FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Programovatelný řídicí systém pro ovládání periferií náročných na přesné časování
Student:	Martin Fujda
Vedoucí:	Ing. Pavel Kubalík, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Počítačové inženýrství
Katedra:	Katedra číslicového návrhu
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

1) Prozkoumejte existující řešení zadané problematiky.
2) Analyzujte dostupné periférie, protokoly komunikace a ovládaní moduly.
3) Vyberte vhodnou HW platformu a navrhněte pro ni integrační desku.
4) Navrhněte vlastní zařízení založené na zvolené architektuře.
5) Zařízení se bude skládat z: kontroléru, síťového modulu, LCD TFT displeje s SD kartou.
6) Zařízení bude navrhnuto tak, aby bylo možné připojit externí periferie pomocí nastavitelného portu.
7) Realizujte obslužnou aplikaci pro zařízení a periferie.
8) Řešení se bude skládat ze serverové části na pozadí a obslužné uživatelské části.
9) Implementujte funkční prototyp, řádně ho zdokumentujte a otestujte.

Elektronicky schválil/a prof. Ing. Hana Kubátová, CSc. dne 5. února 2024 v Praze.

Bakalárska práca

PROGRAMOVATELNÝ ŘÍDICÍ SYSTÉM PRO OVLÁDÁNÍ PERIFERIÍ NÁROČNÝCH NA PŘESNÉ ČASOVÁNÍ

Martin Fujda

Fakulta informačních technologií
Katedra číslicového návrhu
Vedúci: Ing. Pavel Kubalík, Ph.D.
15. mája 2024

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov kódu

Chcel by som sa srdečne poďakovať vedúcemu bakalárskej práce, Ing. Pavlovi Kubalíkovi, Ph.D; za jeho vytrvalosť a významnú pedagogickú, metodickú a odbornú pomoc a Bc. Petrovi Guľovi za vytvorenie 3D modulu šasi prototypu a jeho fyzickú realizáciu.

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 Prahe dňa 15. mája 2024

Abstrakt

Táto bakalárska práca sa zaoberá návrhom programovateľného riadiaceho systému pre ovládanie periférií, ktoré sú náročné na presné časovanie pri použití architektúry MicroBlaze na FPGA obvode spoločnosti Xilinx Inc. Cieľom práce je riešenie implementácie ovládania periférií vo forme modulov, ako je LAN8720 Ethernet Board, 3,5" TFT LCD Shield spolu so slotom SD karty a prevodníkom UART - RS-232. Návrh a realizácia dosky plošného spoja a špecifického rozhrania používaného na komunikáciu s externými zariadeniami je rovnako cieľom práce. Výsledkom je zariadenie postavené na prípravku Cmod A7-35T od spoločnosti Digilent Inc; ktorý obsahuje obvod Artix-7, spolu s obslužnou aplikáciou, ktorá je ovládateľná cez sieť LAN pomocou protokolu Telnet. Táto komunikácia je spravovaná TCP/IP stackom s názvom uIP. Status tejto aplikácie sa zobrazuje na displeji. Systém je schopný emulácie protokolu CBM/IEC s pomocou modulu FatFs a úložiskom SD karty.

Klíčová slova riadiaci systém, ovládanie periférií, presné časovanie, emulácia rozhrania, MicroBlaze, FPGA, LCD displej, Ethernet, Telnet, SD karta

Abstract

This bachelor's thesis covers the development of a programmable control system for controlling peripherals requiring precise timing using the MicroBlaze architecture implemented on an FPGA chip from Xilinx Inc. The goal of the thesis is the implementation of control of peripherals in the form of modules, such as the LAN8720 Ethernet Board, a 3.5" TFT LCD Shield with SD card slot and a UART - RS-232 converter. The creation and realization of PCB design, and the realization of a specific interface used for communication with external devices, is another goal of the thesis. The result is a device based on the development board Cmod A7-35T by Digilent Inc., which contains the Artix-7 chip, together with the service application which is controlled over LAN using Telnet protocol. This communication is managed by a TCP/IP stack named uIP. The status of the application is shown on the display. The system can emulate the CBM/IEC protocol with the help of FatFs module and SD card storage.

Keywords control system, peripheral control, precise timing, interface emulation, MicroBlaze, FPGA, LCD display, Ethernet, Telnet, SD card

Zoznam skratiek

AXI	Advanced eXtensible Interface
BRAM	Block RAM
BUFG	Global Buffer
CAD	Computer-aided design
CPLD	Complex Programmable Logic Device
DBI	Display Bus Interface
DSP	Digital Signal Processor
ETH	Ethernet
FF	Flip-flop
FPGA	Field Programmable Gate Array
GPIO	General-purpose input/output
I^2C	Inter-Integrated Circuit
IO	Input/Output
IP core	Intellectual Property core
ISO/OSI	International Organization for Standardization/Open Systems Interconnection
JTAG	Joint Test Action Group
LAN	Local Area Network
LCD	Liquid-Crystal Display
LED	Light-Emitting Diode
LUT	Look-up table
MAC	Media Access Control
MCU	Microcontroller Unit
MDC	MDIO Interface Clock
MDIO	Management Data Input/Output
MII	Media Independent Interface
MIPI	Mobile Industry Processor Interface
MMCM	Mixed-Mode Clock Manager
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MPU	Microprocessor Unit
OLED	Organic LED
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PHY	Physical Layer
PS/2 port	Personal System/2 port
RAM	Random Access Memory
RGB	Red Green Blue
RMII	Reduced MII
ROM	Read Only Memory
SD	SecureDigital
SIO (Atari)	Serial Input/Output system
SMD	Surface-mounted device
SoC	System on Chip
SPI	Serial Peripheral Interface
TFT	Thin-Film-Transistor
THT	Through-hole technology
UART	Universal Asynchronous Received-Transmitter
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit program

Úvod

Dnešných svet je postavený na inováciách. Určite to platí v odbore elektroniky a počítačových systémov. Vývoj nových, dokonalejších a efektívnejších zariadení je základným heslom firiem, ak si chcú udržať svoju relevanciu na trhu. Táto motivácia je posilnená možnosťou rozširovať prístroje o pripojenie k internetu. V mnohých stránkach je takáto integrácia pozitívna, existuje ale dostatočne veľa negatívnych stránok takéto pokroku. Jednou z nich je plánované zastarávanie. Pre mnohých toto znamená nutnosť vymieňať zariadenia za nové, čo zvyšuje náklady a v mnohých prípadoch nie je potrebné. Čo ak sa zákazník rozhodne, že si chce ponechať a ďalej využívať takýto prístroj? Takéto rozhodnutie si bude určite po niekoľkých rokoch vyžadovať použitie doplnkového zariadenia, ktoré bude schopné preklenúť takúto novovzniknutú bariéru.

Práve odtiaľ prichádza myšlienka vytvoriť takéto zariadenie, ktoré je predmetom tejto bakalárskej práce. Vytvoriť prototyp zariadenia, ku ktorému bude možné pripojiť staršie zariadenia pomocou v dnešnej dobe nepoužívaných rozhraní. Niektoré z takýchto rozhraní a zberníc sú veľmi náročné na presné časovanie. Iné zas pracujú na rôznych logických úrovniach. Práve preto musí takýto prototyp obsahovať obsluhu, ktoré dokáže reagovať dostatočne rýchlo a zároveň porty, ktoré pracujú na najčastejšie používaných logických úrovniach.

Súčasťou plánovaného zariadenia musia byť komponenty, ktoré sú v dnešnej dobe dostupné na trhu a teda aj zaužívané protokoly, rozhrania a formáty, ktoré sú používané dnešnými systémami. Takto bude zaručená možnosť spravovať vytvorený prototyp zariadenia použitím moderných počítačov a rovnako možnosť opraviť nefunkčné časti. Rovnako je potrebné vytvoriť možnosti pre zmeny softvérovej výbavy zariadenia, či už pri aktualizácií alebo pri celkovej zmene konfigurácie.

Hlavným cieľom tejto bakalárskej práce je preto vytvoriť programovateľný riadiaci systém pre ovládanie periférií náročných na presné časovanie. Výsledný prototyp bude rozdelený na dve časti. Základná serverová časť, ktorá bude realizovať pripojenie k sieti LAN spolu s Telnet serverom, prístup k lokálnemu úložisku, riadenie zabudovaných periférií a nakoniec ovládanie externých zariadení a dodatočná užívateľská časť, ktorá bude prezentovať stav, konfiguráciu a iné údaje používateľovi a bude spracovávať jeho príkazy.

Súčasťou bakalárskej práce bude analýza existujúcich riešení, kde budú identifikované ich nedostatky, analyzované použité metódy a technológie, analýza dostupných modulov nutných pre tento projekt, porovnanie ich parametrov a ceny. Praktická časť sa bude venovať návrhu a realizácií dosky plošného spoja, prepojeniu vybraných modulov, implementácií potrebných funkcií a vývoju demo aplikácie, ktorá bude obsahovať možnosť emulácie rozhrania CBM/IEC náročného na presné časovanie. Takáto emulácia bude testovaná pripojením k počítaču Commodore 64.

Kapitola 1
Cieľ

Cieľom bakalárskej práce je navrhnúť a vytvoriť programovateľný riadiaci systém s konfigurovateľným rozhraním pre komunikáciu s presným časovaním spolu s ovládaním periférnych zariadení. Súčasťou práce bude aj návrh a realizácia dosky plošného spoja a výroba prototypu zariadenia.

Prieskum existujúcich riešení je prvým cieľom práce. V tejto časti budú analyzované vybrané projekty, metódy a technológie použité pri týchto projektoch, ich kladné a záporné stránky, funkcionalita a flexibilita výsledného produktu.

Po prieskume budú na základe výsledkov analyzované dostupné moduly pre zjednodušenie vývoja aplikácie a návrhu prototypu. Rozbor sa bude venovať preskúmaniu dostupných technológií, porovnaniu parametrov týchto modulov a ich cien.

Ďalším cieľom práce bude, na základe vybraných modulov, rozbor protokolov a rozhraní potrebných pre komunikáciu s perifériami a taktiež s externými zariadeniami. Bude nutné preskúmať dostupnosť knižníc pre zvolené moduly, IP jadier alebo iných čiastkových riešení tejto problematiky.

Na základe dôkladnej predchádzajúcej analýzy sa bude praktická časť bakalárskej práce venovať realizácii samotného hardvérového prototypu a softvérovej aplikácie.

Prvým cieľom praktickej časti je návrh schémy a dosky plošného spoja, ich realizácia, následné prepojenie všetkých používaných modulov, podporných súčiastok a nakoniec vytvorenie hardvérového prototypu.

Druhým cieľom praktickej časti je implementácia aplikácie. Súčasťou je vytvorenie obsluhy všetkých periférií a taktiež externých zariadení, riešenie serveru Telnet a konfigurovateľného rozhrania spolu so zavedením podporných knižníc a/alebo IP jadier.

Poslednou časťou a cieľom bude testovanie výsledného prototypu spolu s jeho aplikačnou výbavou. Na základe testovania budú vyvodené závery a možné rozšírenia tejto práce spolu s pridaním podpory pre ďalšie zariadenia.

Kapitola 2
Prieskum existujúcich riešení

Kapitola sa zaoberá preskúmaním dostupných riešení problematiky, analýzou použitých metód, technológií a návrhu výsledného produktu. Riešenia sú ohodnotené, pri každom sú uvedené kladné a záporné stránky.

2.1 CBM/IEC emulátor Pi1541

CBM/IEC emulátor Pi1541 je emulátorom disketovej mechaniky (Commodore 1541) a riadiaci systém pre sériové rozhranie CBM/IEC Bus (IEEE-488), ktoré využívajú 8-bitové počítače od spoločnosti Commodore. Funguje s modelmi počítačov Commodore 64, Commodore 128, Commodore VIC20, Commodore 16 a Commodore PLUS4. Postavené na prípravku Raspberry Pi. Podporované sú len modely Raspberry Pi 3B, 3B+, 3A+ alebo Raspberry Pi 4. [1]

Využíva obojsmerné prevodníky logickej úrovne z 3.3 V na 5 V a naopak. Využívané sú len štyri vodiče zo sériového rozhrania (RESET, DATA IN/OUT, CLK IN/OUT, ATN OUT). Návrh obsahuje päť tlačidiel, pomocou ktorých je možné spustiť alebo ukončiť emuláciu, prechádzať cez priečinky a dostupné obrazy diskov a pridávať disky do množiny používaných obrazov. Ďalej je súčasťou piezo bzučiak a LED pre signalizáciu stavu emulácie.

Obrazy diskov sa spolu s operačným systémom ukladajú na SD kartu, ktorá sa potom vloží do slotu a prípravok sa pripojí k napájaniu. V konfiguračných súboroch je možné zmeniť nejaké nastavenia správania emulátora, napr. zmena ID zariadenia, výmena systémovej ROM, zmena rozlíšenia obrazovky a ďalšie.

Výhody:

+ Komunikácia cez sériové rozhranie je real-time a cycle exact
+ Využíva bežne dostupný prípravok Raspberry Pi

Nevýhody:

– Ovládanie nie je veľmi intuitívne
– Podporuje len jedno špecifické rozhranie
– Nekompatibilita so staršími verziami Raspberry Pi

Z tohoto projektu je zrejmé, že je potrebné použiť prevodník logickej úrovne pre prácu s rozhraním CBM/IEC Bus.

2.2 SIO emulátor SDrive-MAX

Emulátor disketovej (Atari 1050) a kazetovej mechaniky (Atari 1010). Zároveň riadi sériové rozhranie SIO, je používané 8-bitovými počítačmi od spoločnosti Atari. Rozšírenie existujúceho projektu sdrive-ng pridaním dotykového TFT displeja, použitím Arduino Uno a prevodom ovládania cez dotyk. Podporované sú zariadenia Atari 400/800, Atari 1200XL, Atari 600XL/800XL a Atari 65XE/130XE. [2]

Pretože projekt využíva Arduino Uno, ktorého základom je čip ATmega328p pracujúci s logickými úrovňami 5 V, nie je potrebné využívať prevodník na komunikáciu s externým počítačom. Napájanie je dosiahnuté použitím dostupného napätia na rozhraní SIO, odporúčané je ale použiť externý zdroj [2]. Využité sú tri dátové vodiče (DATA IN, DATA OUT, COMMAND) a napájanie (5V a GND). Pri dátovom vodiči DATA IN je nutné použiť Schottky diódu.

Na Arduino je potrebné nahrať dva konfiguračné súbory, ktoré obsahujú programové vybavenie. Potom je možné čítať dáta z SD karty, ktorá je zasunutá do slotu na displej modulu. Pomocou dotykového vstupu je možné ovládať aplikáciu, ktorú je možné nastaviť na emuláciu až štyroch mechaník, kde na každú z nich je potrebné vybrať obraz disku z dostupných súborov na SD karte.

Výhody:

+ Možnosť emulovať viacero mechaník zároveň
+ Ovládanie pomocou dotykového displeja
+ Schopné pracovať ako samostatná jednotka

Nevýhody:

– Podporuje len jedno špecifické rozhranie
– Pre zmenu konfigurácie potrebné nahrať dva programy

Na základe tohoto projektu a jednoduchosti ovládania je zrejmé použitie dotykového TFT displeja. Ukážka fungujúceho modelu na obr. 2.1.

Obr. 2.1 SDrive-MAX [2]

2.3 Emulátor ZXPicoMD

Emulátor mikro-kazetovej mechaniky (ZX Microdrive) využívanej počítačom ZX Spectrum od spoločnosti Sinclair. Riadenie prebieha cez rozhranie ZX Interface 1. Poskytuje možnosť čítať údaje z fyzickej mechaniky, ktorú je možné pripojiť k zariadeniu. [3]

Projekt využíva Raspberry Pico ako základ, ku ktorému je pripojená čítačka SD karty, OLED displej, päť tlačidiel pre ovládanie, dve LED pre signalizáciu prístupu na SD kartu a signalizáciu prístupu k zápisu na fyzický disk, ak je pripojený. Na OLED displeji sa zobrazuje menu, kde je pomocou tlačidiel možné presúvať sa cez ponuku a konfigurovať emulátor. Staršia verzia využívala prevodník logickej úrovne, v novšej už nie je pre čítanie potrebný.

Konfigurácia zariadenia je jednoduchá, stačí pripojiť Raspberry Pico cez USB a uložiť na neho súbor programu. Tento potom sprístupní emuláciu, čítanie a zápis súborov na SD kartu a prípadné využitie fyzickej mechaniky. Aplikácia je schopná emulovať až osem mechaník, kde každá má vlastný obraz disku.

Výhody:

+ Možnosť využiť fyzickú mechaniku na zálohovanie dát
+ Grafické menu s ovládaním pomocou tlačidiel

Nevýhody:

– Celkom komplikovaný návrh
– Pre pripojenie k počítaču je potrebný špecifický kábel

V tomto projekte je znova využitý prevodník logických úrovní, displej a SD karta ako úložisko súborov. Ukážka na obr. 2.2.

Obr. 2.2 ZXPicoMD [3]

Kapitola 3
Analýza

Na základe výsledkov prieskumu uvedených v kapitole 2 sa táto kapitola venuje analýze potrebných modulov a podporných súčiastok, ich parametrov a cien. Následne sú analyzované potrebné protokoly a rozhrania pre periférie.

3.1 Dostupné moduly

V tejto sekcii sa nachádza prieskum dostupných modulov, ktoré budú použité pre dosiahnutie cieľov bakalárskej práce. Súčasťou je porovnanie parametrov, funkcií a ceny modulov u každého z nich. Nakoniec sú zvolené tie, ktoré sú na základe analýzy ohodnotené ako najlepšie.

3.1.1 Ovládacie a riadiace prvky

Každý projekt, ktorý pozostáva z ovládania, komunikácie, emulácie a obsluhy servera si nutne vyžaduje požitie počítača alebo kontroléru. Keďže sa táto práca sústreďuje aj na vývoj prototypu, je potrebné, aby bola použitá menšia verzia takýchto zariadení, inak povedané ich mikro-prevedenie.

V dnešnej dobe sú na trhu dostupné viaceré typy riadiacich jednotiek, najpoužívanejšie z nich sú mikrokontrolér, mikroprocesor a System on Chip. Porovnanie týchto riešení na základe ich kladov a záporov je uvedené v tab. 3.1.

Tabuľka 3.1 Porovnanie parametrov riadiacich jednotiek

Parameter	MCU	MPU	SoC
Komplexnosť	nízka	vysoká	stredná
Potrebné periférie	nie	áno	nie
Veľkosť	malá	malá	veľká
Výkon	nízky	stredný	vysoký
Cena	nízka	nízka	vysoká

Na základe porovnania parametrov je zjavné, že najlepšou voľbou pre potreby tejto práce je použitie mikrokontroléra. Nevýhodou použitia vybranej riadiacej jednotky je jej výkon, ktorý pre potreby súborového serveru nebude dostatočný. Tento nedostatok by sa prejavil hlavne pri prenose dát cez počítačovú sieť a pri komunikácii s SD kartou. Ďalším problémom je podpora rozhraní, kde väčšina mikrokontrolérov obsahuje zabudovanú implementáciu niektorých najpopulárnejších, najmä SPI, I^2C a UART [4]. Pridanie ďalších rozhraní je komplikované a v niektorých prípadoch nemusí byť možné.

Existuje ale riešenie, ktoré má funkcionalitu MCU, vyššiu rýchlosť, vysokú rozšíriteľnosť, nízku komplexnosť programovania, malú veľkosť a na svoju činnosť si nevyžaduje žiadne periférie. Nazývajú sa programovateľné logické obvody. Dostupné sú viaceré technológie, ktoré sa v dnešnej dobe používajú. Patria medzi ne obvody CPLD a FPGA. Porovnanie parametrov oboch technológií, pre ukážku obvodov AMD CoolRunner II [5] a AMD Artix 7 [6], sa nachádza v tab. 3.2.

Tabuľka 3.2 Porovnanie parametrov technológií programovateľných logických obvodov [5, 6]

Parameter	CPLD CoolRunner II	FPGA Artix 7
Počet logických hradiel	750-12.000	12.800-215.360
Maximálna frekvencia	323 MHz	266,67 MHz
Počet V/V portov	33-270	150-500
Zabudovaná pamäť	0 kB	750-13.140 kB
Cena	4-330 €	85-510 €

Po porovnaní je zrejmé, že obvod FPGA poskytuje viac logických hradiel a zabudovanej pamäte, čo sú dva najpodstatnejšie parametre pre túto prácu. Využitie internej blokovej pamäte je dôležité pre rýchlu dátovú a inštrukčnú cache, ktorá je schopná urýchliť výslednú aplikáciu.

Obvody FPGA sa od výrobcu dodávajú ako čipy, pre ktoré je nutné vyrobiť dosku plošného spoja so všetkými potrebnými súčiastkami, aby bolo možné obvod začať používať. To je časovo a finančne náročné a preto sa pri tejto bakalárskej práci bude používať na trhu dostupný vývojový prípravok. Cena prípravku je vyššia ako samotného čipu, pretože väčšinou obsahujú aj sériové NOR Flash pamäte, SRAM alebo DRAM, prevodník USB – UART, programátor JTAG, regulátor napätia a iné periférie [7].

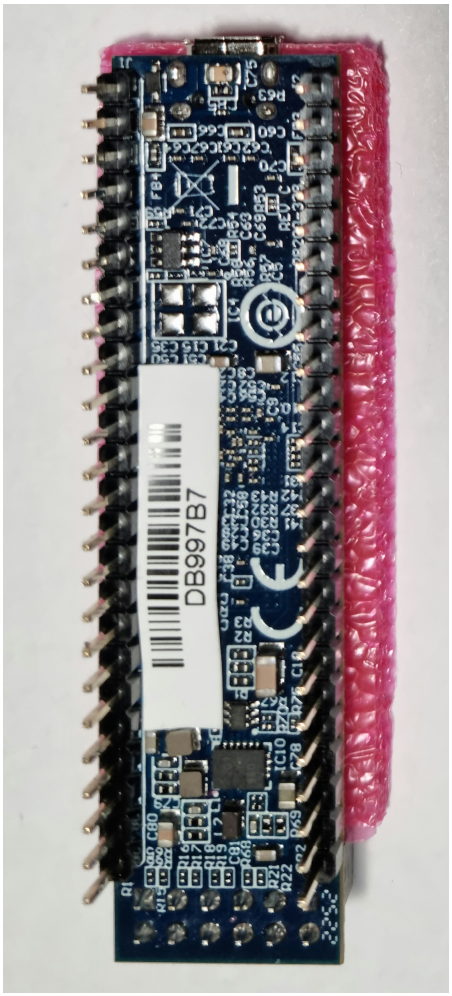
Pretože sú trhu dostupné obvody FPGA len od spoločností Intel, firmy Advanced Micro Devices a Lattice Semiconductor, z ktorých každá má vlastnú softvérovú výbavu s IP jadrami a špecifických dodávateľov vývojových prípravkov, je potrebné vybrať si produkt od jednej z nich. Na základe doterajšej skúsenosti s produktami firmy AMD Inc. a Digilent Inc., je výber obmedzený len na ich produkty. Proces vývoja softvérovej časti a riešenia obsahu obvodu FPGA bude nutne prebiehať v softvérovom balíku AMD Vivado Design Suite a bude použitá verzia 2018.3 [8] z dôvodu väčšej skúsenosti a znalosti funkcionalít, problémov a potrebnej konfigurácie. Súčasťou je možnosť využitia architektúru MicroBlaze, ktorá sa dodáva ako súčasť softvéru. Tomuto procesoru je možné zmeniť alebo pridať funkcie, ktoré nie sú dostupné u MCU.

Ponuka vývojových prípravkov firmy Digilent je široká, ponúkané sú prípravky s rôznymi perifériami, komplexnejšími alebo jednoduchšími návrhmi. Výber je zameraný na tie, ktoré sú cenovo dostupné, obsahujúce dostatočne veľkú pamäť RAM a Flash, obvod FPGA s väčším počtom dostupných logických hradiel a vyšším počtom vývodov pre externé periférie. Porovnanie parametrov takýchto vývojových prípravkov sa nachádza v tab. 3.3. Podrobnosti jednotlivých prípravkov sú dostupné na stránkach výrobcu Digilent [9, 10, 11, 7].

Tabuľka 3.3 Porovnanie parametrov vývojových prípravkov firmy Digilent Inc. [9, 10, 11, 7]

Parameter	Basys 3	Arty A7	Nexys A7	Cmod A7
Obvod FPGA	XC7A35T	XC7A100T	XC7A100T	XC7A35T
Typ externej pamäte	-	DDR3	DDR2	SRAM
Veľkosť RAM	-	256 MB	128 MB	512 kB
Veľkosť Flash	32-Mbit	128-Mbit	128-Mbit	32-Mbit
Počet použiteľných vývodov	4*8	4*8 + 49	5*8	44 + 8
Zabudované periférie	VGA, PS/2	ETH	ETH, VGA, SD	-
Cena	160 €	278 €	530 €	95 €

Na základe predchádzajúceho porovnania a s ohľadom na cenu bol pre bakalársku prácu zvolený prípravok Cmod A7. Obsahuje dostatočné kapacity RAM na výslednú aplikáciu a obvod FPGA má dostatočný počet hradiel pre obsluhu procesora MicroBlaze, periférnych IP jadier, pamäte cache a ďalších súčastí projektu. Rozloženie jednotlivých komponentov na prípravku a jeho vývodov je na obr. 3.1 a 3.2.



Obr. 3.1 Digilent Cmod A7-35T; pohľad spredu **Obr. 3.2** Digilent Cmod A7-35T; pohľad zozadu

3.1.2 Moduly sieťovej komunikácie

Na komunikáciu cez počítačovú sieť je nutné k prípravku pridať modul, ktorý je schopný takéto prenosy uskutočniť. Je na výber z veľkého množstva dostupných modulov s rôznymi parametrami. Na začiatku je potrebné vybrať, akú technológiu prenosu bude modul využívať. Dve najpoužívanejšie sú Ethernet a Wi-Fi. Porovnanie oboch technológií v kontexte modulov, ich kladov a záporov, sa nachádza v tab. 3.4.

Väčšina modulov využívajúcich technológiu Wi-Fi je obmedzená rýchlosťou externej zbernice SPI, UART alebo I^2C. Pretože bez modulu, ktorý komunikuje cez rozhranie USB alebo PCI, nie je možné docieliť uspokojujúce rýchlosti prenosu, je pre udržanie jednoduchosti nutné si zvoliť technológiu Ethernet, pomocou ktorej sa dá bez väčších problémov dosiahnuť rýchlosť až 1 Gbps.

Tabuľka 3.4 Porovnanie parametrov sieťových technológií

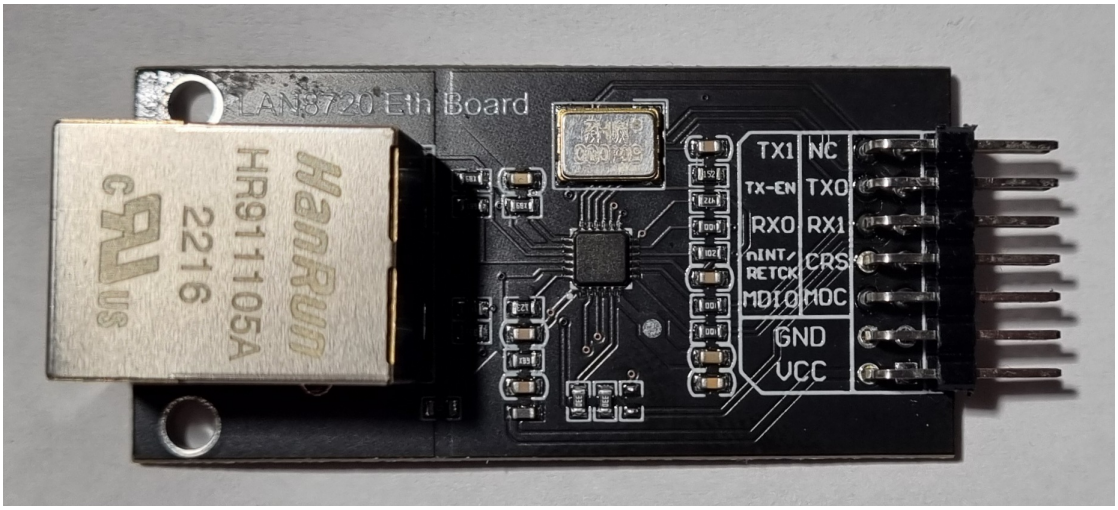
Parameter	Ethernet	Wi-Fi
Komplexnosť konfigurácie	nízka	vysoká
Efektívna rýchlosť	100/1000 Mbps	150 Mbps
Maximálny dosah	100 m	70 m
Potrebná obsluha	minimálna	vyššia
Flexibilita	malá	väčšia

Po zvolení technológie je potrebné vybrať z dostupných modulov na trhu. Existuje veľké množstvo rôznych riešení, ktoré sa líšia hlavne externým rozhraním a maximálnou rýchlosťou prenosu. Podľa používaného rozhrania závisí, či je potrebné využiť knižnice dodávané s takýmto modulom alebo či postačuje obecné riešenie. Porovnanie parametrov niektorých Ethernet modulov sa nachádza v tab. 3.5. Údaje o jednotlivých modulov boli prebrané z technickej dokumentácie každého modulu [12, 13, 14].

Tabuľka 3.5 Porovnanie parametrov Ethernet modulov [12, 13, 14]

Parameter	Mini W5500	USR-TCP232-T2	LAN8720
Externé rozhranie	SPI	UART	RMII + MDIO
Maximálna rýchlosť	100 Mbps	100 Mbps	100 Mbps
Efektívna rýchlosť	80 Mbps	460 kbps	100 Mbps
Obsluha vrstvy	PHY & MAC	PHY & MAC	PHY
Zabudovaný procesor	áno	áno	nie
Cena	6 €	20 €	4 €

Z porovnania dostupných modulov je možné vyvodiť záver, že najlepšia voľba podľa priorít je LAN8720 ETH board, ktorý je zosnímaný na obr. 3.3 a kde je možné vidieť vyvedenie rozhrania RMII a MDIO spolu s napájaním na úrovni 3,3 V. Vyžaduje si síce nadstavbu v zmysle implementácie MAC vrstvy, ale keďže softvérový balík Vivado poskytuje IP jadro, ktoré túto vrstvu rieši na hardvérovej aj softvérovej úrovni, nevyžaduje sa žiadna samostatná implementácia alebo riešenie.



Obr. 3.3 LAN8720 Eth board

3.1.3 Vstupno/výstupné moduly

Nevyhnutnosť získať údaje o aktuálnom stave zariadenia a mať možnosť upraviť jeho nastavenia predstavuje potrebu k periférиám zaradiť vstupno/výstupný modul. Nie je nutné, aby boli obe funkčnosti časťou jedného modulu. Preto je možné zaradiť do porovnania aj periférie, ktoré sú schopné realizovať aspoň jeden smer komunikácie s užívateľom.

Výstupný smer (komunikácia zariadenia s užívateľom) je možné riešiť rôznymi spôsobmi. Pokiaľ nie je potrebné stanovisko o stave, dá sa použiť indikácia pomocou viacerých LED, kde každá reprezentuje nejaký presne definovaný stav. Ak je komplexnosť oznámenia vyššia, je lepšie dáta zobrazovať na displeji. Existuje veľké množstvo displejov, ktoré sa líšia veľkosťou, použitou technológiou, komunikačným rozhraním, obnovovacou frekvenciou a ďalšími parametrami. Keďže táto práca sa zameriava na vytvorenie serveru a emulátoru, u ktorých je možné a v istých prípadoch aj nutné pre správne fungovanie získavať špecifické a komplexné informácie o stave, je efektívnejšie využiť displej. V tab. 3.6 sa nachádza porovnanie parametrov dostupných technológií displejov, ich kladných a záporných stránok.

