FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

# Zadání bakalářské práce

| | |
|---|---|
| **Název:** | Inovace řadiče pro grafický LCD displej elektronického psacího stroje |
| **Student:** | Boris Pankovčin |
| **Vedoucí:** | Ing. Pavel Kubalík, Ph.D. |
| **Studijní program:** | Informatika |
| **Obor / specializace:** | Počítačové inženýrství |
| **Katedra:** | Katedra číslicového návrhu |
| **Platnost zadání:** | do konce letního semestru 2022/2023 |

## Pokyny pro vypracování

1) Prozkoumejte existující řešení ovládání LCD displeje psacího stroje typu Triumph-Adler Gabriele PFS.
2) Z dokumentace k radiči MSM6255 analyzujte a na existujícím řešení ověřte průběhy pro ovládání displeje.
3) Na základě získaných dát navrhněte a zrealizujte vlastní řadič displeje s pomocí FPGA obvodu.
4) Pro účel testování propojte navrhnutý radič s vývojovou deskou RaspberryPI.
5) Výsledné řešení řádně otestujte

---

*Elektronicky schválil/a prof. Ing. Hana Kubátová, CSc. dne 31. ledna 2022 v Praze.*

Bakalárska práca

# INOVACE ŘADIČE PRO GRAFICKÝ LCD DISPLEJ ELEKTRONICKÉHO PSACÍHO STROJE

**Boris Pankovčin**

Fakulta informačních technologií
Katedra číslicového návrhu
Vedúci: Ing. Pavel Kubalík, Ph.D.
8. mája 2022

# Obsah

# Zoznam obrázkov

## Zoznam tabuliek



## Zoznam výpisov kódu

*Chcel by som sa poďakovať predovšetkým vedúcemu tejto práce, Ing. Pavlu Kubalíkovi, Ph.D. za jeho neustálu ochotu, trpezlivosť pri konzultáciách a za jeho promptnú komunikáciu. Ďalej chcem poďakovať svojej rodine a kamarátom, ktorý ma podporovali nie len pri písaní tejto práce, ale aj počas celého môjho vysokoškolského štúdia. Bez nich by som dnes tieto slová nepísal. Ďakujem!*

# Vyhlásenie

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 8. mája 2022 .....

# Abstrakt

Táto bakalárka práca sa zaoberá návrhom a realizáciou funkčného prototypu radiča staršieho typu LCD, ktorý bude slúžiť ako náhrada pôvodneho radiča typu MSM6255. Analytická časť práce sa detailne venuje opisu spôsobu riadenia LCD radičom typu MSM6255 na základe jeho dokumentácie. Súčasťou je podrobný opis účelu každého riadiaceho signálu a ako naň reagujú segmentové a spoločné radiče umiestnené priamo na paneli LCD. Praktická časť sa zaoberá implementáciou radiča LCD v jazyku VHDL na základe získaných informácií. Vyvinutý radič dokáže nielen spoľahlivo zobrazovať obrazové dáta z pamäti, je možné ho tiež pripojiť k nadradenému modulu pomocou rozhrania SPI. Vytvorený prototyp je založený na školskej vývojovej doske Basys3. Súčasťou prototypu je tiež mikropočítač Raspberry Pi 3 A+, ktorý je k radiču pripojený rozhraním SPI a slúži na jeho komplexnú validáciu. Praktická časť sa nakoniec zaoberá aj testovaním jednotlivých entít a radiča ako celku.

**Klíčová slova** LCD, SPI, FPGA, Raspberry Pi, MSM6255, MSM5299A, MSM5298A, LCD radič, segmentový radič, spoločný radič, adresovanie LCD

# Abstract

This bachelor thesis deals with the design and implementation of a functional prototype of a controller for an older type of LCD, which will serve as a replacement for the original controller type MSM6255. The analytical part of the thesis deals in detail with the description of the LCD control method using the controller type MSM6255 based on its documentation. Further it includes a detailed description of the purpose for each control signal and how the segment and common controllers placed directly on the LCD panel respond to these signals. The practical part is dedicated to describe implementation of LCD controller in the VHDL language based on the acquired information. The developed controller can not only reliably display image data from the memory, it can also be connected to the master module using the SPI interface. The developed prototype is based on the Basys3 school development board. A Raspberry Pi 3 A + microcomputer is also a part of the prototype, being connected to the controller via an SPI interface as it serves for its comprehensive validation. Finally, the practical part also contains testing of individual entities and the controller as a whole.

**Keywords** LCD, SPI, FPGA, Raspberry Pi, MSM6255, MSM5299A, MSM5298A, LCD driver, segment driver, common driver, LCD addressing

# Zoznam skratiek

| | |
|---|---|
| AMLCD | Active Matrix LCD |
| BRAM | Block RAM |
| D-F/F | Double Flip-Flop |
| FIFO | First In First Out |
| FPGA | Field Programmable Gate Array |
| FSM | Finite State Machine |
| GPIO | General Purpose Input Output |
| HW | Hardware |
| LCD | Liquid Crystal Display |
| LSB | Least Significant Bit |
| LUT | Look Up Table |
| MISO | Master In Slave Out |
| MOSI | Master Out Slave In |
| MSB | Most Significant Bit |
| PMLCD | Passive Matrix LCD |
| RAM | Random Access Memory |
| SPI | Serial Peripheral Interface |

# Kapitola 1
# Úvod

V posledných rokoch je možné v spoločnosti pozorovať silnejúci trend dopytu po retro technológiách, napríklad preto, že ľudí láka dizajn, ktorý v nich evokuje spomienky na mladosť. Zároveň by v ňom chceli mať najnovšie technológie, kvôli jednoduchej obsluhe. Výrobcovia tak idú zákazníkom naproti a vyrábajú úplne nové zariadenia v obale s retro dizajnom. Zároveň je v dnešnej dobe, najmä v rozvinutých krajinách, veľký dopyt po ekológii. V súčastnosti to znamená hlavne to, že na výrobcov bol vyvinutý dostatočný tlak aby sa snažili vyrábať z ľahko recyklovateľných materiálov. Avšak z hľadiska ekológie sa stále nejedná o ten najlepší možný prístup k problému, ktorým je, aspoň z môjho pohľadu, čo najväčšie využitie funkčnej časti starého produktu. Kladiem si teda otázku, či by nebolo možné tieto trendy vzájomne prepojiť.

Ak sa zamerám bližšie na sféru elektroniky, ako ekologický prístup vidím znovupoužitie periférií starších zariadení. Touto cestou by bolo možné ponúknuť spotrebiteľom ekologickejší a pravý retro dizajn, kombinovaný s pohodlnosťou moderných technológií. Bohužiaľ takéto periférie disponujú častokrát rozhraním, ktoré je dnes už možné označiť ako neštandardné. Pre ich úspešné použitie v novom produkte je teda nutné navrhnúť nový radič, ktorý bude schopný zabezpečiť komunikáciu medzi moderným počítačom a danou periférioiu.

## 1.1 Motivácia

Venovať sa tomuto problému ma inšpiroval môj elektronický písací stroj typu Triumph-Adler Gabriele PFS, ktorý je výborným príkladom tohto problému. Veľmi rád by som ho plne využíval pre tlač dokumentov u ktorých by som chcel dosiahnuť autentického štýlu písacieho stroja. Bohužiaľ je pre mňa prakticky nepoužiteľný a to iba kvôli zastaranému programu, procesoru a nedostatočnej pamäti. Totiž ak by som si chcel vytlačiť nejaký dokument, musel by som ho do tohto stroja prepísať ručne. Je síce možné použiť disketu pre uloženie dokumentu, jedná sa však o 3 palcovú disketu, čiže pre počítače ide o neštandardnú veľkosť, nehovoriac o tom, že dnešné počítače už ani neobsahujú mechaniky pre toto médium. Navyše je schopný prečítať iba dokumenty uložené vo formáte špecifickom pre toto zariadenie alebo dokumenty vo formáte, ktorý používal operačný systém MS-DOS.

Rozhodol som sa preto vymeniť starú základnú dosku za nový hardvér tak, aby bolo možné využívať periférie tohto stroja bez nepríjemností starého softvéru. Základom bude RaspberryPi 3, ktoré zabezpečí pohodlné prostredie osobného počítača a zároveň poskytuje výborné možnosti pre napojenie periférii.

## 1.2 Ciele práce

Hlavný cieľom tejto práce je vyvinúť samostatný radič grafického LCD displeja ako náhradu za pôvodný radič typu MSM6255, ktorý sa nachádza priamo na základnej doske. Tento radič bude schopný okrem ovládania displeja aj komunikácie s nadradeným zariadením pomocou rozhrania SPI ktorým bude prijímať obrázky, ktoré následne zobrazí na displeji. Pre dosiahnutie tohoto cieľa budem postupovať v niekoľkých samostatných krokoch.

Keďže neexistuje verejne dostupný ucelený návod ako presne funguje ovládanie LCD pomocou použitého radiča bude prvým krokom analýza existujúceho riešenia z funkčného písacieho stroja a zber informácií z dostupnej dokumentácie k radičom, ktoré boli pre ovládanie LCD pôvodne použité. Takto získané informácie bude potrebné následne náležite overiť meraním priebehu signálov.

Na základe analýzy je ďalším cieľom implementovať základ radiča, ktorý bude schopný spoľahlivo zobraziť dáta na displeji. Pre tento radič zatiaľ nebude potrebná obrazová pamäť, pretože nebude na displeji zobrazovať ucelené obrázky. Jeho cieľom bude nielen potvrdiť, že riadiace a datové signály boli správne analyzované a implementované, ale taktiež overiť, že radič vnútorne správne pracuje s obrazovými súradnicami.

Posledným cieľom tejto práce je prepojiť radič LCD s mikropočítačom Raspberry Pi a otestovať jeho funkcie. Pre naplnenie tohto cieľa bude potrebné rozšíriť samotný základný radič z predchádzajúceho odstavca o časť schopnú prijímať dáta cez rozhranie SPI a vyrovnávaciu pamäť pre prijaté obrázky, ktoré budú čakať na zobrazenie na displeji.

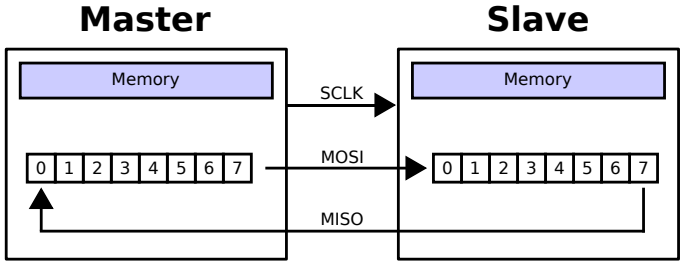
# Kapitola 2
# Teoretický základ

*V tejto kapitole opíšem základné teoretické poznatky týkajúce sa jednotlivých častí, ktoré budú neskôr analyzované, či implementované. Najprv sa budem venovať rozhraniu SPI, potom opíšem niektoré vybrané koncepty jazyka VHDL, ktoré som použil v implementačnej časti tejto práce a nakoniec zhrniem základné charakteristiky LCD a ich adresovania.*

## 2.1 Rozhranie SPI

*Serial Peripheral Interface* (ďalej len SPI), je sériový synchronný komunikačný protokol. Toto rozhranie bolo vyvinuté spoločnosťou Motorola, pôvodne ako jednoduché a lacné rozhranie pre periférne zariadenia. Jednotlivé zariadenia sa delia na tzv. *master*, ktorý môže byť na zbernici iba 1 a tzv. *slave*, ktorých množstvo nie je protokolom obmedzené. Počet slave zariadení je limitovaný iba počtom výstupov, ktoré môže master použiť. Rozhranie pracuje na CMOS a TTL komatibilných napäťových úrovniach (0V a 3V), preto je jeho zapojenie veľmi jednoduché bez potreby ďalších elektrických zariadení ako napr. pull-up rezistorov. [1, 2]

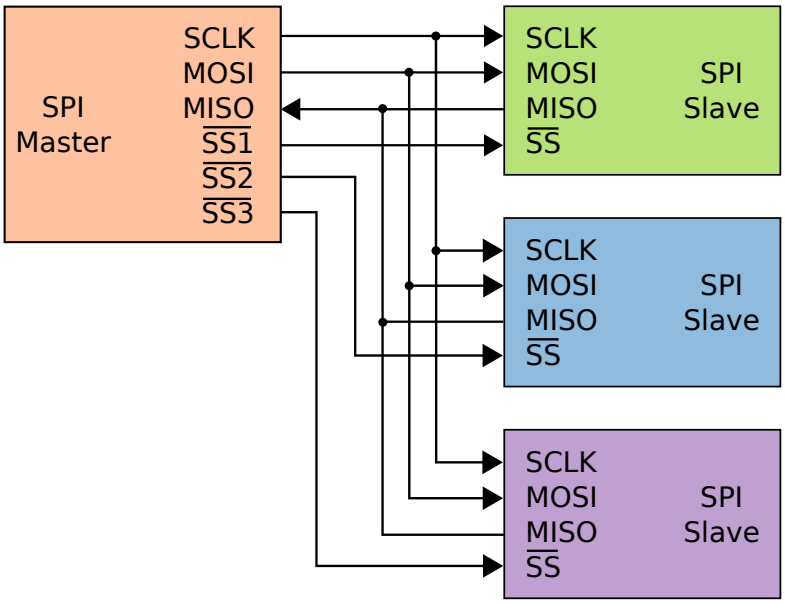
**Obr. 2.1** Prenos rozhraním SPI pri použití dvoch posuvných registrov. Prevzaté z [3]




Toto rozhranie používa na prenos 4 signály. V prvom rade to sú datové linky `MOSI` (data odosielané z master do slave) a `MISO` (data odosielané zo slave do master). Odosielanie dát na týchto linkách prebieha v plne duplexnom režime, teda simultánne sa odosiela datový bit v smere z master do slave zariadenia a naopak. Bežne sa tento princíp znázorňuje ako implementácia pomocou dvoch posuvných registrov (jeden v master a jeden v slave zariadení), ako na obr. 2.1. [4]

Zvyšné 2 signály sú riadiace. V prvom rade to je signál SCLK, ktorý slúži na synchronizáciu prenosu dát. Tento signál generuje master zariadenie. Nakoniec je to signál $\overline{\text{SS}}$. Ten slúži na výber konkrétneho slave zariadenia, ktorý má prijímať, resp. odosielať dáta cez toho rozhranie. Pre každé slave zariadenie je potrebné použiť dedikovaný signál, čo znamená, že každé ďalšie periférne zariadenie zaberie ďalší výstup master zariadenia. Vizualizácia napojenia viacerých slave zariadení na jednu zbernicu SPI je na obr. 2.2. Fakt, že master zariadenie generuje hodinový signál a určuje, ktoré slave zariadenie má po zbernici komunikovať znamená, že všetka komunikácia medzi zariadeniami je iniciovaná iba master zariadením. Ak potrebuje slave zariadenie niečo odoslať master zariadeniu, musí počkať pokiaľ master zahájí komunikáciu. [5]

**Obr. 2.2** Zbernica SPI: jedno master a tri slave zariadenia. Prevzaté z [6]




Prenos sa začína aktiváciou signálu $\overline{\text{SS}}$ a končí jeho deaktiváciou, pričom bežne sa prenáša iba jedno slovo, teda vymenia sa obsahy registrov. Niektoré periférie, ako napríklad pamäte, ale využívajú tzv. viac-slovný prenos. V takomto prípade musí zostať signál $\overline{\text{SS}}$ aktivovaný po celú dobu transakcie, teda prenosu viacerých slov. Napríklad pamäť môže očakávať vrámci jednej transakcie niekoľko bajtov adresy a potom niekoľko bajtov dát. Ak by teda došlo k deaktivácii tohto signálu, porušila by sa štruktúra transakcie. [1]

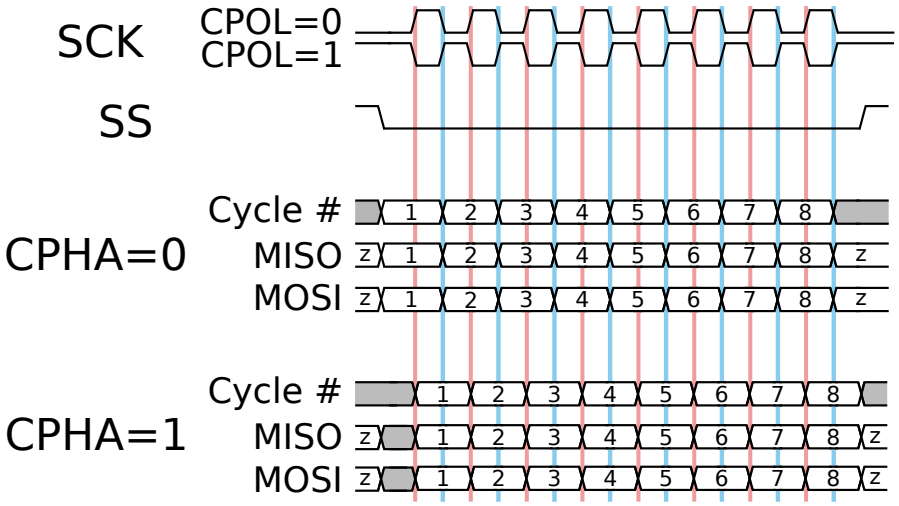
### 2.1.1 Prenosové módy rozhrania SPI

Prenosy SPI môžu byť realizované až v 4 rôznych módoch podľa nastavenej hodinovej polarity a fázy. V zariadeniach, ktoré dovoľujú všetky 4 prenosové módy (hlavne master) sa zvyknú nastavovať 2 konfiguračnými bitmi nazvanými CPOL (Clock POLarity) a CPHA (Clock PHAse) podľa konvencie, ktorú zaviedla Motorola. Pre správny prenos musia mať master a slave zariadenia nastavený rovnaký prenosový mód, preto jeho voľba vždy závisí od ich možností. [7]

CPOL definuje v akej logickej úrovni sa nachádza signál `SCLK` keď je nečinný. Ak je CPOL = 0, potom je `SCLK` v nečinnosti v log. 0. V opačnom prípade keď je CPOL = 1, `SCLK` je v nečinnosti v log. 1. [1]

CPHA definuje okamžik vystavenia a vzorkovania dát. Ak je CPHA = 0, vystavenie dát prebehne pri prechode z aktívnej do kľudovej úrovne signálu `SCLK` a ich vzorkovanie pri prechode z kľudovej do aktívnej úrovne signálu. Pri nastavení CPHA = 1 je to opačne. Priebeh transakcie pri nastavení rôznych prenosových módov je nznázornený na obr. 2.3. [1]

**Obr. 2.3** Časový diagram priebeh prenosu rozhraním SPI v závislosti od zvoleného módu prenosu. Prevzaté z [8]




## 2.2 Vybrané koncepty jazyka VHDL

Jazyk VHDL je tzv. *hardware description language*, slúži teda pre návrh hardvéru, nejedná sa o jazyk programovací. Zdrojové kódy jazyka VHDL sa spracujú tzv. syntézou pomocou syntetizátora, ktorý ich prevedie na samotný obvod. Jazyk VHDL vie opísať hardvér na troch základných logických úrovniach. Podľa zvolenej úrovne ide o popis:

**Behaviorálny** - ktorý špecifikuje funkciu obvodu bez toho aby špecifikoval akékoľvek detaily ohľadom implementácie.

**Data flow** - ktorý popisuje obvod tak, že definuje jeho datové závislosti.

**Štrukturálny** - ktorý špecifikuje logické hradlá a ich prepojenie v obvode, teda popisuje jeho fyzickú implementáciu. [9]

Jeden zdrojový súbor jazyka VHDL zvyčajne definuje jeden modul návrhu. Takýto modul pozostáva z popisu *entity* a popisu *architektúry*. Popis entity deklaruje vstupné a výstupné signály, inak povedané, opisuje rozhranie uzavretej časti návrhu. Naopak popis architektúry určuje vnútorné chovanie modulu, teda napr. ako sa jeho výstupy zachovajú pri zmene vstupov. [9]

Pri opise architektúry je možné použiť *priame priradenie* alebo *procesy*. Priame priradenie do signálu alebo výstupu sa vyhodnocuje vždy keď dôjde k zmene signálu na ktorom závisí, preto je vhodné skôr pre kombinačnú logiku. Pre sekvenčnú logiku, ktorá reaguje na zmeny striktne synchrónne s hodinovým signálom, sa používajú procesy. Vnútri procesu sa nachádzajú príkazy, ktoré sa vykonávajú paralelne. Odlišnosť procesu oproti priamemu priradeniu je (okrem iného) v tom, že je možné určiť na zmeny *ktorých* signálov bude reagovať. Toto sa určuje pomocou zoznamu signálov, na ktoré má proces reagovať, v literatúre bežne označovaný ako *sensitivity list*. [9]

Príklad pre pochopenie tohto rozdielu je vo výpise kódu 2.1. Zatiaľ čo priame priradenie na riadku 7 bude reagovať na oba vstupy, proces aktualizuje výstup iba pri zmene vstupu A. Je to kvôli tomu, že na jeho zozname senzitivity, ktorý je na riadku 8 v zátvorkách, je iba vstup A.

```
1  entity MY_AND is
2     port(A,B : in  std_logic;
3          O   : out std_logic);
4  end MY_AND;
5  architecture Behavioral of MY_AND is
6  begin
7     O <= A and B;
8     process(A)
9     begin
10       O <= A and B;
11    end process;
12 end Behavioral;
```

**Výpis kódu 2.1** Príklad rozdielu priamého priradenia a procesu v jazyku VHDL.

V tomto jazyku sa pre návrh používa hierarchický dizajn čo umožňuje vytvoriť komplexné systémy z jednoduchších prepojených celkov. Pre tento účel sa používajú *komponenty*. Komponenta je rezprezentáciou entity, ktorá je použitá vnútri inej entity. [10]

### 2.2.1 Konečný automat

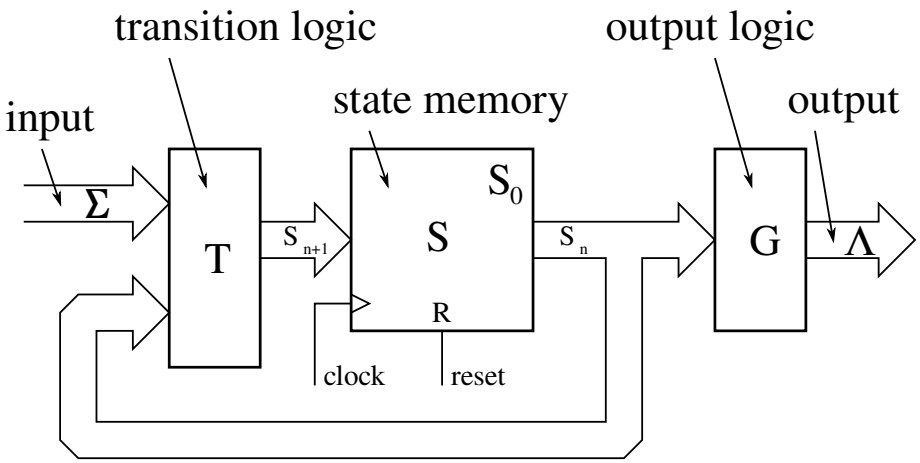
Pre riadenie obvodov, ktoré majú vykonávať jednotlivé kroky v určitej sekvencii sa zvyčajne používa *konečný automat* (často označovaný aj skratkou FSM). Vo všeobecnosti sa skladá z množiny *vstupov*, *výstupov*, *stavov*, *prechodovej* a *výstupnej funkcie*. Množina stavov je abstraktná a používa sa pre znázornenie jednotlivých krokov automatu. Automat je vždy v nejakom stave z tejto množiny a pri nábežnej hrane hodín prejde do nového stavu podľa prechodovej funkcie. Prechody závisia od aktuálneho stavu a hodnoty vstupov. [9, 11]

Stavové automaty sa často delia na 2 základné typy a to *Mealy* a *Moore*. V automate typu Mealy závisia výstupy na aktuálnom stave a hodnotách vstupov, zatiaľ čo v automate typu Moore závisia len na aktuálnom stave. To znamená, že na rozdiel od automatu typu Mealy sa v automate typu Moore výstupy menia vždy iba pri nábežnej

hrane hodín spolu so stavom. Teoreticky je možné každý automat typu Mealy previesť na automat typu Moore a naopak. V praxi sa používajú obe, pričom automat typu Mealy oproti Moore zvyčajne ponúka menej stavov, ale dodržať časové požiadavky môže byť náročnejšie kvôli väčšiemu oneskoreniu vstupov a výstupov. [11]

Ďalej popíšem len realizáciu automatu typu Moore nakoľko v tejto práci používam iba tento typ. Z pohľadu HW sa konečný automat skladá z dvoch kombinačných logík, výstupnej a prechodovej a z registra stavov. Takéto zapojenie je znázornené na obr. 2.4. Pre takýto návrh v jazyku VHDL je možné použiť jeho základné funkcie tak, že jednotlivé bloky sa modelujú pomocou samostatných procesov. Pre prehľadnú reprezentáciu stavov je možné si definovať vlastný typ pomocou príkazu `type STATE is (s1,s2)` a následne vytvoriť 2 signály tohto typu, ktoré budú reprezentovať súčasný a následovný stav. [11]

