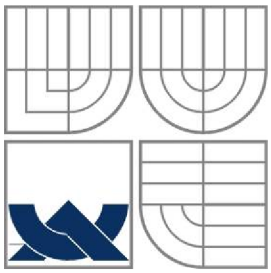
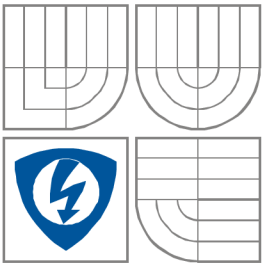


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

JÁDRO OBVODU FPGA PRO ZOBRAZENÍ DAT NA MONITORU PROSTŘEDNICTVÍM PORTU VGA

FPGA CORE FOR DATA DISPLAYING ON COMPUTER MONITOR USING VGA PORT

BAKÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE Adam Pišl
AUTHOR

VEDOUCÍ PRÁCE Ing. Michal Kubíček
SUPERVISOR

BRNO, 2008

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: Adam Pišl
Bytem: Trlicova 58, Nový Jičín, 74101
Narozen/a (datum a místo): 11. října 1985 v Bílovci
(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií
se sídlem Údolní 53, Brno, 602 00
jejímž jménem jedná na základě písemného pověření děkanem fakulty:
prof. Dr. Ing. Zbyněk Raida, předseda rady oboru Elektronika a sdělovací technika
(dále jen „nabyvatel“)

Čl. 1

Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):

   ☐ disertační práce
   ☐ diplomová práce
   ☒ bakalářská práce
   ☐ jiná práce, jejíž druh je specifikován jako ..... (dále jen VŠKP nebo dílo)

   Název VŠKP: Jádro obvodu FPGA pro zobrazení dat na monitoru prostřednictvím portu VGA
   Vedoucí/ školitel VŠKP: Ing. Michal Kubíček
   Ústav: Ústav radioelektroniky
   Datum obhajoby VŠKP: _____

   VŠKP odevzdal autor nabyvateli*:

   ☒ v tištěné formě – počet exemplářů: 2
   ☒ v elektronické formě – počet exemplářů: 2

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.

3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.

4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

* hodící se zaškrtněte

Článek 2

Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.

2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.

3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti

   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)

4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3

Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.

2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.

3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.

4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: 2. června 2008

…….. ……

Nabyvatel Autor

ABSTRAKT

Cílem tohoto projektu je předvést studii možného řešení způsobu ovládání a využití počítačového monitoru připojeného k portu typu VGA řízeného standardizovanými řídícími signály generovanými obvodem typu FPGA. Jedná se o jádro hradlového pole, které je poté možné použít jako součást složitějšího designu a využít jej například pro komfortnější uživatelské rozhraní.

Projekt obsahuje řešení základní části obvodu generující standardní řídící signály a zobrazující text zadaný v ASCII kódu prostřednictvím sériového portu.

KLÍČOVÁ SLOVA

FPGA, Xilinx, VGA, monitor, FPGA VGA modul, FPGA sériový port

ABSTRACT

The aim of this project is to perform the study of a driver for controlling computer monitor using VGA port. The driver is based on FPGA which is used to generate VGA signals. The main purpose of the project is to design a hardware core for gate array which can be used as part of some complex FPGA design to provide a comfortable user interface.

The project describes the main part of the VGA driver – module for generating control signals and module for displaying text information that is sent from a PC via serial port interface.

KEYWORDS

FPGA, Xilinx, VGA, monitor, FPGA VGA module, FPGA Serial port interface

BIBLIOGRAFICKÁ CITACE

PIŠL, A. *Jádro obvodu FPGA pro zobrazení dat na monitoru prostřednictvím portu VGA*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. Vedoucí bakalářské práce Ing. Michal Kubíček.

Prohlášení

Prohlašuji, že svůj semestrální projekt na téma Jádro obvodu FPGA pro zobrazení dat na monitoru prostřednictvím portu VGA jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské dále prohlašuji, že v souvislosti s vytvořením tohoto projektu jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 2. června 2008 .....

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Michalu Kubíčkovi za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne 2. června 2008 .....

podpis autora

OBSAH

SEZNAM ZKRATEK

ASCII	American Standard Code for Information Interchange
ASIC	Application-Specific Integrated Circuit
CLB	Configurable Logic Block
CPLD	Complex Programmable Logic Device
DCM	Digital Clock Manager
DDR	Double Data Rate
EEPROM	Electrically Erasable and Programmable Read Only Memory
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IBM	International Business Machines Corporation
IEEE	Institute of Electrical and Electronics Engineers
ISE	Integrated Simulation Environment
JTAG	Joint Test Action Group
LED	Light-Emitting Diode
PC	Personal Computer
PLD	Programmable Logic Device
RAM	Random Access Memory
ROM	Read Only Memory
SDRAM	Synchronous Dynamic Random Access Memory
SRAM	Static Random Access Memory
SPLD	Simple Programmable Logic Device
UART	Universal Asynchronous Receiver/Transmitter
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuits

1. ÚVOD

Největší část člověkem vnímaných informací představuje informace obrazová. Pomocí zraku vnímáme barvy, dokážeme se orientovat v prostoru nebo rozpoznávat mimoslovní komunikaci. Obrazové vjemy jsou také nejčastěji používané v různých rozhraních člověk – stroj. Již první počítače zahrnovaly nějakou formu zobrazovací jednotky.

Úkolem této práce je představení možné koncepce využití zobrazovací jednotky standardu VGA propojené s logickým hradlovým polem FPGA k zobrazení počítačových dat. Obvody FPGA nacházejí uplatnění v široké řadě aplikací, především tam, kde se očekává vysoký výkon zpracování informací při jejich velkém objemu. K těmto oblastem patří především zpracování multimediálních signálů, jejichž nejvýznamnější složkou je právě obraz. Je-li v zařízení již obsažen některý z obvodů hradlových polí, je možné díky popisovanému modulu jednoduše doplnit zobrazovací subsystém a tím dosáhnout vyššího uživatelského komfortu.

Jako cílové zařízení byl zvolen obvod řady Spartan-3 [4] firmy Xilinx umístěný ve vývojovém kitu [5] používaném v laboratorní výuce. Jednoduchý VGA řadič, dostupný studentům ve formě IP jádra, umožní studentům zobrazovat data na běžném monitoru. Bude jim tak názorně předvedena další možnost použití obvodů FPGA, jenž je v praxi často využívána.

Práce navazuje na předchozí teoretickou studii BB1E a praktickou část BB2E, kterou dále rozšiřuje a doplňuje dalšími moduly nutnými k vytvoření prakticky použitelného modulu, který bude využit pro podporu výuky programování hradlových polí.

2. POUŽITÉ TECHNOLOGIE

Úlohou modulu je načíst data v definovaném tvaru a zobrazit je uživateli na počítačovém monitoru. Aby bylo možné tento úkol splnit, je třeba dodržet určité standardy a normy dnes běžně používané.

Jako zdroj dat vhodných pro zobrazení jsem zvolil text kódovaný podle kódové stránky Windows-1250, která je pravděpodobně nejrozšířenější znakovou sadou mezi běžnými uživateli. Vstupem jsou osmibitová slova (byty), přenášená z obslužného PC pomocí programu Hyperterminál přes standardní sériový port. Toto rozhraní je sice již běžně nahrazováno rozhraním USB, ale moje volba byla limitována možnostmi dostupného vývojového kitu.

Jako výstup obrazové informace slouží standardní počítačový monitor. Výstupní signál modulu odpovídá normě VGA, kterou jsou, díky zpětné kompatibilitě, schopny zpracovat všechny monitory, se kterými se lze běžně setkat.

Modul, včetně všech nástavbových modulů nutných pro demonstraci jeho funkce, je určen pro implementaci do obvodu FPGA Spartan-3 firmy Xilinx, (v obvodu XC3S200 je jeho implementací využito necelých 50% jeho strukturálních prvků). Vývoj probíhal v prostředí ISE WebPack téže firmy, design je však možné implementovat i do libovolného jiného FPGA (i jiného výrobce), které bude mít dostatečné množství použitelných logických bloků a možnost syntézy potřebného hodinového signálu.

2.1. Sériový port (RS-232)

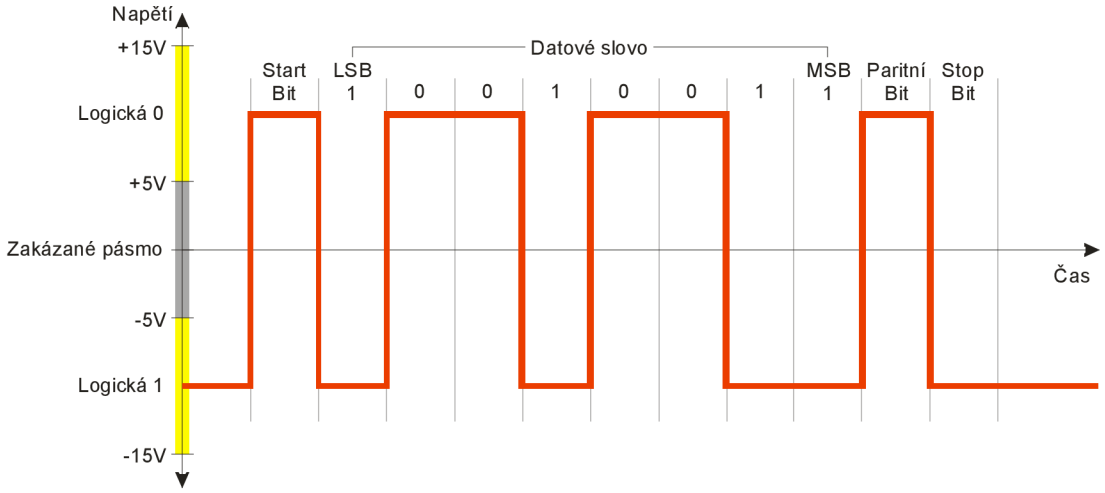
Standard RS-232 a hlavně jeho poslední varianta RS-232C patřil k nejpoužívanějším komunikačním rozhraním osobních počítačů. Umožňuje obousměrný přenos dat mezi dvěma zařízeními sériovým způsobem za použití minimálně tří vodičů. Sériová komunikace znamená, že data jsou přenášena po jednom bitu vždy za sebou.