Tabuľka 3.6 Porovnanie parametrov zobrazovacích technológií displejov

Parameter	E-ink	LCD	LED	OLED	TFT
Rozlíšenie	priemerné	nízke	veľmi nízke	vysoké	vysoké
Obnovovacia frekvencia	veľmi nízka	nízka	stredná	vysoká	vysoká
Potrebné podsvietenie	nie	áno/nie	nie	nie	áno
Spotreba	nízka	vysoká	vysoká	vyššia	vysoká
Hrúbka	malá	veľká	veľká	malá	malá
Uhlopriečka	malá	veľmi malá	malá	veľká	veľká
Cena	vysoká	nízka	nízka	vyššia	priemerná

Cez porovnanie dnes používaných technológií displejov je pre túto prácu najefektívnejšie použiť technológiu TFT. Inak by bola zvolená technológia OLED, ak by bola vyžadovaná nízka spotreba a cena by nebola pri výbere podstatná. Po zvolení si technológie je možné preskúmať moduly dostupné na trhu. Každý z nich má svoje špecifiká v zmysle používaných protokolov komunikácie, napájania, realizácie a dodatočných funkcií. Parametre, ktoré sú uplatniteľné pre každý z nich, sú veľkosť uhlopriečky obrazovky a podpora funkcie dotykového vstupu. V nasledujúcej tab. 3.7 sa nachádza porovnanie parametrov displejov na základe viacerých parametrov. Uvedené údaje sú súčasťou dokumentácie od výrobcov [15, 16, 17, 18].

Tabuľka 3.7 Porovnanie parametrov TFT displej modulov [15, 16, 17, 18]

Parameter	MSP1308	MAR3501	NX4024T032	MPI4008
Uhlopriečka obrazovky	1,3”(3,3 cm)	3,5”(8,9 cm)	3,2”(8,1 cm)	4”(10,2 cm)
Rozlíšenie obrazovky	240x240	320x480	400x240	400x800
Rozhranie	SPI	MIPI DBI	UART	HDMI
Driver čip	ST7789	ILI9486	STM32F030	NT35510
Snímanie dotyku	nie	áno	áno	áno
Doplnky	-	μSD slot	-	-
Cena	5 €	14 €	47 €	36 €

Po preskúmaní dostupných možností je najviac vyhovujúci displej modul MAR3501. Obsahuje aj snímanie dotyku, čo môže zabezpečiť vstupný smer (komunikácia užívateľa so zariadením) a aj slot μSD karty využiteľný pre realizáciu úložiska servera. Nie je teda potrebné riešiť smer ľudskej komunikácie iným spôsobom, napr. pomocou tlačidiel a nie je nutné pridávať ďalší modul

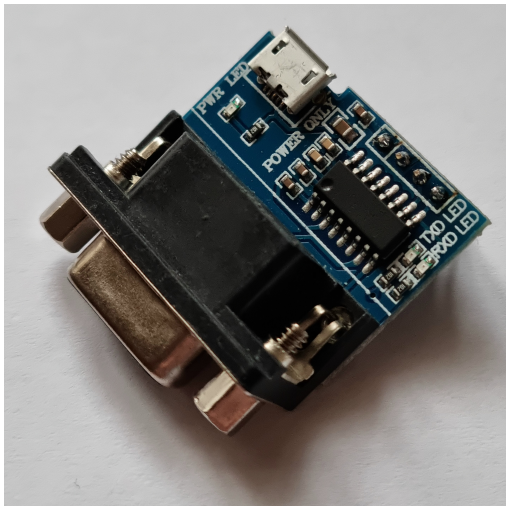
reprezentujúci nejakú formu úložiska. Ukážka prednej a zadnej strany takéhoto modulu je viditeľná na obr. 3.4 a 3.5. Pre displej je dostupná knižnica LCDWIKI, ktorá obsahuje komplexné riešenie komunikácie. Dostupná je na stránke [16] v sekcií **Program Download**.

Obr. 3.4 TFT displej modul MAR3501; pohľad spredu

Obr. 3.5 TFT displej modul MAR3501; pohľad zozadu

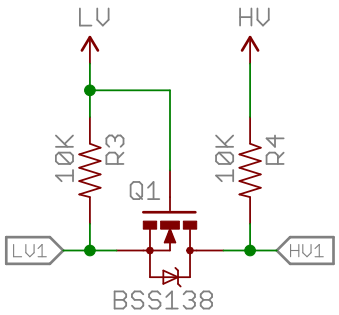
3.1.4 Podporné moduly a súčiastky

Pre rozšírenie možností výsledného prototypu bol pri neskoršom vývoji pridaný do návrhu **prevodník UART – RS-232**. Na trhu existuje veľké množstvo takýchto prevedníkov. Pretože väčšina z nich je postavená na čipe **MAX3222**, nie je potrebné ďalej analyzovať ich funkcie. Tento čip obsahuje všetky potrebné logické obvody spolu s tzv. **charge pump**. To znamená, že pri dodaní napätia v rozmedzí od 3 V do 5,5 V dokáže pomocou externých kondenzátorov generovať potrebné kladné a záporné napätia pre štandard **RS-232** [19]. Preto nie sú potrebné žiadne ďalšie integrované obvody. Na základe dostupnosti a ceny bol zvolený generický modul prevodníka, ktorý je dostupný u mnohých distribútorov elektronických súčiastok. Jeho fyzické riešenie je viditeľné na obr. 3.6. Zvolený modul sa následne jednoducho inkorporuje do návrhu PCB a blokového návrhu, keďže protokol **UART** je univerzálny.



Obr. 3.6 Modul UART – RS-232 prevodníka

Pre uskutočnenie komunikácie na logickej úrovni 5 V je potrebné použiť tzv. **level shifter**, čo je obojsmerný prevodník logickej úrovne na inú logickú úroveň, najčastejšie z 3,3 V na 5 V a naopak. Existujú integrované obvody, ktoré realizujú takýto prevod bez potreby ďalších súčiastok. Obvod ako **TXB0108** spĺňuje požiadavky na spomínaný prevod, ale keďže je postavený na základe tzv. **buffer**, čo nutne znamená oneskorenie signálu [20]. Ak by bol v návrhu takýto obvod použitý pri rozhraní, kde je nutné dodržiavať presné načasovanie signálov, mohlo by oneskorenie vytvárať problémy. Preto je lepšie použiť jednoduchú konfiguráciu s **N-channel MOSFET** a rezistormi. Schéma takého zapojenia je na obr. 3.7, kde **LV** = 3,3 V a **HV** = 5 V.



Obr. 3.7 Schéma prevodníka logických úrovní [21]

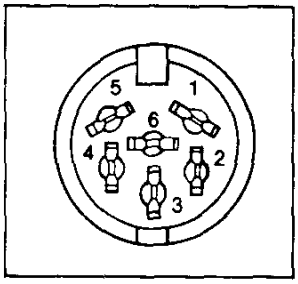
3.2 Protokoly a rozhrania

Táto sekcia sa venuje preskúmaniu špecifikácií zberníc, rozhraní a protokolov, používaných pri tejto bakalárskej práci. Z dostupnej dokumentácie sú vybrané tie najdôležitejšie informácie potrebné k pochopeniu základov problematiky. U každej z opisovaných tematík sú uvedené grafické reprezentácie pripojenia a/alebo časové diagramy prenosu údajov.

3.2.1 CBM/IEC Bus

Rozhranie CBM je využívané počítačmi od spoločnosti Commodore International Corporation, od modelu VIC-20 až po model 128. Základ je špecifikácia **IEEE-488**, ktorá opisuje 8-bitové paralelné rozhranie typu multi-master. Táto verzia bola použitá u počítača PET, z ktorého potom vychádzajú všetky nasledujúce verzie. Pre počítač VIC-20 a neskoršie modely bola vytvorená synchrónna sériová verzia spomínanej zbernice, ktorá používa 4 vodiče: $\overline{CLK}$ pre hodinový signál, $\overline{DATA}$ na obojsmerný prenos dát, $\overline{ATN}$ signál ovládaný počítačom a používaný na riadenie prevádzky a signál $\overline{RESET}$, ktorý je u niektorých modelov používaný na reštartovanie počítača. Hardvérovo je rozhranie riešené na logickej úrovni **5 V** typu **otvorený kolektor** s pull-up, čo znamená že logická 1 je definovaná ako hodnota 0 a logická 0 zasa hodnota 1, dáta sú posielané po 8 bitoch počnúc LSB a sú validné pri nábežnej hrane hodinového signálu. [22] Rozloženie vývodov takéhoto rozhrania je na obr. 3.8.

PIN	DESCRIPTION
1	SERIAL SRQ IN
2	GND
3	SERIAL ATN IN/OUT
4	SERIAL CLK IN/OUT
5	SERIAL DATA IN/OUT
6	NO CONNECTION



Obr. 3.8 Schéma vývodov rozhrania CBM/IEC [23]

Je potrebné dodať, že každé zariadenie na zbernici má pridelené **ID**. Identifikované sú aj zariadenia interné, takže ID 0-3 sú rezervované a keďže zariadení, ktoré bolo možné pripojiť k počítaču cez rozhranie bol obmedzený počet, boli identifikátory zaužívané a priradené v tab. 3.8.

Tabuľka 3.8 Identifikátory zariadení na zbernici CBM/IEC [22]

ID zariadenia	Typ zariadenia
0-3	klávesnica, kazetový port, RS-232 na používateľskom porte, obrazovka
4-5	tlačiareň
6-7	plotter
8-15	disketová mechanika
16-30	nedefinované
31	rezervované (príkaz pre všetky zariadenia - neoverené)

Zariadenia na zbernici komunikujú pomocou príkazov. Existuje 7 zaužívaných príkazov, na ktoré musia zariadenia reagovať. Všetky z nich sú v tab. 3.9.

Tabuľka 3.9 Príkazy používané zbernicou CBM/IEC [22]

Hodnota	Príkaz
0x20	**LISTEN** + identifikátor zariadenia (0-30)
0x3F	**UNLISTEN**
0x40	**TALK** + identifikátor zariadenia (0-30)
0x5F	**UNTALK**
0x60	**OPEN CHANNEL/DATA** + sekundárna adresa/kanál (0-15)
0x70 - 0xDF	nedefinované
0xE0	**CLOSE** + sekundárna adresa/kanál (0-15)
0xF0	**OPEN** + sekundárna adresa/kanál (0-15)

Príkazy sú definované ako hexadecimálna hodnota, ku ktorej sa následne pomocou operácie **OR** pridajú do zvyšku dodatočné údaje (napr. 0x28 znamená **LISTEN DEVICE 8**, 0x45 predstavuje **TALK DEVICE 4**). Hodnota 0, 1 a 15 sekundárnej adresy príkazu **OPEN CHANNEL/DATA** sú rezervované pre čítanie súboru PRG, zápis súboru PRG a DOS príkaz/status zariadenia v spomínanom poradí. Hodnoty 2-14 reprezentujú rôzne kombinácie typov súborov a súborových operácií (READ, WRITE, READ-WRITE, atď.) pri spomínanom príkaze. Súbor **PRG** je definovaný ako programový súbor. Obsahuje kód spustiteľného programu, ktorý môže byť po priatí priamo skopírovaný do pamäte počítača a následne spustený príkazom **RUN**. [22]

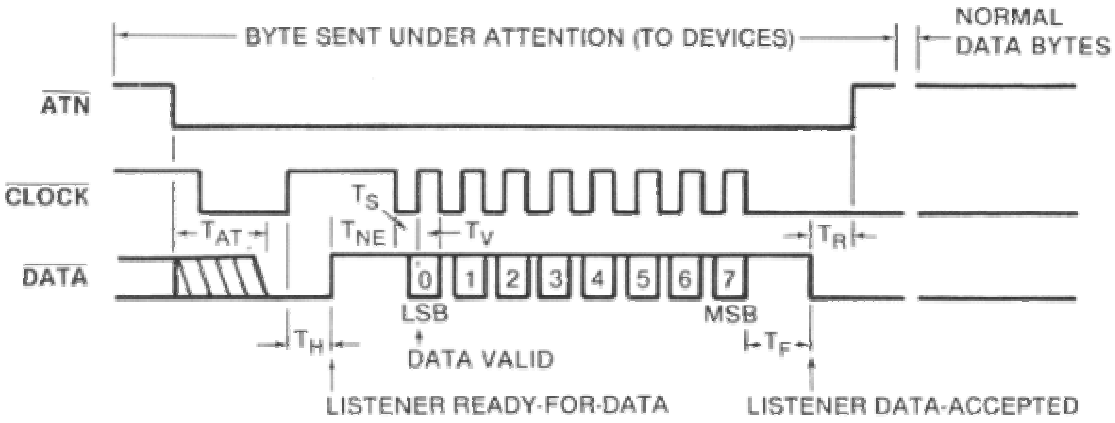
Komunikácia cez zbernicu si vyžaduje logickú postupnosť príkazov a signálov, rovnako dôležité je aj ich načasovanie. Najjednoduchším spôsobom, ako objasniť tento postup je použitie vhodného vzorového príkladu. **Počítač** chce z **disketovej mechaniky** s **ID 8** prečítať súbor s názvom *****. Pred začatím je potrebné deklarovať zopár časových hodnôt, ktoré budú neskôr v príklade používané, v tab. 3.10.

Tabuľka 3.10 Časové limity pre komunikáciu cez CBM/IEC [22]

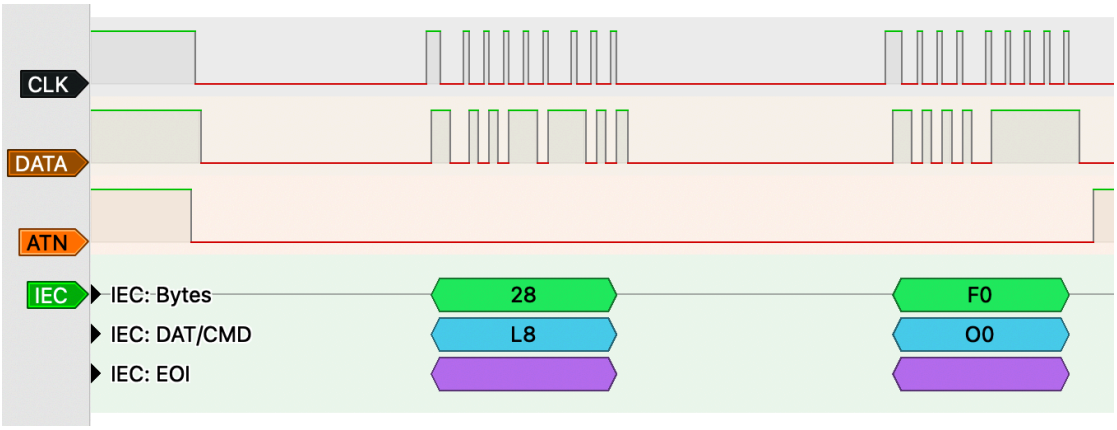
Symbol	Definícia	Min.	Typ.	Max.
T_{AT}	odpoveď na stav $\overline{ATN}$ (povinná)	-	-	1000 μs
T_H	potvrdenie od **LISTENER**	0	-	∞
T_S	príprava hodnoty $\overline{DATA}$	20 μs	70 μs	-
T_V	hodnota $\overline{DATA}$ validná	20 μs	20 μs	-
T_F	potvrdenie prijatia dát	0	20 μs	1000 μs
T_R	ukončenie stavu $\overline{ATN}$	20 μs	-	-
T_{BB}	čas medzi bajtmi	100 μs	-	-
T_{YE}	odpoveď na **EOI**	200 μs	250 μs	-
T_{EI}	príprava zariadenia po **EOI**	60 μs	-	-
T_{RY}	limit odpovede **TALKER**	0	30 μs	60 μs
T_{FR}	čas medzi bajtmi po **EOI**	60 μs	-	-
T_{TK}	zmena z role **TALKER** na rolu **LISTENER**	20 μs	30 μs	100 μs
T_{DC}	potvrdenie zmeny role na **TALKER**	0	-	-
T_{DA}	príprava zariadenia po **TURN AROUND**	80 μs	-	-
T_{NE}	odpoveď na prenos bez **EOI**	0	40 μs	200 μs

Postup pri komunikácií je opísaný na nasledujúcich stránkach. Je rozdelený do krokov, kde každý z nich je reprezentovaný ukážkou správania sa signálov s časovými obmedzeniami. Z dokumentu [22] sú odvodené správania jednotlivých popisovaných zariadení. Dvojicu s ňou tvorí ukážka záznamu komunikácie opísanej v príklade, ktorá bola vytvorená použitím existujúceho riešenia s počítačom **Commodore 64** s komunikáciou medzi nimi zaznamenanej logickým analyzátorom [24] s použitím softvéru **PulseView** zo stránky [25].

Krok č. 1, obr. 3.9, 3.10: Počítač ako **TALKER** prideľuje rolu **LISTENER** externému zariadeniu. Počítač signalizuje všetkým zariadeniam na zbernici nastavením signálu $\overline{ATN}$ na log. 0, že chce komunikovať. Všetky zariadenia na zbernici musia pozastaviť svoju činnosť. Zároveň počítač nastaví signál $\overline{CLK}$ na log. 0. V čase T_{AT} musia všetky pripojené zariadenia zareagovať na tento signál nastavením $\overline{DATA}$ na log. 0. Po prekročení maximálnej doby na reakciu počítač usudzuje, že **zariadenie je neprítomné**. Počítač následne nastaví $\overline{CLK}$ na log. 1, čo znamená, že je pripravený komunikovať. Na toto reagujú všetky zariadenia nastavením $\overline{DATA}$ na log. 1 v čase T_H. Po potvrdení počítač začína posielať dáta, na konci necháva $\overline{CLK}$ v log. 0. Čas na prípravu hodnoty na $\overline{DATA}$ je T_S, po ktorom sú validné po dobu T_V. Potvrdenie prijatia dát zariadenie oznámi nastavením $\overline{DATA}$ na log. 0 v čase T_F. Ak zariadenie neodpovie v stanovenom časovom intervale, nastane **chyba rámca**. Dáta, ktoré počítač odoslal boli ekvivalentné hodnote **0x28**, čo je príkaz **LISTEN DEVICE 8**. Takto je určený prijímateľ komunikácie. Signál $\overline{ATN}$ zostáva na hodnote log. 0. Následne počítač rovnakým spôsobom ako predtým (nastavenie $\overline{CLK}$ na log. 1, potvrdenie od zariadenia $\overline{DATA}$ na log. 1, prenos dát, potvrdenie prijatia dát zariadením pomocou $\overline{DATA}$ na log. 0) odošle hodnotu **0xF0**, čo znamená **OPEN FILE 0**. Až teraz počítač nastavuje $\overline{ATN}$ na log. 1 v čase T_R.

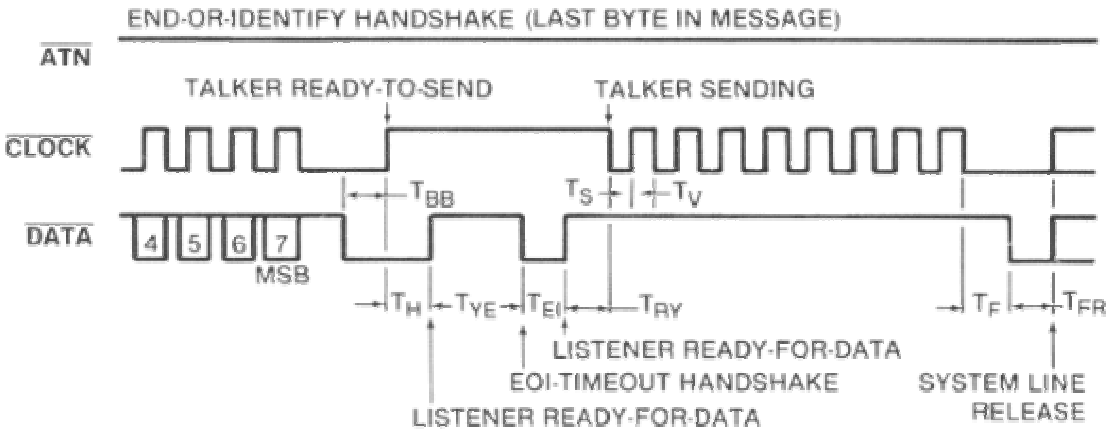


Obr. 3.9 Diagram signálov zbernice CBM/IEC; signalizácia $\overline{ATN}$ [22]

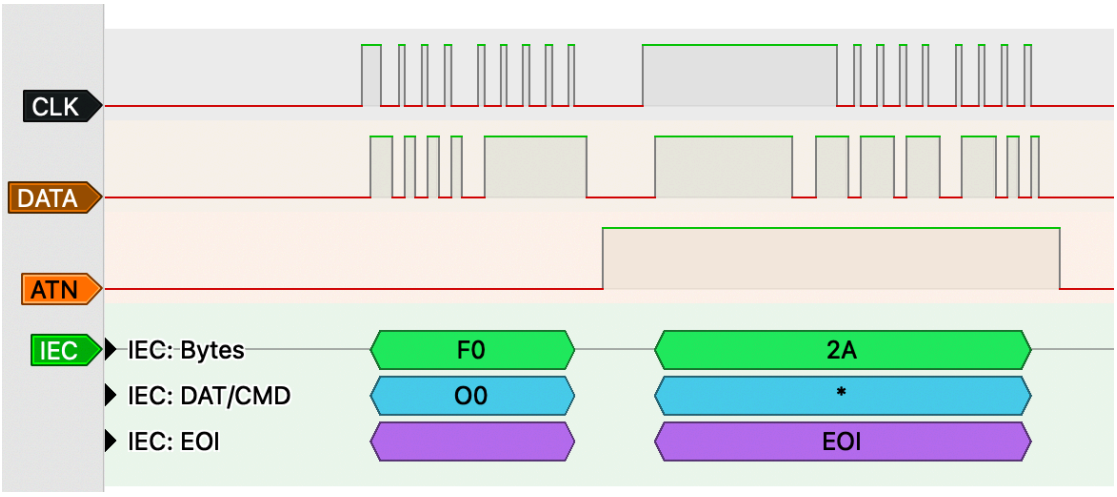


Obr. 3.10 Diagram signálov zbernice CBM/IEC; signalizácia $\overline{ATN}$ s príkazmi

Krok č. 2, obr. 3.11, 3.12: Počítač posiela dáta s názvom súboru. Keďže názov súboru je * (podľa ASCII tabuľky ekvivalentné hodnote 0x2A), potrebuje počítač signalizovať **koniec dát** (podobné EOF) nazývaný **End-Or-Identify**. Po predchádzajúcom prenose sú $\overline{CLK}$ a $\overline{DATA}$ nastavené na log. 0 a $\overline{ATN}$ na log. 1. Keďže si počítač v predchádzajúcom kroku zvolil **LISTENER**, ktorý načúva a reaguje na komunikáciu. Počítač signalizuje **EOI** nastavením $\overline{CLK}$ na log. 1 v čase T_{BB} od konca predchádzajúceho prenosu. Nastavenie tejto hodnoty naznačuje, že počítač je pripravený poslať dáta. V čase T_H musí zariadenie odpovedať nastavením $\overline{DATA}$ na log. 1, čo značí, že je pripravené prijať dáta. Počítač drží signál $\overline{CLK}$ v log. 1, čím naznačuje **EOI**. Zariadenie po čase T_{YE} odpovedá nastavením $\overline{DATA}$ na log. 0. Signalizuje tým, že rozpoznal **EOI**. Túto hodnotu musí udržať po dobu T_{EI}, ak je **LISTENER** externé zariadenie, mala by byť minimálna hodnota T_{EI} rovná 80 μs. Teraz zariadenie nastavuje $\overline{DATA}$ na log. 1, čím signalizuje pripravenosť prijímať dáta. V čase T_{RY} reaguje počítač začatím posielania dát. Odoslané dáta sú ekvivalentné názvu súboru. Po odoslaní posledného bitu nastavuje počítač $\overline{CLK}$ na log. 0 a v čase T_F potvrdzuje zariadenie prijatie dát nastavením $\overline{DATA}$ na log. 0. Nasledujúci prenos začína najskôr po čase T_{FR}.

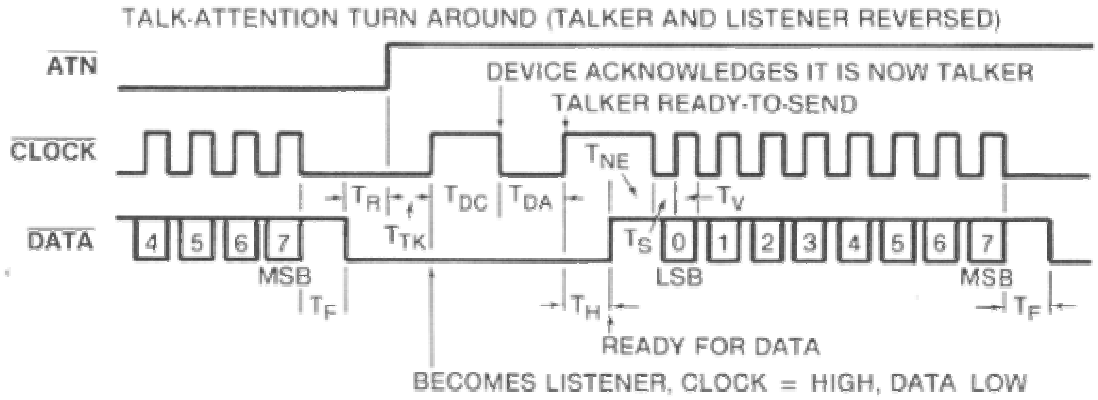


Obr. 3.11 Diagram signálov zbernice CBM/IEC; signalizácia EOI [22]

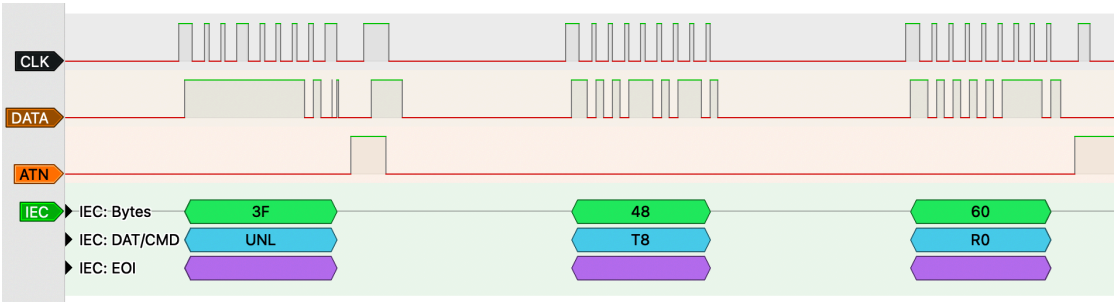


Obr. 3.12 Diagram signálov zbernice CBM/IEC; signalizácia EOI s dátami

Krok č. 3, obr. 3.13, 3.14: Počítač ako **TALKER** odníma rolu **LISTENER**, následne prideľuje rolu **TALKER** zariadeniu a on sám sa stáva **LISTENER**. Po ukončení prenosu dát s názvom súboru, počítač rovnakým spôsobom ako predtým (nastavenie $\overline{CLK}$ na log. 1, potvrdenie od zariadenia $\overline{DATA}$ na log. 1, prenos dát, potvrdenie prijatia dát zariadením pomocou $\overline{DATA}$ na log. 0) odosiela príkaz **UNLISTEN** s hodnotou **0x3F**. To znamená, že všetky zariadenia, ktoré boli v role **LISTENER**, strácajú svoju rolu. Následne počítač znova nastavuje $\overline{ATN}$ na log. 0, čím zaznačuje prichádzajúci príkaz. Podobne ako v kroku č. 1 odosiela príkaz **TALK DEVICE 8** s hodnotou **0x48**, čo znamená, že zariadenie s ID 8 sa stane **TALKER** a počítač sa stane **LISTENER**. Nasleduje príkaz **OPEN DATA 0**, hodnota **0x60**, čo značí, že počítač chce **čítať súbor typu PRG** zo zariadenia. Po potvrdení prijatia v čase T_F sa začína proces tzv. **TALK-ATTENTION TURN AROUND**. Akonáhle zariadenie potvrdí prijatie, v čase T_R počítač zmení hodnotu $\overline{ATN}$ na log. 1. Následne počítač nastavuje $\overline{CLK}$ v časovom intervale T_{TK} na log. 1 a $\overline{DATA}$ na log. 0, čím sa prakticky dostáva do role **LISTENER**. Zariadenie musí do T_{DC} potvrdiť svoju novú rolu **TALKER** nastavením signálu $\overline{CLK}$ na log. 0. Po potvrdení ponecháva zariadenie log. 0 aspoň T_{DA}. Zmenou $\overline{CLK}$ na log. 1 oznamuje nový **TALKER**, že je pripravený na prenos dát, čo ale spúšťa prenos dát.

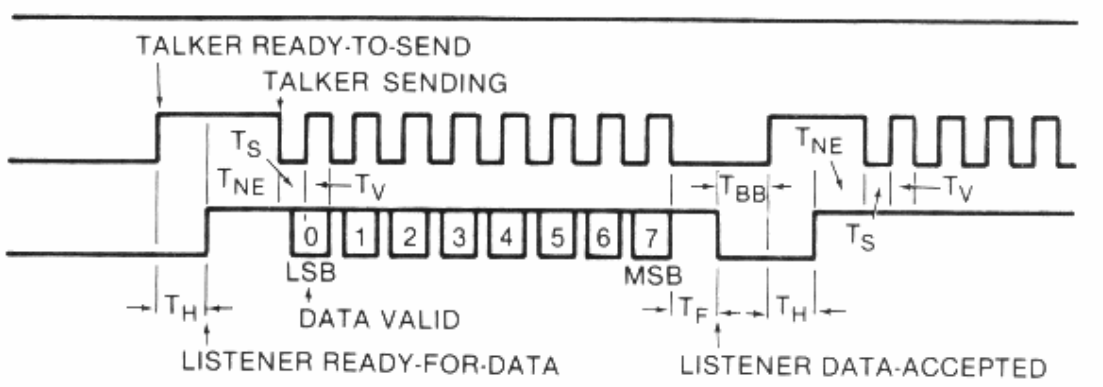


Obr. 3.13 Diagram signálov zbernice CBM/IEC; zmena rolí [22]

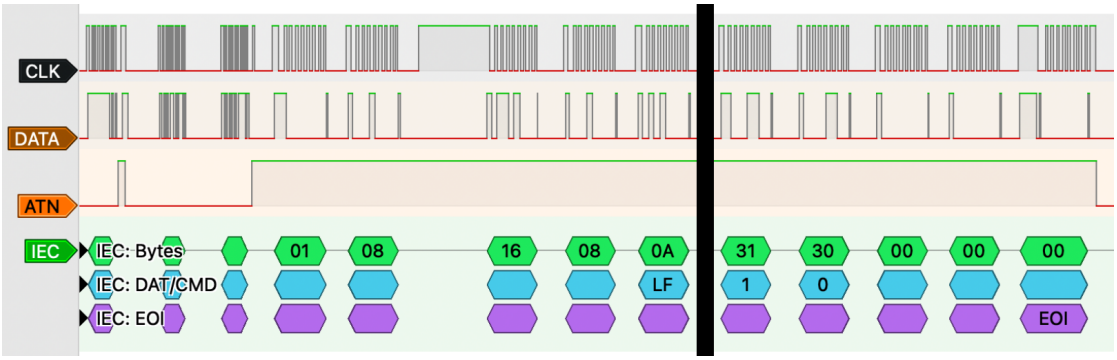


Obr. 3.14 Diagram signálov zbernice CBM/IEC; ukončenie načúvania a zmena rolí

Krok č. 4, obr. 3.15, 3.16: Zariadenie v roli **TALKER** odosiela dáta. Po vymenení rolí zariadenie oznamuje počítaču, že je pripravené odoslať dáta nastavením $\overline{CLK}$ na log. 1, na čo v čase T_H zareaguje počítač nastavením $\overline{DATA}$ do log. 1, čo znamená pripravenosť prijať dáta. Do času T_{NE} musí **TALKER** reagovať nastavením $\overline{CLK}$ na log. 0, inak značí **EOI**, čo vyžaduje špeciálnu reakciu počítača. Následne v intervaloch T_S pre prípravu hodnoty na $\overline{DATA}$ a T_V pre značenie validity dát na $\overline{DATA}$ zariadenie posiela 8 bitov. Na konci prenosu zariadenie nastavuje $\overline{CLK}$ na log. 0 a počítač potvrdzuje prijatie dát nastavením $\overline{DATA}$ na log. 0 v intervale T_F od poslednej zmeny $\overline{CLK}$. Takto sa opakuje prenos s medzičasom T_{BB} medzi bajtmi, až kým zariadenie neukončí prenos značením **EOI** a prenesením posledného bajtu tak ako je to opísané v kroku č. 2. Po dokončení prenosu nastavuje počítač signál $\overline{ATN}$ na log. 0, čím si vyžiada pozornosť.



