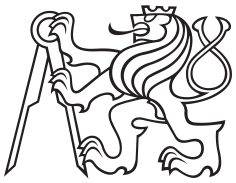
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra telekomunikační techniky

Komunikace s přípravkem Spartan-3E pomocí rozhraní RS232

Jan Šedivý

Vedoucí práce: Ing. Pavel Lafata, Ph.D.
Studijní program: Elektronika a komunikace
Květen 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Šedivý** Jméno: **Jan** Osobní číslo: **483884**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra telekomunikační techniky**

Studijní program: **Elektronika a komunikace**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Komunikace s přípravkem Spartan3E pomocí rozhraní RS232

Název bakalářské práce anglicky:

Using RS232 Interface of Spartan3E Kit

Pokyny pro vypracování:

Seznamte se s přípravkem Xilinx Spartan 3E a jeho obsluhou pomocí jazyka VHDL. Vytvořte základní kódy v jazyce VHDL pro realizaci obousměrné komunikace mezi přípravkem a PC připojeným prostřednictvím sériového rozhraní RS232 a hyperterminálu (nebo podobného programu na PC). Z připojeného PC by mělo být možné ovládat základní prvky na přípravku, např. stavové LED diody, naopak přípravek Spartan 3E by měl po stisknutí tlačítek či přepnutí přepínačů odeslat do PC předem stanovené zprávy (znaky). Rozšiřte VHDL kódy o obsluhu znakového LCD displeje na přípravku a také o standardizovaný výstup VGA tak, aby textový řetězec odeslaný z PC byl zobrazen na LCD displeji a monitoru připojeném přes VGA k přípravku.

Seznam doporučené literatury:

[1] P. J. Ashender, The VHDL Cookbook (First Edition), dostupné na: https://www.ics.uci.edu/~alexv/154/VHDL-Cookbook.pdf
[2] Manuály a datasheety k přípravku Spartan 3E

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Pavel Lafata, Ph.D., katedra telekomunikační techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **26.01.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **30.09.2022**

Ing. Pavel Lafata, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studenta

Poděkování

Chtěl bych tímto poděkovat svému vedoucímu práce, Ing. Pavlu Lafatovi, Ph.D., za pomoc při tvorbě této práce a také za zapůjčení vývojové desky Spartan-3E včetně potřebného příslušenství.

Prohlášení

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.

V Praze, 21. května 2021

.....

Abstrakt

Tato bakalářská práce popisuje návrh číslicových obvodů pomocí jazyka VHDL. Pro implementaci návrhu byl použit přípravek s hradlovým polem Spartan-3E. Nejprve je vytvořena obousměrná komunikace přes rozhraní RS-232 s počítačem. Dále je implementován výstup přijatého textu na LCD displej a na VGA rozhraní. Pro funkci VGA výstupu byla použita bloková paměť FPGA.

Klíčová slova: FPGA, VHDL, Spartan-3E, RS-232, UART, LCD displej, VGA, rastrový font

Vedoucí práce:
Ing. Pavel Lafata, Ph.D.

Abstract

This thesis describes the design of digital circuits using the VHDL language. For the implementation of the design, the Spartan-3E FPGA board was used. Firstly, a bidirectional communication with a computer through the RS-232 interface is created. Furthermore, an output of the received text on the LCD screen and the VGA interface is implemented. For the VGA output feature, the FPGA's block memory was used.

Keywords: FPGA, VHDL, Spartan-3E, RS-232, UART, LCD screen, VGA, bitmap font

Title translation: Using RS232 Interface of Spartan3E Kit

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

V této bakalářské práci se zabývám návrhem číslicových obvodů na programovatelném hradlovém poli v jazyce VHDL. K návrhu jsem použil přípravek Spartan-3E Starter Board od firmy Xilinx. S obsluhou přípravku pomocí jazyka VHDL jsem měl možnost se seznámit v rámci semestrálního projektu, který předcházel této práci.

Cílem práce bylo v jazyce VHDL navrhnout obvody pro obousměrnou komunikaci s počítačem prostřednictvím rozhraní RS-232. Dále bylo úkolem rozšířit návrh o funkce znakového výstupu na LCD zobrazovač umístěný na přípravku a na rozhraní VGA. V textu práce popisuji cesty k vytyčeným cílům spolu s dosaženými výsledky.

Text práce má dvě stěžejní kapitoly. V teoretické části uvádím základní pojmy a principy k uvedení do oblasti zájmu práce. Praktická část se potom zabývá tím, jak jsem jednotlivé funkční celky navrhoval, a tím, jak pracují. Ukazuji zde také výsledky realizovaných obvodů na fotografiích.

Poznámka ke grafické úpravě

V této práci jsou `strojovým písmem` značeny:

- názvy signálů, entit, portů, klíčových slov VHDL a dalších součástí kódu,
- názvy souborů a výpis jejich obsahu,
- příkazy příkazové řádky a nebo názvy programů s textovým rozhraním,
- kombinace kláves,
- webové odkazy.

Čísla v šestnáctkové soustavě značím bezpatkovým písmem a příponou „h“ (hexadecimální).

Kapitola 2
Teoretická část

V této kapitole uvádím a popisuji nejdůležitější pojmy a principy, které jsem při tvoření práce využil. Nejdříve představuji programovatelná hradlová pole spolu s jazykem VHDL. Jako další uvádím užité technické vybavení (hardware) a programové vybavení (software).

V navazujících částech popisuji princip komunikace se znakovým LCD zobrazovačem, komunikaci přes rozhraní RS-232 (UART) a výstup prostřednictvím grafického rozhraní VGA. Nakonec se krátce zmiňuji o kódování znaků.

2.1 Programovatelné hradlové pole

Programovatelné hradlové pole, často značeno anglickou zkratkou FPGA (Field Programmable Gate Array), je uživatelsky konfigurovatelný logický integrovaný obvod. Obsahuje řadu logických funkčních bloků, jejichž chování a vzájemné propojení je možné naprogramovat.

Hradlová pole řady Spartan-3 výrobce Xilinx mají podle datového listu [1] pět základních druhů funkčních bloků:

- Konfigurovatelné logické bloky – implementují logické funkce a zároveň obsahují klopné obvody, které fungují jako elementární paměť.
- Vstupně-výstupní bloky – mají za úkol spojovat vnitřní logiku s fyzickými vývody integrovaného obvodu. Podporují různé standardy logických úrovní.
- Bloky paměti RAM – o nich se zmíním dále.
- Násobičky – umí násobit dvě 18bitová čísla. V této práci jsem je nevyužil.
- Bloky pro správu hodinových signálů – zajišťují distribuci, dělení kmitočtů a některé další funkce.

2.1.1 Bloková paměť SelectRAM

Hradlová pole Xilinx obsahují samostatné bloky paměti nazvané „SelectRAM“. Pro případy, kdy je potřeba zpracovat data o velikosti v řádu jednotek až

desítek kB, lze s výhodou použít tyto zabudované bloky paměti. Nemusí se tak připojovat externí paměťový čip a implementovat jeho rozhraní.

Datový list [1] uvádí, že mnou použitý Spartan-3E má v sobě dostupných celkem 20 bloků této paměti. Každý blok je tvořen 18432 bity statické RAM paměti. Z toho data tvoří 16384 bitů (2 kB).

Fyzicky jsou k paměti připojeny dva nezávislé porty (brány) pro zápis a čtení. Každá brána obsahuje datové, adresní a řídicí signály, dále má také vstup hodinového signálu. Bloky mohou být skládány dohromady pro větší paměťový prostor, může se také volit z několika různých šířek datové sběrnice (např. 1, 2, 4 nebo 8 bitů).

2.1.2 Poznámka: značení pamětí

Podle učebnice [2] je RAM (Random Access Memory, překládané jako *paměť s libovolným přístupem*) protějškem paměti se sériovým přístupem. Ke každému místu paměti RAM lze libovolně přistupovat nezávisle na adrese, ke které se předtím přistupovalo. Podle definice toto označení nevypovídá o tom, zda paměť lze přepisovat.

Správně by se přepisovatelná paměť (s libovolným přístupem) měla nazvat RWM-RAM (Read-Write Memory-Random Access Memory). Protějškem paměti RWM je ROM (Read Only Memory) čili paměť pouze pro čtení[1].

V této práci označením RAM, ačkoliv nepřesně, značím paměť, která je určena k přepisování (RWM-RAM). Takovéto značení je i v praxi běžně používáno.

2.2 Jazyk VHDL

Jazyk VHDL – VHSIC (Very High Speed Integrated Circuit) Hardware Description Language – je jazyk pro popis logických číslicových obvodů. Popisovat v něm lze jak kombinační, tak sekvenční obvody a jejich kombinace. Tento jazyk se používá k realizaci logických obvodů v hradlových polích.

Není to programovací jazyk v tom smyslu, že by popisoval posloupnost příkazů, ale určuje se v něm chování a propojení logických celků. Oproti například jazyku C se v kódu nerozlišují velká a malá písmena. Lze například psát `signal` i `SIGNAL`, výsledek je stejný.

2.2.1 Struktura jazyka

Základním stavebním blokem ve VHDL je entita, která představuje „zapouzdřený“ logický obvod a s okolím komunikuje pomocí definovaných vstupů a výstupů (portů). Každá entita může být implementována více různými tzv. architekturami. Jedna entita může obsahovat další podřazené entity, neboli komponenty. Ty se deklarují v architektuře, kde se dále vytvoří jejich instance (neboli realizace, kopie). Každá instance se posléze propojí s okolními obvody pomocí tzv. signálů.

[1] Mohla by být označena i jako ROM-RAM, pokud lze přistupovat k libovolné adrese.

Jak uvádí [3], popis architektury se dělí na tři základní úrovně abstrakce:

- Strukturální popis – popisuje obvod na základě propojování logických hradel a entit pomocí vodičů (signálů).
- RTL (Register Transfer Logic) popis – obvod je popisován logickými funkcemi a přesuny dat mezi registry. Programátor má menší kontrolu nad výsledným zapojením obvodu.
- Behaviorální popis – toto je nejvyšší stupeň abstrakce architektury ve VHDL. Programátor popisuje chování obvodu v čase, ale neovlivní výslednou implementaci na hradlovém poli.

Já jsem ve většině případů zvolil behaviorální popis entit, protože se s ním lépe popisují komplikovanější struktury, např. stavové automaty.

Popis každé entity může být v samostatném zdrojovém `.vhd` souboru, ale popisy více entit lze i sloučit do jediného souboru. Zde jsem narazil na drobný „chyták“. A to takový, že přestože jsou entity v jednom souboru, ke každé zvlášť se musí deklarovat použité knihovny. Takovouto věc jsem nečekal po zkušenostech s programovacími jazyky, kde je deklarace knihoven platná pro celý soubor.

2.2.2 Datové typy

Signály mohou mít různé datové typy, které určují reprezentaci informace, kterou nesou. Základními datovými typy jsou jednobitová hodnota `std_logic` a bitové pole (vektor) `std_logic_vector`. Dále tu jsou číselné datové typy `integer` a `unsigned`.

U číselného typu `unsigned` je možno používat číselné operace (např. součet), zároveň je však možné přistupovat k jednotlivým bitům. Toto je užitečná vlastnost vzhledem k tomu, že typ `std_logic_vector` nepodporuje číselné operace a typ `integer` nemá přístupné jednotlivé bity.

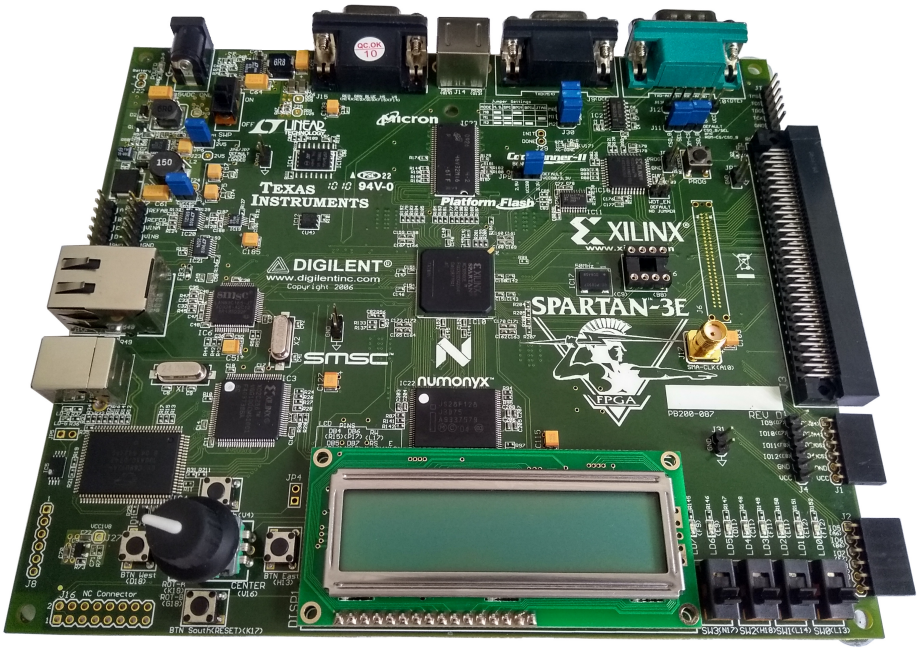
VHDL má velmi přísné typování. To znamená, že se mezi datovými typy (i příbuznými) musí explicitně převádět pomocí funkcí. Tato vlastnost mi byla často spíše překážkou. Kvůli složitosti převodu mezi znakovým typem `character` a bitovým vektorem `std_logic_vector` jsem se raději rozhodl používat všude pouze osmibitový vektor pro proměnné pracující se znaky.

2.3 Použité technické vybavení

V této části popisuji technické vybavení, které hraje podstatnou roli v této práci. Jako první představím přípravek Spartan-3E, poté se zmíním o USB kabelu s převodníkem na RS-232 a použitém LCD monitoru.

2.3.1 Vývojová deska Spartan®-3E Starter Board

Tento přípravek obsahuje programovatelné hradlové pole Xilinx XC3S500E Spartan-3E spolu s dalšími obvody, konektory a periferiemi. Vývojová deska



Obrázek 2.1: Vývojová deska Spartan-3E Starter Board

je vyfocena na obrázku 2.1. Integrovaný obvod FPGA je uprostřed desky, napravo od něj leží oscilátor o kmitočtu 50 MHz, jež je zdrojem hodin pro hradlové pole.

V této práci na desce používám znakový LCD displej (dole uprostřed), z menších součástek pak tlačítka (vlevo dole), posuvné spínače a osmici svítivých diod (vpravo dole). Dále využívám konektory typu D-sub (samice) pro rozhraní RS-232 a VGA (nahoře uprostřed).

Deska je napájena síťovým adaptérem s výstupním napětím 5 V, který se k ní připojuje válcovým konektorem (vlevo nahoře). Tento adaptér mi byl zapůjčen spolu s přípravkem vedoucím mé práce. Programování probíhá přes konektor USB typu B (vlevo).

Je více možností, odkud může být konfigurace hradlového pole načtena po připojení napájení. Pro účel testování je konfigurace nahrána jen přímo do FPGA, kde zůstane až do vypnutí napájení nebo resetu[2]. Trvalé uložení konfigurace je možné, mimo jiné, zápisem do flash paměti XCF04S. Tímto způsobem jsem zapsal konečnou verzi svého obvodového návrhu.

2.3.2 Kabel s převodníkem RS-232 na USB

Pro připojení rozhraní RS-232 desky Spartan-3E k počítači jsem potřeboval tento kabel, protože můj počítač nemá konektor D-sub pro RS-232. V dnešní době je toto rozhraní na osobním počítači spíše výjimkou. Kabel má v sobě

[2]Tlačítko reset se nachází v pravém horním rohu desky.

Obrázek 2.2: Převodník RS-232 na USB

převodník na USB (s konektorem typu A), takže lze připojit do USB portu počítače. Na obrázku 2.2 je jeho fotografie.

2.3.3 LCD Monitor

Pro zobrazení VGA výstupu jsem využil počítačový monitor značky BENQ. Desku jsem k němu připojil propojovacím VGA kabelem. Monitor podporuje rozlišení až 1920×1080 bodů.

Drobnou nepříjemností je to, že rozlišení u tohoto monitoru se automaticky škáluje na maximální plochu. To způsobí, že mnou použitá výška obrazu 480 se zvětší na 1080 bodů a šířka se zvětší ve stejném poměru. Jelikož však tyto rozměry jsou v neceločíselném poměru, objevují se v některých místech obrazu neostré hrany.

