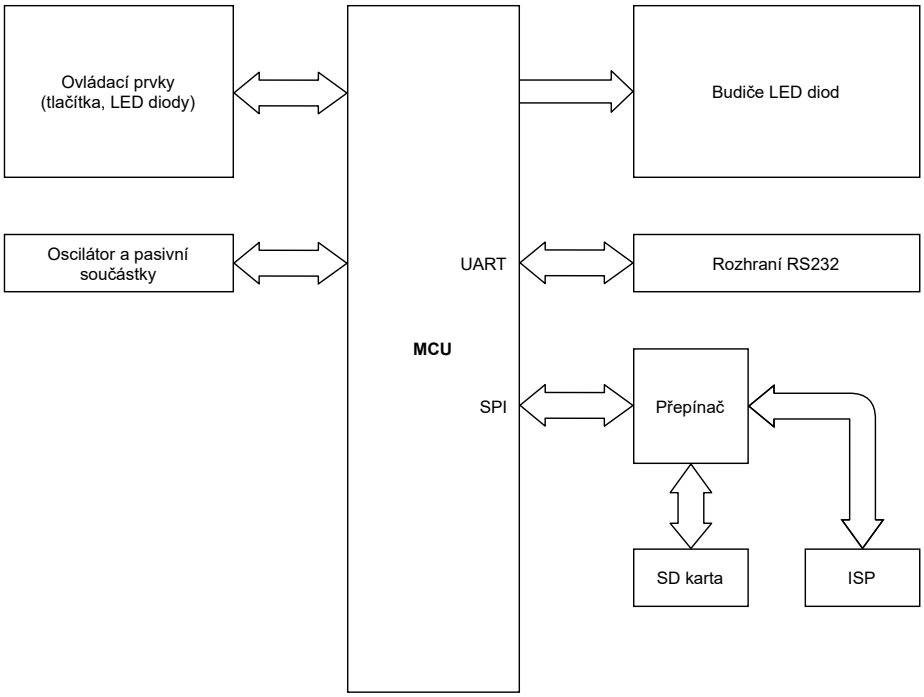


Obrázek 2.4: Blokový koncept simulátoru

Kapitola 3

Návrh hardwarové části simulátoru

První část výroby simulátoru spočívá v navržení desky tištěných spojů, jinak řečeno hardwaru. Než je možné začít, musí se zohlednit co vše zařízení umí, jaké má vstupy a výstupy. Podle toho se rozhodnout jakým způsobem bude celé zařízení fungovat a vytvořit si z konceptu blokové schéma celého zapojení, viz obrázek 3.1. Pro takovéto typy zařízení z logiky věci vyplývá schéma s MCU (mikrokontrolérem) uprostřed a připojené periferie kolem.



Obrázek 3.1: Blokové schéma zapojení simulátoru

Protože terminál používá pro komunikaci standard RS232, který využívá jiné napěťové úrovně než MCU (viz dále), tak je k MCU připojený blok vykonávající převod těchto úrovní. Mikrokontrolér v zařízení musí být možné naprogramovat tj. nahrát do něj zkompilovaný program který vykonává. K této akci se u mikrokontrolérů využívá ISP, což znamená způsob naprogramování mikrokontroléru, který je již v zařízení zabudován. Tudíž není nutné mikrokotrolér vyjmout a naprogramovat ho mimo zařízení. ISP využívá sběrnici SPI a tak i SD karta. ISP a SD karta by se bohužel rušily navzájem, takže je potřebný přepínač pro přepnutí připojené periferie k SPI. Přepínač bude nutné využít pouze pro naprogramování zařízení nebo během ladění, v běžném provozu bude přepínač stále přepnut do polohy komunikace s SD kartou. Blok "Oscilátor a pasivní součástky" bude obsahovat potřebné součástky, jejichž zapojení vychází z typického zapojení MCU dané výrobcem (viz dále). Bloky "Ovládací prvky"a "Budiče LED diod" jsou výsledkem hardwarového návrhu vzhledem ke konceptu zařízení.

3.1 Výběr součástek

Než je možné vytvářet zapojení, je potřeba zvolit nejvhodnější součástky co se týče parameterů i fyzického provedení. Proto tato podkapitola řeší průběh výběru klíčových součástek, zpravidla integrovaných obvodům. Vetšinu součástek jsem vybíral přímo v internetových obchodech. Takové obchody nabízejí propracované filtrování, díky kterému se lze dopracovat k nejvhodnější součástce. Rutinní postup pro každou součástku znamenal filtrování pouze naskladněného zboží a seřazení podle nejnižší ceny. Avšak zásadní pro výběr správné součástky je požadovaná funkce a parametry, které jsou pro každou součástku individuální. Vždy jsem se nakonec musel rozhodovat mezi desítkami součástek se stejnými parametry a funkcí. Vybral jsem tu nejlevnější s nejlépe zpracovaným datovým listem. Konkrétnímu rozboru a zapojení součástek dle datových listů popisuje až další podkapitola 3.2.

Mikrokontrolér

První součástkou a to skutečně esenciální součástkou je mikrokontrolér. Srdce celého simulátoru. Vybírat MCU je potřeba s rozmyslem, především vzhledem k periferiím, které budou k MCU připojené. Na trhu je mikrokontrolérů mnoho. Nejprve je důležité stanovit si jasné požadavky na MCU a tím zmenšit okruh výběru. Pro tuto aplikaci jsou požadavky následující:

- Podpora sběrnic SPI a UART. Pro obsluhu SD karty a RS232.
- Alespoň 4 porty (1 port = 8 vstupně výstupních pinů). Vzhledem k počtu připojených periferií.
- Více než 8kB SRAM paměti. 512B je nutných pro načítání bloku dat z SD karty. Další přibližně 1kB je potřeba vyhradit pro globální proměnné. A vzhledem k situaci, že simulátor bude interpretovat pseudokód, je dobré ponechat dostatek místa pro proměnné na zásobníku, tedy alepoň 6kB.
- Vyrábí se v THT pouzdru. Pro možnost výměny, případně z důvodu naprogramování MCU mimo zařízení.
- Samozřejmě dané MCU lze koupit.

Jelikož v této aplikaci není potřeba nikterak závratný výkon, proto je možné vyřadit ARM mikrokontroléry a jiné výkonnější MCU, jejichž potenciál by zůstal zbytečně nevyužitý. Po zkoumání různých typů MCU jsem se na nakonec úchylil k výrobci Atmel a architektuře 8-bit AVR, protože s ní mám jako autor dobré zkušenosti. Popis architektury 8-bit AVR je mimo obsah této práce. Nicméně tento projekt by bylo možné zpracovat také pomocí jiných, možná desítek, mikrokotrolérů. Rozhodnutí musí být učiněno a tak jsem vybral mikrokontrolér `ATmega1284-PU` [7] z rodiny 8-bit AVR, který splňuje daná kritéria a mám s ním pozitivní zkušenosti.

Budič LED diod

S výběrem budiče LED diod souvisí rozhodnutí o počtu obsluhovaných LED diod. Konektor pro připojení LED diody má svojí velikost, proto je potřeba zvolit takový počet, aby konektor nebyl nevhodně velký, ale zároveň simulátor dokázal obloužit všechny potřebné LED diody. Navíc je vhodné, aby byl tento počet dělitelný počtem výstupů z jednoho budiče. Počet výstupů budiče je zpravidla 16. Takže se nabízí počet 32, který s rezervou zvládne obsloužit všechny potřebné LED diody.

Při tomto počtu připojených LED diod, by ani nebylo možné použít jednoduché připojení na porty MCU. Z důvodu zaplnění všech volných vstupně výstupních pinů a především to nedovoluje elektronická charakterstika mikrokontroléru. Ta dovoluje maximální proud čipem `200mA`, to znamená (za předpokladu odběru jedné LED diody `20mA`), že by mohlo v jeden moment svítit pouze 10 LED diod. To je omezení, které nedává smysl. Proto je nutné LED diody opravdu obsluhovat externím budičem. Samotný budič funguje na principu posuvného registru ovládající zdroje proudu pro každou připojenou LED diodu. Díky proudovému zdroji není nutné řešit ochranné rezistory pro LED diody a je možné diody bez starostí připojit přímo k simulátoru. Z nabídky jsem vybral budič `SCT2026` [13], který přináší dostačující funkcionalitu za dobrou cenu. Pro úsporu místa na výsledné desce jsem zvolil provedení v SMD pouzdru.

RS232 převodník

Ke komunikaci s terminálem se používá zmíněné RS232 [15] [19]. Tento standard, mimo jiné, definuje napěťové úrovně pro jednotlivé logické úrovně sběrnice UART, viz tabulka 3.1. Jak je vidno, nelze jednoduše připojit výstup sběrnice UART z MCU (který používá napěťové úrovně TTL) k terminálu, je nutné vložit převodník. Takovýchto převodníků není příliš mnoho, těch které lze rozumně koupit je ještě méně. Pro tyto účely je používaný například integrovaný obvod `MAX232`. Já jsem nakonec vybral integrovaný obvod `ICL3221E` [11], který má stejnou funkčnost ale nižší cenu. Integrovaný obvod jsem opět volil v SMD provedení, obecně všechny integrované obvody jsem vybíral v SMD provedení, krom MCU.

Log. úroveň	TTL (UART)	RS232
log. 1	5V	-3V až -25V
log. 0	0V	3V až 25V

Tabulka 3.1: Napěťové úrovně logických úrovní podle TTL (UART) a RS232

Regulátor pro 3.3V

Celý simulátor bude napájený `5V`, jelikož toto napětí je hojně používané pro podobné aplikace. MCU dokonce jiné napětí použít neumožňuje, takže není potřeba vymýšlet žádné složitosti. Ovšem SD karta pracuje s napájecím napětí `3.3V`, proto se někde na desce musí nacházet zdroj tohoto napětí. A jelikož SD karta není konstantní zátěží, jednoduchý napěťový dělič nepřipadá v úvahu a je nutné zvolit spolehlivější řešení. Tímto řešení je regulátor napětí, přímo vyrobený pro převod `5V` na `3.3V`. Vybral jsem regulátor `MCP1603T` [6], protože jeho maximální zatížení postačuje pro napájení SD karty a zároveň se prodává za lepší cenu než ostatní podobné regulátory. Při výběru regulátoru hrála roli i dostupnost dalších pasivních součástek, potřebných pro zapojení doporučené výrobcem.

Převodník úrovní pro SD kartu

MCU s SD kartou komunikuje pomocí sběrnice SPI. Jelikož je SD karta napájena `3.3V`, tak z logiky věci opět vzniká napěťový konflikt podobně jako na sběrnici UART. Jak vidí napěťové úrovně vzhledem k logickým úrovním MCU a jak SD karta, ukazuje tabulka 3.2. Tudíž před SD kartou musí figurovat převodník napěťových úrovní mezi `5V` a `3.3V`. Vybral jsem integrovaný obvod `MAX3392E` [5]. Jde o univerzální převodník libovolných napěťových úrovní, podle toho jaká dvě napájecí napětí jsou připojena. Konkrétně `MAX3392E` provádí převod na 3 vodičích ve směru z vyššího napětí na nižší a na 1 vodiči v opačném směru. Pro sběrnici SPI je tato konfigurace ideální. Řada `MAX339x` nabízí i různé další konfigurace směrů převodu.

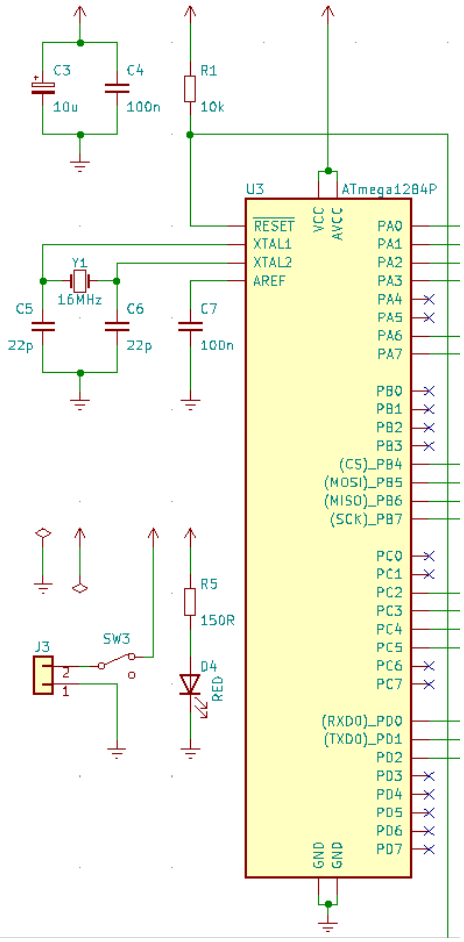
Log. úroveň	MCU	SD karta
log. 1	5V	3.3V
log. 0	0V	0V

Tabulka 3.2: Napěťové úrovně logických úrovní na sběrnici SPI podle MCU a SD karty

3.2 Elektronické schéma

V tuto chvíli již nic nebrání přistoupit k samotnému zapojení celého simulátoru. Pro návrh elektronického schématu a desky tištěných spojů jsem použil program KiCAD. Tento návrhový program je zdarma a pro tento druh projektu poslouží velmi dobře. Tato kapitola rozebere podrobně jednotlivé části zapojení. Celé elektronické schéma se nachází v příloze na obrázku C.1.

Zapojení mikrokontroléru



Obrázek 3.2: Část schématu s MCU

Část schématu týkající se MCU znázorňuje obrázek 3.2. Tato část schématu obsahuje navíc součástky, které přímo nesouvisí s MCU, jsou ovšem důležité pro funkci desky jako celku. Dominantou celého schématu je samotné MCU `U3`, k jehož portům jsou připojeny všechny periferie z ostatních částí schématu.

Vlevo nahoře se nachází stabilizační kondenzátory `C3` a `C4`. Na výsledné desce mají místo blízko napájecího konektoru a jejich funkcí je odfiltrování šumu zdroje a udržování stabilníno napájecího napětí. `C3` stabilizuje šum nižších frekvencí a `C4` vyšší frekvence. Hodnoty `10`μ`F` a `100`η`F` jsou běžně používané v podobných aplikacích.

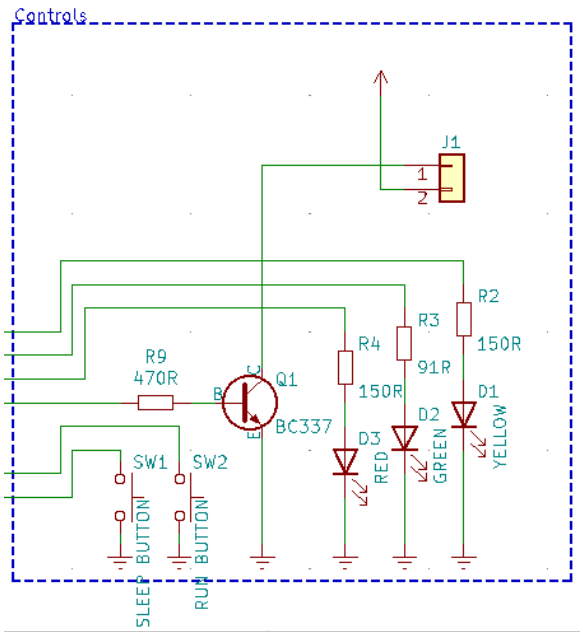
Vedle těchto kondenzátorů je pull-up rezistor `R1` pro pin $\overline{\texttt{RESET}}$, který je spínaný log. 0. Hodnota může být dle datového listu MCU i vyšší, ale už při `10k`Ω je proud tímto rezistorem zanedbatelný. K pinu $\overline{\texttt{RESET}}$ je připojený vývod z ISP, který pro naprogramování MCU tento pin využívá. Rezistor `R1` udržuje na pinu $\overline{\texttt{RESET}}$ hodnotu log. 1 a proto je reset MCU v běžném stavu zakázaný. Do resetu může MCU uvést pouze připojený ISP programátor.

Další stabilizační kondenzátor `C7` je připojený k pinu `AREF` pro analogový převodník. Analogový převodník se sice nepoužívá, ale z robustních důvodů je lepší tento kondenzátor přidat do zapojení. Jeho hodnota `100`η`F` je typická podle datováho listu,

K pinům `XTAL1` a `XTAL2` je připojen krystal `Y1` s kondenzátory `C5` a `C6`. Společně vytvářejí externí oscilátor pro frekvenci 16 MHz. `ATmega1284-PU` obsahuje vnitřní 8 MHz oscilátor, ale s externím oscilátorem je možné pracovat až na frekvenci 20 MHz. Je dobré využít této možnosti a zvýšit výkon MCU. Zapojení krystalu a kondenzátoru včetně jejich hodnot určuje datový list MCU.

V levém spodním rohu se nachází napájecí značky pro KiCAD, které mají pouze vnitřní účely pro kontrolní funkce KiCADu. Pod nimi je napájecí konektor `J3` s hlavím přepínačem `SW3`. Poslední nepopsanou součástkou je napájecí LED dioda `D4` s ochranným rezitorem `R5`, která svítí vždy, když je připojené napájení a hlavní přepínač přepnut do polohy zapnuto. Důvod a hodnotu rezistoru `R5` není potřeba vysvětlovat, pokud je napájecí napětí 5V.