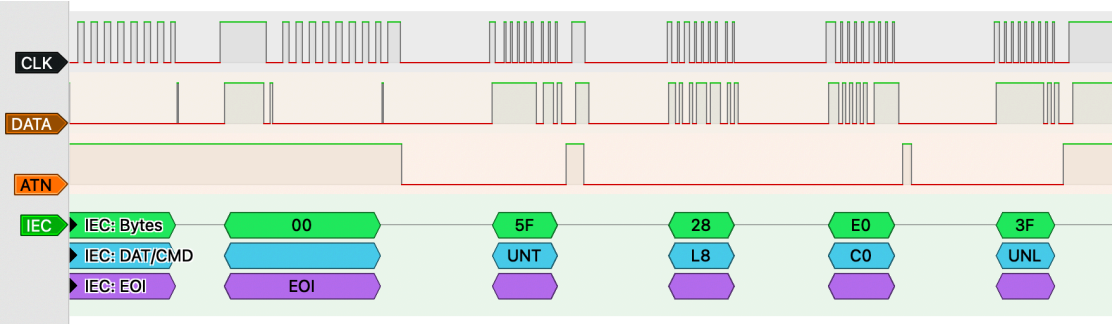
Obr. 3.15 Diagram signálov zbernice CBM/IEC; prenos bajtu [22]



Obr. 3.16 Diagram signálov zbernice CBM/IEC; prenos dátového súboru

Diagram 3.16 obsahuje čierny vertikálny blok, ktorý reprezentuje vynechanie prenosu niektorých bajtov, ktoré nie sú kritické pre túto postupnosť krokov. Tým pádom je možné použiť ukážku, ktorá je čitateľnejšia.

Krok č. 5, obr. 3.17: Počítač ukončuje komunikáciu. Po prenesení všetkých údajov zo zariadenia počítač nastavuje $\overline{ATN}$ na log. 0, čo signalizuje jeho záujem o odoslanie príkazu. Všetky zariadenia v akejkoľvek roli prestávajú komunikovať a načúvajú príkazu. Počítač odosiela podobným spôsobom ako v kroku č. 1 príkaz **UNTALK** s hodnotou **0x5F**. Tým oznamuje zariadeniu, ktoré bolo doteraz **TALKER**, že sa mu odníma táto rola. Následne počítač prideľuje znova rolu **LISTENER** zariadeniu príkazom **LISTEN DEVICE 8**, hodnota **0x28**, čím si v podstate počítač prideľuje rolu **TALKER**. Po priradení odosiela **CLOSE FILE 0** s hodnotou **0xE0**, čím prikazuje zatvorenie súboru, ktorý mu zariadenie donedávna posielalo. Ukončenie komunikácie prebieha odoslaním príkazu **UNLISTEN**, hodnota **0x3F**, a nastavením $\overline{ATN}$, $\overline{CLK}$ na log. 1, po prijatí nastavuje aj zariadenie $\overline{DATA}$ na log. 1, čím s zbernica vracia do počiatočného stavu. Časové intervaly sa zhodujú s tými, ktoré boli použité pri podobnom prenose v kroku č. 1.

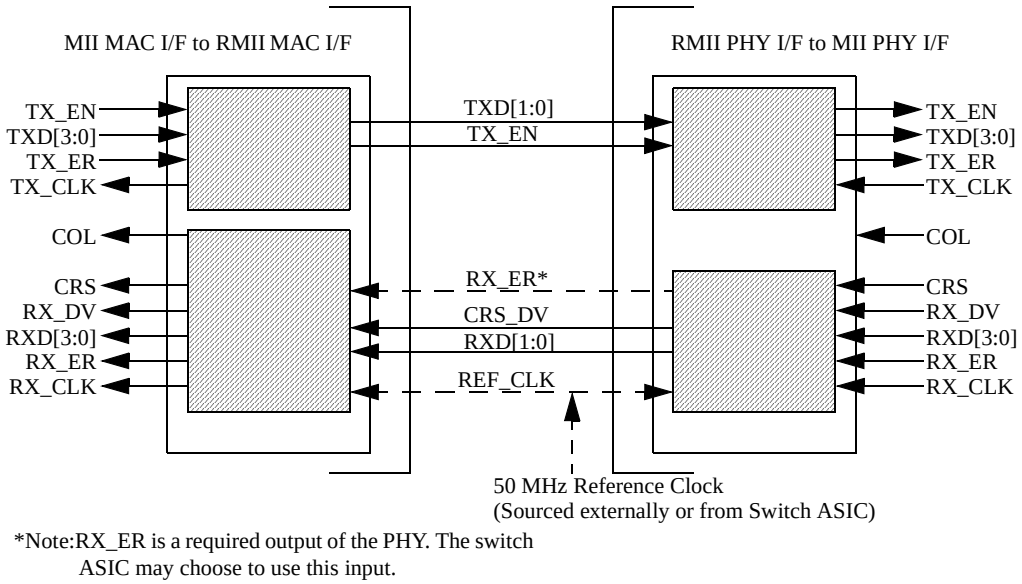


Obr. 3.17 Diagram signálov zbernice CBM/IEC; ukončenie komunikácie

Po zakončení prenosu na zbernici neprebiehajú ďalšie prenosy kým si ich počítač nevyžiada. Prejdením všetkými krokmi si počítač nahraje dáta z disku do RAM. Podobným spôsobom je počítač schopný zapísať dáta, ktoré má uložené v RAM na disk. Rozdiel je v tom, že na miesto príkazu **UNLISTEN** a **TALK DEVICE 8** v kroku č. 3, by odoslal príkaz **OPEN DATA 1**, čím signalizuje **zápis súboru typu PRG**. Pri prenose teda nenastane **TALK-ATTENTION TURN AROUND** a počítač zostáva po celú dobu v roli **TALKER** a zariadenie v roli **LISTENER**. Všetky časové intervaly sa aplikujú pri takom prenose rovnako ako v spomínanom riešení. Iné typy prenosov nebudú v tejto práci opísané, pretože sa vyskytujú buď ojedinele alebo využívajú dodatočné funkcie zariadení na zbernici.

3.2.2 Protokoly MII a RMII štandardu IEEE 802.3

V štandarde IEEE 802.3u je Media Independent Interface (MII) definované ako prepojenie medzi vrstvou Media Access Control (MAC) a Physical Layer (PHY) entitami a medzi PHY vrstvou a Station Management (STA) entitami. Rozhranie zároveň podporuje prenosové rýchlosti 10 Mb/s a 100 Mb/s. Fyzicky sa MII skladá z 18 signálových vodičov, kde **MDC** a **MDIO** uskutočňujú komunikáciu medzi PHY a STA, vodiče **TX_ER**, **TXD<3:0>**, **TX_EN**, **TX_CLK** sú využívané pre odosielanie dát na PHY a vodiče **RX_ER**, **RXD<3:0>**, **RX_DV**, **RX_CLK** pre prijímanie dát od PHY. Signál **COL** upozorňuje, ak nastane kolízia a signál **CRS**, že médium sa nenachádza v nečinnom stave, oba sú ovládané PHY. [26]

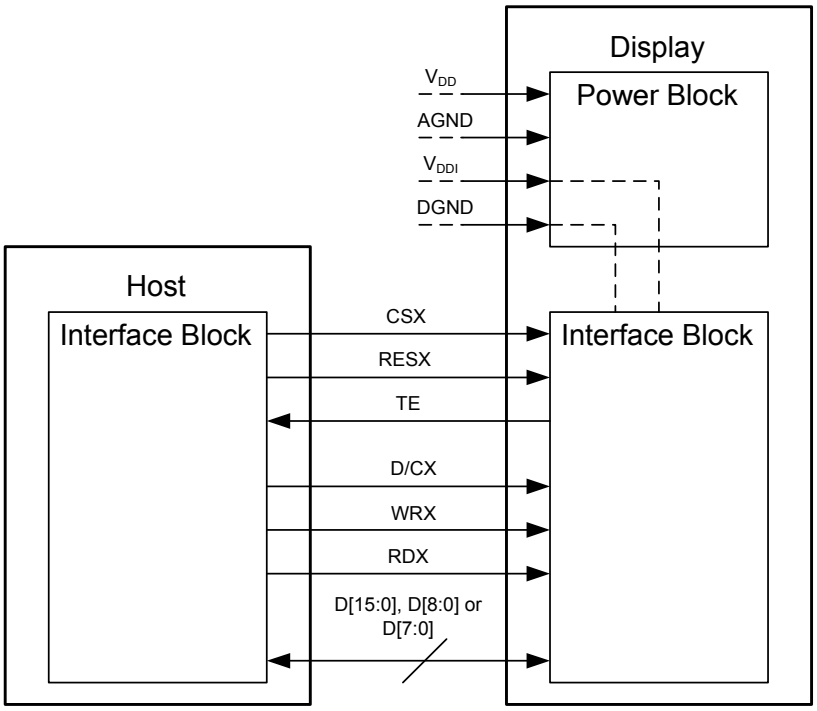


Obr. 3.18 Reprezentácia RMII prepojenia MAC a PHY [27]

Keďže MII si vyžaduje vysoký počet signálnych vodičov, z ktorých je možné zdieľať len dva s viacerými PHY, bola vypracovaná špecifikácia RMII. Tento protokol má byť lacnejšia alternatíva MII a týka sa výlučne dátovej a kontrolnej časti, správcovská časť (MDC a MDIO) zostáva rovnaká. Podporuje rýchlosti prenosu 10 Mb/s a 100 Mb/s a ako zdroj referenčného hodinového signálu sa používa buď MAC alebo externý vstup a je spoločný pre MAC a PHY. Na dátový prenos sa využívajú dva bity široké zbernice, jedna pre každý smer. [27] Počet signálových vodičov je teda zredukovaný z pôvodných 16 u MII na 7 u RMII (8, ak sa použije **RX_ER**, ktorý je definovaný ako nepovinný). Ukážkové prepojenie MAC a PHY je uvedené na obr. 3.18, výraz „switch ASIC“ je možné zameniť s výrazom „MAC“.

3.2.3 Protokol DBI aliancie MIPI

Štandard Mobile Industry Processor Interface (MIPI) Display Bus Interface (DBI) je definícia rozhrania určeného pre displej moduly. DBI je konfigurovateľné, existuje možnosť použiť 1, 2, 8, 9 alebo 16 dátových signálov. Elektrické prepojenie medzi hostiteľským procesorom a displej modulom sa skladá z dvoch blokov, napájanie a rozhranie. Bloky rozhrania sú určené na prenos informácií medzi hostiteľským procesorom a displej modulom. Definované sú tri typy implementácie DBI nazvané A, B a C. Vodiče **RESX**, **CSX**, **D/CX**, **WRX** a **RDX** slúžia na obsluhu modulu pričom zbernica **D[7:0]** je určená pre obojsmerný prenos dát. Vodič **TE** je niekedy používaný ako identifikátor, že na displeji nastal tzv. tearing effect. [28] Pretože displej modul, podľa oficiálnej dokumentácie čip setu [29], využíva typ B rozhrania DBI s 8 dátovými vodičmi, nebudú typy DBI A a C ďalej v tejto práci popisované. Diagram na obr. 3.19 určuje spôsob prepojenia hostiteľskej časti a modulu.

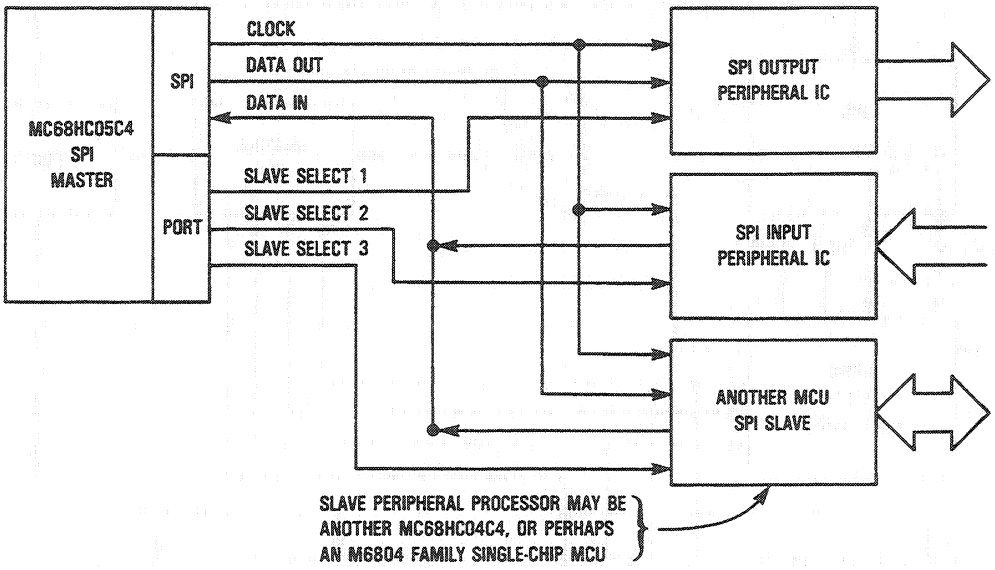


Obr. 3.19 Blokový diagram MIPI DBI Type B [28]

Podobne je nutné pripojiť displej modul zvolený v sekcii 3.1.3, aj keď podľa údajov uvedených v schéme PCB modulu [30], je zjavné, že rozhranie DBI je typu A, keďže vodič **TE** nie je vyvedený. Jeho vynechanie by ale nemalo mať zásadný vplyv na funkčnosť modulu.

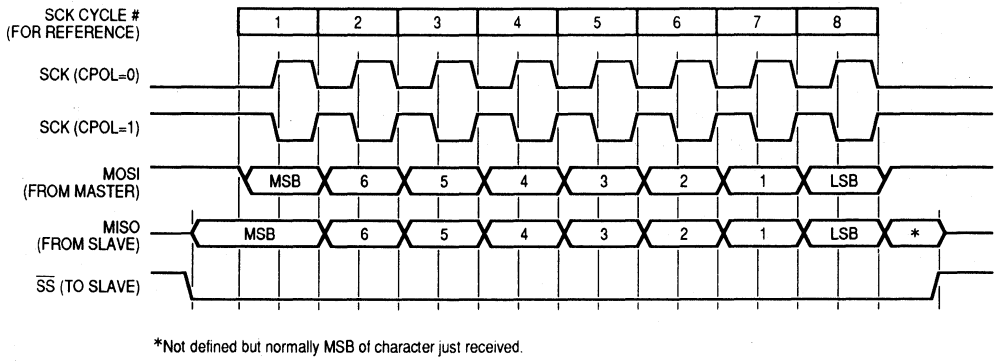
3.2.4 Protokol SPI spoločnosti Motorola

Serial Peripheral Interface (SPI) je sériové rozhranie primárne určené na umožnenie komunikácie mikrokontroléra s perifémymi zariadeniami. Použitím SPI je umožnená aj komunikácia medzi viacerými procesormi v systéme multiple-master. Počas dátového prenosu sú dáta odosielané (posúvané sériovo von) a zároveň prijímané (posúvané sériovo dnu). [31] Ukážková realizácia prepojenia zariadení typu jeden nadriadený (Single Master) je viditeľná na obr. 3.20.



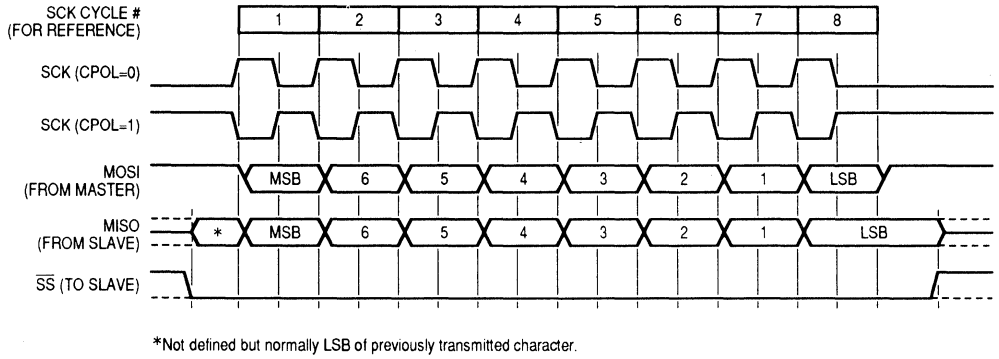
Obr. 3.20 Diagram prepojenia nadriadeného a podriadených zariadení SPI [32]

Hodinový signál **System Clock (SCK)** synchronizuje prenosy na dvoch dátových prenosových vodičoch **Master In/Slave Out (MISO)** a **Master Out/Slave In (MOSI)**. Pomocou signálu **Slave Select (SS)** je umožnená presná voľba podriadeného (slave) zariadenia, s ktorým chce nadriadený (master) komunikovať. Periférne zariadenia, ktoré nie sú zvolené nezasahujú do prebiehajúceho prenosu. Na strane nadriadeného je možné pomocou signálu **SS** naznačiť, že pripojenie je typu multiple-master. SPI je schopné podľa nastavení polarity hodinového signálu **CPOL** a fázy hodinového signálu **CPHA** prenášať dáta 4 rôznymi spôsobmi, inak nazvané módy. [31]



Obr. 3.21 Diagram prenosu SPI s fázou hodinového signálu rovnej 0 pre obe polarity [31]

Diagram ukážkového dátového prenosu medzi nadriadeným a podriadeným zariadením je zobrazený na obr. 3.21 so vstupným prenosom na nábežnú hranu a výstupným prenosom na dobežnú hranu s oboma polaritami hodinového signálu a na obr. 3.22 s vstupným prenosom na dobežnú hranu a výstupným prenosom na nábežnú hranu s oboma polaritami hodinového signálu.



Obr. 3.22 Diagram prenosu SPI s fázou hodinového signálu rovnej 1 pre obe polarity [31]

Prenos na SPI začína vždy signalizáciou pomocou SS, ktoré podriadené zariadenie má prijímať alebo posielať dáta. Prvým vystaveným je najvýznamnejší bit (MSB) nadriadeným alebo podriadeným zariadením. Údaje sú prenášané postupne, až kým nie je prenesený posledný bit (LSB). Podľa nastavenej bitovej šírky SPI sa prenáša 8 alebo viac bitov.

Existujú viaceré rozšírenia SPI, napr. Dual SPI, Quad SPI alebo QPI, ktoré upravujú dátové signály na obojsmerný prenos dát a rozširujú počet dátových vodičov, čím je možné dosiahnuť väčšiu šírku prenosového pásma. Tieto verzie nie sú oficiálne definované, ale sú bežne používané.

3.3 Zhrnutie

Po preskúmaní dostupných riadiacich prvkov a periférií boli ako súčasť budúceho prototypu zvolené tieto komponenty:

- Digilent Cmod A7-35T prípravok s čipom FPGA spoločnosti Xilinx, model Artix-7
- LCD TFT displej s uhlopriečkou 3,5“, rozlíšením 480x320, rozhraním MIPI DBI spolu s SD kartou a rozhraním SPI
- Ethernet modul LAN8720 typu Fast Ethernet s rozhraním RMII
- Prevodník UART – RS-232

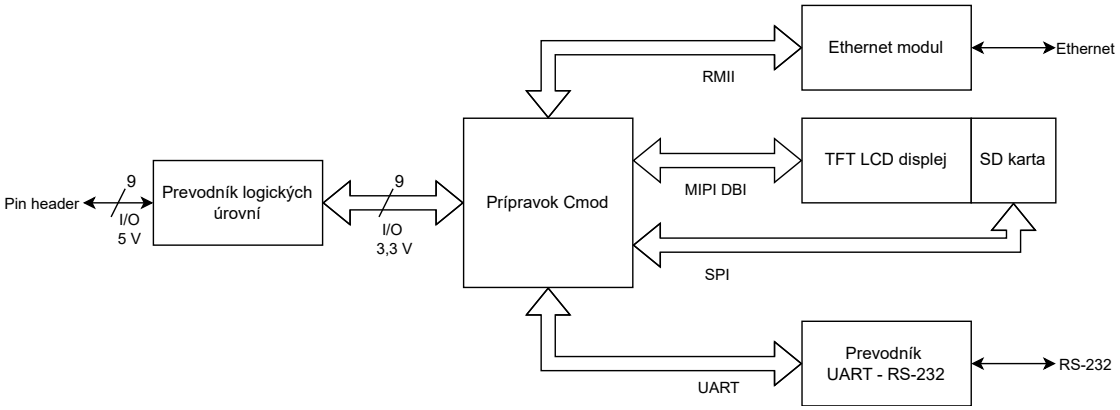
Spomínané komponenty budú integrované na doske PCB spolu s prevodníkom logických úrovní postavenom na N-kanálovom MOSFET. Analýza protokolov (SPI, MIPI DBI, RMII, CBM/IEC) umožní jednoduchší prístup k implementácií v softvérovej časti práce.

Kapitola 4
Návrh zariadenia

Kapitola sa venuje návrhu hardvérovej a softvérovej časti práce, vytvoreniu blokovej schémy prototypu spolu s plánom na rozmiestnenie jednotlivých komponentov na doske PCB, preskúmanie existujúcich IP jadier použiteľných pri práci a rozdelenie softvérových komponentov s analýzou existujúcich knižníc a modulov.

4.1 Návrh hardvérovej časti

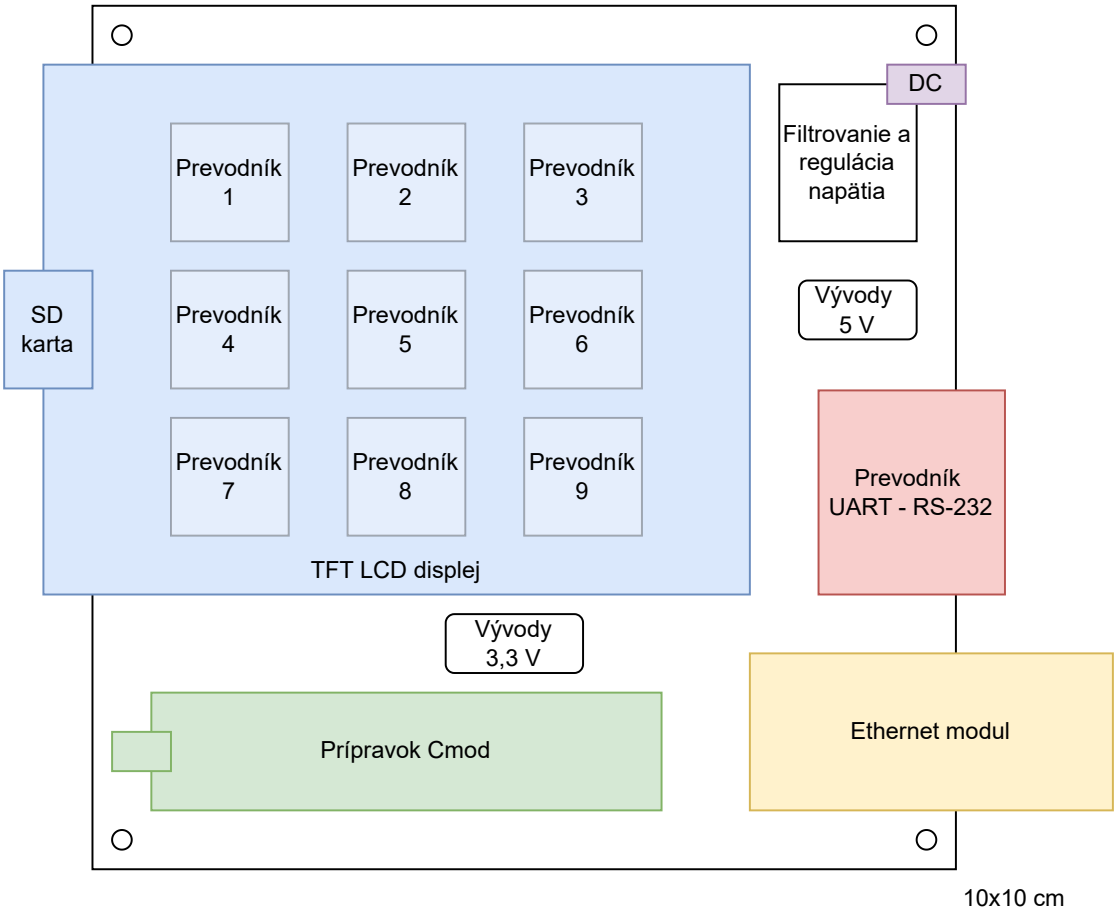
Hardvérová časť práce rieši prepojenie jednotlivých komponentov vybraných v analytickej časti práce pomocou protokolov, ktoré boli rovnako analyzované v kapitole 3. Použitím ovládacieho prvku je z väčšej časti určené rozloženie následnej blokovej schémy 4.1. Prípravok Cmod A7-35T ovláda všetky pripojené periférie, každú pomocou špecifického protokolu. Preto je potrebné neskôr využiť vlastnosť obvodu FPGA použiť IP jadrá, ktoré riešia komunikáciu cez potrebné protokoly. V schéme sú reprezentované svojimi názvami a prepájajú prípravok s modulom. Poslednou časťou sú vývody pre konfigurovateľné rozhranie, ktoré je potrebné previesť cez prevodník logických úrovní. Počet signálov použitých pre takéto rozhranie je 9, keďže väčšina zariadení, na ktoré sa prototyp bude zameriavať, si vyžaduje maximálne 9 vodičov, čo vyplýva z kapitoly 3.



Obr. 4.1 Bloková schéma návrhu hardvérovej časti

Pretože má prípravok Cmod vyvedených 48 pinov (z toho 46 použiteľných na komunikáciu), ostane po pripojení všetkých periférií dostupných 8, z nich 2 vstupy A/D prevodníka. Tieto signály sú vyvedené na tzv. **header** a môžu byť použité na rozšírenie systému o ďalšie periférie.

Rozmiestnenie jednotlivých komponentov sa riadi orientáciou konektorov spomínaných modulov. Preto ich optimálne rozmiestnenie pri štvorcovom pôdoryse **10x10 cm** je na obr. 4.2. Všetky konektory sú orientované od dosky PCB, prevodníky logickej úrovne sú umiestnené pod displejom, napájacia časť je umiestnená na voľnom mieste v pravom hornom rohu. Podľa nákresu bude realizovaná doska PCB.

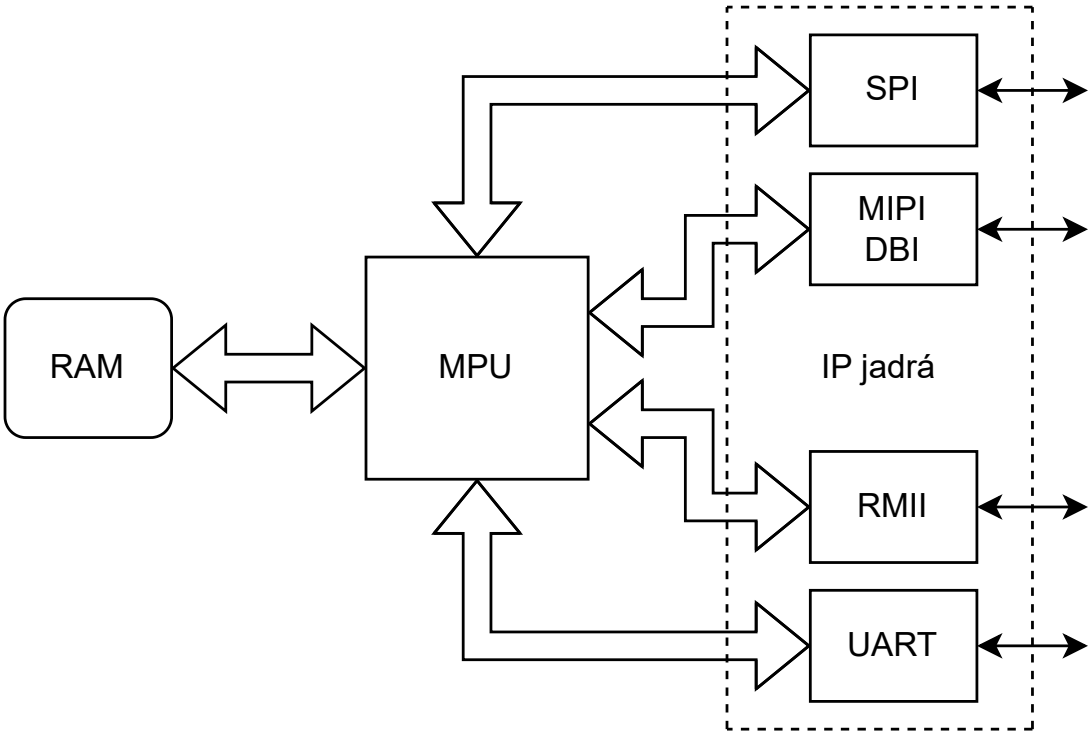


Obr. 4.2 Schéma návrhu rozmiestnenia komponentov na doske

Konfigurovateľné rozhranie je realizované rovnako ako dodatočné vývody a to použitím tzv. **pin header**. Tento typ vývodu sa nazýva taktiež **Berg** alebo **DuPont** [33]. Najpoužívanejší rozmer je 2,54 mm medzi jednotlivými vývodmi. V tomto prípade je potrebné vyviesť 9+1 signálov pre konfigurovateľné rozhranie (DATA0-8 + GND) a zvyšných 8+4 v konfigurácií I/O0-5, ADC0, ADC1 + 2xGND, 5V, 3V3, čo dáva možnosť využiť prototyp aj ako zdroj napätia pre periférie.