2.4 Použité programové vybavení

Pro vývoj projektu jsem na svém počítači používal 64bitový operační systém Linux Mint verze 20.1 s grafickým prostředím Cinnamon. Na počítač jsem mimo jiné nainstaloval vývojové prostředí Xilinx ISE (licence WebPack™) spolu se sériovým terminálem `picocom`. Tyto programy zde blíže popisuji.

2.4.1 Xilinx ISE Design Suite

Xilinx ISE Design Suite je vývojové prostředí pro programovatelná hradlová pole od firmy Xilinx. Základní licence WebPack™ je zdarma[3] – má určitá omezení, která ale tuto práci neovlivnila.

Toto vývojové prostředí již není nadále vyvíjeno, jeho poslední vydaná verze je Xilinx ISE 14.7. Nástupcem Xilinx ISE je Vivado® Design Suite, které již nepodporuje hradlová pole řady Spartan-3E.

[3] Vyžaduje však registraci na webových stránkách výrobce – `https://www.xilinx.com/`.

Verze 14.7 však není zcela kompatibilní s Windows 10. Oficiální verze pro Windows 10 obsahuje virtuální počítač s Linuxem, ve kterém běží Xilinx ISE. Celá takováto instalace zabere přes 40 GB, což je značná část disku.

Xilinx ISE a Linux

Xilinx ISE má verzi i pro Linux – oficiálně podporuje jen komerční distribuce [4]. Nejprve jsem zkoušel program nainstalovat na CentOS 7, což je distribuce založená na komerčním Red Hat Enterprise Linuxu, ale nepodařilo se mi vyřešit problémy s knihovnami.

Překvapivě na distribuci Linux Mint stačilo doinstalovat jen několik knihoven. Poté jsem zde Xilinx ISE úspěšně nainstaloval a používal bez větších problémů. Instalace zabírá na disku zhruba 21 GB.

Poslední věc, kterou bylo třeba vyřešit, byly USB ovladače vývojové desky. Nainstaloval jsem je s pomocí návodů na internetu. Podrobný návod hodný zmínky jsem nalezl na webu ArchWiki[4].

Součásti Xilinx ISE

Vývojové prostředí Xilinx ISE má mnoho funkcí, které jsou rozděleny do různých nástrojů. Hlavní částí prostředí je Project Navigator, ten zahrnuje vytváření projektů, správu zdrojových souborů a syntézu návrhu obvodu (syntéza vytvoří konfigurační soubor hradlového pole). K nahrání konfiguračních souborů do FPGA slouží nástroj iMPACT. Pro simulaci navržených obvodů lze použít součást ISim.

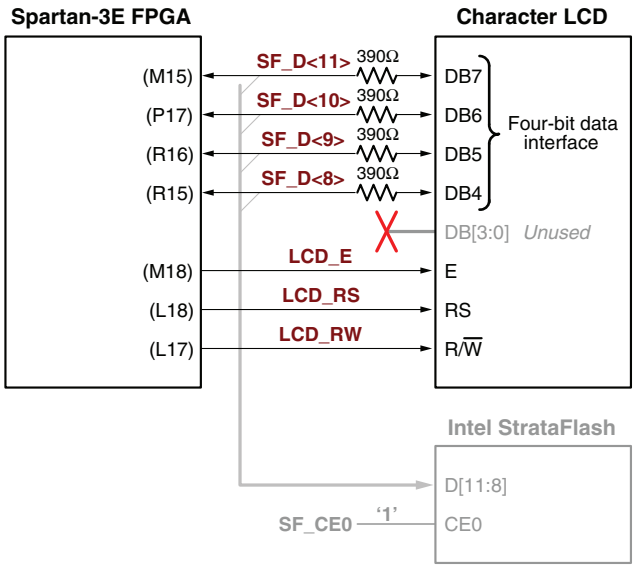
2.4.2 Sériový terminál `picocom`

Program `picocom` je sériový terminál s textovým rozhraním, který má svobodnou licenci. Přes něj lze v terminálovém emulátoru (například v mnou použitém `gnome-terminal`) obsluhovat sériový port. V mém případě je to virtuální port RS-232 přes rozhraní USB. Na Linux Mint byl nainstalován pomocí správce instalačních balíčků `apt`.

2.5 Znakový LCD zobrazovač

Znakový LCD displej na přípravku Spartan-3E umí zobrazit 16 znaků na každém ze dvou řádků. Jako řadič používá integrovaný obvod Sitronix ST7066U, ten je kompatibilní se standardním obvodem Hitachi HD44780 používaným pro řízení tohoto typu zobrazovače.

[4] `https://wiki.archlinux.org/index.php/Xilinx_ISE_WebPACK#Xilinx_Platform_Cable_USB-JTAG_Drivers`



Obrázek 2.3: Schéma připojení LCD k hradlovému poli [5]

2.5.1 Rozhraní LCD

Komunikace s LCD může využívat 4 nebo 8 datových vodičů, k nim jsou ještě přidruženy tři řídicí signály. Hradlové pole Spartan-3E má k displeji připojeny vývody pouze ke čtyřem datovým vodičům (DB4 – DB7), takže odesílání jednoho osmibitového znaku nebo příkazu se provádí po dvou čtveřicích bitů. Schéma zapojení LCD na vývojové desce ukazuje obrázek 2.3.

Řídicí signály jsou tyto:

- E (Enable, *povolit*) – pulz logické 1 přikáže řadiči displeje přečíst datovou sběrnici.
- RS (Register Select, *výběr registru*) – indikuje, zda na sběrnici je zapisován příkaz (log. 0) nebo data (log. 1).
- R/$\overline{\text{W}}$ (Read/$\overline{\text{Write}}$, *zápis*/$\overline{\textit{čtení}}$) – vybírá mezi režimem zápisu na displej (log. 0) a čtení z registrů displeje (log. 1).

Na schématu je možné vidět šedou barvou naznačenou paměť StrataFlash. Jelikož zobrazovač používám v režimu pouze pro zápis, tj. vodič R/$\overline{\text{W}}$ je napevno připojen na logickou nulu, není potřeba ji brát v úvahu (podle tabulky 5-2 na straně 44 v [5]).

2.5.2 Komunikace

Čtyřbitová komunikace probíhá podle příručky [5] takto:

1. Nastaví se signál Register Select podle toho, zda je posílán příkaz, nebo jsou posílána data.

2. Na datovou sběrnici se nastaví horní 4 bity příkazu/dat, jedničkový pulz na Enable je odešle.
3. Vyčká se po dobu alespoň 1 µs.
4. Na sběrnici se nastaví dolní 4 bity příkazu/dat, pulzem na Enable se odešlou.
5. Počká se alespoň 40 µs (1,64 ms v případě některých příkazů) na zpracování displejem.

Zobrazovač se před použitím inicializuje danou sekvencí příkazů, která nastavuje čtyřbitovou komunikaci. Tuto sekvenci uvádím v praktické části 3.4.3.

2.5.3 Sada znaků LCD

Znaky jsou v dolní polovině rozsahu kódů zobrazovány podle sedmibitové tabulky ASCII (kromě prvních 32 řídicích znaků a znaku vlnovky). Horní polovina tabulky přidává některé další znaky (například matematické nebo japonské). Každý znak na displeji má šířku 5 a výšku 7 obrazových bodů. V tabulce lze vytvořit až 8 vlastních symbolů, ale tuto funkci jsem nevyužil. Tabulka C.3 v příloze ukazuje přehled znaků, které lze na LCD zobrazit včetně jejich příslušných kódů.

2.6 Rozhraní UART (RS-232)

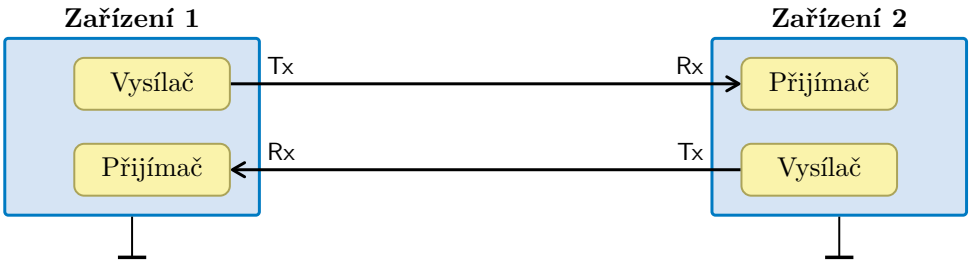
UART (Universal Asynchronous Receiver/Transmitter) – česky *Univerzální asynchronní přijímač/vysílač* – je asynchronní sériové rozhraní používané pro komunikaci mezi dvěma elektronickými zařízeními. Obě zařízení mohou naráz vysílat i přijímat. Standard RS-232 definuje fyzickou vrstvu komunikace pomocí UART.

Asynchronní znamená, že mezi zařízeními není přenášen hodinový synchronizační signál. Každé musí mít vlastní zdroj hodinového taktu, který by nejlépe měl mít kmitočet shodný s kmitočtem taktu protějšího zařízení. Aby se zajistila součinnost přijímače i vysílače, je při přenosu dat prováděna synchronizace.

Sériové rozhraní se vyznačuje tím, že jednotlivé bity informačního obsahu se posílají za sebou v čase. Komunikující zařízení mají z tohoto důvodu na vstupu a výstupu posuvné registry, do kterých se nasunují nebo ze kterých se vysunují data po bitech. Výhodou oproti paralelní komunikaci je to, že pro (v případě UART jednosměrný) přenos stačí jediný datový vodič.

2.6.1 Fyzická komunikační vrstva

Rozhraní UART může být implementováno na fyzické úrovni různě. Na počítačích dříve používaný sériový port RS-232 používá (podle [6]) -12 až -5 V pro logickou jedničku a 5 až 12 V pro logickou nulu. RS-232 obsahuje navíc kromě datových signálů další pomocné signály.



Obrázek 2.4: Princip komunikace přes sériové rozhraní UART

V aplikacích s číslicovými obvody se používá UART s logickými úrovněmi TTL/CMOS logiky. Logická nula tedy odpovídá napětí 0 V a logická jednička napájecímu napětí, což bývá typicky 5 V nebo 3,3 V.

Na vývojové desce Spartan-3E jsou logické úrovně TTL/CMOS na straně FPGA převedeny na úrovně pro RS-232 na straně konektoru obvodem MAX3232 [5].

2.6.2 Princip přenosu dat

Komunikace probíhá přes 2 datové vodiče – jeden přijímací (Rx – Receiver) a jeden vysílací (Tx – Transmitter). Jak je znázorněno na obrázku 2.4, vysílací vodič Tx je vždy připojen na vstup přijímače Rx protějšího zařízení. Pro správné rozlišení logických úrovní musí být obě zařízení na stejném zemním potenciálu.

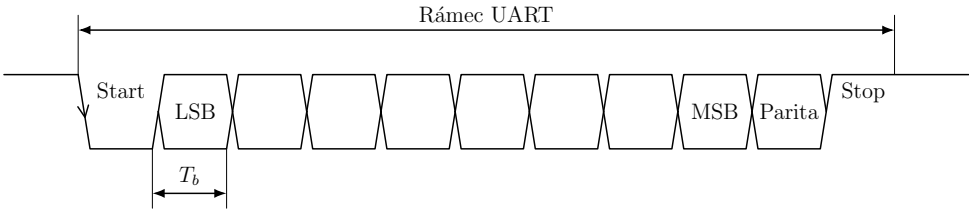
Přenos dat je rozdělen na datové jednotky nazvané „rámec“. Konfigurací přenášeného rámce je více, přijímač a vysílač musí mít shodnou konfiguraci rámce, aby došlo ke správnému přenosu. V klidovém stavu, kdy nedochází ke komunikaci, je na výstupu vysílače stav log. 1.

2.6.3 Struktura rámce

Časový průběh rámce znázorňuje obrázek 2.5. Rámec se skládá z těchto částí:

- Start bit – logická nula, která svou sestupnou hranou pomáhá synchronizovat takt přijímače pro vzorkování bitů.
- Datové bity – zdroj [7] uvádí, že datových bitů může být 5 až 8. Nejčastěji se používá 8 bitů čili 1 bajt. Pořadí posílání bitů je od nejméně významného (LSB – Least Significant Bit) po nejvýznamnější (MSB – Most Significant Bit).
- Paritní bit – může být nastaven na sudou či lichou paritu, nebo se v rámci vůbec nemusí vyskytovat. Jak je popsáno v [8], paritní bit má takovou hodnotu, aby počet jedničkových bitů datové části spolu s paritním bitem byl sudý, respektive lichý[5]. Používá se jako jednoduchá forma kontroly přenosu.

[5]Například pokud v datech je lichý počet jedniček a je pro přenos nastavena sudá parita, bude mít paritní bit hodnotu 1.



Obrázek 2.5: Časový průběh rámce UART

- Stop bit – logická jednička na konci rámce, která značí konec vysílání. Po stop bitu může ihned následovat start bit dalšího rámce.

2.6.4 Časové parametry

Každý přenášený bit má pevně určenou dobu trvání, jejíž hodnota závisí na zvolené konfiguraci. Tato hodnota (doba jednoho bitu T_b) je dána vztahem

$$T_\mathrm{b} = \frac{1}{R_\mathrm{b}} \quad [\mathrm{s; b/s}], \tag{2.1}$$

kde R_b je přenosová rychlost. Standardní přenosové rychlosti, které zmiňuje [7], jsou 9600, 19200, 38400, 57600 a 115200 b/s. Někdy se místo jednotky b/s (bit za sekundu) používá jednotka baud. V této práci se držím používání jednotky b/s.

2.7 Rozhraní VGA

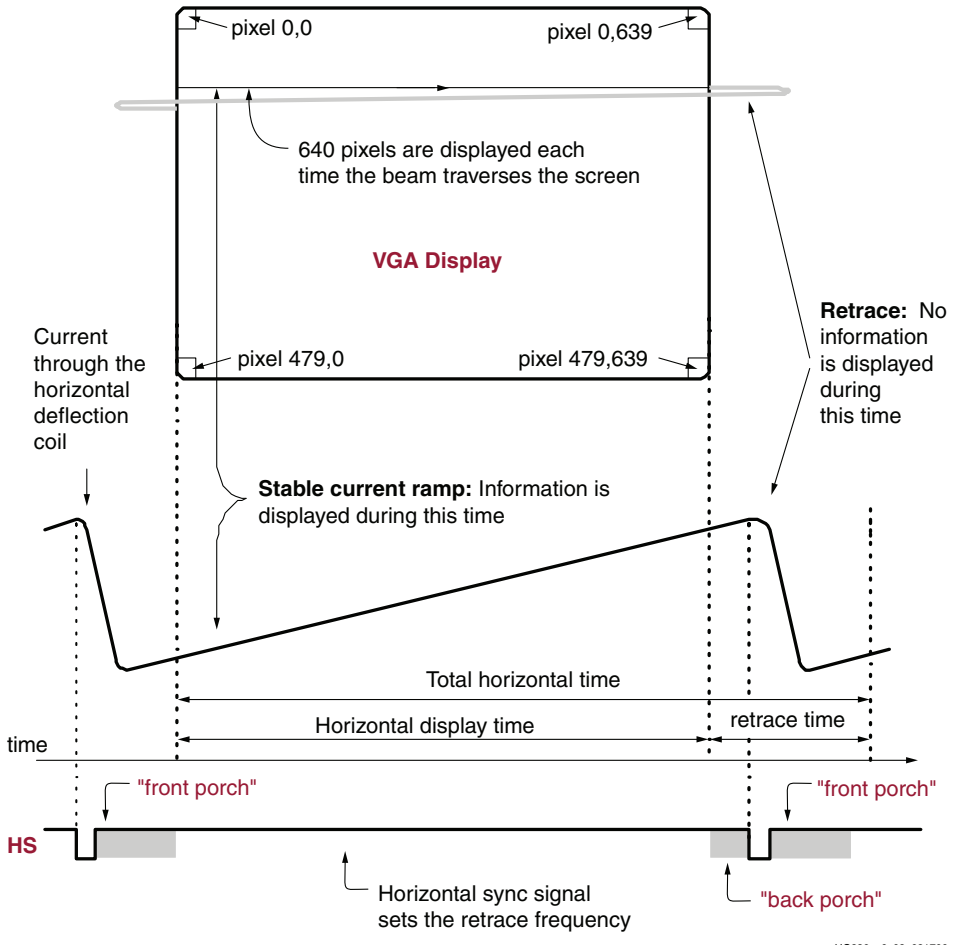
VGA (Video Graphics Array) je analogové rozhraní pro přenos obrazu. Základem je pět signálových vodičů – dva digitální pro synchronizaci a tři analogové, nesoucí barevnou informaci. Informace o VGA byly čerpány z uživatelských příruček [5] a [9].