Zapojení tlačítek, kontrolních LED diod a relé

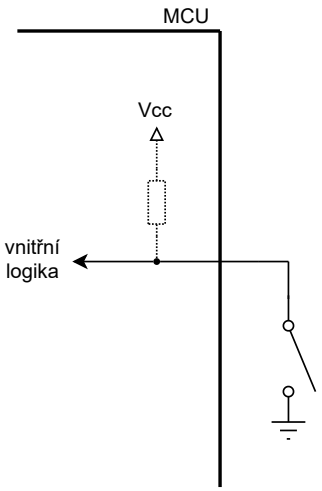


Obrázek 3.3: Část schématu s kontrolními prvky

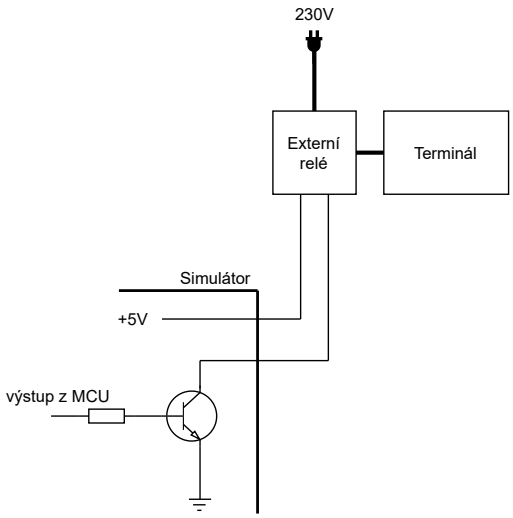
V této části schématu na obrázku 3.3 mají své místo kontrolní prvky. Kontrolní LED diody, ovládací tlačítka a tranzistor spínající napájecí réle terminálu. Tento blok je připojen na `PORT A` mikrokontroléru. LED diody `D1`, `D2` a `D3` mají své ochranné rezistory `R2`, `R3` a `R4`. Poněvadž zelená LED dioda `D2` pracuje na jiném napětí, je i hodnota rezistoru `R3` rozdílná. Význam jednotlivých LED diod řeší příloha B.

Tlačítka mohou spínat k vstupnímu pinu MCU log. 1 nebo log. 0. Vždy je ovšem nutné zajistit, aby byla při nestisknutém tlačítku připojena na pin MCU log. hodnota opačná. To řeší pull-up nebo pull-down rezistory pro tlačítko spínající log. 0 nebo log. 1. Navržená tlačítka `SW1` a `SW2` spínají log. 0, takže jsou potřeba pull-up rezistory. Avšak `ATmega1284-PU` nabízí softwarové zapnutí vnitřních pull-up rezistorů pro jednotlivé piny MCU, viz obrázek 3.4, proto není potřeba přidávat žádné rezistory.

Tranzistor `Q1` je v zapojení zvaném "otevřený kolektor". K externímu relé vedou 2 vodiče pomocí konektoru `J1`. Vodič `5V` a vodič spínaný tranzistorem k `0V`. Toto zapojení je univerzální a externí rélé může být libovolně zvoleno. Způsob zapojení externího relé lépe popisuje obrázek 3.5. Tranzistor má svůj bázový rezistor `R9` omezující proud bází. Rezistor má takovou hodnotu, aby bylo možné tranzistor plně otevřít a zároveň nedošlo k jeho destrukci.

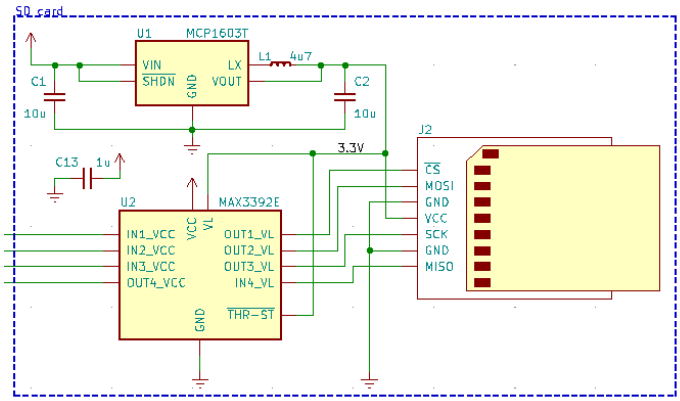


Obrázek 3.4: Vnitřní pull-up rezistor MCU. Čárkovaný rezistor je možné softwarově připojit nebo odpojit.



Obrázek 3.5: Zapojení simulátoru a extreního relé spínající napájení terminálu.

Zapojení SD karty



Obrázek 3.6: Část schématu s SD kartou

Zapojení s SD kartou ukazuje obrázek 3.6. Hlavním prvkem je zde konektor pro SD kartu `J2`. Konektor je zapojený s předpokladem, že SD karta bude komunikovat pomocí sběrnice SPI. Totiž SD karta nabízí i další způsob připojení, takzvaný SD mód. Tento mód přináší lepší výkon a rychlost přenosu dat, nicméně komplexností mnohonásobně převyšuje sběrnici SPI, proto mu nebude věnována pozornost. Rychlost a možnosti komunikace pomocí sběrnice SPI pro potřeby simulátoru zcela postačuje.

Krom konektoru se v této části schématu nachází dva intergrované obvody `U1` a `U2`. `U1` je regulátor napětí pro `3.3V`. Podle datového listu ho doplňují požadované součástky `C1`, `C2` a `L1`. Datový list určuje minimální hodnotu kondenzárů na 4.7μ`F`, maximální hodnotou je omezen pouze kondenzátor `C2` a to na 22μ`F`. Lze říci že čím větší hodnota kondezátorů, tím stabilnější výsledné napětí bude. Hodnota 10μ`F` tudíž s rezervou vyhovuje. Cívka `L1` má dle datového listu pro dosáhnutí nejlepších vysledků doporučenou hodnotu 4.7μ`H`. Nicméně je dovoleno použít cívku v rozmezí hodnot 3.3μ`H` až 10μ`H`. Funkci vývodů regulátoru `MCP1603` popisuje tabulka 3.3. Tato tabulka vysvětluje způsob zapojení, možná až na pin $\overline{\texttt{SHDN}}$, který je připojen stále na log. 1, protože regulátor nebude třeba vypínat. A zapojení cívky `L1` k pinu `LX`, které konkrétně určuje datový list.

Vývod	Funkce
`VIN`	Vstupní napětí (`5V`)
`VOUT`	Výstupní napětí (`3.3V`)
`LX`	Pin pro připojení cívky
$\overline{\texttt{SHDN}}$	Kontrolní pin pro vypnutí regulátoru přivedením log. 0
`GND`	Zem (`0V`)

Tabulka 3.3: Funkce vývodů regulátoru `MCP1603`

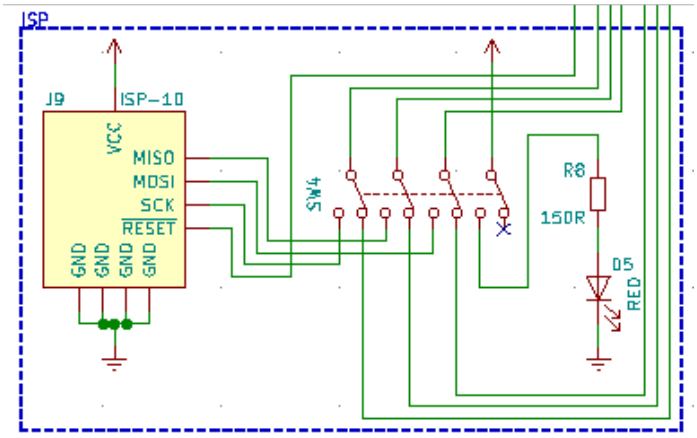
Výstupních `3.3V` je přivedeno jako napájecí napětí pro SD kartu a zároveň jako převáděné nízké napětí pro `U2`. `U2` převádí napěťové úrovně na sběrnici SPI přes kterou SD karta komunikuje s MCU. Tento převodník umí převádět napěťové úrovně pouze v jednom směru pro jednotlivý vodič. Převod provádí na prvních 3 vodičích z napětí `VCC` (`IN?_VCC`) na napětí `VL` (`OUT?_VL`) a na posledním vodiči opačně z `IN4_VL` na `OUT4_VCC`. SPI definuje směr toku dat na jednotlivých vodičích, tedy vodičích $\overline{\texttt{CS}}$, `SCK` a `MOSI` od MCU k SD kartě a na vodiči `MISO` od SD karty k MCU. Nyní již není potřeba dál vysvětlovat, jak musí být převodník zapojen pro správnou funkci. Význam ostaních pinů převodníku popisuje tabulka 3.4

Vývod	Funkce
VCC	Napájecí napětí vyšší (5V)
VL	Napájecí napětí nízké (3.3V)
GND	Zem (0V)
$\overline{\text{THR-ST}}$	Kontrolní pin pro zapnutí třetího stavu vysoké impedance na výstupních pinech převodníku přivedením log. 0

Tabulka 3.4: Funkce vývodů převodníku MAX3392E

Poslední nepopsanou součástkou v této části je stabilizační kondezátor C13. Tento kondezátor má funkci stabilizace napěťových výkyvů a chrání tím tak integrovaný obvod U2 před podpětím. Výkyvy v napětí můžou způsobovat integrované obvody s proměnlivým odběrem proudu, jako například zmíněný regulátor U1 nebo převodník napěťových úrovní pro RS232 probíraný dále. C13 má své místo na desce co nejblíže k převodníku U2.

Zapojení ISP



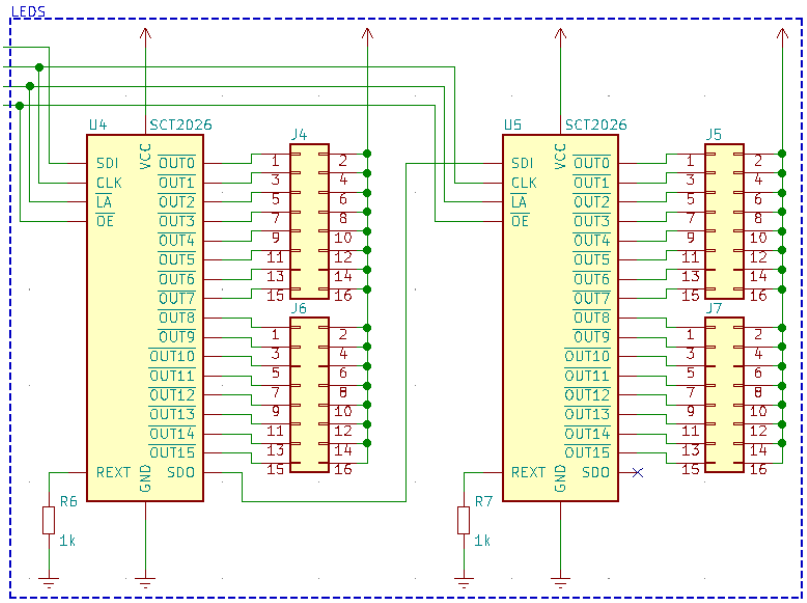
Obrázek 3.7: Část schématu s ISP

Část schématu se zapojením ISP na obrázku 3.7 musí brát v úvahu zmiňované sdílení sběrnice SPI SD kartou. Sběrnice SPI sice podporuje více připojených zařízení, ale pro robustní ISP je lepší odpojit od SPI sběrnice všechna ostatní zařízení, které by mohla sběrnici rušit. Ideálně mechanicky, přecejenom již blokové schéma naznačovalo mechanický přepínač. Zvolil jsem tedy jednoduché řešení pomocí 4 pólového přepínače SW4, přepínající zařízení připojené přes sběrnici SPI k MCU mezi konektorem ISP J9 a SD kartou. Vývod $\overline{\text{RESET}}$ není potřeba přepojovat, protože ho sběrnice nesdílí, tak ani vývod $\overline{\text{CS}}$ sběrnice SPI, který ISP vůbec nepoužívá.

Je důležité zmínit že vyvíjené zařízení je prototyp, který musí umožňovat ladění programu. Pro další sériovou výrobu přepínač postrádá smysl vzhledem ke skutečnosti, že by byl použit pouze jednou. Pro výrobu odladěného finálního zařízení je lepším řešením, ISP konektor úplně vynechat a vkládat do desky již naprogramované MCU.

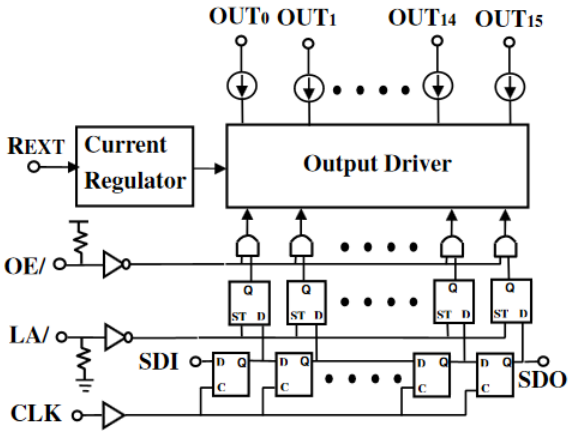
Před naprogramováním MCU pomocí ISP tak stačí pouze přepnout přepínač, naprogramovat MCU a opět přepnout zpátky do polohy pro připojení SD karty. Není nutno nijak manipulovat s MCU ani SD kartou. Zda je sběrnice SPI přepnuta do programovacího módu značí LED dioda D5 s ochanným rezistorem R8. Narozdíl od všech použitých LED diod je tato jako jediná v SMD provedení, poněvadž bude mít na desce své místo hned vedle přepínače.

Zapojení budičů LED diod



Obrázek 3.8: Část schému s budiči LED diod

Část schématu na obrázku 3.8 se zapojením budičů LED diod je triviální, ačkoliv se to na první pohled nemusí zdát. Budiče `U4` a `U5` jsou, zjednodušeně řečeno, pouze 2 posuvné registry zapojené do série. Kde log. 1 v posuvném registru zapínají přiřazené proudové zdroje pro jednotlivé LED diody. Výstup z budiče se připojuje na katodu LED diody, takže obrazně řečeno budič "nasává" proud přes LED diodu. Z toho plyne, že k výstupním konektorům pro LED diody `J4-7` je potřeba připojit také napájecí napětí. Budič poté zajistí, že každou LED diodou poteče nastavený proud. Lépe vnitřní zapojení popisuje blokové schéma přímo z datového listu na obrázku 3.9. Budiče jsou ovládány na běžných vstupně výstupních pinech, protože k sběrnice SPI mikrokontroléru je již připojena SD karta.



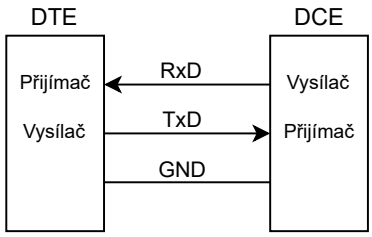
Obrázek 3.9: Blokové schéma integrovaného obvodu SCT2026, převzato z [13]

Pozornost je třeba věnovat pinu `REXT`. Tento pin pomocí připojeného rezistoru nastavuje požadovaný proud proudových zdrojů na jednotlivých výstupních pinech. Závislost hodnoty rezistoru na proudu výstupních pinů je dle datového listu následující:

$$R_{EXT} = 30 \left(\frac{630}{I_{OUT}(mA)} \right) \ \Omega \tag{3.1}$$

Drtivá většina LED diod potřebuje pro provoz `20mA`. To znamená, že po dosazení do rovnice, vychází potřebný odpor `945Ω`. Nejbližší dostupná větší hodnota rezistoru je `1kΩ`. Po zpětné kontrole dosazením do inverzní rovnice vychází výsledný proud cca `18,9mA`. Při takovém proudu se sice nepatrně sníží svítivost, avšak prodlouží se životnost LED diody, oproti proudu větším než `20mA`. Rezistory `R6` a `R7` proto mají hodnotu `1kΩ`. Jakým způsobem MCU konkrétně komunikuje s budiči je popsáno podrobněji v podkapitole 4.

Zapojení RS232 rozhraní

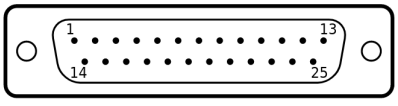


Obrázek 3.10: Blokové schéma komunikace pomocí RS232

Sběrnice UART je charakterstická tím, že komunikace jedním směrem probíhá pouze po jednom vodiči. Tento vodič propojuje vysílač prvního zařízení a přijímač druhého zařízení. Aby mohly přijímače komunikujících zařízení rozlišit napěťové úrovně na tomto vodiči, musejí sdílet svoji zem (`0V`), tudíž přibývá další vodič konkrétně zemnící. Pro obousměrnou komunikaci jsou logicky potřeba 2 komunikační vodiče. S tím souvisí problém, jak korektně propojit vysílače a přijímače obou propojených zařízení, jak je vidět na obrázku 3.10.

Proto se již dávno v začátcích sériové komunikace vytvořili dvě skupiny zařízení a to DTE (Data terminal equipment) a DCE (data comunication equipment). Každá skupina má svoje vysílače a přijímače na svých konektorech zapojeny zrcadlově, tudíž zbývá dodržet pravidlo, že se vždy propojí pouze zařízení rozdílných skupin. Je-li potřeba propojit zařízení ze stejné skupiny používají se na to různé prostředky např. "null-modem" kabel a nebo se musí jiným způsobem prohodit vodiče.

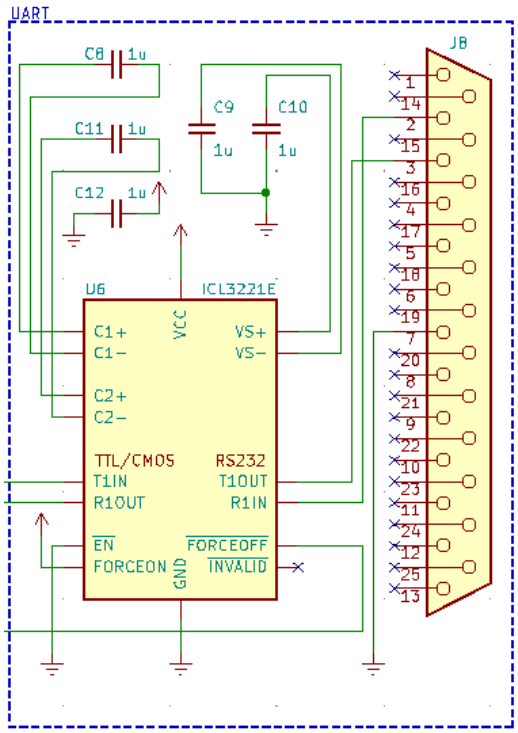
Skupiny DTE a DCE přináší i označení samotných komunikačních vodičů, tedy `TxD (transmitted data)` a `RxD (recieved data)`. Vodič `TxD` přenáší data ve směru z DTE do DCE a vodič `RxD` opačným směrem, jak je také vidět na obrázku 3.10. Je zřejmé že pojmenování vychází z perspektivy skupiny DTE. Terminál má zabudovaný konektor DB-25, to jak vypadá je vidět na obrázku 3.11. Konektor obsahuje 25 vodičů, ale důležité jsou pouze zmíněné 3, konkrétně `TxD`, `RxD` a `GND`. Kde mají své místo v konektoru vysvětluje tabulka 3.5. Tabulka navíc ukazuje jaké zařízení (vysílač nebo přijímač) je připojené na daném vývodu vzhledem ke skupinám zařízení DTE a DCE [15] [19].