Rozhraní je již v současné době nahrazeno výkonnějším USB, nicméně v průmyslové praxi se stále používá (především jeho modifikace RS-422 a RS-485). Především kvůli své jednoduchosti a možnosti dosáhnout velkých vzdálenosti mezi zařízeními. Specifikace sice uvádějí maximální vzdálenost 15m, ale použitím kvalitních kabelů je možné tento dosah zvětšit až na několik set metrů (ovšem při snížené přenosové rychlosti).

Přenos probíhá asynchronně s pevně stanovenou přenosovou rychlostí. Synchronizace je zajištěna sestupnou hranou startovacího impulzu. Pořadí přenosu datových bitů je od nejméně významného (LSB) po nejvíce významný (MSB), jejichž počet je volitelný (4 – 9, obvykle se používá 8 bitů) s možností doplnění detekce chyb pomocí paritního bitu. Každé sekvenci dat předchází startovací bit, kterým se linka přepne z klidové úrovně log. 1 na úroveň log. 0 na dobu jednoho symbolu, a tím dá druhému zařízení na vědomí, že má očekávat příchozí zprávu. Po odeslání dat a případného paritního bitu následuje jeden až dva stopbity, kterým se přenos ukončí.

Logický stav je reprezentován pomocí dvou úrovní napětí; –12V pro vyjádření logické jedničky a +12V pro vyjádření logické nuly. Toto jdou nejčastěji používané hodnoty. Rozhraní však dovoluje rozptyl vysílaných úrovní –5V až –15V pro log. 1 a +5 až +15V pro log. 0. Základní signální vodiče jsou tři: RXD – pro příjem dat, TXD – pro vysílání a společný zemní vodič GND. Rozhraní definuje ještě další signály určené pro řízení toku dat (například při komunikaci s analogovými modemy), nezavádí však nutnost jejich použití. Lze jich však použít například pro napájení připojeného zařízení, které však musí mít velmi malé

nároky na spotřebu, neboť rozhraní disponuje ochranou proti zkratu a dovolí maximální výstupní proud cca 20 mA. V popisovaném modulu nejsou tyto signály využity.



Obrázek 1: Průběh vysílaného signálu RS-232

Maximální přenosová rychlost dnešních sériových portů je 128000 Baudů. Do této přenosové rychlosti se ovšem počítají i bity nutné ke správné funkci rozhraní (startovací bit, ukončovací bit, paritní bit), tedy nikoliv samotná uživatelská data.

2.2. Standard VGA

Termínem VGA (Video Graphics Array) je označován jak standard pro zobrazování informací pomocí počítačové obrazovky, připojované pomocí 15 pinového konektoru, tak i rozlišení s maticí 640×480 bodů. S obojím přišla na trh firma IBM v roce 1987 poprvé ve svém modelu IBM PC PS/2.

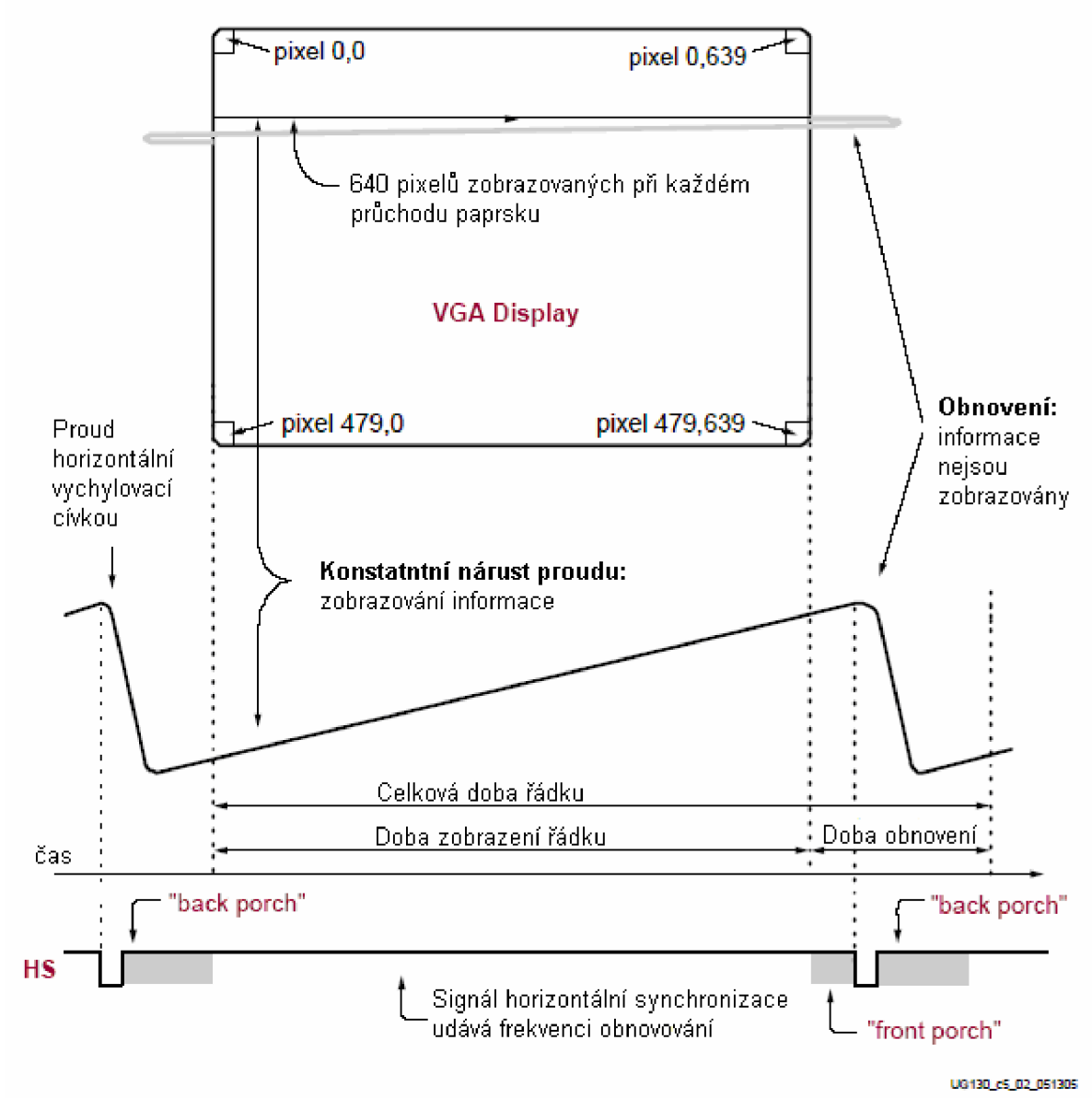
Standard VGA byl posledním grafickým standardem firmy IBM, který byl přejímán širokým spektrem výrobců hardware. V dnešní době je nepsaným pravidlem, že veškerý hardware, který má co do činění s počítačovým zobrazováním, podporuje tento standard bez vynucené přítomnosti specifických ovladačů. Lze to pozorovat například při spouštění systém Windows, kdy úvodní obrazovka se zobrazuje právě ve VGA režimu ještě před tím, než jsou zavedeny ovladače grafického adaptéru.

Mezi hlavní charakteristiky standardu patří:

- maximum 720 sloupců a 480 řádků
- Snímková frekvence až 70 Hz
- 256 KB paměti
- 16 nebo 256 barev
- charakteristická impedance 75Ω

Průběh signálů lze rozdělit na několik fází. Na obrázku 2 jsou zobrazeny průběhy horizontálního vychylovacího a synchronizačního signálu. Signál horizontální synchronizace je tvořen takto: Nejdříve se provede synchronizace pomocí krátkého impulsu, poté následuje doba potřebná pro zpětný běh paprsku k přesunu na začátek následujícího řádku (back porch). Doba určená k zobrazení informace je následována krátkým čekáním na příchod dalšího synchronizačního pulsu (front porch) a celý děj se opakuje. Doby Front Porch, impuls a Back

Porch se souhrnně označují jako doba obnovení paprsku (retrace time). Stejně pracuje i vertikální synchronizace, jen s tím rozdílem, že vše se děje pomaleji, protože mezi dvěma vertikálními impulsy se musí vykreslit obraz na celou obrazovku. Horizontální a vertikální signál mohou mít různou polaritu. Podle polarity a periody impulsu potom rozlišuje vnitřní elektronika monitoru jednotlivé režimy zobrazení.



Obrázek 2: Časování VGA monitoru (převzato z [5])

Bohužel standardizované kmitočty a doby nejsou dodržovány obecně. Monitor se tak musí vyrovnat s vysokým rozptylem časových parametrů. Na druhou stranu je to výhoda, neboť není potřeba přesný taktovací kmitočet (který činí 25,175MHz), ale postačí i taktování 25 MHz. Takový signál je výrazně jednodušší syntetizovat, pokud máme k dispozici jeden krystal například 50 MHz, ze kterého budíme i jiné části zapojení. Tím můžeme ušetřit na jednotkách pro syntézu hodinového kmitočtu.

	Horizontální	Vertikální
Front Porch	8 pixelů	2 řádky
Šířka impulsu	96 pixelů	2 řádky
Back Porch	40 pixelů	25 řádků
Okraj obrazu	8+8 pixelů	8+8 řádků
Viditelný obraz	640 pixelů	480 řádků
Celkem	800 pixelů	525 řádků

Taktovací kmitočet: 25,175 MHz
Řádkový kmitočet: 31,469 kHz
Obrazový kmitočet: 59,94 Hz
Polarita HS: negativní
Polarita VS: negativní

Tabulka 1: Časování v režimu 640×480×60Hz

K dalším používaným režimům obrazu můžeme zařadit textový režim 640x400 70 Hz, používaný v DOS a při startu PC a nebo 640x350 70 Hz používaný kdysi jako kompatibilní s EGA. Vyšších rozlišení a obnovovacích frekvencí lze dosáhnout použitím novějších standardů jako SVGA, XGA, apod.

V použitém kitu je k dispozici pouze 3bitově zapojený převodník pro datový barevný signál (1 bit pro každou barvu R, G, B). Z toho vyplývá omezení počtu zobrazovaných barev na 8. Přehled použitelných barev je v tabulce 2.