Každý obrazový bod má svou barvu složenou ze 3 základních barev: červené, zelené a modré (používá se anglická zkratka RGB). Z těchto barevných složek je u VGA každá přenášena po vlastním vodiči. Analogové barevné signály nabývají úrovně napětí v rozmezí 0 až 0,7 V, přičemž velikost napětí je přímo úměrná intenzitě barvy.

2.7.1 Princip

Princip zobrazování VGA pochází z dob elektronkových obrazovek, kdy vychylováním elektronového paprsku ve vodorovném a svislém směru vznikal na luminoforu obraz.

Pomyslný paprsek přebíhá zleva doprava po jednotlivých řádcích postupně odshora dolů. K odvození polohy paprsku jsou použity dva synchronizační signály – horizontální a vertikální synchronizace. Obrázek 2.6 znázorňuje časový průběh proudu vychylující cívkou elektronkové obrazovky a jeho vztah k časování horizontální synchronizace.



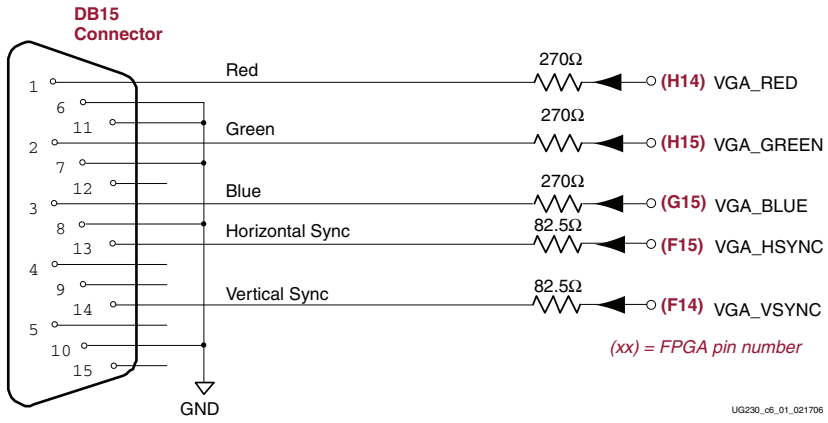
Obrázek 2.6: Princip časování horizontálního synchronizačního signálu [5]

2.7.2 Časový průběh

Paprsek je vychylován po obrazovce vodorovně doprava a tím kreslí jednotlivé body v řádku. Po přeběhnutí zobrazovacího pole paprsek dobíhá po dobu nazvanou „back porch“. Pulz logické nuly na horizontální synchronizaci spouští návrat paprsku doleva. Návrat trvá mnohem kratší dobu než přeběh doprava. V této době je elektronový paprsek zatemněn.

Paprsek se zastaví na levé straně mimo zobrazovací pole, je o řádek posunut dolů a začíná přebíhat doprava k vykreslení dalšího řádku. Na návrat paprsku do zobrazovací části obrazovky je vyhrazena doba zvaná „front porch“. Cyklus se takto opakuje pro všechny řádky.

Obdobný princip je použit i pro vertikální vychylovací cívku a vertikální synchronizační signál. V takovém případě je pohyb paprsku mnohem pomalejší, ale stále jsou zde vymezeny doby zatemnění paprsku (při kreslení obrazu o výšce 480 bodů paprsek projde celkem 521 řádků).



Obrázek 2.7: Schéma připojení konektoru VGA k hradlovému poli [5]

2.7.3 VGA na desce Spartan-3E

Vývojová deska Spartan-3E má barvonosné vodiče připojeny na digitální výstupy, takže lze zobrazit jen omezený počet barev. Obrázek 2.7 ukazuje přesné zapojení.

Jak uvádí [5], rezistory velikosti 270 Ω tvoří spolu s ukončovací impedancí kabelu 75 Ω dělič napětí. Pokud je na barevném výstupu 3,3 V (logická 1), na vstupním konektoru obrazovky je napětí $3,3 \cdot \frac{75}{270+75} = 0,717$ V. To přibližně odpovídá plné úrovni barvy. Při logické nule na výstupu (0 V) bude i barva na nulové úrovni.

Každý ze tří barevných signálů lze nastavit na úroveň plné barvy nebo nulovou úroveň. Z toho vychází jako maximální možný počet $2^3 = 8$ barev. Není to mnoho, ale na dvoubarevný textový výstup je to dostačující.

2.8 Znaková kódování

V číslicové technice se pro reprezentaci znaků používá znakové kódování. To znamená, že každému znaku je přiřazeno unikátní číslo – kód.

Standardním kódováním písmen anglické abecedy a některých dalších znaků se stalo kódování ASCII (American Standard Code for Information Interchange). Toto kódování přiřazuje znakům sedmibitový kód, počet jedinečných znaků ASCII je tudíž $2^7 = 128$. V tabulce C.1 jsou uvedeny tisknutelné znaky, v tabulce C.2 pak řídicí znaky ASCII s příslušnými kódy.

Pro jazyky se znaky, které nejsou v anglické abecedě, byla vytvořena celá řada rozšíření ASCII tabulky, často s osmibitovým kódováním, které může obsahovat až 256 znaků.

České znaky s diakritikou (á, ý, č, ř, ň, ...) jsou plně obsaženy například v 8bitovém kódování ISO 8859-2, někdy značeném jako Latin-2. Toto kódování, zmiňované v [10], zahrnuje znaky jak pro češtinu, tak pro některé další evropské jazyky, jako je slovenština, polština, němčina, maďarština, ...

Kapitola 3
Praktická část

V této části budu popisovat svou realizaci logického návrhu funkčních celků podle zadání v jazyce VHDL. Popis je rozdělen na několik částí, které popisují jednotlivé funkční celky. Tyto části jsou následující:

- popis hlavního obvodu,
- popis přijímače UART,
- posílání řetězců přes UART stiskem tlačítek,
- řízení LCD displeje,
- příjem textu přes UART, jeho zobrazení na displeji a provádění příkazů,
- zobrazování přijatých znaků na monitoru přes VGA.

Největší pozornost bude věnována popisu VGA výstupu, jelikož je nejkomplexnější a jeho vytvoření zabralo nejvíce času.

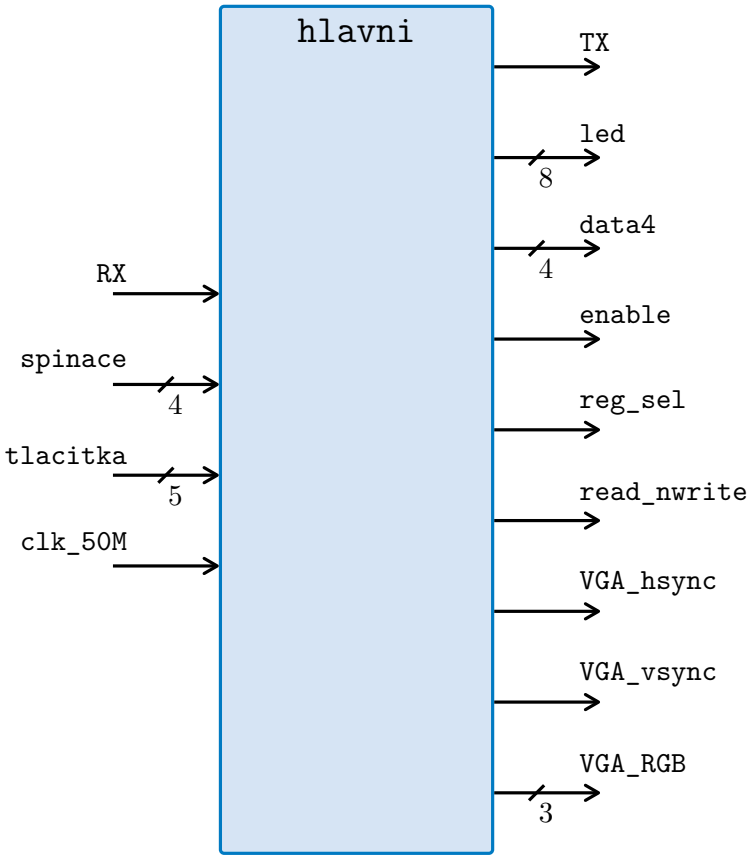
Při popisování často zaměňuji názvy entit a jejich instancí (podle toho, který název mi přijde vhodnější), ale vzhledem k tomu, že skoro všechny entity mají jedinou instanci, nemělo by dojít k nedorozumění.

3.1 Hlavní obvod

Všechny funkční celky jsou obsaženy v jedné nejvyšší entitě nazvané `hlavni`. Ve VHDL je totiž potřebné mít jedinou entitu sdružující všechny obvody dohromady. Blokové znázornění obvodu spolu se vstupními a výstupními porty je na obrázku 3.1. Číslo v místě přeškrtnutí šipky signálu ukazuje bitovou šířku tohoto signálu, pokud není jednobitový.

Hlavní entita sdružuje všechny použité fyzické vstupy a výstupy hradlového pole. Přiřazení vstupních a výstupních portů entity k jednotlivým vývodům FPGA určuje textový soubor `hlavni.ucf`. V tomto souboru se také definují např. napěťové logické úrovně vývodů. Nastavil jsem, aby všechny vývody pracovaly s úrovněmi CMOS 3,3 V.

Tabulka 3.1 ukazuje přiřazení vstupů a výstupů. Skupiny vývodů (tj. bitové vektory) mají v hranatých závorkách označen rozsah indexů bitů. V odpovídajícím pořadí je potom vypsán seznam jmenných pozic vývodů FPGA.



Obrázek 3.1: Hlavní VHDL entita

3.1.1 Hierarchie

Na obrázku 3.2 je vyobrazena hierarchie projektu v Xilinx ISE. Stojí za zmínku, že první název je jméno instance entity (jako komponenty) v nadřazeném obvodu, zatímco druhý název za pomlčkou je název použitý při vlastní definici této entity. V závorce je uveden název zdrojového souboru VHDL, kde je komponenta definovaná.

Obvod `LCD_prijimac` slouží k tisknutí znaků přijatých přijímačem UART na displej. Samostatný celek `UART_vysilac` (jméno může být zavádějící) posílá přes UART název stisknutého tlačítka. Zobrazování znaků přes VGA zajišťují prvky `zRAM` (paměť znaků), `zobrazovac_znaku` a `zapisovac_znaku`. Tyto vyjmenované komponenty jsou všechny přímými potomky hlavní entity.

Kromě souborů obsahujících VHDL entity jsem vytvořil společnou knihovnu v souboru `knihovna.vhd`. V této knihovně jsou definovány některé mnou používané konstanty, datové typy a funkce.

3.2 UART přijímač

Přijímač rozhraní UART je obsažen v entitě `UART_prijimac`. Je konfigurován pro příjem rámce s osmi datovými bity a bez paritního bitu přenosovou rych-

Tabulka 3.1: Seznam přiřazených vývodů FPGA

Název	Směr	Pozice vývodu
`clk_50M`	vstup	C9
`tlacitka[4...0]`	vstupy	V16, D18, K17, H13, V4
`spinace[3...0]`	vstupy	N17, H18, L14, L13
`RX`	vstup	R7
`TX`	výstup	M14
`led[7...0]`	výstupy	F9, E9, D11, C11, F11, E11, E12, F12
`enable`	výstup	M18
`reg_sel`	výstup	L18
`read_nwrite`	výstup	L17
`data4[3...0]`	výstupy	M15, P17, R16, R15
`VGA_vsync`	výstup	F14
`VGA_hsync`	výstup	F15
`VGA_RGB[2...0]`	výstupy	H14, H15, G15

lostí 115200 b/s. Příslušnou konfiguraci jsem zvolil proto, že tato přenosová rychlost je standardní a kontrolovat správnost přenosu paritním bitem nebylo nezbytné.

Na vstup obvodu je přiveden signál pro příjem UART – `RX` a hodinový takt 50 MHz – `clk_50M`. Výstupem je přijatý osmibitový znak `data` a signál indikující dokončení příjmu `DTR` (DaTa Ready)[1]. Obvod je znázorněn jako blok na obrázku 3.3.

3.2.1 Dělení kmitočtu

Dělička kmitočtu, která je zdrojem bitového taktu, je realizována pomocí čítače. Každou vzestupnou hranu hodinového signálu oscilátoru se jeho hodnota zvýší o jedničku. Jestliže čítač dočítá do přednastavené hodnoty úměrné podílu vstupního a výstupního kmitočtu, překlopí se výstupní signál děličky.

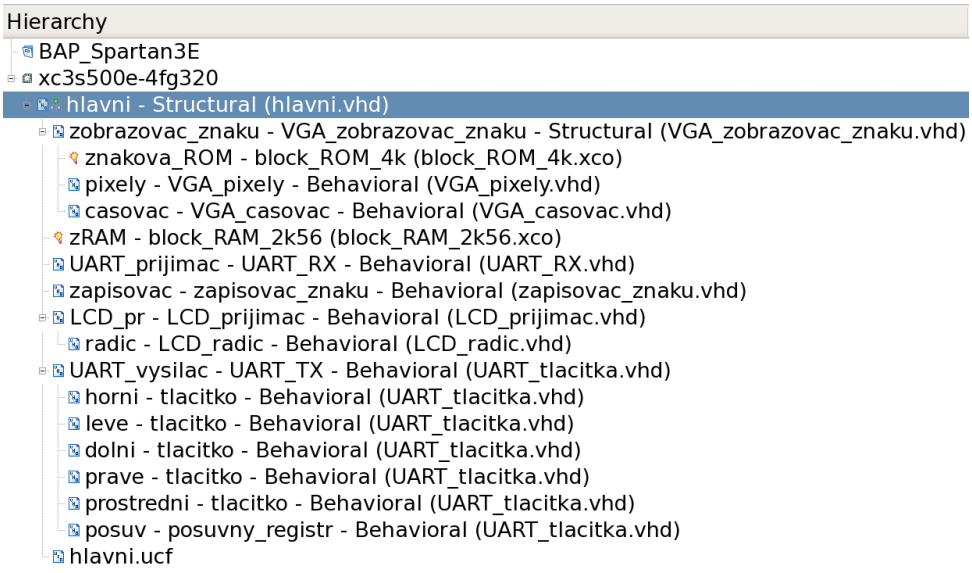
Chceme-li vydělit kmitočet celým číslem N, výstup děličky se musí překlopit každých $\frac{N}{2}$ period hodin. To znamená, že čítač čítá po každou půlperiodu vyděleného taktu od 0 do $\frac{N}{2} - 1$. Po dosažení horní hranice se výstup překlopí a čítač vynuluje.

Pro můj případ vytvoření taktu 115,2 kHz je kmitočet oscilátoru 50 MHz dělen v poměru

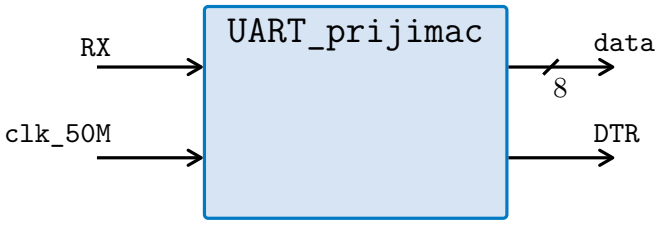
$$N = \frac{50 \cdot 10^6}{115200} = 434,02778 \doteq 434. \tag{3.1}$$

Ve VHDL kódu děličky je hodnota, do které se má čítat automaticky vypočtena ze zadaných kmitočtů. To dovoluje změnit bitovou rychlost jedním parametrem. Pouze se musí ověřit, že rozsah čítače je dostatečně velký. Děličky kmitočtu, které jsou v ostatních obvodech, pracují na stejném principu. Nebudu je dále při popisu podrobně rozebírat.

[1] Tento název byl inspirován názvem řídicího signálu RS-232. Jeho funkce však není ekvivalentní.