Obrázek 3.11: Konektor DB-25, Převzato z [2]

Název vodiče	Číslo pinu	DTE	DCE
TxD	2	vysílač	přijímač
RxD	3	přijímač	vysílač
GND	7	-	-

Tabulka 3.5: Popis vývodů na konektoru DB-25



Obrázek 3.12: Část schématu s RS232 rozhraním

Nyní je potřeba zjist jaké je terminál skupiny, zda DTE či DCE. V dnešní době pojmenování skupin ztrácí význam, avšak terminál se pojmenování drží a patří tedy do skupiny "data terminal equipment", aneb DTE. Z čehož plyne, že na pinu 2 bude možné naslouchat stisknuté znaky a na pin 3 odesílat znaky které vypsat na terminál.

Jak je tedy zapojená samotná část schématu s převodníkem pro RS232 ukazuje obrázek 3.12. Konektor pro DB-25 `J8` má připojeny zmíněné 3 vodiče `TxD`, `RxD` a `GND`. Vývod `GND` je logicky připojen k zemi. `TxD` a `RxD` vedou do převodníku `U6` a poté do MCU. Připojený je i přijímač přestože není pro zadání potřeba. Avšak námaha připojení přijímače se limitně blíží nule a tudíž není důvod ho nepřipojit. Díky tomu je v budoucnu otevřená možnost přijímat i data z terminálu a to pouze změnou softwaru, bez nutnosti zásahu do návrhu hardwaru zařízení.

K pinům převodníku `C1+-`, `C2+-` a `VS+-` jsou připojeny kondezátry `C8-12`, které včetně jejich hodnot určuje datový list převodníku. Hodnoty jsou opět trochu předimezované pro lepší stabilitu. Dle datového listu by postačovaly i kodenzátory s hodnotou `0.1`μ`F`. Tyto konezátory využívá vnitřní regulátor napětí generující přibližně `5.5V` a `-5.5V` pro napěťové úrovně RS232. Funkce ostatních pinů popisuje tabulka 3.6. Jsou zapojeny přímo na log. 0 nebo log. 1 až na pin $\overline{\texttt{FORCEOFF}}$. Tento pin umožňuje uspat převodník a tím snížit odběr proudu, který by zbytečně využívaly vnitřní napěťové regulátory.

Opět se v této části nachází stabilizační kondenzátor, konkrétně `C12`. U tohoto obvodu je velmi důležtý, protože vnitřní regulátor střídavě odebírá proud a poklesy napětí by způsobovaly problémy, jako například podpětí u ostatních integrovaných obvodů.

Vývod	Funkce
`VCC`	Napájecí napětí (`5V`)
`VS+-`	Generované napěťové úrovně pro RS232 (`+5.5V` a `-5.5V`)
`C1+-, C2+-`	Piny pro připojení externích kondenzátorů
`T1IN, T1OUT`	Vysílač, převod z TTL na RS232
`R1IN, R1OUT`	Přijímač, převod z RS232 na TTL
`EN`	Vypnutí převodníku při log. 0
`FORCEON`	Zakázání automatického uspávání převodníku při log. 1
$\overline{\texttt{FORCEOFF}}$	Uspání převodníku při log. 0
$\overline{\texttt{INVALID}}$	Výstupní pin, detekce neznámých logických úrovní na přijímači
`GND`	Zem (`0V`)

Tabulka 3.6: Funkce vývodů převodníku `ICL3221E`

3.3 Deska tištěných spojů

Návrh desky tištěných spojů spočívá v nakreslení reálného zapojení podle kterého se deska poté vyrobí. Hlavním cílem celého procesu je umístit všechny součástky na potřebná místa na desce a propojit je podle schématu s co nejmenším počtem křížení cest. Konektory se ve většině případů nachází na okrajích desky, často i trochu přesahující okraj, aby byly i při zapouzdření do krabičky stále dostupné. Celá tato činnost je spíše designového rázu, ke kterému jsou potřeba zkušenosti a přiměřený cit pro věc. Popisovat postup celého návrhu desky tištěných spojů není v ambicích této práce. Avšak ve zkratce lze říci, že postup prochází postupně těmito činnostmi:

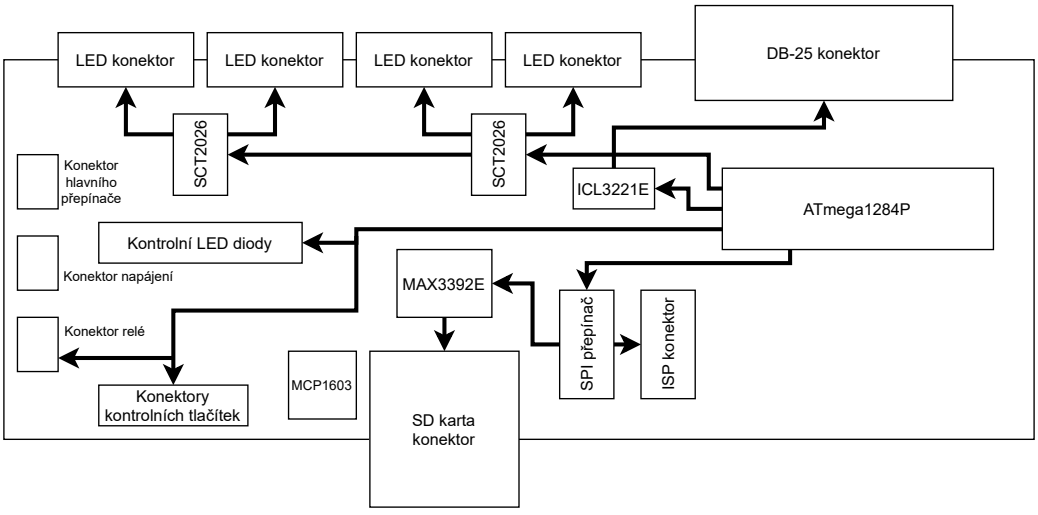
1. Vytvoření plošek, ke kterým se součástka přiletuje, pro každý použitý typ součástky. Potřebné rozměry a vzdálenosti lze vždy nalézt v datovém listu. Avšak součástky často používají obvyklá pouzdra, které návrhový program obsahuje předpřipravené.

2. Rozmístění součástek dle požadavků na umístění konektorů nebo i ostatních součástek. Obvykle je cílem vytvořit desku o co nejmeších rozměrech. Z logiky věci vyplývá, že součástky které jsou propojeny větším množstvím vodičů, je dobré umístit blíže k sobě.

3. Propojení vývodů součástek dle připraveného schématu. Navrhový program radí jaké vývody jsou potřeba zapojit a především kontroluje, zda je vše zapojeno korektně se schématem.

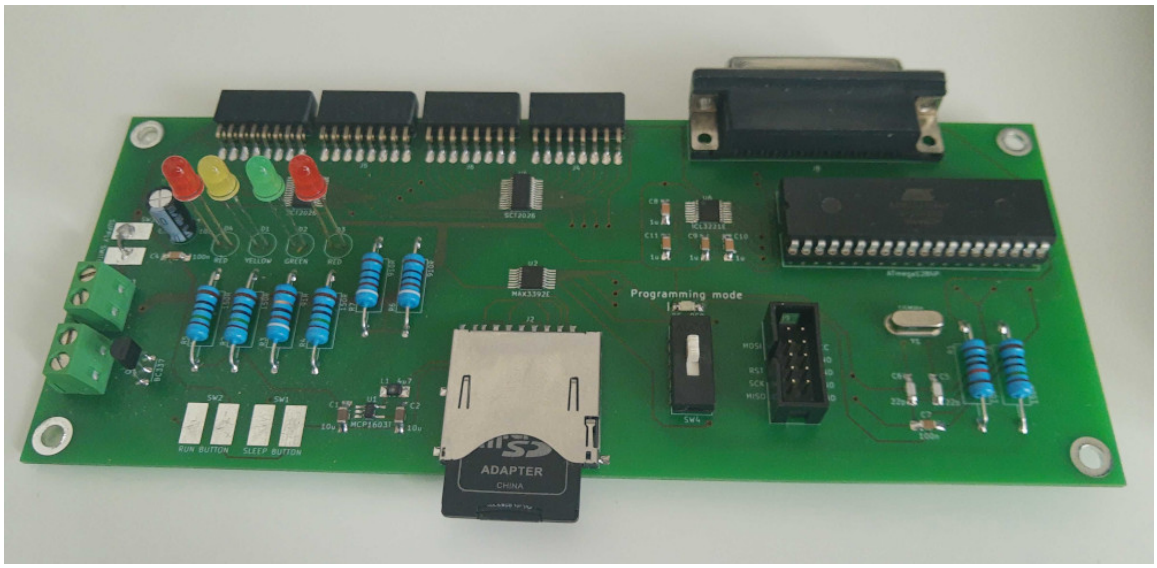
4. Vizuální korekce nevzhledných tažení cest apod.

V tuto chvíli je ovšem důležitý výsledný návrh desky simulátoru, který je vyobrazen v přílohách C.2 a C.3. Zjednodušené blokové schéma celé desky tištěných spojů ukazuje obrázek 3.13 Toto blokové schéma mimo jiné ukazuje přibližné datové cesty komunikace MCU s integrovanými obvody a dalšími součástkami. Nicméně opomíná ostatní nezajímavé pasivní součástky.



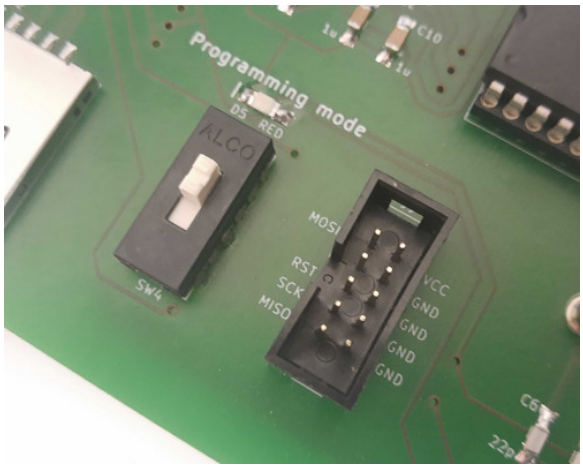
Obrázek 3.13: Blokové schéma desky tištěných spojů

Když je návrh připraven stačí učinit poslední krok, nechat vyrobit desku. Firmy zabívající se výrobou desek tištěných spojů, nabízí také osazení součástkami. Avšak tato deska neobsahuje žádké komplikované zapojení, proto jí lze osadit ve školních podmínkách. Jak výsledná deska vypadá ukazuje obrázek 3.14.



Obrázek 3.14: Deska tištěných spojů simulátoru

Jediná část desky které je vhodné věnovat bližší kousek pozornosti, je část s přepínačem sběrnice SPI. Nad přepínačem se nachází LED dioda indikující přepnutí sběrnice SPI k ISP tedy do programovacího módu. Pro odladěný program ve finálním zařízení lze tuto část z návrhu desky zcela vypustit.



Obrázek 3.15: Přepínač SPI sběrnice na desce tištěných spojů

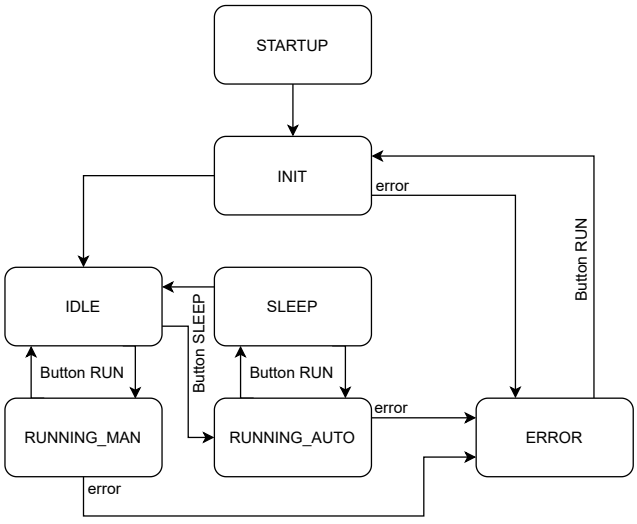
Kapitola 4

Návrh softwarové části simulátoru

Celý program mikrokontroléru je napsaný v jazyce C s využitím knihovny `avr-libc`. Kompilace probíhá pomocí odnože kompilátoru `gcc`, konkrétně `avr-gcc`. Tento kompilátor je přímo vytvořený pro mikrokontroléry architektury AVR. Pro naprogramování mikrokontroléru lze použít libovolný programátor připojením k ISP konektoru. Způsob obsluhy je různorodý pro rozdílné typy programátorů, proto zde nebude nijak popisován. Tato kapitola nevysvětluje problematiku kompilace kódu pro architekturu AVR, ani tak nepopisuje kód programu řádek po řádku. Pouze poukáže na zásadní principy a zajímavé pasáže softwarového návrhu. Všechny potřebné zdrojové soubory se nachází na přiloženém médiu, včetně návodu na kompilaci.

4.1 Struktura řídícího programu

Základní struktura programu se drží myšlenky konečného automatu, jelikož k tomu koncept spouštění simulace, uspávání simulátoru apod. nabádá. Z implemetačního hlediska program standardně začíná funkcí `main`, provede základní nastavením MCU a končí v nekonečné smyčce, ta obsahuje převážně kód zmíněného konečného automatu. Jaké má konečný automat programu stavy a přechody ukazuje obrázek 4.1.



Obrázek 4.1: Reprezentace struktury řídícího programu konečným automatem

Významy jednotlivých stavů jsou následující:

- `STARTUP` - Stav ihned po zapnutí simulátoru, inicializace MCU, otestování LED diod a zkouška odeslání znaků na terminál.
- `INIT` - Inicializace SD karty a pokus o načtení souboru ze souborového systému FAT32.
- `IDLE` - Simulátor je připraven pro spuštění simulace.
- `SLEEP` - Spánkový režim, simulace se spustí sama za určitou dobu.
- `RUNNING_MAN` - Bežící ručně spuštěná simulace.
- `RUNNING_AUTO` - Běžící automaticky spouštěná simulace.
- `ERROR` - Během inicializace nebo simulace nastala chyba.

Po připojení napájení se simulátor nachází ve stavu `STARTUP`, ve kterém provede inicializaci MCU. To znamená nastavení portů a všech ostatním periferií jako např. časovačů, přerušení, sběrnicí SPI a UART atd. Součást inicializace je také vyzkoušení všech LED diod, jak kontrolních LED diod, tak i těch ovládaných budiči. Zkouška spočívá v rozsvícení každé LED diody na krátký časový úsek. V posledním kroku se simulátor pokusí vypsat úvodní hlášku na terminál.

Po dokončení akcí stavu `STARTUP` se simulátor automaticky přesune do stavu `INIT`, v kterém se nachází část programu pro inicializaci SD karty a rozbor souborového systému FAT32. Pokud se podaří úspěšně navázat komunikaci s SD kartou a najít požadovaný soubor, inicializace končí úspěchem a simulátor se přesune do stavu `IDLE`. V opačném případě simulátor vypíše na terminál chybovou hlášku a skončí ve stavu `ERROR`. Jak probíhá inicializace SD karty a rozbor souborového systému popisují podkapitoly 4.4 a 4.5.

Ve stavu `IDLE` simulátor vyčkává na stisknutí kontrolních tlačítek. Podrobné funkce kontrolních tlačítek a jak kontrolní LED diody indikují uvedené stavy popisuje příloha B. Z tohoto stavu lze ručně spustit simulaci, nebo spustit simulaci s aktivovaným automatickým spouštěním. Ručně spuštěnou simulaci reprezentuje stav `RUNNING_MAN`, ze kterého se simulátor po skončení simulace vrací do stavu `IDLE`. Simulaci spuštěnou s aktivovaným automatickým spouštěním značí stav `RUNNING_AUTO`, ze kterého simulátor přechází po skončení simulace do stavu spánkového režimu `SLEEP`. Spánkový režim uspí MCU do `power-save` módu, který nabízí `ATmega1284P` pro snížení spotřeby, když není MCU aktivně využíván.

Z tohoto módu může MCU mimo jiné probudit přerušení vyvolané časovačem, čehož právě simulátor využívá. S každým přerušením se zkontroluje, zda již uplynul stanovený čas, pokud ano, spustí se simulace jinak se MCU opět uspí. Před vstupem do spánkového režimu MCU vypne relé, které kontroluje napájení terminálu, zhasne veškeré LED diody a také uspí převodník napěťových úrovní RS232 `ICL3221E`. Po probuzení simulátor spustí terminál pomocí relé a probudí převodník. Čeká určitý čas, než se CRT obrazovka terminálu zahřeje a jakmile je vše připraveno znovu spustí simulaci.

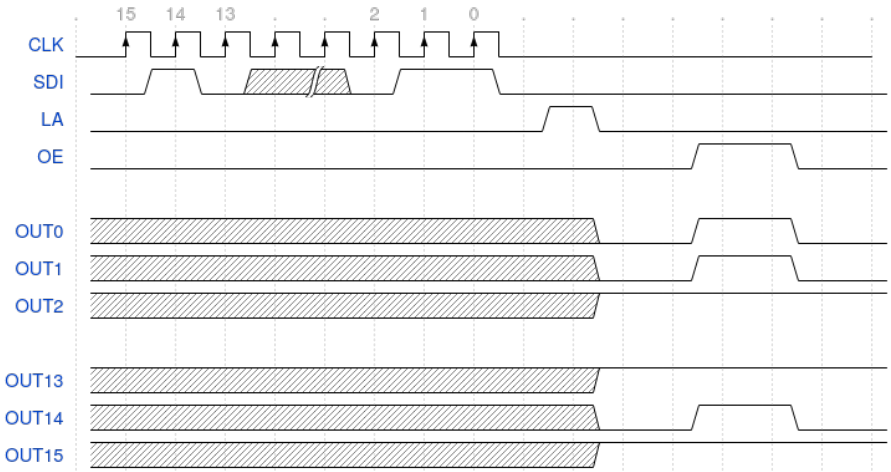
Během běžící simulace může také nastat chyba, což způsobí stejnou akci jako u stavu `INIT`, tedy přechod do stavu `ERROR`. Ze stavu chyby lze pomocí stisknutí tlačítka opakovat inicializaci a tím pádem i celý proces simulace.