Číslo barvy	R	G	B	Barva
0	0	0	0	černá
1	1	0	0	červená
2	0	1	0	zelená
3	1	1	0	žlutá
4	0	0	1	modrá
5	1	0	1	purpurová
6	0	1	1	azurová
7	1	1	1	bílá

Tabulka 2: Zobrazované barvy

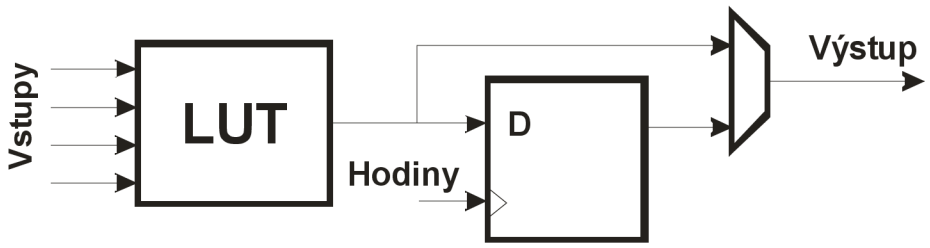
2.3. Obvody FPGA

Obvody FPGA (Field Programmable Gate Array; polem programovatelné hradlové pole) patří do rodiny takzvaných PLD (Programmable Logic Devices), tedy programovatelných logických obvodů. Jejich hlavní výhodou je to, že proti použití mikroprocesorů plnících stejnou funkci mohou být výrazně rychlejší. Použití mikroprocesoru v zapojení je ale výrazně levnější. Je proto na vývojovém pracovníkovi, aby zvolil nejefektivnější variantu. Ostatně i mikroprocesor lze implementovat v dostatečně dimenzovaném PLD.

Nejjednodušší obvody PLD se označují SPLD (Simple PLD). Byly původně vyvinuty pro zjednodušení návrhu plošných spojů a úsporu místa na desce. Byly schopny nahradit i několik desítek logických členů a tím ušetřit náklady na výrobu zařízení. Pokud sdružíme více obvodů SPLD do sebe a umožníme jejich vzájemné propojení, vznikne CPLD (Complex PLD). FPGA mají proti CPLD odlišnou strukturu. Jejich architektura (tzv. Fine-Grain) se skládá z malých konfigurovatelných logických bloků (CLB) a struktury pro jejich vzájemné propojení. Každý logický blok obsahuje tabulku funkcí typicky čtyř vstupních signálů, která definuje jeden výstupní signál, a hradlo řízené hodinovým signálem. Novější hradlová pole mohou obsahovat výkonnější bloky CLB obsahující tabulku pro šest vstupů a dva výstupy. Hodinový signál se v obvodu rozvádí pomocí několika vyhrazených propojovacích sítí. Jeho

úprava (změna frekvence či fáze, odstranění fázového šumu) je možná pomocí speciálních obvodů zahrnutých do struktury FPGA.

Obvody CPLD jsou složeny z menšího počtu složitějších buňek. Každá buňka obsahuje více vstupů a výstupů, které mohou být pomocí programovatelné propojovací struktury připojeny na vstupní a výstupní piny obvodu nebo na vstupy a výstupy dalších buněk. CLPD jsou schopny realizovat poměrně komplexní logické funkce a díky velmi malému odběru jsou často používány v aplikacích citlivých na odběr ze zdroje napájení.



Obrázek 3: Struktura CLB

FPGA mohou být vybaveny dalšími speciálními bloky jako RAM, specializované rychlé násobičky, obvody pro syntézu hodinových signálů, apod. Tyto obvody už dokážou nahradit i stovky tisíc původních logických členů (obvodů řad 74 nebo 4000). Konfigurace celého hradlového pole je nejčastěji uložena ve vnější paměti EEPROM či FLASH a po startu se musí do pole nahrát. Ovšem existují i obvody s implementovanou konfigurační pamětí přímo na čipu, nebo dokonce obvody pevně naprogramované. Je také možné do konfigurační paměti nahrát více možných konfigurací a za chodu mezi nimi přepínat. Obvody FPGA jsou také vybaveny rozhraním pro programování a ladění přímo v zapojení (JTAG).

K programování všech obvodů PLD se používá programovací jazyk typu HDL (Hardware Description Language). I když označení programovací je zde poněkud zavádějící. Jedná se spíše o popis vlastností jednotlivých dílčích prvků obvodu a způsobu jejich propojení. V zásadě se dnes používají tři jazyky. Prvním z nich je jazyk ABEL, který se používá převážně pro menší obvody SPLD a CPLD. Dnes je už ale pomalu na ústupu. Dalšími jsou jazyky VHDL a Verilog, které se používají pro návrh obvodů FPGA. V poslední době je citelný trend přiblížit jazyky HDL co nejvíce běžně používanému jazyku C (jazyky System Verilog a System C). Kód v C je mnohem efektivnější, ale jeho syntéza je problematická a prozatím je používán především pro simulaci, kterou dokáže výrazně zrychlit (tisíckrát i více).

2.3.1. Spartan-3

Obvody typu Spartan-3 jsou FPGA vyráběné firmou Xilinx. Navazují na předchozí řadu Spartan IIE a představují levný typ hradlových polí určený pro všeobecné použití ve spotřební elektronice. Řadu IIE ve všech směrech rozšiřuje a přebírá některé funkce i z vyspělé řady Virtex II.

Přímo na čipu obsahuje blokovou paměť typu RAM, 4 obvody pro syntézu hodinových signálů, dedikované 18bitové násobičky a JTAG rozhraní. Dále podporuje připojení pamětí DDR SDRAM až do frekvence 333 MHz a celkem 26 různých standardů logických úrovní na I/O pinech pro komunikaci s externími obvody. Navíc je plně podporován v návrhovém systému Xilinx ISE, který jsem zvolil pro realizaci celé práce. Podrobné parametry použitého obvodu XC3S200FT256 lze nalézt v [4].

Cílový obvod je osazen ve vývojovém kitu vyvinutém rovněž firmou Xilinx. Jedná se o Spartan-3 Starter Kit Board [5]. Kromě již zmíněného FPGA obsahuje také 2 Mb paměti

typu Flash, 1 MB SRAM s přístupovou dobou 10ns sestavenou ze dvou čipů 256k×16, 3bitově zapojený VGA port, sériový port s převodníkem, PS/2 port pro myš nebo klávesnici, 4místný sedmisegmentový displej, 8 přepínačů a diod LED, 4 mžiková tlačítka a JTAG port pro programování a ladění aplikace.

Z prostředků implementovaných v kitu využiji v projektu pouze samotné FPGA, VGA port, paměť a přepínače případně tlačítka.

2.4. VHDL a Xilinx ISE

VHDL je programovací jazyk určený pro popis hardware. Je použitelný jak pro návrh, tak pro simulaci digitálních integrovaných obvodů, programovatelných hradlových polí (CPLD, FPGA) nebo zákaznických obvodů ASIC. VHDL je standardizován IEEE od roku 1987 s revizí v roce 1993. Po revizi je možné jej používat i pro návrh analogových obvodů, přestože plně automatická syntéza analogových obvodů je stále nevyřešeným problémem.

Programovací jazyk Verilog má stejné určení jako jazyk VHDL. Narozdíl od VHDL však není přísně typový, je tedy možné provést operaci s operandy různého typu bez nutnosti jejich přetypování. O tom, zda bude nový design programován v jazyku VHDL či Verilog, v praxi obvykle rozhoduje zadavatel projektu (oba jazyky jsou si velmi podobné a přechod z jednoho na druhý je velmi snadný). Jelikož se na Ústavu Radioelektroniky používá jazyk VHDL, použil jsem jej ve svojí práci pro popis řadiče VGA.

Výhodou je možnost využití VHDL k popisu spolupracujících systémů sestavených z mnoha částí pracujících v jednom okamžiku. Na rozdíl od „klasických“ programovacích jazyku jako například C, Delphi nebo Java, které se vyznačují tím, že vykonávání jejich příkazů se děje pouze sekvenčně, je možné v jazyce VHDL popsat i paralelní struktury a toky dat.

Konfigurace obvodů FPGA je komplexní úkol, který není možné zvládnout bez specializovaných nástrojů. Proto výrobci FPGA dávají k dispozici integrovaná vývojová prostředí, která tuto práci významně zjednodušují, a tím umožňují větší rozšíření svých obvodů. Firma Xilinx dává na svých webových stránkách k dispozici vývojové prostředí Xilinx ISE WebPack, které bylo v době začátku práce na projektu dostupné ve verzi 9.2. Toto prostředí zahrnuje nástroje pro tvorbu, kontrolu, simulaci a implementaci návrhu v jazyce VHDL nebo Verilog. Právě proto jsem jej použil pro svou práci.

3. POPIS MODULU

Modul sestává z několika částí a jeho funkci lze rozdělit do několika bloků. Modul příjme přes sériový port data, která představují kód znaku, který se má zobrazit na obrazovce, v kódové stránce Windows 1250. Přijatá data jsou uložena do paměti na příslušnou adresu, která je dána aktuální pozicí kurzoru. Další část modulu zajistí jejich správné zobrazení na VGA monitoru v rastru 80×30 znaků (formát jednoho znaku je 8×16 obrazových bodů). Pro zobrazení je tedy využit grafický VGA režim. Druhou funkcí modulu je generování testovacího signálu, jehož vzor je pevně dán. Mezi textovým a grafickým režimem je nutno přepínat, nelze je provozovat zároveň.

VGADriver má za úkol řízení monitoru připojeného pomocí VGA portu. Jeho hlavní funkcí je generování synchronizačních signálů pro monitor a generování adresy právě zobrazovaného bodu pro použití v dalších blocích. Pouze on má přístup k výstupnímu VGA portu.

CharGenerator je blok starající se o dodání správných dat pro zobrazení bloku *VGADriver*. On sám je složen z více modulů, které budou dále popsány.