Obrázek 3.2: Hierarchie hlavní VHDL entity v Xilinx ISE



Obrázek 3.3: Blokové znázornění UART přijímače

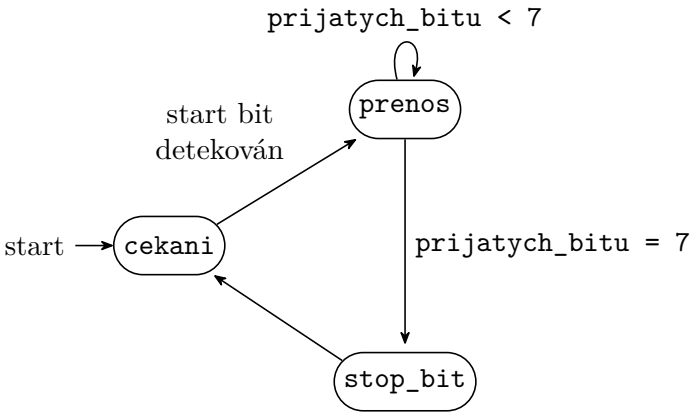
3.2.2 Potíže s detekcí sestupné hrany

Klíčovou úlohou přijímače je, aby detekoval každou sestupnou hranu signálu `RX`. Pokud by přijímač nerozpoznal hranu start bitu, bral by další přechod z logické jedničky na nulu v datové části rámce jako start bit. To by způsobilo celkovou desynchronizaci přijímače a přijaté znaky by nebyly platné až do delší časové mezery mezi rámci nebo do konce přenosu.

Takovýto problém jsem brzy po vytvoření přijímače zaznamenal. Původně se to jevilo jako náhodná chyba v přenosu, ale následně jsem zjistil, že se chyba systematicky objevuje při příjmu několika znaků bezprostředně po sobě. Z toho jsem usoudil, že problém je v synchronizaci.

Všechny simulace navzdory tomu probíhaly bez problémů, což mi ztížilo situaci. Potíže byly jen s fyzickým obvodem. Zde jsem poprvé a naposled použil USB logický analyzátor v rámci této práce, proto jsem jej ani nezařadil do seznamu použitých zařízení. Z přijímače jsem na fyzické výstupy vyvedl signály indikující detekci hrany. Rozborem změřeného průběhu signálů jsem zjistil, že sestupná hrana není vždy obvodem detekována.

Řešení tohoto problému jsem našel díky otázce [11] na webu Stack Exchange. Vstupní signály se musí synchronizovat na hodinový signál. A to tak, že se navzorkují do řady (dvou) klopných obvodů spojených za sebou a taktovaných



Obrázek 3.4: Stavový diagram přijímače UART

kmitočtem oscilátoru. Po ošetření vstupu takovýmto způsobem jsem přestal mít potíže s chybnou detekcí start bitu.

3.2.3 Činnost přijímače

Činnost přijímače je řízena stavovým automatem se třemi stavy. Automat je taktován kmitočtem 115200 Hz, který odpovídá bitové rychlosti. Tento kmitočet určuje, jak často se může měnit stav automatu. Obrázek 3.4 ukazuje jeho stavový diagram. V počátečním stavu, `cekani`, obvod vyčkává na sestupnou hranu start bitu.

Detekce hrany

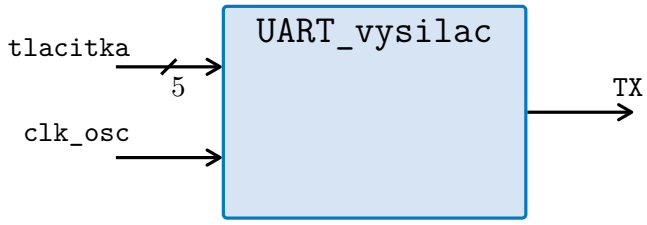
Detekce hrany je realizována tříbitovým posuvným registrem připojeným na vstup `RX`. S taktem oscilátoru 50 MHz je do něj nasouván (směrem od prvního ke třetímu bitu) vstupní signál. Obvod vyvodí, že došlo k sestupné hraně, jestliže druhý bit (neboli stabilní současná hodnota) má hodnotu logické 0 a třetí bit (předchozí hodnota) má hodnotu log. 1.

Pokud je detekována sestupná hrana signálu `RX` (start bit), přechází automat do stavu `prenos`. Tato hrana způsobí vynulování čítače děličky kmitočtu, ze které se odvozuje takt 115200 Hz pro čtení příchozích bitů (tj. vynuluje se fáze hodin). Tím je zaručeno, že čtení proběhne přibližně v polovině doby trvání jednoho přijímaného bitu, kdy vstupní hodnota je ustálená.

Čtení bitů

Ve stavu `prenos` se každou vzestupnou hranou čtecích hodin nasune do vstupního registru datový bit ze vstupu `RX`. Počet přijatých bitů čítá registr `prijatych_bitu`, který je nulován začátkem každého příchozího rámce.

Po osmi čtecích taktech jsou přijaty všechny datové bity, které se zapíší na výstup `data`. Automat přechází s vzestupnou hranou, která čte osmý datový bit, do třetího stavu, nazvaného `stop_bit`. Signál dokončení příjmu



Obrázek 3.5: Blokové znázornění UART vysílače

`DTR` přejde do stavu logické 1. Tento signál se nuluje přijetím start bitu dalšího rámce. Stav `stop_bit` trvá jen jeden takt hodin, poté se obvod vrací zpět do stavu `cekani`.

3.3 Posílání řetězců přes UART stiskem tlačítek

V této části popisuji obvod `UART_vysilac`, který posílá přes UART pevně zvolený řetězec při stisku příslušného tlačítka. Obsažený vysílač posílá rámce bitovou rychlostí 115200 b/s bez paritního bitu. Tento obvod pracuje nezávisle na ostatních komponentách, např. i na přijímači UART.

Blokově je obvod znázorněn na obrázku 3.5. Vstupy obvodu jsou takt oscilátoru – `clk_osc` a stav pěti tlačítek na desce – `tlacitka` (pětibitová hodnota). Jediným výstupem je výstup vysílacího posuvného registru `TX`.

Obvod `UART_vysilac` je tvořen děličkou kmitočtu 50 MHz na 115200 Hz, dále posuvným registrem, který vysouvá datový rámec na výstup vysílače, komponentami, které ošetřují zákmity tlačítek a nakonec řídicí logikou.

3.3.1 Výstupní posuvný registr

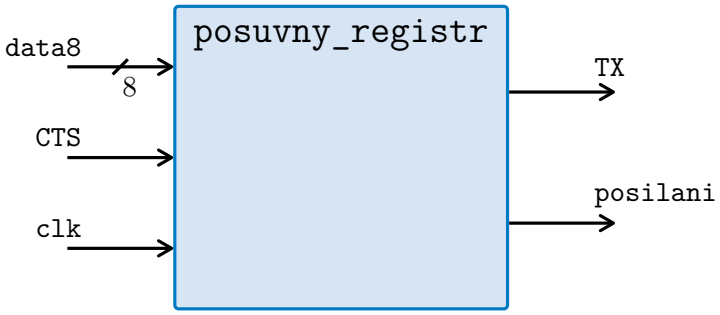
O posílání rámce UART se stará entita `posuvny_registr`, která obsahuje vysílací posuvný registr s okolní logikou. Na obrázku 3.6 je ukázáno rozhraní této komponenty. Vstup `clk` přivádí bitový kmitočet 115,2 kHz. Osm datových bitů, které se mají poslat, se přivádí na vstup `data8`. Poslední vstup, `CTS` (Clear to Send)[2], slouží k zahájení vysílání rámce s daty. Výstup `TX` je skrze nadřazené obvody přiveden až na konektor rozhraní RS-232. Pomocný výstup `posilani` signalizuje probíhající posílání rámce úrovní log. 1.

Průběh vysílání

Obvod `posuvny_registr` pracuje následovně: Periodicky je čten stav signálu `CTS`. Pokud se tento stav změní z log. 1 na log. 0 mezi dvěma takty, je toto vyhodnoceno jako spouštěcí sestupná hrana. Jestliže je detekována sestupná hrana a obvod není v procesu odesílání, zahájí se odesílání rámce. Výstup `posilani` přejde do log. 1.

Odesílaný rámec se sestaví tak, že přečtená 8bitová data jsou doplněna start (log. 0) a stop (log. 1) bity. Takto získaná 10bitová hodnota je v každém

[2]Signál s tímto názvem také existuje u rozhraní RS-232, jímž jsem se inspiroval.



Obrázek 3.6: Blokové znázornění vysílacího posuvného registru

dalším taktu vysunována bit po bitu (od nejnižšího bitu po nejvyšší) na výstup `TX`. Po vysunutí všech deseti bitů přenos končí, `posilani` se vrací na úroveň log. 0. Výstup `TX` se ustálí na klidové úrovni log. 1. Obvod čeká na povel k přenosu dalšího rámce.

3.3.2 Čtení tlačítek

Pět tlačítek na vývojové desce je rozmístěno do kříže, přičemž to prostřední je součástí rotačního enkodéru. Stisknuté tlačítko přivede logickou jedničku na vstup. Když je však rozepnuté, logická úroveň není určena.

V souboru `.ucf`, který určuje přiřazení vývodů hradlového pole, jsem navíc nastavil, aby vstupy tlačítek byly přes velký vnitřní rezistor připojeny k zemi (tzv. pulldown rezistor). Takovéto ošetření je doporučeno v manuálu vývojové desky [5] na stranách 16–17. To zajistí, že v případě rozepnutého tlačítka bude logická úroveň definována právě tímto rezistorem.

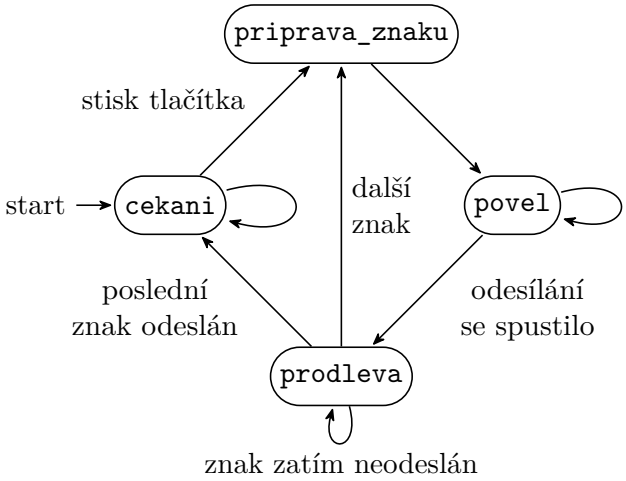
Čtení tlačítka zajišťuje VHDL komponenta `tlacitko`. Obsahuje jednoduchý stavový automat, který ošetřuje zákmity kontaktu tlačítka. Vstup této komponenty je propojen s tlačítkem. Obvod je taktován kmitočtem 50 kHz, který vzniká vydělením kmitočtu oscilátoru v nadřazené entitě `UART_vysilac`. V té je vytvořeno pět instancí komponenty `tlacitko`, pro každé fyzické tlačítko jedna.

Zaznamenáním vzestupné hrany (stisku tlačítka) vygeneruje obvod krátký pulz o délce jedné periody 50 kHz. Ošetření zákmitů probíhá tak, že po každém stisku či uvolnění tlačítka dojde na dobu 30 ms k prodlevě, při které jsou všechny změny na vstupu ignorovány.

3.3.3 Řídicí logika

K řízení vysílání řetězců byl v obvodu `UART_vysilac` použit stavový automat, znázorněný na obrázku 3.7. Proces automatu je taktován frekvencí bitového toku 115,2 kHz. Dále bude popisována funkce tohoto automatu.

Text, který se má poslat, je obsažen v řetězci `k_odeslani`. Tento řetězec má pevnou délku 15 znaků. Pokud je obsažený text kratší, je zbytek řetězce doplněn znaky s ASCII kódem `00h` (NUL). Zvolil jsem, že text bude popisovat polohu tlačítka na desce (česky, bez háčků a čárek). Pravému tlačítku přísluší



Obrázek 3.7: Stavový diagram řízení vysílače UART

text „prave“, dolnímu „dolni“ a tak dále. Na konci textu je přidán řídicí znak zalomení řádku LF.

Automat se na počátku nachází ve stavu `cekani` V tomto stavu jsou vyhodnocovány výstupy tlačítkových komponent. Pulz stisku z komponenty `tlacitko` je dostatečně dlouhý na to, aby jej stavový automat, taktovaný více než dvojnásobným kmitočtem, zaznamenal.

Při zaznamenání stisku tlačítka se do řetězce `k_odeslani` vloží pevný text určený pro každé z pěti tlačítek. Stavový automat přejde do stavu `priprava_znaku`.

Ve stavu `priprava_znaku` se na datový vstup výstupního registru zapíše znak, který je určen k odeslání. Na pozici právě posílaného znaku v řetězci `k_odeslani` ukazuje číselný signál `index_znaku`. Pro první znak z řetězce má `index_znaku` hodnotu 1.

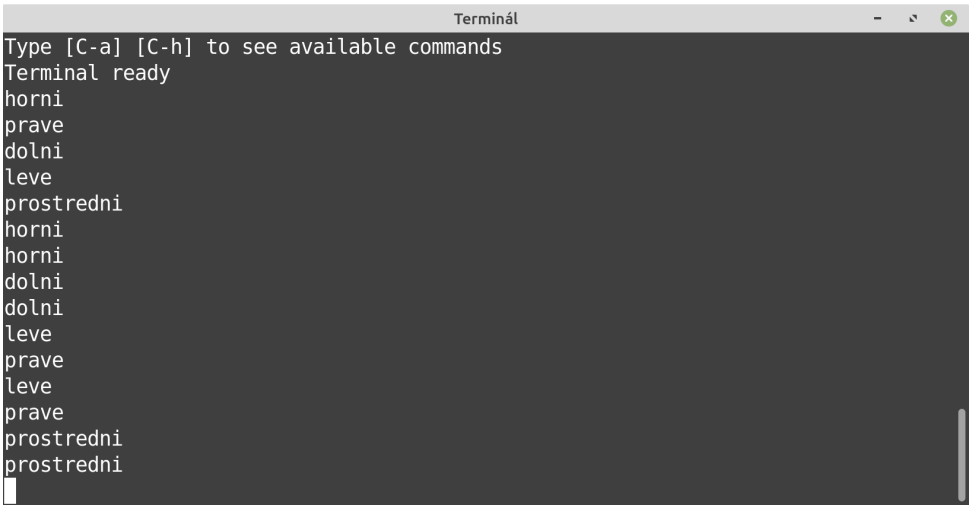
Po tomto kroku přejde obvod do stavu `povel`. Ten slouží výhradně k vytvoření sestupné hrany signálu `CTS`. Do dalšího stavu, `prodleva`, přejde automat, jakmile posuvný registr zahájí odesílání (hodnota signálu `posilani` přejde do log. 1). Stav `prodleva` je aktivní, dokud není dokončeno vysílání rámce se znakem.

Další změna stavu je určena pozicí `index_znaku`. Pokud je následující znak NUL nebo `index_znaku` se rovná maximální délce řetězce, znamená to konec přenosu textu. Ukazatel `index_znaku` se nastaví zpět na 1 a dalším stavem je stav `cekani`.

V opačném případě se `index_znaku` zvýší o jedničku, ukazuje tedy na následující znak. Činnost obvodu pokračuje opět ve stavu `priprava_znaku`. Tento proces se opakuje až do splnění podmínky pro ukončení přenosu.

3.3.4 Výsledek

Výstup RS-232 jsem připojil kabelem s převodníkem k počítači. Na počítači jsem otevřel příslušný port v sériovém terminálu `picocom`. Stiskáním tlačítek



Obrázek 3.8: Výpis ze sériového terminálu `picocom`

se vypisoval jejich název do terminálu. Obrázek 3.8 ukazuje přijatý text v tomto terminálu.

3.4 Řízení LCD displeje

O řízení znakového LCD zobrazovače se stará komponenta `LCD_radic`. Po spuštění inicializuje displej a následně na něj tiskne text. A to tak, že řetězec přivedený na její vstup cyklicky posílá přes čtyřbitové datové rozhraní do paměti displeje.

3.4.1 Vývoj návrhu

Příkazy displeje a způsob komunikace s ním jsem čerpal z uživatelské příručky pro Spartan-3E [5] a datového listu [12]. Moje první pokusy o řízení displeje nebyly úspěšné. Byl to první obvod pro tuto práci, který jsem navrhoval. Neměl jsem zatím zkušenosti s tím, jak realizovat nějakou sekvenci úkonů ve VHDL. Inicializace se totiž skládá z několika pevně daných kroků s různou dobou čekání. Navíc data se musí posílat po dvou částech. Po zvážení několika různých možností jsem nakonec navrhl dvojitý stavový automat, který již zvládl správně řídit inicializaci displeje.