4.2 Obsluha kontrolních tlačítek

K obsluze kontrolních tlačítek lze přistoupit dvěma způsoby. Kontrolovat jejich stisk v nekonečné smyčce, nebo reagovat na stisk přerušením. Jelikož je v nekonečné smyčce simulátoru implementován konečný automat, nepřipadá tento způsob v úvahu. Je potřeba využít přerušení MCU. `ATmega1284P` nabízí přerušení při změně stavu vstupního pinu, které je voláno vždy když se log. úroveň na vybraných pinech změní. Ovšem je nezbytné uvědomit si následující skutečnosti. Vektor přerušení je společný pro všechna tlačítka. Tudíž nelze v obsluze přerušení pouze kontrolovat, jaké tlačítko bylo stisknuto a podle toho jednoduše vykonat příslušnou akci. Je zásadní zapamatovat si informaci, zda akce tlačítka již byla vykonána a pokud tlačítko mezi více voláními vektoru přerušení nebylo uvolněno, akci tlačítka vícekrát neprovádět. Může totiž nastat situace volání vektoru přerušení vyvolané tlačítkem `B`, zatímco tlačítko `A` je stále stisknuté z předchozího volání a bez kontroly uvolnění by se provedla akce tlačítka `A` znovu. Další záležitostí je vyřešení zákmitů. Stisknutí tlačítka často není dokonalé a jedno stisknutí může MCU interpretovat jako 2 nebo více stisknutí. Jednoduchým řešením je v obsluze přerušení vyčkat např. 1 ms a poté teprve kontrolovat zda bylo tlačítko opravdu stisknuto.

4.3 Komunikace s budiči LED diod

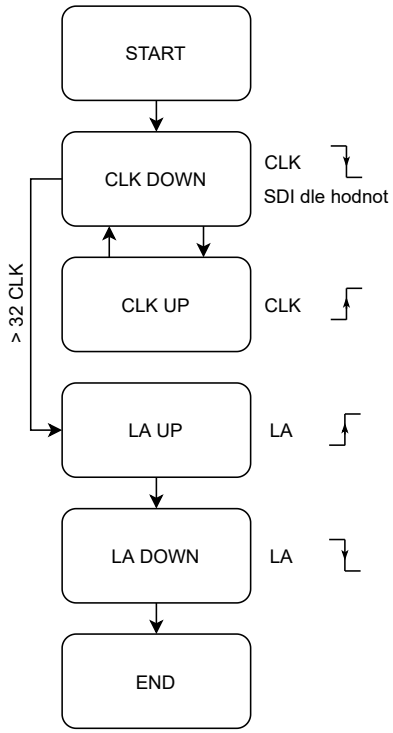
Komunikace s budiči `SCT2026` probíhá na podobném principu jako na sběrnici SPI. Na pin `SDI` se přivádějí data synchonizovaná s hodinový signálem `CLK`. Tímto způsobem se nahrají data do posuvného registru budiče s tím, že log. 1 znamená rozsvítit danou LED didodu a log. 0 zhasnout. Pokud je přenos dat do posuvného registru dokončený, je potřeba přesunout data do střádacího registru, ovládající přímo proudové zdroje. K tomu slouží pin $\overline{\texttt{LA}}$. Přesun se provede odesláním jednoho pulzu log. 1. Budič také nabízí odpojení výstupů přivedením log. 1 na pin $\overline{\texttt{OE}}$. Jelikož rozhraní SPI mikrokotroléru již používá SD karta, není možné jej pro budiče využít. Takže je potřeba celou komunikaci obsloužit ručně na obecných vstupně výstupních pinech. Ukázku průběhu komunikace znázorňuje diagram na obrázku 4.2.



Obrázek 4.2: Ukázka průběhu komunikace s budičem LED diod `SCT2026`

K vytvoření takového průběhu poslouží další konečný automat, viz obrázek 4.3. Pin $\overline{\texttt{OE}}$ není využíván, protože není potřeba odpojovat LED diody, je tedy vždy připojen na log. 0. Přechod konečného automatu se provede vždy při přetečení časovače které nastaná s frekvencí přibližně 31kHz. Konečný automat je tedy implementován v obsluze přerušení vyvolaného přetečením časovače.

Obsluha budiče odešle postupně 32 vzestupných hran na výstupní pin, ke kterému je připojen pin `CLK` budiče. Při sestupných hranách změní stav pinu `SDI` podle nastaveného stavu rozsícení LED diod. Po odeslání stavů všech 32 led diod, přichází pulz na pin $\overline{\texttt{LA}}$ pomocí stavů `LA UP` a `LA DOWN`.



Obrázek 4.3: Stavy konečného automatu pro generování průběhu komunikace s `SCT2026`

4.4 Komunikace s SD kartou

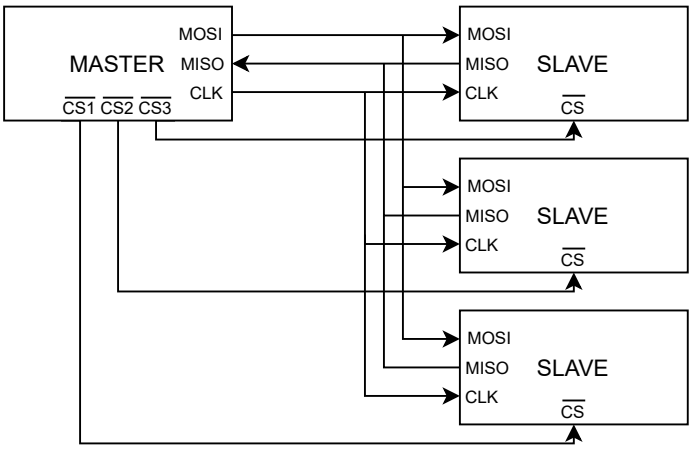
SD karta je nejkomplexnějším zařízením s jakým MCU komunikuje. Výrobců SD karet není málo a navíc exitují různé typy a rychlostní třídy. Naštěstí dnes již všechny paměťové média nesoucí označení SD karta podlehájí standardu, který spravuje organizace SD Assiociation. Na internetu je dostupná zjednodušená specifikace tohoto standardu, v které se vyskytují všechny potřebné informace k implementování komunikace s SD kartou [12] [3]. Veškeré následující informace o komunikaci s SD kartou vycházejí právě z dané specifikace. Známými typy SD karet jsou `SC`, `HC` a `XC`, jejichž hlavním rozdílem je maximální kapacita. Je vhodné aby simulátor podporoval všechny tyto typy.

Dle specifikace SD karta podporuje dva řežimy komunikace. Pomocí sběrnice SPI nebo speciálního SD protokolu. SD protokol oproti sběrnici SPI nabízí širší paletu funkčnosti a větší maximální rychlost přenosu dat. Bohužel komplexnost SD protokolu přesahuje celou tuto práci. Tudíž simulátor používá řežim komunikace po sběrnici SPI, který pro potřeby simulátoru spolehlivě postačuje.

Samotná specifikace se postupem času vyvíjela a má různé verze. Proto některé starší karty nemusí podporovat příkazy, které určuje dnešní verze specifikace. Naštěstí existují postupy jak zajistit zpětnou kompabilitu a podporovat tak většinu karet. Pro obyčejné načítání souboru stačí, aby simulátor podporoval klasické `SC`, `HC` a `XC` karty.

Sběrnice SPI

Sběrnice SPI je obecně známá, ale pro další popis řešení zde budou uvedeny nejpodstatnější principy. Sběrnice podporuje více připojených zařízení. Vždy platí že jedno zařízení musí být `MASTER` a ostatní `SLAVE`, viz obrázek 4.4. Zařízení `MASTER` si vybírá s jakým zařízením chce komunikovat pomocí vodiče $\overline{\texttt{CS}}$ na který přivede pro vybrané zařízení log. 0. Hodinový signál je generován zařízením `MASTER` [16].

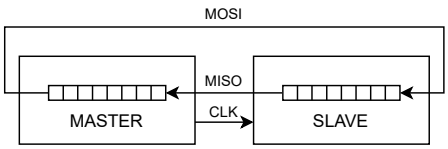


Obrázek 4.4: Zapojení zařízení k sběrnici SPI

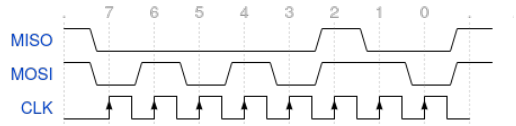
Sběrnice využívá následující 3 vodiče:

- `CLK` - Synchonizační hodinový signál.
- `MISO` - Data odesílaná od `SLAVE` k `MASTER` (`MASTER IN SLAVE OUT`)
- `MOSI` - Data odesílaná od `MASTER` k `SLAVE` (`MASTER OUT SLAVE IN`)

Samotná komunikace funguje na principu výměny dat vnitřích registrů zařízení. Takže si lze zapojení představit jako dva posuvné registry zapojené do kruhu, viz obrázek 4.5. S hodinovým signálem data odesílají a přijímají vždy obě zařízení. Pokud chce zařízení `SLAVE` odeslat data, musí počkat, než zařízení `MASTER` poskytne patřičný hodinový signál. Příklad odeslání hodnoty `0x56` ze zařízení `MASTER` a hodnoty `0x04` ze zařízení `SLAVE` znázorňuje diagram 4.6.



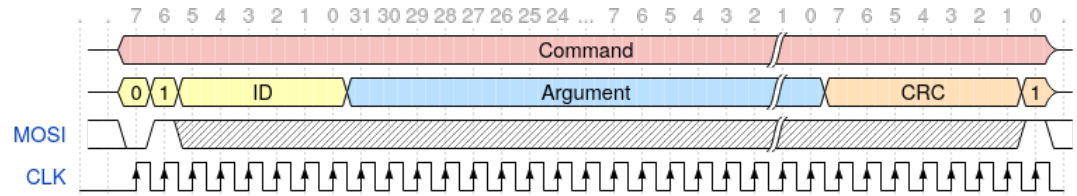
Obrázek 4.5: Zapojení vodičů sběrnice SPI k vnitřním registrům zařízení



Obrázek 4.6: Příklad průběhu komunikace po sběrnici SPI

Formát příkazu SD karty

Komunikace SD karty s MCU probíhá pomocí zasílání příkazů a přijímaní patřičných odpovědí. MCU je na sběrnici v roli `MASTER` a SD karta v roli `SLAVE`. Nejmenší přenášenou jednotkou je jeden byte, tudíž si MCU s SD kartou vždy vymění celý obsah vnitřího SPI registru. Příkazy se označují prefixem `CMD` následující číslem (identifikátorem) příkazu. Formát příkazu odesílaného SD kartě ukazuje diagram na obrázku 4.7.



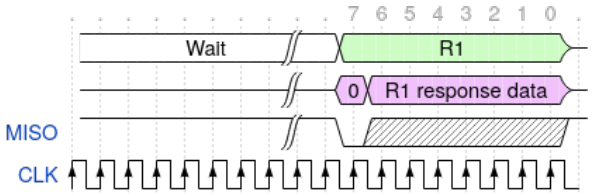
Obrázek 4.7: Formát příkazu SD karty

Příkaz je vždy dlouhý 6 bytů. Nižších 6 bitů prvního bytu nesou číslo (identifikátor) příkazu. Předposlední bit musí být vždy log. 1 a MSB log. 0. Například pro zaslání příkazu s identifikátorem 17, by tento byte obsahoval hodnotu `01010001`. Hned záhy následují 4 byty obsahující argument příkazu. Argument má pro každý příkaz individuální reprezentaci. Horních 7 bitů posledního bytu obsahují hodnotu kontrolního CRC součtu předchozích 5 bytů příkazu. V SPI módu SD karta provádí kontrolu CRC pouze u příkazů `CMD0` a `CMD8`. U ostatních příkazů může být hodnota CRC libovolná. Nultý bit posledního bytu musí být vždy log. 1. Specifikace určuje výpočet kontrolního CRC součtu následovně:

$$
\begin{aligned}
&Generator\ polynom : G(x) = x^7 + x^3 + 1 \\
&M(x) = (first\ bit) * x^n + (second\ bit) * x^{n-1} + ... + (last\ bit) * x^0 \\
&CRC = Remainder[(M(x) * x^7)/G(x)]
\end{aligned}
\tag{4.1}
$$

Odpověď SD karty zpravidla nepřichází ihned v dalším přeneseném bytu po odeslání příkazu. MCU musí čekat neurčitý počet bytů odesíláním hodnoty `0xff`, čímž poskytuje hodinový signál. Dokud nemá SD karta připravenou odpověď odesílá také `0xff`. Tudíž si SD karta a MCU vyměňují `0xff`, než SD karta odešle validní odpověď.

Jakmile je příkaz odeslán, MCU musí čekat na odpověď 0 až 8 bytů zmíněným odesíláním hodnoty `0xff`. Pokud SD karta ani po výměně 8 bytů neodpoví, pravděpodobně příkaz nezná nebo nastala jiná chyba. Zda je přijatý byte validní odpověď, lze rozeznat podle MSB přijatého bytu, který bude log. 0. Na různé příkazy SD karta odpovídá různými typy odpovědí, nicméně pro potřeby jednoduchého čtení dat jsou zásadní typy `R1`, `R3` a `R7`. Formát odpovědi `R1` ukazuje diagram na obrázku 4.8. Tato odpověď je dlouhá pouze jeden byte. Reprezentaci jednotlivých bitů v odpovědi `R1` popisuje tabulka 4.1.

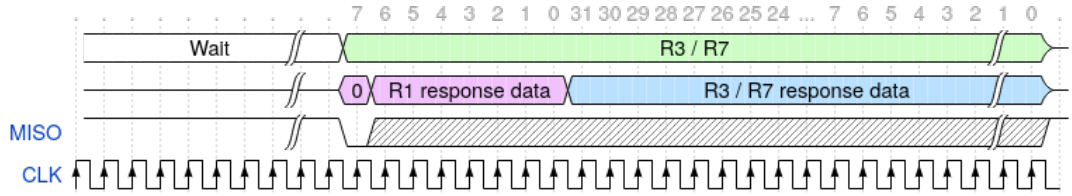


Obrázek 4.8: Formát odpovědi SD karty typu R1

Bit	Význam (při log. 1)
0	SD karta je v `idle` stavu a beží inicializační proces.
1	Mazací sekvence byla spuštěna.
2	Neznámý příkaz.
3	Chyba kontrolního CRC součtu.
4	Chyba při mazací sekvenci.
5	Neplatná adresa pro čtení nebo zápis dat.
6	Neplatný argument.
7	Vždy log. 0

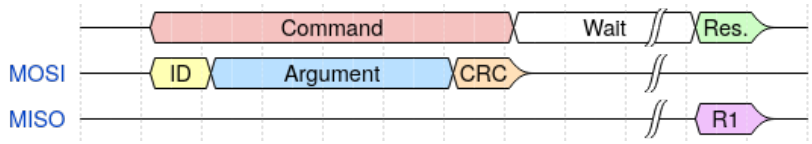
Tabulka 4.1: Význam bitů v odpovědi SD karty typu R1

Odpovědi `R3` a `R7` jsou dlouhé 5 bytů. První byte má stejnou reprezentaci jak u samostatné odpovědi `R1`. Význam dalších 4 bytů je u obou odpovědí vysvětlen později, protože je přímo spjatý s příkazy, na které SD karta těmito odpovědmi reaguje. Formát odpovědi `R3` a `R7` ukazuje diagram na obrázku 4.9.

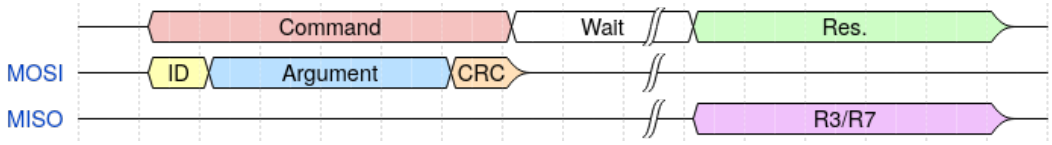


Obrázek 4.9: Formát odpovědi SD karty typu R3 nebo R7

Průběhy odeslání příkazu včetně přijmutí odpovědi z pohledu MCU i SD karty v zjednodušeném podání vysvětlují diagramy na obrázcích 4.10 a 4.11. Veškerá komunikace s SD kartou a MCU se drží tohoto schématu. Diagramy mají na rozdíl od předchozích jako základní jednodnotku jeden byte.



Obrázek 4.10: Zjednodušený průběh odeslání příkazu SD kartě s odpovědí typu `R1`.



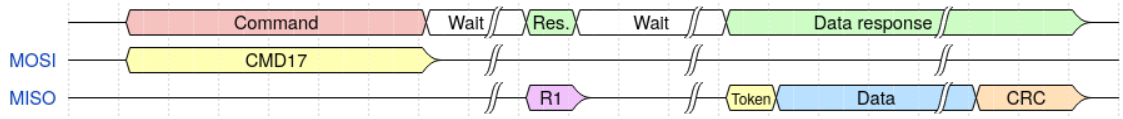
Obrázek 4.11: Zjednodušený průběh odeslání příkazu SD kartě s odpovědí typu `R3` nebo `R7`.

Čtení dat z SD karty funguje na principu bloků. To znamená, že nejmenší jednotkou kterou lze samostatně přečíst z úložného prostoru SD karty, je daný blok. U SD karet typu `SC` je možné specifikovat požadovanou velikost bloku pomocí příkazu `CMD16`. SD karty `HC` a `XC` mají již fixní velikost bloku 512 bytů. Pro adresování těchto bloků existují dva způsoby, adresování po blocích nebo po bytech. Rozdíl mezi těmito způsoby vysvětluje tabulka 4.2. Při adresování po bytech může být adresa libovolná v rozsahu daného bloku. `SC` SD karty používají adresování po bytech a `HC` a `XC` adresují po blocích. Proto je důležité při inicializaci zjistit jaký typ karty je připojen a podle toho korektně adresovat.