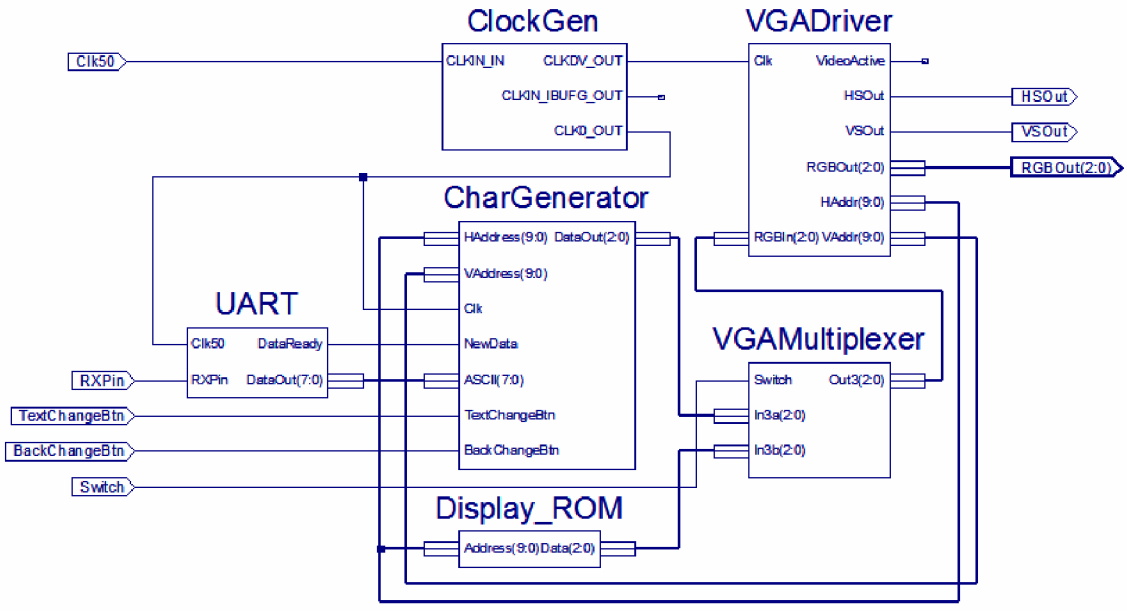
Blok s názvem *UART* realizuje funkci sériového portu (serializace/deserializace dat, časování, generování impulzů). Přijatá data ve formě ASCII znaků předává *CharGeneratoru*.

VGAMultiplexer je pouze přepínač vstupního signálu pro *VGADriver* mezi bloky *CharGenerator* a *Display_ROM*.

Display_ROM obsahuje jeden řádek obrazu tvořeného osmi barevnými pruhy

Poslední částí je *ClockGen*. V tomto případě je realizován blokem DCM (Digital Clock Manager), jehož vlastnosti je možné jednoduše nastavovat pomocí průvodce přímo v návrhovém prostředí ISE.

Zdrojové kódy všech bloků, které nejsou tvořeny pouze schématem nebo průvodcem pro vytvoření, jsou uvedeny v příloze. Nejsou uvedeny bloky *CharROM* a *Display_ROM*. Jejich výpis by byl neúměrně dlouhý. Výpis bloku *CharROM* je nahrazen mapou znaků, které tento blok definuje.



Obrázek 4: Celkové schéma modulu

3.1. VGADriver

Tento blok má na starosti generování správných horizontálních a vertikálních synchronizačních impulsů, adresy právě zobrazovaného bodu a pomocného signálu „VideoActive“ sloužícího pro signalizaci právě probíhaného zobrazení. Jeho vstupy tvoří hodinový signál 25 MHz a data právě zobrazovaného bodu.

Celý blok je tvořen dvěma čítači, jedním horizontálním a druhým vertikálním. Tyto čítače jsou doplněny rozhodovací logikou pro správné generování synchronizačních signálů a oddělení adresy řádku a sloupce.

Horizontální čítač je buzen vstupním hodinovým signálem 25 MHz. Dle normy by bylo optimální použít 25,175 MHz, ale přiklonil jsem se k řešení použít co nejméně zdrojů FPGA, a proto jsem použil pouze jednoduché dělení signálu 50 MHz z krystalu na kitu. I tento nepřesný signál je běžnými monitory v pořádku zpracován. Vertikální čítač je inkrementován vždy, když dojde k vynulování horizontálního čítače.

Rozhodovací logika na základě stavu jednotlivých čítačů přepíná synchronizační signály a spíná výstup datových vodičů. Některé monitory v době neaktivního zobrazení testují výchozí úroveň signálů. Proto je dobré v neaktivní (zatemňovací) části cyklu držet datové vodiče v neaktivní úrovni. V opačném případě může dojít k chybnému zobrazení.

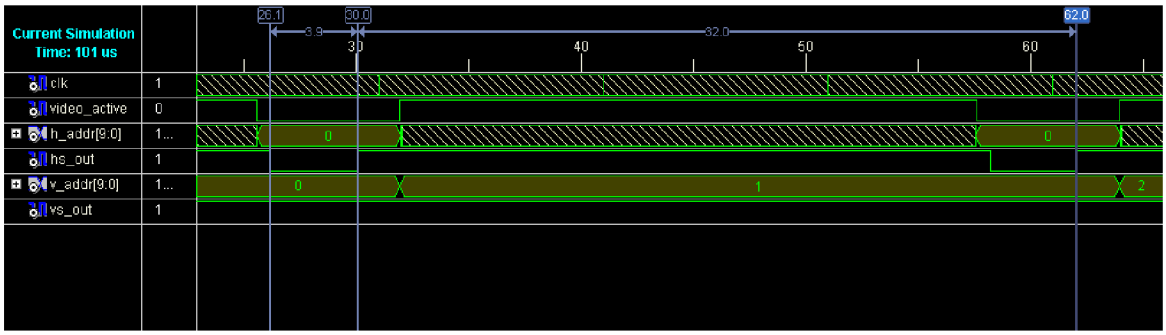
Časové relace jsou dány stavy jednotlivých čítačů. Při určování stavů rozhodování jsem vycházel z tabulky 1.

Horizontální a vertikální čítač jsou tvořeny proměnnými *H_count* a *V_count*. Hodnota *H_count* je inkrementována s každou nástupnou hranou hodinového impulsu. *V_count* je zvyšován, pokud dojde k přetečení *H_count*. K tomu dojde při překročení hodnoty 799, viz tabulka 1. Čítače čítají od nuly a v pásmu hodnot odpovídajícím rozlišení, tedy 0 ÷ 639 pro horizontální a 0 ÷ 479 pro vertikální, se jejich hodnota přímo převádí na výstup do adresních sběrnic. Obě adresy jsou pro zjednodušení 10 bitové i přes to, že není využito celých 10 bitů pro adresování. Je to příprava na případné použití vyššího rozlišení. Vyšších hodnot nabývají čítače pouze v neaktivním režimu a adresní sběrnice jsou udržovány na nulové hodnotě, aby při přístupu do paměti nedošlo k nesprávnému adresování. Také výstupní datové vodiče jsou udržovány v neaktivním stavu na nulové úrovni, a tím umožňují detekci výchozí úrovně monitorem.

Na následujících obrázcích jsou zobrazeny výsledky simulace bloku *VGADriver*. Všechny simulace byly provedeny v ISE simulátoru. Signály *RGB_in* a *RGB_out* nejsou v simulaci zahrnuty, neboť na této úrovni simulace není možné ověřit správnost jejich funkce.

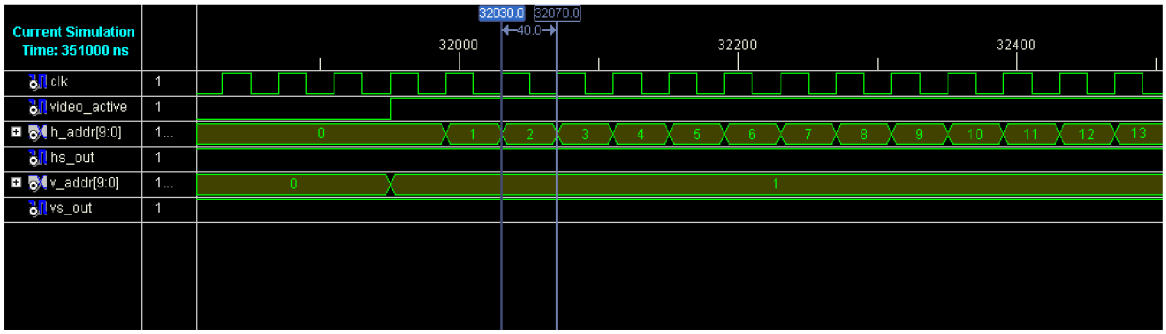
Význam jednotlivých signálů v simulaci:

- clk – hodinový signál 25MHz
- video_active – příznak aktivního zobrazování snímku
- h_addr[9:0] – 10bitová adresa sloupce převedená na dekadické číslo
- hs_out – výstupní signál horizontální synchronizace
- v_addr[9:0] – 10bitová adresa řádku převedená na dekadické číslo
- vs_out – výstupní signál vertikální synchronizace

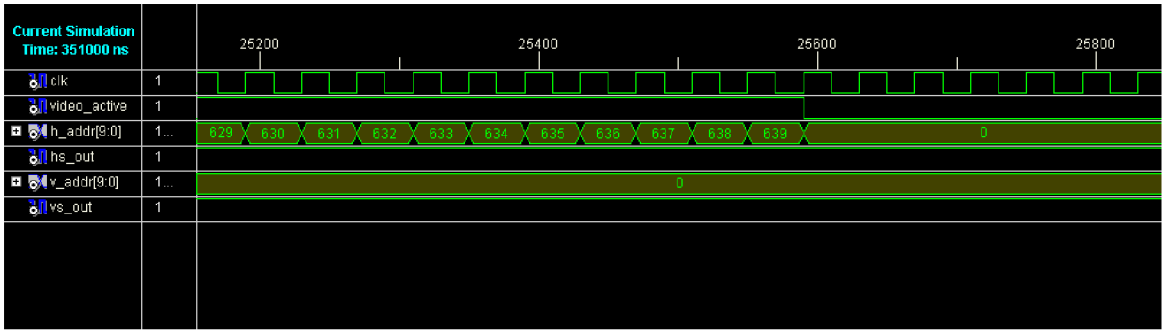


Obrázek 5: Simulace jednoho řádku obrazu

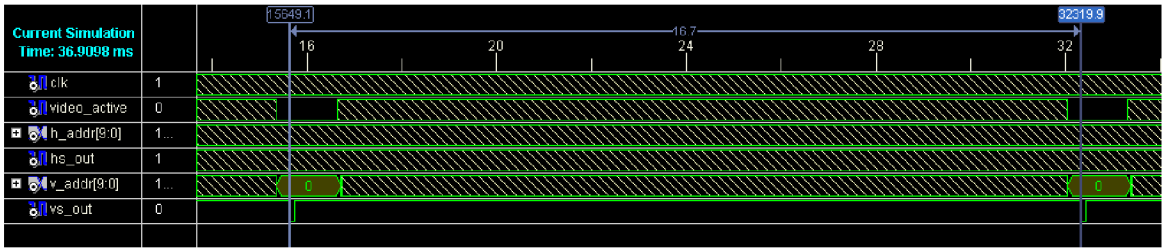
Z obrázku 5 je patrné, že doba trvání jednoho řádku je 32 µs. To odpovídá horizontální frekvenci 31,25 kHz. Délka horizontálního synchronizačního impulsu je 3,9 µs.