Následně jsem navrhoval obvod, který zobrazí na LCD pevně daný řetězec. Při testování jsem odhalil některé chyby, které se v návrhu vyskytly a opravil je. K hledání příčin chyb jsem využíval i simulátor, jež je součástí prostředí Xilinx ISE.

Jako další jsem návrh pozměnil tak, aby se na zobrazovač periodicky vypisoval text, jež byl měněn podle stavu spínačů. Textový řetězec byl uložen v signálu datového typu `string` o délce 32 znaků. Tím jsem obsáhl veškerou zobrazovací kapacitu displeje, tj. 16 znaků × 2 řádky. Ve VHDL jsem ošetřil, aby byl kurzor zobrazovače posouván i mezi oběma řádky.

Dalším cílem bylo zobrazit text přijatý ze sériového rozhraní RS-232. Zde se ukázalo přísné typování jazyka VHDL jako problém. Přijaté znaky byly typu osmibitového bitového pole `std_logic_vector`. Řetězec typu `string` je pole osmibitového typu `character`. Převod mezi typy `std_logic_vector` a `character` nebylo snadné vyřešit. Z tohoto důvodu jsem změnil typ řetězce na vlastní, definovaný jako pole znaků typu `std_logic_vector`. Protože je tento typ používán i ve vstupním portu komponenty `LCD_radic`, bylo nutné jej definovat ve vlastní knihovně.

3.4.2 Vstupy a výstupy

Jediným vstupem komponenty `LCD_radic` kromě taktu oscilátoru je řetězec `text`, jehož typ byl definován jako pole 8bitových znaků `std_logic_vector` s délkou 32. Takováto datová struktura jako vstupní port mi nepřipadá jako ideální řešení, avšak nic lepšího mne nenapadlo.

Čtyřbitový datový výstup `data4` je přiveden na datovou sběrnici LCD. Dále jsou tu řídicí výstupy `enable`, `reg_sel` a `read_nwrite`, které jsou přivedeny na vstupy displeje Enable, Register Select a Read/$\overline{\text{Write}}$ popsané v části 2.5. Výstup Read/$\overline{\text{Write}}$ je v komponentě trvale připojen na úroveň log. 0.

3.4.3 Stavový automat

Řídicí obvod displeje ST7066U, se kterým probíhá komunikace, je vzhledem k taktu FPGA 50 MHz pomalý. Proto byla použita dělička kmitočtu na 100 kHz. Tímto kmitočtem je stavový automat taktován.

Automat je dvojnásobný čili má dvě sady stavů. První (budu jej značit nadřazený) stav udává, jaký je význam čtveřice bitů na datové sběrnici. Stavový diagram této části automatu je na obrázku 3.9. Ve druhém, podřazeném stavu, je stav odesílání oné čtveřice bitů. Na obrázku 3.10 je příslušný diagram.

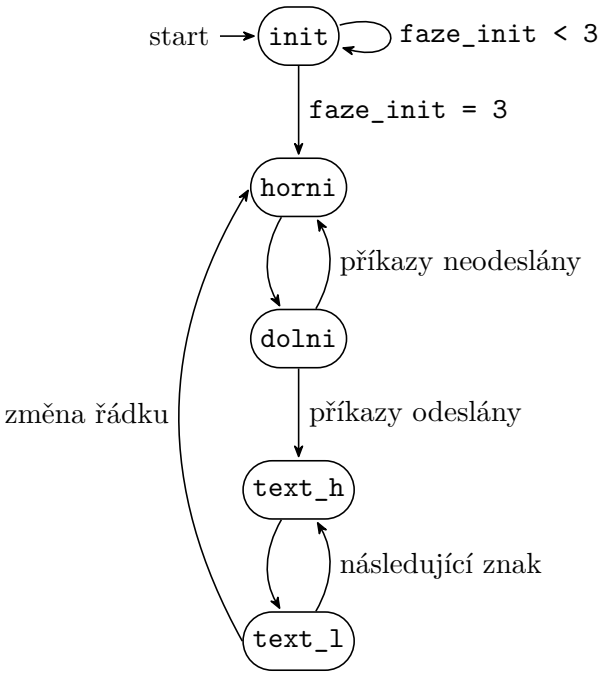
Popis nadřazené části automatu

Jeden přechod stavu v této části nastane pro každý dokončený cyklus stavů podřazené části. V procesu automatu je proměnná `faze_init`, která čítá kroky inicializace komunikace s displejem. Ta je provedena ve stavu `init`.

Inicializace probíhá takto:

1. Datové vodiče `data4` se nastaví na hodnotu `3h` (dvojkově 0011), následuje zapisující pulz (log. 1) na `enable` a čekání 4,1 ms.
2. Hodnota datových vodičů `data4` je stále `3h`, vyšle se zapisující pulz, poté se čeká 100 µs.
3. Opět stejná hodnota dat, následuje vyslání pulzu a čekání 40 µs.
4. Na `data4` se nastaví hodnota `2h`, vyšle se zapisující pulz a počká se 40 µs.

Pokud jsou všechny kroky inicializace dokončeny, přechází činnost automatu na odesílání příkazů, které nastaví chování displeje. Ty jsou uloženy jako



Obrázek 3.9: Stavový diagram nadřazené části automatu

konstanty v poli, z nějž jsou čteny. Na příkaz v poli, který se má provést, ukazuje číselný signál, který se po dokončení příkazu zvýší o 1.

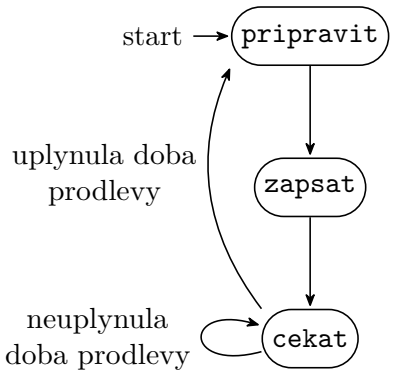
Příkazy uložené v poli jsou následující:

1. Function Set s hodnotou `28h`,
2. Entry Mode Set (`06h`),
3. Display On (`0Ch`),
4. Clear Display (`01h`),
5. nastavení kurzoru na 1. řádek (`80h`),
6. nastavení kurzoru na 2. řádek (`C0h`).

Přičemž poslední dva se opakují při cyklickém přepisování obsahu zobrazovače.

Horní 4 bity příkazu se odešlou ve stavu `horni`, následuje dolní polovina příkazu ve stavu `dolni`. Prodleva po odeslání příkazu je nastavena na 2 ms (`kolik_taktu` = 200). Pokud jsou všechny připravené příkazy odeslány, obvod přejde na odesílání textu.

Obdobně jako u příkazů, `text_h` odesílá horní čtyři bity znaku z řetězce a `text_l` dolní 4 bity znaku. Po odeslání dolních bitů následuje prodleva 40 µs (`kolik_taktu` = 4). Pokud je odeslán celý řádek (16 znaků), odešle se příkaz ke změně řádku (`80h`, případně `C0h`) opět ve stavech `horni` a `dolni`. Tento proces se cyklicky opakuje (kromě inicializace). Text zobrazovače je tudíž periodicky přepisován obsahem pole znaků.



Obrázek 3.10: Stavový diagram podřazené části automatu

Popis podřazené části automatu

K přechodu mezi stavy této části dochází při vzestupné hraně taktu 100 kHz, pokud je splněna podmínka k přechodu. V počátečním stavu `pripravit` jsou na sběrnici nastaveny bity určené podle nadřazeného stavu.

V dalším taktu dochází k bezpodmínečnému přechodu do stavu `zapsat`. Zde se nastaví stav výstupu `enable` na logickou 1, čímž se obsah datové sběrnice zapíše do displeje.

Následuje opět přímý přechod do stavu `cekat`, ve kterém automat setrvá po takový počet taktů, kolik je nastaveno v proměnné `kolik_taktu`. Tato hodnota závisí na významu právě zapsaných dat.

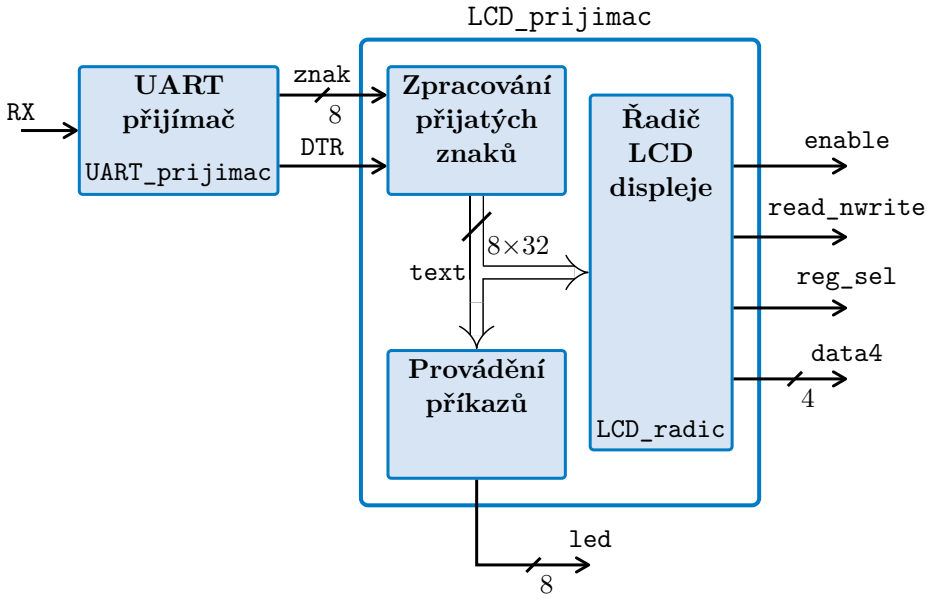
3.4.4 Výsledek

S takto navrženým automatem jsem získal VHDL entitu pro ovládání LCD zobrazovače, jejímž vstupem je pole znaků délky 32 a takt oscilátoru 50 MHz. Vstup pole znaků jsem připojil k obvodu, který v sobě ukládá přijaté znaky ze sériového rozhraní RS-232. Tento obvod budu dále popisovat.

3.5 Příjem textu přes UART, jeho zobrazení na LCD displeji a provádění příkazů

Tento obvod propojuje přijímač UART a řadič LCD zobrazovače z kapitoly 3.4. Obvod je realizován entitou `LCD_prijimac`, jež je v hlavní entitě propojena s obvodem `UART_prijimac`. Jako podobvod je v entitě `LCD_prijimac` obsažen již popsaný řadič displeje `LCD_radic`.

Obrázek 3.11 ukazuje blokové schéma tohoto funkčního celku. Ve schématu není zakreslen signál oscilátoru, který je přiveden ke všem blokům. Kromě výstupů řadiče LCD je zde i výstup `led` přivedený k osmi svítivým diodám na přípravku Spartan-3E.



Obrázek 3.11: Blokové schéma entity `LCD_prijimac` spolu s přijímačem UART

3.5.1 Zpracování přijatých znaků

Přijaté znaky jsou ukládány do řetězce `text` o délce 32 znaků. Znaky v něm jsou indexovány od jedničky, nikoliv od nuly jak bývá u řetězců zvykem. To je pozůstatek z doby, kdy jsem zkoušel používat typ `string`, který nelze od nuly indexovat. Od tohoto typu jsem nakonec upustil, jak zmiňuji v části 3.4.1.

Tento řetězec je přiveden na vstupní port řadiče `LCD_radic`, který se stará o vlastní zobrazení tohoto řetězce na LCD. Ukazatel na konec řetězce `index_textu` sleduje počet přijatých znaků.

V počátečním stavu je řetězec naplněn prázdnými znaky. Ukazatel na konec řetězce má hodnotu 1. Prázdný znak v řetězci je reprezentován znakem mezery, na jeho místě se nic nezobrazí.

Přijetím znaku se do řetězce tento znak zapíše (jako 8bitový kód) na pozici danou hodnotou ukazatele `index_textu`. Hodnota ukazatele se poté zvýší o 1. Pokud má `index_textu` hodnotu 32, přijetí dalšího znaku ho resetuje do 1 a způsobí smazání předešlého textu. Na displeji tudíž nejde souvisle zobrazit řetězec s více jak 32 znaky, protože se zobrazí pouze přesahující část.

Speciálním případem je přijetí znaku pro nový řádek CR (Carriage Return) s ASCII kódem `0Dh` nebo LF (Line Feed) s kódem `0Ah`. Tímto se ukazatel `index_textu` vrátí na začátek řetězce, bez ohledu na jeho předchozí hodnotu. Následný přijatý znak se zobrazí na začátku prvního řádku, přičemž všechny znaky za ním jsou smazány.

3.5.2 Provádění příkazů

Do komponenty `LCD_prijimac` jsem z důvodu, že již obsahuje stavový automat čtení znaků do řetězce, vložil i funkci rozpoznání a provedení přijatých

Obrázek 3.12: Ukázka zobrazení textu odeslaného ze sériového terminálu

příkazů. Nemusel jsem tak vytvářet další komponentu.

Vytvořil jsem příkaz pro ovládání stavu osmi LED diod na vývojové desce. Příkaz je formátu `led HH`, kde parametr `HH` je šestnáctkové číslo obsahující binární stav každé LED. Nejvyšší bit přísluší svítivé diodě nejvíce vlevo, nejnižší bit odpovídá diodě úplně vpravo.

Příkaz musí být ukončen znakem nové řádky, aby se provedl[3]. Po vzoru jazyka VHDL, jenž nerozlišuje velikost písmen, ji ani tento příkaz nerozlišuje. Příkaz navíc toleruje vynechání mezery před číselným parametrem. Příkazy `led A7`, `LEDa7` a `lEd A7` jsou tudíž ekvivalentní.

Rozpoznání příkazu je provedeno porovnáním prvních tří znaků v řetězci s malými či velkými písmeny v příkazu `LED`. Následně je každý ze dvou znaků parametru převeden pomocnou funkcí `hex_to_4bit` z jednoznakového šestnáctkového čísla na dvojkové 4 bity. Znaky neplatné v šestnáctkové soustavě se převedou na nulu.

3.5.3 Výsledek

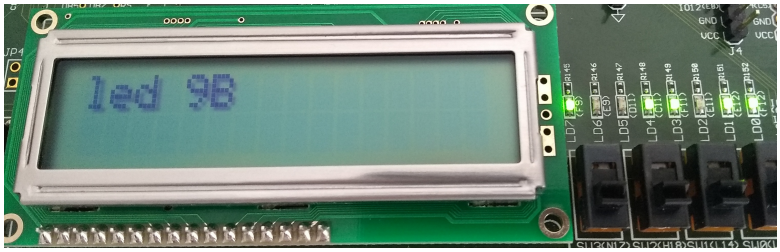
Desku jsem pomocí převodníku z RS-232 na USB propojil s počítačem. V programu `picocom` jsem otevřel příslušný sériový port a odeslal na něj text. Na LCD se tento text zobrazil, jak je vidět na obrázku 3.12. Displej nepodporuje znaky s diakritikou[4], takže není zcela vhodný pro češtinu. Ověřil jsem také provádění příkazu k rozsvěcení LED diod. Ukázka provedeného příkazu `led 9B` je na obrázku 3.13. Diody svítí v konfiguraci 10011011 zleva doprava, což odpovídá šestnáctkovému číslu `9Bh`.

3.6 Zobrazování znaků přijatých přes UART na monitoru přes VGA

Při návrhu grafického zobrazení znaků jsem se inspiroval textovým terminálem, jaký je například v operačním systému DOS. Vybral jsem režim obrazu o rozlišení 640×480 bodů se snímkovou frekvencí 60 Hz. Tento režim je

[3] V krajním případě zaplnění zobrazovače 32 znaky se příkaz také provede.

[4] Osm znaků by bylo možné dodefinovat, tento počet však pro česká písmena nestačí.



Obrázek 3.13: Ukázka ovládání LED diod pomocí sériového terminálu

zdokumentován v uživatelské příručce pro Spartan-3E [5] na stranách 55–59. Za rozměry znaku jsem zvolil šířku 8 obrazových bodů a výšku 16 bodů. Do zvoleného rozlišení se vejde 80 sloupců a 30 řádků takovýchto znaků.

3.6.1 Vývoj návrhu

Nejdříve jsem navrhl generátor synchronizačních signálů. Jeho správnou činnost jsem ladil pomocí simulace. Když byla synchronizace funkční, testoval jsem ji na monitoru kresbou barevných obdélníků, která se snadno realizuje odvozením barevného signálu z polohy paprsku.