**Obr. 2.4** Bežná reprezentácia automatu typu Moore v HW. Prevzaté z [12]



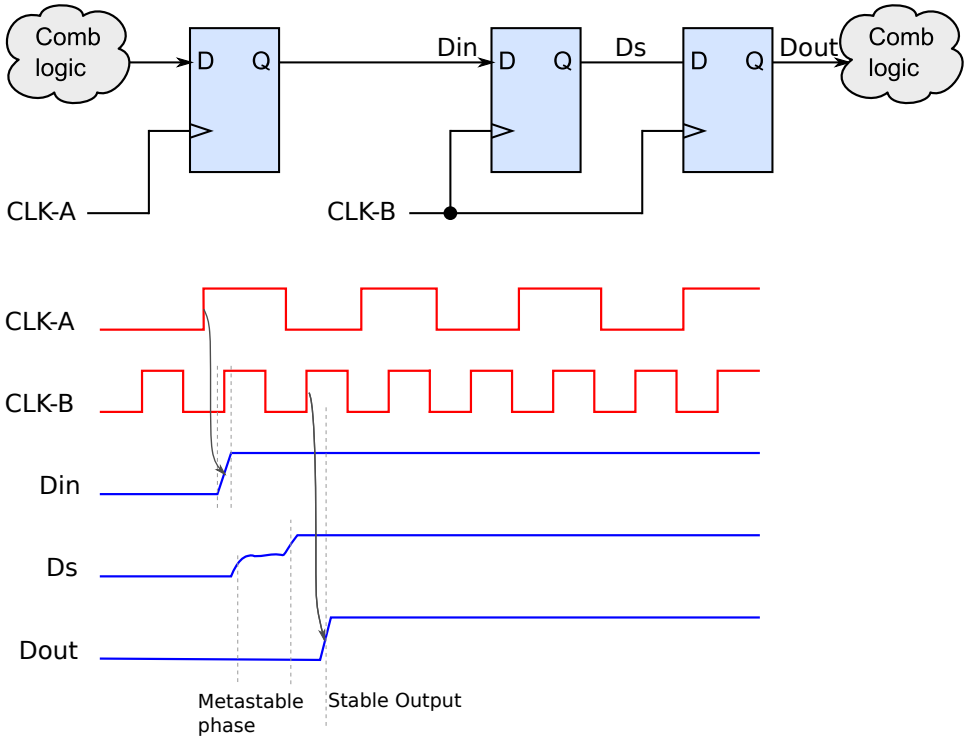

### 2.2.2 Metastabilita a asynchrónne vstupy

V každom HW dizajne, ktorý obsahuje preklápacie obvody je potrebné dodržať aby bol vstup preklápacieho obvodu stabilný po určitú dobu pred nábežnou hranou, inak sa môže dostať do tzv. *metastabilného stavu*. Jedná sa o stav v ktorom jeho výstup nie je definovaný a nie je definovaný ani doba po ktorú bude tento stav trvať. Z toho dôvodu potom hrozí, že nebudú dodržané časové požiadavky obvodu, ktorý je napojený na takýto výstup. [11]

V rámci vnútorných signálov synchrónneho obvodu (asynchrónnym návrhom sa táto práca zaoberať nebude) je jednoduché zabezpečiť, že táto požiadavka predstihu bude dodržaná, ale pre signály, ktoré sú generované externe a vstupujú do obvodu napr. zo systému s iným hodinovým signálom je nemožné garantovať jej splnenie. Takéto signály sa nazývajú *asynchrónne vstupy* a ak by boli napojené priamo na nejakú časť obvodu, mohli by nastať problémy s metastabilitou. Bežným postupom pre ošetrenie týchto vstupov je napojiť ich na tzv. *synchronizátor* hneď pri vstupe do obvodu a potom použiť jeho výstup vo zvyšku obvodu. V dnešnej dobe je v bežných aplikáciách postačujúci

a najčastejšie používaný synchronizátor vo forme 2 preklápacích obvodov (ďalej len D-F/F), znázornený na obr. 2.5. Ak nastane situácia, že prvý z nich vstúpi do metastabilného stavu, má čas 1 hodinovej periódy aby sa jeho výstup stal stabilný. [11]

**Obr. 2.5** Metastabilita a jej ošetrenie pomocou D-F/F. Prevzaté z [13]




### 2.2.3 Možnosti pre návrh pamäte

Niektoré aplikácie vyžadujú pre svoje fungovanie ukladať veľké množstvo dát, pre ktoré by občajné registry nestačily, preto je potrebné uvažovať nad tým, ako v danom návrhu vytvoriť pamäť. Pri vytváraní návrhu pre čipy FPGA je možné vytvoriť pamäť z tzv. *Look-Up Tables* (ďalej len LUT). Tie slúžia ako základ pre implementáciu navrhnutého obvodu, kde kombinácia 1 LUT spolu s 1 preklápacím obvodom je považovaná za základný prvok. Takáto pamäť je rýchla, môže byť asynchrónna alebo synchrónna, ale zaberá „priestor“ pre zvyšok návrhu a v krajnom prípade by mohla byť pamäť príliš veľká a počet LUT by nestačil. Preto sa do čipov FPGA pridávajú dedikované pamäte, ktoré disponujú veľkosťou 16Kb až 10Mb. Ide o pamäť typu RAM a je organizovaná v blokoch (preto sa v FPGA spoločnosti od Xilinx nazýva BRAM) na okraji hradlového poľa. Aj keď sa jedná o bloky pamäte, je možné nastaviť jej šírku, čo je pomerne dôležitý aspekt pri návrhu HW. Aby syntetizátor VHDL kódu vedel identifikovať, či je žiadúce danú pamäť vytvoriť pomocou LUT alebo BRAM, je potrebné jej VHDL kód špecificky napísať. Ľahšou alternatívou je popis pamäte vygenerovať z dedikovaného nástroja, v prípade FPGA od spoločnosti Xilinx ide o *Block Memory Generator*. [9]

## 2.3 Základné charakteristiky LCD

Displej s kvapalnými kryštálmi (ďalej len LCD) je jeden z typov zobrazovacích prostriedkov. Tieto displeje buď pomocou podsvietenia, alebo odrazom prirodzeného svetla vysielajú svetlo do očí užívateľa, pričom princíp zobrazovania informácie je založený na tom, že sa v určitých oblastiach displeja prenos svetla zablokuje, čím vznikne obraz. Pre zablokovanie svetla využívajú efekt polarizácie svetla. Nepolarizované svetlo vychádzajúce z podsvietenia sa polarizuje čo umožní jeho zablokovanie pomocou *kvapalných kryštálov*. Ide o medzifázu tuhých kryštálov a izotropnej tekutiny. Zložky sú organické molekuly v tvare tyčí alebo diskov, ktoré je možné orientovať elektrickým napätím, čím sa zabezpečí zmena polarizácie prechádzajúceho svetla. V kľudovom stave sa polarizácia nezmení a svetlo neprejde druhým polarizátorom čo sa na LCD javí ako tmavé miesto. Pri aplikovaní určitého napätia, pričom na jeho polarite nezáleží, sa polarizácia prechádzajúceho svetla zmení tak aby prešlo druhým polarizátorom a na LCD sa zobrazí ako svetlé miesto. [14]

Kvôli drobným nedokonalostiam LCD segmentov dochádza pri dlhodobom aplikovaní jednosmerného napätia k ukladaniu iontových nečistôt na elektródy, čím môže dôjsť až k takému poškodeniu segmentu, že nebude reagovať na elektrické napätie a bude trvalo tmavý. Využitím faktu, že nezáleží na polarite aplikovaného napätia sa tomu dá predísť tak, že sa na LCD segmenty striedavo aplikuje rovnaké napätie s rozdielnou polaritou. Priemerné napätie na segmente je tak z dlhodobého pohľadu nulové. [15]

Jednotlivé kryštály je možné na displeji usporiadať rôzne, podľa toho, čo má zobrazovať. Veľmi známe sú napríklad 7-segmentové displeje, pomocou ktorých je možné zobrazovať napr. číslice, ale ich vužitie je limitované tvarom segmentov. Pre všeobecné použitie zobrazovania rôznych obrázkov sa používajú maticové displeje. V tých sú jednotlivé segmenty sformované do malých štvorcov, tzv. *pixelov*, ktoré sú usporiadané do matice. [14]

Podľa [16] sa maticové LCD delia ešte do 2 základných skupín:

- **Pasívne**
  V PMLCD sa napätie pripojuje priamo na elektródy jednotlivých segmentov. Vzhľadom k povahe maticového adresovania (opísane v ďalšej kapitole) to spôsobuje tzv. *cross-talk*, kde ovládanie jedného pixelu ovplyňuje aj iné pixely. Zatiaľ čo tento spôsob je postačujúci pre jednoduché displeje (zväčša do veľkosti 480x128 pixelov), pre monitory PC a televízory je nedostačujúci.

- **Aktívne**
  V AMLCD sa rieši tento problém rozdelením adresovania pixelov a zápisu dát, ktoré prebiehajú samostatne. Toto bolo možné dosiahnúť pomocou pripojenia tranzistora na elektródy pixela. Ten slúži okrem iného aj na udržanie napätia pixela aj keď už nie je priamo adresovaný. AMLCD sa ďalej nebudem zaoberať nakoľko LCD s ktorým pracujem v rámci tejto práci je pasívny.

### 2.3.1 Adresovanie PMLCD

Segmentové displeje sú bežne adresované tak, že všetky segmenty majú 1 spoločnú elektródu a druhú majú samostatnú. Riadiaci obvod tak môže ovládať každý segment jednotlivo a nezávisle na ostatných. Tento spôsob sa nazýva *priame adresovanie*. [17]

V maticových displejoch s veľkým počtom pixelov je ale tento systém nepraktický, pretože by bol počet výstupov neúmerne veľký. Preto jednotlivé pixely spoločne zdieľajú elektródy tak, že jedna je napojená na spoločnú elektródu vrámci riadku matice a druhá na spoločnú elektródu vrámci jedného stĺpca. Takýmto spôsobom má potom displej o M riadkoch a N stĺpcoch M+N výstupov, čo je znateľne menej ako v predchádzajúcom prípade. Pri použití spoločných elektród nie je možné zobrazovať rôzne informácie na všetkých pixeloch naraz a preto je potrebný algoritmus, ktorý ich bude adresovať postupne. [15]

Hoci by sa mohlo zdať, že základným algoritmom pre adresovanie PMLCD bude postupné adresovanie jednotlivých pixelov, obdobne ako tomu bolo v prípade starších CRT displejov, nie je to tak. Tento spôsob by nefungoval, pretože jednotlivé pixely v LCD potrebujú na aktiváciu oveľa viac času ako v prípade CRT, preto by bola obnovovacia frekvencia celého displeja veľmi nízka. [15]

V PMLCD sa teda používa tzv. *adresovanie po riadkoch*, ktoré ponúka oveľa efektívnejší spôsob adresovania, hoci za cenu zložitejšej riadiacej logiky. Funguje na princípe, že sa na displeji postupne zobrazujú jednotlivé riadky. Základný princíp sa dá predstaviť tak, že na stĺpcové elektródy svetlých pixelov sa napojí napätie, zatiaľ čo stĺpcové elektródy tmavých pixelov sa uzemnia. Zároveň sa uzemní iba 1 elektróda požadovaného riadku, pričom takto sa postupne vystriedajú všetky riadky. Aby nedošlo k tzv. *crosstalk* javu, elektródy neadresovaných riadkov sa napoja na špecifické napätie tak, aby celkové napätie pixelu bolo blízke prahovému napätiu pre použité kvapalné kryštály. [15]

# Kapitola 3
# Analýza existujúcich riešení

*V tejto kapitole analyzujem existujúce riešenia riadenia LCD v elektronickom písacom stroji typu Trimuph-Adler Gabriele PFS. Najprv uvediem jednotlivé riešenia tohto problému, ktoré sú pre mňa dostupné. Potom podrobnejšie analyzujem riešenie použité v tomto stroji, teda radič typu MSM6255. Ten analyzujem jednak pomocou dostupnej dokumentácie, ale aj pomocou merania priebehu jednotlivých signálov na rozhraní LCD panela.*

LCD panel, pre ktorý budem implementovať nový radič, je súčasťou písacieho stroja typu Triumph-Adler Gabriele PFS, ktorý je funkčný. Jedným z možných riešení ovládania tohto LCD je teda jeho originálny radič. Po odstránení jednotlivých krytov stroja som sa dostal k jeho základnej doske, na ktorej som lokalizoval radič LCD typu MSM6255.

Na základe tohto zistenia som potom zisťoval, či existujú nejaké riešenia riadenia LCD, ktorý bol originálne riadený radičom typu MSM6255. Jediné riešenie, ktoré som našiel, bol blog [18] na ktorom bola opísaná výmena starej základnej dosky elektronického písacieho stroja za mikropočítač s vlastnými radičmi. Hoci ide o iný typ písacieho stroja, rovnako disponuje LCD o veľkosti 480x128 pixelov, ktorý je riadený radičom typu MSM6255. Na tomto blogu je síce zmienená výmena tohto radiča, bohužiaľ ale nie je nijako opísaný spôsob riadenia LCD, teda funkcie a priebehy jednotlivých riadiacich signálov.

Zistil som ale, že aktuálne zdrojové kódy tohto projektu sú verejne dostupné na stránke [19]. Urobil som teda základnú analýzu časti zdrojového kódu, ktorý sa týkal práve riešenia náhrady radiča MSM6255. Po prečítaní časti blogu o výmene radiča pre LCD a zbežnom preskúmaní zdrojového kódu som prišiel k záveru, že toto riešenie pre mňa nie je vhodné z týchto dôvodov:

1. Autor použil pre implementáciu FPGA čip od spoločnosti Lattice. Tento čip použil navyše na špeciálnej platforme, ktorú si sám vytvoril a ktorá teda nie je komerčne dostupná. Bolo pre mňa teda jednoduchšie vyvinúť radič na platformách od spoločnosti Xilinx používaných mojou fakultou.

2. Autor navrhol HW v jazyku VHDL pomocou veľmi zložitých konceptov, ktorým som nebol schopný plne porozumieť a pochopiť ako presne sú prepojené. Z toho dôvodu som nemohol použiť ani jeho zdrojový kód, či prípadne jeho časti.

Nakoľko disponujem funkčným kusom radiča MSM6255, ktorý je integrovaný do základnej dosky písacieho stroja, môžem okrem analýzy dokumentácie overiť priebehy signálov aj meraním. Preto som sa rozhodol radšej sám analyzovať pôvodne riešenie a na základe toho vytvoriť vlastný radič LCD.

## 3.1 Analýza rozhrania LCD panela

V tejto podkapitole budem analyzovať ovládanie LCD tak ako bolo originálne vyriešené v tomto stroji. Najprv analyzujem dokumentáciu jednotlivých integrovaných obvodov, ktoré sa podieľajú na riadení LCD. Potom, nakoľko je stroj funkčný, využijem aj možnosť analýzy priebehu signálov na rozhraní LCD panela pomocou merania osciloskopom.

Radič, ktorý budem nahradzovať, je LCD radič typu MSM6255. Musel som teda analyzovať jeho výstupy, najmä riadiace signály, ktoré vysiela na rozhranie LCD panela. Analýzou jeho dokumentácie je možné získať nejaké informácie, ale je ich príliš málo pretože dokumentácia sa zaoberá hlavne jeho vstupmi, teda spôsobom ako s ním má komunikovať procesor.

Tento radič nie je napojený priamo na elektródy LCD aby pomocou aplikovania napätia na elektródy zobrazoval obraz. K tomuto účelu slúžia ďalšie súčiastky, v prípade výrobcu OKI Semiconductor ide o tzv. segmentové a spoločné radiče. Výstupy LCD radiča MSM6255 potom slúžia ako vstupné riadiace a datové signály do týchto radičov.

Pre podrobnú analýzu som teda potreboval naštudovať aj dokumentáciu k týmto radičom. Problémom bolo, že som nevedel presne o aký typ radičov ide, pretože tieto radiče sú priamo v LCD paneli. Nakoľko je LCD panel vytvorený z plastu, ktorý má dnes už viac ako 30 rokov hrozilo, že by sa pri snahe o prístup k týmto radičom nenávratne poškodili. Preto nemôžem s istotou vedieť aký typ spoločných a segmentových radičov je použitý, urobil som preto kvalifikovaný odhad na základe dostupných informácii:

1. Problém náhrady radiča MSM6255 už na svojom blogu [18] riešil Markus Koch, ktorý úspešne kryt LCD panela otvoril a fotograficky zdokumentoval, že sú v ňom použité spoločné radiče typu MSM5298A a segmentové radiče typu MSM5299A. Jedná sa síce o iného výrobcu písacieho stroja, ale používa LCD s rovnakou veľkosťou 480x128 pixelov a je približne rovnako starý.

2. Ostatné typy spoločných a segmentových radičov od rovnakého výrobcu by v tomto prípade nemohli byť použité. Buď by išlo o typ, ktorý nie je vôbec určený na spoluprácu s radičom typu MSM6255, alebo by išlo o také typy, ktoré sú síce pre tento LCD radič určené, ale ak by boli použité, zloženie signálov na rozhraní LCD panela by bolo odlišné.

Ďalej budem teda pre účely analýzy dokumentácie predpokladať, že v tomto LCD paneli sú použité spoločné radiče typu MSM5298A a segmentové radiče typu MSM5299A. Vzhľadom k počtu výstupov, MSM5298A má 68 a MSM5299A má 80, ich počtu na fotke dosky na spomínanom blogu [18], kde sú 2 radiče MSM5298A a 6 radičov MSM5299A a rozmeru LCD, ktorý je 480x128 pixelov, je možné vyvodiť, že spoločné radiče budú napojené na riadkové elektródy LCD a segmentové radiče na stĺpcové elektródy. Toto budem ďalej predpokladať, a z prevedenej analýzy rozdielov vo funkčnostiach týchto radičov sa to aj potvrdí.

### 3.1.1 Funkcia spoločného radiča MSM5298A

Spoločné radiče MSM5298A podľa blokového diagramu v ich dokumentácii [20] obsahujú 68-bitový obojsmerne posuvný register a obvody pre zmenu napäťovej úrovne, ktoré slúžia na napojenie špecifického napätia na elektródy LCD. Výber pripojeného napätia závisí od konkrétnych dát, teda či je pre daný výstup v registry log. 0 alebo log. 1, ale tiež od vstupu DF a vstupu $\overline{\texttt{DISP OFF}}$. Posledný vymenovaný slúži samozrejme na odpojenie LCD segmentov, na rozhraní LCD panela sa ale nenachádza. Vstup DF je dôležitý, pretože strieda napäťové úrovne napojené na elektródy. Tieto vstupy napájacieho napätia sú označené ako $V_1$, $V_2$, $V_5$ a $V_{EE}$ a ich napojenie na výstup podľa kombinácie logických úrovní dát a vstupu DF je vyznačené v tab. 3.1. Účelom striedania napätia je, aby nedošlo k vypáleniu LCD segmentov. [20]

| Dáta v registry | DF | Výstupné napätie |
|---|---|---|
| 0 | 0 | $V_2$ |
| 0 | 1 | $V_5$ |
| 1 | 0 | $V_{EE}$ |
| 1 | 1 | $V_1$ |

**Tabuľka 3.1** Napäťové výstupy radiča MSM5298A v závislostí od dát a vstupu DF

Posuvný register má vstupno-výstupné porty označené ako $IO_1$ a $IO_{68}$. Posun riadi vstup CP, pričom k posunu dôjde vždy pri zostupnej hrane. Smer posuvu závisí od vstupu SHL. Tento vstup nie je na rozhraní LCD panela, preto sa ním ďalej nebudem zaoberať a budem predpokladať, že $IO_1$ je vstupom a $IO_{68}$ výstupom. [20]

V dokumentácii sa uvádza, že je možné napojiť viacero týchto radičov za sebou v sérii. Práve k tomuto účelu disponuje aj výstupným portom. V tomto prípade je teda port $IO_{68}$ *prvého* spoločného radiča napojený na port $IO_1$ *druhého* spoločného radiča. Dáta na vstupe tak postupne prejdú cez všetkých 136 výstupov, čiže cez všetkých 128 riadkov (predpokladám, že 8 výstupov je nazvyš a nezapojených). [20]

Ďalej som z dokumentácie analyzoval časové požiadavky, ktoré sú dôležité pre túto prácu. Tie sú zhrnuté v tab. 3.2. Venoval som sa požiadavkám na vstupy $IO_1$, CP a DF. Ostatné vstupy analyzovať nebudem, nakoľko nie sú prítomné na rozhraní LCD panela. Neprítomnosť vstupu SHL nie je ani tak veľmi dôležitá, ale neprítomnosť vstupu $\overline{\texttt{DISP OFF}}$ znamená, že nebude možné LCD „vypnúť“, teda uviesť ho do stavu, kedy je jeho riadiaca elektronika napojená, ale segmenty v LCD nie sú napájané. [20]

| Popis | Symbol | Min. hodnota | Max. hodnota | Jednotky |
|---|---|---|---|---|
| Frekvencia CP | $f_{CP}$ | - | 1 | MHz |
| Dĺžka pulzu CP | $t_{W(CP)}$ | 125 | - | ns |
| Predstih dát ($IO_1$) | $t_{SETUP}$ | 100 | - | ns |
| Presah dát ($IO_1$) | $t_{HOLD}$ | 100 | - | ns |

**Tabuľka 3.2** Vybrané časové požiadavky vstupov radiča MSM5298A

Z prevedenej analýzy je možné vidieť, že tieto radiče disponujú len veľmi jednoduchými schopnosťami. Schopnosť meniť logickú úroveň je samozrejmosťou pri ovládaní LCD, preto jedinou „funkčnou“ schopnosťou je, že vedia prijať 1-bitové dáta, ktoré sa následne posúvajú cez všetky riadky LCD.

### 3.1.2 Funkcia segmentového radiča MSM5299A

Segmentový radič MSM5299A je z hľadiska obsiahnutých obvodov podobný spoločnému radiču MSM5298A, no trochu zložitejší. Rovnako ako predchádzajúci obsahuje obvody pre zmenu napäťovej úrovne, ktoré sú ovládané pomocou vstupov DF a $\overline{\texttt{DISP OFF}}$. Ich funkcia je rovnaká ako som opísal v predchádzajúcej podkapitole. Napäťové vstupy týchto radičov sa ale v dvoch prípadoch líšia. Pre tieto radiče sú vstupy označené ako $V_1$, $V_3$, $V_4$ a $V_{EE}$. Ich napojenie na výstup podľa dát a vstupu DF je v tab. 3.3. [21]

| $D_0$ až $D_3$ | DF | Výstupné napätie |
|---|---|---|
| 0 | 0 | $V_3$ |
| 0 | 1 | $V_4$ |
| 1 | 0 | $V_1$ |
| 1 | 1 | $V_{EE}$ |

**Tabuľka 3.3** Napäťové výstupy radiča MSM5299A v závislosti od dát a vstupu DF

Ďalej obsahujú, podobne ako v predchádzajúcom prípade, posuvný register pre dáta, ale so značnými odlišnosťami. Nejde o jeden, ale o štyri 20-bitové posuvné registry, keďže dáta sú v nich prijímané po 4 bitoch naraz. Narozdiel od predchádzajúceho disponuje *iba vstupnými* portmi pre dáta. Posun je ovládaný vstupom CP a prebehne pri zostupnej hrane. Pomocou vstupu SHL je tiež možné ovládať smer posunu dát, avšak tento vstup nie je vyvedený na rozhranie LCD panela. [21]

Tak ako v predchádzajúcom prípade je možné zapojiť viacero radičov za sebou. V tomto prípade to dokumentácia označuje ako *kaskádne* zapojenie a vykonáva sa rozdielne. Nakoľko radiče nemajú datový výstup, sú všetky napojené na spoločnú datovú zbernicu a obsahujú radič, ktorý riadi či majú dáta na zbernici prijímať alebo nie. Na ovládanie tejto funcie slúžia vstupno-výstupné porty $\overline{\texttt{EL}}$ a $\overline{\texttt{ER}}$. Pri použití kaskádneho zapojenia a vstupu SHL v log. 0 je port $\overline{\texttt{ER}}$ vstupom a port $\overline{\texttt{EL}}$ výstupom. V takom prípade je potrebné vstup $\overline{\texttt{ER}}$ *prvého* radiča uzemniť a prepojiť jeho výstup $\overline{\texttt{EL}}$ na vstup $\overline{\texttt{ER}}$ *druhého* radiča a takto postupovať ďalej. Obrazne povedané si tak radiče budú medzi sebou podávať „štafetový kolík“ a dáta zo zbernice ukladať postupne. [21]

Nakoniec disponuje tento radič ešte D-F/F, ktorý sa nachádza medzi posuvnými registrami a obvodmi pre zmenu napäťovej úrovne. Dáta v posuvných registroch tak priamo neovplyvňujú výstupné napätie privedené na elektródy LCD ako tomu bolo v predchádzajúcom prípade, ale je potrebné ich najprv uložiť. To sa ovláda pomocou vstupu LOAD, pričom k uloženiu dôjde pri jeho zostupnej hrane. [21]

Z dokumentácie som preskúmal aj časové požiadavky na vybrané vstupy, ktoré sú zhrnuté v tab. 3.4. Tak ako v predchádzajúcom prípade zaujímal som sa iba o vstupy vyvedené na rozhranie LCD panela. Konkrétne ide o $D_0$ až $D_3$, CP, DF a LOAD. [21]

| Popis | Symbol | Min. hodnota | Max. hodnota | Jednotky |
|---|---|---|---|---|
| Frekvencia CP | $f_{CP}$ | - | 3.4 | MHz |
| Dĺžka pulzu CP a LOAD | $t_W$ | 100 | - | ns |
| Predstih dát ($D_0$ až $D_3$) | $t_{DSU}$ | 50 | - | ns |
| Presah dát ($D_0$ až $D_3$) | $t_{DHD}$ | 80 | - | ns |
| Predstih CP pred LOAD | $t_{LSU}$ | 90 | - | ns |
| Presah CP po LOAD | $t_{LC}$ | 200 | - | ns |

**Tabuľka 3.4** Vybrané časové požiadavky vstupov radiča MSM5299A

Z analýzy je možné vyvodiť, že segmentové radiče disponujú väčšou funkcionalitou ako tie spoločné, ktoré dokázali v podstate iba posúvať dáta. Tieto radiče disponujú lepším viac-násobným zapojením pomocou vnútorného radiča, ktorý bol zvolený pravdepodobne z toho dôvodu, aby mali radiče nižšiu spotrebu energie. Každopádne toto zapojenie nie je možné ovplyvňovať na rozhraní LCD panela, súbor 6 radičov sa pri práci s nimi teda javí ako jeden veľký segmentový radič so 480 výstupmi, ktorý pokrýva všetky stĺpce LCD. Takto nad tým budem aj ďalej uvažovať.

Oveľa zaujímavejším faktom je, že dokážu dáta uložiť takým spôsobom, že ich postupné posúvanie v posuvnom registry neovplyvňuje výstup na LCD. Nakoľko s vysokou pravdepodobnosťou signál na vstupe `LOAD` zdieľajú, dôjde k ich preklopeniu naraz vo všetkých segmentových radičoch.