Obrázek 6: Simulace začátku řádku



Obrázek 7: Simulace konce řádku obrazu



Obrázek 8: Simulace celého snímku obrazu

Na obrázku 8 je vidět doba trvání jednoho snímku 16,7 ms. Z toho vyplývá, že snímková obnovovací frekvence je 59,88 Hz.

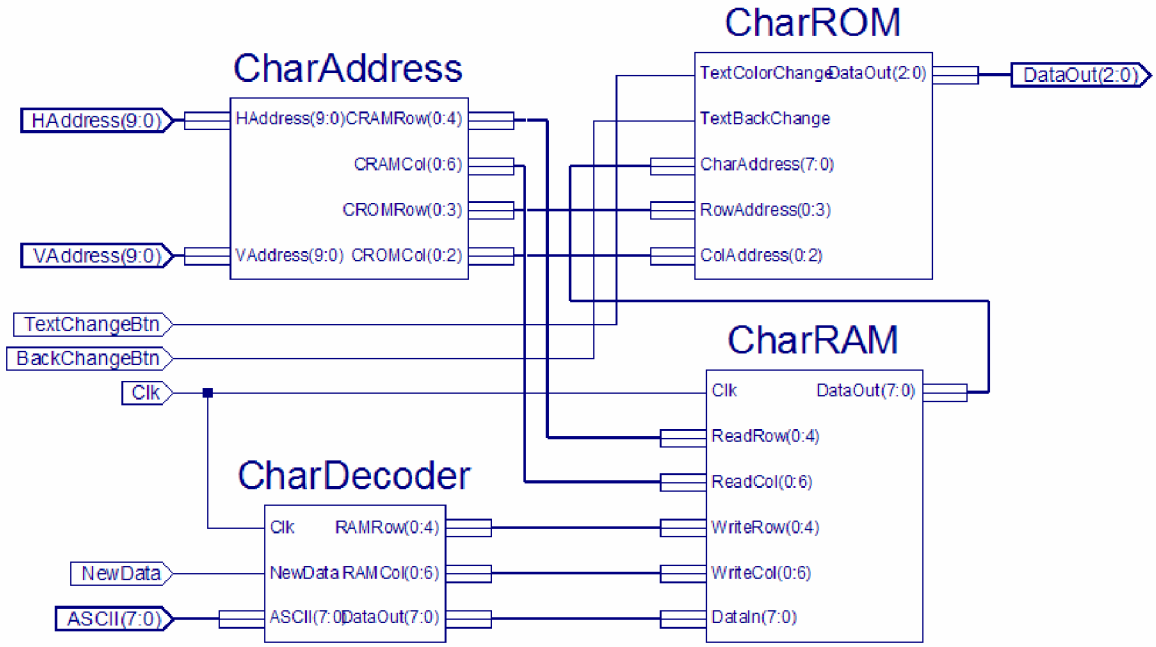
Šrafované průběhy v simulaci vyjadřují změny natolik rychlé, že by je nebylo možno při aktuálním nastavení časové osy korektně vykreslit.

3.2. CharGenerator

Jde o nejsložitější blok celého modulu. Jeho struktura je patrná z obrázku 9. Hlavní částí je dvouportová paměť RAM, obsažená v podbloku *CharRAM*, která zároveň odděluje dvě funkce bloku. První funkcí je přijetí korektních dat z ASCII vstupu a jejich uložení na příslušné místo v paměti. Druhá funkce se stará o jejich správné zobrazení na základě informace o právě zobrazovaném bodu obrazu.

Blok je též schopen rozpoznat některé funkční klávesy. Jedná se o klávesu Enter, která zajistí přechod na nový řádek, klávesu Backspace, která maže poslední zadaný znak, a klávesu Esc sloužící k návratu na první znak prvního řádku.

Další funkcí bloku je možnost výběru kterékoliv z osmi barev pro zobrazení textu stejně jako výběr barvy pozadí. Výběr se provádí přivedením nástupné hrany na příslušné vstupy pomocí tlačítek na vývojové desce.



Obrázek 9: Schéma bloku CharGenerator

3.2.1. CharAddress

Tento modul má za úkol správně rozdělit adresy VAddress a HAddress na část určující pozici znaku v obraze a část určující pozici zobrazovaného bodu ve znaku. Vzhledem k použitému rastru znaků jde pouze o oddělení nejnižších 4 (v případě sloupcového indexu) případně 3 bitů (pro řádkový index) vstupních adres předávaných z bloku *VGADriver*.

3.2.2. CharROM

Tato část bloku obsahuje kompletní mapu znaků (byla vytvořena nová sada znaků 8x16 bodů), ze které je vybíráno pomocí tří signálů. Signál *CharAddress* vybírá požadovaný

znak. Jeho částí má být právě zobrazovaný bod, který je dále určen pomocí signálů *RowAddress* a *ColAddress*.

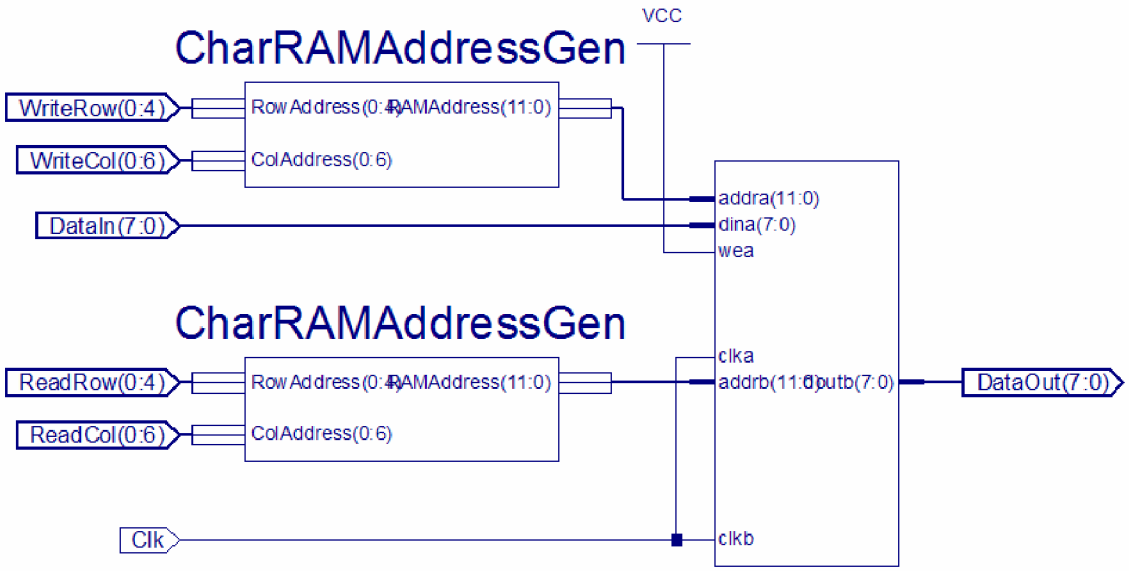
Právě tato část má vliv na to, že je možno použít kódování znaků používané běžně ve Windows. Jeho úpravou by bylo možné použít i jiné kódové stránky používané v PC, ovšem s tím omezením, že celý blok je schopen zpracovat kódování pouze 8 bitové. To však vylučuje použití znaků Unicode, které vyžaduje 16 bitů.

Další funkcí je také možnost ovlivnění barvy textu i pozadí. Toto nastavení je pak platné pro celou obrazovku. Základní nastavení (zelený text na černém pozadí) je možno změnit pomocí tlačítek na vývojové desce a to pro pozadí i pro text zvlášť. Změna je provedena cyklicky (vystřídají se všechny barvy, a pak se cyklus opakuje) při detekci nástupné hrany signálů *TextColorChange* nebo *TextBackChange*.

U tohoto modulu jsem byl nucen upravit původní myšlenku třírozměrného pole, které v použité verzi vývojového prostředí nelze realizovat. Při programování syntezéru nebylo na tuto možnost myšleno. Podle údajů výrobce bude tato možnost dostupná v příští verzi. Postup výběru příslušného bodu je tedy rozdělen na výběr nejdříve znaku a řádku (dvourozměrné pole), čímž je vybráno 8 bitů z matice, které se pak přesunou do pomocného registru, ze které je pak vybrán již pouze jeden požadovaný bit. Toto řešení, i když spotřebuje více funkčních bloků oproti přímému adresování třírozměrného pole, však nemá žádný vliv na výkonnost.