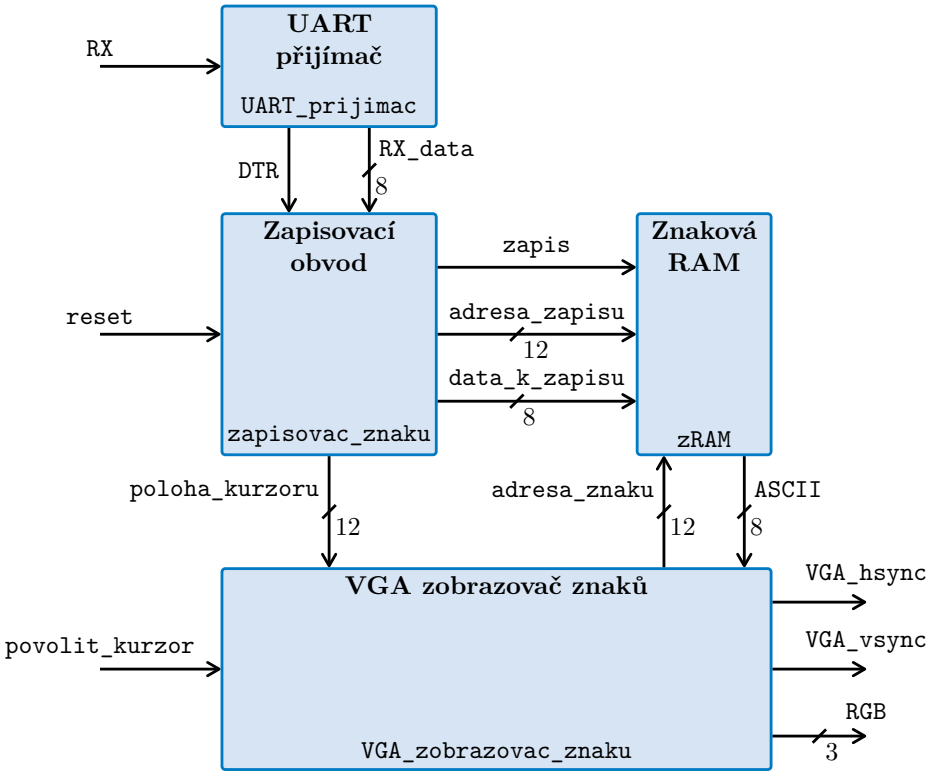
Na monitoru se v rané fázi testování zobrazovaly artefakty – barevné přechody. Zjistil jsem, že se objevují, pokud barevné signály nejsou nulové v době, kdy je pomyslný paprsek mimo obraz. Ošetřil jsem, aby v tuto dobu byly signály neaktivní.

Dále jsem obraz rozdělil na jednotlivá znaková políčka, kde každé má 8×16 bodů. V tomto okamžiku jsem ještě neměl znakovou ROM, takže jsem „ručně“ vykreslil znak „0“ do každého políčka.

Dal jsem se do hledání vhodného rastrového fontu, protože tvorba vlastního o zvolené velikosti by byla náročná. Po tom, co jsem si vybral font, jsem hledal, jak přidat a převést písmo do svého obvodu. Vhodný nástroj pro převod jsem nenalezl, rozhodl jsem se proto napsat vlastní v jazyce C++. To bych byl nucen udělat, i kdybych vytvořil vlastní písmo.

Po nějaké době jsem zdárně převedl vybrané písmo na kýžený formát a tak bylo možné jej použít. Vytvořil jsem znakovou paměť ROM (dohledal jsem, jak toto udělat v knize [13]) a inicializoval ji převedeným fontem. Za účelem otestování jsem vykreslil veškerý obsah ROM, adresaci paměti jsem opět odvodil z polohy paprsku.

Mým posledním krokem bylo vytvoření paměti RAM, která obsahuje znaky, které se mají na obrazovce zobrazit a zároveň obvodu, který znaky z UART do paměti zapisuje. Pro přehlednost jsem ještě přidal zobrazení polohy kurzoru. Tímto byla část s rozhraním VGA dokončena. Správnou činnost jsem ověřil posíláním různých textů z počítače. Jako barvu textu jsem nakonec zvolil bílou a k tomu modré pozadí. K tomu mne inspirovaly znakové LCD zobrazovače s modrým podsvícením.



Obrázek 3.14: Zapojení komponent, které tvoří obvod pro znakový VGA výstup

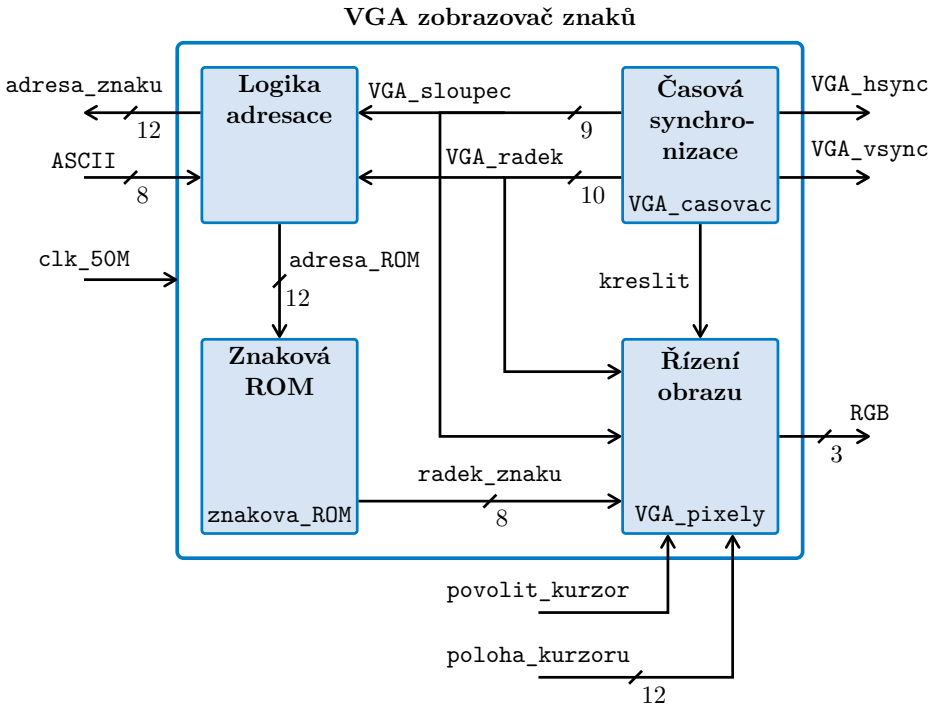
3.6.2 Struktura funkčního celku

Celkovou funkčnost zajišťuje několik obvodů. Z přijímače UART jsou čteny znaky a zapisovány do paměti RAM komponentou `zapisovac_znaku`. Tato paměť je cyklicky čtena entitou `VGA_zobrazovac_znaku`. Ta obstarává veškeré činnosti potřebné pro vytvoření obrazu z obsahu RAM a ROM pamětí. Zapojení všech komponent, které se podílejí na funkci textového VGA výstupu, je na obrázku 3.14.

3.6.3 Zobrazování znaků

Každý snímek obrazu je vykreslován řádek po řádku. V každém řádku prochází pomyslný elektronový paprsek zleva doprava. Řádek je rozdělen na 80 znaků, to znamená, že každých 8 vodorovných pixelů (obrazových bodů) se přečte odpovídající znak a zobrazí se jeho grafická podoba v aktuálním řádku. Takto je každých 16 obrazových řádků vykreslován jeden řádek znaků.

O zobrazování znaků přes VGA se stará entita `VGA_zobrazovac_znaku`. Její blokové znázornění spolu s vnitřními obvody a jejich propojením je na obrázku 3.15. Tyto bloky nyní blíže popíši.



Obrázek 3.15: Entita starající se o generování VGA obrazu

Vnitřní komponenty

Entita `VGA_zobrazovac_znaku` obsahuje další tři komponenty – generátor vertikální a horizontální synchronizace `VGA_casovac`, obvod pro řízení barevných signálů `VGA_pixely` a paměť s uloženým fontem `znakova_ROM`. Čtvrtý blok, označený „Logika adresace“, je implementován přímo v architektuře entity `VGA_zobrazovac_znaku`.

V obrázku 3.15 není, z důvodu přehlednosti, zakreslena distribuce přivedeného hodinového signálu. Všechny vnitřní bloky kromě znakové ROM běží na kmitočtu 25 MHz. Znaková paměť jako jediná je taktována přivedeným kmitočtem 50 MHz, a to z důvodu rychlejší adresace.

Blok „Logika adresace“ zajišťuje odvození adresy znaku v paměti RAM z polohy paprsku. Stará se také o předpřipravení kódu následujícího znaku, aby bylo možné rychle naadresovat řádek grafické podoby znaku v ROM během přechodu paprsku z jednoho znaku na druhý.

Vstupy a výstupy

V blokovém zapojení 3.15 jsou mezi bloky nakreslené signály spolu s jejich názvem ve zdrojovém kódu. Výstupy obvodu `VGA_RGB`, `VGA_hsync` a `VGA_vsync` jsou připojeny přímo k fyzickým výstupům na VGA konektor. Dvanáctibitový výstup `adresa_znaku` slouží k adresaci RAM znaků. Naadresovaný znak vstupuje přes osmibitový signál `ASCII`.

Pomocné vstupy `poloha_kurzoru` a `povolit_kurzor` jsou určeny k zob-

razení kurzoru. Poloha kurzoru je získávána z obvodu `zapisovac_znaku`. Přiveden je také hodinový signál `clk_50M`.

3.6.4 Generování synchronizačních signálů

Komponenta `VGA_casovac` zajišťuje správné časování VGA signálů. Na základě vstupního taktu 25 MHz generuje signály horizontální a vertikální synchronizace. Pro okolní komponenty je zde výstupem aktuální hodnota souřadnic paprsku (`VGA_radek` a `VGA_sloupec`) a příznak, že paprsek se nachází v zobrazovací oblasti – `kreslit`.

Synchronizační impulzy jsou generovány pomocí dvojitého stavového automatu. Dvojnásobný je proto, že vertikální synchronizace (VS) a horizontální synchronizace (HS) na sobě závisí, a tak jsou generovány společně.

Jak stav VS, tak stav HS mají ekvivalentní části průběhu signálu, proto je pro ně použit stejný výčtový datový typ. Jednotlivé fáze signálu se cyklicky opakují. Ty jsou dány pohybem onoho pomyslného paprsku a jsou čtyři: `pulz`, `navrat_paprsku`, `pixely`, `dobeh_paprsku`. Obrázek 3.16 zobrazuje stavový diagram HS i VS (v závorkách jsou podmínky přechodu platné pro VS).

Činnost stavového automatu

Synchronizační signál (`VGA_hsync` či `VGA_vsync`) je ve stavu logické nuly, dokud je příslušný automat ve stavu `pulz`. V ostatních stavech má signál stav logické 1.

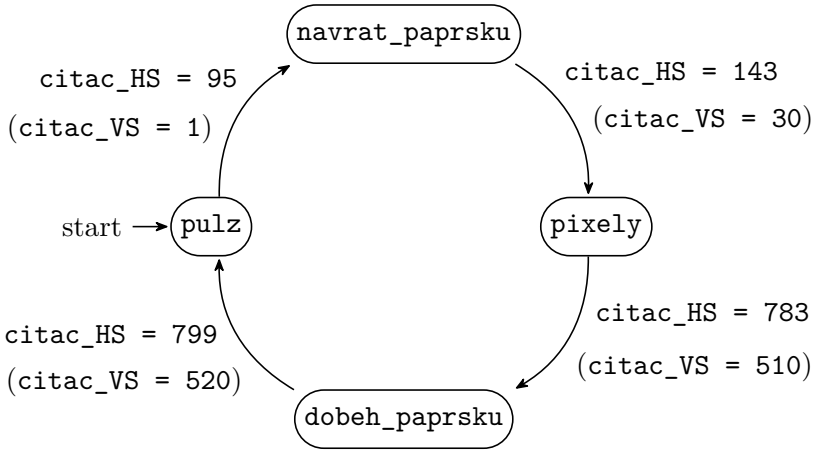
Stav `pixely` značí, že paprsek se nachází v aktivní oblasti obrazovky. Hodnota `VGA_sloupec` udává číslo sloupce právě kresleného bodu, proto je čítána tehdy, když je HS v tomto stavu. Obdobně `VGA_radek` udává číslo řádku, a je čítán pouze tehdy, je-li VS ve stavu `pixely`. Jestliže jsou oba automaty v tomto stavu, je signál `kreslit` v úrovni log. 1.

Změny stavů jsou dány hodnotami čítače `citac_HS`, respektive `citac_VS`. Hodnota čítače `citac_HS` je inkrementována každý takt hodin, hodnota `citac_VS` je zvýšena o 1 na konci každého řádku (tj. po celém cyklu HS). Po každém cyklu stavů (fází signálu) se čítače nulují.

3.6.5 Řízení barevných signálů

O řízení barevných signálů se stará komponenta `VGA_pixely`, jež je umístěna vpravo dole ve schématu 3.15. Její jediný výstup je tříbitový barevný signál RGB. Přivedený hodinový takt 25 MHz odpovídá kmitočtu kreslení obrazu. Vstupy `VGA_radek` a `VGA_sloupec` jsou přivedeny z entity `VGA_casovac`. Obsahují souřadnice aktuálního kresleného obrazového bodu.

Osmibitový vstup `radek_znaku` je přiveden z datového výstupu paměti ROM. Vyjadřuje grafickou podobu aktuálního řádku znaku. Nejvyšší bit představuje obrazový bod na levém okraji znaku. Obdobně nejnižší bit leží na pravém okraji. Na základě této informace je rozhodnuto, zda se vykreslí barva pozadí (pokud je bit 0) nebo barva popředí/znaku (bit je 1).



Obrázek 3.16: Stavový diagram časování horizontální (vertikální) synchronizace

Signál `kreslit` indikuje, že paprsek není zatemněn čili je v oblasti pro vykreslování. U mnou použitého LCD monitoru nesmí být žádný signál barvy aktivní mimo dobu vykreslovaného obrazu. V opačném případě se objeví barevné artefakty.

Vstupy `poloha_kurzoru` a `povolit_kurzor` slouží ke zvýraznění polohy kurzoru. To je provedeno prohozením barvy pozadí a barvy znaku. Vstup `povolit_kurzor`, který zapíná či vypíná zobrazení kurzoru, je připojen ke druhému spínači zprava na přípravku Spartan-3E.

3.6.6 Písmo

Pro textový výstup na VGA jsem potřeboval mít k dispozici grafickou podobu znaků přístupnou hradlovému poli ve vhodném formátu. Toho jsem dosáhl převedením vybraného fontu na mnou zvolený dvojkový formát, a následně zapsáním tohoto formátu do paměťové komponenty ROM v hradlovém poli. Níže podrobně rozepisuji, jaké kroky mne vedly k tomuto cíli.

Výběr stylu písma

Mými kritérii pro výběr fontu byly tyto náležitosti:

- Font obsahuje české znaky s diakritikou – přijde mi vhodné mít možnost plně zobrazit i český text.
- Má svobodnou licenci – aby bylo jisté, že mám právo ho používat či kopírovat/distribuovat bez licenčních poplatků.
- Je rastrový s pevnou velikostí, která se vejde do obdélníku 8×16 včetně drobné mezery (mezi znaky vedle sebe).
- Je dostupný v textovém formátu BDF (Glyph Bitmap Distribution Format), který je jednoduše zpracovatelný.

Těmto kritériím vyhovovalo více písem, proto jsem dále požadoval, aby písmo bylo distribuováno pro znakovou sadu Latin-2 (ISO 8895-2), jež obsahuje všechna písmena používaná v češtině. Navíc má tato sada osmibitové kódování kompatibilní se sedmibitovým kódováním ASCII. Tento požadavek značně zúžil výběr písem.

Nakonec jsem si vybral písmo nazvané „UW ttyp0“ [14], jehož autorem je doktor Uwe Waldmann z německé Společnosti Maxe Plancka. Licence dovoluje písmo šířit, pokud je přiložen licenční soubor. Pokud bych však písmo upravil, musel bych dle licence změnit název písma.

Toto písmo je z mnou nalezených unikátní tím, že se jeho soubory generují pomocí nastavitelných skriptů. Z vygenerovaných řezů písma se mi nejvíce líbila varianta normálního písma o velikosti 15 bodů. Tu jsem posléze v rámci této práce použil.