### 3.1.3 Funkcia LCD radiča MSM6255

Radič typu MSM6255 budem v tejto práci nahradzovať, analyzovať budem teda iba jeho výstupy. Tie sú vyvedené na rozhranie LCD panela a teda slúžia ako vstupy pre vyššie popísané spoločné a segmentové radiče. Bohužiaľ sa v dokumnetácii neuvádza presné napojenie jeho výstupov a rovnako ani nijak podrobne nepribližuje ich funkcia. V tejto podkapitole teda zhrniem dostupné informácie o týchto signáloch z jeho dokumentácie a v následujúcej podkapitole využijem informácie zo všetkých troch dokumentácii pre skonkretizovanie ich prepojenia a funkcie. [22]

Tento radič má až 2 datové výstupy, ktoré môžu mať rôznu šírku. Ide o výstupy $\texttt{UD}_0$ až $\texttt{UD}_3$ a $\texttt{LD}_0$ až $\texttt{LD}_3$. Šírka `UD` a `LD` môže byť 1, 2 alebo 4 bity, dokopy teda môže radič doosielať až 8 bitov dát naraz. Pre LCD panel s ktorým pracujem v tejto práci sú dáta odosielané po 4-bitovej zbernici a ďalej budem pracovať len s týmto nastavením. Okrem týchto dátových výstupov disponuje tento radič aj piatimi *riadiacimi* výstupmi. Podľa [22] sú ich funkcie následujúce:

**CLP** - hodinový signál pre posun dát v LCD

**LIP** - signál pre preklopenie dát v LCD

**FRP** - snímkový signál, synchronizácia LCD

**FRMB** - výstup symetrického periodického signálu

**$CE_\phi$** - hodinový signál pre obvod aktivácie jednotlivých segmentových radičov. Systém aktivácie jednotlivých segmentových radičov vyžadoval v niektorých typoch týchto radičov aj samostatný hodinový signál. Tento signál nie je prítomný na rozhraní LCD s ktorým pracujem.

Bližší popis k týmto signálom dokumentácia k radiču MSM6255 neposkytuje. Je v nej ale znázornených niekoľko časových diagramov, ktoré ukazujú niektoré časti priebehu týchto signálov. Tieto diagramy som teda analyzoval a zistil niekoľko dodatočných informácií.

V prvom rade som analyzoval odosielanie dát. Tie sa odosielajú na LCD synchrónne so signálom `CLP`, pričom k zmene dochádza vždy pri jeho vzostupnej hrane. [22]

Ďalej som analyzoval súvislosti medzi signálmi `CLP` a `LIP`. Signál `CLP` je aktívny pri odosielaní dát na LCD. Keď sa odošlú všetky dáta, dôjde k jeho deaktivácii, po určitej dobe nastane pulz signálu `LIP` a po nejakom čase je `CLP` znovu aktívny a odosielajú sa dáta. Takto sa ich priebeh vždy opakuje. [22]

Ďalej som k tomu pridal signál `FRP`. Z jedného diagramu je jasne vidieť, že sa aktivuje pri začiatku odosielania dát prvého riadku na LCD, zostane aktivovaný aj po dobu deaktivácie `CLP` po odoslaní celého riadku a až po ukončení pulzu `LIP` sa s určitým časovým odstupom deaktivuje. [22]

Na rovnakom diagrame je tiež znázornené ako sa signál `FRMB` aktivuje presne s deaktiváciou signálu `LIP` medzi odosielaním dát prvého a druhého riadku. Toto je jediná informácia k signálu `FRMB` a iba z dokumentácie by bolo ťažké vyvodiť jeho priebeh. Našťastie disponujem funkčným prístrojom preto som priebehy signálov mohol zmerať osciloskopom. Až meranie mi dovolilo plne nahliadnúť na jeho celý priebeh a pochopiť tak jeho funkciu. [22]

### 3.1.4 Analýza fyzického priebehu signálov

V tejto časti opíšem analýzu priebehu signálov meraním pomocou osciliskopu. Samotný LCD panel je pripojený k základnej doske 3 káblami, z ktorých 2 sú jednožilové a slúžia na napájanie podsvietenia, preto sa nimi ďalej nebudem zaoberať. Zaoberať sa budem iba zostávajúcim káblom, ktorým je plochý 16-žilový kábel s rasterom 1.25mm. Tento kábel sa dá považovať za rozhranie LCD panela s ktorým pracujem, nakoľko obsahuje všetky riadiace a datové linky.

Jednotlivé linky v konektore pre tento kábel, ktorý je umiestnený na základnej doske nie sú označené, preto som pri meraní musel postupovať metodicky a najprv signály patrične roztriediť. Pre meranie osciloskopom som do konektora zapojil vlastný kábel a jednotlivé linky vyviedol na nepájivom poli. Sondu som uzemnil príslušným výstupom základnej dosky a po zapnutí stroja som premeral samostatne všetky linky. Identifikoval som 8 liniek s nekonštantným napätím, ktoré oscilovalo medzi 0V a 5V, jedná sa teda o riadiace a datové signály. Ďalej je prítomná linka ktorá má konštantné napätie 0V, ide teda o zem pre elektronické komponenty v LCD paneli, označím ju ako $V_{SS}$. K nej prislúcha linka ktorú označím ako $V_{DD}$, ktorá slúži na napájanie elektronických zariadení v LCD paneli a má konštantné napätie 5V. Nakoniec zostalo 6 liniek s konštantným napätím, ktoré bolo pre každú linku rôzne, pôjde teda o napájacie linky $V_1$ až $V_{EE}$ pre napájanie elektród LCD.

V písacom stroji je možné programovo nastaviť kontrast displeja. Ako sa dalo očakávať, pri zmene kontrastu sa mení aj napätie liniek pre napájanie LCD okrem jednej, ktorá zostáva konštantná. Ich napätie v závislosti od kontrastu uvádzam v tab. 3.5. Ako je možné vidieť, používa sa až 6 rôznych napätí. Nakoľko vytvoriť zdroj pre takéto napájanie nebolo v mojich časových možnostiach, rozhodol som sa, že pre účely proto-

typu využijem napájanie z pôvodného písacieho stroja. Z tohto dôvodu a tiež preto, že táto práca nemá za cieľ skúmať napájanie segmentov LCD som tieto napájacie linky nekategorizoval ani ďalej neskúmal.

| | Úroveň kontrastu od najväčšej (8) po najmenšiu (1) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| Napájacie linky LCD | 5.09 | 5.09 | 5.09 | 5.09 | 5.09 | 5.09 | 5.09 | 5.09 |
| | 3.56 | 3.55 | 3.53 | 3.51 | 3.50 | 3.49 | 3.47 | 3.46 |
| | 2.03 | 2.00 | 1.97 | 1.94 | 1.91 | 1.88 | 1.85 | 1.82 |
| | -10.67 | -10.80 | -10.98 | -11.12 | -11.26 | -11.40 | -11.57 | -11.72 |
| | -12.20 | -12.35 | -12.54 | -12.70 | -12.84 | -13.00 | -13.19 | -13.35 |
| | -13.73 | -13.90 | -14.10 | -14.27 | -14.44 | -14.60 | -14.80 | -14.98 |
| | Napätie[V] podľa úrovne kontrastu | | | | | | | |

**Tabuľka 3.5** Napäťové úrovne napájacích liniek v závislosti od zvolenej úrovne kontrastu

Potreboval som teda analyzovať zvyšných 8 signálov. Štyri z nich som mohol hneď označiť za datové, kvôli ich nepravidelnému priebehu. Datovým signálom som sa nijako zvlášť nepotreboval venovať, jedinú vec, ktorú som overil bolo, či sa naozaj menia pri vzostupnej hrane signálu `CLP`. Toto sa naozaj potvrdilo ako je možné vidieť na priloženom obr. A.4.

Zvyšné 4 merané linky mali pravidelný priebeh signálu, ide teda o riadiace signály. Analýza týchto signálov je najdôležitejšia pre túto prácu, preto som ich priebehy podrobne skúmal pomocou osciloskopu. Zatiaľ sa budem venovať len analýze ich priebehov a funkcií a v ďalšej podkapitole analyzujem aj ich presné časové charakteristiky.

V prvom rade som musel určiť podľa priebehov signálov o ktorý sa jedná. Tento krok bol pre mňa pomerne zložitý keď som signály meral samostatne. Z ich jednotlivých charakteristík bolo možné urobiť určité odhady, ktoré boli relativne presné, ale nemal som v nich žiadnu istotu. Najlepším príkladom je signál `FRMB`. Priradiť tento signál len podľa jeho priebehu nie je možné, nakoľko dokumentácia žiadne informácie k jeho priebehu neposkytuje. Poskytuje len informáciu, že sa mení pri zostupnej hrane signálu `LIP`. Nachádzal som sa teda v situácii kedy som nedokázal presvedčivo priradiť signály k jednotlivým linkám na rozhraní LCD panela a nemohol som pokročiť s návrhom.

V tomto sa mi podarilo posunúť až keď som začal merať všetky 4 signály naraz. Pri pohľade na priebeh týchto signálov na obr. A.2 som už dokázal jednotlivé signály identifikovať veľmi ľahko. Všetky 4 signály sa chovali presne ako je to znázornené na diagrame v dokumentácii k radiču MSM6255 a mohol som ich teda s istotou priradiť k jednotlivým linkám konektora. Toto priradenie som vypísal v tab. 3.6. Je potrebné mať na pamäti, že tento konektor je špecifický pre môj písací stroj a iný typ písacieho stroja, hoc aj bude disponovať radičom typu MSM6255, môže mať na rozhraní LCD panela iný konektor.

| Označenie signálu | FRP | $V_x$ | $V_{SS}$ | $V_{DD}$ | $D_0$-$D_3$ | FRMB | LIP | CLP |
|---|---|---|---|---|---|---|---|---|
| Linka rozhrania LCD | 1 | 2-7 | 8 | 9 | 10-13 | 14 | 15 | 16 |

**Tabuľka 3.6** Kategorizácia signálov k jednotlivým linkám konektora rozhrania LCD panela v elektronickom písacom stroji typu Triumph-Adler Gabriele PFS

Tým som už teda overil priebeh signálov CLP a LIP a zmenu signálu FRMB zároveň so zostupnou hranou signálu LIP. Priebeh signálu FRP som overil pri miernom zväčšení časovej základne na osciloskope. Ako je vidno na obr. 3.2, naozaj je aktívny po celú dobu odosielania dát jedného obrázku. Nakoniec som ďalším zväčšením časovej základne (obr. A.1) zistil celkový priebeh signálu FRMB, ktorý som doteraz z dokumentácie nepoznal.

### 3.1.5 Výsledky analýzy rozhrania LCD panela

Z analýzy dokumentácie spoločných a segmentových radičov som zistil aké poskytujú funkcionality, ktoré som už zhrnul v predchádzajúcich častiach. Z nich som usúdil, že tento LCD je adresovaný metódou *po riadkoch*, teda základnou metódou pre maticové LCD. Vzhľadom k jeho pomerne nízkemu rozlíšeniu a nevysokých nárokov to nie je prekvapujúce. Konkrétna realizácia tejto metódy adresovania funguje v tomto LCD ako súhra funkcionalít spoločných a segmentových radičov následovne:

- **Spoločné radiče**
  Ich úlohou je vybrať, ktorý riadok je aktuálne adresovaný. Pri adresovaní LCD metódou po riadkoch je v jeden moment adresovaný iba 1 riadok a riadky sa adresujú postupne. Preto obsahujú iba jednoduchý posuvný register, ktorého dáta sú priamo prepojené na LCD (narozdiel od segmentových, kde je D-F/F). Na začiatku odosielania obrázku na LCD sa na vstup $IO_1$ privedie log. 1, ktorá sa pri posune registra uloží a bude sa posúvať postupne cez všetky riadky.

- **Segmentové radiče**
  Ich úlohou je zobrazovať dáta na LCD. Nakoľko ide o adresovanie po riadkoch, zobrazujú dáta jedného riadku. V tomto prípade ide o 480 bitov dát. Nakoľko sa odosielajú postupne a ukládajú sa do posuvných registrov, tento posun by ovplyvnil obraz na LCD ak by boli posuvné registry napojené priamo na LCD tak ako v spoločných radičoch. Preto obsahujú D-F/F. Pomocou neho môžu na LCD zobrazovať dáta riadku *n*, zatiaľ čo do registrov postupne prijímajú dáta následujúceho riadku *n+1*. Tie, po ich kompletnom prijatí, tiež preklopia na LCD a začnú prijímať nový riadok.

Až po pochopení spôsobu adresovania LCD pomocou spoločných a segmentových radičov som mohol priradiť jednotlivým riadiacim signálom radiča MSM6255 ich „úlohy“, ktoré vykonávajú. Prepojenie výstupov MSM6255 na vstupy MSM5298 a MSM5299 je vypísané v tab. 3.7. Diagram prepojenia týchto radičov a ich napojenie na LCD som znázornil na obr. 3.1.

Signál CLP som mohol celkom priamočiaro napojiť ako vstup CP v segmentových radičoch. Slúži na postupné ukládanie dát v ich posuvných registroch.

Signál LIP je trochu komplexnejší. V prvom rade je asi zjavné, že slúži ako vstup LOAD v segmentových radičoch, teda slúži k preklopeniu dát z registrov na LCD. Tomu zodpovedá aj jeho priebeh, nakoľko sa aktivuje vždy po odoslaní dát jedného riadku. Okrem toho je zároveň napojený na vstup CP v spoločných radičoch. K tomuto záveru som dospel, pretože pri preklopení nových dát na LCD je nutné aj aktualizovať adresovaný riadok, teda urobiť posun registra v spoločných radičoch. Spoločný riadiaci vstup tak zabezpečí synchronizáciu týchto aktivít.

| Výstup MSM6255 | Vstup MSM5298A | Vstup MSM5299A |
|---|---|---|
| CLP | - | CP |
| LIP | CP | LOAD |
| FRP | $IO_1$ | - |
| FRMB | DF | DF |
| $UD_{0\text{-}3}$ | - | $D_{0\text{-}3}$ |
| $CE_\phi$ | - | - |

**Tabuľka 3.7** Prepojenie výstupov MSM6255 so vstupmi MSM5298A a MSM5299A

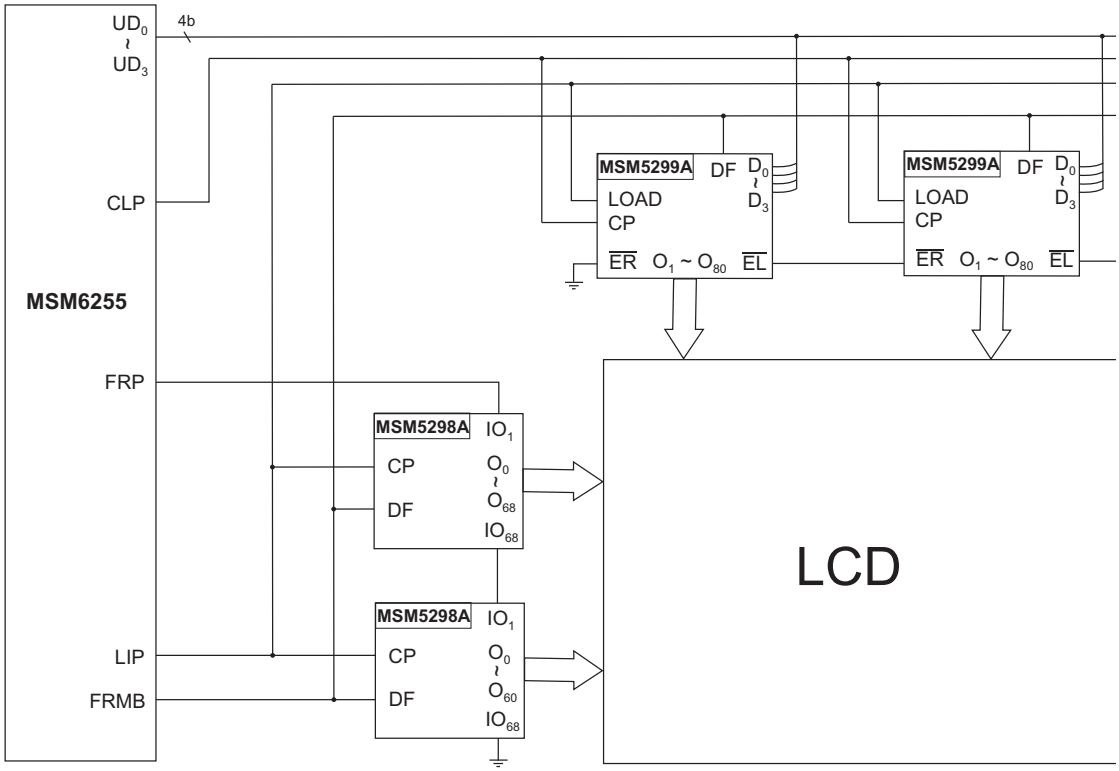
Signál `FRP` sa aktivuje len pri začiatku odosielania obrázku na LCD. Preto sa bude jednať o dátový vstup $\texttt{IO}_1$ na spoločných radičoch. Jeho aktivita po celú dobu odosielania dát podľa môjho názoru nie je nutná. Vedel by som si predstaviť, že by to bolo nutné ak by vstup $\texttt{IO}_1$ bol priamo napojený na výstup $\texttt{O}_1$, čo by znamenalo, že sa adresuje *prvý* riadok LCD. V tomto prípade to ale nie je možné, pretože je tento signál aktívny po dobu keď sa na LCD adresuje *posledný* riadok LCD a dáta prvého riadku sa ešte len odosielajú. Ďalej som to neskúmal, pretože som sa neskôr rozhodol imitovať priebehy signálov vo výsledom prototype.

Nakoniec som signál `FRMB` priradil ku vstupom `DF` spoločných a segmentových radičov. Hoci mi spočiatku robilo problém porozumieť jeho funkcii, nakoniec som pochopil, že slúži len na to, aby sa na elektródach LCD segmentov pravidelne menilo napätie, čím sa zabráni ich vyhoreniu.

Prevedená analýza bola pomerne podrobná a vyčerpávajúca, každopádne nakoľko cieľom práce je vytvoriť radič, ktorý bude schopný riadiť LCD práve cez takéto rozhranie to bolo potrebné. Z analýzy som pochopil ako prebieha adresovanie LCD pomocou spoločných a segmentových radičov, čo bolo veľmi dôležité pre pochopenie funkcie jednotlivých riadiacich signnálov. Okrem toho som na základe tejto analýzy dokázal odhadnúť aj celkové prepojenie všetkých 3 typov radičov, čo som znázornil aj v diagrame na obr. 3.1. Podobný diagram je síce dostupný v dokumentácii k radiču MSM6255, ale neudáva presne na aké vstupy sú riadiace signály napojené a zobrazuje aj použitie signálu $\texttt{CE}_\phi$, ktorý v tomto prípade nebol použitý. Diagram zapojenia v dokumentácii bol pre mňa teda v konečnom dôsledku viac mätúci ako nápomocný.

Nakoniec som po prevedení analýzy zistil aj dôležitú skutočnosť, že LCD neobsahuje žiadnu vnútornú pamäť obrázkov a dáta sa doň musia neprestajne odosielať aj keď nedôjde k zmene obrázku. Toto bolo dôležité zistenie, ktoré ovplyvnilo výber technológie pre implementáciu samotného radiča. Radič bude musieť odosielať dáta a generovať priebehy riadiacich signálov pre LCD *bez prestania* s pomerne presným časovaním. Zároveň bude musieť dáta prijímať z rozhrania pre PC a ukládať ich do pamäte. Preto, hoci samotné riadenie LCD z analýzy vyplynulo ako sekvenčné, celkovo bude musieť byť radič schopný vykonávať jednotlivé úkony *paralelne*. Z toho plynie, že pre návrh radiča budem musieť použiť platformu FPGA a radič teda implementovať ako návrh HW, nakoľko HW zo svojej podstaty vykonáva úlohy paralelne.

**Obr. 3.1** Prepojenie jednotlivých radičov LCD

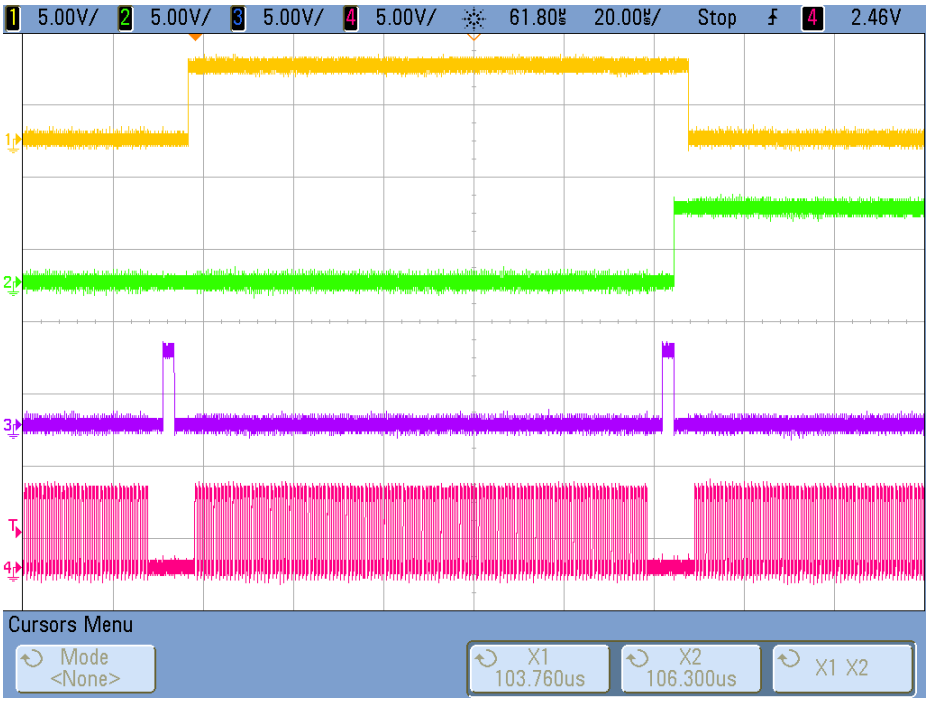



## 3.2 Časová analýza priebehu signálov

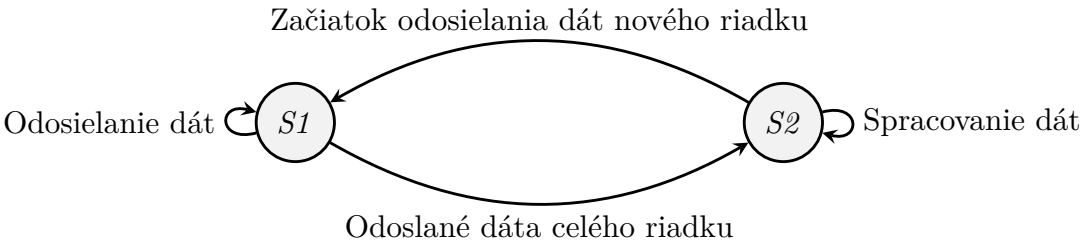
V predchádzajúcej sekcii sa mi podarilo meraním kategorizovať jednotlivé riadiace a datové signály a z ich priebehov získať prehľad o ich funkcii. Nakoľko som sa rozhodol v implementovanom radiči presne imitovať priebehy signálov tak, ako tomu bolo pri použití radiča MSM6255, potreboval som ešte získať presné časové charakteristiky priebehov signálov. V tejto podkapitole sa budem venovať ich opisu tak ako som ich odmeral pomocou osciloskopu. Jednotlivé merania sú znázornené na obrázkoch exportovaných z osciloskopu. Meranie som urobil pomocou kurzorov. Ich umiestnenie je indikované žltými prerušovanými čiarami umiestnenými vertikálne. Meraný čas, ktorý vznikne ako rozdiel pozície týchto kurozorov je viditeľný v ľavej dolnej časti obrázkov, označený ako „ΔX“.

Ešte pred popisom jednotlivých časov je vhodné si rozdeliť priebeh signálu do dvoch samostatných fází, ktoré sa neustále opakujú. Tie sa dajú znázorniť pomocou automatu o 2 stavoch, ktorého diagram je na obr. 3.3. Jednotlivé stavy riadenia LCD sa dajú odlíšiť pomocou aktivity signálu `CLP`. Stav *S1* reprezentuje odosielanie dát na LCD, teda obdobie kedy signál `CLP` osciluje. Stav *S2* naopak reprezentuje stav kedy je signál `CLP` neaktívny. Pri meraní bolo možné pozorovať toto rozdelenie najlepšie na obr. 3.2. K tomuto rozdeleniu som pristúpil, pretože jednotlivé stavy riadenia LCD majú veľmi odlišný účel a udalosti, ktoré sa v nich dejú nepresahujú do druhého stavu. Z toho dôvodu bude toto rozdelenie vhodné aj neskôr pri implementácii radiča.

**Obr. 3.2** Priebeh riadiacich signálov radiča LCD počas odosielania dát jedného riadku. Zhora dole: FRP, FRMB, LIP a CLP




**Obr. 3.3** Diagram automatu, ktorý reprezentuje 2 stavy riadenia LCD




Stav *S1* je charakteristický nie len osciláciou signálu `CLP` a odosielaním dát, ale hlavne tým, že počas celej tejto fázy nedochádza *nikdy* k zmene riadiacich signálov `FRP`, `FRMB` a `LIP` čo je jasne vidno na obr. 3.2 a A.2. Tieto signály ostávajú vždy v takej log. úrovni, v akej boli keď táto fáza započala prvou vzostupnou hranou `CLP`. Jediným dôležitým časovým údajom pre túto fázu je perióda signálu `CLP`, ktorú označím ako $T_{CLP}$. Na obr. A.5 je znázornené jeho meranie. Nameraná hodnota je 840ns, časové požiadavky z tab. 3.2 sú teda splnené.

Stav *S2*, charakteristický tým, že je signál `CLP` konštantne v log. 0 je presným opakom prvej fázy. Ako je možné vidieť na obr. A.3 dáta zostávajú v tomto stave v takej log. úrovni v akej boli keď doň vstúpili. Zatiaľ čo signál `CLP` je neaktívny, ostatné signály sa menia práve v tomto stave riadenia. Zatiaľ čo signál `LIP` vykoná pulz v každej instancii tohto stavu, signály `FRP` a `FRMB` sa menia iba v určitých instanciách v závislosti od aktuálne odoslaného riadku čo som konkrétnejšie opísal v predchádzajúcej podkapitole.