3.2.3. CharRAM

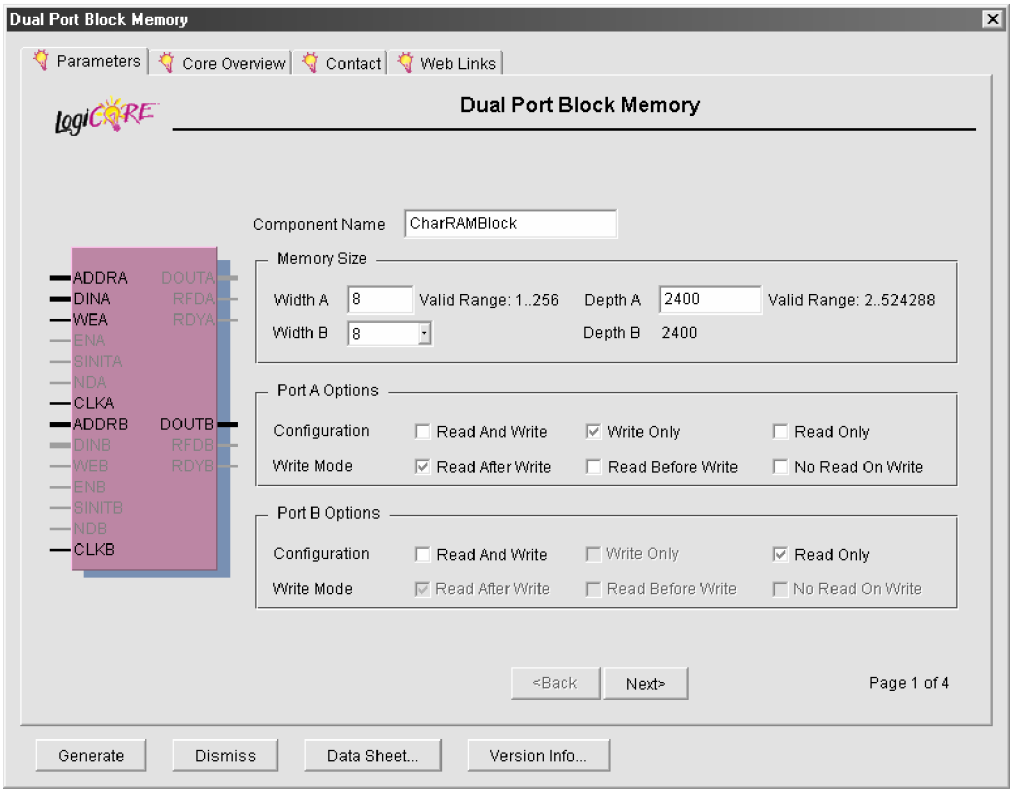
Jak již název naznačuje, tato část bloku je vybavena pamětí. Paměť je použita pro uložení informace o zobrazovaných znacích. Potřebnou velikost je možné jednoduše vypočítat. Máme k dispozici 30 řádků s 80 znaky v každém řádku. Při použití 8 bitů na každý znak nám vychází, že velikost použité paměti musí být nejméně 30×80×8 = 19200b. Tento požadavek je schopen uspokojit samotný cílový obvod [4], neboť má ve své struktuře k dispozici několik bloků RAM o celkové kapacitě 216 kb, které jsou schopny pracovat jako dvouportová paměť, což významně zjednodušuje možnost její implementace do modulu. Navíc díky průvodci zahrnutému ve vývojovém prostředí, je její začlenění rychlé a intuitivní.



Obrázek 10: Schéma bloku CharRAM

Paměť má oddělený port pro zápis a pro čtení. Je to z toho důvodu, že zápis a čtení probíhají nezávisle na sobě na různých adresách. Paměť je jediný společný prvek obou částí bloku. Pro správnou funkci jsou zde také přítomny převodníky adresy, které ze dvou adresních signálů pro řádek a pro sloupec vytvoří jeden signál pro přístup k paměti. Realizuje matematickou funkce *Řádek×80 + Sloupec*. Oba porty paměti mohou být řízeny zvláštními hodinovými signály. Tuto vlastnost ale v modulu nevyužívám.

V původním návrhu modulu jsem počítal s využitím modulů SRAM umístěných na vývojové desce. Díky ukládání pouze adresy znaků a jejich následným přímým generováním jsem tyto nároky snížil. Proto bylo možné tuto koncepci zjednodušit na stávající řešení, které má tu výhodu, že všechny potřebné zdroje jsou již přítomny v použitém obvodu [4].



Obrázek 11: Průvodce vytvořením paměti

3.2.4. CharDecoder

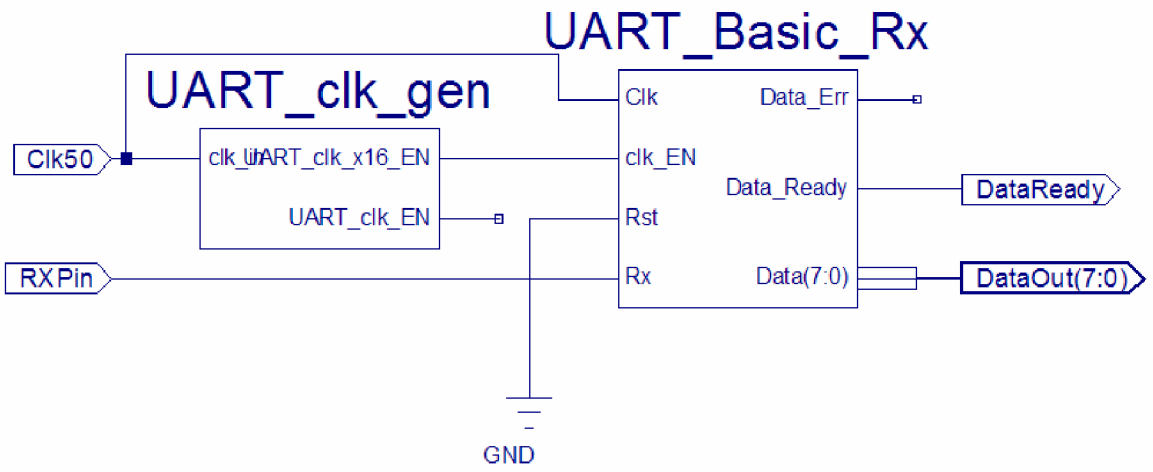
Poslední částí bloku *CharGenerator* je *CharDecoder*. Vstup této části tvoří datový 8 bitový signál obsahující kód právě přijatého znaku, který je zjišťován při detekci nástupné hrany na vstupu *NewData*. Pro detekci nového symbolu jsem byl nucen zavést pomocný signál *PrevState*. Pokud bych chtěl využít konstrukci „if NewData'event", byla by výsledkem chyba syntézy. Proces je totiž již taktován pomocí signálu *Clk*. U obvodu FPGA principiálně není možné taktovat blok více signály.

Při přijetí zobrazitelného znaku je tento přímo převeden na výstup *DataOut* spolu s ukazatelem pozice, který je tvořen signály *Row* a *Col*. Ukazatel je poté inkrementován a připraven pro zápis dalšího znaku na výstup.

Pokud není přijímán žádný nový znak, je zobrazen blikající kurzor, tak jak jej známe například z MS-DOS (na pozici následující za posledním přijatým znakem). Blikání obstarává čítač tvořený signálem *CursorBlink* inkrementovaný s každým hodinovým impulsem.

Při přijetí nezobrazitelného znaku se tento znak zahodí. Toto pravidlo má ale tři výjimky. Jsou jimi klávesa Enter, Esc a Backspace. Při přijetí klávesy Enter se řádkový index zvýší a sloupcový index se nastaví na začátek řádku. V případě že by se řádkový index měl zvýšit do nezobrazitelné oblasti, je přesunut na první řádek obrazu. Nový text potom přepisuje odshora text starý. Pro budoucí rozšiřování by bylo také možné implementovat funkci, kdy po stisku klávesy Enter se celý text odsune o řádek výše a kurzor se přesune na začátek posledního (téhož) řádku. Pokud by se současně zvětšila použitelná paměť v modulu CharRAM, bylo by možno zobrazit textu i několik stránek, mezi kterými by bylo možno přecházet. Klávesa Esc má za následek nové nastavení indexů na pozici prvního řádku a prvního sloupce. Přijetí Backspace má za následek dekrementaci řádkového indexu a tím posun kurzoru zpět. Původní znak díky tomu zmizí. Pokud by mělo dojít ke snížení indexu do záporných hodnot (mimo obraz), je kurzor přesunut na poslední pozici řádku předcházejícího.

3.3. UART



Obrázek 12: Schéma bloku UART

Jak je vidět ze schématu, blok je tvořen dvěma částmi. *UART_Basic_Rx* je vlastní přijímač sériového signálu podle specifikací RS-232. Druhá část *UART_clk_gen* slouží pro generování hodinového signálu odvozeného od hlavních hodin modulu. Pro případ, kdy je požadováno také vysílání dat je možné sestavu doplnit ještě blokem *UART_Basic_Tx* sloužícím právě tomuto účelu. Nicméně v mém modulu jsem využil pouze příjem.

Část bloku pro generování správného hodinového signálu pracuje následovně. Aby se dosáhlo správné rychlosti přenosu (v mém případě 9600 baudů) je zapotřebí vložit mezi hlavní hodiny modulu a samotný přijímač děličku hodin. Tato dělička ale nepracuje obvyklým způsobem, kdy na výstupu jsou přítomny „prodloužené“ impulsy. Generování hodin v bloku *UART_clk_gen* probíhá tím způsobem, že se počítá počet period hlavních hodin a při příhodu impulsu, jehož nástupná hrana se shoduje s nástupnou hranou plánovaného výstupního signálu, se tento jeden impuls nechá projít na výstup. Tento způsob, i když je implementačně poněkud náročnější, je z hlediska syntézy daleko bezpečnější. Je při něm totiž zachována koherence hodinového signálu. Tím taktování bloku je konzistentní se zbytkem modulu a je omezena možnost chyb způsobených špatným zfázováním hodin.

Blok, mi byl poskytnut jako hotové řešení vedoucím mé práce, které jsem pouze zakomponoval do modulu a které se ukázalo být plně funkční.