Znaky tohoto fontu jsou ukázány v tabulce C.4 na straně 52. Sloupce jsou označeny dolní částí (číslicí) šestnáctkového kódu znaku, řádky pak horní částí kódu. Kód znaku je dán součtem čísla pro sloupec a čísla pro řádek, ve kterém znak nachází. Z tabulky jsem vyřadil znaky s kódy `80h` – `9Fh`, jelikož jsou prázdné.

Nutno podotknout, že znaky s kódy `04h`, `0Ah`, `0Ch` a `0Dh` nelze zobrazit, neboť mají funkci řídicích znaků v navrženém obvodu. Při porovnání s tabulkou řídicích ASCII znaků C.2 si lze povšimnout, že znaky reprezentovány dvěma písmeny jejich názvu (v 1. řádku tabulky) mají odlišný kód od jejich pozice v ASCII tabulce. To je drobná chyba v tomto fontu.

Dvojkový formát písma

Binární podobu písma jsem zvolil takovou, aby ji šlo snadno číst a interpretovat. Každý znak je reprezentován šestnácti bajty uloženými na adresách jdoucích za sebou, kde první bajt reprezentuje první řádek, druhý bajt druhý řádek, atd. Tam, kde v osmici bitů je binární 1, je vykreslena barva znaku. Tam, kde je 0, je vykreslena barva pozadí. Podoba takového znaku je znázorněna na obrázku 3.17.

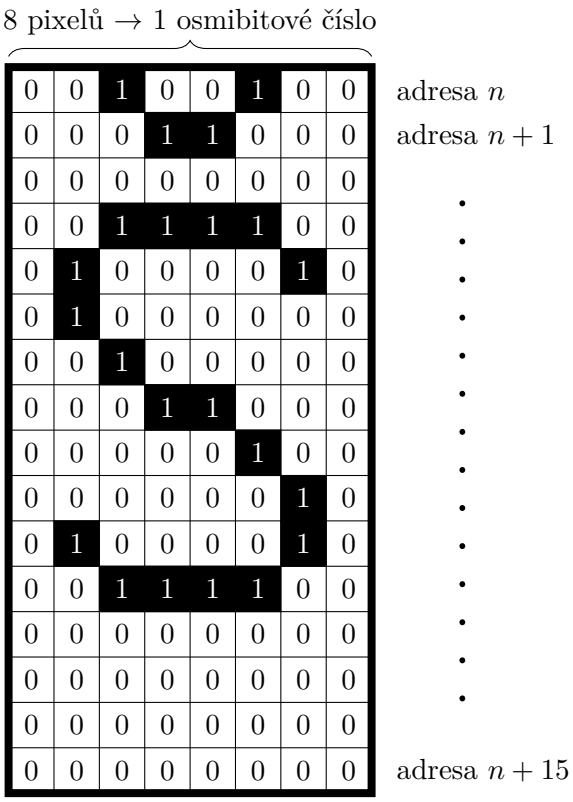
Převod písma

Soubor písma formátu BDF jsem potřeboval převést na svůj dvojkový formát, který se následně nahraje do blokové paměti FPGA. K inicializaci paměti slouží v Xilinx ISE textový formát souboru `.coe`.

Pro převod písma do tohoto formátu jsem nenašel vhodný nástroj, proto jsem se rozhodl vytvořit vlastní. Nástroj jsem napsal v jazyce C++. Jeho činnost je inspirována programem `bdf2c`[5], který převádí fonty BDF na hlavičkový soubor jazyka C. Můj program `bfd_to_coe` je o něco jednodušší (má méně funkcí a je méně univerzální[6]). Svůj účel však zvládá náležitě plnit.

[5] `https://github.com/pixelmatix/bdf2c`

[6] Podporuje pouze fonty s maximální šířkou 8 bodů.



Obrázek 3.17: Dvojkový formát znaku

3.6.7 Znaková paměť ROM

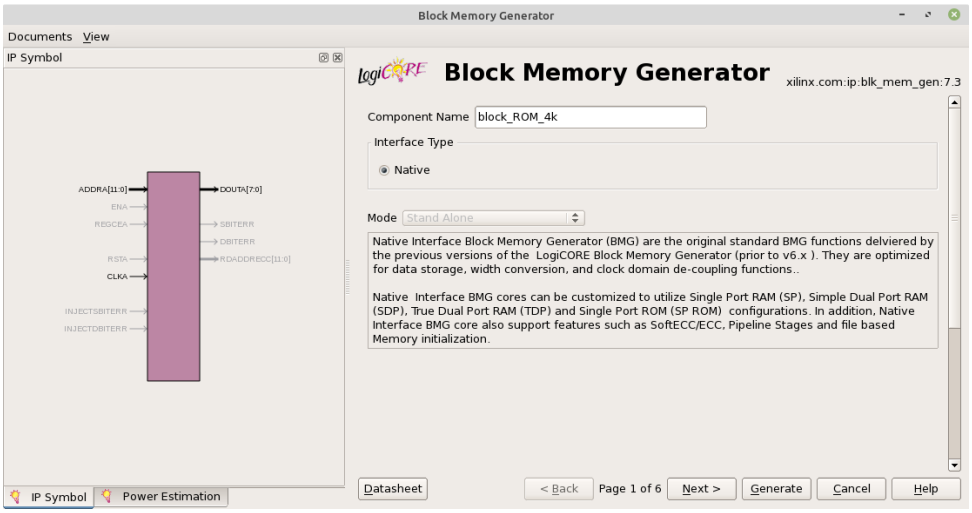
Tato paměť obsahuje grafickou podobu znaků mnou použitého fontu, jež byla získána mnou vytvořeným nástrojem na převod. Jelikož obsah takovéto paměti není žádoucí měnit, byl zvolen typ pouze pro čtení – ROM (Read Only Memory).

Formát uložených dat byl již v předchozí části popsán. Font o velikosti 8×16 bodů s 256 znaky zabírá 4 kB paměti. K adresaci takto veliké paměti je použita adresa dlouhá 12 bitů. Dvanáctibitová adresa znaku v paměti ROM je složena ze dvou částí:

- Horních 8 bitů odpovídá kódu znaku v kódové tabulce Latin-2.
- Dolní 4 bity, tedy indexy 0 až 15, značí číslo řádku grafické podoby znaku.

Vytvoření blokové ROM

V programu Xilinx CORE Generator jsem vytvořil komponentu pro blokovou paměť pomocí průvodce „Block Memory Generator“. Paměť jsem pojmenoval `block_ROM_4k` (4k značí velikost 4 kB). Její instanci jsem poté nazval `znakova_ROM`.



Obrázek 3.18: Okno průvodce pro generování blokové paměti

Jako typ vygenerované paměti jsem zvolil „Single-port ROM“, což znamená paměť pouze pro čtení s jedním datovým portem. V hradlovém poli je však stále implementována pomocí blokové RAM.

Šířku paměťového místa (a tudíž i datové sběrnice) jsem nastavil na 8 bitů, hloubku (neboli velikost) na 4096 paměťových míst. Tomu odpovídá šířka adresové sběrnice 12 bitů. Takto vytvořenou paměť jsem přidal jako komponentu do entity `zobrazovac_znaku` a propojil ji s okolními obvody.

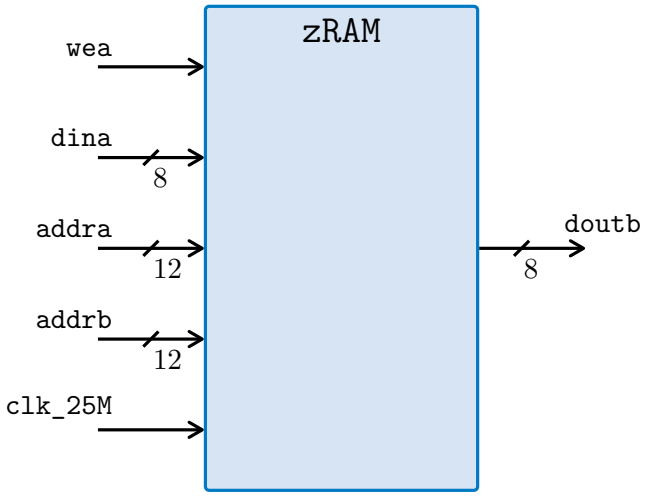
Snímek okna průvodce pro vytvoření blokové paměti je na obrázku 3.18. Aktivními vstupy (na obrázku černě) jsou pouze adresní sběrnice `ADDRA` a hodinový signál `CLKA`. Jediným aktivním výstupem jsou naadresovaná data `DOUTA`.

Inicializace ROM

Jak je uvedeno v [1], vygenerovaná bloková paměť se inicializuje textovým souborem „koeficientů“ `.coe` (coefficients file). V hlavičce je uveden základ číselné soustavy (v desítkové soustavě), ve které jsou jednotlivé inicializační hodnoty zapisovány. Poté je vypsán vlastní seznam číselných hodnot obsahu paměti. Každé z čísel odpovídá jednomu paměťovému místu. Hodnoty jsou vypisovány postupně od adresy 0 a oddělené čárkami.

Jako ukázka je zde vložena část obsahu inicializačního souboru pro paměť o šířce 8 bitů. Hodnoty jsou v šestnáctkové soustavě, jak je definováno na první řádce:

```
memory_initialization_radix=16;
memory_initialization_vector=
00, 00, 5a, 42, 42, 00, 42, 42, 00, 42, 42, 5a, 00, 00, 00, 00,
.
.
.
00, 00, 00, 10, 10, 7c, 10, 10, 10, 10, 12, 0c, 08, 04, 08, 00,
00, 18, 18, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00;
```



Obrázek 3.19: Bloková RAM znaků

3.6.8 Znaková paměť RAM

K ukládání informace o tom, kde na obrazovce leží jaký znak, slouží znaková paměť RAM. Tato paměť je obsažena přímo v hlavní entitě. Její rozhraní ukazuje obrázek 3.19.

Je to druhá bloková paměť vygenerovaná nástrojem Xilinx CORE Generator. Na rozdíl od znakové ROM však má dva nezávislé porty – jeden určen pro zápis (port A) a druhý pro čtení (port B). Oba porty jsou taktovány stejným kmitočtem oscilátoru 50 MHz.

Čtecí brána B je připojena pomocí datové a adresové sběrnice ke komponentě `VGA_zobrazovac_znaku`. Zápis přijatých znaků je potom řízen obvodem `zapisovac_znaku`. Ten ovládá vstupy brány A. Oproti čtecímu portu je zde navíc vstup povolení k zápisu `wea` (Write Enable A).

Z nějakého důvodu je datový typ vstupu `wea` jednoprvkový bitový vektor `std_logic_vector`, nikoliv bitová hodnota `std_logic`. To jsem musel vzít v úvahu při propojování portů, protože tyto dva typy nelze bez převodu propojit[7].

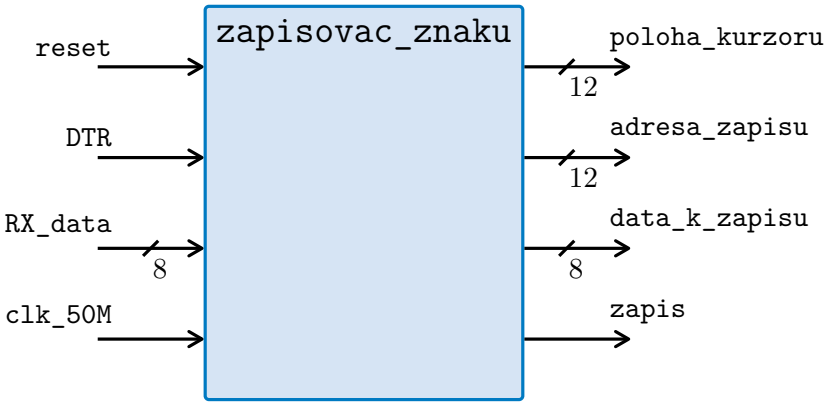
Způsob adresace znaků v RAM

Každé místo v paměti obsahuje osmibitový kód znaku (v kódování Latin-2). Poloha znaku na obrazovce, která danému paměťovému místu odpovídá, je dána formátem adresy.

Dvanáctibitová adresa je tvořena těmito částmi:

- Horní část (7 bitů) je číslo znakového sloupce. V mé konfiguraci toto číslo nabývá hodnot 0 až 79. Sedm bitů by mohlo adresovat až 127 sloupců, tyto vyšší adresy však nejsou v adresním prostoru zahrnuty.

[7] Převod se provede přístupem k nultému prvku vektoru, který již je typu `std_logic`.



Obrázek 3.20: Entita `zapisovac_znaku`

- Dolní část (5 bitů) je číslo znakového řádku. Obraz je rozdělen na 30 řádků znaků, tomu odpovídají hodnoty části adresy 0 až 29. Protože však pětibitové číslo může nabývat hodnot až do 31, zůstávají indexy řádku znaku 30 a 31 nevyužity.

Výsledná velikost potřebné paměti je tudíž $80 \times 32 = 2560$ bajtů. Tu jsem při generování této komponenty zadal do průvodce. Tento formát má za následek, že v paměti jsou znaky uloženy „transponovaně“, tedy postupně podle adresy jsou za sebou znaky v sloupci a ne v řádku.

Uvažoval jsem také o více přirozeném formátu adresy, kdy horní a dolní část by byly prohozené. Tím by byly znaky, které jsou na řádku za sebou, za sebou také adresově. Nevyužité sloupce 80 až 127 bych však musel zahrnout do adresního prostoru, a tím by byly větší nároky na velikost paměti (4 kB).

Trochu mne zklamal fakt, že přestože jsem zvolil menší paměť, v FPGA stejně zabírá celé 4 kB paměti (tj. dva 2kB bloky). Nakonec jsem si tedy nepomohl. Na hradlovém poli s jemněji rozdělenými bloky paměti by však tento krok nějaké místo ušetřil.

3.6.9 Zapisování přijatých znaků do RAM

Přijaté znaky přijímačem UART jsou zapisovány do znakové paměti RAM komponentou `zapisovac_znaku`. Ta je znázorněna na obrázku 3.20.

Vstupy `DTR` a `RX_data` jsou připojeny k výstupům UART přijímače. Vstup `reset` aktivuje v úrovni logické 1 mazání paměti, a tudíž zobrazených znaků. Výstupy `adresa_zapisu`, `zapis` a `data_k_zapisu` slouží k zapisování do RAM. Obvod také zajišťuje posouvání kurzoru, jehož polohu předává na výstup označený jako `poloha_kurzoru`.

Čtení znaku z přijímače je založeno na detekci vzestupné hrany signálu `DTR`. Detekce hrany je implementována porovnáváním současné hodnoty a hodnoty v předchozím taktu hodin. Jakmile je hrana detekována, je nový znak zpracován a případně zapsán do znakové RAM.

Obvod se zčásti chová jako textový terminál, tj. každý přijatý tisknutelný znak posouvá kurzor o 1 znak doprava, případně kurzor přesune na nový

řádek. Kurzor ukazuje na místo, kam se zapíše následující přijatý znak, tím pádem je jeho poloha použita k adresaci paměti.

Obvod `zapisovac_znaku` rozpoznává tyto řídicí znaky, které nezapisuje:

- Znaky CR (Carriage Return, *Návrat vozíku*) a LF (Line Feed, *Posun řádku*) posunou kurzor na začátek dalšího řádku, případně na první řádek, pokud je kurzor na posledním. Terminál `picocom` posílá znak CR stiskem klávesy `Enter`.
- Znak FF (Form Feed, *Nová stránka*) vrátí kurzor na počátek obrazovky. Tento znak je možné v sériovém terminálu poslat kombinací kláves `Control+L` (někdy značeno jako `^L`).
- Znak EOT (End of Transmission, *Konec přenosu*) smaže obrazovku. Je možné jej poslat klávesami `Control+D`.

Mazání obrazovky

Mazání lze spustit jak řídicím znakem EOT, tak spínačem nejvíce vpravo na desce (posunutím jezdce do horní polohy, tj. do stavu log. 1). Spínač blokuje příjem znaků, dokud setrvává v horní poloze.

Mazání obrazovky nemůže být provedeno celé najednou, protože použitá paměťová komponenta nepodporuje smazání celé paměti. Navíc použitý font má neprázdný znak s kódem `00h`, takže „mazání“ je ve skutečnosti vyplňování znakem mezera (s kódem `20h`). Vyřešil jsem to tak, že paměť je přepisována bajt po bajtu, kdy každý takt je změněn právě jeden bajt.