Z analyzovaných vlastností priebehu signálov v stave *S2* som dospel k záveru, že nie je potrebné skúmať vlastnosti jednotlivých signálov `FRP`, `FRMB` a `LIP` ako napr. ich frekvenciu. Pre návrh radiča je oveľa prínosnejšie meraním všetkých 4 signálov súčastne analyzovať časové rozdiely medzi jednotlivými zmenami signálov v stave *S2*. Postupne som teda definoval a analyzoval 5 takýchto časov, ktorých prehľad je v tab. 3.8

Prvým z nich je čas medzi poslednou zostupnou hranou signálu `CLP` a vzostupnou hranou signálu `LIP`, ktorý označím ako $t_{CL}$. Meranie tohto času je znázornené na obr. A.6, pričom nameraná hodnota je 3.39$\mu$s. Pre tento konkrétny čas neudáva dokumentácia žiadne podmienky.

Ďalší dôležitý časový úsek v poradí je doba po ktorú je signál `LIP` aktívny, označený ako $t_{LL}$. Toto meranie je možno vidieť na obr. A.2, kde bola nameraná hodnota 2.54$\mu$s. Časové podmienky určené pre tento signál v tab. 3.2 (vstup `CP`) a 3.4 (vstup `LOAD`) sú splnené.

Ďalej som meral čas $t_{LF}$ medzi zostupnou hranou signálu `LIP` a invertovaním signálu `FRP`, čo je možné zhliadnuť na obr. A.7. Pre tento signál platia časové požiadavky vstupu $\texttt{IO}_1$ v tab. 3.2, ktoré sú splnené.

Ďalší čas ktorý som meral som označil $t_{FC}$. Jeho meranie je znázornené na obr. A.8. Nameraná hodnota je približne 1.45$\mu$s. Pre tento čas nie je stanovená žiadna podmienka, keďže signály `FRP` a `CLP` nie sú napojené na rovnaké radiče.

Posledný meraný čas bol medzi zostupnou hranou signálu LIP a prvou vzostupnou hranou signálu `CLP`, ktorý som označil ako $t_{LC}$. Nameraná hodnota je 4.59$\mu$s, podľa obr. A.9. Takýto výsledok som očakával, nakoľko sa jedná o súčet časov $t_{LF}$ a $T_{FC}$. Napriek tomu som považoval za potrebné odmerať tento čas, nakoľko sa aplikuje v prípade, že boli odoslané dáta riadku iného ako 1. alebo 2. (kedy dochádza k zmene `FRP` a `FRMB`). Časové požiadavky pre vstupy `LOAD` a `CP` v tab. 3.4 sú splnené.

| $t_{CLP}$ | $t_{CL}$ | $t_{LIP}$ | $t_{LF}$ | $t_{FC}$ | $t_{LC}$ |
|---|---|---|---|---|---|
| 840ns | 3.4$\mu$s | 2.52$\mu$s | 3.14$\mu$s | 1.45$\mu$s | 4.6$\mu$s |

**Tabuľka 3.8** Prehľad nameraných hodnôt analyzovaných časov

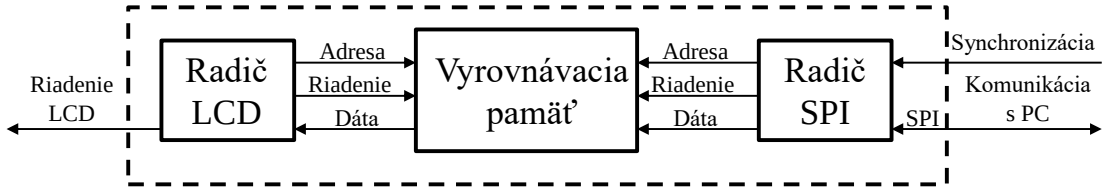
# Kapitola 4
# Návrh riešenia

*V kapitole opíšem návrh prototypu radiča LCD, ktorý bude slúžiť ako náhrada za starý radič typu MSM6255. Najprv predstavím celkové riešenie a jeho rozdelenie na 3 hlavné časti. Potom pre jednotlivé časti postupne opíšem ich funkciu a vlastnosti.*

Ako už vyplynulo z analýzy, radič LCD bude potrebné implementovať na platforme FPGA. Návrh riešenia opísaný v tejto kapitole je teda z pohľadu návrhu HW.

Celkové riešenie kompletného prototypu radiča LCD sa dá rozdeliť do 3 najdôležitejších častí. Ide radič LCD, vyrovnávaciu pamäť a radič rozhrania SPI. Na obr. 4.1 je nakreslený blokový diagram znázorňujúci tieto 3 časti a ich vzájomné prepojenie.

**Obr. 4.1** Blokové schéma návrhu prototypu radiča LCD




## 4.1 Návrh radiča LCD

Radič LCD bude slúžiť pre obsluhu riadiacich a datových signálov na rozhraní LCD panela. Bude čo najpresnejšie imitovať časové prechody signálov tak, ako boli namerané pri použití pôvodného radiča typu MSM6255. Aj keď by bolo možné jednotlivé časové charakteristiky upraviť v súčinnosti s dokumentáciou, takéto riešenie som zvolil z toho dôvodu aby to zväčšilo názornosť tejto práce. Radič bude získavať dáta, ktoré odošle na LCD z vyrovnávacej pamäte, ktorú bude ovládať tak, že signálom bude od pamäti požadovať nový obrázok. Radič nebude vedieť o stave zaplnenosti pamäte, ani nebude nijak zisťovať či dostal nový alebo rovnaký obrázok. Jeho úlohou teda bude vygenerovať riadiace signály a dáta z pamäte len korektne preposlať na LCD.

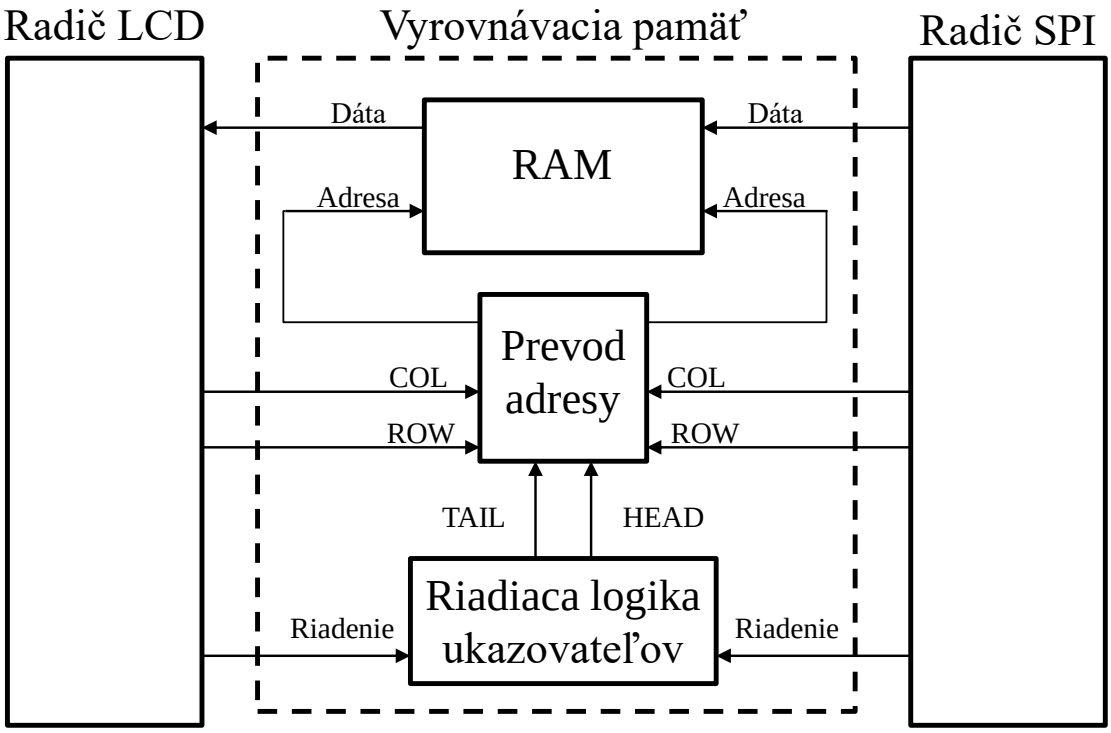
## 4.2 Návrh vyrovnávacej pamäte

Vyrovnávacia pamäť bude slúžiť pre jednoduchý a bezpečný prenos dát medzi ich prijatím na rozhraní SPI a odoslaním na LCD pomocou radiča LCD. Vyrovnávacia pamäť je dôležitá nielen preto, že frekvencia zobrazovania a prijímania dát sa môže líšiť, ale aj preto, lebo na SPI dáta nemusia prichádzať kontinuálne. Nakoľko radič bude na LCD niečo zobrazovať aj v takej situácii, je potrebné aby nejaké dáta zostali uložené v pamäti. Vlastnosti, ktoré som pre pamäť určil sú následovné:

1. Pamäť bude 2-bránová. Toto je potrebné pre to, aby do nej mohol zapisovať radič SPI a zároveň čítať dáta radič LCD. Vzhľadom k tomu, že radič SPI bude iba zapisovať a radič LCD iba čítať môže sa jednať o zjednodušenú 2-bránovú pamäť, ktorá pomocou jednej adresy umožní iba zápis a pomocou druhej iba čítanie.
2. Veľkosť slova v pamäti bude 4 bity. Takáto veľkosť je najvhodnejšou pretože radič LCD odosiela dáta na LCD po dátovej zbernici o šírke 4b.
3. Pamäť bude možné adresovať pomocou súradníc, ktoré označím ako `ROW` a `COL`. Toto adresovanie som zvolil z toho dôvodu, že radič LCD aj radič SPI budú interne pracovať so súradnicami jednotlivých 4-bitových položiek a môžu ich teda priamo použiť aj pre prácu s pamäťou.
4. Posledný platný obrázok v pamäti sa pri žiadosti o nový obrázok neodstráni a bude sa naďalej zobrazovať. Zachovať posledný obrázok je dôležité pretože radič LCD musí na displej neustále odosielať dáta, aj keď žiadne nie sú prijímané.
5. Ak bude pamäť plná nedôjde k pretečeniu a prepísaniu obrázku, ktorý sa aktuálne odosiela na LCD, ale k prepísaniu naposledy vloženého obrázku. Toto riešenie som zvolil pretože to zjednoduší riadiacu logiku zápisu dát do pamäte radiča SPI.
6. V pamäti bude na prvej pozícii uložený základný obrázok, ktorý pamäť nebude prepisovať. Tento obrázok sa po začatí prijímania nových obrázkov už nebude zobrazovať aj keď bude naďalej v pamäti. Po resetovaní celého prípravku sa začne znovu zobrazovať. Túto požiadavku som zvolil aby radič zobrazoval na displeji aspoň nejaký obrázok aj bez nutnosti prenosu cez SPI.
7. Pamäť bude disponovať celkovým priestorom pre 4 obrázky. Tým sa zabezpečí že pre bezpečný prenos dát bude postačovať aj jednoduchá logika riadenia ukazovateľov.

Pre lepšie pochopenie vymenovaných vlastností je na obr. 4.2 nakreslený blokový diagram znázorňujúci hlavné časti vyrovnávacej pamäte.

**Obr. 4.2** Blokové schéma návrhu vyrovnávacej pamäte




## 4.3 Návrh radiča SPI

Pre komunikáciu s PC využijem štandardné rozhranie, ktoré je už dostupné na použitom PC. Nakoľko pre účely tejto práce používam zariadenie Raspberry Pi 3 A+, mohol som si vybrať z 3 dostupných sériových rozhraní. Každé z nich má na prípravku dedikované výstupy a operačný systém pre tieto zariadenia disponuje aj ovládačom pre tieto rozhrania. Ide o rozhrania UART, $I^2C$ a SPI. Z tých som vybral rozhranie SPI z nasledujúcich dôvodov:

1. Je z nich najrýchlejšie a bežne sa práve toto rozhranie používa k prepojeniu s periférami, ktoré vyžadujú rýchly prenos veľkého množstva dát, ako napr. pamäť alebo LCD. [4]
2. Jeho protokol je pomerne jednoduchý a je možné si ho prispôsobiť.
3. Je veľmi jednoduché na fyzickú realizáciu. Používa napäťové úrovne kompatibilné s TTL a CMOS zariadeniami, čiže log. 1 je pri napätí väčšom ako 2,4V, zatiaľ čo pre log. 0 je použité napätie menšie ako 0,4V. V praxi to znamená, že je možné výstupy rozhrania SPI napojiť priamo na vstupy FPGA platformy. [2]

Radič SPI bude teda slúžiť na komunikáciu s PC pomocou rozhrania SPI, na ktorom bude prijímať obrazové dáta. Do počítača bude odosielať späť prijaté dáta pre overenie funkčnosti v testovacej aplikácii. Prijaté obrazové dáta uloží do vyrovnávacej pamäte. Konkrétne vlastnosti navrhnutého radiča budú následovné:

1. Na zbernici SPI sa bude tento radič chovať ako *slave*.
2. Radič bude pracovať s dĺžkou slova 8 bitov. Je to z dôvodu, že Raspberry Pi dokáže na rozhraní SPI pracovať len so slovami takejto dĺžky.
3. Radič bude schopný prijímať rovnako tak klasické prenosy, ako aj viac-slovné prenosy pri ktorých sa prenesie viacero slov bez deaktivácie signálu $\overline{\text{SS}}$. Toto som zvolil z dôvodu možnosti rýchleho odosielania veľkého množstva dát.
4. Radič bude pracovať na rozhraní SPI s nastavením módu prenosu na *CPOL=0*, pretože si myslím, že by signál SCLK mal byť v nečinnosti v log. 0 a *CPHA=1*, pretože si myslím, že by k zmene dát malo dochádzať vždy synchrónne so signálom SCLK (toto nie je splnené v prvej zmene dát pri nastavení *CPHA=0*). Mód prenosu som si mohol zvoliť ľubovoľne, pretože Raspberry Pi podporuje všetky štyri.
5. Radič bude predpokladať, že PC odosiela obrazové dáta postupne od prvého po posledný bajt obrázku.
6. Radič bude možné synchronizovať pomocou dedikovaného signálu. Po aktivácii signálu bude radič predpokladať, že sa začne odosielať nový obrázok od začiatku.
7. Radič bude sám ovládať adresu na ktorú zapíše obrazové dáta, adresa teda nebude súčasťou SPI prenosu.
8. Radič bude ovládať vyrovnávaciu pamäť tak, že bude požadovať nové miesto v pamäti pre nový obrázok. Signalizovať to bude vždy po kompletnom prijatí obrázku. Nebude zisťovať, či a ako túto požiadavku pamäť spracovala.

# Kapitola 5

# Realizácia riešenia

*V tejto kapitole opíšem postup realizácie môjho riešenia a riešenie jednotlivých problémov, ktoré pri implementácii vznikly. Pri realizácii budem postupovať v rovnakom poradí ako v predchádzajúcej kapitole.*

Nakoľko som chcel aby táto práca bola čo najzrozumiteľnejšia, často som pri implementácii použil riešenie pomocou stavového automatu. Nielenže je popis stavových automatov v zdrojovom kóde veľmi zrozumiteľný, ale je aj možné ich znázorniť na diagrame. Vždy keď to bolo možné som teda zvolil realizáciu práve pomocou stavového automatu. Všetky stavové automaty použité v tejto realizácii sú typu Moore.

Kompletne navrhnutý radič bude reprezentovaný hlavnou entitou nazvanou `TOP`. Jednotlivé úkony, ktoré má vykonávať hotový radič sú reprezentované samostatnými komponentami, ktoré sa ďalej členia podľa ich čiastkových úloh. Logická štruktúra celého projektu je následovná:

- `TOP`
  - `DEBOUNCE` ..... ošetrenie zákmitov tl. RESET
  - `LCD_MANAGER` ..... radič LCD
    - `LCD_CONTROLLER` ..... automat riadenia LCD
    - `FRP_CONTROLLER` ..... automat invertovania FRP
    - `FRMB_CONTROLLER` ..... automat invertovania FRMB
  - `RING_BUFFER` ..... vyrovnávacia pamäť
    - `MEM_INST` ..... vygenerovaná pamäť z BRAM
  - `WRITE_MANAGER` ..... príjem dát z SPI a ich zápis do pamäte
    - `SPI_SLAVE` ..... radič SPI
    - `WRITE_CONTROLLER` ..... radič zápisu dát

Jednotlivé komponenty sú prepojené tak ako to už bolo znázornené v predchádzajúcej kapitole na obr. 4.1. Ako som vysvetlil pri návrhu pamäte, adresa je zložená z 2 adresných zberníc a to `ROW` a `COL`.

Komponenta `DEBOUNCE` slúži na ošetrenie zákmitov, ktoré vznikajú pri stlačení tlačítka, ktoré slúži ako vstup `RESET`. Zdrojový kód tejto entity som prevzal z knižnice poskytnutej fakultou v rámci predmetu BI-SAP, preto ho ďalej nebudem popisovať.

Pri implementácii radiča SPI som narazil na problém s metastabilitou. Preto obsahuje entita `TOP` okrem vymenovaných komponent aj proces pre ošetrenie metastability dát na vstupe. Vstupné signály u ktorých je potrebné tento problém ošetriť sú `SPI_SS`, `SPI_SCLK`, `SPI_MOSI` a `SYNCHR`. Metastabilitu týchto vstupov som ošetril štandardným spôsobom opísaným v teoretickej časti, teda pomocou D-F/F a jeho realizáciu je možné vidieť v priloženom výpise kódu B.1.

## 5.1 Radič LCD

V tejto podkapitole sa budem najprv venovať entite `LCD_MANAGER` a potom podrobnejšie opíšem jej jednotlivé komponenty a ich fungovanie.

Táto entita figuruje v rámci môjho riešenia ako radič LCD. Entita dokáže fungovať aj samostatne, pričom v takom prípade vie zabezpečiť riadenie LCD, ale iba so statickými dátami, čo znamená, že každé 4-bitové slovo, ktoré odošle na LCD je rovnaké. Dôležité je, že kompletne zabezpečuje generovanie priebehu riadiacich signálov pre LCD.

Pre riadenie LCD obsahuje táto entita 2 komponenty. Ide o `LCD_CONTROLLER`, ktorá riadi priebeh riadiacich signálov a kompentu `SIGNAL_CONTROLLER`. Tá je instanciovaná samostatne pre signál `FRP` a signál `FRMB` a slúži na riadenie ich logickej úrovne.

V analýze som priebeh riadiacich signálov rozdelil do dvoch stavov (obr. 3.3). Toto rozdelenie využijem aj v tejto podkapitole. Ako som v analýze spomínal, riadiace signály `FRP` a `FRMB` sa menia iba v stave *S2* a v rovnakej logickej úrovni potom zostanú po celú dobu stavu *S1*. Ak by ich celý priebeh mal riadiť hlavný automat, musel by si počas vykonávania úkonov stavu *S1* „pamätať“ ich logickú úroveň. To by bolo zbytočne neefektívne. Vyriešil som to tak, že hlavný automat iba vyšle pokyn na ich invertovanie a starosťo konkrétnu logickú úroveň som prenechal na samostanú komponentu.

V entite `SIGNAL_CONTROLLER` som funkciu invertovania signálov `FRP` resp. `FRMB` implementoval ako jednoduchý stavový automat o 4 stavoch, ktorý vždy pri vzostupnej hrane vstupu `SIGNAL_CHANGE` invertuje signál na výstupe. Diagram navrhnutého automatu je možné vidieť na priloženom obr. B.2

### 5.1.1 Generovanie priebehu riadiacich signálov

Vzhľadom k tomu, že nejde o úplne triviálny radič potrebuje k správnemu fungovaniu, okrem samotného stavového automatu aj niekoľko pomocných procesov. Najprv opíšem tieto procesy a potom samotný automat.

V prvom rade, keďže môj radič preposiela dáta na datovú zbernicu priamo z výstupu pamäte, implementoval som v jednom procese jednoduchý 4-bitový register, ktorý uloží dáta zo vstupu (a teda vystaví na výstup) ak je aktivovaný signálom `LD_DATA`. Tým zabezpečí ich stabilitu počas zostupnej hrany signálu CLP a teda dodržanie predstihu a presahu.

Ako som už ukázal v analýze, použitý LCD musí byť riadený neustále a s rozdielným časovaním pre jednotlivé úkony. To znamená, že automat, ktorý realizuje funkciu radiča, bude rozhodovať o prechodoch medzi stavmi nie na základe vstupov, ale na základe časovača. Pre tento účel som implementoval 2 nezávislé časovače s ktorými automat pracuje a ktoré sú implementované pomocou dvoch procesov. Jeden proces popisuje čítač, ktorý je možno nastaviť do požadovanej hodnoty signálom `LD_CNTx` a dekrementovať

signálom EN_CNTx, ide teda o sekvenčnú časť časovača. Druhý proces reprezentuje jeho kombinačnú časť. Reaguje na jeho hodnotu a ak detekuje, že sa dostal do nuly, vyšle do automatu riadiaci signál ZEROx. Použitie 2 časovačov bolo potrebné z toho dôvodu, že viaceré stavy, ktoré potrebujú časovanie nasledujú hneď ze sebou. V takom prípade sa nedá použiť jeden časovač, nakoľko oba stavy potrebujú tento časovač držať spustený a tým pádom by nebol priestor na jeho nastavenie. Naopak 2 časovače je možné striedať a teda je čas na ich nastavenie.

Vzhľadom k tomu, že pracujem s displejom o určitej výške a šírke, je potrebné aby radič vždy „vedel“ na akých súradniciach sa aktuálne nachádza. Pre tento účel som implementoval 2 pomocné čítače, riadkový a stĺpcový, každý v samostatnom procese. Obe sú resetovateľné a ich inkrementácia sa povoľuje riadiacim signálom. Pri riadkovom čítači využívam efekt pretečenia. Použil som datový typ unsigned o šírke 7 bitov, hodnota 127 sa teda inkrementuje späť do 0. Počet stĺpcov je vo vnímaní tohto radiča 120, nakoľko sa dáta odosielajú po 4 bitoch naraz. Pre stĺpcový radič som teda implementoval umelé pretečenie, ktoré som nastavil tak, že ak je jeho hodnota 119 a má dôjsť k pričítaniu, namiesto toho sa nastaví späť do 0. Jeho implementáciu je možné zhliadnúť v priloženom výpise kódu B.2.

#### 5.1.1.1 Stavový automat

Z analýzy signálov som už zistil, že ich priebeh je sekvenčný a jednotlivé udalosti nastávajú potupne za sebou. S výnimkou už spomenutého invertovania signálov je teda možné riadenie LCD reprezentovať jedným stavovým automatom, ktorý je znázornený diagramom na priloženom obr. B.1.

Stav riadenia *S1*, kedy dochádza k odosielaniu dát na LCD a aktivite CLP je implementovaný postupne pomocou stavov CLP_DOWN, LOAD_DATA, CLP_UP a COL_INC. Prirodzene som nemohol dáta ukládať do registra v stavoch CLP_DOWN, kedy by došlo k poručeniu presahu a CLP_UP, kde by došlo k porušeniu predstihu, pretože by sa mohli zmeniť tesne pred prechodom, vytvoril som pre to teda samostatný stav. Inkrementácia stĺpcového čítača si vyžaduje samostatný stav, pretože signál COL_EN_CNT smie byť aktivovaný len po dobu jedného hodinového taktu. Nakoľko stĺpcový čítač slúži ako súčasť adresy pre vyrovnávaciu pamäť, umiestnil som tento stav tak, aby k zmene nedochádzalo počas načítania dát z pamäte.

Stav riadenia *S2*, kedy dochádza k zmenám ostatných riadiacich signálov nastáva po odoslaní všetkých dát na LCD s tým, že signál CLP ostáva deaktivovaný. V prípade implementovaného automatu to teda znamená, že zo stavu CLP_DOWN vedú 2 rôzne hrany do ďalších stavov. Tie nastávajú pri aktivácii signálu ZERO2 a o konkrétnej hrane sa rozhodne na základe stĺpcového čítača. Ak je v hodnote 0, teda bolo odoslaných všetkých 120 stĺpcov, prejde do stavu, ktorým sa začína vykonávanie funkcií stavu *S2*.

Ako som ukázal v analýze pri meraní priebehu signálov, prvá polovica priebehu stavu *S2* je vždy rovnaká. Implementoval som ju pomocou stavov ROW_INC, LIP_WAIT a LIP_UP. Pre stav ROW_INC, ktorý slúži na inkrementáciu čítača riadkov rovnako platí, že smie trvať len po 1 hodinový cyklus. Tento stav je zároveň počiatočným stavom automatu po resete.

Po deaktivácii signálu LIP môže dôjsť k 3 rôznym scenárom. Preto zo stavu LIP_UP vedú až 3 rôzne hrany. Opustenie tohto stavu sa sputí signálom ZERO2 a o konkrétnej hrane sa rozhodne podľa aktuálnej hodnoty riadkového čítača. Vzhľadom k tomu, že už bol inkrementovaný predchádzajúcim stavom, jeho hodnota reprezentuje dáta riadku, ktorý ešte len *bude odoslaný.* Jednotlivé možné scenáre sú následujúce:

1. Ak sa budú odosielať dáta prvého riadku (čítač riadkov je v 0), invertuje sa signál FRP. Toto je tiež ideálna príležitosť pre riadenie vyrovnávacej pamäte, aby poskytla nový obrázok. V automate je reprezentovaný stavmi LIP_DOWN_0, TAIL_INC a FRP_INVERT.

2. Ak sa budú odosielať dáta prvého riadku (čítač riadkov je v 1), invertuje sa signál FRP aj FRMB. V automate je reprezentovaný stavom LIP_DOWN_1, ktorý je oproti predchádzajúcemu rozdielny v tom, že v ňom dôjde aj k invertovaniu signálu FRMB a stavom FRP_INVERT, ktorý zdieľa s predchádzajúcim prípadom.

3. V ostatných prípadoch nedochádza k žiadnej ďalšej udalosti. To som implementoval pomocou stavov LIP_DOWN a LOAD_CNT_1. Tento stav nemá žiadnu úlohu z pohľadu riadiacich signálov, ide iba o pomocný stav. Jednak je potrebné nastaviť časovač 1, čo nemôže urobiť stav LIP_DOWN pretože ho používa, okrem toho je tiež potrebné čakať rovnakú dobu ako čaká stav FRP_INVERT, ktorý v tomto prípade nie je použitý.