Blok paměti v přečtení (v bytech)	Bloková adresa	Bytová adresa
512 - 1023	1	512 - 1023
2048 - 2559	3	2048 - 2559

Tabulka 4.2: Rozdíl mezi adresováním datových bloků SD karty po blocích nebo po bytech

Pro přečtení bloku dat uložených na SD kartě je nejpre nutné zaslat příkaz `CMD17` s požadovanou adresou v argumentu a poté počkat na odpověď `R1`. Neznačí-li odpověď `R1` chybu přichází další čekání, konkrétně na `data token`. Validní hodnotou `data tokenu` je `0xfe`. Ihned za `data tokenem` následuje požadovaných 512 bytů načteného bloku. A nakonec 2 byty kontrolního CRC součtu, i když program neprování kontrolu CRC je nutné tyto 2 byty přečíst. Celý průběh zobrazuje diagram na obrázku 4.12.



Obrázek 4.12: Průběh komunikace s SD kartou při čtení dat

Příkazy SD karty

Pro korektní inicializaci karty a jednoduché načítání bloků dat stačí znalost pouze následujících 10 příkazů.

Příkaz	Funkce	Argument	Odpověď
`CMD0`	Reset SD karty	nedefinovaný	`R1`
`CMD8`	Kontrolní příkaz	viz dále	`R7`
`CMD55`	Předzvěst aplikačního příkazu	nedefinovaný	`R1`
`ACMD41`	Inicializační příkaz novějších karet	viz dále	`R1`
`CMD1`	Inicializační příkaz pro staré karty	viz dále	`R1`
`CMD16`	Nastavení velikosti bloku	velikost bloku	`R1`
`CMD17`	Načtení bloku dat	adresa	`R1` + data
`CMD58`	Načtení OCR registru	nedefinovavý	`R3`
`CMD9`	Načtení CSD registru	nedefinovaný	`R1` + data
`CMD10`	Načtení CID registru	nedefinovaný	`R1` + data

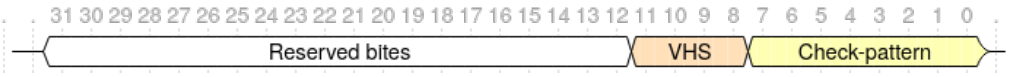
Tabulka 4.3: Klíčové příkazy SD karty

CMD0

Tento příkaz zresetuje kartu a uvede jí do stavu `idle`. U tohoto příkazu je vyžadována validní hodnota CRC. Jelikož příkaz nemá žádné argumenty a hodnota argumentu je tedy `0x00000000`, pak je CRC hodnota pro každé volání příkazu stejná, konkrétně 74. Celý poslední byte příkazu má tedy hodnotu `0x95`.

CMD8

Hlavní funkcí příkazu je kontrola zda SD karta může pracovat na definovaném pracovním napětí a zda komunikace probíhá korektně. Příkaz je definový pouze od specicikace verze 2.0 a vyšší. Pokud tedy SD karta korektně odpoví na tento příkaz, lze předpokládat, že SD karta podléhá specifikaci verze 2.0 a vyšší. Jakou mají reprezentaci 4 byty argumentu příkazu ukazuje obrázek 4.13 s tabulkou 4.4.



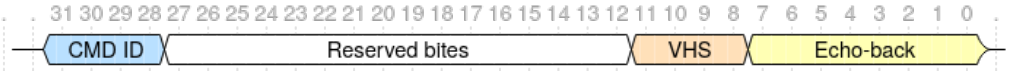
Obrázek 4.13: Formát argumentu příkazu `CMD8`

Hodnota VHS	Význam
`0b0001`	`2.7V - 3.6V`
`0b0010`	Rezervováno pro nízké napětí
`0b0100`	Rezervováno
`0b1000`	Rezervováno
Ostatní	Nedefinováno

Tabulka 4.4: Význam bitů pole `VHS` v argumentu příkazu `CMD8`

Pole VHS v argumentu reprezentuje podporované napětí zařízením, ke kterému je karta připojena. Druhé pole `Check-pattern` může obsahovat libovolnou hodnotu např. `0xcd` a slouží ke kontrole jestli komunikace probíhá korektně. Opět jako u `CMD0` musí být kontrolní CRC součet validní. Argument je konstatní proto lze použít předpočítanou hodnotu kontroního CRC součtu. Pro argument hodnoty `0x000001cd` vychází CRC součet 42, poslední byte příkazu má tedy hodnotu `0x55`.

SD karta reaguje na tento příkaz odpovědí `R7`, formát datových 4 bytů odpovědi ukazuje obrázek 4.14. V poli `CMD ID` je vždy hodnota 8. V poli `VHS` SD karta vrací podporované napětí ve stejném formátu jak u argumentu příkazu. A v posledním poli `Echo-back` karta vrátí zadanou hodnotu `Check-pattern` v argumentu. Pole `Check-pattern` a `Echo-back` se musí shodovat, jinak komunikace s SD kartou neprobíhá korektně.



Obrázek 4.14: Formát odpovědi typu `R7`

CMD55

Specifikace rozšiřuje množinu standardních příkazů o takzvané aplikační příkazy, jejichž označení nese prefix ACMD. Aplikační příkaz sestává ze sekvence dvou příkazů. První je vždy CMD55, který dáva SD kartě najevo, že má další přijatý příkaz vyhodnottit jako aplikační. Za CMD55 následuje příkaz s identifikátorem chtěného aplikačního příkazu. Takže pokud je zaslán SD kartě např. příkaz CMD13 karta ho logicky vyhodnotí jako příkaz CMD13. Ovšem sekvenci příkazů CMD55, CMD13 karta vyhodnotí jako jeden příkaz ACMD13.

ACMD41

Aplikační příkaz ACMD41 slouží pro spuštění inicializace SD karty. Předposlední bit argumentu nese označení HCS. Log. 1 v tomto bitu dává kartě najevo, že zařízení ke kterému je karta připojena podporuje SD karty typu HC i XC. Pokud je tedy SD karta typu HC nebo XC a bit HCS v argumentu příkazu ACMD41 je log. 0, inicializace neproběhne úspěšně. Nicméně pokud SD karta neodpoví na CMD8, což znamená, že karta podléhá specifikaci veze nižší než 2.0, pak karta bit HCS ignoruje. V době verze specifikace verze nižší než 2.0, totiž ještě karty typu HC a XC neexistovali. Jelikož je ACMD41 aplikační příkaz, jde ve skutečnosti o 2 příkazy CMD55 a CMD41.

CMD1

Tento příkaz má stejnou funkci jako ACMD41, tedy spuštění inicializace SD karty. Ovšem tento příkaz je možné použít pouze pro staré MMC karty. Pro inicializaci karty je silně doporučeno použít příkaz ACMD41 a pouze v případě, kdy ho karta nerozpozná, pak použít jako poslední možnost CMD1. Reprezentace argumentu příkazu je identická s příkazem ACMD41.

CMD16

Pokud je při inicializaci SD karty zjištěno, že se jedná o kartu typu SC, je nutné pomocí tohoto příkazu, nastavit délku čteného bloku dat na 512 bytů. Jenom tak je zajištěna identičnost funkce příkazu načítání bloků dat pro všechny podporované typy karet.

CMD17

Pomocí tohoto příkazu lze požádat SD kartu o zaslání bloku uložených dat. Jak již bylo zmíněno je potřeba brát ohled na rozdílné adresování a pomocí příkazu CMD16 zajistit, aby tento příkaz načítal vždy blok o stejné délce, tedy 512 bytů. Průběh čtení dat byl již ukázán v předchozí podkapitole.

CMD58

Příkaz CMD58 se používá pro načtení OCR registru SD karty. Registry SD karty jsou vnitřní paměťové jednotky, které obsahují často užitečné informace o kartě. Příkaz má svůj druh odpovědi R3, jejíž formát není pro potřeby simulátoru postatný krom 30. bitu. Ten nese označení CCS. Je-li tento bit log. 1, znamená to, že je karta typu HC nebo XC v opačném případě je karta typu SC. Tento příkaz se používá především právě pro zjištění typu karty.

CMD9

Pomocí příkazu `CMD9` lze požádat SD kartu o zaslání vnitřního registru `CSD`. Registr `CSD` je dlouhý 16 bytů a lze z něho zjistit celkovou kapacitu SD karty. Proto jeho data následují po odpovědi `R1` s počátečním `data tokenem` dle identického schématu jako u příkazu `CMD17`. S jediným rozdílem, že karta neodesílá 512 bytů, ale pouze 16. Registr nemá pro všechny karty jednotnou strukturu a jeho struktura podléhá verzím. Registr pro SD karty typu `SC` používá strukturu verze 1.0 a karty typu `HC` nebo `XC` používají strukturu verze 2.0. Tabulky 4.5 a 4.6 ukazují, kde se v registru nachází potřebné položky pro výpočet kapacity SD karty.

Název	Pozice	Hodnota	Délka
`CSD_STRUCTURE`	127-126	`0b00`	2
`READ_BL_LEN`	83-80	`?`	4
`C_SIZE`	73-62	`?`	12
`C_SIZE_MULT`	49-47	`?`	3

Tabulka 4.5: Důležité hodnoty z registru SD karty s označením `CSD` verze 1.0

Název	Pozice	Hodnota	Délka
`CSD_STRUCTURE`	127-126	`0b01`	2
`C_SIZE`	69-48	`?`	22

Tabulka 4.6: Důležité hodnoty z registru SD karty s označením `CSD` verze 2.0

Z verze registru 1.0 lze z těchto hodnot vypočítat kapacita celé SD karty pomocí rovnice:

$$Capacity = (C\_SIZE + 1) * 2^{C\_SIZE\_MULT+2} * 2^{READ\_BL\_LEN} \tag{4.2}$$

Pro verzi 2.0 pak pomocí rovnice:

$$Capacity = (C\_SIZE + 1) * 512 \tag{4.3}$$

CMD10

Příkaz `CMD10` funguje na stejném principu jako `CMD9` s jediným rozdílem, že načítá vnitřní registr `CID`. Tento registr je stejně jako `CSD` velký 16 bytů. Lze v něm naleznout informace o výrobci, sériovém čísle, produktovém jméně atd. Tabulka 4.7 ukazuje jaké informace simulátor o SD kartě načítá a kde se v `CID` registru nachází. Narozdíl od `CSD` registru má tento jednotou strukturu pro všechny druhy karet.

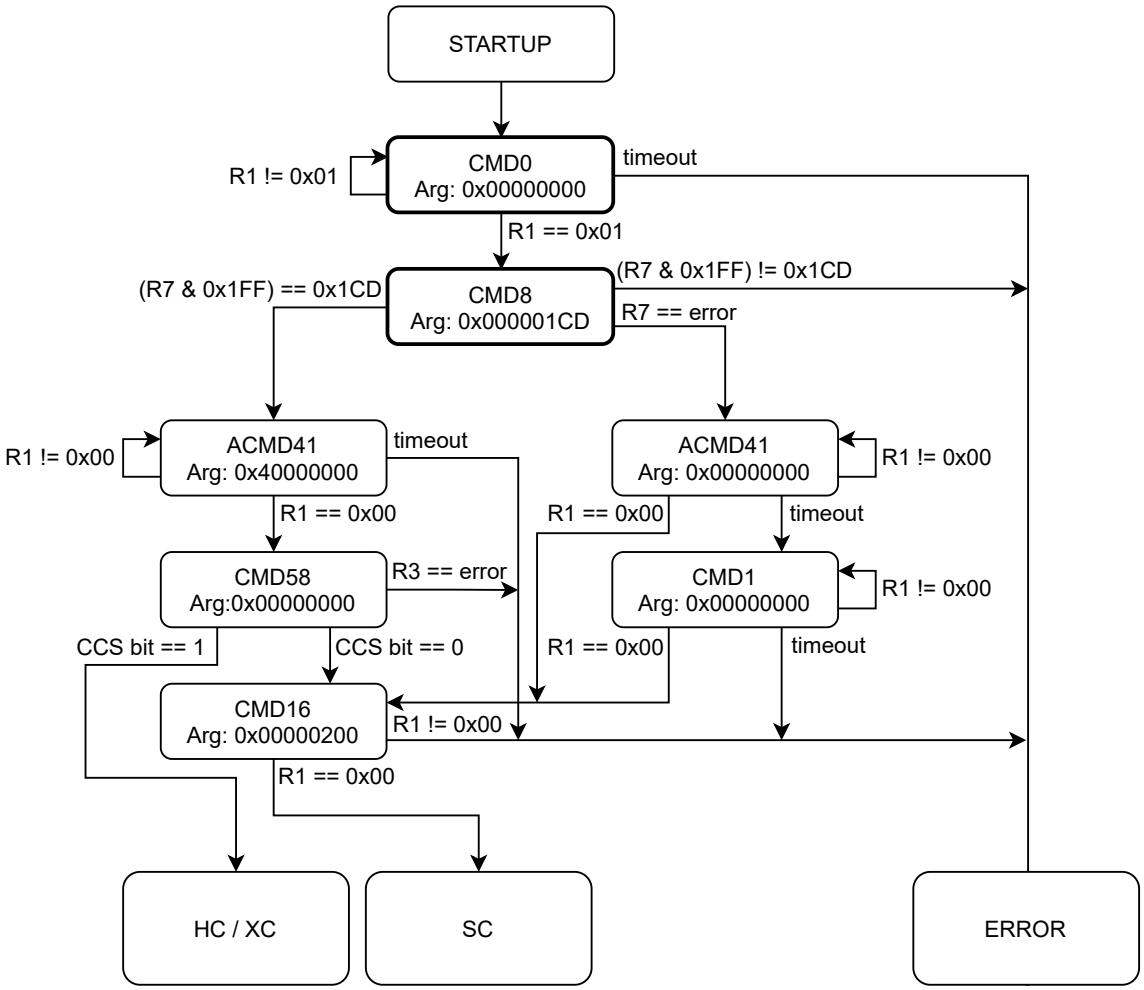
Název	Pozice	Hodnota	Délka
Výrobce	127-120	`?`	8
Produktové číslo	103-64	5 znaků v ASCII	40

Tabulka 4.7: Důležité hodnoty z registru SD karty s označením `CID`

Každý výrobce vlastní unikátní 8-bit číslo. Nejznámější výrobci mají přižené tyto hodnoty: `0x01` - Panasonic, `0x03` - SanDisk, `0x1b` - Samsung, `0x41` - Kingston atd.

Inicizace SD karty

Konkrétné postup inicializace SD karty znázorňuje diagram na obrázku 4.15.



Obrázek 4.15: Diagram sekvence příkazů pro inicializaci SD karty

Stav `STARTUP` zahrnuje postupně tyto akce:

1. Nastavení frekvence SPI sběrnice v rozmezí 100 kHz až 400 kHz
2. Přivedení log. 1 na pin $\overline{\texttt{CS}}$
3. Vyčkání alespoň 10 ms
4. Zaslání více než 74 hodinových pulzů
5. Přivedení log. 0 na pin $\overline{\texttt{CS}}$

Vykonání akcí ze stavu `STARTUP` je klíčové pro úspěšnou inicializaci. Teprve poté může přijít první příkaz inicializace, konkrétně `CMD0` s korektním CRC součtem. Poté následuje příkaz `CMD8`, který je dle specifikace vyžadován před zavolám příkazu `ACMD41`. Tento příkaz také vyžaduje korektní CRC součet.

Pokud karta na příkaz `CMD8` neodpoví, znamená to, že připojená karta podléhá specifikaci verze nižší než 2.0. To znamená, že bit `HCS` v příkazu `ACMD41` bude ignorován a karta může být pouze typu `SC`. Pokud karta nevrátí kokrektně pole `Echo-back` a `VHS`, pak karta není v pořádku a nelze pokračovat v její inicializaci. `CMD8` tedy rozděluje inicializaci do dvou větví podle verze specifikace karty.

V obou větvích následují příkazy `AMCD41` pouze s rozdílem, že příkaz ve větvi pro starší karty nemá nastaven bit `HCS` v argumentu. Tento příkaz spustí inicializaci, která může trvat i stovky ms. Proto je nutné příkaz opakovat, dokud karta nevrátí odpověď `R1` s hodnotou `0x00`, která znamená že je karta úspěšně inicializována. Pokud se u starších karet nepodaří úspěšně provést příkaz `ACMD41`, je možné zkusit příkaz `CMD1`. Podaří-li se získat odpověď `R1` s hodnotu `0x00` od příkazů `ACMD41` nebo `CMD1` ve větvi pro starší karty, pak vždy následuje příkaz `CMD16` pro nastavení velikosti čteného bloku a inicializace končí jako úspěšná.

Ve větvi pro novější karty je potřeba zjistit o jaký typ karty se jedná pomocí příkazu `CMD58`. Pokud je `CSS` bit v odpovědi na tento příkaz log. 1, znamená to, že je karta typu `HC` nebo `XC` a inicializace může skončit. Je-li bit log. 0 pak je karta typu `SC`, musí být také zavolán příkaz `CMD16` a pak teprve inicializace úspěšně končí.