4.2 Návrh schémy pre čip FPGA

Základnou vlastnosťou obvodu FPGA je možnosť programovania. Preto je možné vytvoriť veľké množstvo návrhov, ktoré je potom možné vložiť do takéhoto čipu. Jeden z nich bude súčasťou tejto práce. Na začiatku je dobré vytvoriť návrh, na základe ktorého bude možné v ďalšom kroku založiť blokovú schému pre spomínaný obvod. Podľa vytvoreného návrhu v sekcii 4.1 bola vytvorená jednoduchá bloková schéma 4.3, ktorá reprezentuje požadovanú funkcionalitu obvodu. Základným prvkom každého návrhu musí byť procesor, v tomto prípade **MPU**. Ten si vyžaduje pre správne fungovanie prístup do pamäte **RAM**. Táto pamäť môže byť interná (realizovaná v čipe FPGA) alebo externá (fyzický čip SRAM/DRAM pripojený k obvodu FPGA). Pretože moduly komunikujú pomocou protokolov, sú v blokovej schéme vytvorené tzv. **IP jadrá**, ktoré realizujú spomínanú komunikáciu a tým pádom riešia stránku protokolov. Jadrá komunikujú s procesorom pomocou spoločnej zbernice ako je napr. **AXI**.

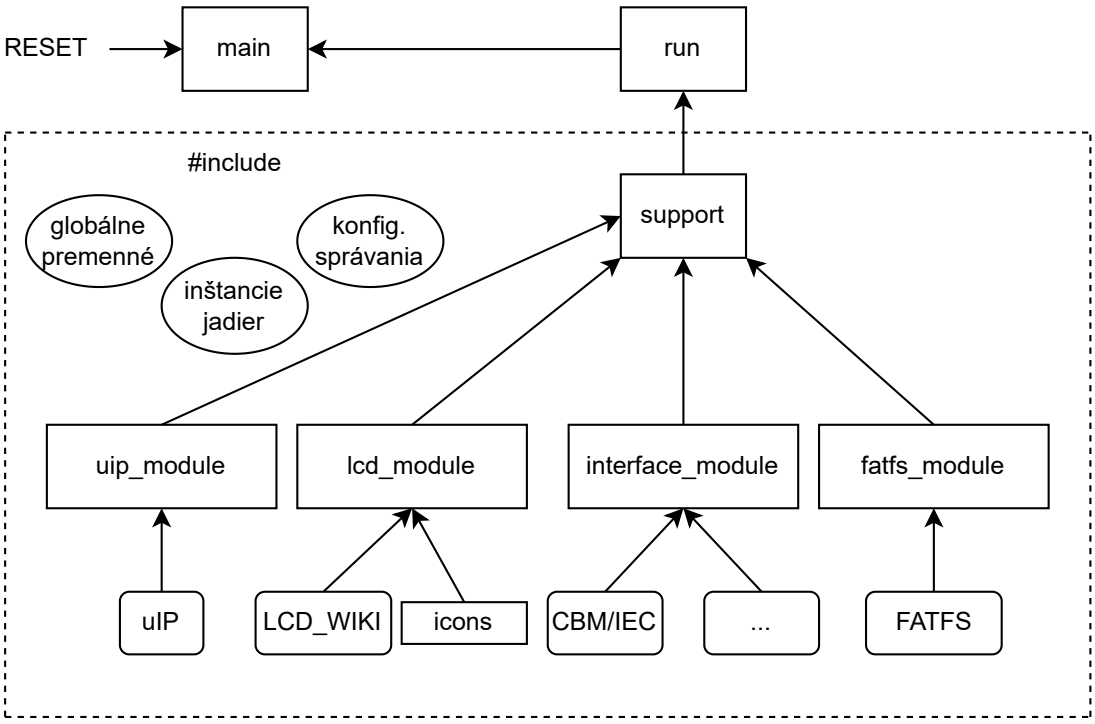


Obr. 4.3 Bloková schéma návrhu interného riešenia pre čip FPGA

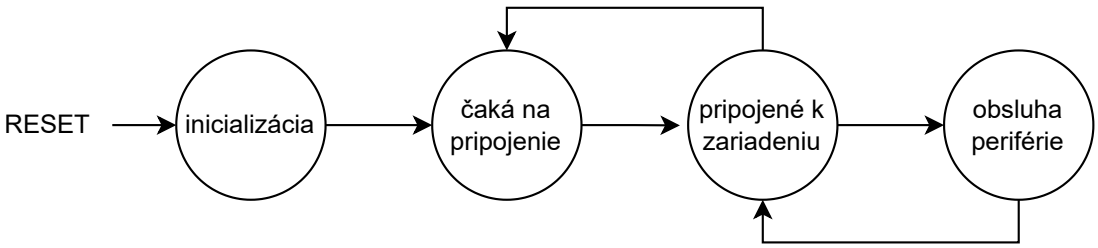
Takto navrhnutý systém je dobrým základom pri následnom vytváraní blokovej schémy pre obvod FPGA. Neobsahuje ale ďalšie dôležité prvky ako generátor hodinového signálu, správcu prerušení a časovač. Tieto bloky je potrebné, resp. dobré pridať do schémy, aby mohol byť systém použiteľný.

4.3 Návrh softvérovej časti

Softvérová časť práce reflektuje na organizáciu súborov, funkcií, inštancií a premenných, ktoré v celku tvoria aplikáciu. Schéma 4.4 reprezentuje takéto usporiadanie, kde **main** a **run** obsahujú základ aplikácie. Z nich sa volajú funkcie zo **support**, kde sa nachádzajú obsluhy pre inicializáciu zariadenia a pre obsluhu. Ďalej sú tu obsiahnuté globálne premenné, inštancie jednotlivých jadier a konfiguračné údaje definujúce správanie aplikácie. Tie sú rovnako použité aj v ďalších častiach ako sú jednotlivé moduly podľa toho, či sú tam vyžadované. Tieto časti majú zabezpečiť obsluhu jednotlivých periférií systému využívaním inštancií a existujúcich riešení ako μ**IP**, **LCD_WIKI** alebo **FatFs**. Funkcie sú volané zo **support**, ktorý realizuje konečný automat na základe vstupov z modulov podľa obr. 4.5. Výsledná aplikácia bude postavená podľa tejto schémy.



Obr. 4.4 Bloková schéma návrhu softvérovej časti



Obr. 4.5 Schéma návrhu konečného automatu

Kapitola 5
Realizácia hardvérovej časti

Kapitola sa venuje riešeniu hardvérovej časti práce, vytvoreniu schémy prepojenia komponentov s napájacou časťou a konfigurovateľného rozhrania spolu s vytvorením jeho proprietárnej realizácie, prevodu schémy do CAD modelu dosky plošného spoja, vymedzeniu pozície použitých periférií a súčiastok a následnej tvorbe fyzickej PCB a výsledného prototypu. Koniec kapitoly sa venuje realizácií blokovej schémy, jej prevodu do jazyka VHDL a definícii constraints.

5.1 Tvorba schémy prepojenia

Na vytvorenie schémy prepojenia sa využíva softvérový balíček Autodesk Eagle, ktorý je v dnešnej dobe súčasťou Fusion 360. Pre väčšiu kompatibilitu a taktiež na základe doterajších skúseností bude použitá verzia Autodesk EAGLE 9.6.2 [34].

Pri tejto práci je potrebné do schémy zaradiť vývody typu **FE12-1** (dutinková lišta s rozstupom 2,54 mm) pre prípravok Cmod A7-35T (označenie SV1-SV4), displej modul (SV8-SV11) a prevodník UART – RS-232 (SV6). Pre Ethernet modul (SV5) je potrebné použiť dvojradový konektor. Každý z modulov má na konektory vyvedené napájanie (+5 V, resp. +3,3 V a GND) a dátové vodiče, ktoré sú v schéme združené do **Bus** pre lepšiu čitateľnosť. Označenia napovedajú ich určeniu, napr. **ETH_RMII** zlučuje všetky signály potrebné pre protokol RMII.

Ako vývody konfigurovateľného rozhrania a zvyšných nepoužitých portov použijeme typ **MA05-2** (SV7), resp. **MA06-2** (SV12) (kolíky s rozstupom 2,54 mm), ku ktorým bude jednoduché následne pripojiť potrebné vodiče. U konektoru s označením SV7 je spolu s dátovými signálmi vyvedená sieť GND, u SV12 sú vyvedené aj napätia +5 V a +3,3 V.

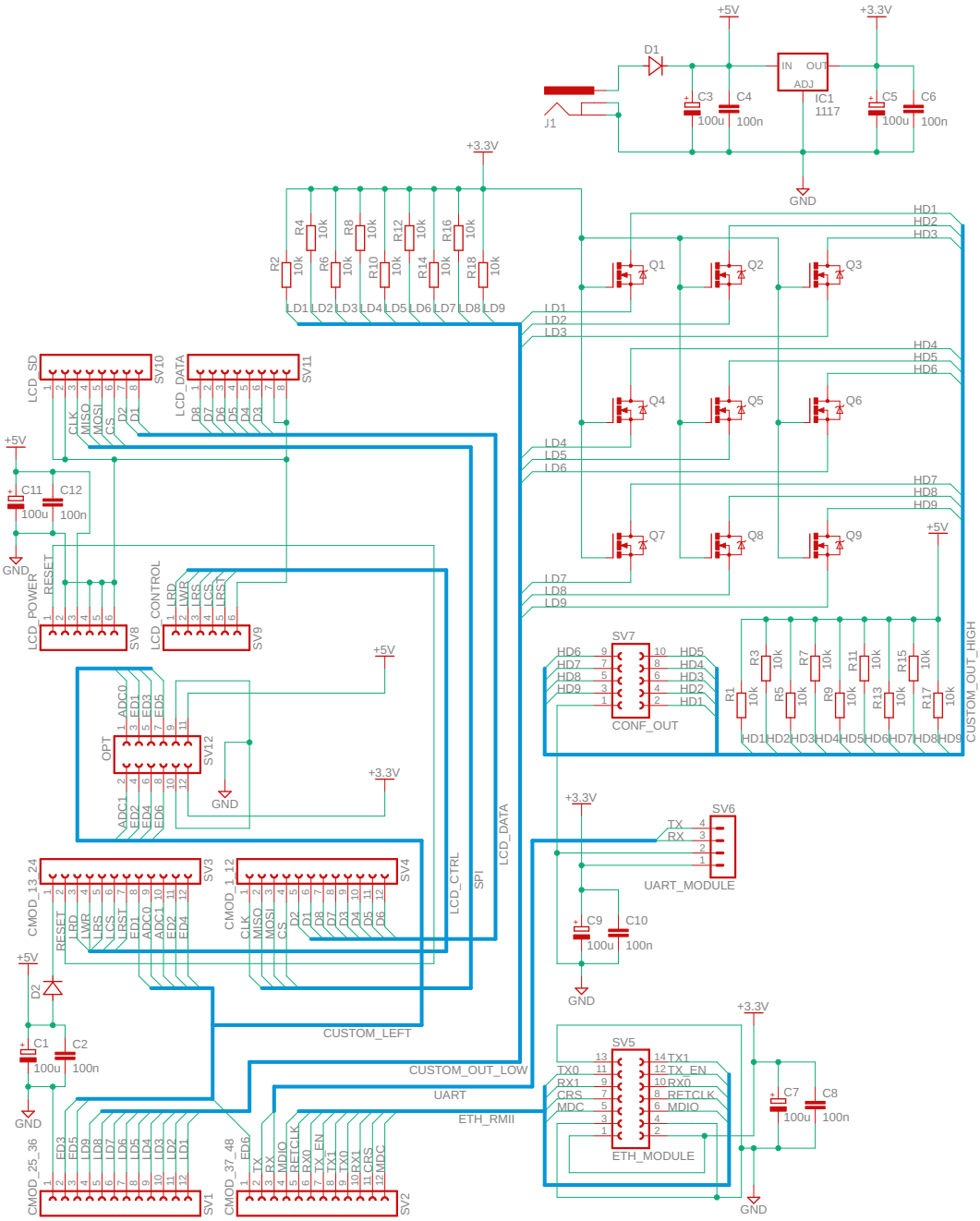
Potreba komunikácie na logickej úrovni 5 V si vyžaduje použitie prevodníkov logickej úrovne, ktoré sa skladajú z N-kanálových unipolárnych tranzistorov MOSFET, typ **BSS138-7-F** (Q1-Q9) v púzdre SOT-23 a pomocných rezistorov **SMD** rozmeru **R0805** s hodnotou 10 kΩ (R1-R18), ktoré sú pripojené k obom stranám signálov spôsobom pull-up.

Súčasťou schémy je regulátor napätia potrebný pre napájanie komponentov na logickej úrovni 3,3 V. Hlavnú časť predstavuje integrovaný obvod **LM1117** (IC1) v púzdre SOT-223. Potrebné sú aj polarizované elektrolytické kondenzátory **THT** s rozstupom **2,5 mm**, priemerom 6 mm (C1, C3, C5, C7, C9, C11) a keramické kondenzátory **THT** s rozstupom **5 mm** (C2, C4, C6, C8, C10, C12), ktoré sú rozmiestnené tak, aby mal každý komponent (Cmod, LCD displej, Eth modul, UART modul) vlastný pár. Podľa odporúčania sú dva páry umiestnené aj pri regulátore napätia (C3, C4 a C5, C6).

Napájanie je zabezpečené cez konektor **POWER_JACK** (J1) s centrálnym pozitívnym kolíkom, na ktorom je očakávané napätie max. 5,5 V. Ako ochrana proti reverznej vstupnej polarite

je použitá Schottky dióda **MBRA340T3** (D1) v púzdre SMA. Na ochranu proti spätnému toku prúdu pri zapojení Cmod pomocou USB je pridaná ekvivalentná dióda (D2) na vývode **VU**.

Výsledná schéma je na obr. 5.1. **CUSTOM_LEFT**, **CUSTOM_OUT_HIGH**, **CUSTOM_OUT_LOW**, **ETH_RMII**, **LCD_CTRL**, **LCD_DATA**, **SPI** a **UART** sú dátové signály združené v zbernici spomínaného typu **Bus**. Jednotlivé signály sú označené v mieste zakončenia.

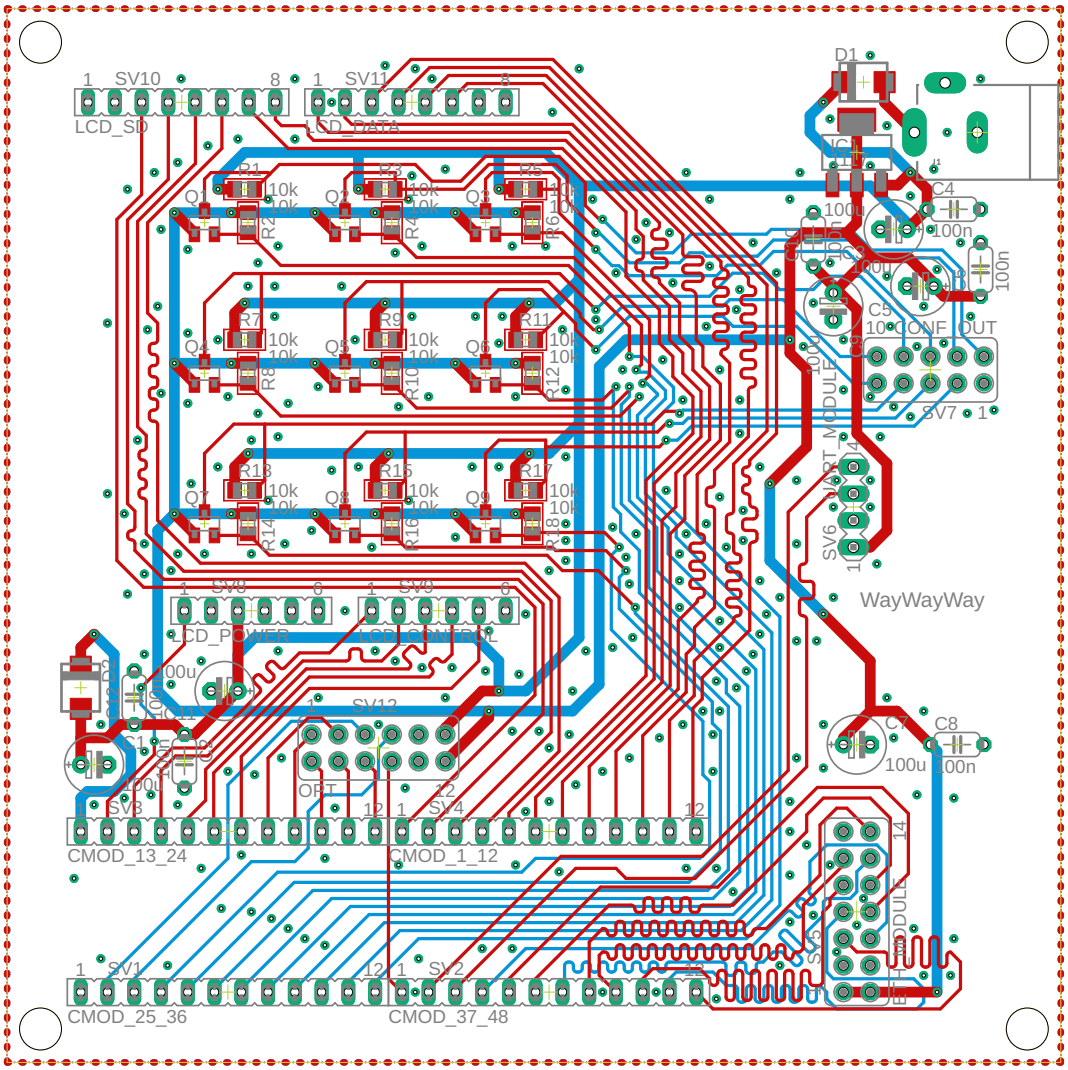


Obr. 5.1 Schéma prepojenia modulov prototypu s podpornými súčiastkami

5.2 Realizácia dosky plošného spoja

Zo schémy je potrebné vytvoriť CAD realizáciu dosky plošného spoja. Po prevode je nutné rozmiestniť všetky konektory, vývody a súčiastky na plochu dosky tak, aby neboli v konflikte. Ďalej sa musí dbať na rozmery modulov, ktoré sa budú k vývodom pripájať. Základom je doska s rozmerom **10x10 cm**, tento rozmer bude považovaný za maximálny z cenových dôvodov. Kvôli tomuto obmedzeniu nie je možné rozložiť komponenty tak, aby boli všetky plne obsiahnuté rozmerovo na PCB. Moduly Ethernet a UART budú preto posunuté mierne mimo dosku, hlavne čo sa týka konektorov DB9 a RJ45.

Základnou myšlienkou pri rozvrhnutí pozície všetkých častí prototypu je ich fyzická dostupnosť, keďže všetky z nich obsahujú časti, ku ktorým sa budú pripájať vodiče a dátové úložisko alebo sa bude cez nich ovládať výsledné zariadenie.



Obr. 5.2 CAD návrh rozloženia komponentov na doske plošného spoja

Výsledná návrh je vykreslený na obr. 5.2. Rozmiestnenie komponentov je z tohto obrázku zjavné, preto nie je potrebné ho ďalej slovne opisovať.

Na základe viacerých riešení rozloženia komponentov bolo zvolené jedno, ktoré bude použité v tejto práci. Podstatnou časťou je dodatočné obmedzenie použiteľnej plochy na 9,5x9,5 cm a následné vytvorenie montážnych dier s priemerom **158 mil** (4.0132 mm) vo vzdialenosti **125 mil** (3,17 5mm) od každého rohu dosky.

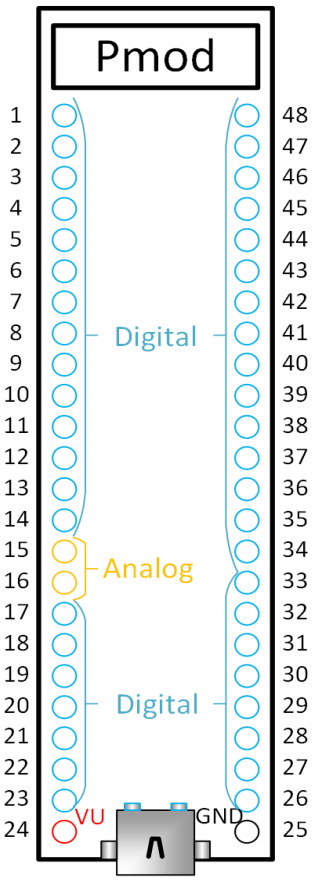
Napájanie všetkých častí je vedené cestami o šírke **40 mil** (1,016 mm) od napájacieho konektora alebo od regulátora napätia. Každý modul má pri konektore umiestnený pár kondenzátorov čo v najmenšej vzdialenosti. Dátové signály sú vedené cestami o šírke **12 mil** (0,3048 mm).

Konektor pre Ethernet modul je prioritne umiestnený čo najbližšie k modulu Cmod z dôvodu dodržania nízkych dĺžok signálových vodičov. Pri návrhu bol použitý nástroj **Meander** na dosiahnutie pomerne rovnakých dĺžok signálov, čo si znova vyžaduje protokol RMII.

Pre zníženie pravdepodobnosti, že podobný problém pri rozdieli dĺžok vodičov bude vplývať na spoľahlivosť komunikácie bol nástroj použitý u všetkých dátových signálov okrem konfigurovateľného rozhrania, kde sa neočakávajú vysoké prenosové rýchlosti. Medzi a okolo dátových signálov boli pridané tzv. **Vias**, ktorými sa zaručuje prepojenie oboch strán PCB a ich uzemnení.

Po pripojení všetkých konektorov bolo pomocou nástroja **Ratsnest** vytvorené uzemnenie na oboch stranách dosky, cez ktoré sú uzemnené všetky komponenty. Po tomto kroku sa stáva CAD návrh nečitateľný, preto je na obr. 5.2 verzia pred týmto krokom.

FPGA pin	PCB pin	
M3	LCD_D6	1
L3	LCD_D5	2
A16	LCD_D4	3
K3	LCD_D3	4
C15	LCD_D7	5
H1	LCD_D8	6
A15	LCD_D1	7
B15	LCD_D2	8
A14	SD_CS	9
J3	SD_MOSI	10
J1	SD_MISO	11
K2	SD_CLK	12
L1	OPT_ED4	13
L2	OPT_ED2	14
G2/G3	OPT_ADC1	15
J2/H2	OPT_ADC0	16
M1	OPT_ED1	17
N3	LCD_RST	18
P3	LCD_CS	19
M2	LCD_RS	20
N1	LCD_WR	21
N2	LCD_RD	22
P1	RESET	23
VU	+5V	24

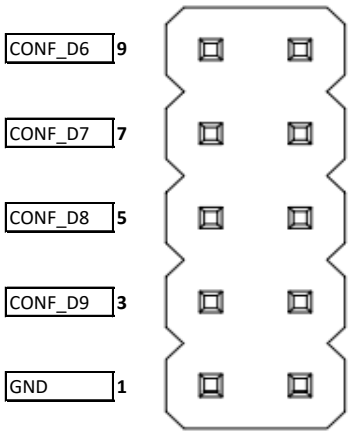


	PCB pin	FPGA pin
48	ETH_MDC	V8
47	ETH_CRS	U8
46	ETH_RX1	W7
45	ETH_TX0	U7
44	ETH_TX1	U3
43	ETH_TX_EN	W6
42	ETH_RX0	U2
41	ETH_RETCLK	U5
40	ETH_MDIO	W4
39	UART_RX	V5
38	UART_TX	U4
37	OPT_ED6	V4
36	CONF_D1	W5
35	CONF_D2	V3
34	CONF_D3	W3
33	CONF_D4	V2
32	CONF_D5	W2
31	CONF_D6	U1
30	CONF_D7	T2
29	CONF_D8	T1
28	CONF_D9	R2
27	OPT_ED5	T3
26	OPT_ED3	R3
25	GND	GND

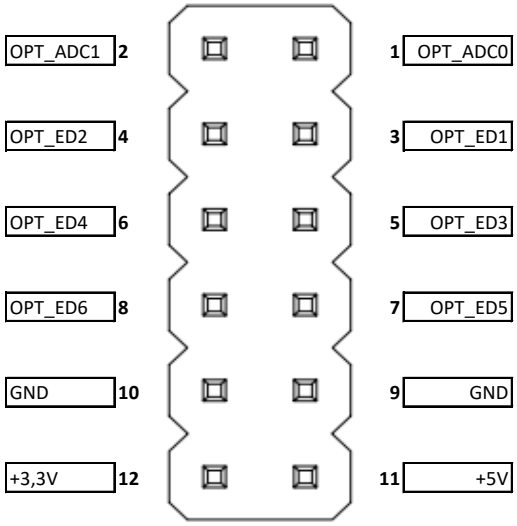
Obr. 5.3 Návrh pripojenia vývodov Cmod na doske PCB a čipe FPGA

Pripojenie dátových signálov na jednotlivé vývody konektorov je presne určené pre moduly LCD displej, Ethernet a UART – RS-232 prevodník. Čo sa týka modulu Cmod je rozloženie podrobne opísané na obr. 5.3.

Pre konektory s označením SV7 a SV12 je rozloženie podobne zaznačené na obr. 5.4, resp. na obr. 5.5. Toto vyvedenie je možné vyčítať aj zo schémy prepojenia, spomínané obrázky sú výňatkom učené pre lepšie pochopenie zapojenia.

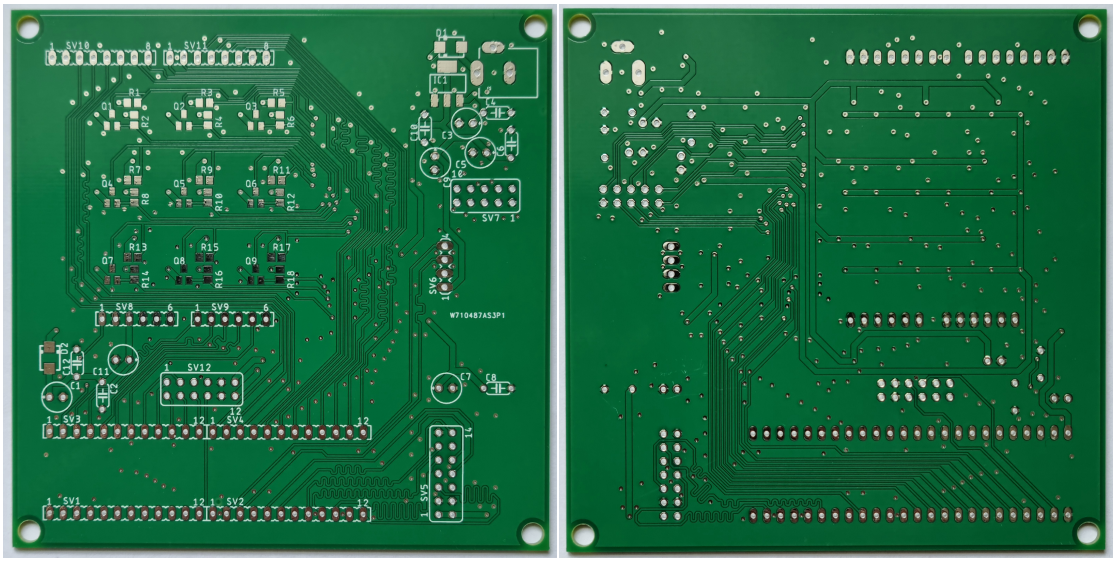


Obr. 5.4 Schéma vývodov konektora SV7

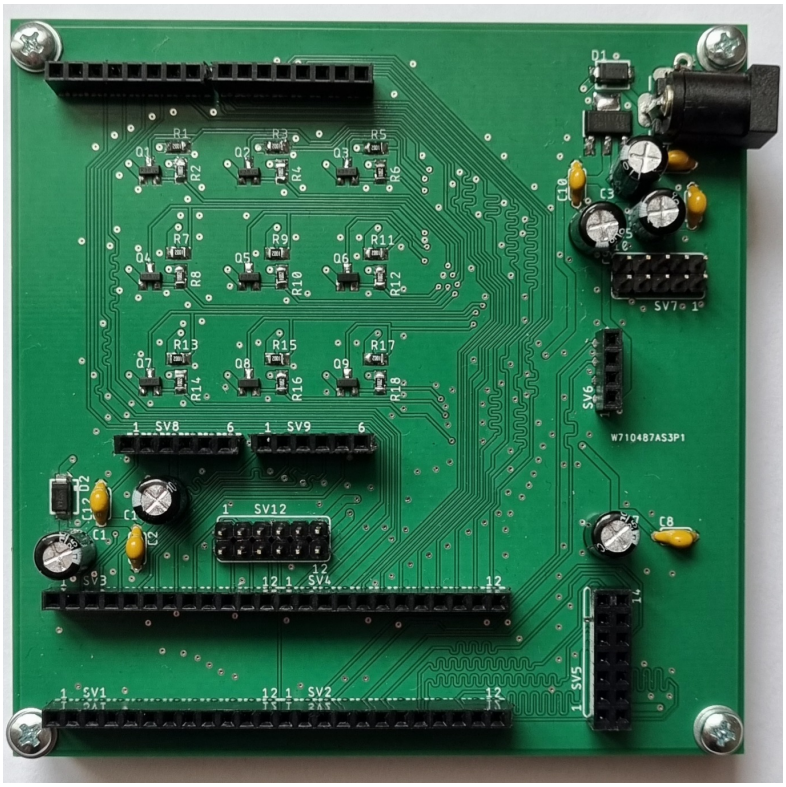


Obr. 5.5 Schéma vývodov konektora SV12

Po finalizácií schémy a CAD modelu dosky plošného spoja budú vygenerované tzv. **Gerber files**, čo sú univerzálne súbory určené pre výrobcu dosky PCB. Výsledná doručená PCB doska vyrobená na základe exportovaných špecifikácií je na obr. 5.6. Po osadení SMD a THT komponentov na PCB spolu s konektormi a podpornými nožičkami je výsledná doska na obr. 5.7. Pridaním modulov zvolených v analytickej časti tejto práce je výsledný prototyp na obr. 5.8. Pre prevodník UART – RS-232 a Ethernet modul je možné vytvoriť a následne pridať oporné šasi, ktoré zníži efekt fyzickej manipulácie s týmito časťami pri pripájaní alebo odpájaní káblov. Taktiež je možné vytvoriť plastovú kostru pre celý prototyp pomocou 3D tlačiarne, ktorá taktiež zabezpečí väčšiu rigiditu.