## 5.2 Kruhová vyrovnávacia pamäť

V tejto podkapitole opíšem entitu RING_BUFFER. Tá implementuje vyrovnávaciu pamäť, ktorá slúži pre jednoduchý a bezpečný prenos dát medzi rozhraním SPI a radičom LCD. Najprv sa budem venovať vygenerovaniu samotnej pamäte z BRAM. Pre tú potom vytvorím pomocné procesy, ktoré zabezpečia chovanie ako som navrhol v predchádzajúcej kapitole. Nakoniec sa budem venovať inicializácii pamäte.

Ako som opísal v teoretickej časti, pamäť je v FPGA možné vytvoriť viacerými spôsobmi. Ja som zvolil jej implementáciu pomocou BRAM najmä kvôli tomu, že pôjde o pomerne veľkú pamäť. Priestor potrebný pre 4 obrázky je až 245760 bitov.

V prvom rade je potrebné vytvoriť entitu, ktorá bude pamäť reprezentovať tak, aby zabezpečila pri syntéze použitie BRAM. Takúto entitu nebudem implementovať manuálne, ale pomocou dedikovaného programu určeného na generovanie pamäte. Použil som program *Block Memory Generator*, ktorý je integrovaný v programe *Vivado 2018.2*, ktorý som používal pre vývoj radiča.

Aj keď je samotná entita vygenerovaná, je potrebné vyplniť nastavenia pamäte. Typ pamäti som nastavil na jednoduchú 2-bránovú pamäť ako som stanovil v požiadavku č. 1. Ďalej som nastavil šírku a hĺbku pamäte. Šírka pamäte je veľkosť jednej položky, nastavil som ju teda na 4 bity podľa požiadavku č. 2. Hĺbka pamäte označuje počet položiek v pamäti a nastavil som ju na 61440. Počet položiek získam ak vynásobím veľkosť obrázka (61440) počtom uložených obrázkov v pamäti (4) a vydelím šírkou pamäti (4). Hĺbka sa teda v tomto prípade rovná 61440. Pre tento počet položiek bola potom samozrejme adresa do pamäti vygenerovaná ako 16-bitová.

Program vygeneroval popis tejto pamäte ako entitu s názvom blk_mem_gen_0, v súbore, ktorý je ibe pre čítanie. Tú som potom použil ako komponentu v rámci entity RING_BUFFER.

### 5.2.1 Realizácia ukazovateľov a stavu zaplnenosti

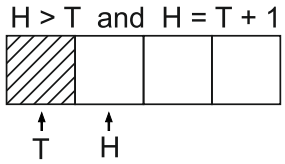
Splnenie ďalších požiadaviek nie je možné urobiť pomocou nastavení pri generovaní pamäte. Preto ich sám implementujem v rámci entity `RING_BUFFER`. Ide najmä o procesy ktoré budú pracovať najmä s adresami a ukazovateľmi do pamäte. Pre zvyšné požiadavky je teda potrebné implmentovať:

- ukazovatele `HEAD` a `TAIL` do pamäte a kontrolnú logiku ich inkrementovania na žiadosť ostatných komponent v závislosti od zaplnenia pamäte. Pre túto úlohu sú potrebné:
  - procesy na detekciu stavu pamäte, teda či je pamäť plná prípadne prázdna
  - čítače ukazovateľov, ktoré ich inkrementujú iba pri splnení prílušných podmienok odvíjajúcich sa od zaplnenia pamäte
- prevod ukazovateľa asúradníc `ROW` a `COL` na ucelenú adresu v pamäti

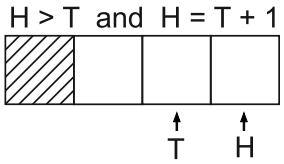
Najprv si stanovím podmienky pri ktorých budem pamäť považovaná za plnú, resp. prázdnu. Nakoľko vytváram vyrovnávaciu pamäť, ide o pamäť typu *FIFO*. To znamená, že pamäť bude *prázdna* ak je ukazovateľ `TAIL` hneď *za* ukazovateľom `HEAD`. Naopak *plná* bude ak bude ukazovateľ `TAIL` hneď *pred* ukazovateľom `HEAD`. Oproti štandardnému riešeniu pamätí FIFO sa líši tým, že na prvej pozícii bude nemenný obrázok, ako uvádza požiadavka č. 6. Ten sa nebude prepisovať a bude sa zobrazovať iba po resete, preto som potreboval presne špecifikovať situácie, ktoré môžu nastať.

Pri rozhodovaní, či je pamäť *prázdna* nehrá základný obrázok veľkú rolu. Ako je možné vidieť na obr. 5.1a a 5.1b, podmienka zostáva rovnaká. Malá zmena nastáva v prípade na obr. 5.1c, kde je za za začiatok pamäte považovaný index 1, nie 0 ako by tomu bolo v štandardnom prípade.

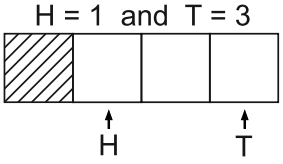
**Obr. 5.1** Prípady v ktorých je pamäť považovaná za prázdnu




**(a)** `TAIL` je hneď za `HEAD` v stave po resete keď sa zobrazuje základný obrázok.



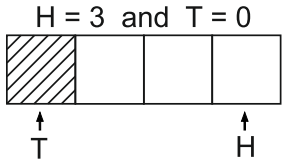

**(b)** `TAIL` je hneď za `HEAD` v štandardnej situácii.



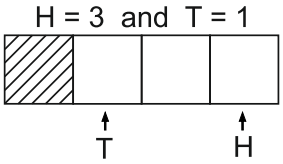

**(c)** `TAIL` je na konci pamäte. Použije sa špeciálna podmienka.

Naopak pri rozhodovaní, či je pamäť *plná*, vznikli kvôli základnému obrázku nie 2, ale až 3 podmienky. V prípade, že sú ukazovateľe „hneď za sebou“, teda platí, že `TAIL` je väčší ako `HEAD`, žiadna špeciálna situácia nenastane ako vidno na obr. 5.2c. Ale v prípade, že sa ukazovateľ `HEAD` nachádza na konci pamäte, je nutné pamäť považovať za plnú keď je ukazovateľ `TAIL` na indexe 0 (obr. 5.2a), ale aj ak je na indexe 1 (obr. 5.2b)

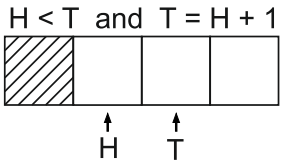
**Obr. 5.2** Prípady v ktorých je pamäť považovaná za plnú




**(a)** `HEAD` je na konci pamäte v špeciálnej situácii kedy `TAIL` stále ukazuje na základný obrázok.




**(b)** `HEAD` je na konci pamäte v štandardnej situácii, teda `TAIL` ukazuje na začiatok prepisovateľnej pamäte.




**(c)** `TAIL` je hneď pred `HEAD`.

Takto definované podmienky som implementoval v dvoch kombinačných procesoch pomocou príkazu `if`. Tie ovládajú signály `EMTPY` a `FULL`, ktoré slúžia pre riadenie inkrementácie ukazovateľov.

Samotné ukazovatele sú implementované pomocou datového typu `unsigned` a pre ich inkrementovanie je pre každý ukazovateľ vytvorený vlastný proces. Jedná sa o sekvenčnú logiku, nakoľko ide o čítače. K inkrementácii dôjde pri každej vzostupnej hrane hodinového signálu ak platia obe podmienky:

- Inkrementovanie je povolené signálom, ktorý prichádza z entity `LCD_MANAGER` alebo `WRITE_MANAGER`.
- Z hľadiska stavu zaplnenosti pamäte je možné konkrétny čítač inkrementovať. V závisloti od ukazovateľa platia tieto podmienky:
  - Podľa požiadavky č. 5 je možné ukazovateľ `TAIL` inkrementovať iba v prípade, že pamäť *nie je prázdna*, teda `EMPTY=0`. Podľa požiadavky č. 6 sa pri pretečení nastaví na hodnotu 1 aby preskočil základný obrázok na pozícii 0.
  - Podľa požiadavky č. 4 je možné ukazovateľ `HEAD` inkrementovať iba v prípade, že pamäť *nie je plná*, teda `FULL=0`. Podľa požiadavky č. 6 sa pri pretečení nastaví na hodnotu 1 aby neprepísal základný obrázok

Nakoniec je potrebný prevod súradníc na adresu do pamäte. Výber jednotlivej položky závisí od 3 premenných a to: `ROW`, `COL` a `TAIL/HEAD`. Použitý ukazovateľ závisí od toho, či ide o adresu pre zápis alebo čítanie dát. Dáta obrázkov sú v pamäti za sebou usporiadané postupne, preto je pre získanie adresy potrebné všetky premenné vynásobiť nejakou konštantou a sčítať. Jednotlivé konštanty som určil následovne:

- `IDX_MULTIPLIER = 15360`, pretože v jednom obrázku je obsiahnutých 61440 pixelov, čo sa po vydelení 4 (veľkosť 1 položky) rovná 15360
- `ROW_MULTIPLIER = 120`, pretože v jednom riadku je obsiahnutých 480 pixelov, čo sa po vydelení 4 rovná 120
- `COL_MULTIPLIER = 1`, pretože stĺpec je najmenšou jednotkou a teda ho nie je potrebné násobiť žiadnou konštantou

Zloženie adresy som implementoval pomocou paralelného priradenia ako je možné zhliadnuť vo výpise kódu 5.1. Datový typ `integer` disponuje matematickými operáciami ako sčítanie a násobenie bez nutnosti vlastnej implementácie sčítačky či násobičky. Toho som využil tak, že najprv všetky premenné prevediem na `integer` a následne späť na `std_logic_vector`, pretože adresa do pamäte musí byť v tomto datovom type.

```
1 RD_ADDR  <= std_logic_vector(
2                 to_unsigned((IDX_MULTIPLIER * (to_integer(TAIL)))
3                 + (ROW_MULTIPLIER * to_integer(unsigned(RD_ROW)))
4                 + to_integer(unsigned(RD_COL)), 16));
5 WR_ADDR  <= std_logic_vector(
6                 to_unsigned((IDX_MULTIPLIER * (to_integer(HEAD)))
7                 + (ROW_MULTIPLIER * to_integer(unsigned(WR_ROW)))
8                 + to_integer(unsigned(WR_COL)), 16));
```

**Výpis kódu 5.1** Prevod súradníc a ukazovateľa do pamäti na adresu.

### 5.2.2 Inicializácia pamäte

Pre splnenie požiadavky č. 6 som v pamäti ešte napevno implementoval jeden základný obrázok. Ten som implementoval pomocou možnosti inicializácie pamäte. Táto možnosť sa nachádza priamo v programe *Block memory Generator* a hodnota na ktorú sa má inicializovať sa nahráva zo špeciálneho súboru. Súbor `fit-logo.coe` je priložený s ostatnými zdrojovými súbormi projektu.

Obrázok, ktorý som zvolil som najprv previedol na monochromatický o veľkosti 480x128 pixelov vo formáte `bmp`. Z toho som následne získal binárne dáta, ktoré som uložil do výsledného súboru ako `memory_initialization_vector`. Nakoniec som do tohto súboru pridal ešte definovanie číselného základu dát pomocou príkazu `memory_initialization_radix=2`. Takto vytvorený súbor som potom použil pri generovaní pamäte pre jej inicializáciu.

## 5.3 Radič prijímania dát a ich zápisu do pamäte

V tejto kapitole opíšem entitu `WRITE_MANAGER`, ktorá je schopná prijímať dáta pomocou rozhrania SPI a tie následne uložiť do pamäte. Táto entita slúži pre zaobalenie 2 komponent, ktoré vykonávajú čiastkové úlohy.

Ide o komponentu `SPI_SLAVE`, ktorá je implementáciou SPI protokolu pre rozhranie s PC. Po každom prenose sú prijaté dáta dostupné na výstupe tejto komponenty a ich dostupnosť je indikovaná deaktiváciou signálu `BUSY`.

Na základe tohto signálu si dáta prevezme komponenta `WRITE_CONTROLLER`. Tá rozdelí prijaté 8-bitové slovo na dva 4-bitové slová, ktoré následne uloží do pamäte.

### 5.3.1 Radič rozhrania SPI

V tejto časti sa budem venovať entite `SPI_SLAVE`. Tá implementuje rozhranie SPI pre príjem dát z PC. Rozhranie SPI disponuje vlastným hodinovým signálom pre synchrónne odosielanie dát. Ja som celý radič vytvoril ako *synchrónny* dizajn s vnútorným hodinovým signálom `CLK`. Rozhodol som sa aj túto entitu vytvoriť ako synchrónnu a hodinový signál rozhrania SPI tak vnímať iba ako riadiaci signál. Všetky vstupy rozhrania SPI som tak synchronizoval s hodinovým signálom radiča už v entite `TOP`, čo som opísal na začiatku kapitoly.

Nakoľko som zvolil implementáciu tejto entity ako synchrónnu k `CLK`, mohol som ju implementovať ako stavý automat. Tento spôsob som zvolil, rovnako ako v predchádzajúcich prípadoch, pre dosiahnutie vysokej zrozumiteľnosti návrhu. Navrhnutý radič, ktorého diagram je priložený aj na obr. B.3, pozostáva z následujúcich stavov:

`SLEEP` - indikuje, že na zbernici aktuálne neprebieha prenos a teda je možné spracovať prijaté dáta. Všetky ostatné stavy doň prejdú hneď pri detekcii, že $\overline{SS}$ bol deaktivovaný. Pre splnenie požiadavky č. 3 je tiež nutné, aby doň automat prešiel po zistení, že došlo k prenosu 8 bitov. Na to slúži pomocný čítač.

`SCLK_UP_SHIFT` - označuje detekciu vzostupnej hrany signálu `SCLK`. Slúži na aktiváciu posuvného registra, teda vystavenie nových dát na výstup `SPI_MISO`. V ďalšom hodinovom takte prechádza do stavu čakania.

`SCLK_UP_WAIT` - automat v tomto stave čaká na zostupnú hranu signálu `SCLK`.

`SCLK_DOWN_SHIFT` - detekovanie zostupnej hrany signálu `SCLK`. Dôjde k uloženiu vstupu `SPI_MOSI`. V ďalšom hodinovom takte prechádza do stavu čakania alebo do stavu `SLEEP` ak bol práve uložený posledný, teda ôsmy bit.

`SCLK_DOWN_WAIT` - automat v tomto stave čaká na vzostupnú hranu signálu `SCLK`.

Okrem implementácie samotného automatu potrebuje radič rozhrania SPI k svojej funkcii aj niekoľko pomocných procesov. V prvom rade, je to čítač uložených bitov, ktorý je potrebný pre splnenie požiadavky č. 3. Jedná sa o jednoduchý 3-bitový čítač, ktorý sa inkrementuje pri každej vzostupnej hrane hodinového signálu kedy je to povolené povoľovacím vstupom, ktorý ovláda stavový automat.

Ďalšie 2 procesy sú `SHIFT_REG` a `OUT_REG`. Prvý z nich slúži na postupné ukládanie dát prichádzajúcich z PC, je teda implementovaný ako 8-bitový resetovateľný posuvný register s povoľovacím vstupom. Druhý z nich slúži na uloženie dát na výstup `SPI_MISO`. To vykonáva pomocou 1-bitového resetovateľného registra s povoľovacím vstupom. Ten pri aktivácii uloží na výstup konkrétny bit z 8-bitových dát na vstupe a to podľa aktuálnej hodnoty čítača.

### 5.3.2 Radič zápisu dát

V tejto časti sa budem venovať entite `WRITE_CONTROLLER`. Jedná sa o malý radič, ktorý spravuje zápis dát do pamäte. Dáta preberá z entity `SPI_SLAVE`, pri detekcii deaktivácie signálu `BUSY`. Tieto dáta majú veľkosť 8 bitov, radič ich teda rozdelí na dva 4-bitové slová a postupne uloží do pamäte.

Tento radič disponuje, podobne ako radič LCD, riadkovým a stĺpcovým čítačom. Radič počíta s tým, že PC bude odosielať dáta obrázku za sebou postupne preto používa čítače, ktoré postupne inkrementuje. Tieto „súradnice“ využíva hlavne ako adresu dát do pamäti. Sekundárne ich využíva aj na detekciu ukončenia dát jedného obrázku, kedy vyšle signál do pamäte so žiadosťou o nové miesto na ukládanie nového obrázku. Tieto radiče sú implementované rovnakým spôsobom ako som opísal v časti venujúcej sa radiču LCD.

Okrem toho disponuje aj registrami pre rozdelenie vstupných dát na 2 časti. Horná polovica dát sa ukladá do registra, ktorý je napojený priamo na výstup do pamäte, zatiaľ čo dolná polovica dát sa dočasne uloží do druhého registra, z ktorého sa potom uloží do prvého registra.

Zároveň obsahuje register pre dáta na výstupe, ktoré nemajú byť odoslané do pamäte, ale do radiča SPI kde sa odošlú do PC. Pre účely jednoduchej testovacej aplikácie som implementoval radič tak, že na tento výstup priamo uloží prijaté dáta. PC tak pri ďalšom prenose príjme späť dáta, ktoré odoslal. To som implementoval pomocou obyčajného 8-bitového registra s povoľovacím vstupom.

Samotnú riadiacu logiku zápisu dát do pamäte som implementoval ako stavový automat. Jeho diagram je znázornený na priloženom obr. B.4. Priebeh diagramu sa dá rozdeliť na 3 logické časti. Prvá z nich je čakanie na prebehnutie prenosu na rozhraní SPI. Ten sa detekuje tak, že sa signál `BUSY` aktivuje (prenos začal) a následne deaktivuje (prenos skončil). Pre tento účel slúžia stavy `BUSY_DOWN` a `BUSY_UP`.

Akonáhle detekujem skončenie prenosu, uložím si prijaté dáta do registrov pre hornú a dolnú polovicu a začnem ich odosielať do pamäte. Pre tento účel slúžia stavy od `DATA_SAMPLE` až po `STOP_LOWER`, ktoré idú za sebou. Medzi nimi sa nachádza aj stav `UPDATE_LOWER`, ktorý nielen, že presunie dolnú polovicu dát na výstup, inkrementuje aj čítač stĺpcov, pretože dolnú polovicu dát je potrebné uložiť na novú adresu.

Po uložení dát ešte dôjde k inkrementácii stĺpcového, prípadne aj riadkového radiča. V prípade ak bol prijatý celý obrázok, dôjde aj k vyslaniu signálu `HEAD_CHANGE` do pamäte, ktorý spôsobí aktualizovanie ukazovateľa na miesto v pamäti kam sa majú dáta zapisovať. Pre tento účel slúžia stavy `UPDATE_COL`, `UPDATE_COL_ROW` a `UPDATE_COL_ROW_HEAD`, pričom vždy sa vyberie iba jeden z nich v závislosti od aktuálneho riadku a stĺpca.

Pri práci s pamäťou som samozrejme musel dbať na dodržanie predstihu a presahu dát a adresy. Nakoľko časová analýza implementovaného HW bola úspešná a pamäť z BRAM uloží dáta pri vzostupnej hrane hodinového signálu, pravdepodobne by bolo možné uložiť dáta tým spôsobom, že v jednom stave aktualizujem adresu a dáta, pričom hneď v ďalšom stave povolím zápis do pamäte a v poslednom stave zápis deaktivujem. Ja som sa rozhodol zapisovať dáta opatrnejším spôsobom, kedy najprv dáta a adresu s predstihom aspoň 1 hodinového cyklu aktualizujem, následne povolím zápis signálom `WR_EN` po 2 hodinové takty a nakoniec ponechám presah dát a adresy po dobu aspoň jedného hodinového taktu. Tým som zabezpečil nielen predstih a presah dát, ale aj

predstih a hlavne presah signálu `WR_EN`. Nakoľko tento signál je ovládaný automatom, ktorý mení stavy pri vzostupnej hrane hodinového signálu, dochádza aj k zmene tohto signálu veľmi blízko pri vzostupnej hrane hodinového signálu, teda v dobe kedy by tento signál mal byť stabilný. Týmto spôsobom som sa teda zabezpečil, že minimálne počas jednej hrany hodinového signálu bude stabilný a dáta sa bezpečne zapíšu do pamäte.

Samozrejme, rátal som s tým, že s takouto implementáciou pravdepodobne vytváram zbytočné stavy a „mrhám“ časom. Avšak treba tiež zvážiť, koľko hodinových cyklov *minimálne* zaberie jeden prenos na rozhraní SPI. Počas jedného prenosu prejde automat v entite `SPI_SLAVE` celkovo cez 31 stavov. Ak by som teda uvažoval krajný prípad, kedy nepotrebuje čakať na zmeny signálu `SCLK`, potrebuje na 1 prenos minimálne 31 hodinových cyklov. V tomto automate potrebujem minimálne 4 stavy, dva pre signál `BUSY` a dva pre inkrementovanie čítačov (jeden počas odosielania a jeden na konci). To znamená, že k dispozícii som mal až 27 stavov, ktoré som mohol dať do tohto automatu a stále by bol schopný pracovať s entitou `SPI_SLAVE`. Preto som nevidel dôvod na to, aby som do pamäte zapisoval s minimálnymi možnými časovými charakteristikami.

# Kapitola 6
# Testovanie

*V tejto kapitole opíšem spôsoby a výsledky testovania realizovaného riešenia. Najprv sa budem venovať simulácii jednotlivých komponent a celého radiča. Nakoniec opíšem testovanie radiča naprogramovaného do prípravku Basys3, ku ktorému sa pripojím cez rozhranie SPI mikropočítačom Raspberry Pi 3 A+.*

## 6.1 Simulácie

V tejto podkapitole sa budem zaoberať simuláciami jednotlivých komponent realizovaného riešenia. Všetky simulácie som realizoval ako *post-syntézne*. Okrem simulácie entity `TOP`, teda celkového riešenia, opíšem aj simulácie jej jednotlivých komponent, pričom postupovať budem v rovnakom poradí ako tomu bolo v predchádzajúcich kapitolách o návrhu a realizácii. Pomocou simulácií som overil, do akej miery boli realizáciou splnené požiadavky, ktoré som pre jednotlivé komponenty stanovil.

Pri simulácii som využil 2 prístupy k overeniu správnosti riešenia. Jedným z nich je analýza priebehu signálov pomocou grafického zobrazenia, ktoré ponúka simulátor. Týmto spôsobom som „manuálne“ overil napr. priebehy riadiacich signálov LCD a ich časové charakteristiky. Druhý spôsob, ktorý som použil je overenie správnosti pomocou príkazu `assert`. Pomocou neho je možné naprogramovať zdrojový kód simulácie tak, aby automaticky kontroloval správnosť zvolenej podmienky. Hoci tento spôsob ponúka jednoduché simulovanie, bez nutnosti analýzy simulovaných priebehov signálov, v niektorých prípadoch je pomerne ťažké a hlavne zdĺhavé ho nastaviť správne. V takých prípadoch je zvyčajne rýchlejšie a jednoduchšie „manuálne“ analyzovať priebehy signálov. Preto som tento spôsob využil iba pri simulácii vyrovnávacej pamäte.

### 6.1.1 Simulácia radiča LCD

Hlavnou požiadavkou na radič LCD bolo vygenerovať priebeh riadiacich signálov tak, aby čo najviac imitovali analyzované priebehy signálov z radiča typu MSM6255. Okrem toho je jeho úlohou aj ovládať pamäť, pomocou signálu `TAIL_CHANGE`, ktorým si vyžiada nový obrázok a dáta z pamäti len presmeruje na výstup na LCD.

Nakoľko riadenie LCD je závislé striktne len do časovania, jediné jeho vstupy sú `CLK`, `RESET` a `RD_DATA` z pamäte. Vygenerovanie vstupov pre jeho simuláciu bolo teda celkom

priamočiare. Tak ako v každej simulácii som vygeneroval priebeh hodinového signálu a na začiatku pulz signálu RESET. Vstupné dáta, ktoré v kompletnom riešení prichádzajú z pamäte som vygeneroval v závislosti od adresy (ktorú generuje samotný radič) tak, aby som mohol overiť správnu funkčnosť preposielania týchto dát. Spôsob generovania dát je k dispozícii v priloženom výpise kódu C.1.

Overenie správnosti výsledkov simulácie som urobil pomocou analýzy grafického výstupu priebehov signálov. Tak ako pri meraní, je nutné jednotlivé signály analyzovať pri rozdielnom priblížení. Na priloženom obr. C.1 je možné vidieť celkový priebeh signálov, ktorý vznikne pri odoslaní 2 celých obrázkov na LCD. Pri takomto priblížení som overil, že celkový priebeh signálov FRP a FRMB je správny. Okrem toho je možné vidieť aj správnu funkciu signálu TAIL_CHANGE, ktorý sa aktivuje na konci odosielania každého obrázku.

Na obr. C.2 je zobrazený priebeh signálov pri odosielaní celého prvého riadku obrázku. Overil som pomocou neho správny priebeh signálov LIP a CLP.

Pri ešte väčšom priblížení na obr. C.3 je možné vidieť priebeh dáta na výstupe LCD_DATA. Overil som teda, že k zmene dát dochádza pri vzostupnej hrane signálu CLP a pri jeho zostupnej hrane sú stabilné. K ich zmene dochádza síce s miernym časovým omeškaním 10 ns, ale to nie je problémom vzhľadom k celkovej perióde signálu CLP. Toto je spôsobené samozrejme tým, že k preklopeniu dát na výstup registra dôjde až v nasledujúcom hodinovom takte po aktivácii príslušného signálu, ktorý to povolí.

Nakoniec som overil aj zmeny hodnôt stĺpcových a riadkových čítačov pri ukončení odosielania jedného obrázku. Ako je viditeľné na obr. C.3, obe sa správne nastavili do 0.

Po kontrole správnosti priebehu signálov som ešte skontroloval jednotlivé časové charakteristiky podľa tab. 3.8. Niektoré časové charakteristiky si vyžadovali mierne upraviť časovanie v riadiacom automate radiča. Po úprave boli ale všetky časové charakteristiky, s jednou výnimkou, rovnaké ako som nameral v analýze a teda som splnil požiadavku o rovnakom priebehu signálov ako v originálnom riešení.