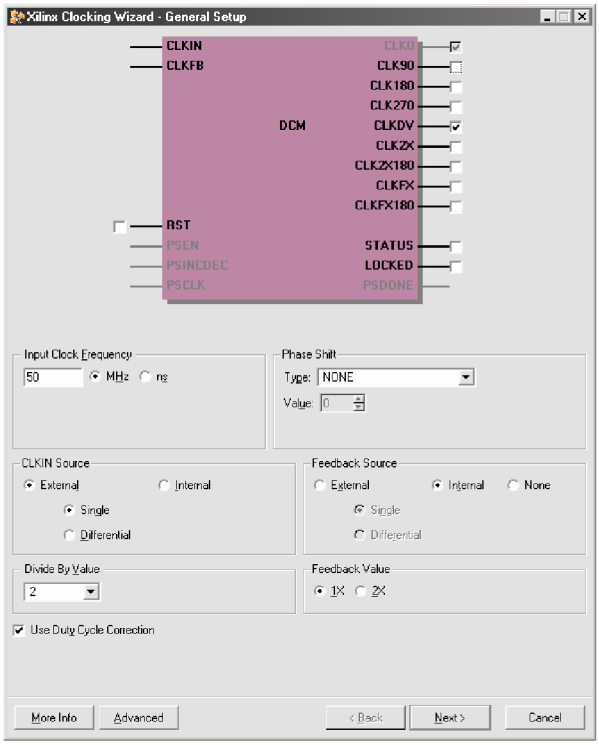
3.4. ClockGen

Nejjednodušší blok modulu. Jeho návrh spočíval pouze ve spuštění průvodce pro návrh obsaženého ve vývojovém prostředí. Jedná se o integrovaný obvod syntézy hodinového signálu (Digital Clock Manager), kterým je možné vstupní hodinový signál různým způsobem opravovat.

Obvod DCM má následující funkce:

- Násobení a dělení vstupního hodinového signálu
- Úprava signálu pro zajištění střídy 1:1
- Posun fáze vstupního signálu
- Vylepšení strmosti hran hodinového signálu

Všechny funkce je obvod schopen vykonávat najednou. Lze tedy získat i několik hodinových signálů s definovanou vzájemnou závislostí z jediného obvodu DCM.



Obrázek 13: Průvodce nastavením DCM

3.5. Display_ROM

Blok podobný bloku *CharROM*. I zde se nejedná o nic jiného než pevně definovanou paměť. Paměť obsahuje data jednoho řádku testovacího obrazce. Tento řádek se neustále opakuje, není proto nutné využívat pro adresování signálu *VAddr* z bloku *VGADriver*.

Obrazec je složen z osmi barevných pruhů, každý o šířce 80 bodů. Při případné změně testovacího obrazu je nutno vzít na vědomí, že se obraz každý řádek opakuje. Není proto možné vytvořit například obraz šachovnice apod.

3.6. VGAMultiplexer

Poslední blok modulu. Jeho funkcí není nic jiného, než podle úrovně na vstupu Switch poslat na výstup *Out3* data buď ze vstupu *In3a* nebo *In3b*. Tím se realizuje výběr funkce celého modulu. Na signál *Switch* je na vývojové desce připojen přepínač. Lze jej tedy libovolně uživatelsky ovládat.

Blok pouze přepíná vstupní signál do bloku *VGADriver*. Ostatní jím nejsou nijak ovlivněny. Z toho vyplývá, že při zobrazení testovacího signálu je stále možné posílat data přes sériový port a tato data jsou zpracována. Po přepnutí zpět do textového módu je potom zobrazen nový stav.

4. REALIZACE

Jak jsem již popsal, vývoj celého modulu probíhal na vývojovém kitu [5] společnosti Xilinx osazeném obvodem Spartan-3. Z jeho možností jsem využil pouze vlastní obvod Spartan-3, port VGA a port RS-232 společně s převodníkem úrovní a standardní 5V napájecí zdroj. V případě přímé realizace zapojení ve spojení s mým modulem jsou tedy potřeba pouze tyto komponenty.

Pro ověření funkce jsem používal digitální CRT monitor Samsung SyncMaster 550s. Pro vstup dat byl použit program Hyperterminál, který je standardní součástí OS Windows. Vývoj i testování probíhaly na mém notebooku Compaq Evo N620c s nainstalovaným Xilinx ISE Webpack 9.2i.

Spojení mezi obslužným PC a vývojovým kitem bylo provedeno pomocí sériové linky RS-232. Rychlost přenosu 9600 baudů. Byla nastavena sudá parita a jeden stop bit.

Návrh modulu byl rozložen do tří částí, podle zadání předmětů BB1E, BB2E a BBCE. První část BB1E obsahovala teoretickou studii možností poskytnutého obvodu a jeho vývojového kitu. V průběhu práce na bakalářském projektu BB2E jsem navrhl a úspěšně otestoval blok VGADriver, jehož úkolem je generování správných synchronizačních signálů pro monitor a pro další bloky modulu. Práce obsahovala též návrh dalšího postupu v rámci BBCE. Tento postup jsem se po další úvaze rozhodl pozměnit, a to tím způsobem, že jsem zredukoval množství dat, která jsou nutná k uchování v obrazové paměti. Tím jsem dosáhl toho, že v textovém módu bylo možné upustit od využití externích paměťových čipů a spolehnout se výhradně na vnitřní zdroje FPGA.

Pro implementaci modulu je potřeba 1424 LUT z 3840. To odpovídá využití obvodu XC3S200 z 37%. Z toho vyplývá, že obvod XC3S200 je nejmenší možný obvod Spartan-3, který lze pro implementaci použít. Pro srovnání uvádím v tabulce přehled využití pro jiné obvody firmy Xilnix.

Typ obvodu	Využito LUT	Počet Slice	Počet Slice Flip-Flop	Počet násobiček MULT18x18	Počet bloků BRAM	Počet využitých GCLK
Spartan–3 XC3S200	1424 (37%)	929 (48%)	181 (4%)	2 (16%)	2 (16%)	4 (50%)
Spartan–3 XC3S1000	1424 (9%)	929 (12%)	181 (1%)	2 (8%)	2 (8%)	4 (50%)
Spartan–3 XC3S1500	1424 (5%)	929 (6%)	181 (1%)	2 (6%)	2 (6%)	4 (50%)
Virtex-II Pro XC2VP2	1404 (49%)	918 (65%)	181 (6%)	2 (16%)	2 (16%)	4 (25%)
Virtex-II Pro XC2VP20	1404 (7%)	918 (9%)	181 (1%)	2 (2%)	2 (2%)	4 (25%)
Virtex-4 XC4VFX20	1382 (78%)	909 (10%)	170 (1%)	-	2 (2%)	4 (12%)
Virtex-5 XC5VSX50	653 (2%)	218 (2%)	184 (1%)	-	1 (0%)	4 (12%)

Tabulka 3: Využití při implementaci do jiných obvodů

Modul se podařilo úspěšně realizovat. Realizaci dokládají níže přiložené obrázky.

Obrázek 14: Přehled zobrazovaných barev textu

Obrázek 15: Zobrazení testovacího signálu

Obrázek 16: Testovací sestava

5. SHRNUTÍ

Podle zadání má modul sloužit pro zobrazení dat na počítačovém monitoru. Tuto úlohu splňuje a navíc ji rozšiřuje o možnost použití některých funkčních kláves a možnost výběru barvy zobrazení. Druhou funkcí obvodu je generování testovacího barevného obrazce. Mezi funkcemi lze libovolně přepínat.

Možnosti modulu lze dále rozšiřovat. Doplněním o vysílací modul pro UART a blokem pro dekódování signálů standardní PS/2 klávesnice, jejíž port je také přítomen na kitu, je například možné vytvořit vzdálený počítačový terminál pracující v příkazovém řádku. Takový terminál je vhodný i v dnešní době pro Unixové systémy. Dalším přínosem mého modulu je možnost jeho využití jako učební pomůcky v předmětu Programovatelné logické bloky na našem ústavu.

Modul lze také využít pro rozšíření aplikací, v nichž je obvod FPGA použit. Poskytne tak komfortní uživatelské rozhraní, které je ve většině případů realizováno pouze displejem LCD či diodami LED.

6. POUŽITÁ LITERATURA A ZDROJE

[1] KOLOUCH, J. *Programovatelné logické obvody [Skriptum VUT v Brně].* Brno 2002. ISBN 80-214-2197-5

[2] KOLOUCH, J. *Programovatelné logické obvody – Počítačová cvičení [Skriptum VUT v Brně].* Brno 2006. ISBN 80-214-3271-3

[3] DOUŠA, J. *Jazyk VHDL.* Vydavatelství ČVUT, Praha 2003. ISBN 80-01-02670-1

[4] Xilinx, Inc. *Spartan-3 Complete Data Sheet*, *DS099.* Duben 2006. Dostupné z http://www.xilinx.com

[5] Xilinx, Inc. *Spartan-3 Starter Kit User Guide, UG130.* Květen 2005. Dostupné z http://www.xilinx.com

[6] PINKER, J.; POUPA, M. *Číslicové systémy a jazyk VHDL*. BEN – technická literatura, Praha 2006. ISBN 80-7300-198-5

[7] *RS232* [online]. 2003 [cit. 2008-05-28]. Dostupné z http://rs232.hw.cz

[8] *Graphic LCD panel - Text* [online]. 2008 [cit. 2008-05-28]. Dostupný z http://www.fpga4fun.com/GraphicLCDpanel4.html

[9] PIŠL, A. *Jádro obvodu FPGA pro zobrazení dat na monitoru prostřednictvím portu VGA*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Brno 2008. Vedoucí semestrální práce Ing. Michal Kubíček.