Obr. 5.6 Neosadená doska PCB; pohľad spredu a zozadu

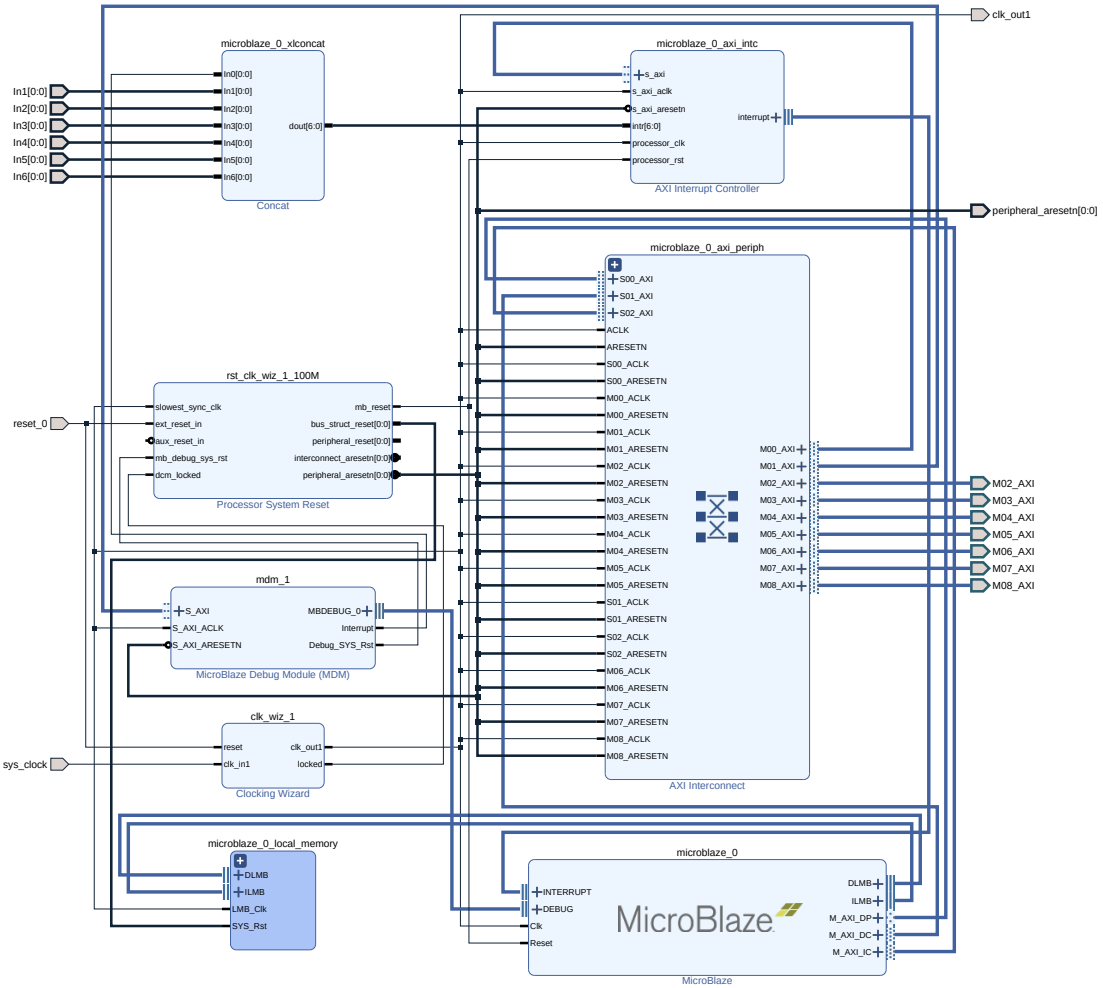


Obr. 5.7 Osadená doska PCB; pohľad spredu

Obr. 5.8 Doska PCB s pridanými modulmi

5.3 Realizácia návrhu blokovej schémy pre obvod FPGA

Po dokončení riešenia všetkých fyzických časti je možné sa presunúť k riešeniu funkčnosti obvodu FPGA. Takýto čip si vyžaduje pre správne fungovanie konfiguráciu. Prvým krokom je vytvoriť projekt v softvérovom balíku **Vivado** od spoločnosti Xilinx, Inc. [8]. Pri vytváraní je nutné zadefinovať používaný prípravok alebo aspoň model čipu FPGA. Definície prípravkov od spoločnosti Digilent, Inc. spolu s postupom inštalácie týchto súborov sú dostupné na stránke [35]. Ak je vytvorený projekt s definovaným prípravkom je možné vytvoriť **blokovú schému**, ktorá graficky reprezentuje vnútornú organizáciu čipu FPGA. Podľa návrhu blokovej schémy v kapitole 4 je možné jednoducho realizovať blokovú schému v tomto programe pridávaním potrebných blokov. Kvôli jednoduchosti pri vytváraní projektu bude ako základ systému použitý procesor **MicroBlaze**. Pretože spomínaná architektúra je dodávaná spoločnosťou Xilinx, Inc; je možné použiť pokročilé funkcie softvérového balíka ako automatické generovanie podporných obvodov a automatická konfigurácia spomínaného procesora s možným povolením **cache**, **prerušení**, **výnimiek**, atď.

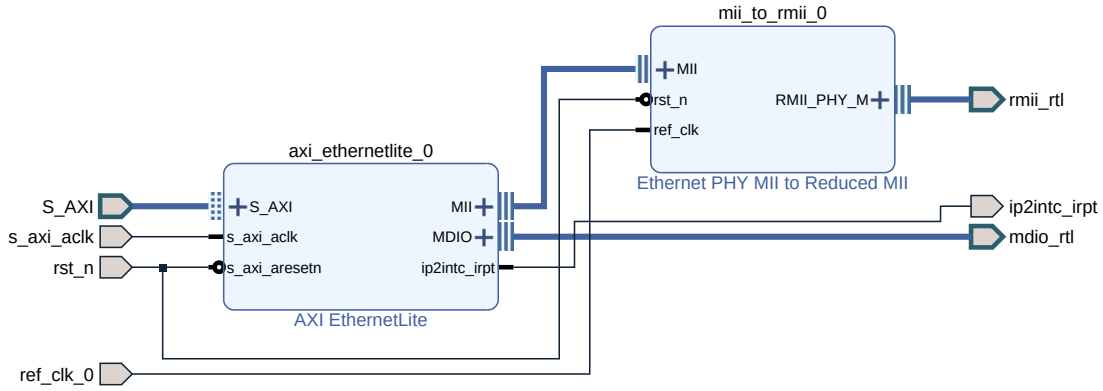


Obr. 5.9 Bloková schéma realizácie riadiacej časti pre obvod FPGA

Po vytvorení blokovej schémy je ako základný prvok pridaný modul **MicroBlaze MCU**, po ktorého pridaní je spustená tzv. **block automation**. Ako prvý krok je zvolenie tzv. **Microcon-**

troller preset. Hodnota **Local Memory** je určená na maximum 128KB. Nastavenie **Cache** je v tomto prípade zvolené na hodnotu 16KB. **Debug Module** je Debug Only, **Peripheral AXI Port** je povolený spolu s **Interrupt Controller**. U **Clock Connection** je zvolená možnosť New Clocking Wizard. Po spustení generácie systém vytvorí všetky spomínané moduly. K vygenerovanému **Clocking Wizard** je následne pripojený hodinový signál prípravku Cmod tak, ako je definovaný v sekcií **Board**. Rovnakým spôsobom je pripojený aj signál **Reset**. Výsledok takéhoto postupu je na obr. 5.9. Týmto je konfigurácia procesoru, pamäte RAM a obslužných obvodov ukončená. Všetky tieto moduly je možné následne zabaliť do nového bloku a vytvoriť tak hierarchiu, čo zlepší čitateľnosť výslednej schémy. Názov bloku je nastavený ako **base_system**. Tento blok tvorí základ systému a všetky ostatné bloky budú na neho napojené pomocou zbernice **AXI** a do modulu **Concat** budú privedené signály prerušenia od ostatných IP jadier, ktoré budú podporovať prerušenie. Tento modul si preto vyžaduje rozšírenie vstupných signálov na 7.

Ako obsluhu rozhrania RMII používaného modulom Ethernet je možné použiť zabudované IP jadro **AXI EthernetLite**, čo predstavuje riešenie na úrovni L1 a L2 modelu OSI/ISO, spolu s jadrom **Ethernet PHY MII to Reduced MII**, ktoré realizuje preklad MII na RMII, opis v sekcii 3.2.2. Konfigurácia jednotlivých blokov ostáva nezmenená z tej pôvodnej. Signály zbernice STA (MDC, MDIO), ref_clk, vstupný signál reprezentujúci referenčný hodinový signál z modulu, spolu so zvyšnými signálmi RMII sú vyvedené na piny prípravku Cmod. Signál prerušenia je pripojený s modulu **Concat** z predchádzajúceho bloku. Výsledná schéma, tak ako je na obr. 5.10, je podobne ako tá predchádzajúca, zabalená do bloku s názvom **ethernet_mac**.



Obr. 5.10 Bloková schéma realizácie Ethernet časti pre obvod FPGA

Nasledujúce IP jadrá sú pridávané do najvyššej úrovne hierarchie, v ktorej sa nachádzajú už vytvorené bloky **base_system** a **ethernet_mac**.

Prvým z IP jadier je **AXI GPIO**. Jeho konfigurácia po vytvorení obsahuje povolenie druhého výstupu a následné obmedzenie počtu vstupne/výstupných signálov na hodnotu 13 pre prvý kanál a na hodnotu 9 pre druhá kanál. Tento modul v systéme realizuje rozhranie **MIPI DBI** určené na komunikáciu s LCD displejom (kanál 1) a konfigurovateľné rozhranie pre externé periférne zariadenia (kanál 2). Pripojený je aj signál prerušenia.

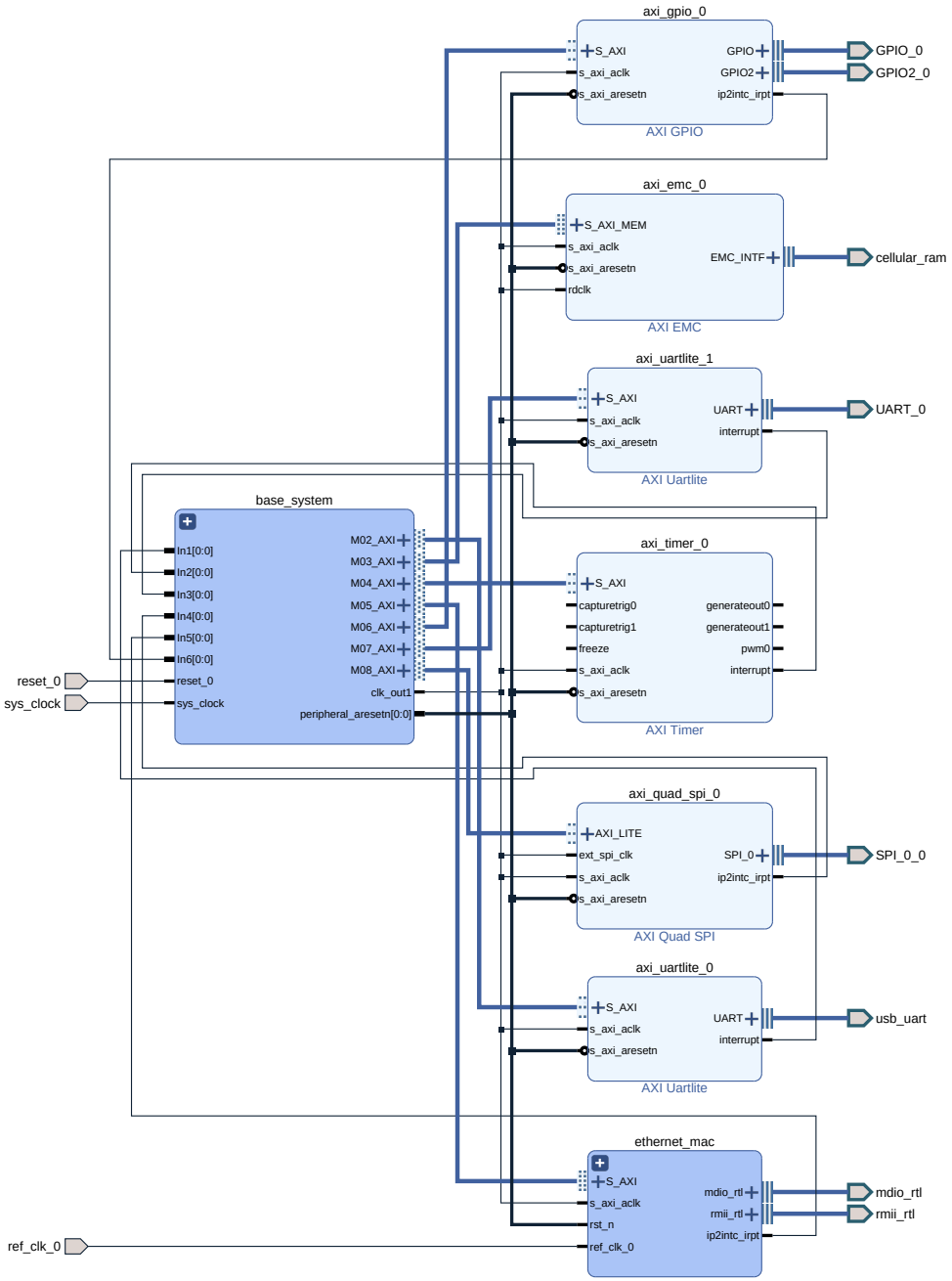
Druhým blokom je **AXI EMC**, ktorý je pridaný cez záložku **Board** a zvolením **Cellular RAM**. Používaný je na rozšírenie interného pamäťového priestoru o externú SRAM, ktorá je súčasťou prípravku Cmod. Tu sa nevyžaduje dodatočná konfigurácia.

Ďalším je **AXI Uartlite** určený na obsluhu UART linky medzi procesorom a externým modulom UART – RS-232 prevodníka. Linka sa skladá z dvoch signálov na prijímanie (RX) a vysielanie (TX) dát. Dodatočne je pridaná kópia cez záložku **Board** a USB UART, ktorý je využívaný na komunikáciu cez sériovú linku USB portu. Obe sú pripojené na samostatný signál prerušenia a nakonfigurované na rýchlosť 115200 baud.

Modul **AXI Timer** je pridaný a nakonfigurovaný ako dvojitý systémový časovač, ktorý dovoľuje procesoru MicroBlaze využívať zabudované softvérové rutiny spánku spolu s druhým časovačom využiteľným na iné účely. Jadro si vyžaduje pripojenie signálu prerušenia.

Posledným modulom je **AXI Quad SPI** využívaný na obsluhu μSD karty pomocou rozhrania SPI. Spravované sú dátové signály MISO a MOSI, hodinový signál SCK a aj signál CS. Dodatočné funkcie modulu ako STARTUP primitive, XIP mode, Dual a Quad SPI mode nie sú využívané a zmenená je aj hodnota **Frequency Ratio** na 8. Signál prerušenia je zapojený.

Výsledná bloková schéma používaná pre konfiguráciu obvodu FPGA je na obr. 5.11.



Obr. 5.11 Kompletná bloková schéma pre obvod FPGA

Z blokovej schémy je následne potrebné vygenerovať tzv. súbor **HDL wrapper**. Potom je v systéme potrebné znovu spustiť generáciu, ale zvoliť možnosť **Copy user generated...**, aby bolo možné tento súbor bez zásahu systému upraviť. Úpravou sa zaistí funkčnosť Ethernet časti, zamedzí sa svieteniu RGB LED diódy na prípravku Cmod a umožní sa reštartovať systém pomocou tlačidla BTN0 na prípravku a zároveň pomocou tlačidla na LCD displeji pripojeného na vývod prípravku. Počet zmien je minimálny, pre signál **rmii_rtl_rx_en** je potrebné zaručiť logickú úroveň 0 viditeľné vo výpise 1 a 2, pre tri samostatné RGB LED logickú úroveň 1 viditeľné vo výpise 1 a taktiež 2 a pre signál **reset_0** logickú funkciu ($\overline{reset\_ext} \vee reset$) vo výpise 3.

```
entity bachelors_hw_wrapper is
  port (
    GPIO2_0_tri_io : inout STD_LOGIC_VECTOR ( 8 downto 0 );
    GPIO_0_tri_io : inout STD_LOGIC_VECTOR ( 12 downto 0 );
    SPI_0_0_io0_io : inout STD_LOGIC;
    SPI_0_0_io1_io : inout STD_LOGIC;
    SPI_0_0_sck_io : inout STD_LOGIC;
    SPI_0_0_ss_io : inout STD_LOGIC_VECTOR ( 0 to 0 );
...
    mdio_rtl_mdc : out STD_LOGIC;
    mdio_rtl_mdio_io : inout STD_LOGIC;
    ref_clk_0 : in STD_LOGIC;
    reset : in STD_LOGIC;
    reset_ext : in STD_LOGIC;
    rmii_rtl_crs_dv : in STD_LOGIC;
    -- rmii_rtl_rx_er : in STD_LOGIC;                          -- removed
    rmii_rtl_rxd : in STD_LOGIC_VECTOR ( 1 downto 0 );
    rmii_rtl_tx_en : out STD_LOGIC;
    rmii_rtl_txd : out STD_LOGIC_VECTOR ( 1 downto 0 );
    sys_clock : in STD_LOGIC;
    usb_uart_rxd : in STD_LOGIC;
    usb_uart_txd : out STD_LOGIC;
    led_rgb : out STD_LOGIC_VECTOR ( 2 downto 0 )           -- added
  );
end bachelors_hw_wrapper;
```

Výpis kódu 1 Ukážka definície entity súboru **bachelors_hw_wrapper.hdl**

```
architecture STRUCTURE of bachelors_hw_wrapper is
...
  signal mdio_rtl_mdio_i : STD_LOGIC;
  signal mdio_rtl_mdio_o : STD_LOGIC;
  signal mdio_rtl_mdio_t : STD_LOGIC;
  signal rmii_rtl_rx_er : STD_LOGIC := '0';     -- added - '0' means no error
  signal reset_comb : STD_LOGIC;                -- added
begin

led_rgb <= "111";                               -- added - LEDs are active low

reset_comb <= not(reset_ext) or reset;          -- added - external reset is active low
...
end STRUCTURE;
```

Výpis kódu 2 Ukážka časti definície architektúry súboru **bachelors_hw_wrapper.hdl**

```
bachelors_hw_i: component bachelors_hw
     port map (
...
        mdio_rtl_mdc => mdio_rtl_mdc,
        mdio_rtl_mdio_i => mdio_rtl_mdio_i,
        mdio_rtl_mdio_o => mdio_rtl_mdio_o,
        mdio_rtl_mdio_t => mdio_rtl_mdio_t,
        ref_clk_0 => ref_clk_0,
        reset_0 => reset_comb,          -- changed from "reset_0 => reset"
        rmii_rtl_crs_dv => rmii_rtl_crs_dv,
        rmii_rtl_rx_er => rmii_rtl_rx_er,
        rmii_rtl_rxd(1 downto 0) => rmii_rtl_rxd(1 downto 0),
        rmii_rtl_tx_en => rmii_rtl_tx_en,
...
    );
```

Výpis kódu 3 Ukážka časti definície komponenty súboru **bachelors_hw_wrapper.hdl**

Nakoniec je potrebné pridať do súboru **constraints** definície pripojenia interných signálov k externým portom obvodu FPGA. Pre pamäť SRAM, USB sériovú linku a hodinový signál nie je nutné manuálne definovať porty, tie budú automaticky pridané pri ďalšom kroku. U portov GPIO, SPI, UART a RMII s MDC a MDIO je ich pripojenie reflektované pomocným diagramom 5.3. Pre RGB LED a BTN0 spolu s pomocnými definíciami na potlačenie varovaní pri implementácií uvedené vo výpise 4.

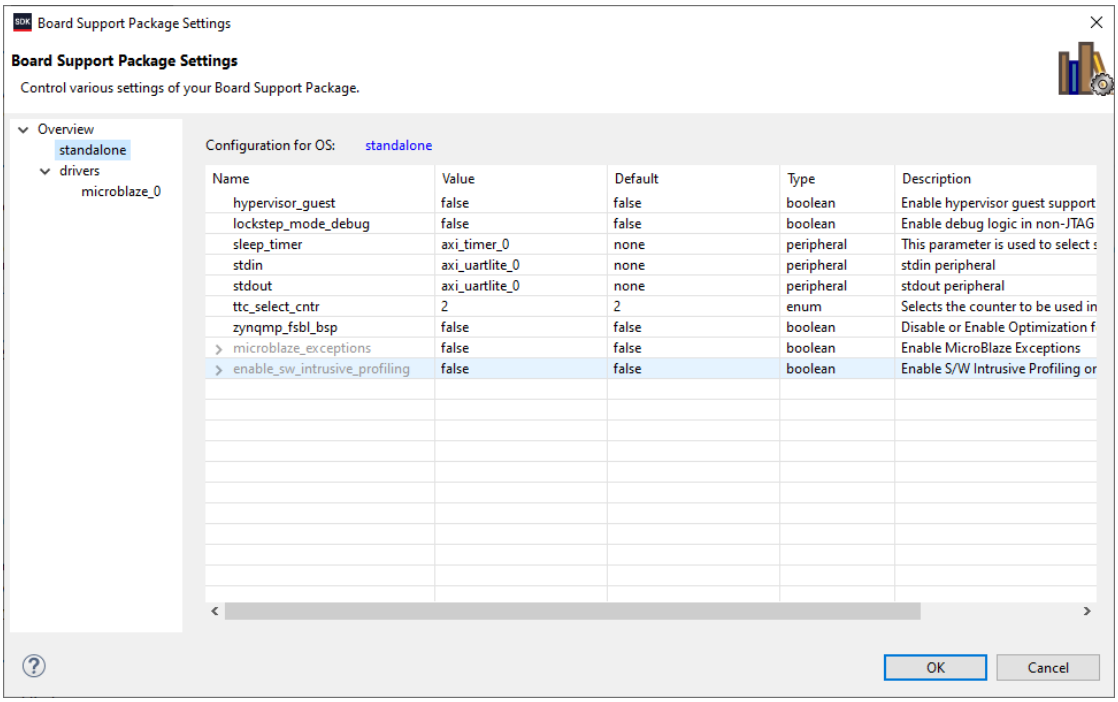
```
## RGB LED
set_property -dict { PACKAGE_PIN B17   IOSTANDARD LVCMOS33 } [get_ports {led_rgb[0]}];
set_property -dict { PACKAGE_PIN B16   IOSTANDARD LVCMOS33 } [get_ports {led_rgb[1]}];
set_property -dict { PACKAGE_PIN C17   IOSTANDARD LVCMOS33 } [get_ports {led_rgb[2]}];

## Buttons
...
set_property -dict { PACKAGE_PIN B18   IOSTANDARD LVCMOS33 } [get_ports {reset}];
...
##  Warning suppression
set_property CLOCK_DEDICATED_ROUTE  FALSE   [get_nets   {ref_clk_0_IBUF}];
set_property IOB                    FALSE   [get_cells  {*/*/*/*/IOFFS_GEN2.DVD_FF}];
set_property IOB                    FALSE   [get_cells  {*/*/*/*/IOFFS_GEN2.RER_FF}];
set_property IOB                    FALSE   [get_cells  {*/*/*/*/IOFFS_GEN2.TEN_FF}];
set_property IOB                    FALSE   [get_cells  {*/*/*/*/IOFFS_GEN[0].RX_FF_I}];
set_property IOB                    FALSE   [get_cells  {*/*/*/*/IOFFS_GEN[1].RX_FF_I}];
set_property IOB                    FALSE   [get_cells  {*/*/*/*/IOFFS_GEN[2].RX_FF_I}];
set_property IOB                    FALSE   [get_cells  {*/*/*/*/IOFFS_GEN[3].RX_FF_I}];
set_property IOB                    FALSE   [get_cells  {*/*/*/*/IOFFS_GEN[0].TX_FF_I}];
set_property IOB                    FALSE   [get_cells  {*/*/*/*/IOFFS_GEN[1].TX_FF_I}];
set_property IOB                    FALSE   [get_cells  {*/*/*/*/IOFFS_GEN[2].TX_FF_I}];
set_property IOB                    FALSE   [get_cells  {*/*/*/*/IOFFS_GEN[3].TX_FF_I}];
```

Výpis kódu 4 Ukážka časti constraints súboru **Cmod-A7-Master.xdc**

Po vytvorení schémy, generácií HDL wrapper a dodefinovaní hodnôt do súboru typu constrains je možné spusiť v softvérovom balíku generáciu tzv. **bitstream**. Tento súbor reprezentuje celú doterajšiu konfiguráciu vo forme binárneho súboru, ktorý je možné následne nahrať do čipu FPGA. Tento súbor sa na konci realizácie aplikácie generuje nanovo a injektuje sa do neho obsah internej pamäte. Počas procesu generácie sa spúšťajú aj procesy **syntézy** a **implementácie**.

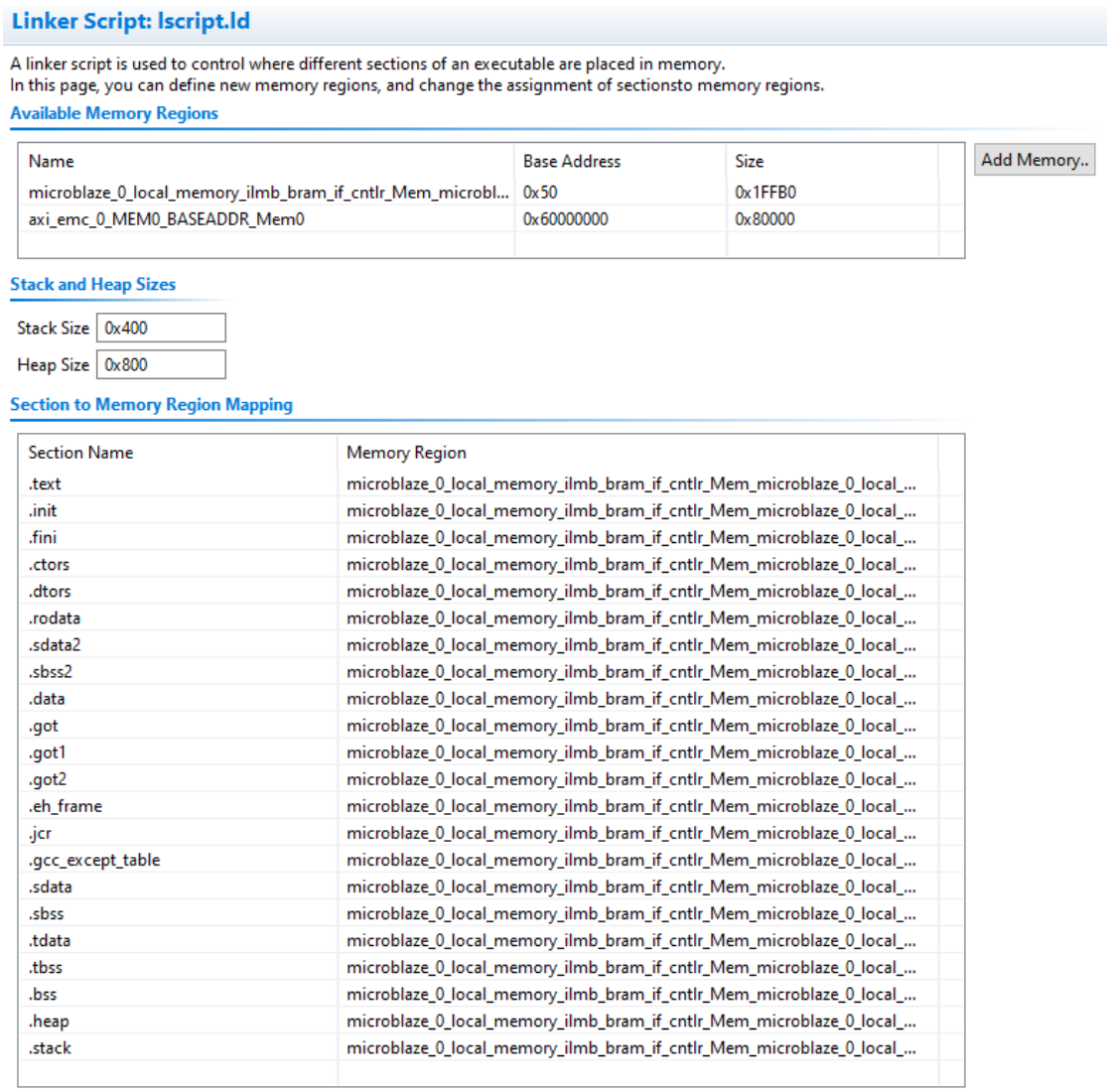
Po vygenerovaní **bitstream** je pre ďalší vývoj potrebné sa presunúť z programu **Vivado** do programu **Xilinx SDK**. Pred tým je ale nutné exportovať hardvér. Pri exportovaní je dobré pridať aj spomínaný **bitsream**. Tento úkon sa vykonáva v časti Vivado. Programovanie čipu FPGA prípravku je možné v oboch programoch, v **Xilinx SDK** je ďalej možné nahrávať program pre fungujúci procesor **MicroBlaze**. Pri vytváraní prvej aplikácie je potrebné vygenerovať tzv. balík **BSP sources**. Generuje sa automaticky pri založení prvej aplikácie. Tento balík obsahuje podporné knižnice pre procesor, všetky používané IP jadrá a taktiež vlastné implementácie **GNU** knižníc. Po vygenerovaní je ale potrebné zmeniť konfiguráciu tohto balíka. Toto nastavenie sa týka použitia **AXI Timer** ako časovača pre vnútorné spánkové rutiny procesora, čo reflektuje nastavenie možnosti **sleep_timer** na hodnotu **axi_timer_0**. Na obr. 5.12 je možné vidieť zmenené nastavenie.