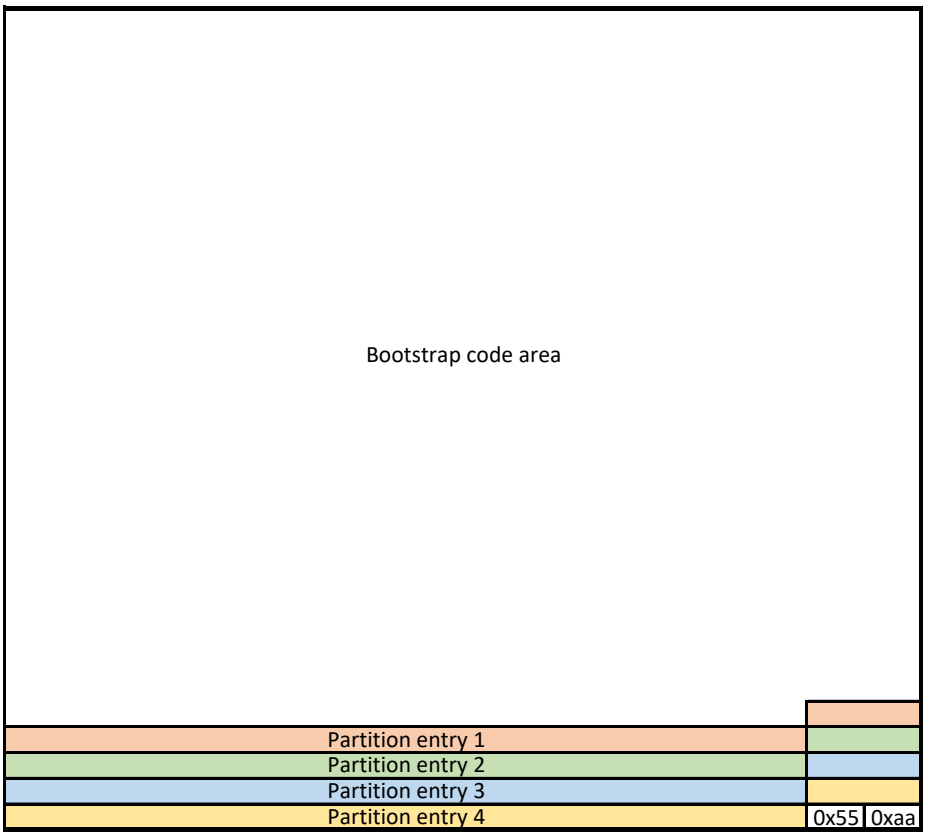
Tato sekvence příkazů je povinná pro úplnou inicializaci SD karty. Z neinicializované karty nelze číst žádná data a tudíž jí nelze nijak používat. Simulátor v rámci inicializace také použije příkazy `CMD9` a `CMD10`, pro zjištění kapacity karty a jejího výrobce. Tyto infomace vypíše na terminál v rámci inicializace celého simulátoru.

4.5 Načítání souboru ze souborového systému FAT32

Souborový systém FAT32 je v celku jednoduchý systém, který podporuje většina operačních systémů. Pro podobné aplikace s MCU a vyměnitelným médiem je tento systém častou volbou. Dokumentace souborového systému FAT32 je veřejně dostupná na internetu, proto není problém najít potřebné informace pro implementaci čtení tohoto systému [8]. Následující popis systému vychází ze zmíněné dokumentace, případně dalších uvedených zdrojů. Navíc, jelikož jde o dobře známý souborový systém, existují články usnadňující pochopení základních principů [14]. Existuje zde známe omezení na velikost souboru, který nesmí přesáhnout 4GB. Ovšem nepřepokládá se, že soubor s instrukcemi simulace bude dosahovat takové velikosti. Ohledně přístupu k datům je datový prostor rozdělen na bloky o délce 512 bytů, identicky jako u SD karty. Adresa jednotlivých bloků nese označení LBA (logical block address) [17]. Terminologie v dokumentaci systému označuje tyto bloky jako sektory. Proto se v této kapitole bude pro blok dat používat označení sektor.

MBR

Prvním krokem při rozboru souborového systému je načtení úplně prvního sektoru datového prostoru, tedy sektoru na `LBA = 0`. V tomto sektoru se nachází takzvaný MBR (Master Boot Record) [9] [18]. Protože součástí vytvoření souborového systému je také vytvoření oddílů. Je nutné zjistit v jakém oddílu a na jaké adrese se samotný souborový systém nachází. A tyto infomace se právě nachází v MBR. Strukturu MBR popisuje obrázek 4.16.



Obrázek 4.16: Struktura sektoru MBR (Master boot record)

Prvních 446 bytů MBR obsahují data potřebná k bootování operačního systému počítače. V této situaci nejsou důležitá. Poté následují záznamy o oddílech, kde každý záznam má délku 16 bytů s formátem ukazující obrázek 4.17. MBR obsahuje záznamy pouze o 4 oddílech, pokud je jich na médiu více, používá se pro to technika zvaná rozšířené oddíly. Jelikož je ale simulátor očekává SD kartu s jedním oddílem, nebudou rozšířené oddíly z důvodu jednoduchosti podporovány. Ovšem je dobré pokusit se najít FAT32 oddíl alespoň ve všech 4 záznamech a nekontrolovat pouze ten první. Předposlední byte MBR má vždy hodnotu `0x55` a poslední hodnotu `0xaa`. Je dobré pokaždé kontrolovat tyto byty, protože pokud mají jinou hodnotu souborový systém je pravděpodobě poškozený.

Obrázek 4.17: Struktura záznamu o oddílu v sektoru MBR

V záznamu oddílu jsou důležité pouze zvýrazněné pole `Partition type` a `Start LBA`, ostatní nejsou pro tuto aplikaci podstatné. Podrobnější informace o těchto polích obsahuje tabulka 4.8. Pole `Partition type` obsahuje infomaci, jaký souborový systém obsahuje daný oddíl, pro FAT32 bude tato hodnota `0x0b` nebo `0x0c`. Prázdný oddíl má přiřazenou hodnotu `0x00`. Nejprve je tedy potřeba najít požadovaný oddíl, který obsahuje souborový systém FAT32. A poté lze z pole `Start LBA` načíst LBA sektoru, ve kterém začíná samotný souborový systém. Pozor na to že veškeré více-bytové hodnoty jsou uloženy se zarovnáním little-endian. Pokud MBR neobsahuje žádný oddíl se souborovým systémem FAT32, pak nelze pokračovat dále.

Název	Pozice v rámci sektoru	Pozice v rámci záznamu	Délka (byty)
Partition type	446 + N*16 + 4	4	1
Start LBA	446 + N*16 + 8	8	4
Signature	510	-	2

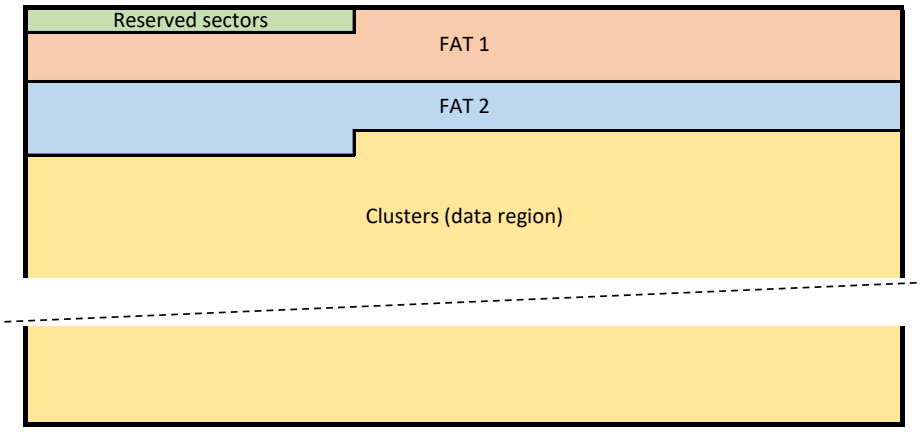
Tabulka 4.8: Důležité hodnoty v sektoru MBR. Všechny pozice jsou počítány od 0. N nabývá hodnoty 0-3.

Z MBR je tedy nutné zjistit LBA oddílu se souborovým systém a uložit si tuto hodnotu pro další použití.

$$partition\_start\_lba = Start\ LBA \tag{4.4}$$

Organizace dat v souborovém systému FAT32

Než je možné začít načítat samotný souborový systém, je nutné si nejpre ujasnit, jak jsou data v celém systému organizována. Organizaci ukazuje obrázek 4.18



Obrázek 4.18: Struktura souborového systému FAT32

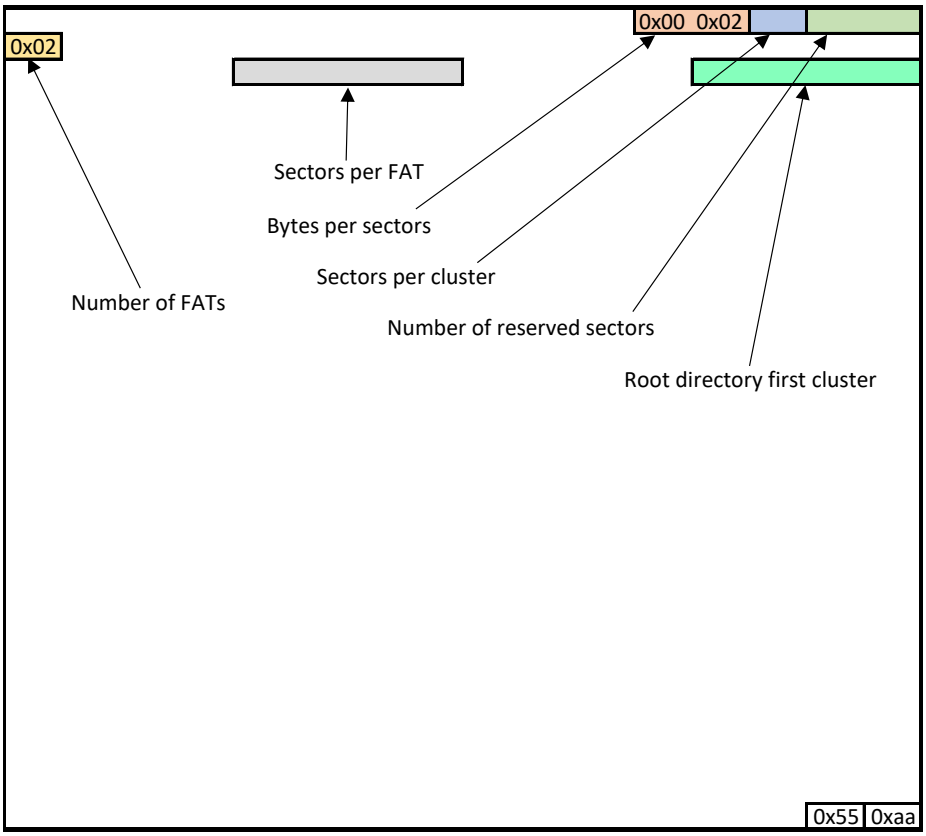
Na začátku souborového systému se nacházení rezervované sektory. První z nich nese označení VBR (Volume boot record) a obsahuje důležité informace o struktuře systému např. počet rezervovaných sektorů, počet sektorů zabírající FAT tabulka atd. Ostatní rezerovované sektory lze ignorovat.

Hned za rezerovanými sektory mají své místo FAT (File Alocation Table) tabulky. Jelikož FAT32 vznikal v dobách, kdy úložistě byla značně nespolehlivá, tak je FAT tabulka uložena zpravidla dvakrát pro případ, že vznikne vadný sektor v jedné z tabulek. Kolik tabulka zabírá sektorů lze zjistit právě ze sektoru VBR. Ve FAT tabulce jsou uloženy informace o tom, kde se nachází soubory, konkrétní princip je vysvětlen dále.

Poslední část obsahuje samotná data souborů a adresářů. Tuto část FAT32 rozděluje na větší části než sektory, takzvané clustery. Cluster je množina několika sektorů a je to nejmenší jednotka, kterou může soubor v systému zabírat. Soubory a adresáře se nachází právě v tomto prostoru rozděleny po clusterech. Kolik sektorů zabírá jeden cluster je opět uvedeno v rezervovaném sektoru VBR. Pozor na číslování clusterů, které začíná číslem 2.

Struktura sektoru VBR

Nyní lze přistupit k rozboru samotého souborového systému. Prvním sektorem systému je zmíněný VBR [10] na `LBA = partition_start_lba`. Strukturu sektoru ukazuje obrázek 4.19.



Obrázek 4.19: Struktura rezervovaného sektoru VBR (Volume boot record)

Název	Pozice v rámci sektoru	Délka (byty)
Bytes per sector	11	2
Sectors per cluster	13	1
Number of reserved sectors	14	2
Number of FATs	16	1
Sectors per FAT	36	4
Root directory first cluster	44	4
Signature	510	2

Tabulka 4.9: Důležité hodnoty v sektoru VBR. Všechny pozice jsou počítány od 0.

Pole `Bytes per sector` má vždy hodnotu 512 a pole `Number of FATs` je vždy 2. Podobně jako u sektoru MBR se na konci VBR nachází 2 byty `0x55` a `0xaa`. Všechny tyto pole je vhodné opět ověřit, pro kontrolu zda není sektor poškozený. Pole `Root directory first cluster` obsahuje číslo prvního clusteru kořenového adresáře. Pole `Sectors per cluster` je nutné opět uložit pro další použití a pomocí ostatních polí lze spočítat LBA začátku FAT tabulky a části obsahující clustery

Uložení potřebných hodnot a rovnice výpočtu daných LBA jsou následující:

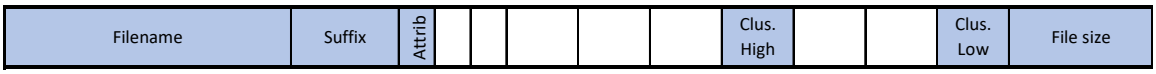
$$\begin{aligned} root\_dir\_first\_cluster &= Root\ dir\ first\ cluster \\ sectors\_per\_cluster &= Sectors\ per\ cluster \\ fat\_start\_lba &= partition\_start\_lba + Number\ of\ reserved\ sectors \\ clusters\_start\_lba &= fat\_start\_lba + (Number\ of\ FATs * Sectors\ per\ FATs) \end{aligned} \tag{4.5}$$

Od této chvíle bude nutné převážně načítat data z části clusterů. Proto je nutné vědět jakým způsobem převést číslo clusteru na LBA prvního sektoru daného clusteru. Poté je samozřejmě nutné načíst ostatní sektory cluteru které jsou logicky za sebou. Výpočet LBA prvního sektoru je následující:

$$lba = clusters\_start\_lba + ((Cluster\ number - 2) * sectors\_per\_cluster) \tag{4.6}$$

Kořenový adresář

Díky informaci o čísle prvního clusteru kořenového adresáře, lze začít načítat obsah tohoto adresáře. Záznamy kořenového adresáře mají délku 32 bytů, v jednom sektoru jich je tedy přesně 16. Strukturu záznamu v adresáři popisuje obrázek 4.20. Důležitá pole jsou vypsány v tabulce 4.10.



Obrázek 4.20: Struktura záznamu adresáře souborového systému FAT32

Název	Pozice v rámci záznamu	Délka (byty)
Filename	0	8
Suffix	8	3
Attribute	11	1
File first cluster high	20	2
File first cluster low FAT	26	2
File size	28	4

Tabulka 4.10: Důležité hodnoty záznamu adresáře souborového systému FAT32. Všechny pozice jsou počítány od 0.

Pole `Filename` obsahuje znaky názvu souboru. Avšak je zde výjimka pro první byte, ten může obsahovat hodnotu `0xe5`. Tato hodnota není ASCII tabulkou definována a znamená, že soubor nebo adresář, na který záznam referuje není platný nebo je smazaný. Pokud má první byte hodnotu `0x00`, pak předchozí záznam v adresáři byl posledním. Při aktivním záznamu byty nesou ASCII znaky jména souboru. Důležité je zmínit, že znak tečky v poli `Filename` neobsahuje a operační systém ho přidává automaticky Pokud je jméno kratší než 8 znaků zbylé byty jsou doplněny znakem mezery. Pole `Suffix` obsahuje znaky přípony souboru.

Byte `Attrib` obsahuje atributy záznamu. Význam jednotlivch bitů ukazuje obrázek 4.21 Důležitý je především bit `Directory` pomocí kterého lze zjistit, jak název napovídá, zda je daný záznam soubor nebo podadresář.

Obrázek 4.21: Význam bitů v poli `Attrib` v záznamu adresáře souborového systému FAT32

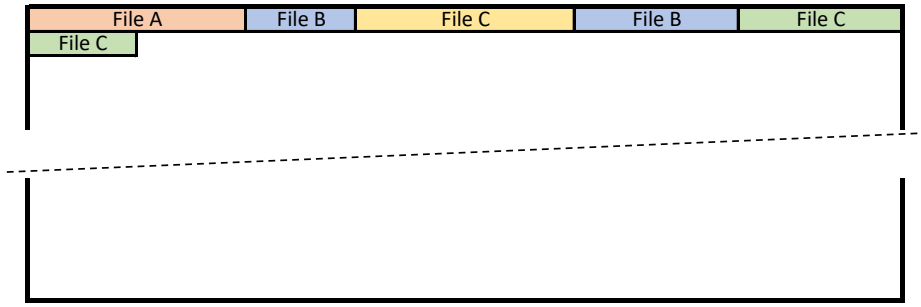
FAT32 podporuje i delší jména souborů, než pouze 8 znaků a to pomocí techniky také nazývané VFAT. Spočívá v přidávání speciálních záznámů s hodnotou pole `Attrib`, která není očekávaná u normálního souboru nebo adresáře. Konkrétné jde o hodnotu `0x0f`, tedy nastavení spodních 4 bitů na log. 1. Úplný princip dlouhých jmen souboru zde nebude popisován, protože simulátor bude načítat soubor se jménem `program.txt`. Je tedy vyhledáván záznam s obyčejným jménem souboru.

Pole `Clus. high` obsahuje horních 16 bitů čísla prvního clusteru souboru. Pole `Clus. low` logicky obsahuje spodních 16 bitů. Pozor na to, že tyto 2 pole je potřeba nejprve převést ze zarovnání little endian a až poté spojit do 32-bit čísla. Poslední pole `Filesize` obsahuje velikost souboru v bytech. Tato hodnota je důležitá, aby bylo jasné kde končí soubor uprostřed clusteru.

Řetězce clusterů

Pomocí těchto již známých informací lze naleznout požadovaný soubor a načíst jeho první cluster. Ale jak u kořenového adresáře, tak i u souboru, je vždy dostupná pouze informace o prvním clusteru. Pro načítání dalších clusterů souboru nebo adresáře slouží doposud opomíjená FAT tabulka. Celá tabulka je velké pole 32-bit čísel. Každý sektor FAT tabulky tedy obsahuje 128 těchto čísel. Každé jedno číslo náleží jednomu clusteru z datové části a obsahuje číslo clusteru který následuje. Hodnota `0x?ffffff8-0x?fffffff` znamená, že aktuální cluster je posledním cluterem souboru. Kde přesně končí soubor v clusteru je nutné zjistit pomocí informace o délce souboru ze záznamu v adresáři.