[10] *Windows 1250* [online]. 2005 [cit. 2008-05-28]. Dostupné z http://www.microsoft.com/globaldev/reference/sbcs/1250.mspx

7. PŘÍLOHY

7.1. Zdrojový kód obvodu VGADriver

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity VGADriver is
    Port ( Clk : in  STD_LOGIC;
           RGBIn : in  STD_LOGIC_VECTOR (2 downto 0);
               RGBOut: out STD_LOGIC_VECTOR (2 downto 0);
           VideoActive : out  STD_LOGIC;
           HAddr, VAddr : out  STD_LOGIC_VECTOR (9 downto 0);
           HSOut, VSOut : out  STD_LOGIC := '1');
end VGADriver;


architecture Behavioral of VGADriver is
 SIGNAL HActive, VActive, Active : STD_LOGIC;
 SIGNAL HSync, VSync : STD_LOGIC;
begin
 process(Clk)
     VARIABLE HCount : integer range 0 to 799 := 0;
      VARIABLE VCount : integer range 0 to 520 := 0;
 begin
      if Clk'event AND Clk = '1' then
            -- horizontální čítač
            if HCount >= 799 then HCount := 0;
            else HCount := HCount + 1;
            end if;

            -- generování adresy pixelu v řádku pro paměť
            if HCount >= 640 then HAddr <= "0000000000";
            else HAddr <= conv_std_logic_vector(HCount,10);
            end if;

            -- příznak aktivního běhu v řádku
            if HCount >= 640 AND HCount <= 798 then HActive <= '0';
            else HActive <= '1';
            end if;

            -- horizontální synchronizace
            if HCount >= 655 AND HCount <= 751 then HSync <= '0';
            else HSync <= '1';
            end if;

            -- vertikalní čítač
            if HCount >= 799 AND VCount >= 520 then VCount := 0;
            elsif HCount >= 799 then VCount := VCount + 1;
            end if;

            -- generování adresy řádku pro paměť
       if VCount >= 480 then VAddr <= "0000000000";
            else VAddr <= conv_std_logic_vector(VCount, 10);
            end if;
```

```
            -- příznak aktivního běhu ve snímku
            if VCount >= 480 AND VCount <= 519 then VActive <= '0';
            else VActive <= '1';
            end if;

            -- vertikální synchronizace
            if VCount >= 489 AND VCount <= 491 then VSync <= '0';
            else VSync <= '1';
            end if;
      end if;
  end process;

  -- výstup hodnot pro řízení monitoru

  Active <= HActive AND VActive;
  VideoActive <= Active;
  RGBOut <= RGBIn when Active = '1' else "000";
  HSOut <= HSync;
  VSOut <= VSync;

end Behavioral;
```

7.2. Zdrojový kód obvodu CharDecoder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity CharDecoder is
    Port ( ASCII : in  STD_LOGIC_VECTOR (7 downto 0);
           Clk : in STD_LOGIC;
           NewData : in  STD_LOGIC;
           RAMRow : out  natural range 0 to 29;
           RAMCol : out  natural range 0 to 79;
           DataOut : out  STD_LOGIC_VECTOR (7 downto 0));
end CharDecoder;

architecture Behavioral of CharDecoder is
 signal Row: integer range 0 to 29;
 signal Col: integer range 0 to 79;
   signal PrevState: STD_LOGIC := '1';
   signal CursorBlink: natural range 0 to 30000000 := 0;
begin
 process (Clk)
 begin
       if Clk'event and Clk = '1' then

         RAMCol <= Col;
             RAMRow <= Row;

         CursorBlink <= CursorBlink + 1;
         if CursorBlink <= 15000000 then
            DataOut <= "01011111";
         else
            DataOut <= "00100000";
         end if;


         if NewData = '1' then
            if PrevState = '0' then
               if to_integer(unsigned(ASCII)) >= 32
               AND to_integer(unsigned(ASCII)) /= 127
               AND to_integer(unsigned(ASCII)) /= 129
               AND to_integer(unsigned(ASCII)) /= 131
               AND to_integer(unsigned(ASCII)) /= 136
               AND to_integer(unsigned(ASCII)) /= 144
               AND to_integer(unsigned(ASCII)) /= 152
               then
                  Col <= Col + 1;
                  if Col >= 79 then
                     Col <= 0;
                     Row <= Row + 1;
                     if Row >= 29 then
                        Row <= 0;
                        Col <= 0;
                     end if;
                  end if;
                  DataOut <= ASCII;
               end if;
```

```
                if to_integer(unsigned(ASCII)) = 13 then
                   DataOut <= "00100000";
                   Row <= Row + 1;
                   Col <= 0;
                end if;

                if to_integer(unsigned(ASCII)) = 27 then
                   DataOut <= "00100000";
                   Row <= 0;
                   Col <= 0;
                end if;

                if to_integer(unsigned(ASCII)) = 8 then
                   DataOut <= "00100000";
                   if Col = 0 then
                      if Row = 0 then
                         Row <= 29;
                      else
                         Row <= Row - 1;
                      end if;
                      Col <= 79;
                   else
                      Col <= Col - 1;
                   end if;
                end if;

                PrevState <= '1';
             end if;
          else
             PrevState <= '0';
          end if;
       end if;
 end process;
end Behavioral;
```

7.3. Zdrojový kód obvodu CharAddress

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;


entity CharAddress is
    Port ( HAddress : in  STD_LOGIC_VECTOR (9 downto 0);
           VAddress : in  STD_LOGIC_VECTOR (9 downto 0);
           CRAMRow : out  natural range 0 to 29;
           CRAMCol : out  natural range 0 to 79;
           CROMRow : out  natural range 0 to 15;
           CROMCol : out  natural range 0 to 7);
end CharAddress;

architecture Behavioral of CharAddress is
begin
 CRAMCol <= to_integer(unsigned(HAddress(9 downto 3)));
 CROMCol <= to_integer(unsigned(HAddress(2 downto 0)));
 CRAMRow <= to_integer(unsigned(VAddress(9 downto 4)));
 CROMRow <= to_integer(unsigned(VAddress(3 downto 0)));
end Behavioral;
```

7.4. Zdrojový kód obvodu CharRAMAddressGen

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity CharRAMAddressGen is
    Port ( RowAddress : in  natural range 0 to 29 := 0;
           ColAddress : in  natural range 0 to 79 := 0;
           RAMAddress : out  STD_LOGIC_VECTOR (11 downto 0));
end CharRAMAddressGen;

architecture Behavioral of CharRAMAddressGen is
begin
   RAMAddress <= conv_std_logic_vector(((RowAddress*80) +
ColAddress),12);
end Behavioral;
```

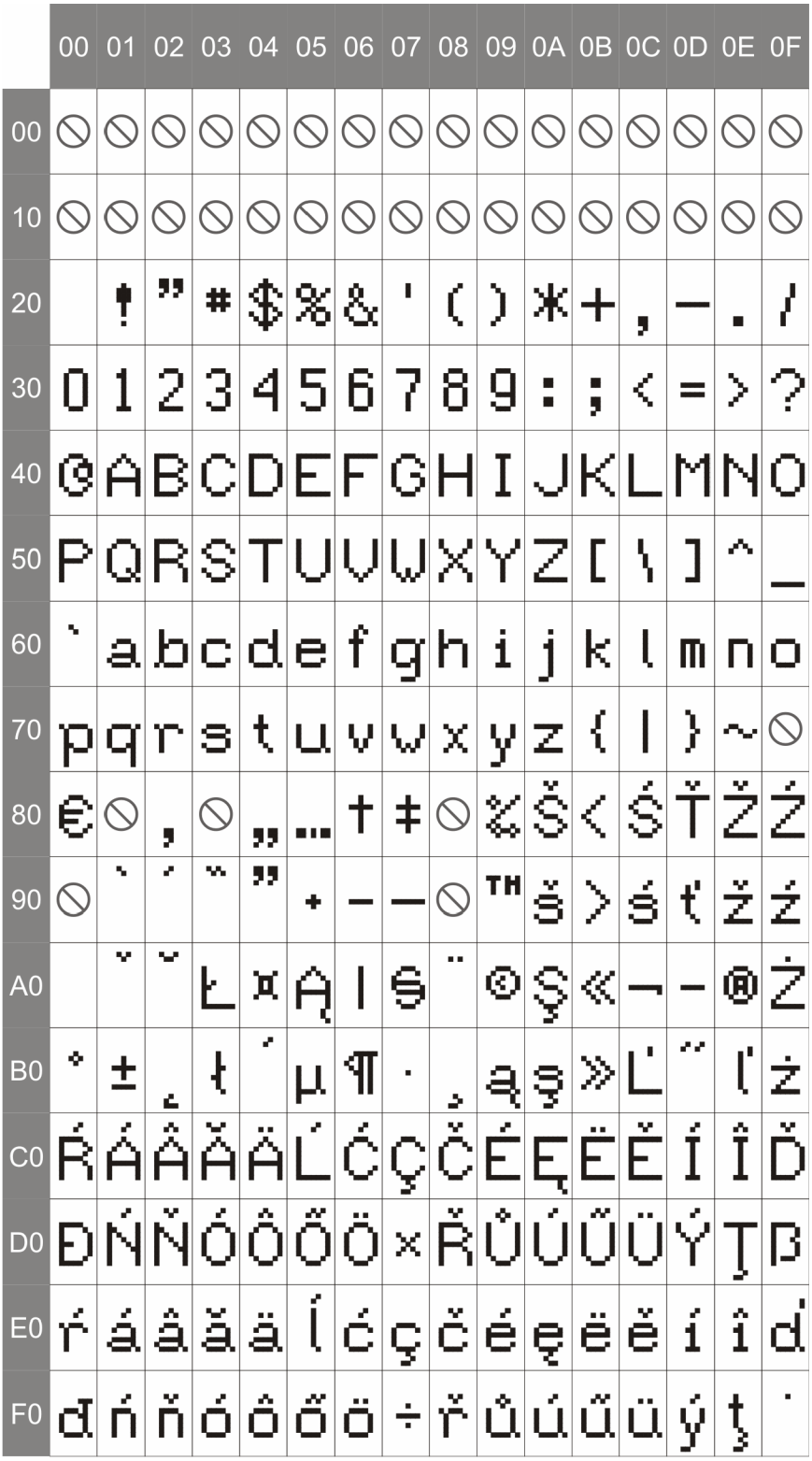
7.5. Zdrojový kód obvodu VGAMultiplexer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity VGAMultiplexer is
    Port ( In3a : in  STD_LOGIC_VECTOR (2 downto 0);
           In3b : in  STD_LOGIC_VECTOR (2 downto 0);
           Switch : in  STD_LOGIC;
           Out3 : out  STD_LOGIC_VECTOR (2 downto 0));
end VGAMultiplexer;

architecture Behavioral of VGAMultiplexer is

begin
   Out3   <= In3a  when Switch = '1' else In3b;
end Behavioral;
```

7.6. Mapa znaků bloku CharROM



Obrázek 17: Mapa znaků CharROM

Záhlaví řádků a sloupců představuje adresu znaku v šestnáctkové soustavě.
Znaky označené symbolem ⦸ nejsou pro zobrazení definovány