Obr. 5.12 Nastavenie balíka BSP

V časti **Overview** je zároveň možné nastavenie **verzie** používaného operačného systému (ak je používaný) a pridanie zabudovaných knižníc ako napr. **lwip**, **xilffs**, **xiltimer**, atď. Tieto knižnice sú dostupné pre používanú architektúru a ich vložením sa jednoducho rozšíria možnosti výslednej aplikácie. Ich konfiguráciu je možné zmeniť pod záložkou **standalone**. Je ale potrebné počítať s tým, že niektoré z nich sú implementované v jazyku C++ a sú tým pádom náročnejšie na pamäť. V záložke **drivers** sú zobrazené používané architektúry procesorov. Pri používaní **MicroBlaze** je možné meniť používaný archivátor, kompilátor a jeho nastavenia. U iných architektúr sa môžu možné nastavenie líšiť. [36]

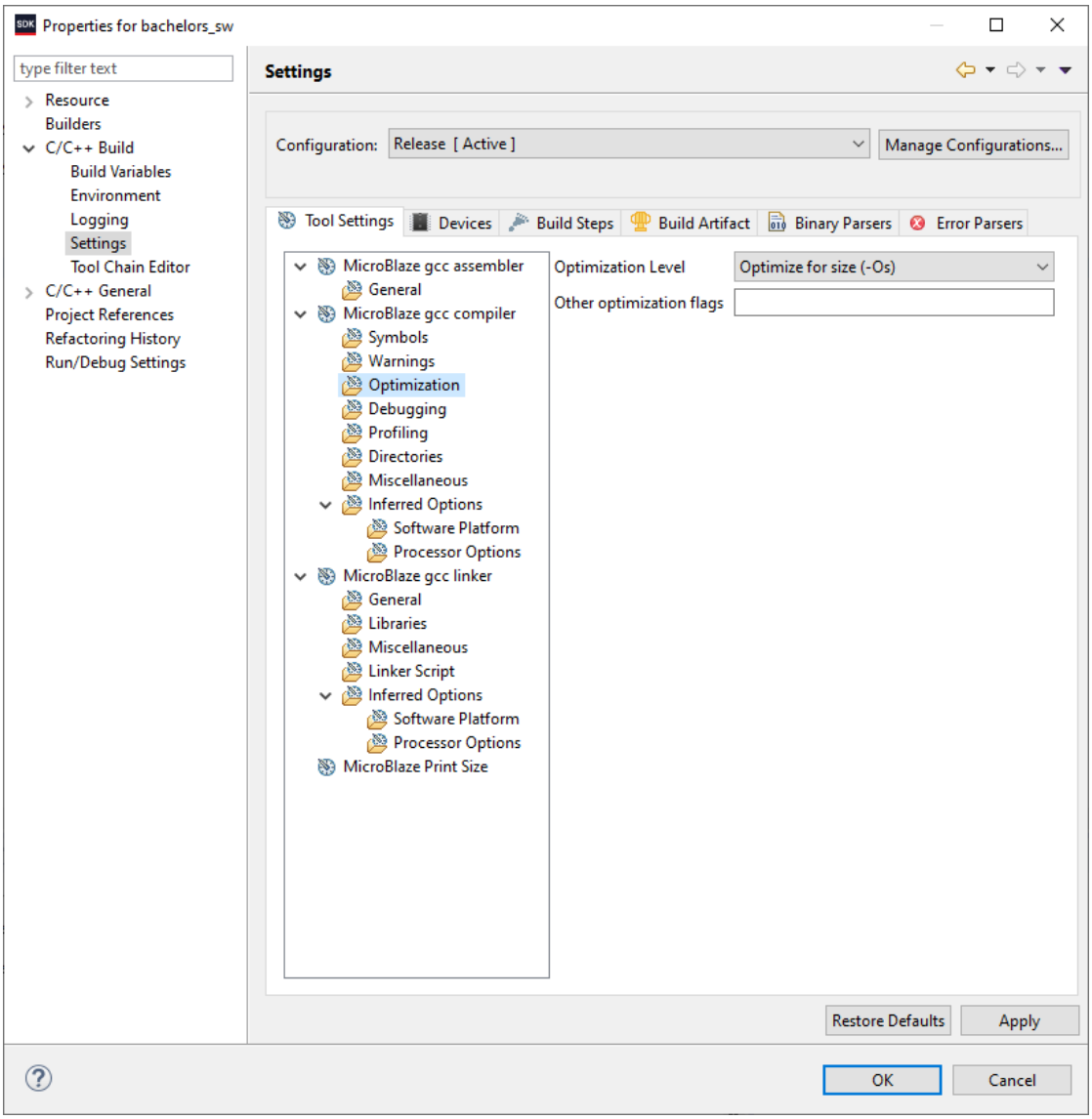
Po tejto zmene je potrebná ešte jedna zmena konfigurácie, ktorá sa týka samotnej aplikácie. Ide o zmenu typu pamäte, ktorá bude využívaná procesorom a programom. Toto nastavenie sa nachádza v tzv. **linker script** súbore **lscript.ld**, ktorý je súčasťou samotnej aplikácie, teda každý aplikačný projekt má svoj samostatný skript. Tu je potrebné zmeniť nastavenie zo základného **axi_emc_0_MEM0_BASEADDR_Mem0** na **microblaze_0_local_memory...**, tak ako je to na obr. 5.13. Ak toto nastavenie ostane v pôvodnej konfigurácií, zariadenie bude fungovať rovnako dobre, ale oveľa pomalšie. Toto nastavenie je vysoko špecifické. Používanú pamäť nemusí byť možné zmeniť, ak nebola pri tvorbe blokovej schémy pre obvod FPGA vygenerovaná interná pamäť, ak nebolo pridané pripojenie na externú SRAM alebo ak externá pamäť neexistuje.



Obr. 5.13 Nastavenie skriptu pre linker

V tomto súbore je možné detailnejšie meniť rozloženie jednotlivých regiónov pamäte, od pozície programu až po umiestnenie a/alebo veľkosť haldy a zásobníka používaných procesorom. Taktiež je možné pridať do mapy pamäte ďalšie rozsahy cez tlačidlo **Add Memory..** Žiadne z týchto častí nie je potrebné pri tejto práci modifikovať.

Pri vývoji aplikácie pre procesor **MicroBlaze** je možné používať rôzne nastavenia kompilátora. Jedno z nich je nastavenie optimalizácie programu. Možnosti začínajú na jej nepoužití (-O0) až po optimalizáciu úrovne 3 (-O3) resp. veľkostnú optimalizáciu (-Os) dostupnú v tomto vývojovom prostredí. Takto je možné ušetriť značnú časť pamäte využívanej programom. Úroveň je možné meniť v preferenciách projektu pod záložkou **C/C++ Build / Settings / MicroBlaze gcc compiler / Optimization**. Pre konfiguráciu **Release** je zvolená možnosť **-Os**. Ukážka je na obr. 5.14. Tu je možné meniť mnoho dodatočných nastavení pri tejto práci, ale ďalšie zmeny nie sú vyžadované.



Obr. 5.14 Nastavenie optimalizácie kompilátora gcc pre procesor MicroBlaze

Po dokončení vývoja na oboch stranách je možné nahrať takúto konfiguráciu do pamäte Flash prípravku Cmod. Podľa návodu [37] je programovanie možné cez **Vivado**. Po naprogramovaní pamäte sa po pripojení napájania k prípravku čip FPGA automaticky naprogramuje a následne sa do pamäte RAM nahraje program a spustí sa jeho vykonávanie na procesore **MicroBlaze**. Tento postup nebude v tejto práci podrobnejšie opísaný.

Kapitola 6
Realizácia softvérovej časti

Kapitola sa zaoberá realizáciou softvérovej strany práce, riešením ovládania periférií a komunikácie s nimi, implementáciou aplikácie, tvorbou ovládania konfiguračného rozhrania a manažmentom úložiska.

6.1 Riešenie ovládania periférií

Ovládanie každého z modulov a rozhraní je riešené iným spôsobom. Základom ovládania je aplikácia, ktorá je nahratá na prípravku. Takúto aplikáciu je možné použitím nástrojov vývojového balíka Xilinx SDK od spoločnosti Xilinx Inc. [38], ktorý je inštalovaný spolu s programom Vivado, vyvíjať a prípadne aj ladiť. Softvérová výbava zodpovedná za komunikáciu s jadrami IP je dodávaná ako súčasť balíka. Takéto knižnice zjednodušujú prácu s používaným hardvérom. Preto sa implementácia aplikácie zameriava na využitie dodávaných funkcií. Vo výnimočných prípadoch, kedy sa správanie zabudovaných makier a funkcií nezhoduje s požadovaným správaním, využívajú sa funkcie ako `uint32_t Xil_In32(uintptr_t Addr)` zodpovedná za čítanie dát z dodanej adresy a `void Xil_Out32(uintptr_t Addr, uint32_t Value)`, ktorá zapisuje dáta na zadanú adresu. Pri obidvoch funkciách existujú aj 8, 16 a 64-bitové verzie s rovnakou funkcionalitou a všetky sú deklarované v hlavičkovom súbore **xil_io.h**. Takto je pri mapovaní periférií do pamäťového priestoru systému možné meniť alebo prečítať hodnoty jednotlivých registrov v pripojených IP jadrách.

Pred použitím akýchkoľvek IP jadier je potrebné realizovať postupnosť nasledujúcich krokov:

1. vytvorenie inštancie IP jadra
2. inicializácia inštancie pomocou volania zavádzajúcej funkcie
3. dodatočná konfigurácia IP jadra pomocou prídavných funkcií (voliteľné)
4. zapojenie systému prerušení a výnimiek (voliteľné)
5. začiatok používania funkcionality pomocou prenosových funkcií
6. zakončenie používania/zakázanie funkčnosti (platí pre výnimky a prerušenia)

Po prejdení všetkých z nich je možné používať funkcionality IP jadier. Ukážky kódu v nasledujúcich sekciách tejto kapitoly reprezentujú príklad realizácie spomínaného postupu pri inicializácií a následnom používaní zabudovaných funkcií. Na základe návrhu zo sekcie 4.3 je umiestnenie používaných inštancií, inicializačných volaní a funkcií, ktoré realizujú prenos presne dané.

6.1.1 Dotykový TFT LCD displej

Komunikácia s modulom dotykového TFT LCD displeja a jeho ovládanie je docielené využitím IP jadra AXI GPIO, keďže súčasťou softvérového balíka Vivado nie je IP jadro, ktoré by sprostredkovávalo takúto komunikáciu natívne. Pri spomínanom jadre je možné nakonfigurovať počet výstupov. Pre tento modul je potrebných osem dátových a päť riadiacich signálov. Programová výbava, ktorá riadi displej, používa knižnicu LCDWIKI_kbv, LCDWIKI_gui a LCDWIKI_touch zo stránky [39]. Všetky tieto knižnice boli prevedené z architektúry **AVR** na používanú architektúru **MicroBlaze** a zároveň z programovacieho jazyka **C++** na **C** z dôvodu potreby úspory pamäte.

```
...
#include <xemaclite.h>
#include <xgpio.h>
#include <xtmrctr.h>
#include <xintc.h>
#include <xspi.h>

#include "config.h"

extern XEmacLite    emac_instance;
extern XGpio        gpio_instance;
extern XIntc        intc_instance;
extern XTmrCtr      tmr_instance;
extern XSpi         spi_instance;
...
```

Výpis kódu 5 Ukážka časti zdrojového kódu hlavičkového súboru **instance.h**

Pre IP jadro **AXI GPIO**, využívané na realizáciu komunikácie v tomto prípade, sa v hlavičkovom súbore **instance.h** s ukážkou 5 definuje jeho inštancia a vkladajú sa tu aj hlavičkové súbory potrebné pre využívanie funkcií jadra (súbor **xgpio.h**). V podpornom súbore **support.c** s ukážkou 6 sa volá inicializačná funkcia spolu s určením smeru jednotlivých vývodov pre LCD kanál (kanál GPIO s hodnotou 1, deklarácia v hlavičkovom súbore **config.h**), kde následne hodnota **0x1FFF** znamená, že všetky porty sú nastavené ako vstupné (maska GPIO predstavuje nastavenie portov od 1 (0x1) až 13 (0x1FFF) do hodnoty log. 1, čo značí vstupný mód signálu, logická 0 tak znamená nastavenie pinu do výstupného módu).

```
...
int initialize_devices() {
    int status;
    ...
    status = XGpio_Initialize(&gpio_instance, XPAR_GPIO_0_DEVICE_ID);
    if (status != XST_SUCCESS) return XST_FAILURE;
    status = XGpio_SelfTest(&gpio_instance);
    if (status != XST_SUCCESS) return XST_FAILURE;
    XGpio_SetDataDirection(&gpio_instance, GPIO_LCD_CHANNEL, 0x1FFF);
    XGpio_SetDataDirection(&gpio_instance, GPIO_CONF_CHANNEL, 0x1FF);
    XGpio_InterruptGlobalDisable(&gpio_instance);
    ...
    return XST_SUCCESS;
}
...
```

Výpis kódu 6 Ukážka časti GPIO zdrojového kódu súboru **support.c**

Prechod z programovacej architektúry Arduino na architektúru Xilinx sa zameriava hlavne na zámenu funkcií a konštánt používaných knižnicami, ktoré sú definované v súboroch **Arduino.h** a **pins_arduino.h**. Ako náhrada boli vytvorené nové definície, ktoré vo výsledku replikujú správanie tých pôvodných. Zmeny tohto typu boli minimálne, keďže väčšina pôvodných definícií kopíruje tie používané vo verejne dostupnom softvéri **GNU** a ten je používaný aj pri architektúre **MicroBlaze**.

Následne sú potrebné rovnaké zmeny špecificky pre architektúru AVR s definíciami v súboroch **pgmspace.h**, **io.h** a mnohých ďalších. V tomto prípade je potrebných viacero modifikácií. Keďže knižnica využíva aj prístup k špecifickým registrom AVR, je potrebné zmeniť celkový pohľad na problematiku. Jeden z problémov je prístup k pinom GPIO. Pôvodne bola akákoľvek operácia vykonávaná cez spomínané registre. Zámena bola docielená použitím podporných funkcií jadra a redukciou problému na logické operácie nad premennými ako je možné vidieť v ukážke 7. Keďže protokol **MIPI DBI** nie je náročný na presné časovanie, je možné operovať s portmi bez potreby prepočítavať čas odozvy. Časť riešenia je zobrazená aj vo výpise kódu 8.

```
...
// These are macros for I/O operations...
void write8(u16 data);

#define read8(dst) {RD_ACTIVE; delay(7); dst = ((XGpio_DiscreteRead(&gpio_instance,
↪  GPIO_LCD_CHANNEL) & DATA) >> 5); RD_IDLE;}

#define setWriteDir()   {XGpio_SetDataDirection(&gpio_instance, GPIO_LCD_CHANNEL,
↪  ALL_DATA_R);}
#define setReadDir()    {XGpio_SetDataDirection(&gpio_instance, GPIO_LCD_CHANNEL,
↪  ALL_DATA);}

// Control signals are ACTIVE LOW (idle is HIGH)
// Command/Data: LOW = command, HIGH = data
// These are single-instruction operations and always inline
#define RD_ACTIVE  {XGpio_DiscreteClear(&gpio_instance, GPIO_LCD_CHANNEL, RD_PORT);}
#define RD_IDLE    {XGpio_DiscreteSet(&gpio_instance, GPIO_LCD_CHANNEL, RD_PORT);}
#define WR_ACTIVE  {XGpio_DiscreteClear(&gpio_instance, GPIO_LCD_CHANNEL, WR_PORT);}
#define WR_IDLE    {XGpio_DiscreteSet(&gpio_instance, GPIO_LCD_CHANNEL, WR_PORT);}
#define CD_COMMAND {XGpio_DiscreteClear(&gpio_instance, GPIO_LCD_CHANNEL, CD_PORT);}
#define CD_DATA    {XGpio_DiscreteSet(&gpio_instance, GPIO_LCD_CHANNEL, CD_PORT);}
#define CS_ACTIVE  {XGpio_DiscreteClear(&gpio_instance, GPIO_LCD_CHANNEL, CS_PORT);}
#define CS_IDLE    {XGpio_DiscreteSet(&gpio_instance, GPIO_LCD_CHANNEL, CS_PORT);}

// Data write strobe, ~2 instructions and always inline
#define WR_STROBE { WR_ACTIVE; WR_IDLE; }
#define RD_STROBE {RD_IDLE; RD_ACTIVE;RD_ACTIVE;RD_ACTIVE;}
#define write16(d) { uint8_t h = (d)>>8, l = d; write8(h); write8(l); }
#define read16(dst) { uint8_t hi; read8(hi); read8(dst); dst |= (hi << 8); }
#define writeCmd8(x){ CD_COMMAND; write8(x); CD_DATA;  }
#define writeData8(x){  write8(x); }
#define writeCmd16(x){ CD_COMMAND; write16(x); CD_DATA; }
#define writeData16(x){ write16(x) }
...
```

Výpis kódu 7 Ukážka časti zdrojového kódu hlavičkového súboru **mcu_8bit_magic.h**

```
...
void write8(uint16_t data) {
    u32 read = XGpio_DiscreteRead(&gpio_instance, GPIO_LCD_CHANNEL);
    u32 read_control = read & CONTROL;
    u32 to_write = data;
    to_write &= DATA;
    XGpio_DiscreteWrite(&gpio_instance, GPIO_LCD_CHANNEL, (read_control | to_write));
    WR_STROBE;
}
...
```

Výpis kódu 8 Ukážka časti zdrojového kódu súboru **mcu_8bit_magic.c**

Pri prevode z C++ na C je potrebné odstrániť objektové rozšírenia, ktoré pridáva jazyk C++ (triedy, dedičnosť, polymorfizmus, preťažovanie funkcií, nové dátové typy, atď.) a rekonštruovať kód tak, aby sa nezmenila jeho funkcionalita. Knižnica LCD_WIKI používa viaceré z výhod jazyka C++ a rozdeľuje sa pomocou tried na KBV, GUI a Touch. Po odstránení tried je nutné premenovať konfliktné funkcie a premenné alebo definovať funkcie, ktoré boli zaručené dedičnosťou s ukážkou 9 (komentáre predstavujú pôvodný kód). Taktiež je potrebné vytvoriť funkciu, ktorá bude realizovať funkcionalitu konštruktora a zameniť prácu s objektami, ich metódami a privátnymi premennými na jednoduché volanie globálne dostupných funkcií a prístup k premenných. Zmenené boli aj definície konštantných metód a istých typov premenných neexistujúcich v jazyku C. Útržky kódu po zmene sú viditeľné vo aj výpise 10, kde komentáre rovnako predstavujú časti originálu.

```
//class {
    //public:
    void LCDWIKI_GUI(void); // Constructor
    //These are defined by the subclass:
    //virtual uint16_t Color_To_565(uint8_t r, uint8_t g, uint8_t b)=0;
    ...
    //virtual int16_t Get_Width(void) const=0;

    //These exist only with LCDWIKI_GUI(no subclass overrides)
    void Set_Draw_color_16(uint16_t color);
    void Set_Draw_color(uint8_t r, uint8_t g, uint8_t b);
    ...
    int Get_Text_Mode(void);// const;
    size_t Print(uint8_t *st, int16_t x, int16_t y);
    //void Print_String(const uint8_t *st, int16_t x, int16_t y);
    //void Print_String(uint8_t *st, int16_t x, int16_t y);
    void Print_String(const char * st, int16_t x, int16_t y);
    ...
    size_t write(uint8_t c);
    int16_t Get_Display_Width(void);// const;
    int16_t Get_Display_Height(void);// const;
    //protected:
    int16_t text_x, text_y;
    uint16_t text_color, text_bgcolor,draw_color;
    uint8_t text_size;
    int text_mode; //if set,text_bgcolor is invalid
    //};
```

Výpis kódu 9 Ukážka časti zdrojového kódu hlavičkového súboru **LCDWIKI_GUI.h**

```
//class LCDWIKI_KBV : public LCDWIKI_GUI {
    //public:
    void LCDWIKI_KBV(uint16_t model,uint8_t cs, uint8_t cd, uint8_t wr, uint8_t
    ↪  rd, uint8_t reset);
    void Init_LCD(void);
    void reset(void);
    void start(uint16_t ID);
    ...
    uint8_t Get_Rotation(void) ;//const;
    void Invert_Display(int i);
    uint16_t Read_Reg(uint16_t reg, int8_t index);
    ...
    void Vert_Scroll(int16_t top, int16_t scrollines, int16_t offset);
    int16_t Get_Height(void); //const;
    int16_t Get_Width(void); //const;
    void Set_LR(void);
    //void Push_Any_Color(uint16_t * block, int16_t n, int first, uint8_t
    ↪  flags);
    //protected:
    uint16_t WIDTH,HEIGHT,width, height, rotation,lcd_driver,lcd_model;
    //private:
    uint16_t XC,YC,CC,RC,SC1,SC2,MD,VL,R24BIT;
    //};
```

Výpis kódu 10 Ukážka časti zdrojového kódu hlavičkového súboru **LCDWIKI_KBV.h**

Po prevode a zmene architektúry je možné použiť funkcionalitu knižníc. **LCDWIKI_KBV** sa zaoberá komunikáciou s displejom na nižšej úrovni, čo predstavuje inicializáciu displeja, zasielanie príkazov a čítanie hodnôt interných registrov kontroléra. Inicializácia obsahuje hlavne reštartovanie kontroléra displeja a nahratie počiatočných hodnôt do interných registrov. Detaily sú obsiahnuté v dokumente [29] pre kontrolér **ILI9488**, ktorý je súčasťou používaného displej modulu. Pôvodný zdrojový kód pozostával aj z mnohých definícií pre iné modely displejov s rôznymi kontrolérmi. Tie boli odstránené kvôli potrebe úspory pamäte. Knižnica **LCDWIKI_GUI** je už vyššia úroveň, ktorá pracuje s grafikou. Obsahuje rôzne funkcie použiteľné na výpis textu, vykresľovanie geometrických útvarov a samostatných pixelov. Ďalej sú tu aj funkcie na zmenu farby, zmenu veľkosti písma, vyplnenie vykreslených útvarov farbami a možnosť zobrazenia obrázkov zo súboru. Definície spomínaných funkcií je možné vidieť vo výpisoch 9 a 10. Nakoniec **LCDWIKI_Touch** rieši vstup z dotykovej časti displeja. Táto časť knižnice nebola využitá.

Možnosti jednotlivých funkcií, ich definície, vstupné/výstupné parametre, návratové hodnoty s poznámkami sú popísané v dokumentácií dodávanej s týmito knižnicami a dostupné na stránke [39] v sekcií **Program Download**. Niektoré funkcie sa s tými v dokumente nezhodujú presne v mene (pôvodný názov ostáva, na konci neho je pridaný doplňujúci text) alebo neexistujú vôbec, čo je dôsledok zmeny programovacieho jazyka. Takýchto funkcií je ale minimum. Použitie knižnice je jednoduché. Po inicializácií jadra sa volajú funkcie `int LCDWIKI_KBV(...)` a `void Init_LCD(void)` s parametrami opísanými v dokumentácií. Potom je možné používať všetky funkcie knižníc.

Výsledná aplikácia obsahuje spomínané súbory knižnice ako súčasť priečinku **lcdwiki_lib**. Okrem prevedenej knižnice sa v pôvodnom priečinku nachádzajú hlavičkové súbory **lcd_font.h** (obsahuje binárnu verziu písma používaného knižnicou), **lcd_mode.h** (obsahuje jednoduchú deklaráciu bitovej šírky systému) a **lcd_registers.h** (obsahuje adresy registrov kontroléra). Toto všetko je následne vložené do priečinku **lcd_module** podľa návrhu zo sekcie 4.3. V tomto priečinku sa nachádzajú súbory **lcd_module_handler.h** a **lcd_module_handler.c**, ktoré zodpovedajú za vykonávanie vypisovania na displej a v nich obsiahnuté funkcie sú volané z podporného súboru **support.c** v koreni projektu.

6.1.2 SD karta

Prevádzkovanie SD karty je docielené pomocou IP jadra **AXI Quad SPI** cez protokol SPI. Jadro má tri výstupné (SCK, CS, MOSI) a jeden vstupný signál (MISO). Softvérové riešenie prístupu k súborom je riadené pomocou modulu **FatFs** zo stránky [40]. Pri použití dostupných implementácií je možné jednoducho sprevádzkovať komunikáciu medzi kartou a modulom. Najjednoduchšia architektúra na prevod je **AVR**, kde si šablóna vyžaduje dodefinovanie funkcií, ktoré realizujú prenos dát. Kód je dodávaný v jazyku **C** a je veľkostne optimalizovaný, čo je ideálne pre túto prácu. Existujú aj implementácie, ktoré pracujú priamo s hardvérovou vrstvou a taktiež verzia modulu **Petit FatFs** s minimalistickou implementáciou, kde je dostupných len zopár funkcií. Inštancia IP jadra s vložením hlavičkového súboru **xspi.h** sa nachádzajú v súbore **instance.h**, ukážka 5. Dodatočná inicializácia sa nachádza spolu s ostatnými v súbore **support.c** a časť zodpovedného zdrojového kódu je v ukážke 11. Konštanta `SPI_OPTIONS` definuje správanie IP jadra. Jej počiatočná hodnota je definovaná v **options.h** (`XSP_MASTER_OPTION`).

```
...
int initialize_devices() {
    int status;
    ...
    status = XSpi_Initialize(&spi_instance, XPAR_SPI_0_DEVICE_ID);
    if (status != XST_SUCCESS) return XST_FAILURE;
    status = XSpi_SetOptions(&spi_instance, SPI_OPTIONS);
    if (status != XST_SUCCESS) return XST_FAILURE;
    status = XSpi_Start(&spi_instance);
    if (status != XST_SUCCESS) return XST_FAILURE;
    XSpi_IntrGlobalDisable(&spi_instance);
    ...
    return XST_SUCCESS;
}
...
```

Výpis kódu 11 Ukážka časti SPI zdrojového kódu súboru **support.c**

Ovládanie komunikácie cez protokol SPI je sprostredkované funkciami pre spomínané IP jadro. Po inicializácií sú využívané funkcie vo zvyšku kódu, riadenie signálu **CS** je ale riešené priamym zápisom do registrov jadra funkciami opísanými v sekcii 6.1. Používajú sa pretože volanie akejkoľvek funkcie obsluhy spôsobuje zmeny stavu signálu **CS**, čo je nevyžadované správanie. Interakcie s týmto signálom reprezentujú makrá `CS_H()` a `CS_L()`. Využitie spomínaných funkcií jadra je čiastočne dokumentované vo výpise kódu 12 súboru **sdmm.c**.

```
...
#define CS_INIT()   XSpi_SetSlaveSelect(&Spi, 1); /* Initialize port for MMC CS as output */
#define CS_H()      Xil_Out32(XPAR_AXI_QUAD_SPI_0_BASEADDR + 0x70, 1); /* Set MMC CS "high" */
#define CS_L()      Xil_Out32(XPAR_AXI_QUAD_SPI_0_BASEADDR + 0x70, 0); /* Set MMC CS "low" */
...
static void rcvr_mmc(
    BYTE *buff, /* Pointer to read buffer */
    UINT bc;    /* Number of bytes to receive */
) {
    memset(buff, 0xFF, bc);
    XSpi_Transfer(&Spi, buff, buff, bc);
}
...
```

Výpis kódu 12 Ukážka časti zdrojového kódu súboru **sdmm.c**

Pred začatím prenosu údajov je nutné inicializovať používanú SD kartu. To sa dá docieliť počiatočným odoslaním 80 cyklov hodinového signálu (odporúčanie je nepresiahnuť frekvenciu 400 kHz pre staršie karty), následným nastavením signálu **CS** na log. 0, odoslaním príkazu **CMD0**, potom **CMD8**. Podľa odpovede karty na tento príkaz sa odvodí typ karty a jej použiteľnosť. Podľa odpovede je voliteľné alebo povinné odoslať príkaz **CMD58** a následne **ACMD41**. Podľa odpovede karty sa určí, či je karta pripravená na komunikáciu alebo nie. Presný postup ako postupovať je špecifikovaný v dokumente [41] od **SD Card Association**.

Súborové operácie sú sprostredkované spomínaným modulom **FatFs**, ktorý obsahuje mnoho rôznych funkcií, ktoré značne zjednodušujú prístup k dátam na SD karte, pretože tento modul rieši aj problém súborového systému. Kompletná dokumentácia k modulu **FatFs** je dostupná na stránke celého projektu [40]. Pred použitím akejkoľvek funkcionality modulu je dobré skontrolovať obsah súboru **ffconf.h**. Tento hlavičkový súbor obsahuje konfiguráciu určujúcu správanie celého modulu. Editáciou možností je možné docieliť globálne zakázanie zapisovania, zmenu prevodu zakončovacích znakov, zmenu lokalizácie, povolenie úložiska s viacerými partíciami, rešpektovanie formátu **ExFAT** a iné možnosti. Začiatkom je vytvorenie inštancie typu `FATFS` definovanej v hlavičkovom súbore **ff.h**. Inicializácia celého systému vrátane SD karty a naviazanie inštancie na používaný súborový systém sa vykonáva pomocou funkcie `FRESULT f_mount (FATFS* fs, const TCHAR* path, BYTE opt)`, kde **fs** reprezentuje spomínanú inštanciu, **path** je cesta v súborovom systéme, ktorá ma byť naviazaná (väčšinou je to tzv. **root**, ktorý je definovaný hodnotou "") a **opt** je možnosť okamžitého napojenia (hodnota 1) alebo napojenia pri prvej súborovej operácií (hodnota 0). Po vykonaní tejto funkcie je možné globálne pristupovať k súborom na karte. Čítanie/zápis súborov je realizovaný pomocou objektu typu `FIL` a pred operáciami je potrebné otvoriť súbor použitím funkcie `FRESULT f_open (FIL* fp, const TCHAR* path, BYTE mode)`, kde **path** je názov súboru a **mode** definuje typy prístupu a iné možnosti. Všetky možnosti tak ako sú definované v súbore **ff.h**:

- `FA_READ` – špecifikuje čítanie
- `FA_WRITE` – špecifikuje zápis, kombinácia s predchádzajúcou možnosťou pre **read-write** prístup
- `FA_OPEN_EXISTING` – špecifikuje otvorenie existujúceho súboru, ak neexistuje, funkcia skončí chybou (základná možnosť)
- `FA_CREATE_NEW` – špecifikuje vytvorenie nového súboru, ak existuje, funkcia skončí chybou s návratovou hodnotou `FA_EXISTS`
- `FA_CREATE_ALWAYS` – špecifikuje vytvorenie nového súboru, ak existuje, bude takýto súbor prepísaný
- `FA_OPEN_ALWAYS` – špecifikuje otvorenie existujúceho súboru, ak neexistuje, bude súbor vytvorený
- `FA_OPEN_APPEND` – rovnaké ako predchádzajúca možnosť, zápis sa začína od konca pôvodného súboru

Ak je návratová hodnota predchádzajúcej funkcie `FR_OK`, je možné pristúpiť k súboru. Čítanie realizuje funkcia `FRESULT f_read (FIL* fp, void* buff, UINT btr, UINT* br)`, ktorej je potrebné dodať odkaz **buff**, čo môže byť napr. pole znakov, ďalej číslo **btr** určujúce koľko bajtov má byť prečítaných a odkaz **br** na premennú, do ktorej bude uložený počet naozaj prečítaných znakov. Zápis realizuje funkcia `FRESULT f_write (FIL* fp, const void* buff, UINT btw, UINT* bw)`, kde znova **buff** je odkaz na pole, kde sa nachádzajú údaje, ktoré budú následne zapísané do súboru, **btw** reprezentuje počet bajtov, ktoré majú byť zapísané a nakoniec **bw** odkaz na premennú, kde bude uložený počet zapísaných bajtov. Obe funkcie vracajú hodnoty, na základe ktorých je možné identifikovať výsledok operácie.

Pomocné makro `f_size(fp)` je výhodné použiť pri potrebe zistiť veľkosť otvoreného súboru. Po ukončení práce so súbormi je možné uzavrieť súborový objekt funkciou `FRESULT f_close (FIL* fp)` a odpojenie súborového systému makrom `f_unmount(path)`, čo je pôvodná funkcia `FRESULT f_mount (FATFS* fs, const TCHAR* path, BYTE opt)` s parametrami **fs** a **opt**, ktoré majú hodnotu **0** a **path** ako cesta na koreňový priečinok. Príklad použitia je v ukážke kódu 13.

Pripojenie k súborovému systému sa vykonáva v súbore **fatfs_handler.c**, pomocou inštancie v súbore **instance.h** je prístup dostupný globálne, prístup k súborom je riešený pri každom použití samostatne.

```
...
FRESULT fr;

unsigned char buff[36];
UINT num = 0;

fr = f_open(&Fil, buffer, FA_READ);
if (fr != FR_OK) {
    xil_printf("FAILURE SD\r\n");
    return XST_FAILURE;
}

file_size = f_size(&Fil);

while (1) {
    ...
    if (need) {
        f_read(&Fil, &buff, 6, &num);
        actual_pos += num;
        if (file_size <= actual_pos) {
            finish_file = num;
        }
        memcpy(data, buff, 6);
        need = 0;
    }
    ...
```

Výpis kódu 13 Ukážka časti zdrojového kódu súboru **out.c**

V ukážke je realizované otvorenie súboru s názvom obsiahnutým v premennej **buffer**, spočítanie veľkosti súboru, s ktorým systém pracuje a následne aj čítanie jeho obsahu. Čítanie prebieha po 6 bajtoch, čo je vyžadované dĺžkou prístupu k dátam a tým vzniknutým oneskorením, ktoré zasahuje do výkonu emulácie.

Súčasťou modulu **FatFs** je mnoho funkcií, napr. `f_mkdir`, `f_opendir`, `f_readdir}` `f_closedir`, `f_chdir` pre prácu s priečinkami, `f_rename`, `f_truncate`, `f_findfirst`, `f_findnext` pre prácu so súbormi. Existujú aj funkcie pre prácu so súborovým systémom `f_mkfs` alebo s médiom ako `f_fdisk`. Tieto funkcie sa používajú len v špeciálnych prípadoch, preto nebude ich funkčnosť v tejto práci bližšie opísaná.