Výnimkou je signál FRMB. Pre ten som síce nestanovil žiadne časové požiadavky v zmienenej tabuľke, avšak bolo to preto, lebo k jeho zmene dochádza vždy spolu s deaktiváciou signálu LIP. V simulácii sa ale ukázalo, že v mojom riešení dôjde k jeho zmene s omeškaním 10ns. Toto omeškanie vzniklo kvôli tomu, že samotná zmena signálu na výstupe prebehne až následujúci hodinový cyklus po vyslaní signálu TAIL_CHANGE, nakoľko hlavný radič LCD aj radič invertovania signálov sú obe realizované ako stavové automaty typu Moore. Vzhľadom k tomu, že v analýze som nezistil žiadne časové požiadavky medzi signálmi LIP a FRMB na vstupoch spoločných a segmentových radičov som sa rozhodol tento stav ponechať pre zachovanie jednoduchšej implementácie.

### 6.1.2 Simulácia vyrovnávacej pamäte

Vyrovnávacia pamäť má za úlohu nielen uložiť obrazové dáta, ale aj riadiť ukazovatele do pamäti. Jej jediným výstupom podľa návrhu radiča sú iba dáta pre radič LCD. Rozhodol som sa samotné hodnoty ukazovateľov a signály EMPTY a FULL vyviesť ako výstupy celého radiča, pretože poskytujú veľmi dobrý prehľad o tom, čo sa deje „vnútri“ radiča. Ich hlavný účel bude pri testovaní pomocou nahrania do prípravku Basys3 v následujúcej podkapitole, každopádne využijem ich aj pri simulácii tejto entity a následnej simulácii celého radiča.

Chovanie tejto komponenty je veľmi odlišné od ostatných komponent. Zatiaľ čo ich výstupy sú riadené automatmi a je teda potrebné sledovať ich *priebehy*, výstupy vyrovnávacej pamäte sú najmä dáta a je potrebné sledovať ich *hodnoty*. Z toho dôvodu som simuláciu tejto entity kompletne automatizoval pomocou príkazu `assert`.

Pre automatickú simuláciu som dáta potrebné ku kontrole a pre zápis uložil do 3 externých súborov. Ide o `fit-logo.txt`, ktorý obsahuje dáta základného obrázku v pamäti, pre jeho kontrolu. Ďalej sú to súbory `test-wr-data.txt` a `test-rd-data.txt`, ktoré obsahujú náhodne vygenerované binárne dáta. Tie slúžia pre kontrolu správneho zápisu do pamäte. Nakoľko jazyk VHDL neumožňuje použitie relatívnej cesty, je potrebné v zdrojovom súbore pre túto simuláciu, ktorá je v súbore `tb-ring-buffer.vhd`, prepísať cestu k týmto súborom.

V simulácii som overil nielen dáta, ale aj chovanie ukazovateľov a príznakov stavu pamäte. Hneď po vygenerovaní pulzu signálu `RESET` overím, či sa jednotlivé signály nastavili správne. Toto overenie je vo výpise kódu 6.1. Ďalšie overenia týchto signálov priebehajú rovnako, iba s odlišnými hodnotami.

```
1 assert TAIL_OUT="00" and HEAD_OUT="01" and EMPTY_OUT='1' and FULL_OUT='0'
2     report  "Default setting corrupted! TAIL is " & to_hstring(TAIL_OUT) &
3             ", should be 0. HEAD is " & to_hstring(HEAD_OUT) &
4             ", should be 1. EMPTY is " & std_logic'image(EMPTY_OUT) &
5             ", sould be 1. FULL is " & std_logic'image(FULL_OUT) &
6             ", should be 0." severity failure;
```

**Výpis kódu 6.1** Automatická kontrola hodnoty jednotlivých výstupov vyrovnávacej pamäte.

Potom v simulácii v cykle prečítam dáta celého základného obrázku. Tie vždy skontrolujem oproti dátam uloženým v súbore `fit-logo.txt`. Táto kontrola prebieha rovnako ako v predchádzajúom prípade automatizovane, len s upravenou pomienkou pre dáta. Takto som overil, že základný obrázok je správne uložený do pamäte.

Potom som podobným spôsobom v cykle načítal náhodné dáta z ďalšieho súboru a tie som uložil na pozíciu 1 v pamäti. Veľkosť dát bola 61440 bitov, teda zapísal som celý priestor vyhradený pre jeden obrázok. Tieto dáta som neskôr cyklicky prečítal a overil som, že prečítané dáta z pamäte sú rovnaké ako v súbore. Takto som teda overil správnu funkciu zápisu dát do pamäte.

Okrem toho som podrobne otestoval aj chovanie ukazovateľov do pamäte a príznakov zaplnenia v závislosti od signálov `TAIL_CHANGE` a `HEAD_CHANGE`. Testoval som pokusy o inkrementáciu `TAIL` pri prázdnej pamäti, resp. pokusy o inkrementáciu `HEAD` pri plnej pamäti a vždy som overil stav jednotlivých signálov. Potom som pamäť uvoľnil a otestoval som aj, či inkrementácia ukazovateľov funguje aj po neúspešnom pokuse. Otestoval tiež, že oba ukazovatele sa pri pretečení nastavia do 1, nie do 0 a nedôjde tak k prepísaniu základného obrázka. Všetky testy prebehly vporiadku.

### 6.1.3 Simulácia radiča SPI

Radič SPI musí byť podľa požiadaviek z návrhu schopný prijať prenosy pomocou rozhrania SPI a prijaté dáta následne zapísať do pamäte. Na výstupe má teda adresu, dáta a riadiace signály pre vyrovnávaciu pamäť. Okrem toho obsahuje aj výstup `SPI_MISO`. Všetky tieto signály sú ovládané pomocou stavového automatu, ich priebehy som teda overil analýzou vygenerovaného priebehu signálov.

Vstupmi tohto radiča sú jednotlivé linky rozhrania SPI a samostatný synchronizačný signál z nadradeného PC. Tieto signály som na začiatku inicializoval do 0 a následne som urobil pulz singálu `RESET`. Aby som overil celkovú funkčnosť tohto radiča, musel som sa v simulácii dostať až na koniec jedného obrázku. Vygeneroval som teda priebehy SPI signálov tak aby imitovali odoslanie jedného obrázku. Na konci som ešte vygeneroval niekoľko rôznych samostatných prenosov pre overenie jednotlivých vlastností.

Spôsob vygenerovania priebehu jedného prenosu na zbernici SPI je k dispozícii vo výpise kódu 6.2. Pri tom som rešpektoval nastavenie radiča SPI v móde *CPOL=0* a *CPHA=1*. Dáta sa teda menia až pri hrane signálu `SCLK`, ktorý som na začiatku inicializoval do 0. Periódu hodinového signálu rozhrania SPI som nastavil na 150ns.

```
1  SPI_SCLK <= '0';
2  SPI_SS <= '0';
3  for i in 7 downto 0 loop
4      wait for TSCLK;
5      SPI_SCLK <= not SPI_SCLK;
6      SPI_MOSI <= mosi_reg(i);
7      wait for TSCLK;
8      SPI_SCLK <= not SPI_SCLK;
9  end loop;
10 wait for TSCLK;
11 SPI_SS <= '1';
```

**Výpis kódu 6.2** Generovanie priebehu signálov na zbernici SPI pre jeden prenos

Následne som radič simuloval a urobil analýzu priebehu jeho výstupných signálov. Na obr. C.4 je znázornený priebeh jednotlivých signálov počas odosielania celkovo 4 prenosov. Tie prebiehajú pri konci odosielania celého obrázka.

V prvom rade je možné overiť, že funguje loopback. To je zreteľne vidno na výstupe radiča `SPI_MISO`, ktorý s oneskorením jedného prenosu imituje priebeh signálu `SPI_MOSI`.

Ďalej je možné vidieť, že radič SPI funguje správne nezávisle od toho, či dôjde k deaktivácii signálu `SPI_SS` medzi jednotlivými prenosmi. V čase 9 944 500 ns je možné vidieť situáciu kedy medzi jednotlivými prenosmi nedošlo k deaktivácii tohto signálu, napriek tomu radič zapisuje dáta do pamäte a inkrementuje čítače.

Na obr. C.5 je možné vidieť situáciu po prenesení celého obrázku do radiča. Overil som, že radič správne vyšle signál `HEAD_CHANGE` a riadkový i stĺpcový čítač sa nastavia späť do 0. Signál `HEAD_CHANGE` sa síce vyšle spolu s deaktiváciou signálu `WR_EN`, avšak samotný ukazovateľ do pamäte sa aktualizuje až v následujúcom hodinovom takte, teda nehrozí porušenie presahu adresy.

Na obr. C.6 je možné vidieť výstupy radiča po synchronizácii. Synchronizácia v tomto prípade znamená najmä to, že sa riadkový a stĺpcový čítač nastavia späť do nuly, čo sa naozaj stalo. Na rovnakom obr. je tiež možno zhliadnuť overenie zápisu dát do pamäte a dodržania predstihov a presahov tak ako som to opísal v predchádzajúcej kapitole.

### 6.1.4 Simulácia celého riešenia

Celé riešenie radiča som otestoval samostatnou simuláciou. Pre vstupy radiča, ktorými su najmä signály rozhrania SPI som využil generovanie priebehov z predchádzajúcej kapitoly, ktoré som prípadne upravil.

Simulované priebehy signálov je možné vidieť na obr. C.7. Jednotlivé priebehy výstupov radiča som kontroloval manuálne a sústredil som sa hlavne na celkovú funkčnosť, nakoľko presné charakteristiky som overil v predchádzajúcich simuláciách.

Overil som teda najmä správny priebeh riadiacich signálov pre LCD a správne chovanie ukazovateľov v pamäti a príznakov stavu zaplnenosti. Ďalej som sa sústredil aj na to, či začal radič LCD ako druhý obrázok odosielať ten, ktorý som „odoslal“ pomocou rozhrania SPI. Všetky signály mali korektný priebeh. Simulácia teda ukázala, že radič LCD by mal fungovať správne.

## 6.2 Testovacie aplikácie pre Raspberry Pi

V tejto podkapitole opíšem otestovanie radiča pomocou LCD panela, prípravku Basys3 a mikropočítača Raspberry Pi 3 A+, ktoré navzájom prepojím. Na mikropočítači vytvorím niekoľko testovacích aplikácií, ktoré budú odosielať do radiča dáta rozhraním SPI a tak otestujú jeho funkčnosť.

Samotný radič som po syntéze implementoval a nahral na školskú vývojovú dosku Basys3, ktorá je osadená FPGA čipom Artix-7 od spoločnosti Xilinx. Vstup `RESET` som priradil na jedno z tlačítiek. Testovacie výstupy z pamäte som priradil na LED, ktorých má táto doska k dispozícii až 16.

Ďalej som musel dosku s radičom prepojiť s mikropočítačom a LCD panelom z písacieho stroja. Prepojenie s mikropočítačom pomocou rozhrania SPI bolo veľmi jednoduché, ako som už spomínal v návrhu, je totiž možné ich prepojiť priamo. Problém nastal pri prepojení s LCD panelom. Z neho vedie 16-žilový plochý kábel s rasterom 1,25mm, pre ktorý som mal mierne problémy zohnať konektor. Konektory, ktoré sa mi nakoniec podarilo objednať majú veľmi tenké piny, ktoré nie sú kompatibilné s výstupmi dosky.

Jedným z riešení tohto problému mohlo byť vytvorenie plošného spoja, ktorý som ale nechcel vytvárať, vzhľadom k tomu, že radič LCD je len jeden z radičov, ktoré plánujem vytvoriť pre celý písací stroj. Do úvahy pripadalo aj napájkovanie káblov na výstupy konektora priamo. Nakoniec som to vyriešil tak, že som káble na konektor pripojil priamo. Samotný konektor je pre tieto piny síce príliš veľký, ale keď sa zapoja všetky naraz, zmestia sa vedľa seba len veľmi tesne a tým pádom na konektore držia. Toto riešenie, hoci nie úplne elegantné je teda funkčné a k otestovaniu radiča bolo postačujúce.

Výstupy dosky Basys3 som na LCD panel nenapojil priamo. Nakoľko tieto výstupy môžu mať maximálnu úroveň napätia 3.3V a LCD panel potrebuje signály na napäťovej úrovni až 5V, musel som medzi ne vložiť súčiastku, ktorá konvertuje napätie logickej úrovne.

Nakoniec som ešte potreboval zapojiť napájanie LCD, teda napätia $V_x$, ktoré som opísal v analýze. Nakoľko riešenie zdroja napätia plánujem riešiť až po realizácii všetkých radičov pre písací stroj, rozhodol som sa pre účely otestovania prototypu využiť napäťové linky, ktoré poskytuje samotný písací stroj a tie som napojil na LCD panel. Celkové zapjenie prototypu je možné vidieť na obr. C.8.

Po úspešnom zapojení som najprv otestoval základnú funkciu radiča LCD. Naprogramoval som teda moje riešenie do prípravku Basys3 a potom spustil písací stroj pre napájanie. Výsledkom bolo zobrazenie základného obrázka na LCD, čo je možno vidieť na obr. C.9. Overil som teda, že moje riešenie riadenia LCD, aj zapojenie prototypu je naozaj funkčné.

Po úspešnom zobrazení základného obrázka som mohol urobiť rozsiahlejšie testy pomocou mikropočítača Raspberry Pi 3 A+, kde som vytvoril 4 testovacie aplikácie. Pre tieto mikropočítače je dostupný operačný systém *Raspbian*, ktorý je pre nich prispôsobený. Okrem iného obsahuje aj radič rozhrania SPI pre ktoré sú vyhradené špeciálne piny v rámci GPIO rozhrania mikropočítača.

K jednoduchej obsluhe tohto radiča som zvolil implementáciu testovacích aplikácií v jazyku Python. Okrem všeobecných výhod pre jednoduché aplikácie je pre tento jazyk verejne dostupná aj knižnica *spidev*, pomocou ktorej je možné veľmi jednoducho ovládať radič SPI. Pre jednotlivé testy som vytvoril niekoľko testovacích obrázkov, pomocou ktorých je možno vizuálne overiť na LCD, že obrázky boli správne prijaté a zobrazené.

Vytvoril som 2 testy, ktoré využívajú synchronizáciu medzi odoslanými obrázkami. Prvý z nich `test_synch.py`, odosiela dáta tak, že medzi jednotlivými prenosmi deaktivuje signál $\overline{\text{SS}}$. Tento test navyše kontroluje aj dáta, ktoré prijme z radiča SPI, pričom očakáva dodržanie spätného odoslania. Druhým testom je `test_synch_continuous.py`, ktorý odosiela dáta pomocou viacslovného prenosu. Celé obrázky som teda odosielal naraz príkazom `xfer3`. Nakoľko celý obrázok bol príliš veľký, tento príkaz ho rozdelil na polovicu a oba polovice odoslal prenosom pri ktorom bol signál $\overline{\text{SS}}$ po celú dobu aktivovaný. Oba testy fungovali bez problémov, teda obrázky sa na LCD zobrazovali postupne tak ako boli odoslané a testovacia aplikácia nevykazovala chyby v spätne odosielaných dátach. Tým som otestoval skoro všetky požiadavky, ktoré som stanovil pre radič SPI a zároveň všetky požiadavky pre pamäť.

Poslednou požiadavkou, ktorú som potreboval otestovať, bola schopnosť prijímať dáta asynchrónne. Pre tento účel som vytvoril test `test_asynch.py`. Tento test je vytvorený špeciálne na to aby ukázal čo sa môže stať bez použitia synchronizácie. Nastavil som ho tak, aby z prvého obrázka odoslal iba dolnú polovicu dát a ostatné obrázky odosielal celé. Výsledkom je, že sa obrázky na LCD zobrazujú tak ako je vidieť na obr. C.10. Po resetovaní radiča LCD sa začnú zobrazovať normálne. Tým som otestoval, že asychrónny prenos funguje a je možný, pričom som zároveň ukázal aj čo sa môže stať ak sa synchronizácia úplne ignoruje.

Posledným testom som chcel ukázať, že môj radič zvláda aj rýchle prenosy. Ide o test `test_asynch_continuous.py`, pre ktorý som vytvoril samostatnú sadu obrázkov, ktoré vytvárajú na LCD animáciu. Tie sa kvôli zachovaniu veľkej rýchlosti a efektu videa odosielajú asynchrónne a pomocou viacslovného prenosu. Aj pri tomto teste fungoval radič LCD spoľahlivo a výsledky testu pri rôznych rýchlostiach prenosu sú na priloženom médiu na videách `test_result_slower.mp4` a `test_result_faster.mp4`

# Kapitola 7
# Záver

Cieľom tejto práce bolo vyvinúť radič LCD displeja, ktorý by nahradzoval pôvodne používaný radič typu MSM6255. Všetky priebežné ciele spolu s hlavným cieľom práce sa podarilo naplniť.

Prvým cieľom práce bolo analyzovať spôsob akým riadi zobrazovanie dát na LCD pôvodný radič a získané informácie overiť podrobným meraním priebehu riadiacich a datových signálov. Výsledkom mojej analýzy bol podrobný popis a vysvetlenie ako funguje komunikácia medzi LCD radičom typu MSM6255 a spoločnými a segmentovými radičmi. Výsledkom boli nielen informácie o celkovom priebehu jednotlivých signálov, ale aj o ich vzájomnej závislosti, ktoré ukázalo meranie všetkých 4 riadiacich signálov zároveň. Posledným výsledkom merania bola podrobná časová analýza priebehu signálov, ktorá umožnila presne emulovať chovanie pôvodného radiča vytvoreným prototypom.

Ďalším cieľom bolo implementovať základný radič LCD, ktorý bude schopný spoľahlivo zobrazovať statické dáta na LCD. Tento radič slúžil hlavne ako základ celého prototypu, pre overenie, že vnútorne správne pracuje so súradnicami a taktiež aj pre vyladenie technického pripojenia k LCD. Pri jeho vývoji došlo k niekoľkým problémom a to najmä v oblasti fyzického pripojenia, ale všetky problémy sa podarilo vyriešiť. Výsledkom tak bol funkčný radič, ktorý dokázal ovládať LCD a na jeho základe bolo možné postaviť kompletný prototyp.

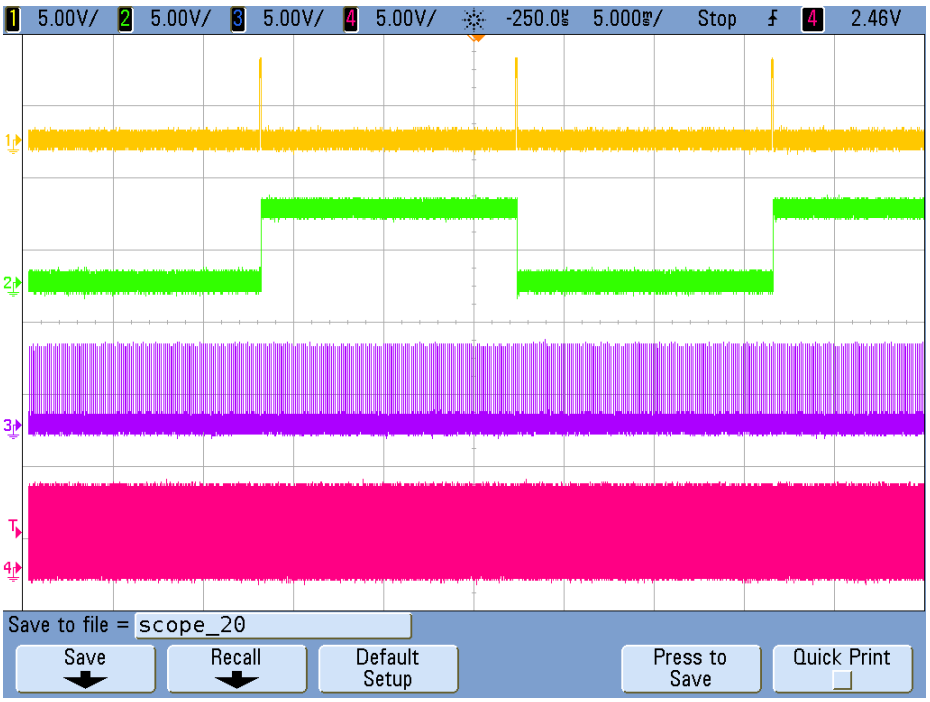
Posledným cieľom bolo na základe implmentovaného ovládania LCD skompletizovať celý radič spolu s vyrovnávacou pamäťou a schopnosťou prijímať dáta z rozhrania SPI. Spočiatku som čelil problému pri snahe vytvoriť dostatočne veľkú pamäť pre uloženie obrázkov. Ako sa ukázalo pre tak veľké pamäťové požiadavky bolo nutné využiť blokovú RAM, ktorú bolo nutné vygenerovať pomocou príslušného nástroja vo vývojovom softvéri. Ďalej došlo aj k problému pri implementovaní prijímania dát, ktoré sa správalo nedefinovaným spôsobom kvôli tomu, že som nezobral do úvahy rozdielnu hodinovú frekvenciu rozhrania SPI. Po konzultáciách sa mi podarilo oba problémy vyriešiť a dokončiť tak celý prototyp. Výsledkom je funkčný radič, ktorý dokáže na LCD zobrazovať obrázky prijímané cez rozhranie SPI, pričom disponuje vyrovnávajúcou pamäťou s kapacitou pre 4 obrázky z čoho jeden je rezervovaný pre obrázok nahraný do pamäte ako základná obrazovka.

Keďže momentálne je súčasťou prototypu na Raspberry Pi iba jednoduchá aplikácia, ktorej jediným cieľom bolo potvrdiť funkčnosť radiča z hľadiska pripojenia pomocou rozhrania SPI, v budúcnosti by bolo možné túto časť prototypu rozšíriť o plnohodnotný ovládač zavedený do jadra Linuxu, ktorý by na displeji zobrazoval napríklad terminál. Prototyp je taktiež možné rozšíriť aj v rámci radiča implementovaného na FPGA, napríklad v oblasti komunikácie radiča s mikropočítačom. Momentálne je prototyp schopný dáta prijímať a odosielať, no nebola vytvorená žiadna zložitá logika pre odosielané dáta. V budúcnosti by bolo možné túto komunikáciu zlepšiť, napr. aby mikropočítač neodosielal ďalší obrázok keď je plná vyrovnávacia pamäť.

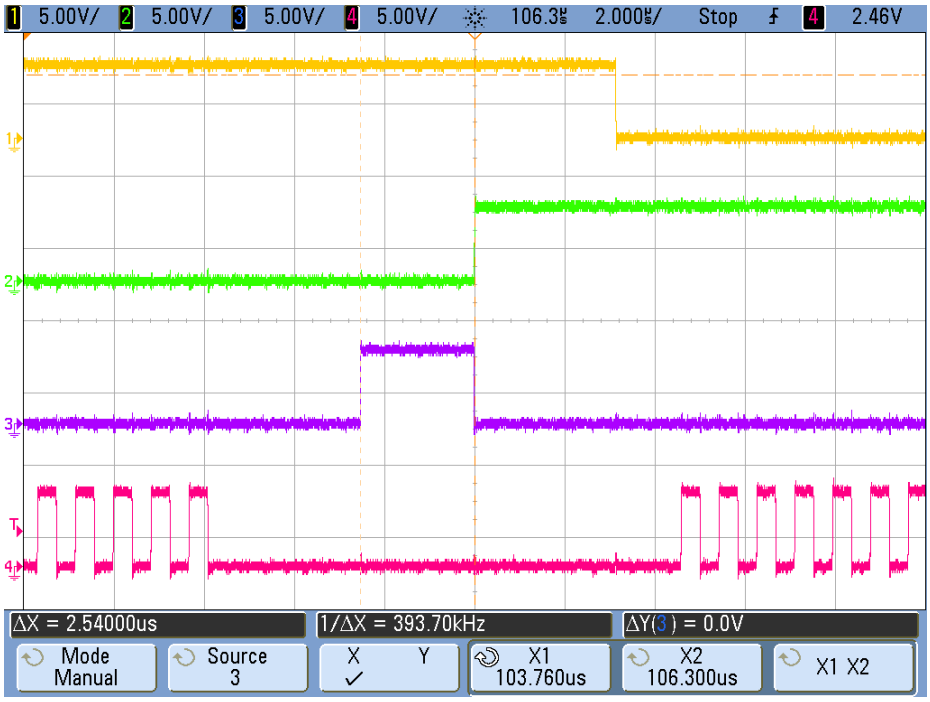
# Dodatok A

# Príloha analýzy

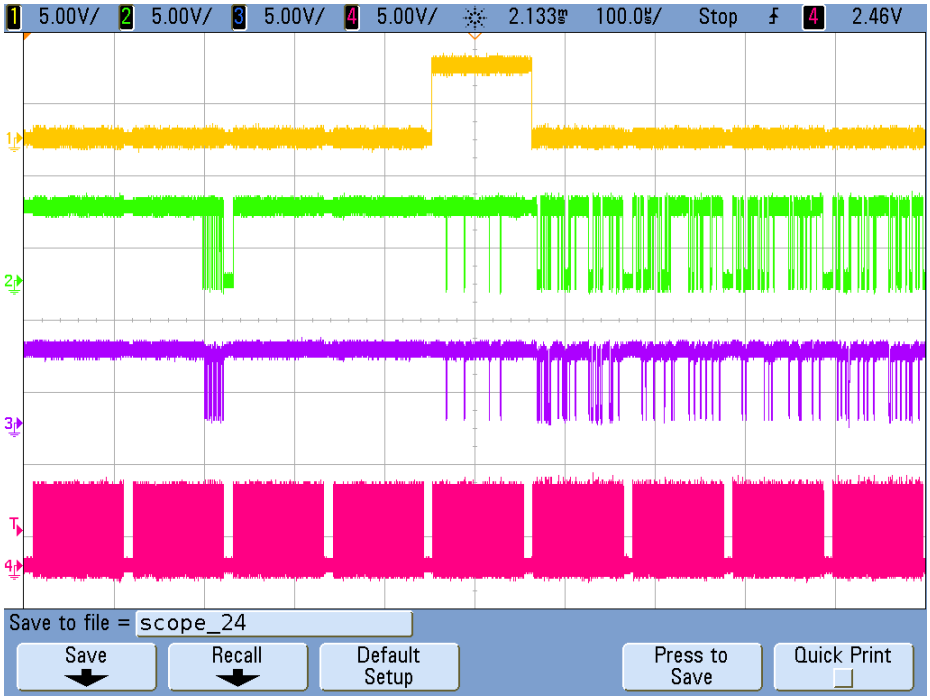
**Obr. A.1** Priebeh riadiacich signálov pri malom priblížení so zameraním na celkový priebeh signálu FRMB. Zhora dole: FRP, FRMB, LIP a CLP