Pokud je nutné načíst další cluster souboru nebo adresáře, postup je následující. 31-7 bit čísla aktuálního clusteru značí číslo sektoru ve FAT tabulce, v kterém se nachází číslo dalšího clusteru. Proto je nutné načíst tento sektor FAT tabulky. První sektor tabulky je na `LBA = fat_start_lba`. 6-0 bit čísla aktuálního clusteru značí pozici čísla v daném sektoru ve FAT tabulce. Číslo na této pozici značí číslo dalšího clusteru, který je možné načíst. Ovšem pokud se nejedná o poslední cluster.



Příklad několika souborů a jejich reprezentaci v tabulce FAT ukazují obrázky 4.22 a 4.23.

Obrázek 4.22: Příklad rozmístění souborů do clusterů v souborovém systému FAT32

0x????????	0x????????	0x00000003	0x00000004
0x00000005	0xffffffff	0x00000007	0x0000000c
0x00000009	0x0000000a	0x0000000b	0xfffffffff
0x0000000d	0x0000000e	0xffffffff	0x00000010
0x00000011	0x00000012	0x00000013	0xfffffffff
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000
0x00000000	0x00000000	0x00000000	0x00000000

Obrázek 4.23: Příklad sektoru FAT tabulky při rozmístění souborů z obrázku 4.22

4.6 Interpretování pseudokódu

Poslední částí softwaru je samotný interpret pseudokódu. Interpret využívá funkcí pro rozbor souborového systému. Stojí tedy na samotném vrcholu implemetačních částí a nepřímo využívá každou z nich. Od funkcí čtení souboru po ovládání budičů LED diod. Rozhraní pro čtení souboru poskytuje následující funkce:

- read - Načtení znaku z čtecí pozice.
- next - Posunutí čtecí pozice o jeden znak dále.
- set - Nastavení konkrétní čtecí pozice.
- get - Získání čtecí pozice.

Čtecí pozice je ukazatel na aktuálně načítaný znak ze souboru. Číslování je samozřejmě od nuly.

Instrukční sada

Instrukce pseuodkódu jsou shrnuty v příloze B. Tato podkapitola se věnuje návrhu celé instrukční sady vzhledem k povaze zařízení. Z konceptu simulátoru vycházejí klíčové funkce pseudokódu jako následující:

1. Nastavení intervalu pro uspání simulátoru.
2. Ovládání LED diod.
3. Odesílání znaků na terminál.
4. Vyčkání určitého času než se provede další instrukce.

První požadavek řeší instrukce `SLEEP` s parametrem určující dobu spánku. Pokud tento příkaz není v souboru nalezen a simulátor přejde do spánkováho režimu, uspí se na výchozí hodnotu předdefinovanou v kódu simulátoru. Následující dva funkční požadavky umožňují příkazy `LED` a `TER`. Příkaz pro výpis na terminál nabízí určité předprogramované escape sekvence pro zjednodučení vypisování znaků např. vypsání nového řádku atd. Instrukce `WAIT` umožňuje vyčkání určitého času před započetím vykonávání další instrukce.

Tím jsou splněny všechny základní funkční požadavky. A již pomocí těchto 4 instrukcí lze úspěšně simulovat výstupy minipočítače. Avšak v pseudokódu se na více místech začnou objevovat pasáže opakujícího se kódu. Proto instrukční sada obsahuje také příkazy `FOR` a `ENDFOR`, umožňující opakování dané části kódu. Každý takový cyklus vytváří proměnnou udržující infomaci s aktuálním číslem opakování cyklu, kterou lze použít v příkazu `TER`. Implementace této funkcionality není náročná a přitom usnadní hodně práce při vytváření pseudokódu a především zlepší jeho přehlednost.

Nabízí se myšlenka pro implementaci dalších řídích instrukcí jako například `IF`, `ELSE` atd. Jenže při hlubších pokusech o implementaci těchto příkazů lze zjistit, že s sebou přináší velké množství problémů. Například zmíněný příkaz `IF` musí správně interpretovat vyhodnocovací výraz. V tomto výrazu se mohou vyskytovat aritmetické nebo logické operace a pro jejich zpracování by již bylo nutné rozložit výraz na strom jednotlivých operací. A jelikož by tento příkaz byl spíše doplňkovým a usilí silně převažuje užitek, pak tento příkaz není implementován. Přeci jenom cíl pseudokódu je pouze instruovat simulátor jaká data postupně zobrazovat a ne implementovat složité algoritmy.

Implementace instrukcí

Tato podkapitola popisuje jakým způsobem jsou vnitřně implementovány příkazy. Každý příkaz se nachází na samostatném řádku. Začíná identifikátorem a je následován argumenty odělenými jednou mezerou, příkaz konží znakem nového řádku. Před příkazem a za příkazem se mohou vyskytovat bílé znaky. Řádek začínající znakem `#` je brán jako komentář a je ignorován. Samotné načítání instrukcí probíhá znak po znaku s kontrolou formátu a extrakcí parametrů. Vzhledem k pojmenování instrukcí je možné již z prvního znaku instrukce rozhodnout o jakou instrukci se jedná a jaký bude následovat formát.

SLEEP

Parametry:

1. Číslo v rozsahu `<0 - 4 294 967 295>`. Časový interval v minutách.

Tato instrukce nemá na vykonávaný kód žádný účinek, pouze nastaví vnitřní proměnnou na zadaný čas uspání. S touto proměnnou se poté porovnává čas strávený ve spátkovém režimu. Nastavení provede vždy a pouze když ji interpret provede. Pokud interpret na tuto instrukci v pseudokódu nenarazí, simulátor bude uspán na výchozí čas který činí 15 minut. Pokud je instrukce v psedokódu vícekrát, vnitřní proměnná je při vykonání instrukce přepsána.

WAIT

Parametry:

1. Číslo v rozsahu `<0 - 4 294 967 295>`. Časový interval v milisekundách.

Instrukce blokuje vykonávání další instrukce po stanovený čas. Čekání je implementováno bez časovače přímo ve vykonáváném kódu. Jelikož by simulátor v době čekání neměl jinou práci, je zbytečné řešit toto čekání pomocí časovače a přerušení. Nicméně čekání je řešeno tak, aby neblokovalo zastavení vykonávání programu tlačítkem.

TER

Parametry:

1. Řetězec znaků až na konec řádku. Řetězec k vypsání na terminál.

Instrukce vypisuje na terminál znak po znaku řetězec v parameteru. Parametr podporuje escape hodnoty, jelikož jsou potřeba pro speciální funkce terminálu. Je možné zadat i přesnou ASCII hodnotu, kterou je potřeba odeslat na terminál. Mimo to se v řetezci může vyskytovat proměnná cyklu (viz dále). Hodnota proměnné cyklu je odeslána jako sekvence ASCII čísel reprezentující tuto hodnotu v desítkové soustavě.

LED

Parametry:

1. Číslo v rozsahu `<0 - 31>`. Index LED diody.
2. Číslo v rozsahu `<0 - 1>`. Požadovaný stav LED diody, 1 - rozsvítit, 0 - zhasnout.

Tato instrukce změní stav dané LED diody ve vnitřní promenné stavů LED diod a odešle budičům novou konfiguraci rozsícení LED diod.

FOR

Parametry:

1. Znak v rozsahu `<a - z>`. Písmeno proměnné v které je uložené aktuální číslo počítadla opakování.
2. Číslo v rozsahu `<0 - 4 294 967 295>`. Počáteční číslo počítadla opakování.
3. Číslo v rozsahu `<0 - 4 294 967 295>`. Konečné číslo počítadla opakování.

Cyklus je nejsložitější instrukce celého interpretu. Interpret podporuje zanoření více cyklů do sebe. Toto zanoření je řešeno pomocí zásobníku struktur cyklů. S instrukcí `FOR` je vytvořen nový záznam o cyklu a přidán na vrchol zásobníku. Tento záznam obsahuje konečnou hodnotu opakování, čtecí pozici prvního znaku první instrukce v bloku cyklu a index do tabulky proměnných.

Každý cyklus má přiřazenou proměnnou s libovolným jednopísmenným identifikátorem. Hodnota této proměnné obsahuje aktuální číslo počítadla opakování a všechny proměnné jsou uloženy v tabulce proměnných, která je implementování jako pole. Pořadí identifikátoru v abecedě je indexem do tohoto pole. Díky této nezávislé tabulce lze přistupvat k hodnotám proměnných jednoduše z příkazu `TER`.

ENDFOR

Instrukce nemá žádné parametry. Tato instukce kontroluje, zda aktuální cyklus (ten na vrcholu zásobníku) nepřekročil počet opakování. Tedy zda hodnota aktuálního počítala opakování je menší než konečné číslo počítadla opakování. Pokud ano, přičte číslo 1 k aktuální hodnotě počítadla opakování cyklu a pokračuje v opakování instrukcí cyklu. V opačném případě odstraní aktuální cyklus ze zásobníku cyklů a pokračuje následující instrukcí.

Kapitola 5

Testování

V průběhu vývoje se samozřejmě vyskytlo množství problémů. Bohužel vzhledem k povaze práce jedna chyba často znamená výrobu nové desky tištěných spojů. Naštěstí na začátku vývoje lze začít programovat i bez vyrobené desky, ovšem je potřeba znát alespoň obecné budoucí zapojení desky. Pro mikrokontroléry AVR existují simulátory, které dokáží simulovat nahraný kód a podle něj generovat průběhy vstupně výstupních pinů. Tuto možnost jsem využil při implementaci obsluhy budičů LED diod, ale například pro obsluhu SD karty již simulátor nebylo možné použít. Nicméně díky této možnosti jsem mohl v začátcích kód obsluhy budičů odladit bez fyzického MCU, protože jinak bych musel ladit pomocí osciloskopu. Při odzkoušení na reálném zařízení, kód pro obsluhu budičů fungoval na první pokus.

Bohužel deska tištěných spojů jako samotná na první pokus nefungovala. V první verzi jsem udělal příliš mnoho chyb a to především ohledně zapojení převodníků napěťových úrovní. Chyby jsem objevil bohužel, až na vyrobené desce pomocí osciloskopu a voltmetru, když se deska nechovala tak, jak měla. Nicméně ani druhá deska nebyla dokonalá a také měla své chyby avšak ne natolik zásadní, aby nešly jednoduše opravit. Druhá verze hardwarového návrhu byla ale, sice po drobných úpravách prokelmováním či prošrábnutím cesty, funkční a schopná simulace.

Na této desce probíhalo hlavní ladění a vývoj software. Zásadní bylo nejprve odzkoušet zda se periferie a celá deska chová podle očekávání. Jestli regulátor pro 3.3V opravdu generuje 3.3V, jestli na desce není nějaký zkrat a jestli převodníky napětových úrovní fungují podle očekávání. Rozhraní pro RS232 po opravě z první verze desky při připojení k terminálu úspěšně odesílalo znaky a komunikace s SD kartou také po opravě začala fungovat podle očekávání.

Samotný postup ladění softwaru sestával z připojeného konektoru pro terminál k USB redukci pro sériovou linku. Na této sériové lince bylo možné naslouchat pomocí programu `putty`. Tato možnost velmi usnadnila proces ladění vzhledem k možnosti výpisu libovolného textu na terminál. Nic nebránilo využít jako terminál reálný terminál minipočítače, ale `putty` nabízí pohodnější a především méně hlučné řešení.

Proces vývoje software měl povahu stavění funkčních bloků na sebe. Nejprve bylo nutné zprovoznit komunikaci pomocí sběrnice SPI. Poté zapouzdřit do funkce odesílání příkazů SD kartě. Pomocí této funkce bylo již možné vytvořit sekvenci inicializace SD karty. Teprve s incializovanou kartou lze načítat data z SD karty a začít zpracovávat souborový systém. Předposledním krokem bylo zapouzdřit funkce souborového systému na funkce pro čtení znaku na zadané adrese ze souboru. A tyto funkce nakonec využívá interpret a nejsou mu vůbec známy žádné clustery nebo příkazy SD karty.

U samotného testování zařízení jsem se soustředil především na SD kartu. Je nutné vyzkoušet více SD karet, jak už starých ze šuplíku, tak nově koupených. Ohledně souborového systému FAT32 existuje nepatrný rozdíl ve způsobu formátování mezi operačními systémy, bylo tedy nutné vyzkoušet zda i tak simulátor souborový systém rozpozná. Zkusil jsem i různě zatížit souborový systém, aby kořenový adresář byl větší než jeden cluster a hledaný soubor se nacházel někde mezi spoustou souborů. Ohledně testování interpretu není moc situací které lze vytvořit, avšak zkoušel jsem různé chybné zápisy instrukcí, zda na to interpret zareaguje. Nicméně těchto situací je konečné množství a lze je rozumným způsobem otestovat.

Poslední část testování znamenala zkouška výpisů na terminál s využitím speciálních znaků a sekvencí, které terminál umožňuje. Pomocí manuálu [4] lze zjistit jaké možnosti terminál nabízí a je nutné vyzkoušet zda pseudokód umí využít těchto možností. Terminál například nabízí více znakových sad, na které se lze přepnout specifickou escape sekvencí. Poté má každá ASCII znak jinou reprezentraci vzhledem k vybrané znakové sadě. Další funkce jsou například posun kurzoru, blikající text, různé úrovně intenzity podsvícení znaku apod. Všechny tyto funkce využívají pouze espcae sekvencí. A jelikož pseudokód podporuje escape sekvence nenarazil jsem na omezení ze strany pseudokódu ohledně využívání speciálních funkcí terminálu.

Kapitola 6

Závěr

Vizi simulátoru oživující starý minipočítač HP 3000 se podařilo úspěšně naplnit. Podařilo se vytvořit zařízení, které dokáže komunikovat s SD kartou, načíst pseudokód a podle něho simulovat výstupy minipočítače. V průběhu práce jsem řešil mnoho úskalí, která tento projekt přinášel. Naštěstí jsem vždy nalezl způsob, jak většinu problémů vyřešit nebo jsem alespoň nalezl alternativní postup k cíli. Tento projekt pro mě měl velký přínos nejenom v množství zkušeností s návrhem vestavěného zařízení, ale také ve znalostech souborového systému FAT32 a komunikace MCU s SD kartou. Byla to výborná příležitost zkusit si vytvořit vestavěné zařízení od začátku až do konce a projít si celým procesem. Zjistil jsem, že je to opravdu komplexní činnost a možnosti kde udělat chybu, jsou na každém rohu. Věřím, že díky mému usilí, oživený minipočítač upoutá pozornost nejednoho studenta. Možná, se tak dostane i k této práci a zjistí, že svět mikrokontrolérů a vestavěných systémů může být zajímavý.

Ovšem možností pro rozšíření simulátoru je mnoho a o některých vím již teď. Například intrukční sadu lze v budoucnu rozšířit o další instrukce pro umožnění zápisu komplexnějších pseudokódů. Instrukce typu "if" nebo podpora vlastních proměnných může rozšířit možnosti pseudokódu, ovšem je na uváženou, zda by to bylo užitečné. Jelikož již stávající hardwarový design umožňuje i přijímat znaky z klávesnice terminálu. Nic nebrání rozšířit aktuální simulátor o reakci na vstup z klávesnice terminálu. Toto rozšíření by mohlo být dopracováno do takové úrovně, že simulátor nebude simulovat pouze výstupy, ale i vnitřní procesy minipočítače. Ovšem předpokládám, že pro takové zadání by již stávající hardwarový návrh s mikrokontrolérem AVR nepostačoval. Nicméně již stávající zařízení s aktuální funčností lze spolehlivě použít pro simulaci výstupů nejen minipočítače HP 3000. Budu rád, pokud někdo mojí práci využije v rámci svého studia nebo se například inspiruje a použije tuto práci jako odrazový můstek pro svůj vlastní projekt.

Literatura

[1] *Bit savers, Index of /pdf/hp/3000* [online]. [cit. 2021-07-20]. Dostupné z: http://www.bitsavers.org/pdf/hp/3000.

[2] Abisys. *DB-25 male connector, via Wikimedia* [online]. 2008 [cit. 2021-07-20]. Dostupné z: https://commons.wikimedia.org/w/index.php?curid=4502539.

[3] Cha, N. *How to Use MMC/SDC* [online], 26. prosince 2019 [cit. 2021-07-20]. Dostupné z: http://elm-chan.org/docs/mmc/mmc_e.html.

[4] Hewlett Packard Company . *2641A 2645A 2645S/N Display Station Reference Manual* [online]. Hewlett-Packard Company, listopad 1978 [cit. 2021-07-20]. Dostupné z: http://www.bitsavers.org/pdf/hp/terminal/264x/2645A/02645-90005_Nov-1978.pdf.

[5] Maxim Integrated Products Inc. . *+-15kV ESD-Protected, 1uA, 16Mbps, Dual/Quad Low-Voltage Level Translators in UCSP MAX3392E Data Sheet* [online]. Maxim Integrated Products Inc., prosinec 2006 [cit. 2021-07-20]. Dostupné z: https://www.tme.eu/Document/7a3d45375d3e2689b56cf9803ed2b400/max3372e.pdf.

[6] Microchip Technology Inc. . *2.0 MHz, 500 mA Synchronous Buck Regulator MCP1603 Data Sheet* [online]. Microchip Technology Inc., říjen 2012 [cit. 2021-07-20]. Dostupné z: https://www.tme.eu/Document/518d7346d2a9f8d39ef0a65f11d90c2b/MCP1603.pdf.

[7] Microchip Technology Inc. . *ATmega164A/PA/324A/PA/644A/PA/1284/P megaAVR® Data Sheet* [online]. Microchip Technology Inc., leden 2020 [cit. 2021-07-20]. Dostupné z: https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega164A_PA-324A_PA-644A_PA-1284_P_Data-Sheet-40002070B.pdf.

[8] Microsoft Corporation . *Microsoft Extensible Firmware Initiative FAT32 File System Specification* [online]. Microsoft Corporation, prosinec 2000 [cit. 2021-07-20]. Dostupné z: http://download.microsoft.com/download/1/6/1/161ba512-40e2-4cc9-843a-923143f3456c/fatgen103.doc.

[9] Microsoft Corporation . Master Boot Record. *Microsoft Docs* [online], 09. listopadu 2008 [cit. 2021-07-20]. Dostupné z: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc976786(v=technet.10)?redirectedfrom=MSDN.

[10] Microsoft Corporation . Boot sector. *Microsoft Docs* [online], 09. listopadu 2008 [cit. 2021-07-20]. Dostupné z: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc976796(v=technet.10)?redirectedfrom=MSDN.

[11] Renesas Electronics Corporation . *RS-232 Transmitters/Receivers ICL3221E Data Sheet* [online]. Renesas Electronics Corporation, květen 2019 [cit. 2021-07-20]. Dostupné z: https://www.tme.eu/Document/2348b36127e85d70e3afe54383577f45/icl3221e-22e-23e-32e-41e-43e.pdf.

[12] SD Association . Physical Layer Simplified Specification. *Simplified Specifications | SD Association* [online], 23. září 2019 [cit. 2021-07-20]. Dostupné z: https://www.sdcard.org/downloads/pls/.

[13] StarChipsTechnology Inc. . *16-bit Serial-In/Parallel-Out Constant-Current LED Driver SCT2026 Data Sheet* [online]. StarChipsTechnology Inc., leden 2008 [cit. 2021-07-20]. Dostupné z: https://www.tme.eu/Document/b79eb201a2b4ac08588ea629c1381be2/sct2026.pdf.

[14] Stoffregen, P. Understanding FAT32 Filesystems. *Paul's 8051 Code Library* [online], 24. února 2005 [cit. 2021-07-20]. Dostupné z: https://www.pjrc.com/tech/8051/ide/fat32.html.

[15] Strangio, C. E. *The RS232 standard* [online]. 1997 [cit. 2021-07-20]. Dostupné z: https://mil.ufl.edu/3744/docs/RS232_standard_files/RS232_standard.html.

[16] Texas Instruments . *KeyStone Architecture Serial Peripheral Interface (SPI)* [online]. Texas Instruments, březen 2012 [cit. 2021-07-20]. Dostupné z: https://www.ti.com/lit/ug/sprugp2a/sprugp2a.pdf.

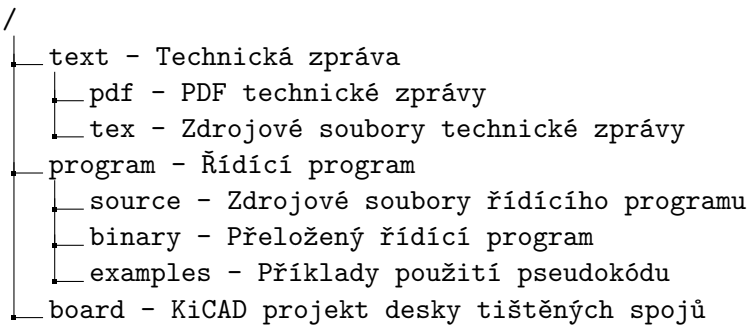
[17] Wikipedia. *Logical block addressing — Wikipedia, The Free Encyclopedia* [online]. 2021 [cit. 2021-07-20]. Dostupné z: https://en.wikipedia.org/wiki/Logical_block_addressing.

[18] Wikipedia. *Master boot record — Wikipedia, The Free Encyclopedia* [online]. 2021 [cit. 2021-07-20]. Dostupné z: https://en.wikipedia.org/wiki/Master_boot_record.

[19] Wikipedia. *RS-232 — Wikipedia, The Free Encyclopedia* [online]. 2021 [cit. 2021-07-20]. Dostupné z: https://en.wikipedia.org/wiki/RS-232.

Příloha A

Obsah přiloženého média



Příloha B

Manuál k obsluze simulátoru

Obecný popis

Simulátor se připojuje k minipočítači pomocí dvou konektorů, konkrétně konektoru DB-25 pro připojení terminálu a pinové lišty pro připojení LED diod panelů. Konektor terminálu se připojuje tak jak je. Pozor, termiál musí být přepnut do režimu příjímání znaků po sériové lince. Pinová lišta sestává z matice zdířek kde horní řada náleží katodám LED diod a spodní anodám. Oba vývody pro jednu LED diodu jsou tak vždy nad sebou a tím pádem jsou jednotlivé LED pomyslně řazeny vedle sebe. Na levé straně vystupují 2 vodiče. První označený 5V vede ke zdroji napětí pro celý simulátor. Druhý dvoužilový vodič obsluhuje relé spínající terminál. Ná této straně se nachází také hlavní přepíč spínající napajení celého zařízení.

Na přední straně se nachází konektor pro SD kartu. SD karta musí být typu `SC`, `HC` nebo `XC`. Musí být naformátována souborovým systémem FAT32 se souborem `program.txt` v kořenovém adresáři. Tento soubor obsahuje pseudokód instrukcí simulace.

Vrchní strana simulátoru obsahuje kontrolní LED diody a kontrolní tlačítka. První LED dioda z leva má červenou barvu a pokud svítí je simulátor připojen k napájecímu napětí a je zapnutý. Ostatní LED diody indikují aktuální stav simulátoru. Obecný význam jednotlivých LED diod je následující. Žlutá LED dioda indikuje aktivní režim automatického spouštění simulace. Zelená LED dioda indikuje spuštěnou simulaci. Červená LED dioda indikuje vznik chyby. Kontrétní význam kombinací indikací LED diod vysvětluje tabulka B.1.

K ovládání simulátoru slouží kontrolní tlačítka. Zelené tlačítko nese označení `RUN` a pomocí něho lze spustit nebo zastavit simulaci. Pomocí žlutého `SLEEP` tlačítka lze přepínat režim automatického spouštění simulace. Konkrétní funkce tlačítek vzhledem ke stavům simulátoru popisuje tabulka B.2

Stavy simulátoru

Ž	Z	Č	Stav	Význam
0	1	1	`INIT`	Probíhá inicializace simulátoru. Průběh je vypisován na terminál.
0	0	0	`IDLE`	Simulátor úspěšně inicializován a je ve stavu připravenosti. Pomocí tlačítek lze sputit simulaci.
1	0	0	`SLEEP`	Spánkový režim, simulace se automaticky spustí za definovaný interval.
0	1	0	`RUNNING_MAN`	Běží ručně spuštěná simulace
1	1	0	`RUNNING_AUTO`	Běží automaticky spuštěná simulace, po skončení přejde simulátor do spánkového režimu
0	0	1	`ERROR`	Při simulaci nebo inicializaci nastala chyba.

Tabulka B.1: Reprezentace stavů simulátoru kontrolními LED diodami

Stav	Tlačítko `RUN`	Tlačítko `SLEEP`
`INIT`	-	-
`IDLE`	Spuštění simulace	Spuštění simulace se zapnutým automatickým spouštěním
`SLEEP`	Předčasné spuštění automatické simulace	Vypnutí automatického spouštění
`RUNNING_MAN`	Zastavení simulace	-
`RUNNING_AUTO`	Předčasné zastavení automatické simulace	-
`ERROR`	Pokus o opětovnou inicilizaci	-

Tabulka B.2: Funkce tlačítek simulátoru vzhledem ke stavům

Formát souboru s pseudokódem `program.txt`

Soubor obsahuje pseudokód s jednoduchou syntaxí. Každý řádek obsahuje jednu instrukci, komentář nebo je prázdný. Na začátku a konci řádku se mohou vyskytovat bílé znaky. Řádky jsou odděleny pomocí ASCII znaku `"LF" (0x0a)`. Řádek s komentářem začíná znakem `"#"` a je interpretem ignorován. Jiný způsob zápisu komentáře není podporován. Řádek s instrukcí začíná identifikátorem instrukce následovaný parametry instrukce oddělelnými znakem mezery. Je podporované zanoření cyklů. Každý cyklus musí mít přiřazenou proměnnou jejíž hodnotu lze vypisovat na terminál.

Instrukce pseudokódu

`SLEEP␣<par1><LF>`

`<par1>` - Číslo v rozsahu `<0 - 4 294 967 295>`. Časový interval v minutách.
Nastavení časového intervalu pro automatické spouštění simualace. Pokud interpret tuto instrukci v pseudokódu nenalezne a simulátor bude uspán, tak na výchozí čas, který činí 15 minut. Pokud je instrukce v psedokódu vícekrát, hodnota časového intervalu je při vykonání instrukce přepsána.

`WAIT␣<par1><LF>`

`<par1>` - Číslo v rozsahu `<0 - 4 294 967 295>`. Časový interval v milisekundách.
Vyčkání časového intervalu před vykonáním další instrukce.

`LED␣<par1>␣<par2><LF>`

`<par1>` - Číslo v rozsahu `<0 - 31>`. Index LED diody.
`<par2>` - Číslo v rozsahu `<0 - 1>`. Požadovaný stav LED diody, 1 - rozsvítit, 0 - zhasnout.
Rozsvícení nebo zhasnutí vybrané LED diody.

`TER␣<par1><LF>`

`<par1>` - Řetězec k vypsání na terminál.
Vypsání řetězce na terminál. Vypisování začíná prvním znakem parametru tedy znakem následujícím hned po znaku mezery. Parametr a tím i výpis končí s koncem řádku. Řetězec může obsahovat escape sekvence začínající znakem `"\"`, které jsou definované následovně. `"\$"` - Odešle na terminál znak `"$"`. `"\\"` - Odešle na terminál znak `"\"`. `"\n"` - Odešle na terminál znak `"CR"(0x0d)` a znak `"LF"(0x0a)`. `"\e"` - Odešle na terminál znak `"ESC"(0x1b)`. `"\c"` - Odešle na terminál znak definovaný následujícím vždy tří ciferným číslem v desítkové soustavě. Odeslání znaku `"a"` by bylo možné pomocí `"\c097"`. K vypsání hodnoty proměnné cyklu je nutné použít znak `"$"` následovaný písmemenem požadavané proměnné. Číslo bude vypsáno v desítkové soustavě pomocí znaků ASCII.

`FOR␣<par1>␣<par2>␣<par3><LF>`

`<par1>` - Znak v rozsahu `<a - z>`. Písmeno proměnné v které je uloženo aktuální číslo počítadla opakování.
`<par2>` - Číslo v rozsahu `<0 - 4 294 967 295>`. Počáteční číslo počítadla opakování.
`<par3>` - Číslo v rozsahu `<0 - 4 294 967 295>`. Konečné číslo počítadla opakování.
Cyklus opakující instrukce mezi touto instrukcí a první následující instrukcí `ENDFOR`. Cyklus se opakuje dokud platí, že hodnota počítadla opakování je menší než konečná hodnota. Cyklus je vždy inkrementační proto musí počáteční hodnota být vždy menší než konečná.

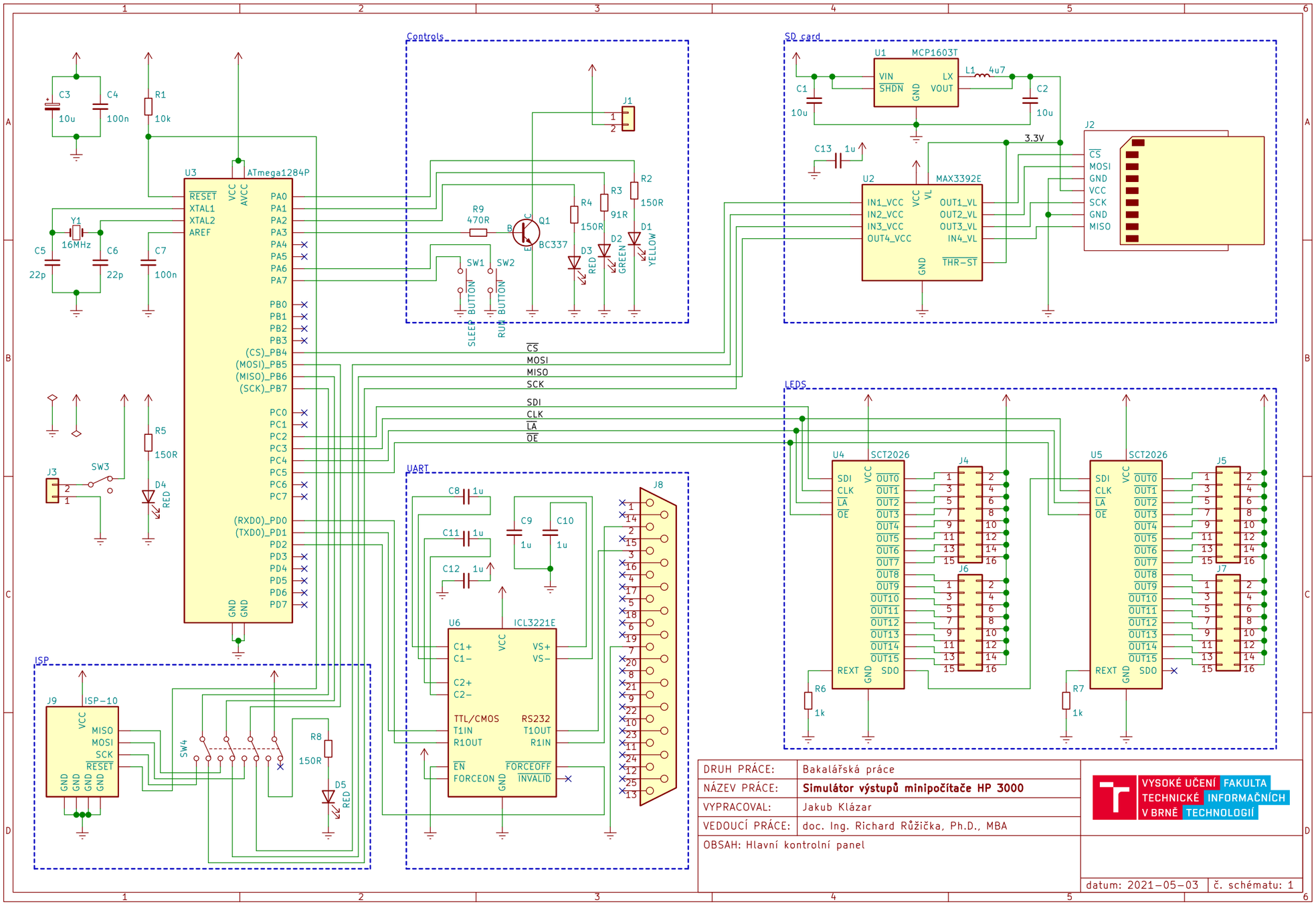
`ENDFOR<LF>`

Instrukce ohraničující konec cyklu.

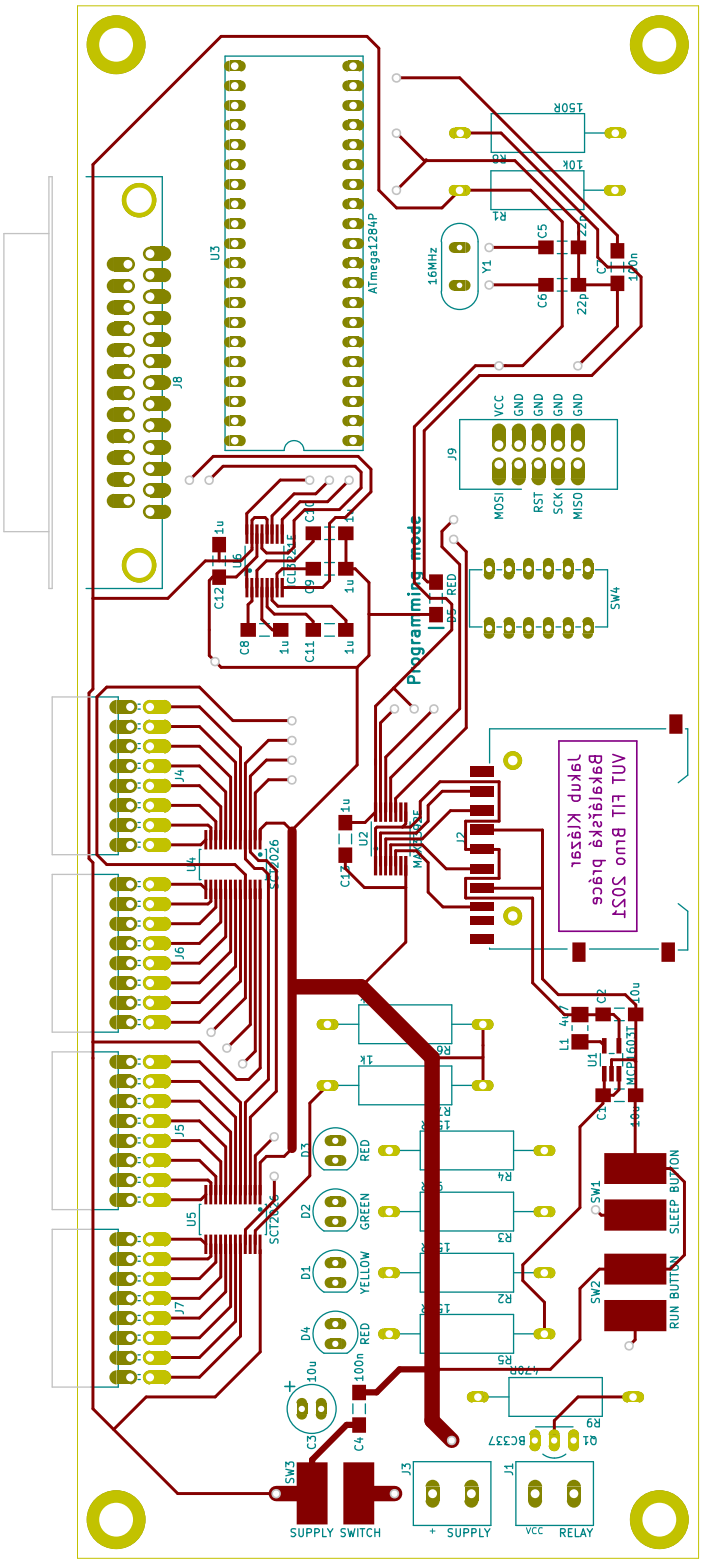
Příloha C

Hardwarový návrh simulátoru

C.1 Elektronické schéma



C.2 Přední strana desky tištěných spojů



C.3 Zadní strana desky tištěných spojů