Rozloženie súborov znova reflektuje na návrh v sekcii 4.3. Celý modul **FatFs** sa nachádza v priečinku **fatfs**, ktorý je súčasťou **fatfs_module**. Volania funkcií sú vykonávané hlavne z obsluhy emulácie, kde sa využíva prístup k súborom. Keďže tento modul spravuje všetko od inicializácie SD karty až po prácu so súbormi, nevyžaduje si dodatočnú obsluhu ako to bolo pri ovládaní displeja v sekcii 6.1.1. Vytvorenie inštancie a pripájanie k súborovému systému je ale vykonávané v súbore **fatfs_handler.c**. To umožní používať inštanciu na prístup k súborom z rôznych miest programu, čo je ideálne pre pridávanie nových funkcionalít do aplikácie.

6.1.3 Modul Ethernet

Modul Ethernet komunikuje so systémom pomocou rozhrania **RMII**, ktoré je vnútorne prekladané na rozhranie **MII**, ktoré následne spracováva IP jadro **AXI EthernetLite**. Táto hierarchia reprezentuje dve vrstvy modelu **ISO/OSI** a to **PHY** (vrstva 1, modul, prevodník a časť IP jadra) a **MAC** (vrstva 2, časť IP jadra). Pretože modul neobsahuje hardvérovo definovanú lokálnu MAC adresu druhej vrstvy, musí byť rozhraniu pridelená. Adresa je zvolená tak, aby pravdepodobnosť konfliktu v lokálnej sieti bola čo najnižšia. Jedna z takých adries je **CA:FE:DE:AD:BE:EF**. Keďže pre uskutočnenie komunikácie zo zvyškom siete je potrebné prevádzkovať aj vrstvu 3 (IP), je dobré použiť tzv. **TCP/IP stack**. Toto softvérové vybavenie rieši komunikáciu s druhou vrstvou a zároveň prevádzkuje spomínanú tretiu vrstvu. Kvôli pamäťovým obmedzeniam je ideálne riešenie **μIP** dostupné na stránke [42]. Tento softvér obsahuje aj riešenia základnej **TCP/UDP** komunikácie spolu s protokolmi **ICMP** a **Telnet**.

```
...
int initialize_devices() {
    int status;

    status = XEmacLite_Initialize(&emac_instance, XPAR_EMACLITE_0_DEVICE_ID);
    if (status != XST_SUCCESS) return XST_FAILURE;
    status = XEmacLite_SelfTest(&emac_instance);
    if (status != XST_SUCCESS) return XST_FAILURE;
    XEmacLite_SetMacAddress(&emac_instance, hardware_address);
    XEmacLite_FlushReceive(&emac_instance);
    ...
    return XST_SUCCESS;
}
...
```

Výpis kódu 14 Ukážka časti EthernetLite zdrojového kódu súboru **support.c**

Podla návrhu v sekcii 4.3 sa inštancia s vloženým hlavičkovým súborom rovnako ako v ostatných prípadoch nachádza v **instance.h**. Inicializácia sa vykonáva v súbore **support.c** s ukážkou 14. Po inicializácií je možné bez problémov komunikovať so zariadeniami na sieti.

Konfigurácia obsluhy **μIP** je komplikovanejšia. Prvým krokom je začlenenie funkcií IP jadra používaným na odosielanie a prijímanie rámcov do softvéru stack-u. Miesto pre funkcie prijímania, resp. odosielania sa nachádza v súbore **uip_handler.c**. Tento súbor je vyššou inštanciou pre spravovanie komunikácie cez sieť. Použije sa funkcia `uint16_t XEmacLite_Recv(XEmacLite *InstancePtr, uint8_t *FramePtr)` na prijímanie rámcov, resp. sa v ňom použije funkcia `int XEmacLite_Send(XEmacLite *InstancePtr, uint8_t *FramePtr, unsigned ByteCount)` na odosielanie rámcov v ukážke 15.

Taktiež je potrebné nahradiť funkcie, ktoré sú súčasťou **UNIX** systémov. Jedna z takých je funkcia s deklaráciou `uint16_t htons(uint16_t hostshort)`. Je možné ju nahradiť za ekvivalent určený pre architektúru **MicroBlaze** s makrom `uint16_t Xil_Htons(uint16_t Data)`, ktoré je deklarované v súbore **xil_io.h**. Nahradenie si vyžaduje aj funkcia `int gettimeofday(struct timeval * __restrict, void * __restrict)` používaná na získanie aktuálneho času, na základe ktorého je prevádzkovaný interný časovač. Tento softvérový časovač je používaný pri obsluhe tzv. **timers** a **polling** pri komunikácií typu **TCP** a používa sa aj ako iniciátor vymazávania starých záznamov z tabuľky **ARP** po uplynutí 10 sekúnd.

Nahradenie tejto funkcie nie je jednoduché, keďže na architektúre **MicroBlaze** bez použitia operačného systému neexistuje ekvivalentná funkcia, ani makro. Preto je potrebné použiť IP jadro **AXI Timer**, ktoré bude po nakonfigurovaní pracovať podobne (po uplynutí 1 ms inkrementuje hodnotu premennej reprezentujúcej tzv. **time since epoch** o 1).

```
...
int run_uip() {
    uip_len = XEmacLite_Recv(&emac_instance, (u8 *)RxFrame);
    memcpy(BUF, RxFrame, uip_len);

    if(uip_len > 0) {
        if(BUF->type == Xil_Htons(UIP_ETHTYPE_IP)) {
            uip_arp_ipin();
            uip_input();
            if(uip_len > 0) {
                ...
                XEmacLite_Send(&emac_instance, (u8 *)&TxFrame, uip_len);
            }
        } else if(BUF->type == Xil_Htons(UIP_ETHTYPE_ARP)) {
            ...
                XEmacLite_Send(&emac_instance, (u8 *)&TxFrame, uip_len);
            ...
        }

    } else if(timer_expired(&periodic_timer)) {
        timer_reset(&periodic_timer);
        for(int i = 0; i < UIP_CONNS; i++) {
            uip_periodic(i);
            if(uip_len > 0) {
                ...
                XEmacLite_Send(&emac_instance, (u8 *)&TxFrame, uip_len);
            }
        }
    ...
    }

    return XST_SUCCESS;
}
...
```

Výpis kódu 15 Ukážka časti zdrojového kódu súboru **uip_handler.c**

Konštanta `RESET_VALUE` obsiahnutá v hlavičkovom súbore **config.h** reprezentuje hodnotu, na ktorú bude čítač časovača nastavený pri inicializácií a následne pri každom jeho prenose. Číslo **100000** nie je zvolené náhodne, je výsledkom rovnice (f_{clock} je frekvencia procesoru):

$$\text{RESET\_VALUE} = (1 * 10^{-3} s / f_{clock})$$

Konfigurácia IP jadra podľa hodnoty v súbore **config.h** znamená počítať smerom dole (odčítať). Potom reprezentuje hodnota `RESET_VALUE` počet takýchto operácií, ktoré je potrebné vykonať aby nastal prenos. Jedna operácia je vykonávaná každých 10 ns, čo reprezentuje frekvenciu 100 MHz. Pri prenose sa aktivuje prerušenie a hodnota čítača sa nastaví naspäť na spomínanú hodnotu. Takto je možné zaručiť prerušenie približne každú 1 ms. Pre použitie tohto jadra je znova potrebné použiť inštanciu zo súboru **instance.h**, inicializovať časovač a nastaviť spomínanú **reset value**. Všetko toto je vykonávané v súbore **support.c** s ukážkou v 16. Spracovanie vyvolaného prerušenia je riešené zavolaním obslužnej funkcie, v ktorej sa inkrementuje globálna premenná deklarované v súbore **global.h**. Konštanty ako `INTC_OPTIONS` (`XIN_REAL_MODE`) a `TMR_OPTIONS` (`XTC_INT_MODE_OPTION` | `XTC_AUTO_RELOAD_OPTION` | `XTC_DOWN_COUNT_OPTION`) sú obsiahnuté v súbore **config.h** a pri konfigurácií nastavujú správanie IP jadier, hodnoty v zátvorkách sú pôvodné nastavenia.

```
...
int initialize_devices() {
    int status;
    ...
    status = XTmrCtr_Initialize(&tmr_instance, XPAR_TMRCTR_0_DEVICE_ID);
    if (status != XST_SUCCESS) return XST_FAILURE;
    status = XTmrCtr_SelfTest(&tmr_instance, TMR_NUMBER);
    if (status != XST_SUCCESS) return XST_FAILURE;

    status = XIntc_Initialize(&intc_instance, XPAR_INTC_0_DEVICE_ID);
    if (status != XST_SUCCESS) return XST_FAILURE;
    status = XIntc_ConnectFastHandler(&intc_instance, TMR_INTR_NUMBER,
    ↪  (XFastInterruptHandler)fast_timer_handler);
    if (status != XST_SUCCESS) return XST_FAILURE;
    status = XIntc_Start(&intc_instance, INTC_OPTIONS);
    if (status != XST_SUCCESS) return XST_FAILURE;
    XIntc_Enable(&intc_instance, TMR_INTR_NUMBER);
    Xil_ExceptionInit();
    Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
    ↪  (Xil_ExceptionHandler)XIntc_InterruptHandler, &intc_instance);
    Xil_ExceptionEnable();

    XTmrCtr_SetHandler(&tmr_instance, timer_handler, &tmr_instance);
    XTmrCtr_SetOptions(&tmr_instance, TMR_NUMBER, TMR_OPTIONS);
    XTmrCtr_SetResetValue(&tmr_instance, TMR_NUMBER, RESET_VALUE);
    XTmrCtr_Start(&tmr_instance, TMR_NUMBER);

    return XST_SUCCESS;
}
...
```

Výpis kódu 16 Ukážka časti Timer zdrojového kódu súboru **support.c**

Súčasťou inicializácie a zavedenia IP jadra je použitie prerušení a výnimiek. Postup ich konfigurácie je taktiež súčasťou súboru **support.c** spolu s inštanciami v **instance.h**. V predchádzajúcej ukážke 16 je zobrazený postup inicializácie a následného použitia.

Ako ďalší krok konfigurácie je editácia súboru **uip-conf.h**, kde sa nachádzajú deklarácie používaných konštánt. Tie definujú obmedzenia komunikácie, limity vyrovnávacej pamäte, endianitu a dodatočné nastavenia pre UDP. Vzorová verzia súboru bola použitá v tejto práci bez zmeny. K základnej výbave programu boli pridané súbory **shell.h**, **shell.c**, **telnetd.h** a **telnetd.c**. Súbor **telnetd.h** musí byť zahrnutý v súbore **uip-conf.h** aby bolo možné použiť funkcie pre **Telnet**. Bez ďalších úprav je možné použiť takúto konfiguráciu **μIP** a po spustení naviazať spojenie cez terminálový emulátor použitím protokolu **Telnet**. Rozšírenie systému o ďalšie príkazy sa dá pridaním funkcií do súboru **shell.c**. Následne je potrebné takúto funkciu zaradiť do štruktúry spolu s textom príkazu, na ktorý ma systém reagovať tak ako v ukážke 17. V súbore **shell.c** sa dajú zmeniť detaily príkazového riadku ako napr. text odpovede na príkaz **help**, meno zariadenia, atď. Rozloženie súborov sa riadi návrhom zo sekcie 4.3, takže priečinok **uip** a nad ním **uip_module**.

```
static struct ptentry parsetab[] = {
    {"help", help}, {"exit", shell_quit}, {"?", help}, {"c64", run_out}, {"ip", change_ip},
    {NULL, unknown}     // default action
};
```

Výpis kódu 17 Ukážka štruktúry ako časť súboru **shell.c**

6.1.4 Konfigurovateľné rozhranie

Konfigurovateľné rozhranie predstavuje vyvedenie signálov druhého kanálu **AXI GPIO** (9 dátových signálov s uzemnením). Spolu so softvérovou výbavou je možné použiť spomínané rozhranie na komunikáciu s externým zariadením náročným na presné časovanie. Pretože pre každý signálový vodič je na doske PCB ekvivalentný prevodník logickej úrovne, fungujú všetky zo spomínaných signálov na logickej úrovni **5 V**.

IP jadro si vyžaduje konfiguráciu pred prvým použitím, tak ako všetky ostatné jadrá. Inštancia sa nachádza v súbore **instance.h** a konfigurácia v **support.c**. Jej ukážka je vo výpise kódu 18. Nastavenie je viazané na to, ktoré je popísané v sekcii 6.1.1. Je potrebné nastaviť prerušenie a výnimky rovnako ako v sekcii 6.1.3, pretože prijímanie dát na nábežnú hranu bez prerušenia je nespoľahlivé a vedie k častým chybám synchronizácie. Po nakonfigurovaní jadra spolu s nastavením prerušení je možné použiť takéto rozhranie. Pretože je komunikácia na tomto rozhraní náročná na presné časovanie, je nutné pred jej začatím **pozastaviť** vykonávanie akýchkoľvek iných procesov (prijímanie/odosielanie dát cez sieť, ovládanie displeja, ovládanie časovača a jeho vynucovanie si prerušení). SD karta a AXI Quad SPI ostávajú pracovať, pretože sú využívané ovládaním na odloženie prenášaných súborov.

```
...
int initialize_devices() {
    int status;
    ...
    status = XGpio_Initialize(&gpio_instance, XPAR_GPIO_0_DEVICE_ID);
    if (status != XST_SUCCESS) return XST_FAILURE;
    status = XGpio_SelfTest(&gpio_instance);
    if (status != XST_SUCCESS) return XST_FAILURE;
    XGpio_SetDataDirection(&gpio_instance, GPIO_CONF_CHANNEL, 0x1FF);
    XGpio_InterruptGlobalDisable(&gpio_instance);
    ...
    status = XIntc_Connect(&intc_instance, GPIO_INTR_NUMBER,
    ↪  (Xil_ExceptionHandler)gpio_handler, &gpio_instance);
    if (status != XST_SUCCESS) return XST_FAILURE;
    ...
    XIntc_Enable(&intc_instance, GPIO_INTR_NUMBER);

    XGpio_InterruptEnable(&gpio_instance, GPIO_CONF_CHANNEL);
    XGpio_InterruptGlobalDisable(&gpio_instance);
    ...
    return XST_SUCCESS;
}
...
```

Výpis kódu 18 Ukážka časti GPIO zdrojového kódu súboru **support.c** pre konfigurovateľné rozhranie

Každý protokol komunikácie cez takéto rozhranie si vyžaduje samostatnú softvérovú implementáciu. Súbory s implementáciami je možné pripojiť k **interface_handler.c** a vo funkcii `int run_interface(int option)` pridať volanie ovládacej funkcie, ktorá následne realizuje obsluhu zvoleného rozhrania. Na realizáciu modelu komunikácie cez **CBM/IEC** rozhranie bola na základe opisu v sekcii 3.2.1 vytvorená softvérová implementácia. Základom je súbor **out.c**, ktorého ukážka je vo výpise kódu 20 a 21. Konštanty, deklarácie funkcií a inštancií sa nachádzajú v hlavičkovom súbore **iec.h**. Funkcie ako `void write_bit(int pin, int value)` a `int read_bit(int pin)` sú používané na zápis/čítanie hodnoty zadaného pinu, boli vytvorené pre zlepšenie čitateľnosti kódu s ukážkou 19. Tieto funkcie sa nachádzajú v súbore **interface_handler.c**, ktorý sprostredkováva všetky potrebné volania.

```
...
int read_bit(int pin) {
    XGpio_SetDataDirection(&gpio_instance, GPIO_CONF_CHANNEL,
    ↪  (XGpio_GetDataDirection(&gpio_instance, GPIO_CONF_CHANNEL) | pin));
    return ((XGpio_DiscreteRead(&gpio_instance, GPIO_CONF_CHANNEL) & pin) ? 1 : 0);
}

void write_bit(int pin, int value) {
    if (value) {
        XGpio_SetDataDirection(&gpio_instance, GPIO_CONF_CHANNEL,
        ↪  (XGpio_GetDataDirection(&gpio_instance, GPIO_CONF_CHANNEL) | pin));
        XGpio_DiscreteSet(&gpio_instance, GPIO_CONF_CHANNEL, pin);
    }
    else {
        XGpio_SetDataDirection(&gpio_instance, GPIO_CONF_CHANNEL,
        ↪  (XGpio_GetDataDirection(&gpio_instance, GPIO_CONF_CHANNEL) & ~pin));
        XGpio_DiscreteClear(&gpio_instance, GPIO_CONF_CHANNEL, pin);
    }
}
```

Výpis kódu 19 Ukážka pomocných funkcií ako časť súboru **interface_handler.c**

Rozloženie jednotlivých súborov znova reflektuje na návrh v sekcii 4.3. Súbory pre špecifický emulovaný protokol sa nachádzajú vo vlastných priečinkoch ako napr. pre CBM/IEC je to **cbm_iec** vo vnútri priečinku **interface_module**. V tomto priečinku sa nachádzajú aj súbory **interface_handler.h** a **interface_handler.c**, ktoré zodpovedajú za rozdelenie práce na implementované protokoly podľa toho, aká možnosť bola zvolená pri volaní z vyšších inštancií programu.

```
while (1) {
    DataIn = 0x0; TimerExp = 0x0;
    ...
    if (read_bit(ATN) == 0) {
        usleep(20);
        if (read_bit(CLK_) == 0) {
            usleep(60);
            write_bit(DATA, 0);
            while (read_bit(CLK_) != 1) {}
            usleep(60);
            write_bit(DATA, 1);                    // ready for DATA
            usleep(100);
            DataIn = read_data(&Gpio);             // receiving DATA
            usleep(60);
            write_bit(DATA, 0);                    // frame handshake
            state_machine(DataIn);
            usleep(150);
        }
    }
```

Výpis kódu 20 Ukážka časti zdrojového kódu súboru **out.c**

```
...
    else if (read_bit(CLK_) == 1 && state == OPEN_FILE) {
        usleep(60);
        write_bit(DATA, 1);
        check_eoi();                                    // check for EOI signal
        write_bit(DATA, 1);                             // ready for DATA
        DataIn = read_data(&Gpio);                      // receiving DATA
        usleep(60);
        write_bit(DATA, 0);                             // frame handshake
        state_machine(DataIn);
        usleep(100);
    }
    else if (read_bit(CLK_) == 1 && state == ON) {
    write_bit(DATA, 1);
    state_machine(DataIn);
    }
    else if (read_bit(CLK_) == 1 && state == OPEN_CHANNEL) {
        write_bit(CLK_, 0);                             // TALK-ATTENTION ACKNOWLEDGE
        usleep(500);                                    // ACK HOLD
        write_bit(CLK_, 1);                             // TALKER ready to send ?
        while (read_bit(DATA) == 1) {}
        state_machine(DataIn);
    }
    else if (state == TALKING) {
        write_bit(CLK_, 1);                             // transmit DATA now
        while (read_bit(DATA) == 0) {}
        usleep(35);
        write_bit(CLK_, 0);
        write_data(data[file_pos]);                     // DATA to send
        write_bit(DATA, 1);
        usleep(15);
        check_timeout();                                // read FRAME HANDSHAKE on DATA
        usleep(150);                                    // OK, continue
        // if end of file, issue EOI, change state
        if (finish_file != 0 && file_pos + 1 == finish_file - 1/* end of file */) {
            write_bit(CLK_, 1);
            while (read_bit(DATA) == 0) {}
            while (read_bit(DATA) == 1) {}
            write_bit(CLK_, 0);
            usleep(40);
            write_data(data[++file_pos]);               // DATA to send
            write_bit(DATA, 1);
            usleep(20);
            check_timeout();                            // read FRAME HANDSHAKE on DATA
            write_bit(CLK_, 1);
            state_machine(0x99);
            fr = f_close(&Fil);
        }
        file_pos++;
        if ((file_pos) % 6 == 0) {
            file_pos = 0;
            need = 1;
        }
    }
}
```

Výpis kódu 21 Ukážka časti zdrojového kódu súboru **out.c**; pokračovanie

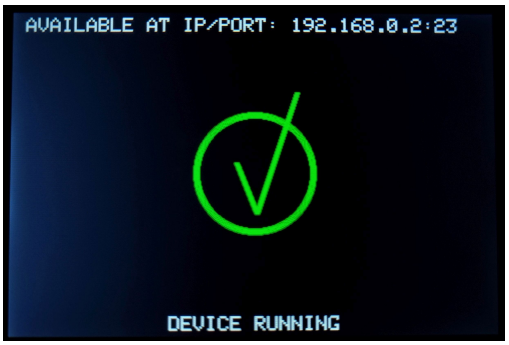
6.2 Vývoj demo aplikácie

Zmysel aplikácie je zjednotiť všetky softvérové časti práce a vytvoriť prostredie prístupné pre používateľa. Riešenie musí tak obsahovať obsluhu displeja, TCP/IP stack, ovládania SD úložiska a riadenie komunikácie konfigurovateľného rozhrania. Toto rozloženie vychádza z návrhu softvérovej časti v sekcii 4.3. Každá časť má svoje obslužné funkcie spolu s konštantami prepojené a inštanciami jednotlivých ovládačov IP jadier. Súčasťou je ukážková emulácia rozhrania CBM/IEC.

Základom je súbor **main.c**, v ktorom sa nachádza funkcia `int main(void)`. Ďalej súbory **run.h** a **run.c**, ktoré obsahujú deklarácie a definície inicializačných funkcií volaných z hlavnej funkcie. Súbory **support.h**, **support.c** riešia spomínanú inicializáciu na nižšej úrovni a taktiež aj funkčnosť aplikácie. Hlavičkový súbor **global.h** obsahuje deklarácie premenných, ktoré sú používané v prostredí aplikácie ako globálne. Súbor **instances.h** deklaruje inštancie jednotlivých IP jadier využívaných vo zvyšku programu na sprostredkovanie komunikácie s hardvérom. Konfiguračný hlavičkový súbor **config.h** obsahuje deklarácie konštánt, ktoré reflektujú následné nastavenie jednotlivých IP jadier, TCP/IP stack, atď.

Ostatné súbory sú rozdelené podľa ich prepojenia na jednotlivé moduly. Preto existujú priečinky **fatfs_module**, **interface_module**, **lcd_module** a **uip_module**. Každý z nich obsahuje hlavičkový a zdrojový súbor s názvom *nazov_modulu*_**handler.h/c**. Dvojica obsahuje funkcie, ktoré riadia svoju časť programu a odkazujú sa na inštancie a globálne premenné. Na nich sa následne odkazuje v obslužnom súbore **support.c**.

Obr. 6.1 Záznam obrazovky prototypu; stav: **initializing**



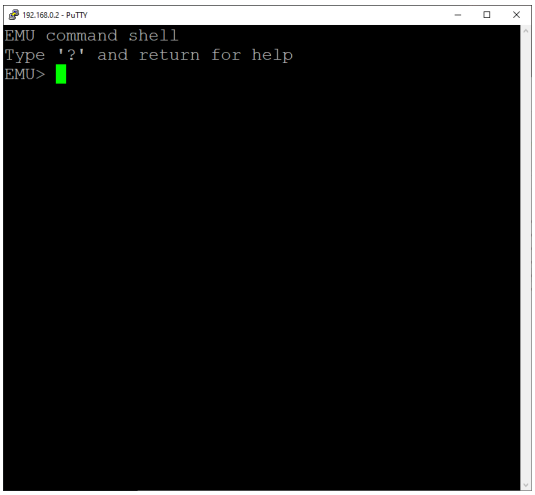
Obr. 6.2 Záznam obrazovky prototypu; stav: **running**

Obr. 6.3 Záznam obrazovky prototypu; stav: **connected**

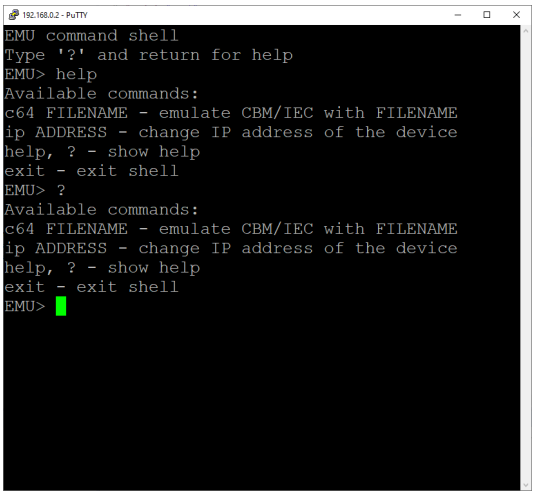
Obr. 6.4 Záznam obrazovky prototypu; stav: **emulating**

Celkový beh programu sa začína inicializáciou (obr. 6.1) všetkých používaných IP jadier postupom opísaným v sekciách od 6.1. Po úspešnom ukončení inicializácie vypísaním na displeji (obr. 6.2) systém informuje o aktuálnom stave zariadenia spolu s IP adresou a portom, na ktorých je dostupný **Telnet** server. V tomto stave čaká systém na pripojenie. Po pripojení zariadenie znova informuje používateľa o zmene stavu s vypísaním IP adresy pripájajúceho sa klienta (obr. 6.3). Ak si pripojený používateľ vyberie v príkazovom riadku emuláciu zariadenia, na displeji sa zobrazí notifikácia s názvom súboru, ktorý je emulovaný (obr. 6.4). Po úspešnom ukončení presunu dát sa systém vracia do stavu po inicializácií s aktuálnou IP adresou. Toto správanie aplikácie reflektuje na návrh konečného automatu zo sekcie 4.3.

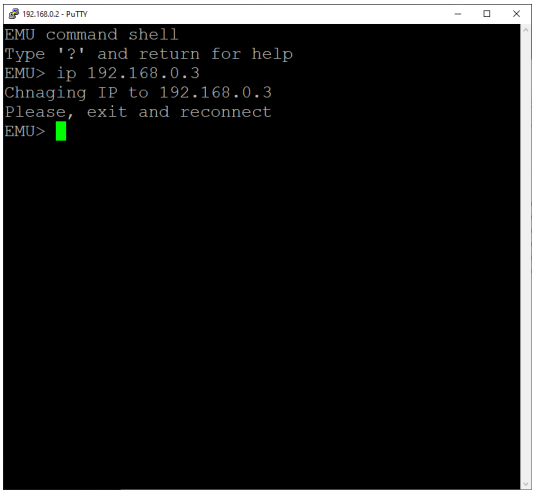
Príkazový riadok (obr. 6.5) obsahuje príkazy, cez ktoré je možné systém ovládať. Výpis všetkých z nich je možný cez zadanie **help** alebo **?** (obr. 6.6). Po zadaní jednotlivých príkazov zariadenie reaguje buď okamžitým vykonaním alebo odoslaním pokynu o uzavretie pripojenia. Príklad príkazov, ktoré si vyžadujú odpojenie sa od zariadenia sú **ip** *ip_adresa* a **c64** *nazov_suboru*. Ukážky príkazov sú na obr. 6.7, resp. obr. 6.8.



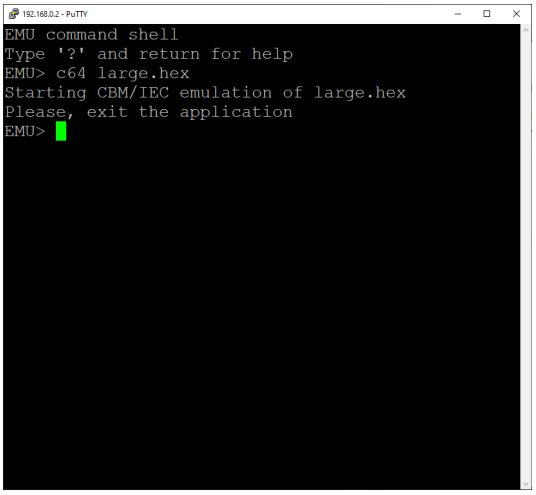
Obr. 6.5 Záznam príkazového riadku po pripojení



Obr. 6.6 Záznam príkazového riadku; príkaz: **help/?**



Obr. 6.7 Záznam príkazového riadku; príkaz: **ip**

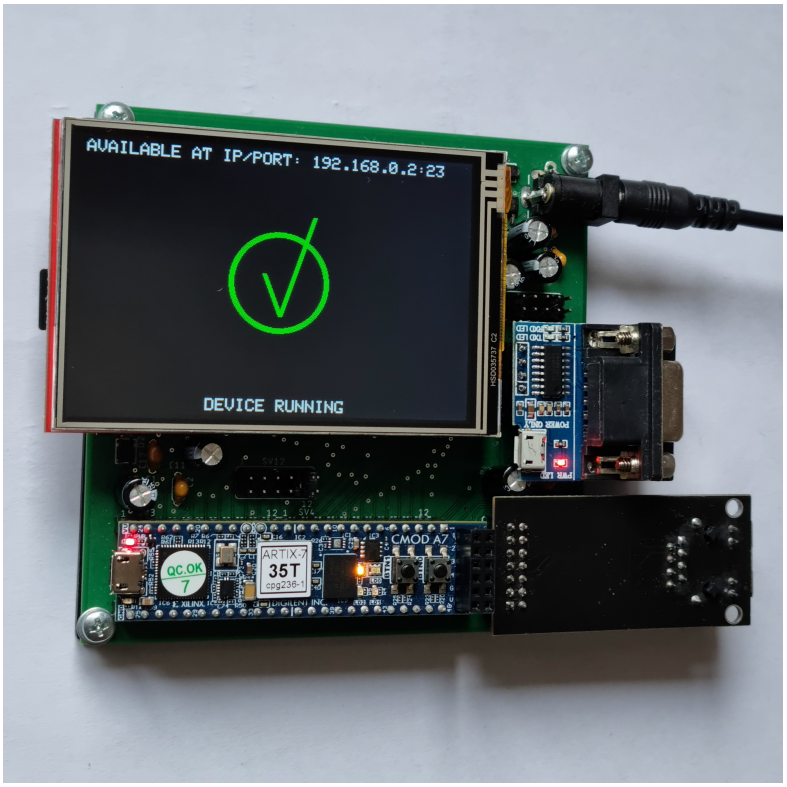


Obr. 6.8 Záznam príkazového riadku; príkaz: **c64**

Používanie konfigurovateľného rozhrania si taktiež vyžaduje odpojenie, keďže jeho komunikácia si vyžaduje presné načasovanie, ktoré môžu negatívne ovplyvniť napr. riešenie spravovania sieťovej komunikácie, časovača a iných častí systému. Preto sa pred uskutočnením komunikácie pozastavuje Telnet server, časovač a ovládanie displeja. Obslužný program beží potom samostatne. Využíva pri behu funkčnosť SD karty a modulu FatFs na prenos súborov. Po ukončení prenosu sa exekúcia programu presúva naspäť a obnovuje sa funkčnosť ostatných častí.

6.3 Zhrnutie

Po vytvorení schémy a CAD modelu dosky PCB, jej vyrobení a osadení súčiastkami spolu s modulmi je hotová hardvérová časť práce. Následnou realizáciou blokovej schémy spolu s generáciou bitstream je ukončená aj softvérová časť obsahu čipu FPGA. Implementáciou Telnet serveru, ovládania displeja a SD karty, riešením konfigurovateľného rozhrania a následného spojenia do obslužnej aplikácie je hotová softvérová časť práce. Dokopy všetky tieto časti tvoria funkčný prototyp, ukážka na obr. 6.9.