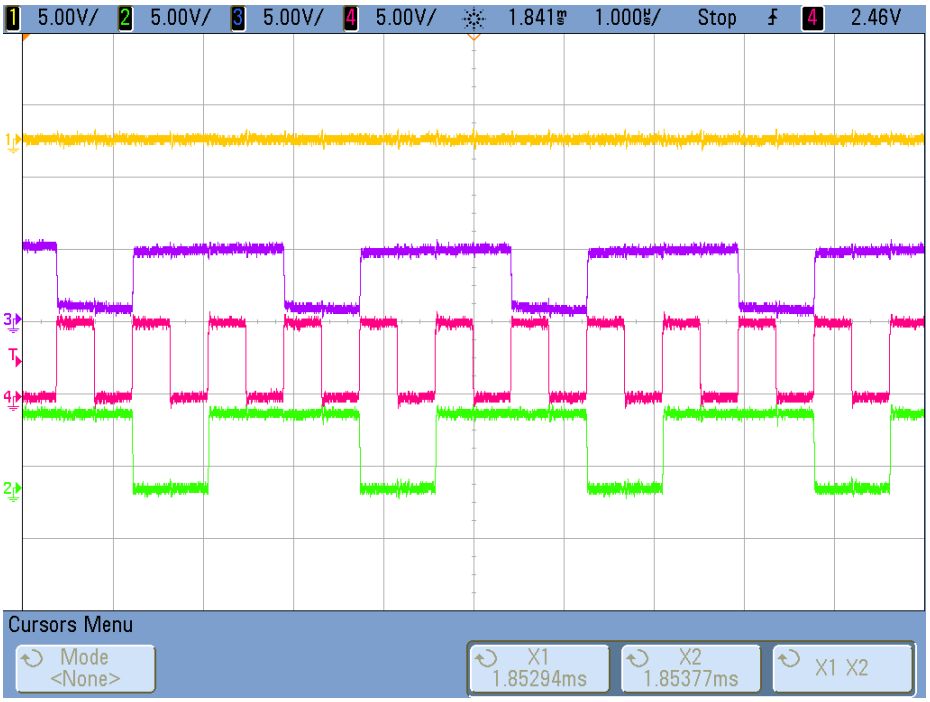
**Obr. A.2** Priebeh riadiacich signálov pri veľkom priblížení a meranie času $T_{LL}$. Zhora dole: FRP, FRMB, LIP a CLP



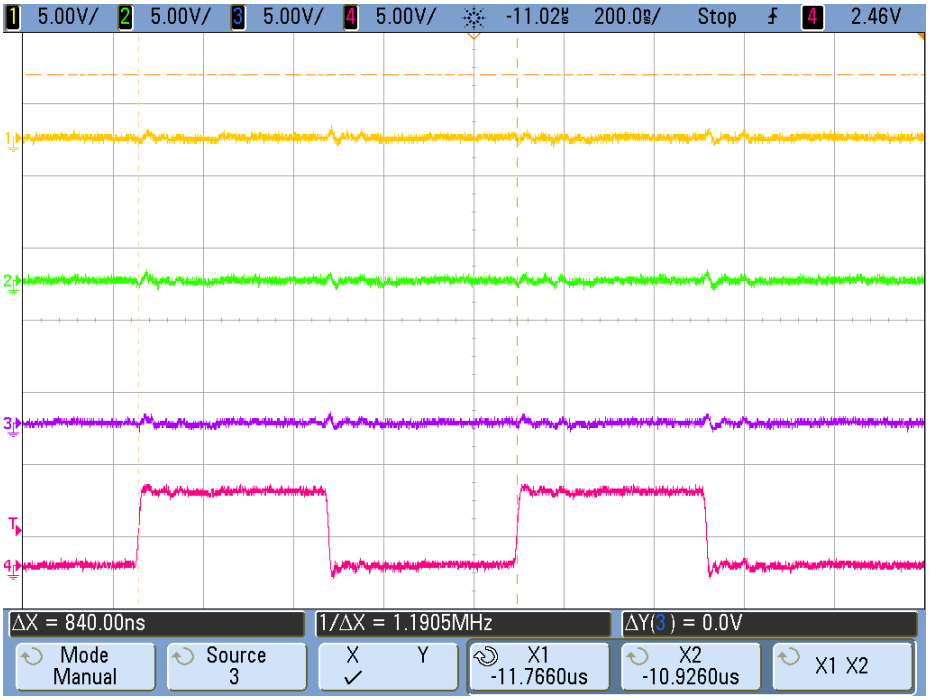

**Obr. A.3** Priebeh dát v závislosti od signálov FRP a CLP. Zhora dole: FRP, dáta, dáta, CLP



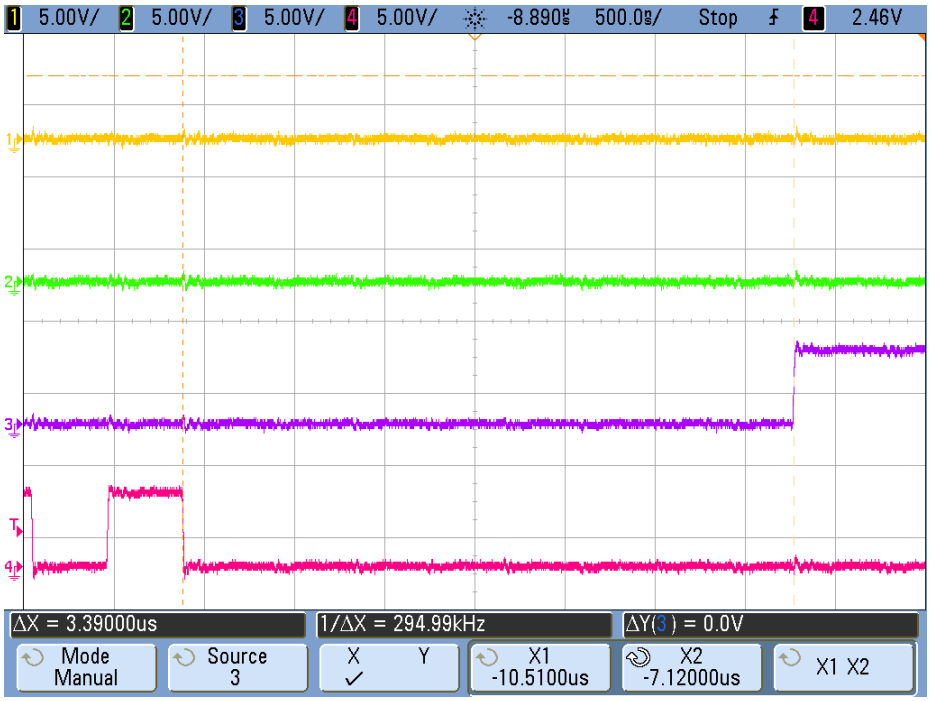
**Obr. A.4** Detailný pohľad na priebeh dát v závislosti od signálu CLP. Zhora dole: FRP, dáta, CLP, dáta



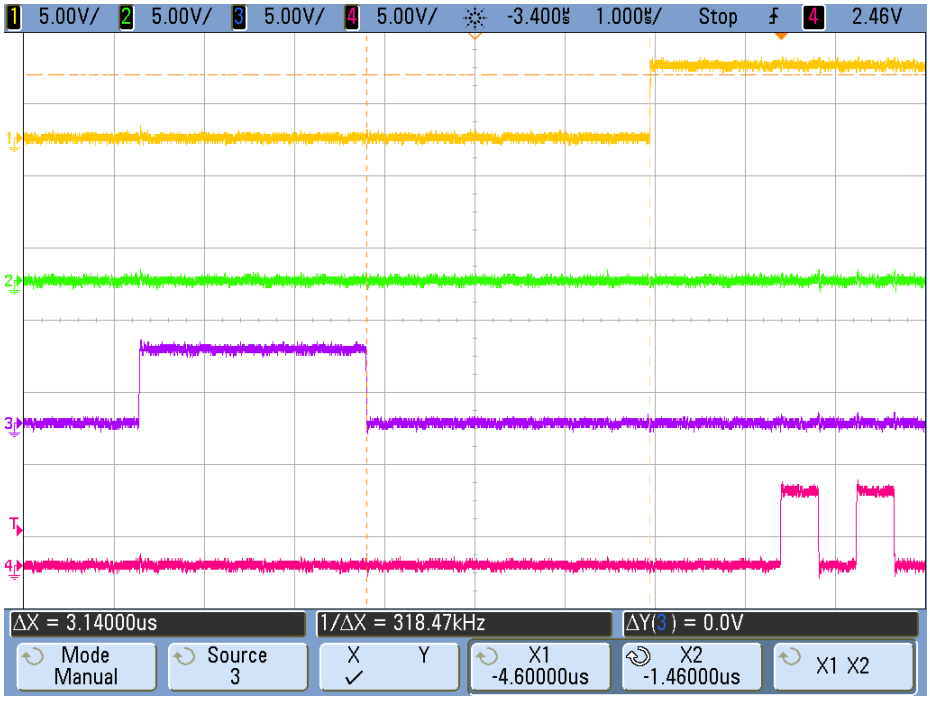

**Obr. A.5** Meranie času $T_{CLP}$



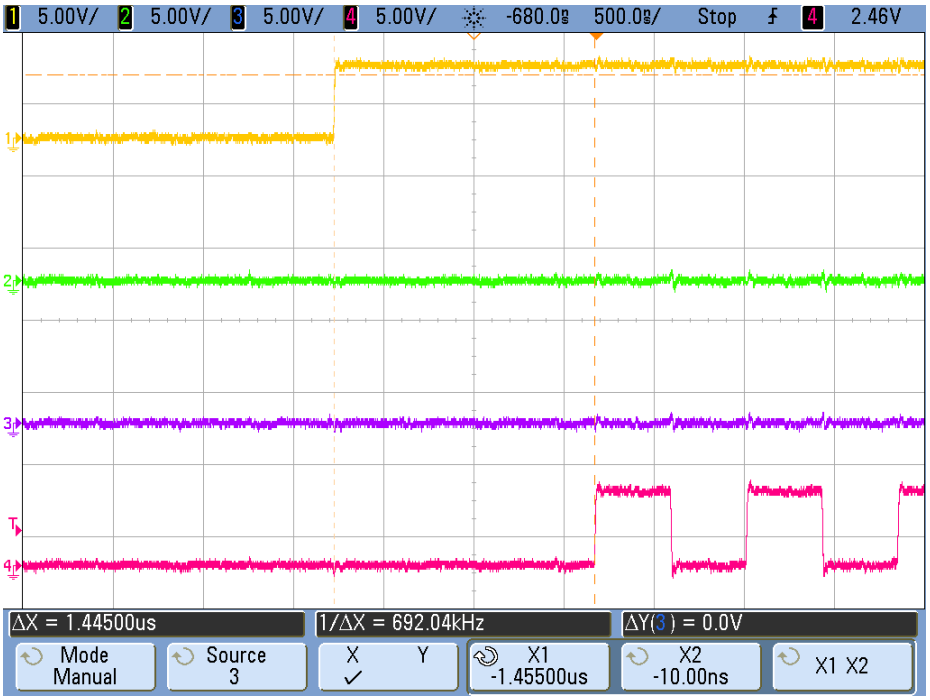
**Obr. A.6** Meranie času $T_{CL}$



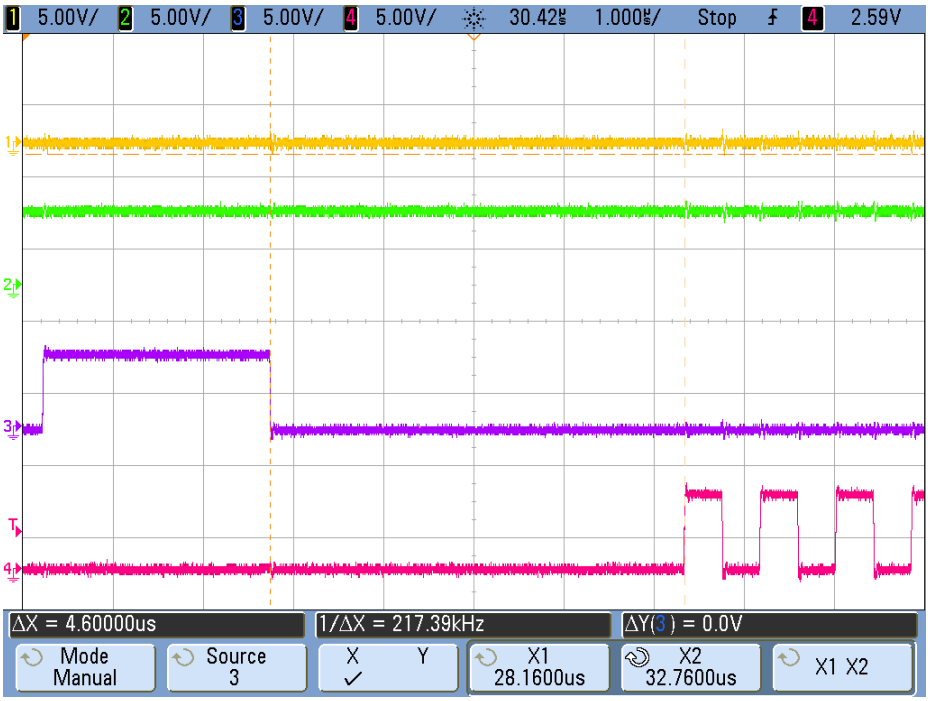

**Obr. A.7** Meranie času $T_{LF}$



**Obr. A.8** Meranie času $T_{FC}$




**Obr. A.9** Meranie času $T_{LC}$



# Dodatok B

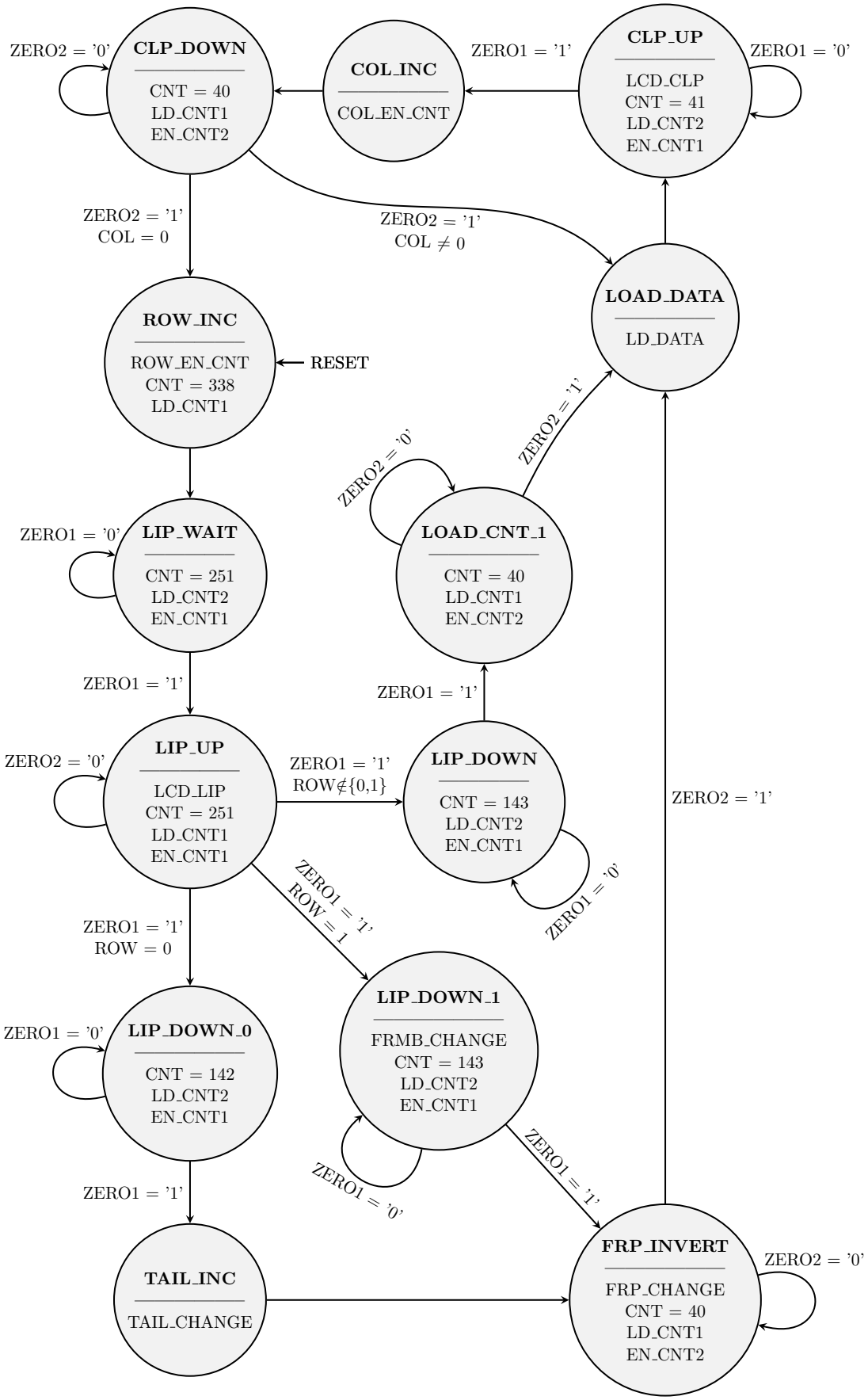
# Príloha implementácie

```
DFF: process (CLK)
begin
    if rising_edge(CLK) then
        SPI_SS_FF1     <= SPI_SS;
        SPI_SS_FF2     <= SPI_SS_FF1;
        SPI_SCLK_FF1   <= SPI_SCLK;
        SPI_SCLK_FF2   <= SPI_SCLK_FF1;
        SPI_MOSI_FF1   <= SPI_MOSI;
        SPI_MOSI_FF2   <= SPI_MOSI_FF1;
        SYNCHR_FF1     <= SYNCHR;
        SYNCHR_FF2     <= SYNCHR_FF1;
    end if;
end process DFF;
```

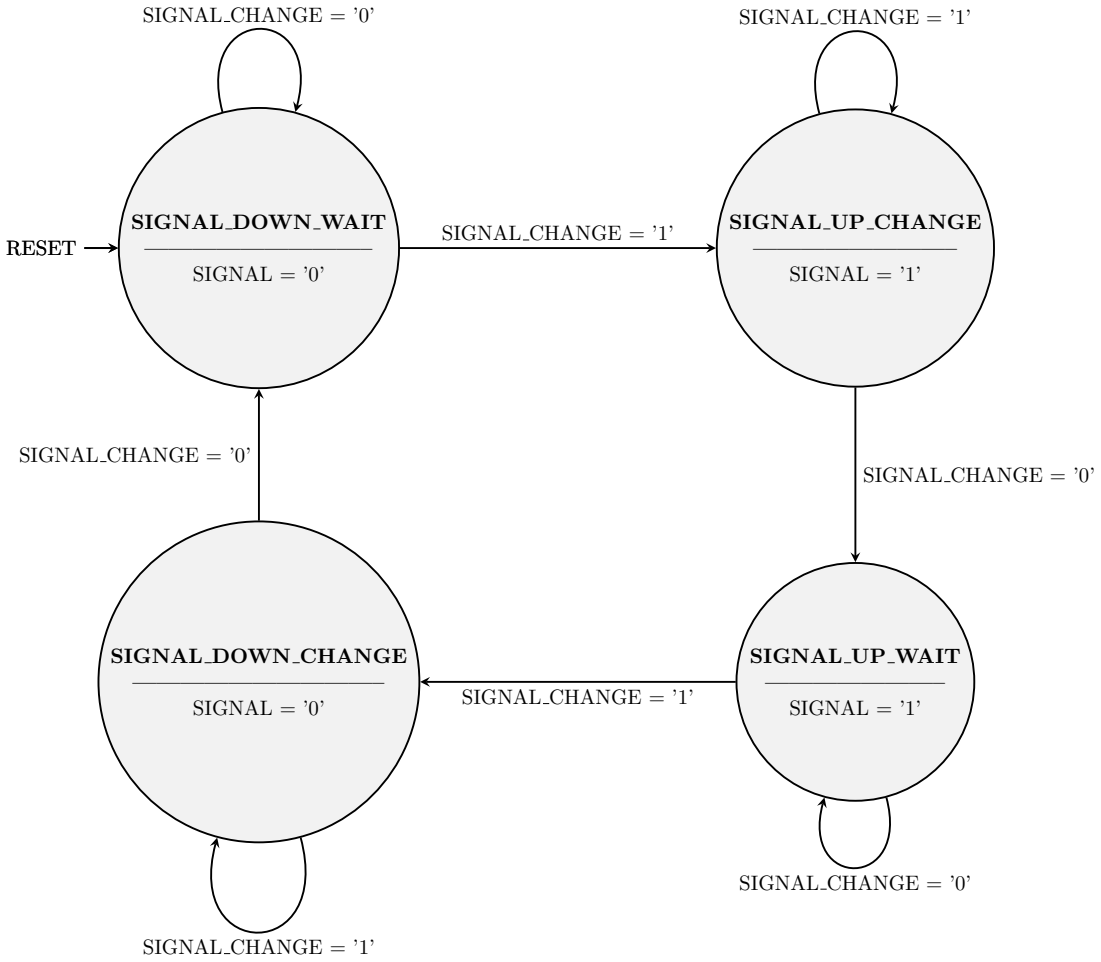
**Výpis kódu B.1** Ošetrenie metastability vstupov v entite TOP pomocou D-F/F

```
COL_COUNTER : process (CLK, RESET)
begin
    if rising_edge(CLK) then
        if RESET = '1' then
            COL <= to_unsigned(0, 7);
        elsif COL_EN_CNT = '1' then
            if COL = 119 then   --artificial overflow
                COL <= to_unsigned(0, 7);
            else
                COL <= COL + 1;
            end if;
        end if;
    end if;
end process;
```

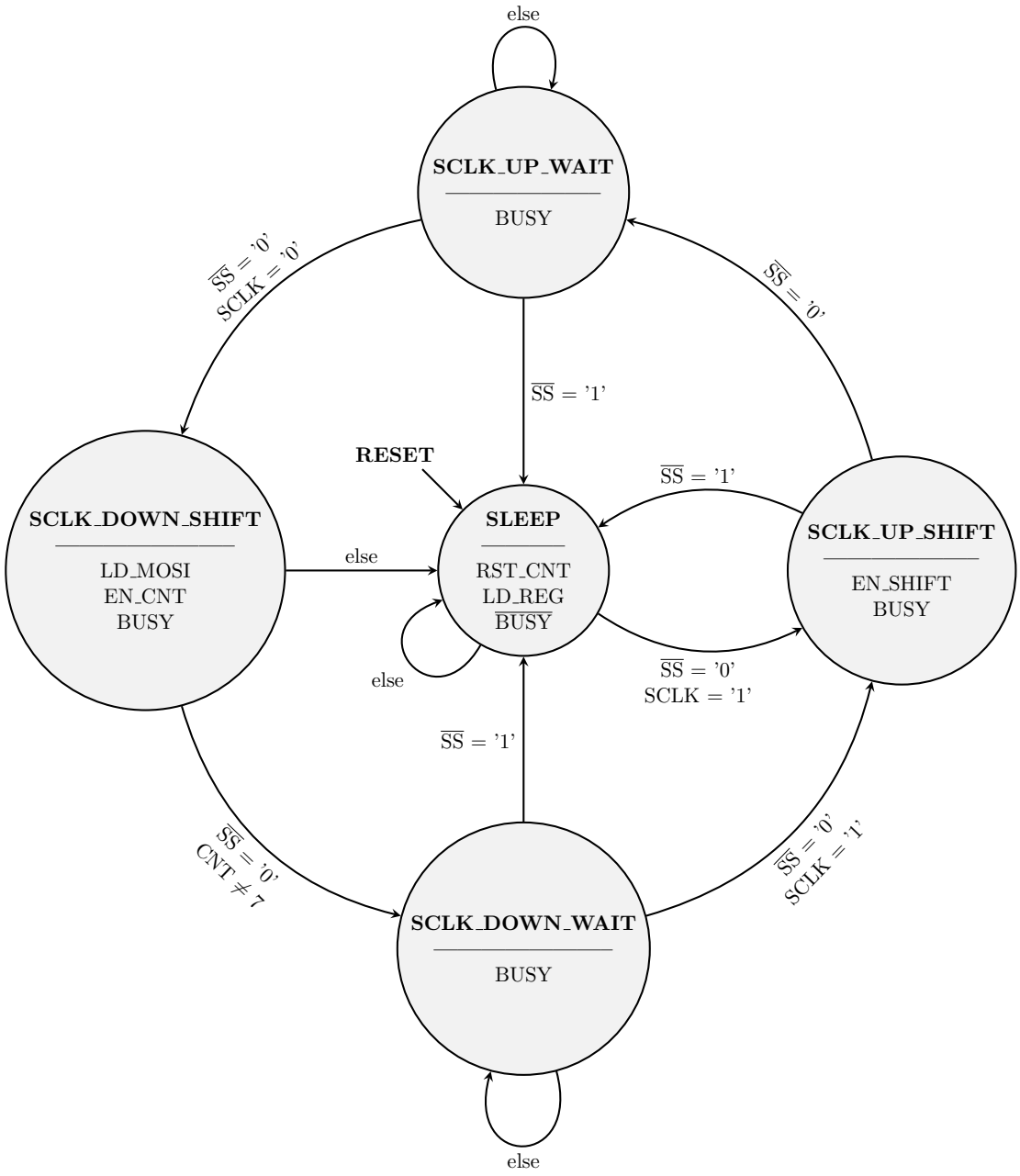
**Výpis kódu B.2** Stĺpcový čítač v radiči LCD



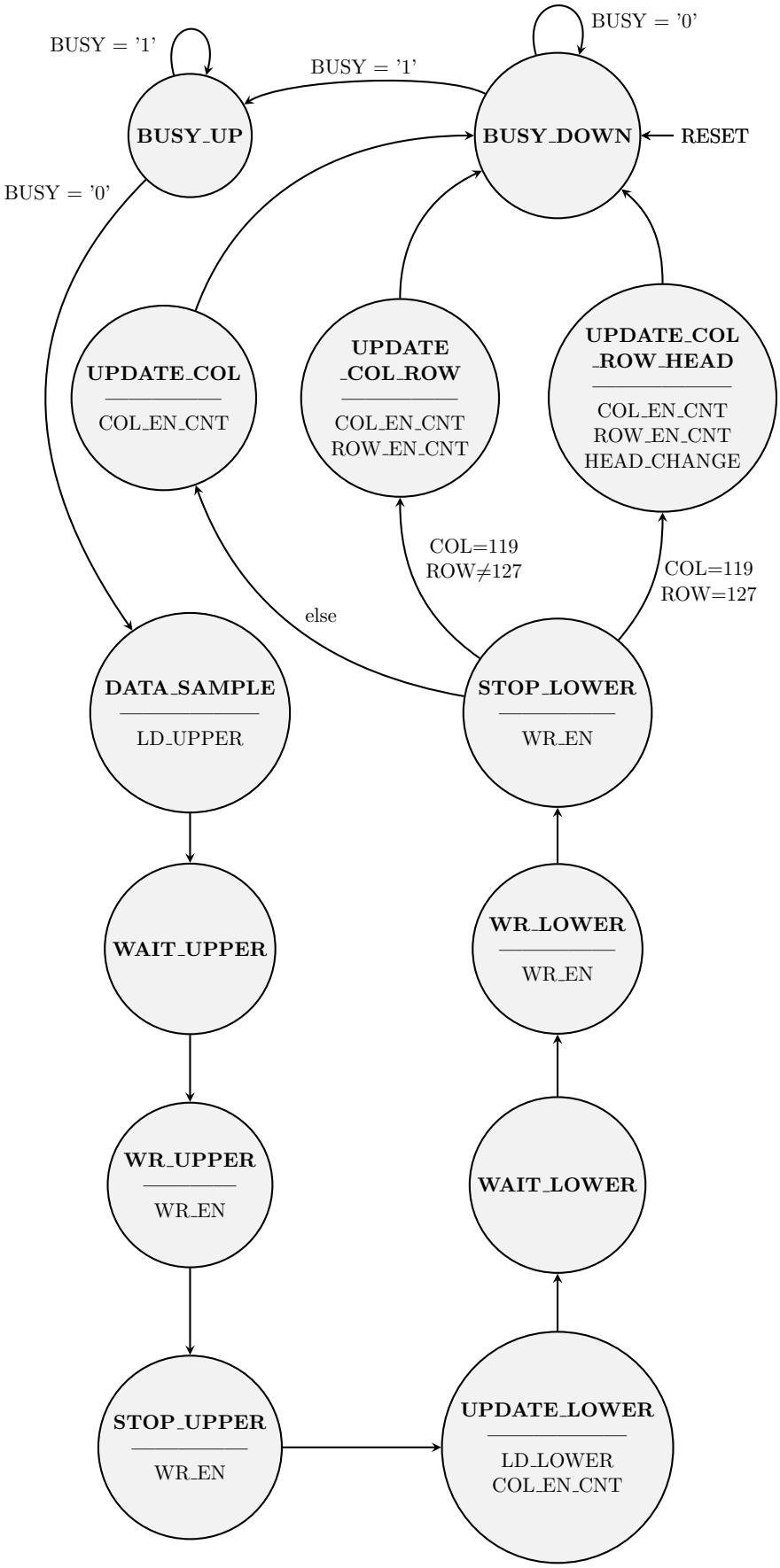

**Obr. B.1** Diagram riadiaceho automatu radiča LCD




**Obr. B.2** Diagram radiča invertovania signálov v entite `SIGNAL_CONTROLLER`




**Obr. B.3** Diagram riadiaceho automatu v entite `SPI_SLAVE`




**Obr. B.4** Diagram riadiaceho automatu zápisu dát v entite `WRITE_CONTROLLER`

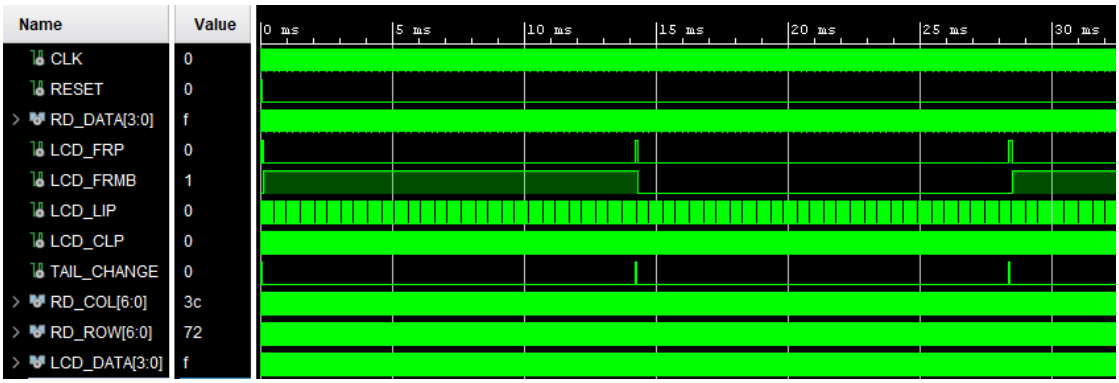
# Dodatok C

# Príloha testovania

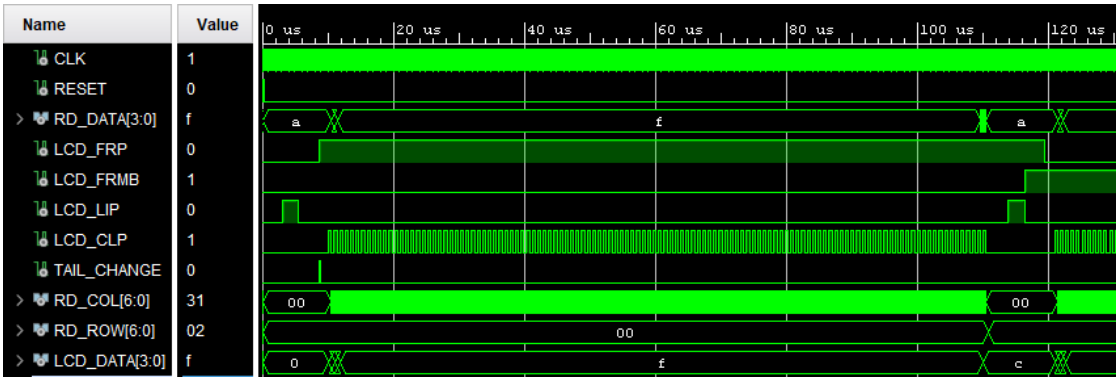
```
1  DATA_GEN : process(RD_COL, RD_ROW)
2  begin
3      if RD_COL = "1110101" then
4          RD_DATA <= x"A";
5      elsif RD_COL = "1110110" then
6          RD_DATA <= x"B";
7      elsif RD_COL = "1110111" then
8          RD_DATA <= x"C";
9      elsif RD_ROW = "0000001" then
10         RD_DATA <= x"D";
11     else
12         RD_DATA <= x"F";
13     end if;
14 end process;
```

**Výpis kódu C.1** Proces generujúci testovacie dáta pre simuláciu radiča LCD

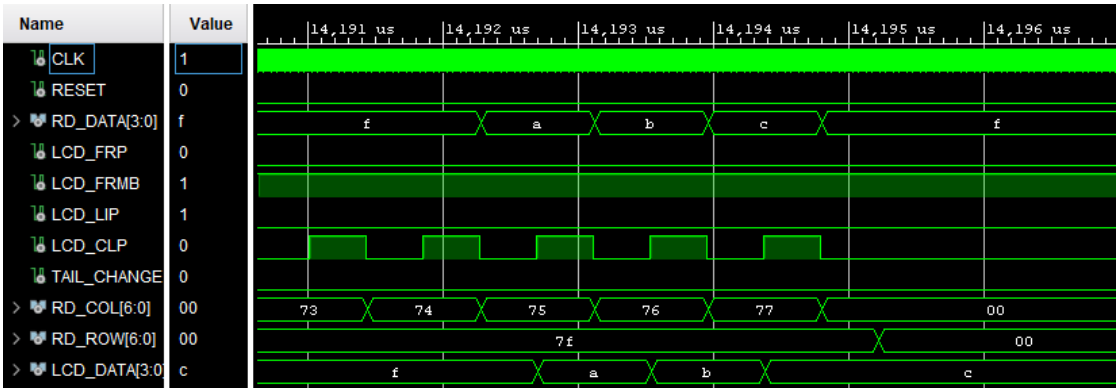
**Obr. C.1** Celkový pohľad na priebeh riadiacich signálov v simulácii radiča LCD



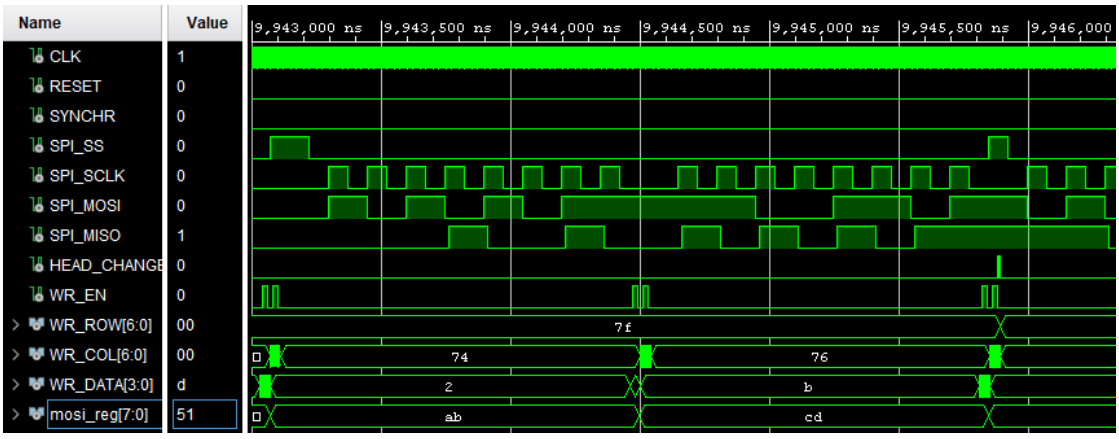
**Obr. C.2** Priebeh signálov v simulácii radiča LCD po dobu odosielania jedného riadku



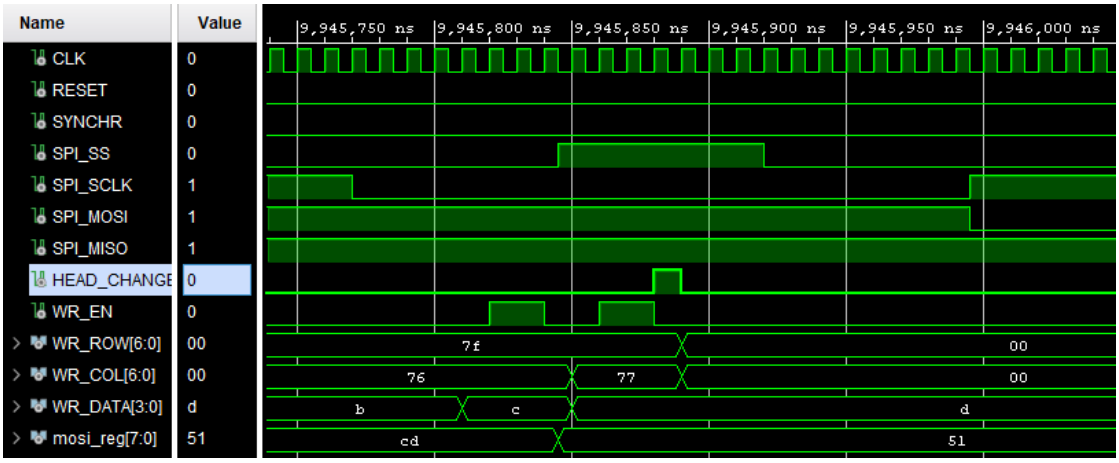

**Obr. C.3** Detailný pohľad na priebeh datových signálov a čítačov v simulácii radiča LCD



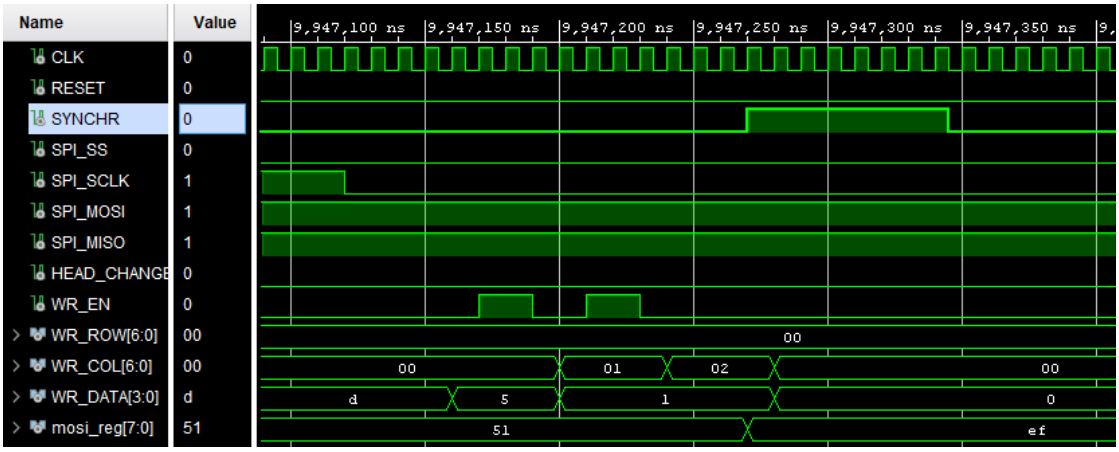

**Obr. C.4** Priebeh signálov v simulácii radiča SPI po dobu dvoch prenosov



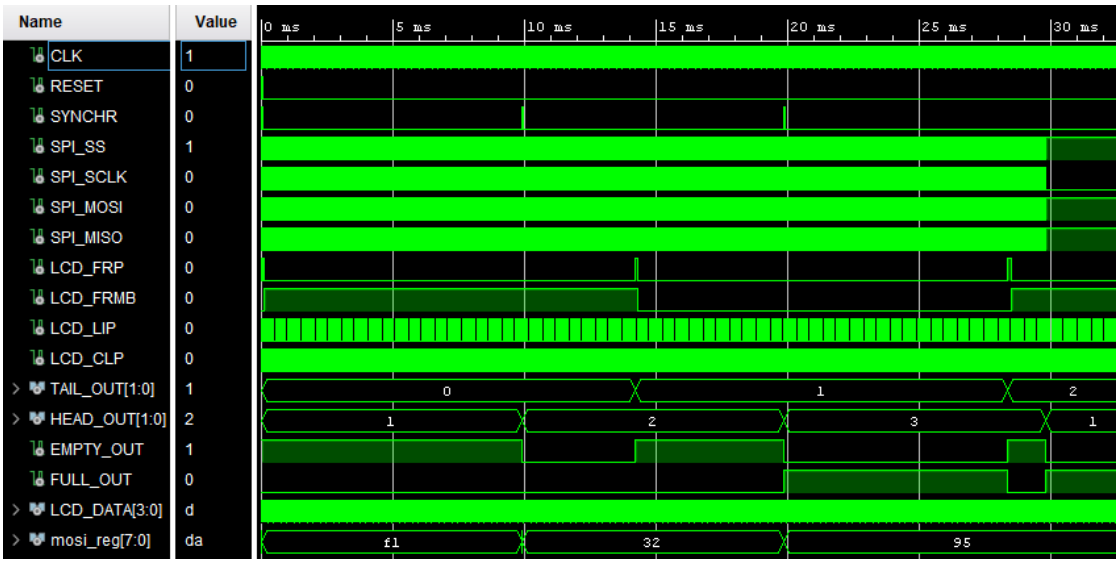
**Obr. C.5** Detailný pohľad na priebeh signálov v simulácii radiča SPI pri zápise do pamäti



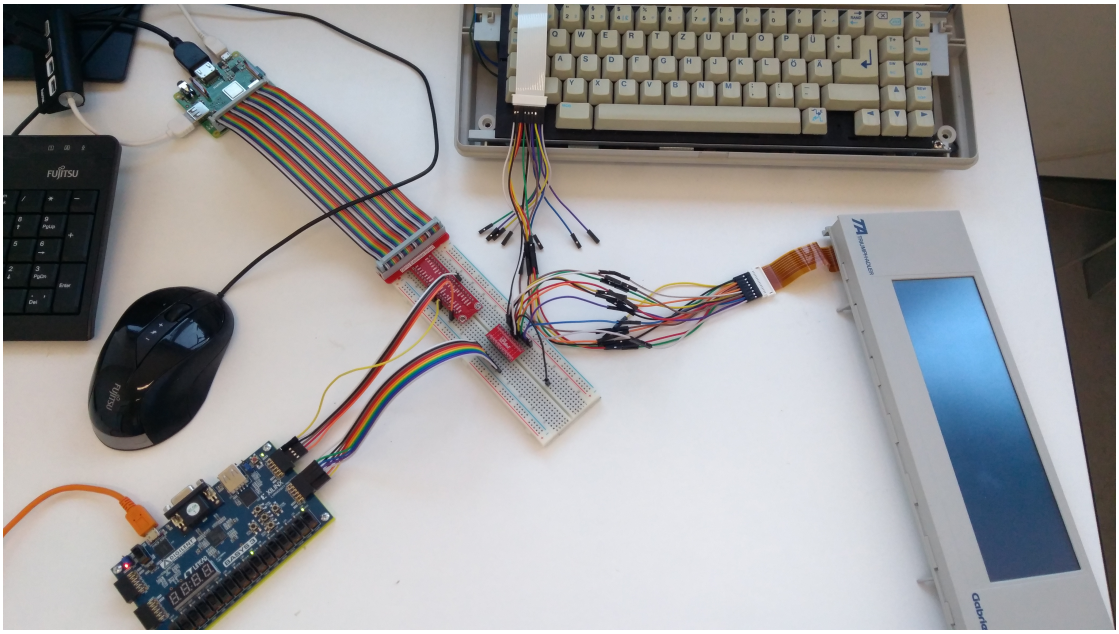

**Obr. C.6** Resetovanie čítačov pri synchronizácii v simulácii radiča SPI




**Obr. C.7** Celkový pohľad na priebeh signálov v simulácii kompletného radiča

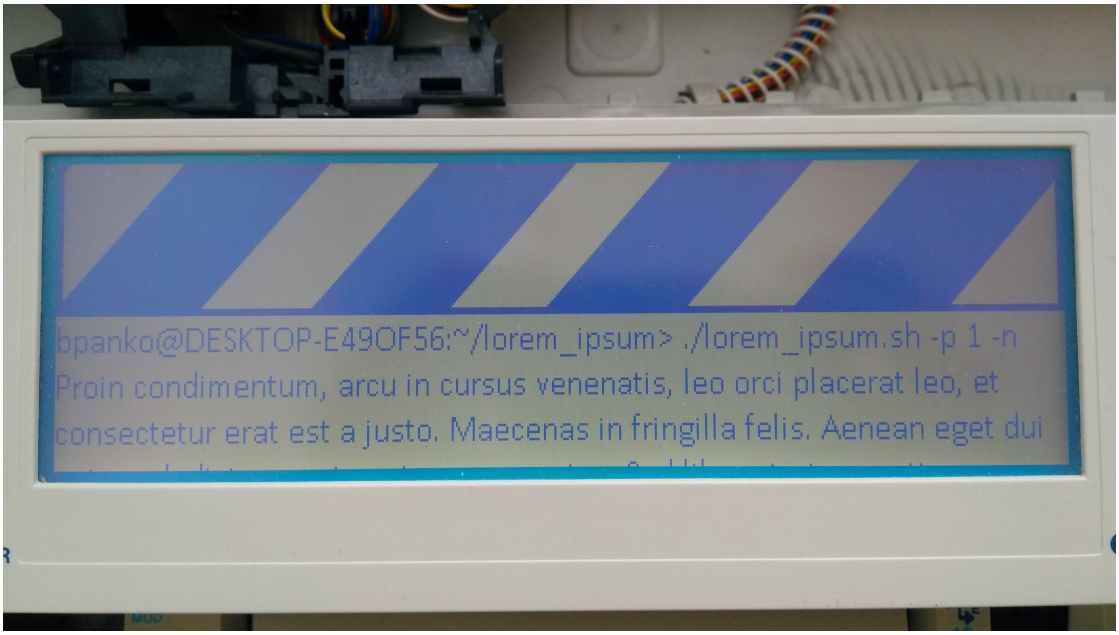


**Obr. C.8** Zapojenie testovaného prototypu radiča LCD



**Obr. C.9** Základný obrázok v pamäti radiča LCD zobrazený na LCD


**Obr. C.10** Test asynchrónneho prenosu pomocou rozhrania SPI



# Bibliografia


1. CATSOULIS, John. *Designing embedded hardware*. 2nd ed. Sebastopol: O'Reilly, 2005. ISBN 978-0-596-00755-3.
2. FRENZEL, Louis E. *Handbook of Serial Communications Interfaces: A comprehensive compendium of serial digital input/output (I/O) standards*. Oxford: Newnes, an imprint of Elsevier, 2016. ISBN 978-0-128-00629-0.
3. BURNETT, C. *An 8-bit Serial Peripheral Interface Bus transfer using two shift-registers* [online]. 2007. Dostupné tiež z: `https://en.wikipedia.org/wiki/File:SPI_8-bit_circular_transfer.svg`. [cit. 2022-04-26].
4. GAY, Warren. *Advanced raspberry pi: Raspbian linux and Gpio Integration*. New York: Apress, 2018. ISBN 978-1-484-23947-6.
5. LACAMERA, Daniele. *Embedded Systems Architecture: Explore Architectural Concepts, Pragmatic Design Patterns, and Best Practices to Produce Robust Systems*. Birmingham: Packt Publishing, 2018. ISBN 978-1-788-83250-2.
6. BURNETT, C. *A single master and three slaves on a Serial Peripheral Interface (SPI) bus* [online]. 2007. Dostupné tiež z: `https://commons.wikimedia.org/wiki/File:SPI_three_slaves.svg`. [cit. 2022-04-24].
7. MOTOROLA, INC. *SPI Block Guide V03.06* [online]. 2003. Dostupné tiež z: `https://web.archive.org/web/20150413003534/http://www.ee.nmt.edu/~teare/ee308l/datasheets/S12SPIV3.pdf`. [cit. 2022-04-21].
8. BURNETT, C. *SPI bus timing diagram* [online]. 2007. Dostupné tiež z: `https://commons.wikimedia.org/wiki/File:SPI_timing_diagram.svg`. [cit. 2022-04-24].
9. ROTH, Charles H.; JOHN, Lizy Kurian. *Digital systems design using VHDL*. 2nd ed. London: Thomson, 2008. ISBN 978-0-534-38462-3.
10. BEZERRA, Eduardo Augusto; LETTNIN, Djones Vinicius. *Synthesizable VHDL design for FPGAs*. Cham: Springer, 2014. ISBN 978-3-319-02546-9.
11. ASHENDEN, Peter J. *Digital design: an embedded systems approach using VHDL*. Burlington: Morgan Kaufmann, 2008. ISBN 978-0-12-369528-4.
12. BIEZL. *Moore machine* [online]. 2010. Dostupné tiež z: `https://commons.wikimedia.org/wiki/File:Moore-Automat-en.svg`. [cit. 2022-04-28].

13. DVORAK, Walter. *Metastability in digital circuits* [online]. 2017. Dostupné tiež z: `https://commons.wikimedia.org/wiki/File:Metastability_D-Flipflops.svg`. [cit. 2022-04-28].

14. YANG, Deng-Ke; WU, Shin-Tson. *Fundamentals of Liquid Crystal Devices*. 2nd ed. Wiley: Chichester, 2015. ISBN 978-1-118-75200-5.

15. RUCKMONGATHAN, Temkar N. *Addressing Techniques of Liquid Crystal Displays*. Chichester: Wiley, 2014. ISBN 978-1-119-94045-6.

16. CHEN, Robert H. *Liquid Crystal Displays Fundamental Physics and Technology*. Wiley: Hoboken, 2011. ISBN 978-0-470-93087-8.

17. ERNST, Lueder. *Liquid Crystal Displays: Addressing Schemes and Electro-Optical Effects*. 2nd ed. Chichester: Wiley, 2010. ISBN 978-0-470-74519-9.

18. KOCH, Markus. *Brother LW-35 Typewriter Modernization* [online]. 2020. Dostupné tiež z: `https://notsyncing.net/?p=blog&b=2020.lw35-mod`. [cit. 2022-04-29].

19. KOCH, Markus. *Brother LW-35 Typewriter Modernization* [online]. 2020. Dostupné tiež z: `https://git.notsyncing.net/electronics/lw35-upgrade`. [cit. 2022-04-30].

20. OKI SEMICONDUCTOR. *Datasheet MSM5598A* [online]. 1997. Dostupné tiež z: `http://pdf.datasheetcatalog.com/datasheet/oki/MSM5298A.pdf`. [cit. 2022-04-07].

21. OKI SEMICONDUCTOR. *Datasheet MSM5599A* [online]. 1997. Dostupné tiež z: `http://pdf.datasheetcatalog.com/datasheet/oki/MSM5299AGS-K.pdf`. [cit. 2022-04-07].

22. OKI SEMICONDUCTOR. *Datasheet MSM6255* [online]. 1997. Dostupné tiež z: `http://pdf.datasheetcatalog.com/datasheet/oki/MSM6255.pdf`. [cit. 2022-04-09].

# Obsah priloženého média

- `src`
  - `impl` ..... zdrojové kódy realizácie radiča a simulácií
  - `project` ..... realizácia radiča vo forme projektu pre Vivado 2018.2
  - `tests` ..... zdrojové kódy testov pre Raspberry Pi
  - `thesis` ..... zdrojové kódy práce vo formáte LaTeX
- `text`
  - `thesis.pdf` ..... text práce vo formáte PDF
- `video` ..... videá výsledkov testovania
  - `test_result_faster.mp4`
  - `test_result_slower.mp4`