Abych se vyhnul konfliktům při mazání a příjmu znaku, ověřil jsem, že se celá obrazovka smaže v čase příjmu jednoho znaku přes UART rychlostí 115200 b/s:
Počet mazaných míst: $80 \times 32 = 2560$, to odpovídá počtu potřebných taktů.
Počet period 50 MHz, kolik zabere příjem jednoho znaku[8]:

$$\frac{T_\mathrm{b} \cdot 8}{T_\mathrm{clk}} = \frac{f_\mathrm{clk} \cdot 8}{R_\mathrm{b}} = 50 \cdot 10^6 \cdot \frac{8}{115200} = 3472, \tag{3.2}$$

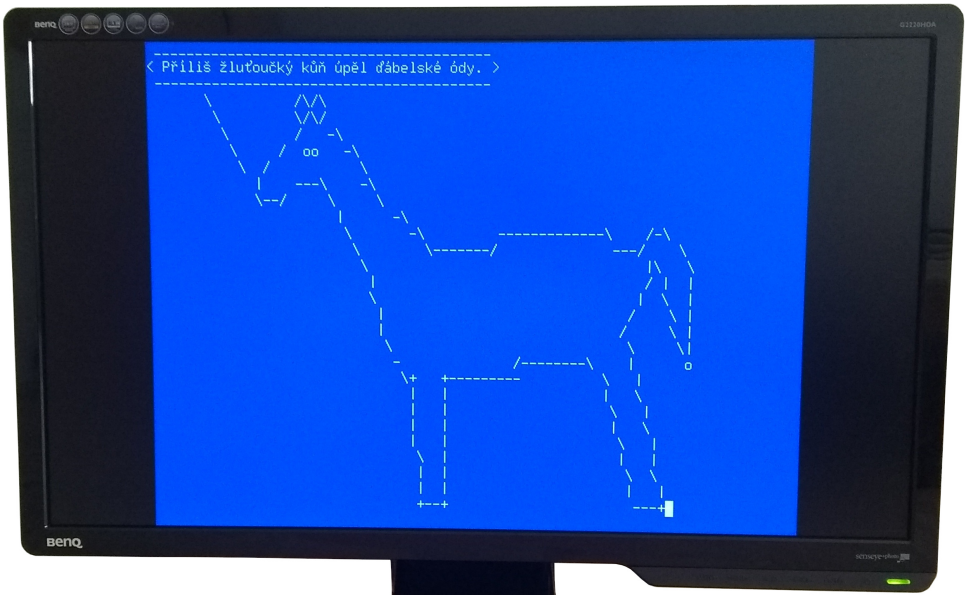
T_b je perioda jednoho bitu UART, T_clk je perioda hodin, f_clk je kmitočet hodin a R_b bitová rychlost.

Z výsledků lze vidět, že mazání obrazu trvá méně taktů hodin než příjem jednoho znaku. Proto by neměl být problém s přijímáním znaků přijatých bezprostředně po znaku FF, který spouští mazání.

3.6.10 Výsledek

Výsledkem této části práce je zobrazování znaků přijatých pomocí UART na obrazovce přes VGA. Obrazovka je rozdělena na 80 znakových sloupců a 30 řádků, což odpovídá standardním rozměrům textového terminálu. Výstupy mnohých programů z příkazové řádky je tudíž možné zobrazit bez

[8] Bez započítání start/stop bitů.



Obrázek 3.21: Ukázka textového výstupu VGA na monitoru

přetečení řádků. Výstup programů s barevným textem však nelze použít, jelikož jsou tam obsaženy řídicí sekvence znaků.

Na ukázku jsem přes sériový terminál poslal výstup programu `cowsay`. Výsledek zobrazený na monitoru ukazuje obrázek 3.21. V pravém dolním rohu lze vidět kurzor jakožto bílý obdélník. Pro správné zobrazení českých znaků jsem text převedl z kódování UTF-8, jež je na Linuxu používáno, na kódování Latin-2 (ISO 8859-2) nástrojem `iconv`. Zde uvádím celý skript použitý k vygenerování a odeslání tohoto ukázkového textu:

```
TEXT=$(cowsay -f pony "Příliš žluťoučký kůň úpěl ďábelské \
ódy." | iconv -f UTF-8 -t ISO8859-2);
picocom -t $'\x04'"$TEXT" -b 115200 -l /dev/ttyUSB0
```

Před textem je poslán řídicí znak EOT (s kódem `04h`), který smaže obrazovku.

Transparentní terminál (tj. stisk klávesy ihned odešle příslušný znak či sekvenci znaků) s automatickým převodem kódování do Latin-2 lze otevřít tímto příkazem:

```
luit -encoding ISO8859-2 picocom --echo -b115200 -l /dev/ttyUSB0
```

Program `luit` se zde stará o převod kódování, mezi sériovým terminálem `picocom` a systémovým terminálem Linuxu. Přepínač `--echo` zapne místní zobrazení odeslaných znaků.

Kapitola 4

Závěr

Dle zadání jsem vytvořil obousměrnou komunikaci mezi počítačem a přípravkem Spartan-3E přes sériové rozhraní RS-232. Stiskem tlačítka se do počítače odešle text určující polohu onoho tlačítka. Přijatý text je tisknut na LCD displej a zároveň je zobrazován přes rozhraní VGA na monitoru. Implementován je také příkaz pro rozsvícení nebo zhasnutí osmi LED diod na přípravku.

V praktické části jsou podrobně popsány principy činnosti jednotlivých komponent, které tvoří dané funkční celky. K popisu jsou využity blokové a vývojové diagramy. Dosažené cíle jsou zdokumentovány fotografiemi. Výsledné zdrojové soubory návrhu v jazyce VHDL jsou z důvodu rozsáhlosti obsaženy v elektronické příloze.

U výstupu na rozhraní VGA jsem navíc implementoval indikaci polohy kurzoru na obrazovce, následně i smazání všech znaků z obrazovky. Pro účely práce jsem rovněž v jazyce C++ napsal nástroj, který převede rastrový font na formát, jenž lze použít v hradlovém poli.

Funkce VGA výstupu by mohla být rozšířena o interpretaci standardních řídicích sekvencí znaků, například pro přesun kurzoru nebo změnu barvy textu. Tato bakalářská práce mi pomohla prohloubit své znalosti jazyka VHDL a blíže se seznámit s obsluhou programovatelných hradlových polí.

Literatura

[1] *Spartan-3 Generation FPGA User Guide* [online]. Xilinx, 2011 [cit. 2021-02-22]. str. 36–183. Dostupné z: https://www.xilinx.com/support/documentation/user_guides/ug331.pdf

[2] ANTOŠOVÁ, Marcela a Vratislav DAVÍDEK. *Číslicová technika*. 4. vyd. České Budějovice: KOPP, 2015. ISBN 978-80-7232-394-4.

[3] *Accolade VHDL Reference Guide* [online]. Altium Limited, 2001 [cit. 2021-04-26]. str. 34–38. Dostupné z: https://www.ics.uci.edu/~jmoorkan/vhdlref/Accolade%20VHDL%20Language%20Guide.pdf

[4] *ISE Design Suite 14: Release Notes, Installation, and Licensing* [online]. Xilinx, 2020 [cit. 2021-04-23]. str. 7-8. Dostupné z: https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/irn.pdf

[5] *Spartan-3E FPGA Starter Kit Board User Guide* [online]. Xilinx, 2011 [cit. 2021-01-06]. Dostupné z: https://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf

[6] OLMR, Vít. HW server představuje – Sériová linka RS-232. *HW.cz – profesionální elektronika* [online]. 2005 [cit. 2021-05-03]. Dostupné z: https://vyvoj.hw.cz/rozhrani/hw-server-predstavuje-seriova-linka-rs-232.html

[7] FRENZEL, Louis E. ml. *Handbook of Serial Communications Interfaces: A Comprehensive Compendium of Serial Digital Input/Output (I/O) Standards* [online]. kap. 25, str. 110–111. Oxford: Elsevier Science & Technology, 2015 [cit. 2021-05-04]. ISBN 978-0-12-800629-0. Dostupné také z: https://ebookcentral.proquest.com/lib/cvut/reader.action?docID=2189944

[8] Serial HOWTO: Voltage Waveshapes. *The Linux Documentation Project* [online]. 2011 [cit. 2021-05-16]. Dostupné z: https://tldp.org/HOWTO/Serial-HOWTO-20.html#ss20.3

[9] *Nexys 3™ FPGA Board Reference Manual* [online]. Digilent, 2016 [cit. 2021-05-09]. Dostupné z: https://reference.digilentinc.com/_media/nexys:nexys3:nexys3_rm.pdf

[10] CZYBORRA, Roman. *The ISO 8859 Alphabet Soup* [online]. 1998 [cit. 2021-05-14]. Dostupné z: `http://czyborra.com/charsets/iso8859.html`

[11] Why isn't this VHDL falling edge detector reliable?. *Electrical Engineering Stack Exchange* [online]. 2017 [cit. 2021-04-22]. Dostupné z: `https://electronics.stackexchange.com/q/295362`

[12] *HD44780U Dot Matrix Liquid Crystal Display Controller/Driver* [online]. Hitachi, 1998 [cit. 2021-02-12]. Dostupné z: `https://www.pololu.com/file/0J72/HD44780.pdf`

[13] FIELD, Mike. *Introducing the Spartan 3E FPGA and VHDL* [online]. kap. 15, str. 61–70. 2014 [cit. 2021-04-30]. Dostupné z: `https://github.com/hamsternz/IntroToSpartanFPGABook`

[14] WALDMANN, Uwe. UW ttyp0 – Monospace Bitmap Screen Fonts for X11. *Max Planck Institute for Informatics – Automation of Logic Department* [online]. 2017 [cit. 2021-02-22]. Dostupné z: `https://people.mpi-inf.mpg.de/~uwe/misc/uw-ttyp0/`

[15] CERF, Vint. *ASCII format for network interchange* [online]. RFC Editor, 1969 [cit. 2021-05-14]. Dostupné z: DOI: 10.17487/RFC0020

[16] GIBARA, Tom. *ASCII table* [online]. GitHub, 2014 [cit. 2021-05-14]. Dostupné z: `https://github.com/tomgibara/ascii-table`

Příloha A

Seznam použitých zkratek

atd.	a tak dále
log.	logická (hodnota)
např.	například
tj.	to jest
tzv.	takzvaně
ASCII	American Standard Code for Information Interchange
BDF	(Glyph) Bitmap Distribution Format
CMOS	Complementary Metal Oxide Semiconductor
CR	Carriage Return
CTS	Clear to Send
DOS	Disk Operating System
D-sub	D-subminiature
DVD	Digital Video Disc
EOT	End of Transmission
FF	Form Feed
FPGA	Field Programmable Gate Array
HS	Horizontální synchronizace
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LF	Line Feed
LSB	Least Significant Bit
MSB	Most Significant Bit
RAM	Random Access Memory
RGB	Red-Green-Blue
ROM	Read Only Memory

RTL	Register Transfer Logic
RWM	Read-Write Memory
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VGA	Video Graphics Array
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VS	Vertikální synchronizace

Příloha B

Obsah elektronické přílohy na DVD

- `BAP_Šedivý_Jan_2021.pdf` ... Elektronická verze této bakalářské práce
- `BAP_Spartan3E/` .. Složka s projektem vývojového prostředí Xilinx ISE
  - `BAP_Spartan3E.xise`
  - `hlavni.ucf` ..... Soubor přiřazení vývodů
  - `hlavni.vhd` ..... Hlavní zdrojový soubor VHDL
  - `impact.ipf`
  - `knihovna.vhd`
  - `LCD_prijimac.vhd`
  - `LCD_radic.vhd`
  - `PROM.mcs`
  - `UART_RX.vhd`
  - `UART_tlacitka.vhd`
  - `VGA_casovac.vhd`
  - `VGA_pixely.vhd`
  - `VGA_zobrazovac_znaku.vhd`
  - `zapisovac_znaku.vhd`
  - `coregen/` ..... Soubory nástroje CORE Generator
    - `block_RAM_2k56.ngc`
    - `block_RAM_2k56.vhd`
    - `block_RAM_2k56.xco`
    - `block_RAM_2k56.xise`
    - `block_ROM_4k.ngc`
    - `block_ROM_4k.vhd`
    - `block_ROM_4k.xco`
    - `block_ROM_4k.xise`
    - `coregen.cgc`
    - `coregen.cgp`
    - `coe/` ..... Inicializační soubory blokové paměti
      - `cowsay.coe`
      - `t0-15-i02.coe`
- `bdf_to_coe/` ..... Nástroj pro převod fontu na formát `.coe`
  - `bdf_to_coe.cpp` ..... Zdrojový soubor v jazyce C++

- `font/` ..... Složka obsahující písmo UW ttyp0
  - `LICENSE` ..... Licence písma
  - `t0-15-i02.bdf` ..... Vygenerovaný soubor fontu ve formátu BDF
  - `t0-15-i02.coe` ..... Font převedený na formát `.coe`
  - `uw-ttyp0-1.3.tar.gz` ..... Zdrojový archiv písma

Příloha C

Tabulky

Tabulka C.1: Tabulka tisknutelných znaků ASCII [15]

	0h	1h	2h	3h	4h	5h	6h	7h	8h	9h	Ah	Bh	Ch	Dh	Eh	Fh
20h	␣	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
30h	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40h	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50h	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60h	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70h	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	

Tabulka C.2: Tabulka řídicích znaků ASCII [15][16]

Kód	Hex	Klávesy[a]	Zkratka	Název
0	00h	`^@`	NUL	Null
1	01h	`^A`	SOH	Start of Heading
2	02h	`^B`	STX	Start of Text
3	03h	`^C`	ETX	End of Text
4	04h	`^D`	EOT	End of Transmission
5	05h	`^E`	ENQ	Enquiry
6	06h	`^F`	ACK	Acknowledge
7	07h	`^G`	BEL	Bell
8	08h	`^H`	BS	Backspace
9	09h	`^I`	HT	Horizontal Tabulation
10	0Ah	`^J`	LF	Line Feed
11	0Bh	`^K`	VT	Vertical Tabulation
12	0Ch	`^L`	FF	Form Feed
13	0Dh	`^M`	CR	Carriage Return
14	0Eh	`^N`	SO	Shift Out
15	0Fh	`^O`	SI	Shift In
16	10h	`^P`	DLE	Data Link Escape
17	11h	`^Q`	DC1	Device Control 1
18	12h	`^R`	DC2	Device Control 2
19	13h	`^S`	DC3	Device Control 3
20	14h	`^T`	DC4	Device Control 4
21	15h	`^U`	NAK	Negative Acknowledge
22	16h	`^V`	SYN	Synchronous Idle
23	17h	`^W`	ETB	End of Transmission Block
24	18h	`^X`	CAN	Cancel
25	19h	`^Y`	EM	End of Medium
26	1Ah	`^Z`	SUB	Substitute
27	1Bh	`^[`	ESC	Escape
28	1Ch	`^\`	FS	File Separator
29	1Dh	`^]`	GS	Group Separator
30	1Eh	`^^`	RS	Record Separator
31	1Fh	`^_`	US	Unit Separator
127	7Fh	`^?`	DEL	Delete

[a]Kombinace kláves (na anglické klávesnici), která znak v terminálu posílá. Znak „`^`“ na začátku představuje klávesu `Control`

Tabulka C.3: Tabulka znaků LCD a jejich kódů [12]

Lower 4 Bits \ Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	@	P	`	p				—	9	≡	α	p
xxxx0001	(2)		!	1	A	Q	a	q			。	ア	チ	ム	ä	q
xxxx0010	(3)		"	2	B	R	b	r			「	イ	ツ	メ	ß	θ
xxxx0011	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
xxxx0100	(5)		$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	σ	ü
xxxx0110	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)		'	7	G	W	g	w			ァ	キ	ヌ	ラ	g	π
xxxx1000	(1)		(	8	H	X	h	x			ィ	ク	ネ	リ	√	x̄
xxxx1001	(2)		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010	(3)		*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
xxxx1011	(4)		+	;	K	[	k	{			ォ	サ	ヒ	ロ	㎏	万
xxxx1100	(5)		,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
xxxx1101	(6)		-	=	M	]	m	}			ュ	ス	ヘ	ン	£	÷
xxxx1110	(7)		.	>	N	^	n	→			ョ	セ	ホ	゜	ñ	
xxxx1111	(8)		/	?	O	_	o	←			ッ	ソ	マ	゛	ö	■

Tabulka C.4: Tabulka znaků fontu UW ttyp0 (Latin-2) o velikosti 8×15 bodů