Obr. 6.9 Fotografia výsledného prototypu

Celý systém je možné pri chybe reštartovať použitím tlačidiel BTN0 na prípravku Cmod a/alebo tlačidlom reset na displej module. Toto je tzv. **hard reset**, čo znamená spustenie aplikácie nanovo a invalidáciu existujúcich údajov v pamäti RAM.

Výsledná aplikácia je uložená na čipe Flash, ktorý je súčasťou prípravku Cmod. Takto je zaručené, že zariadenie pri pripojení k napájaniu automaticky nahraje konfiguráciu čipu FPGA a následne do pamäte RAM vloží program aplikácie. Nakoniec je spustená exekúcia inštrukcií na procesore MicroBlaze a zariadenie je možné používať. Nahrávanie trvá istý čas, preto je prvých pár sekúnd prototyp nefunkčný.

Kapitola 7
Testovanie

V tejto kapitole testovania sa nachádza opis chýb pri návrhu prototypu dosky PCB, výsledky vývoja hardvérového riešenia pre čip FPGA s opisom problémových nastavení a taktiež výsledky testovania aplikácie s použitím doplňujúceho hardvéru.

Po doručení dosky PCB a pri osadzovaní komponentov sa v návrhu ukázala chyba s veľkosťou dier pre konektor s označením **J1**. Na miesto podlhovastých zdierok so šírkou 2,5 mm sú v doske okrúhle dierky s priemerom 0,75 mm. Po úprave konektora priletovaním nožičiek používaných pri tzv. pin header je možné tento konektor priletovať k doske PCB. Túto chybu je rovnako možné opraviť v EAGLE výberom iného typu konektora a následne objednať výrobu novej verzie. Okrem tejto chyby výberu konektora je PCB bez ďalších chýb.

Pripojenie jednotlivých modulov k doske PCB je z väčšej časti bezchybné okrem pripojenia modulu prevodníka UART – RS-232, ktorý pri zasunutí do konektora ostáva naklonený. Toto ale nijako neobmedzuje funkčnosť prototypu. Problém je spôsobený pozíciou kondenzátora s označením **C7**, ktorého horná časť mierne prekáža prevodníku na ľavej strane. Tento problém je možné odstrániť zmenením jeho pozície, použitím nižšieho kondenzátora alebo inštaláciou kondenzátora v inej orientácií (naležato).

Vývoj hardvérového riešenia pre čip FPGA funguje pri správnom zvolení možností bez problémov. Rovnako funguje načítavanie konfigurácie z pamäte Flash dostupnej na prípravku Cmod. Počas vývoja bol identifikovaný problém IP jadra **Ethernet PHY MII to Reduced MII**, ktorý spôsobuje, že medzi PHY a MAC neprebieha komunikácia. Táto chyba nastane pri zvolení inej kombinácie dostupných možností ako tej, ktorá je predvolená pri jeho vytvorení. Pokiaľ je zachované pôvodné nastavenie, IP jadro funguje bez problémov. Ďalšie problémy sa vyskytovali pri počiatočnej konfigurácií modulu **MicroBlaze**, kde pri zvolení **Application preset** a veľkosti internej RAM nad **64 kB** systém automaticky vytvorí inštanciu spomínaného procesora tak, že sa pri postupe generácie tzv. bitstream vyskytne chyba nadmernej alokácie BRAM. To znamená, že návrh si vyžaduje viac BRAM, ako je dostupných na čipe FPGA (v tomto prípade model XC7A35T s obsahom 50 blokov BRAM). Riešenie tejto chybovej hlášky si vyžaduje zníženie veľkosti pamäte internej RAM, resp. zníženie veľkosti **Data Cache** a/alebo **Instruction Cache**. Rovnako je možné nepoužívať pri návrhu cache a/alebo internú RAM. Toto ale vedie k použitiu externej SRAM, ktorá je značne pomalšia.

Pri pohľade na celkové využitie zdrojov čipu FPGA prípravku Cmod pri rovnakej konfigurácií modulov opísanej v sekcií 5.3 sa nachádza v tab. 7.1. Na vyššom využití dostupných zdrojov sa podpisuje používanie cache a ladiaceho modulu spolu s internou RAM. Ak by boli tieto časti vypnuté a odstránené z návrhu, bolo by možné ušetriť značnú časť prostriedkov, porovnanie prípadov je v tab. 7.2.

Tabuľka 7.1 Využitie zdrojov čipu FPGA prípravku Cmod A7-35T

Typ zdroja	Dostupné zdroje	Využité zdroje	Využitie v %
LUT	20800	6406 + 1	30,80
LUTRAM	9600	736	7,67
FF	41600	5615	13,50
BRAMs	50	46	92,00
DSP	90	3	3,33
IO	106	75	70,75
BUFG	32	6	18,75
MMCM	5	1	20,00

Využitie zdrojov ako **LUT**, **LUTRAM**, **FF**, **DSP** je pri použití základného riešenia približne rovný 1/3, čo dáva veľký priestor pre dodanie ďalších IP jadier a/alebo iných RTL blokov k už existujúcemu riešeniu. Takéto rozšírenie by mohlo obmedziť 92% využitie **BRAMs**, čo je spôsobené používaním internej RAM a celkom veľkej cache. Nahradiť tento zdroj by bolo možné použitím **LUTRAM**, ale iba v obmedzenom počte. Údaje o využití **IO** a zvyšných zdrojov hovoria o tom, že veľká časť dostupných pinov obvodu FPGA samotného je priradená a tým pádom využívaná.

Tabuľka 7.2 Porovnanie využitia zdrojov čipu FPGA pri minimálnej/optimálnej konfigurácií

Typ zdroja	Pôvodný počet	Počet po redukcii	Rozdiel využitia v %
LUT	6406 + 1	3623 + 1	13,38
LUTRAM	736	506	2,4
FF	5615	3043	6,19
BRAMs	46	4	84,00
DSP	3	3	0
IO	75	75	0
BUFG	6	6	0
MMCM	1	1	0

Po odstránení všetkých interných pamätí (RAM, cache), nepotrebných modulov (ladenie) a rekonfigurácií modulu **MicroBlaze** je pokles využitia zdrojov značný. Podľa očakávaní je najvyššia úspora u zdroja **BRAMs**. U ostatných zdrojov ako **LUT**, **LUTRAM** a **FF** je úspora minimálna, pretože odstránené bloky nie sú veľmi komplexné a zvyšné ostávajú nezmenené.

Softvérové vybavenie taktiež využíva istú časť zdrojov. Tu sa jedná hlavne o veľkosť pamäte, ktorú si program vyžaduje pre správny chod. Rozdiel je badateľný pri využití veľkostnej optimalizácie kompilátora, ktorá je povolená pri konfigurácií **Release**. Pôvodná zmena z **Debug** na **Release** konfiguráciu aplikácie mala menší vplyv na jej veľkosť, pretože používala len optimalizáciu **-O2**. Porovnanie pamäťovej náročnosti podľa zvolenej optimalizácie je v tab. 7.3. Pre dáta v tabuľke sa počíta so základom RAM o veľkosti 128 kB, ktorá je interná. Pri použití externej pamäte je možné počítať s kapacitou 512 kB.

Tabuľka 7.3 Porovnanie využitia pamäte programom bez použitia a s použitím optimalizácie (v bajtoch)

Optimalizácia	text	data	bss	dec	Využitie pamäte v %
-O0	103976	504	14840	119320	93,22
-Os	82688	504	14832	98024	76,58
Rozdiel	20228	0	8	21296	16,64

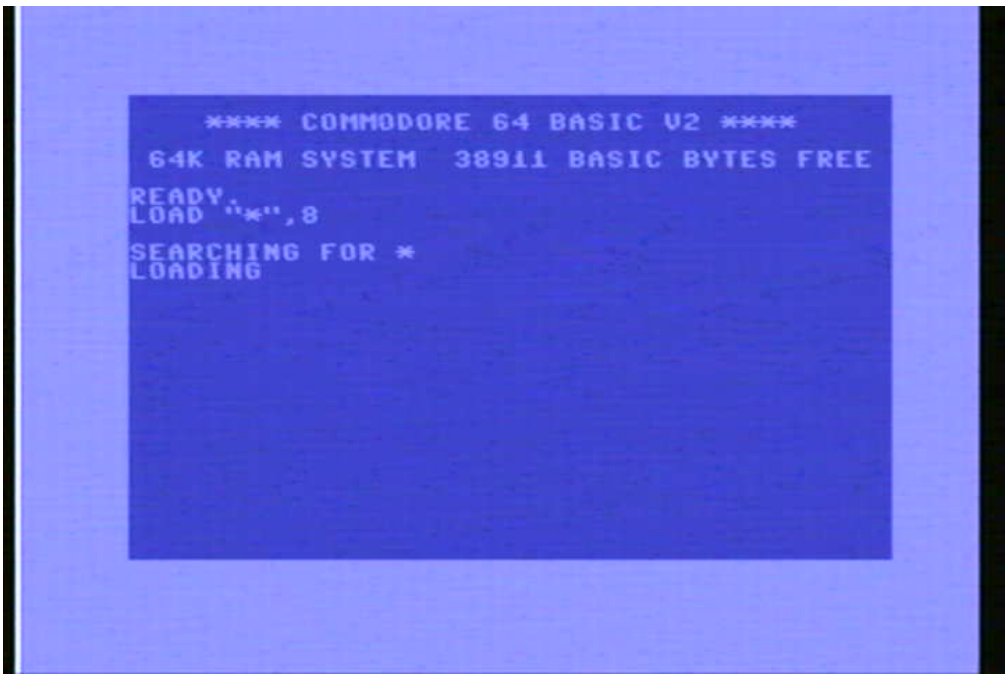
Použitím veľkostnej optimalizácie je možné ušetriť aspoň 21 kB, čo predstavuje značnú časť pri použití internej pamäte. Veľkostne to zodpovedá jednej ikone (rozlíšenie 100x100 px s 16-bitovou farebnou hĺbkou) používanej pri zobrazovaní stavu emulácie na displeji. Ak pri vývoji nebude postačovať kapacita internej pamäte, je možné využiť tú externú. Pri tejto zmene je potrebné počítať s veľkým rozdielom v čase prístupu k uloženým dátam, keďže sa k externej pamäti pristupuje cez rozhranie **AXI** a signály prechádzajú cez viacero registrov I/O portov pred tým, ako dorazia k pamäti. Porovnanie rýchlostí týchto možností je možné použitím základného príkladu prepísania všetkých riadkov displeja na čiernu farbu po inicializácií. Výsledky sú uvedené v tab. 7.4.

Tabuľka 7.4 Časové porovnanie výpisu na displej pri použití externej SRAM a internej BRAM

Externá SRAM	Interná BRAM	Rozdiel
~24,18s	~1,85s	~22,33s (92,35%)

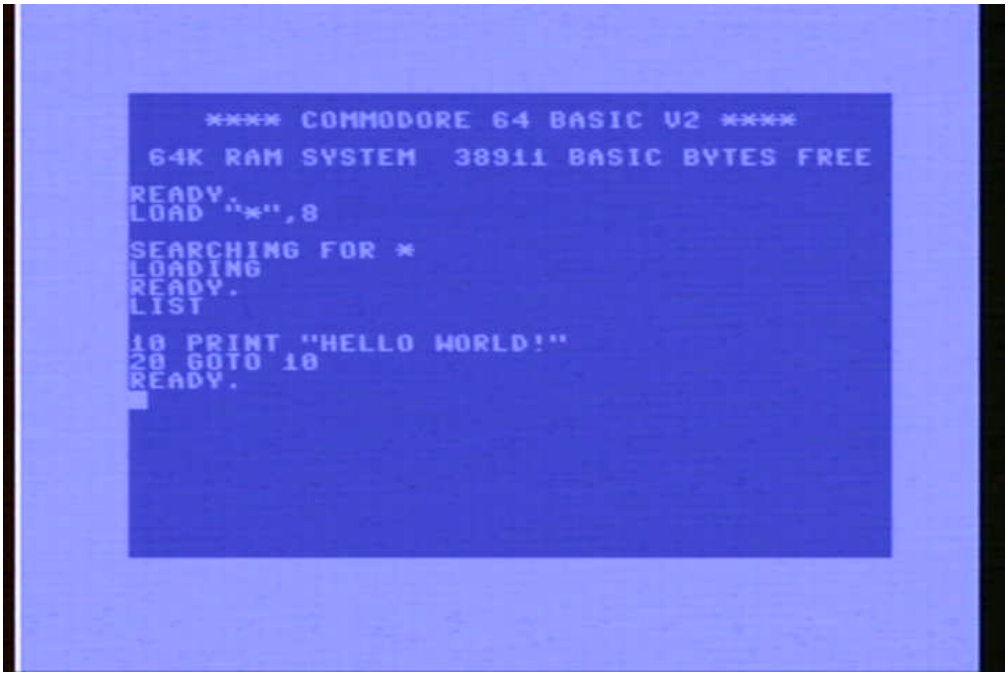
Priemerný rozdiel v časoch vykresľovania na displej je extrémny. Preto je možné predpokladať, že použitý príklad poskytuje skreslené údaje. Na druhej strane bol rozdiel pri testovaní dostatočne veľký na to, že pri emulácií rozhrania CBM/IEC dochádzalo vo viacerých prípadoch k problémom s komunikáciou (vypršanie časového limitu na odpoveď).

Emulácia rozhrania CBM/IEC vo väčšine prípadov funguje bez problémov. Po zadaní príkazu s validným názvom súboru, ktorý sa nachádza na SD karte, cez príkazový riadok prejde zariadenie do módu emulácie. V tomto móde čaká na komunikáciu zo strany počítača. Po zadaní príkazu **LOAD "*",8** na strane **C64** zariadenie odpovedá odoslaním súboru, ukážka strany počítača na obr. 7.1.



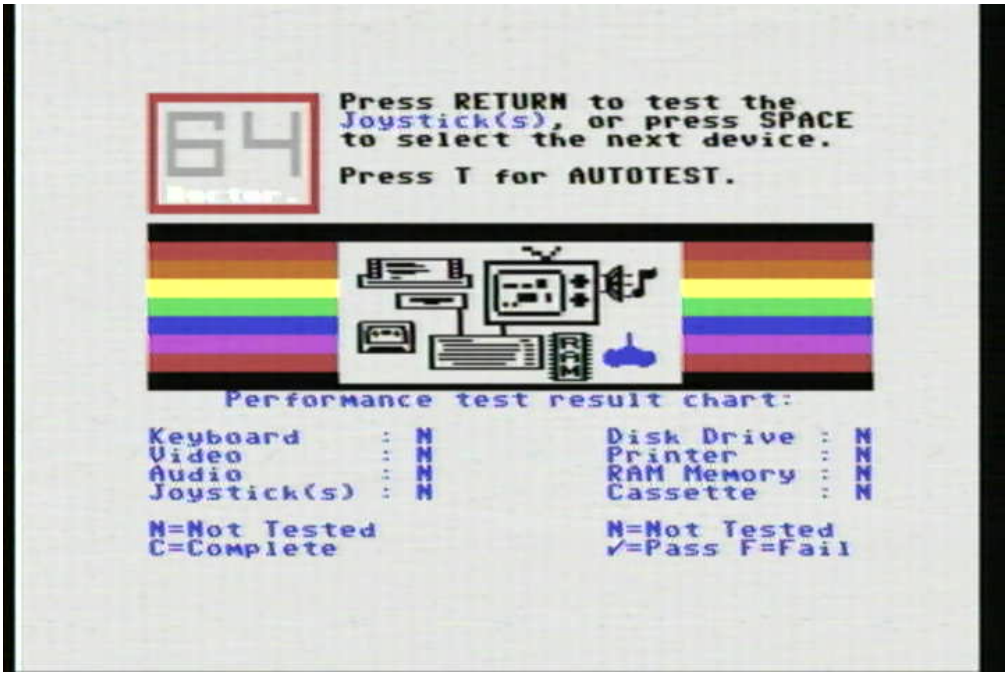
Obr. 7.1 Záznam obrazovky počítača C64; načítavanie súboru

Po úspešnom ukončení načítavania súboru do pamäte prechádza zariadenie naspäť do stavu **running**, tak ako je to opísané v sekcií 6.2. Na strane počítača sa načítavanie taktiež ukončí, čo je možné skontrolovať zadaním príkazu **LIST**, ktorý vypíše obsah prijatého programu, obr. 7.2. Zadaním príkazu **RUN** je možné tento nahratý program spustiť z pamäte.



Obr. 7.2 Záznam obrazovky počítača C64; výpis obsahu pamäte

Iný príklad načítavania súboru je posielanie väčšieho programu ako je napr. **64 Doctor**. Tento program slúži na diagnostiku systému a jeho veľkosť je 13,53 kB. Načítanie takéhoto súboru je časovo náročnejšie, pretože maximálna prenosová rýchlosť originálneho rozhrania **CBM/IEC** je 400 B/s, efektívny bitrate 3200 b/s [22]. Ukážka po spustení na obr. 7.3, čo znamená úspešný prenos.



Obr. 7.3 Záznam obrazovky počítača C64; spustenie programu **64 Doctor**

Záver

Cieľom bakalárskej práce bolo vytvoriť programovateľný riadiaci systém pre ovládanie periférií náročných na presné časovanie. Súčasťou práce bolo navrhnúť dosku plošného spoja a vytvoriť prototyp zariadenia. Tento hlavný cieľ je splnený a existuje funkčné riešenie.

V prvej časti analýzy boli preskúmané dostupné riešenia, analyzované použité metódy, technológie návrhu, ich kladné a záporné stránky. Na konci kapitoly sú vyvodené závery, podľa ktorých je vykonaná nadväzujúca analýza riadiacich prvkov, dostupných modulov a súčiastok potrebných na vytvorenie prototypu s potrebnou úrovňou flexibility. Voľba architektúry, na ktorej je postavené výsledné zariadenie, bola vykonaná na základe preskúmania vlastností existujúcich riešení. Súčasťou rozhodovacieho procesu boli aj úvahy o možnosti rozšírenia práce. Po zvolení tých najlepších komponentov na základe ich vlastností bola vykonaná dodatočná analýza.

Po analýze protokolov a rozhraní využívaných týmito modulmi a potrebných pre komunikáciu boli vyvodené dôležité poznatky. Zjednodušil sa tak prístup k implementácii softvérového riešenia aplikačnej časti využitím existujúcich knižníc a IP jadier dostupných vo vývojovom softvérovom balíku. Použitie knižníc si vyžadovalo ich modifikáciu a pridanie podporných IP jadier, ktoré realizujú funkčnosti vyžadované niektorými z týchto knižníc. Analýza obohatila znalosti autora.

Výsledkom praktickej časti je vytvorenie riešenia schémy, CAD modelu a tak aj integračnej dosky PCB. Doska bola osadená súčiastkami, modulmi a prípravkom Cmod A7-35T, čo tvorí výsledné hardvérové riešenie. Rovnako je výsledkom praktickej časti bloková schéma použiteľná pre čip FPGA s IP jadrami, ktoré riešia komunikáciu s periférami. K zariadeniu je aj preto možné pripojiť periférie cez konfigurovateľné rozhranie a/alebo pomocou rozhrania RS-232.

Softvérová výbava prototypu sa skladá z viacerých modulov, ktoré vzájomne spolupracujú a tvoria tak funkčnú obsluhu zariadenia. Po pripojení k počítačovej sieti prototyp čaká na pripojenie klienta k Telnet serveru. Následným zadávaním príkazov cez tento protokol je možné zariadenie ovládať. Demo aplikácia prijatím špecifického príkazu reaguje emuláciou disketovej mechaniky počítača Commodore 64, kde sú súbory čítané z SD karty. Aktuálny stav zariadenia je opísaný na LCD displeji. Zároveň je softvérová výbava nahraná v pamäti Flash prípravku Cmod, takže pri pripojení prototypu k zdroja napätia je možné zariadenie používať bez potreby dodatočného programovania. Spojením výsledkov praktickej časti vzniklo funkčné zariadenie.

Súčasťou práce bolo testovanie realizovaných častí zariadenia, ako je rozmiestnenie komponentov na doske PCB, funkčnosť servera Telnet, vypisovania na displej, prenosu z SD karty a funkčnosť emulácie periférie komunikáciou s počítačom C64. Rovnako bola otestovaná redukcia využitia zdrojov čipu FPGA, optimalizácia programu aplikácie a použitie externej RAM.

Práca je rozšíriteľná o ďalšie funkcionality ako napr. pridanie schopností emulovať dodatočné protokoly ako Atari SIO, ZX Interface 1, Z-MODEM, atď; zmena blokovej schémy a pridanie ďalších IP jadier na rozšírenie možností zariadenia. Rovnako je možné použiť prototyp pri inom projekte, ktorý môže využiť používané moduly a/alebo konfigurovateľné rozhrania na logickej úrovni 5 V alebo 3,3 V.

Bibliografia

1. WHITE, Stephen James. *Pi1541* [online]. 2024. [cit. 2024-01-06]. Dostupné z: https://cbm-pi1541.firebaseapp.com/.
2. KBR. *SDrive-MAX - Atari 8bit Floppy-Emulator with display* [online]. 2024. [cit. 2024-01-06]. Dostupné z: http://www.kbrnet.de/projekte/sdrive-max/index.html.
3. DALBY, Tom. *Raspberry Pico ZX Spectrum Microdrive Hardware Emulator* [online]. 2024. [cit. 2024-01-06]. Dostupné z: https://github.com/TomDDG/ZXPicoMD.
4. *megaAVR®Data Sheet* [online]. Microchip Technology Inc., 2020. [cit. 2024-01-10]. Dostupné z: https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf. (str. 2).
5. *CoolRunner II* [online]. Advanced Micro Devices Inc., 2024. [cit. 2024-01-10]. Dostupné z: https://www.xilinx.com/products/silicon-devices/cpld/coolrunner-ii.html.
6. *Artix 7 FPGA Family* [online]. Advanced Micro Devices Inc., 2024. [cit. 2024-01-10]. Dostupné z: https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html.
7. *Cmod A7 Reference Manual* [online]. Digilent Inc., 2019. [cit. 2024-01-10]. Dostupné z: https://digilent.com/reference/programmable-logic/cmod-a7/reference-manual.
8. *Vivado 2018.3 Downloads* [soft.]. Advanced Micro Devices Inc., 2018-12-14. UG973 (v2018.3) [cit. 2024-01-10]. Dostupné z: https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html. (dostupné v sekcii 2018.3, Vivado Design Suite - HLx Editions - 2018.3 Full Product Installation).
9. *Basys 3 Reference Manual* [online]. Digilent Inc., 2019. [cit. 2024-01-10]. Dostupné z: https://digilent.com/reference/programmable-logic/basys-3/reference-manual.
10. *Arty A7 Reference Manual* [online]. Digilent Inc., 2018. [cit. 2024-01-10]. Dostupné z: https://digilent.com/reference/programmable-logic/arty-a7/reference-manual.
11. *Nexys A7 Reference Manual* [online]. Digilent Inc., 2019. [cit. 2024-01-10]. Dostupné z: https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual.
12. *W5500 Datasheet* [online]. WIZnet Co., Ltd., 2013. [cit. 2024-01-11]. Dostupné z: https://www.laskakit.cz/user/related_files/w5500_datasheet.pdf. (str. 2).
13. *Serial to Ethernet Converter USR-TCP232-T2* [online]. Jinan USR IOT Technology Limited, 2016. [cit. 2024-01-11]. Dostupné z: https://www.laskakit.cz/user/related_files/usr-tcp232-t2.pdf.
14. *Small Footprint RMII 10/100 Ethernet Transceiver with HP Auto-MDIX Support* [online]. SMSC, 2010. [cit. 2024-01-11]. Dostupné z: https://www.laskakit.cz/user/related_files/lan8720a.pdf. (str. 1).

15. *1.3inch IPS Module* [online]. www.lcdwiki.com, 2023. [cit. 2024-01-11]. Dostupné z: http://www.lcdwiki.com/1.3inch_IPS_Module.

16. *3.5inch Arduino Display-UNO* [online]. www.lcdwiki.com, 2022. [cit. 2024-01-11]. Dostupné z: http://www.lcdwiki.com/3.5inch_Arduino_Display-UNO.

17. *NX4024K032 - ITEAD Wiki* [online]. ITEAD Intelligent Systems Co., Ltd., 2017. [cit. 2024-01-11]. Dostupné z: https://wiki.iteadstudio.com/NX4024K032.

18. *4inch HDMI Display-C* [online]. www.lcdwiki.com, 2023. [cit. 2024-01-11]. Dostupné z: http://www.lcdwiki.com/4inch_HDMI_Display-C.

19. *MAX3222 3-V to 5.5-V Multichannel RS-232 Line Driver and Receiver With ±15-kV ESD Protection* [online]. Texas Instruments Inc., 2016. [cit. 2024-04-21]. Dostupné z: https://www.ti.com/lit/ds/symlink/max3222.pdf?ts=1713711021623. (str. 4-6).

20. *TXB0108 8-Bit Bidirectional Voltage-Level Translator with Auto-Direction Sensing and ±15-kV ESD Protection* [online]. Texas Instruments Inc., 2020. [cit. 2024-05-06]. Dostupné z: https://www.ti.com/lit/ds/symlink/txb0108.pdf. (str. 1, 9, 13).

21. *SparkFun Logic Level Converter - Bi-Directional* [online]. SparkFun Electronics, 2013. [cit. 2014-04-21]. Dostupné z: https://www.sparkfun.com/products/12009.

22. DEROGEE, J. *IEC disected* [online]. 2008. [cit. 2024-04-21]. Dostupné z: https://www.commodoreserver.com/FileDownload.asp?FID=1A8DEC655F34480F846540B6215493EE. (str. 2, 3, 9-12, 15).

23. *Commodore 64 Programmer's Reference Guide* [online]. Commodore Business Machines, Inc., 1982. [cit. 2024-04-22]. Dostupné z: http://cini.classiccmp.org/pdf/Commodore/C64%20Programmer%27s%20Reference%20Guide.pdf. (str. 382).

24. *SL-USB8CH logický analyzátor - 8 kanálů, 24MHz | LaskaKit* [online]. laskakit.cz, 2024. [cit. 2024-05-05]. Dostupné z: https://www.laskakit.cz/sl-usb8ch-logicky-analyzator-8-kanalu--24mhz/.

25. *PulseView - sigrok* [online]. sigrok, 2023. [cit. 2024-05-05]. Dostupné z: https://sigrok.org/wiki/PulseView.

26. IEEE Standards for Local and Metropolitan Area Networks: Supplement - Media Access Control (MAC) Parameters, Physical Layer, Medium Attachment Units, and Repeater for 100Mb/s Operation, Type 100BASE-T (Clauses 21-30). *IEEE Std 802.3u-1995 (Supplement to ISO/IEC 8802-3: 1993; ANSI/IEEE Std 802.3, 1993 Edition)*. 1995. Dostupné z DOI: 10.1109/IEEESTD.1995.7974916. (str. 27, 42-48).

27. *RMII™ Specification* [online]. RMII Consortium™, 1998. [cit. 2024-01-11]. Dostupné z: http://ebook.pldworld.com/_eBook/-Telecommunications,Networks-/TCPIP/RMII/rmii_rev12.pdf. (str. 1-9).

28. *MIPI Alliance Standard for Display Bus Interface* [online]. MIPI Alliance, 2005. [cit. 2024-01-11]. Dostupné z: http://picture.iczhiku.com/resource/eetop/sHkwuKJsUOrfTXMn.pdf. (str. 5-17).

29. *a-Si TFT LCD Single Chip Driver 320RGB x 480 Resolution and 16.7M-color* [online]. ILI Technology Corp., 2012. [cit. 2024-01-12]. Dostupné z: https://focuslcds.com/wp-content/uploads/Drivers/ILI9488.pdf. (str. 16).

30. *Schéma 3.5" 320x480 TFT displej ILI9488, shield Arduino Uno* [online]. laskakit.cz, 2023. [cit. 2024-01-12]. Dostupné z: https://www.laskakit.cz/user/related_files/tft-3-5-uno-ili9488-sch.jpg.

31. *M68HC11 Reference Manual* [online]. Motorola, Inc., 1991. [cit. 2024-01-12]. Dostupné z: https://archive.org/details/bitsavers_motorola68ferenceManualRev3_31028881. (sekcia 8 str. 1-3).

32. *Using the Serial Peripheral Interface to Communicate Between Multiple Microcomputers* [online]. Motorola, Inc., 1987. [cit. 2024-01-13]. Dostupné z: `http://www.bitsavers.org/components/motorola/_appNotes/AN-0991_Using_the_Serial_Peripheral_Interface_to_Communicate_between_Multiple_Microcomputers.pdf`. (str. 6).
33. DAVIS, Larry. *Connector Manufacturers, Headers and Berg Stick Manufacturers* [online]. 2015. [cit. 2024-05-03]. Dostupné z: `http://www.interfacebus.com/connector-header-manufacturers.html`.
34. AUTODESK, Inc. *Download EAGLE | Free Download | Autodesk* [online]. 2024. [cit. 2024-04-02]. Dostupné z: `https://www.autodesk.com/products/eagle/free-download`.
35. *Digilent/vivado-boards* [online]. Digilent, Inc., 2023. [cit. 2024-05-03]. Dostupné z: `https://github.com/Digilent/vivado-boards`. (možnosť stiahnúť ako zip).
36. *Board Support Package Settings Page • Vitis Unified Software Platform Documentation: Embedded Software Development (UG1400) • Reader • AMD Technical Information Portal* [online]. AMD, Inc., 2020. [cit. 2024-05-04]. Dostupné z: `https://docs.amd.com/r/2020.2-English/ug1400-vitis-embedded/Board-Support-Package-Settings-Page`.
37. *Flashing a MicroBlaze Program* [online]. instructables.com, 2020. [cit. 2024-05-01]. Dostupné z: `https://www.instructables.com/Flashing-a-MicroBlaze-Program/`.
38. *embeddedsw/XilinxProcessorIPLib/drivers at master | Xilinx/embeddedsw* [online]. Xilinx, Inc., 2023. [cit. 2024-04-28]. Dostupné z: `https://github.com/Xilinx/embeddedsw/tree/master/XilinxProcessorIPLib/drivers`. (zdrojové súbory sa nachádzajú v podpriečinkoch).
39. *3.5inch Arduino Display-UNO* [online]. www.lcdwiki.com, 2022. [cit. 2024-04-17]. Dostupné z: `http://www.lcdwiki.com/3.5inch_Arduino_Display-UNO`.
40. *FatFs - Generic FAT Filesystem Module* [online]. ChaN, 2022. [cit. 2024-04-19]. Dostupné z: `http://elm-chan.org/fsw/ff/`.
41. *SD Specifications | Part 1 | Physical Layer | Simplified Specification* [online]. SD Card Association, 2017. [cit. 2024-04-19]. Dostupné z: `https://academy.cba.mit.edu/classes/networking_communications/SD/SD.pdf`. (str. 210 - 216).
42. DUNKELS, Adam. *The historical uIP sources* [online]. 2013. [cit. 2014-04-20]. Dostupné z: `https://github.com/adamdunkels/uip`.

Obsah príloh

- `readme.txt` ..... stručný popis obsahu média
- `bit` ..... adresár s binárnou formou implementácie
- `src`
  - `impl` ..... zdrojové kódy hardvéru a demo aplikácie
    - `eagle` ..... zdrojové súbory PCB pre Autodesk EAGLE
    - `sdk` ..... zdrojové súbory demo aplikácie pre Xilinx SDK
    - `vivado` ..... zdrojové súbory FPGA konfigurácie pre Xilinx Vivado
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF