České vysoké učení technické v Praze
Fakulta elektrotechnická

Bakalářská práce

Knihovna ovladačů periferií mikrořadiče AVR

Ondřej Kelka

Vedoucí práce: Ing. Pavel Kubalík

Studijní program: Elektrotechnika a informatika strukturovaný bakalářský

Obor: Informatika a výpočetní technika

leden 2007

Poděkování

Děkuji svému spolužákovi Janu Smržovi za neocenitelnou pomoc během celého studia.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 18. 1. 2007 .....

Abstract

This thesis discusses design and implementation of peripheral drivers on Xilinx Starter Kit development board with Xilinx Spartan-3. Drivers for VGA, RS232, display, PS/2 keyboard and memory are working. Further the possibility of implementation of these drivers into an existing AVR core described in VHDL is discussed.

Abstrakt

Tato práce se zabývá návrhem a implementací ovladačů periferií na vývojové desce Xilinx Starter Kit osazené FPGA obvodem Xilinx Spartan-3. Zprovozněny jsou ovladače pro VGA adaptér, sériovou linku, displej, PS/2 klávesnici a paměťový modul. Dále se tato práce zabývá možností úpravy stávajícího popisu obvodu AVR firmy ATMEL popsaného v jazyce VHDL tak, aby umožnoval použití všech těchto periferii.

Obsah

Kapitola 1

Úvod

FGPA obvody jsou moderní technologií využívanou při návrhu a testování digitálních designů. Oblíbenost FPGA obvodů umožňuje jejich výrobu v pokročilých technologiích, např. 65nm výrobní proces. FPGA se vyznačují nízkými vstupními náklady (cena vývojových prostředků, cena vývoje) a celkovými náklady na implementaci při nižších a středních objemech výroby (do desítek tisíc kusů). Vyznačují se též větší flexibilitou vůči ASIC obvodům. Mezi další výhody patří vybavenost pamětí či hotovými funkčními bloky. Nevýhodou FPGA je omezená možnost optimalizace spotřeby a výkonu. Hlavní využití FPGA leží na poli prototypování (pro ASIC), embedded aplikací (např. konfigurovatelné IP-core vhodné pro integraci do programovatelného designu), hybridní čipy (procesor propojený s FPGA umožňuje aplikačně specifické nasazení) a rekonfigurovatelné obvody (umožňují měnit hardwarovou konfiguraci za běhu).

1.1 Popis platformy

Jedním z největších výrobců FPGA obvodů je společnost Xilinx. Její produkty jsou cenově dostupné, poskytují kvalitní platformu pro digitální design a jsou tudíž velmi rozšířené. Společnost Xilinx dodává též vývojové desky s osazeným FPGA a několika základními periferiemi a rozhraními. Příkladem takové vývojové platformy je Xilinx Spartan-3 Starter Kit Board. Jedná se o vývojový kit s FPGA obvodem generace Spartan-3 a základními periferiemi - SRAM, PlatformFlash, JTAG rozhraní, RS232, PS/2, VGA, ap. Takováto vývojová deska je vhodná pro implementaci kontrolérů, procesorů a IP jader. Zároveň je k dispozici několik rozhraní jejichž pomocí lze komunikovat s nejrozšířenějšími periferiemi. Spartan-3 FPGA je rozumně dimenzován pro implementaci kontroléru, který bude schopen využít těchto rozhraní a je tak předurčen být vhodnou platformou pro pro méně náročné designy.

1.2 Cíl projektu

Mým cílem je za pomocí prostředků jazyka VHDL popsat moduly, které umožní využít všech periferií Starter Kitu tak, aby byly snadno použitelné v dalších projektech soustředěných na implementaci složitějších kontrolérů a procesorů. Konkrétní vymezení:

- SRAM – rozhraní pro čtení a zápis do paměti
- Sedmisegmentové displeje – ovladač displeje, zobrazení šestnáckových a desítkových čísel
- Sériová linka – vysílač/přijímač dat
- VGA – rozhraní pro zobrazování grafických či textových dat
- PS/2 – rozhraní pro příjem dat z klávesnice či myši

Součástí projektu je demonstrační příklad, který využívá periferie Starter Kitu. Jedná se o hru Lloydova patnáctka. Jako rozhraní je využit monitor a klávesnice, je možné pomocí sériové linky do hry nahrát obrazová data. K zobrazení je využit grafický i textový režim.

1.3 Obsah práce

Popisy jednotlivých kapitol:

2. Popis Spartan-3 Starter Kit Board
3. Analýza AVR mikrojádra
4. Implementace – ovladače periférií
5. Testování – demonstační designy
6. Aplikace

Kapitola 2

Popis Spartan-3 Starter Kit Board

2.1 Úvod

Spartan-3 Starter Kit je levná platforma pro návrh a testování FPGA designů. Obsahuje základní rozhraní potřebné pro interakci s běžnými periferiemi či přímý vstup uživatele (např. tlačítka).

Hlavní součástí Starter Kitu je Xilinx Spartan-3 XC3S200 FPGA. Obsahuje 200 000 ekvivalentních hradel, z dalších zajímavých vlastností lze zmínit integrovanou rychlou pamět BlockRAM.

Další součásti Starter Kitu:

1. Xilinx XCF02S Platform Flash – slouží s trvalému uložení konfiguračního PROM souboru pro FPGA, obvod pracuje hned po zapojení - potřebnou konfiguraci má uloženou v této trvalé paměti
2. 1MB SRAM – 2 moduly 256Kx16 SRAM s latencí 10ns, společná adresní sběrnice
3. 3-bitový VGA konektor – umožňuje zobrazovat 8 barev
4. 9-pinový RS-232 konektor – deska obsahuje převodník napětových úrovní (MAX232)
5. PS/2 konektor – možnost připojení myši nebo klávesnice
6. 4 sedmisegmentové displeje
7. 8 LED diod
8. 4 tlačítka
9. 8 přepínačů
10. 50MHz krystal – zdroj hodinového signálu
11. tlačítko, kterým lze vynutit rekonfiguraci FPGA, standardně se tak děje pri připojení napětí
12. 3 40-pinové rozširující porty – možnost připojení dalších periferních zařízení
13. JTAG rozhraní

2.2 Rychlá asynchronní SRAM

Spartan-3 Starter Kit obsahuje 1 megabyte paměti SRAM. Pamět je rozdělena do dvou modulů 256Kx16 ISSI IS61LV25616AL-10T s latencí 10ns. Tyto dva moduly mohou tvořit paměť 256Kx32. Oba moduly mají společné signály write-enable (WE#), output-enable (OE#) a adresa (A[17:0]). Tato skutečnost může vývojáře mírně omezovat. Moduly disponují vlastními signály chip select enable (CE#) a bajtově orientovanými signály UB a LB, pomocí kterých lze vybrat určený byte ze 16-bitového slova. Typickým využitím této paměti je grafická pamět režimu VGA. Pro připojení paměti k FPGA je spotřebováno $2 * 16$ datových pinů, $2 * 3$ řídicí piny, 18 společných adresních pinů a 2 společné řídicí piny – celkem tedy 58 pinů ze 173 dostupných.

2.3 Sedmisegmentové displeje

Deska obsahuje 4 dispeje. Všechny displeje mají společné datové signály, každý má ovšem vlastní anodový řídicí signál. Zobrazení čtyř číslic se děje v časovém multiplexu. Tedy vystavíme data sedmisegmentového displeje a anodový řídící signál nastavíme do 0 pro zvolenou pozici a do 1 pro ostatní. Data tedy bude zobrazovat pouze 1 diplej. Pokud budeme dispeje rozsvěcet na dostatečné frekvenci, poskytneme lidskému oku iluzi stálého zobrazení 4 znaků. Tento způsob připojení dispeje výrazně šetří piny FPGA obvodu. Pro připojení 4-znakového displeje je zapotřebí pouze 12 pinů.

2.4 Tlačítka, přepínače, LED diody

Tlačítka obsažená na desce reagují na stisk přivedením napětí logické 1. Přepínače lze přepnout trvale do jedné logické polohy. LED diody jsou připojeny přes odpory a svítí při logické 1. Na připojení těchto periferií je použito $4 + 8 + 8 = 20$ pinů.

2.5 VGA

Spartan-3 Starter Kit je vybaven VGA konektorem DB15, do kterého lze připojit všechny standardní monitory používající VGA rozhraní. K FPGA jsou připojeny signály R, G, B pro barvy a řídicí signály HS, VS – horizontální, resp. vertikální synchronizace. Pomocí 3 vodičů pro barvy lze získat 8 kombinací stavů, což odpovídá 8 barvám.

2.5.1 Časování VGA signálů

Časování VGA signálů se odvíjí od principu funkce CRT monitorů. Uvnitř CRT monitoru přechází elektronový svazek přes tenkou vrstvu luminoforu uspořádaného v rastru. Paprsek postupně prochází zleva doprava v řádku a shora dolů po řádcích, poté se vrátí na začátek obrazovky. Během jednoho průchodu obrazovkou je nutné generovat synchronizaci a dodat obrazová data v požadovaném rozlišení. Při překleslovací frekvenci 60Hz je zapotřebí toto všechno dělat 60× za sekundu. Synchronizace slouží k označení konce řádku (horizontální) a obrazovky (vertikální). Paprsek dopadá na luminofor pouze při pohybu zleva doprava, při návratu na začátek dalšího řádku, resp. další obrazovky je deaktivován. Paprsek musí být též stabilní určitou dobu před začátkem řádku a za jeho koncem. Značnou část doby je tedy paprsek neaktivní, protože se vrací či ustaluje. Je to způsobeno změnou napětí na vychylovacích cívkách, které se musí postupně změnit a ustálit a až teprve poté lze bezpečně paprsek vést v daném řádku.

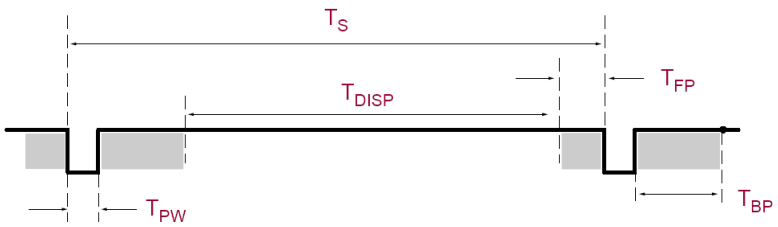
Velikost paprsku, frekvence s jakou paprsek cestuje po monitoru a frekvence s jakou je modulován podmiňují rozlišení monitoru. Moderní monitory podporují různá rozlišení. VGA řadiče jsou schopné generovat signály VGA se zvoleným časováním. Řadiče získávají data pro daný pixel a převádí je na signály RGB se správným časováním.

2.5.2 Časování VGA – 640×480, 60Hz

Chceme dosáhnout rychlosti překleslení 60 obrazovek za sekundu s rozlišením 480 řádků a 640 pixelů na řádek. Významná část doby překleslování je strávena jako neužitečná – nelze zobrazovat, neboť se paprsek vrací či ustaluje. Přesné časování je definováno standardem organizace VESA. Zaveďme pojem šířka pixelu jako dobu potřebnou pro zobrazení jednoho pixelu – paprsek se musí po tuto dobu pohybovat po luminoforu, aby vykreslil 1 pixel. Pro vykreslení 640 pixelů potřebujeme šířku 800 pixelů, šířka 160 pixelů odpovídá době návratu paprsku na začátek řádku a dobu ustálení. Pro vykreslení 480 řádků je zapotřebí doby ekvivaletní zobrazení 521 řádků. Přesné hodnoty zachycuje následující tabulka.

Tabulka 2.1: Časování VGA - 640×480, 60Hz

Symbol	Parametr	Vertical Sync			Horizontal Sync	
		Čas	Šířek px	Řádků	Čas	Šířek px
T_S	Sync pulse time	16.7 ms	416,800	521	32 μs	800
T_{DISP}	Display time	15.36 ms	384,000	480	25.6 μs	640
T_{PW}	Pulse width	64 μs	1,600	2	3.84 μs	96
T_{FP}	Front porch	320 μs	8,000	10	640 ns	16
T_{BP}	Back porch	928 μs	23,200	29	1.92 μs	48



Obrázek 2.1: Časování VGA, převzato z manuálu [1]

Celkem tedy za 1s generujeme $800 * 521 * 60 = 25\,008\,000$ period ekvivalentních šířce 1 pixelu. To odpovídá frekvenci přibližně 25MHz. Jelikož je Starter Kit osazen 50MHz krystalem, lze tuto frekvenci generovat velmi jednoduše.

2.6 PS/2

PS/2 konektor slouží k připojení myši nebo klávesnice. Jedná se o standardní 6-pinový mini-DIN konektor. Připojeny jsou porty napájení (3.3V), zem a porty PS2C a PS2D pro hodiny a data. PS/2 sběrnice se skládá z datového a hodinového signálu, komunikace odpovídá sériové lince. Datová slova jsou 11-bitová, obsahují startbit, 8-bitová data, lichou paritu a stopbit. Význam dat je u klávesnice a myši odlišný. Připojení klávesnice umožňuje obousměrný přenos dat, aby bylo možné rozsvěcet diody na klávesnici.

2.6.1 Časování PS/2

Hodinový a datový signál jsou řízeny pouze během přenosu dat, jinak jsou ve stavu logické 1. Data jsou hostitelem čtena na sestupnou hranu hodin. Zdroj signálu plně kontroluje data i hodiny, musí

ovšem splňovat předepsané časování popsané v následující tabulce. Bližší informace o rozhraní a formátu dat lze získat ze [2].

Tabulka 2.2: Časování PS/2, převzato z manuálu [1]

Symbol	Parametr	Min	Max
T_{CK}	Půlperioda hodin	30 μs	50 μs
T_{SU}	Data-to-clock předstih	5 μs	25 μs
T_{HLD}	Clock-to-data přesah	5 μs	25 μs

2.6.2 Klávesnice

Klávesnice používá připojení s otevřeným kolektorem, které umožňuje řízení sběrnice jak klávesnici, tak hostiteli. Pokud hostitel nikdy neposílá data, může považovat PS/2 signály pouze za vstupní. PS/2 klávesnice předává hostiteli scan kódy kláves. Každá klávesa má přiřazen scan kód. Po stisknutí klávesy je odeslán tento scan kód, pokud je klávesa stále stisknutá, scan kód je opakovaně posílán každých 100ms. Při uvolnění klávesy je nejprve odeslán kód `F0`, který označuje uvolnění, následovaný scan kódem uvolněné klávesy. Rozšiřující klávesy mají dvoubajtový scan kód, kde první bajt odpovídá `E0`. Při uvolnění takové klávesy se odešle nejprve sekvence `E0 F0` následovaná druhým bajtem scan kódu. Hostitel může posílat příkazy klávesnici, např. rozsvícení LED, změna opakovací frekvence a reset.

2.6.3 Myš

Myš neumožňuje příjem dat od hostitele. Při každém pohybu myši jsou vygenerovány 3 datové přenosy. První bajt představuje stav myši, např. stlačená tlačítka (bity L, R), znaménka posunů (bity XS, YS), přetečení hodnoty posunu (bity XV,YV), druhý bajt představuje velikost relativního posunu po ose X a třetí bajt velikost relativního posunu po ose Y. Relativní posun představuje posun vůči minulé pozici myši. Pokud se myš pohybuje stále, trojice dat je odesílána každých 50ms. Myš využívá relativní souřadný systém, kde posun vpravo generuje kladný posun po ose X, tudíž hodnota bitu XS bude 0 (=kladný posun). Posun myší vlevo bude generovat záporný posuv, tudíž bit XS bude roven 1. Obdobně pro osu Y. Pokud bude velikost posunu příliš velká a nebude možné ji popsat 8 bity, je bit XV, resp. YV nastaven do 1.

2.7 Sériová linka

Starter Kit je vybaven konektorem RS-232 DB9 a převodníkem napěťových úrovní (MAX232), umožňuje tudíž bezproblémové využívání sériové linky. Díky zapojení konektoru DB9 na desce máme k dispozici pouze piny RX a TX. Lze tedy realizovat třížilové spojení s kabelem RX-TX-zem. Na pinu RX jsou přijímána data, na pinu TX odesílána. Jedná se o vzájemnou sériovou komunikaci, tedy že jednotlivé datové bity jsou vysílány po jediném vodiči postupně za sebou. Pro komunikaci je nutné, aby propojená zařízení znala rychlost, jakou se data přenášejí. Předpokládá se, že rychlost komunikace v obou směrech bude stejná. Jelikož zařízení komunikují asynchronně, musí existovat způsob synchronizace spojení. Jelikož máme k dispozici jenom datový vodič, synchronizace se děje při poslání definované posloupnosti. V případě sériové komunikace je před vlastními datovými bity odeslán synchronizační start-bit, jehož hodnota je 0. Jelikož klidová hodnota na lince je 1, dojde na synchronizační hranu k přepnutí stavu a zařízení se připraví na komunikaci, která je synchronní. Rámec obsahuje tedy 1 start-bit, 8 bitů dat, 1 bit volitelný paritní a 1 stop-bit, jehož hodnota je 1. Po přenesení stop-bitu zůstává na lince klidová hodnota 1, nebo následuje přenos dalšího rámce dat.

2.8 Zdroje hodinového signálu

Starter Kit obsahuje 50MHz krystal Epson SG-8002JF. Umožňuje též připojení externího krystalu.

Kapitola 3

Analýza AVR mikrojádra

3.1 Úvod

AVR je řada mikrořadičů od firmy Atmel. AVR má RISC architektura. AVR řadiče se dodávají ve dvou rodinách - ATtiny a ATmega. ATtiny má omezenou instrukční sadu a malý počet pinů – 8 až 20. ATmega má plnou instrukční sadu a řadiče periferií, např. SPI, UART, LCD, USB atd.

Do rodiny ATmega patří i mikrokontrolér ATmega103. Přehled vlastností dle [6]:

- 121 instrukcí
- 32 osmibitových GPR registrů
- 4KB interní SRAM
- SPI rozhraní - Master/Slave
- programovatelný Watchdog časovač
- programovatelný UART řadič
- RTC – real-time counter
- 16-bitový časovač/čítač, PWM
- ADC – převodník
- vnitřní a vnější zdroje přerušení

3.2 Architektura

AVR používá harvardskou architekturu – paměť programu a datová paměť jsou odděleny. Vstup do programové paměti probíhá přes jednostupňovou pipeline. Během provádění jedné instrukce je další instrukce přednačítána z programové paměti. Instrukce mají většinou formát 16-bitového slova, tedy každá adresa v programové paměti obsahuje jednu 16-bitovou instrukci.

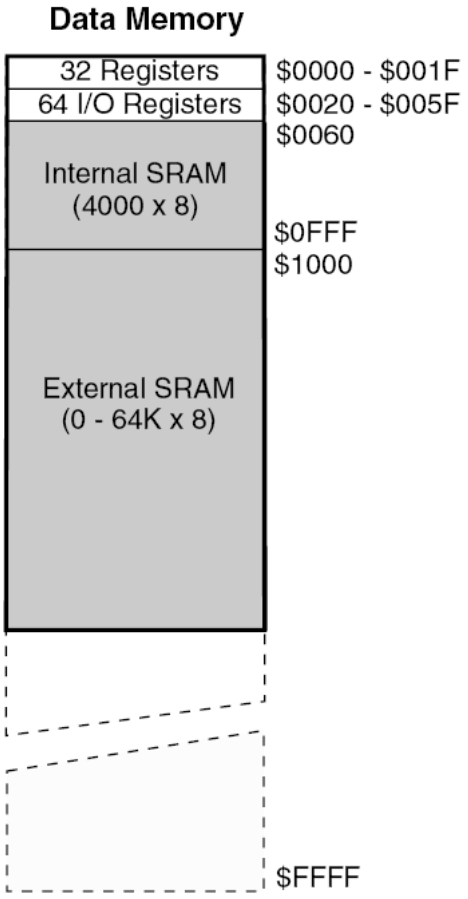
Během přerušení či volání podprogramů je uložena návratová adresa programového čítače PC na zásobníku. Zásobník je alokován v datové paměti. Všechny programy musí inicializovat ukazatel zásobníku SP.

Datová pamět může být adresována 5 adresačními módy.

Řadič přerušení má své řídící registry ve V/V prostoru a bit globálního povolení přerušení ve stavovém registru. Přerušení mají vektor obsluhy v tabulce na začátku programové paměti. Přerušení mají priority podle pozice v tabulce, nižší adresa znamená vetší prioritu.

3.3 Architektura adresního prostoru

Jelikož má AVR harvardskou architekturu, datovou adresou adresujeme registry, porty a datovou paměť. AVR má prvních 96 adres vyčleněno na adresaci registrů a portů, poté následuje SRAM. Adresa je 16-bitová, lze adresovat 65536 položek v paměti. Ve standardním AVR je SRAM rozdělena na interní a externí. Interní se nachází na adresách (96;4095), externí na adresách (4096;65535), vybavení dat trvá o takt déle než u interní SRAM.



Obrázek 3.1: Organizace paměti dat

3.4 Popis obvodu ve VHDL

Na stránce `opencores.org` jsou shromažďovány popisy obvodů ve VHDL a Verilogu. Jedním z projektů je AVR Core, což je popis AVR ATmega103 ve VHDL, viz [7]. Autorem popisu je Ruslan Lepetenok a projekt realizoval v letech 2002-2003. Vlastnosti:

- 32 × 8 GPR registrů
- 23 vektorů přerušení
- podpora až 128 Kb paměti programu a až 64 Kb datové paměti
- programovatelný UART

- dva 8-bitové časovače/čítače s oddělenými předděličkami a PWM
- 8 externích zdrojů přerušení

3.5 Bližší seznámení se s popisem ve VHDL

ISE projekt: avr

Datová paměť je popsána jako RAM se synchronním zápisem a asynchronním čtením. Pokud změníme čtení na synchronní, rozpozná syntézní nástroj BlockRAM. Programová paměť je popsána jako ROM. Popis programové paměti:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity prom is port (
address_in : in  std_logic_vector (15 downto 0);
data_out   : out std_logic_vector (15 downto 0));
end prom;

architecture rtl of prom is
begin
data_out <=
x"E000" when address_in = 16#0000# else
x"E000" when address_in = 16#0001# else
x"BB07" when address_in = 16#0002# else
x"EF0F" when address_in = 16#0003# else
x"BB0A" when address_in = 16#0004# else
x"B326" when address_in = 16#0005# else
x"BB2B" when address_in = 16#0006# else
x"CFFD" when address_in = 16#0007# else
x"ffff";
end rtl;
```

Formát programové paměti odpovídá Intel HEX. Ke konverzi z HEX do PROM je k dispozici program `hex2vhd.exe`. Občas je PROM.vhd zkonvertován s chybou, adresa 0000 má přiřazeny 2 hodnoty.

3.6 Testování

Na testování portů a paměti jsem použil následující program. PORTB je nastaven ke čtení, PORTA je nastaven pro zápis. Ve smyčce je čten PORTB, hodnota je vymaskována 0xAB a zapsána do paměti. Adresa paměti se automaticky inkrementuje.

```
ldi r16, 0xFF
ldi r16, 0x00  ;direction of PORTB is: input (DDRB config)
out $17, r16
ldi r16, 0xFF  ;direction of PORTA is: output (DDRA config)
out $1A, r16

ldi r29, 0x7f
ldi r30, 0xFF  ; register Z cleared
ldi r31, 0x00

ex:
  ldi r17,0xAB  ;mask
```

```
in r18, $16   ;PORTB read
and r17, r18  ;apply mask
st Z, r17     ;write to [Z]
ldi r17,0     ;zeros for sure
ld r17, Z+    ;load R17 from [Z], inc Z
out $1B, r17  ;PORTA write
cpse r31, r29 ;skip next if equal
rjmp ex
ldi r31, 0x00 ;clear part of addr
rjmp ex
```

Tento program bohužel nefunguje. Respektive nějakou dobu lze manipulovat s přepínači a program funguje, ale při přepnutí nejlevějšího přepínače najednou všechny diody zhasnou. Chybu jsem hledal ve stávajícím popisu AVR. Bohužel popis není vybaven dokumentací a tak se mi nepodařilo objevit zdroj chyby. Nepodařilo se mi proniknout do stávajícího designu natolik, abych byl schopen vlastními silami chybu odstranit. Po domluvě s vedoucím práce jsem se rozhodl upustit od úprav AVR jádra, jelikož by to byla pravděpodobně práce přesahující bakalářský projekt. Dále jsem se věnoval tvorbě ovladačů periférií s tím, že testovací aplikace bude založena na automatu v hardwaru.

Kapitola 4

Implementace – ovladače periférií

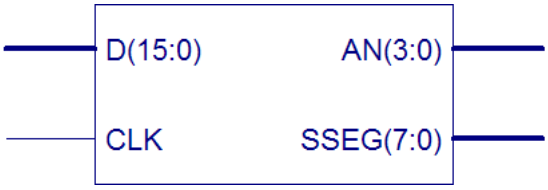
4.1 Ovladač sedmisegmentového dipleje

4.1.1 Ovladač pro zobrazení 4 šestnáctkových cifer

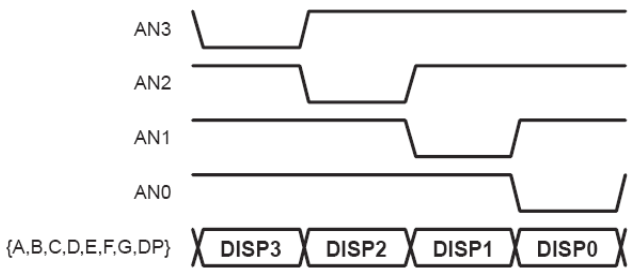
Rozhraní

Definice entity ve VHDL:

```
entity SSEGDISPLAY is port (
   CLK  : in  std_logic;
   D    : in  std_logic_vector (15 downto 0);
   AN   : out std_logic_vector (3 downto 0);
   SSEG : out std_logic_vector (7 downto 0));
end SSEGDISPLAY;
```



Obrázek 4.1: Entita SSEGDISPLAY



Obrázek 4.2: Princip časového multiplexu, převzato z manuálu

Popis

Vstupem jsou hodiny `CLK` a 16-bitové číslo `D`. Výstupem jsou signály budící displeje, resp. anody (`AN`) a segmenty (`SSEG`) sedmisegmentového displeje. Pro buzení displeje je použita technika časového multiplexu popsaná na straně 4. Ve 4-bitovém signálu `AN` postupně aktivujeme vždy jeden bit a příslušný displej je buzen výstupem `SSEG`. Rychlým přepínáním aktivního bitu resp. displeje v signálu `AN` docílíme iluze stálého zobrazení 4 znaků. Frekvence změny celého displeje je $50000000/16384 \doteq 3052$Hz. Jeden displej je buzen po dobu 1/4 periody.

Výsledky syntézy XST

```
Synthesizing Unit <ssegdisplay>.
    Found 16x7-bit ROM for signal <$n0000>.
    Found 1-of-4 decoder for signal <AN>.
    Found 14-bit up counter for signal <CLKDIV>.
    Found 4-bit 4-to-1 multiplexer for signal <DIGIT>.
    Summary:
inferred   1 ROM(s).
inferred   1 Counter(s).
inferred   4 Multiplexer(s).
inferred   1 Decoder(s).
Unit <ssegdisplay> synthesized.

Device utilization summary:
---------------------------
Selected Device : 3s200ft256-5
 Number of Slices:                       18  out of   1920     0%
 Number of Slice Flip Flops:             14  out of   3840     0%
 Number of 4 input LUTs:                 32  out of   3840     0%
 Number of bonded IOBs:                  29  out of    173    16%
 Number of GCLKs:                         1  out of      8    12%

Timing Summary:
---------------
Speed Grade: -5
   Minimum period: 4.088ns (Maximum Frequency: 244.597MHz)
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: 9.978ns
   Maximum combinational path delay: 9.629ns
```

4.1.2 16-bitový BCD převodník

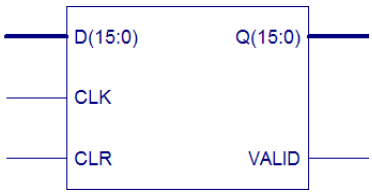
Rozhraní

Definice entity ve VHDL:

```
entity DECIMAL is port(
   CLK   : in  std_logic;
   CLR   : in  std_logic;
   D     : in  std_logic_vector (15 downto 0);
   Q     : out std_logic_vector (15 downto 0);
   VALID : out std_logic);
end DECIMAL;
```

Popis

BCD převodník je sekvenční obvod, který převádí binární číslo do kódu BCD. Tato funkce se používá v případě, že je zapotřebí zobrazovat číslo v lidsky čitelném formátu desítkového čísla např.



Obrázek 4.3: Entita DECIMAL

na displeji. Převodník sériově čte jednotlivé bity binárního čísla a provádí převod do formátu BCD. Na konci převodu je k dispozici převedená hodnota, přičemž převod trvá 64 taktů. Po skončení začne automaticky znovu další převod.

Vstupem převodníku jsou hodiny `CLK`, 16-bitové binární číslo `D` a reset `CLR`, který ukončí převod, znovunačte číslo a začne nový převod. Výstupem je BCD číslo `Q` a signál `VALID`, který označuje platný výstup převodníku, trvá jeden takt a objeví se zároveň s platným výstupem, výstup ovšem platí celých 64 taktů, do ukončení dalšího převodu. Číslo `Q` je 16-bitové a tudíž obsahuje 4 desítkové cifry - zobrazovaný rozsah je tedy (0000;9999). Nejvyšší binární číslo, které lze převést bez přetečení je 14-bitové číslo "10011100001111" (0x270F, 9999). Pokud změníme vstup, výstup získáme po 64–127 taktech. Vznikne tedy zanedbatelné zpoždění výstupu, které můžeme ignorovat, jelikož zobrazujeme čísla, která se mění výrazně méně často než s periodou desítek hodinových taktů (takové změny by člověk nebyl schopen vnímat).

Entita DECIMAL má vnitřní strukturu. Entita REMAINDER provádí vlastní převod. Vstupní registr SR16CL, který zároveň slouží jako posuvný registr pro ukládání mezivýsledků. Čítač COUNTER ovládá načítání převáděného čísla a LOAD pro výstupní registry. Výstupní registry FD4CL slouží k uložení vypočtených hodnot, před jejich vystavením na výstup.

Problém převodu, algoritmus

Mějme 4-bitové binární číslo. Desítkově může nabývat hodnot 0–15, zabírá tedy až dvě cifry. Naším cílem je získat binární hodnoty těchto cifer, tedy převést 4-bitové binární číslo na 8-bitové BCD číslo. Číslo, které chceme převést uložíme do posuvného registu. Výstup budeme ukládat do 2 výstupních registrů. Dále potřebujeme převodní registr.

1. vynulujeme převodní registr
2. sečteme hodnotu převodního registru a MSb převáděného čísla
3. pokud je tato hodnota menší než 5, další hodnota registru bude (hodnota registru)*2 a přetečení bude 0
4. pokud je tato hodnota vetší nebo rovna 5, další hodnota registru bude (hodnota registru)*2-10 a přetečení bude 1
5. opakujeme body 2-4 pro všechny bity převáděného čísla
6. v převodním registru na konci převodu bude číslo odpovídající zbytku po dělení 10, obsah uložíme do jednoho výstupního registru
7. sériová posloupnost hodnot přetečení je hodnota výsledku celočíselného dělení 10, zopakujeme body 1-6 pro tuto hodnotu

Příklad: "1100" je dekadických 12, 12 mod 10 = 2 (jednotky), 1 mod 10 = 1(desítky). Výsledek je 12.

Doba trvání převodu je závislá na šířce čísla a na počtu cifer, které chceme získat. Tedy pokud máme 4-bitové číslo a chceme z něj získat 2 cifry, bude převod trvat 4*2=8 taktů. Můžeme se pokoušet získat i více cifer, např. 3, ale všechny cifry třetí počínaje budou 0. Převod 16-bitového

Tabulka 4.1: Převod binárního čísla

Čas	Stav čítače	Data	Příští stav čítače	Přetečení
	převáděné číslo	↓1100		
0	0000	1	0001	0
1	0001	1	0011	0
2	0011	0	0110	0
3	0110	0	**0010**	1
	převáděné číslo	↓0001	←	↙
4	0000	0	0000	0
5	0000	0	0000	0
6	0000	0	0000	0
7	0000	1	**0001**	0

čísla na 4 dekadické cifry zabere 16*4=64 taktů. Pro převod 16-bitového čísla bychom v předchozí tabulce změnili „převáděné číslo“ na 16-bitové číslo a vynulování čítače by probíhalo po 16 taktech.

Výsledky syntézy XST

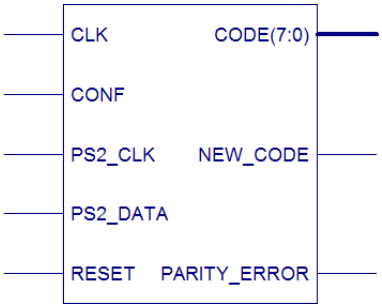
```
HDL Synthesis Report

Macro Statistics
# Counters                          : 1
 6-bit up counter                   : 1
# Registers                         : 27
 1-bit register                     : 20
 4-bit register                     : 7
# Multiplexers                      : 1
 1-bit 4-to-1 multiplexer           : 1

Device utilization summary:
---------------------------
Selected Device : 3s200ft256-5

 Number of Slices:                       38  out of   1920     1%
 Number of Slice Flip Flops:             54  out of   3840     1%
 Number of 4 input LUTs:                 40  out of   3840     1%
 Number of bonded IOBs:                  35  out of    173    20%
 Number of GCLKs:                         1  out of      8    12%

Timing Summary:
---------------
Speed Grade: -5
   Minimum period: 5.047ns (Maximum Frequency: 198.143MHz)
   Minimum input arrival time before clock: 4.833ns
   Maximum output required time after clock: 9.115ns
   Maximum combinational path delay: No path found
```

4.2 PS/2

Rozhraní

Definice entity ve VHDL:

```
entity PS2 is port(
   CLK          : in std_logic;
   RESET        : in std_logic;
   PS2_CLK      : in std_logic;
   PS2_DATA     : in std_logic;
   CONF         : in std_logic;
   NEW_CODE     : out std_logic;
   PARITY_ERROR : out std_logic;
   CODE         : out std_logic_vector(7 downto 0));
end PS2;
```



Obrázek 4.4: Entita PS2

Popis

Vstupem PS/2 přijímače jsou hodiny `CLK`, reset `RESET`, PS/2 signály (`PS2_CLK`, `PS2_DATA`) a potvrzení `CONF`. Výstupen jsou deserializovaná přijatá data `CODE`, oznámení `NEW_CODE` a chybový signál `PARITY_ERROR`.

Přenos dat se odehrává v sériovém formátu. Máme k dispozici signály `PS2_CLK` a `PS2_DATA`. Na sestupnou hranu hodinového signálu `PS2_CLK` nasuneme do posuvného registru data na `PS2_DATA`. Po ukončení přenosu budou deserializovaná data k dispozici na výstupu `CODE` a signál `NEW_CODE` bude v 1. Čeká se na potvrzení přijetí na vstupu `CONF`, poté signál `NEW_CODE` bude v 0. Výstup `PARITY_ERROR` signalizuje chybnou (lichou) paritu.

Hodiny `PS2_CLK` vzorkujeme na frekvenci vstupních hodin. Jestliže je zjištěna sestupná hrana hodin, nastal na lince `PS2_DATA` přenos rámce dat. Očekáváme přenos 11 bitů v rámci. Jelikož frekvence hodin `PS2_CLK` není přesně definována, nelze očekávat přenos v definovaných okamžicích a proto je nutné regenerovat hodiny po celou dobu přenosu. Příjem dat je zajištěn pomocí stavového automatu. Jestliže na lince neprobíhá přenos, je automat v klidovém stavu Idle. Jestliže je započnut přenos, přepne se do stavu Receive. V tomto stavu přijímá do posuvného registru data. V nasunutí dat dojde při sestupné hraně regenerovaných hodin. Parita je uložena v samostatném registru. Startbit a stopbit se neukládají. Každý bit je připočten k čítači bitů. Jestliže dosáhne hodnoty 10, tak přenos končí a na lince je přenášen stopbit. Automat vystaví přijatá data na výstupu `CODE` a vystavení oznámí signálem `NEW_CODE`.

Ve vnitřní struktuře je využita entita PULSE, která slouží v vytvoření pulzu v délce periody hodin, pokud vstupní signál změnil hodnotu z 0 na 1. Vstupem jsou negované regenerované hodiny, tím změníme sestupnou hranu na vzestupnou. Výstup je použit jako SHIFT ENABLE pro přijímací posuvný registr.

Výsledky syntézy XST

```
HDL Synthesis Report
Macro Statistics
# Counters                          : 1
 4-bit up counter                   : 1
# Registers                         : 19
 1-bit register                     : 17
 8-bit register                     : 2
# Multiplexers                      : 3
 1-bit 4-to-1 multiplexer           : 2
 8-bit 4-to-1 multiplexer           : 1
# Xors                              : 2
 1-bit xor2                         : 2

Device utilization summary:
---------------------------
Selected Device : 3s200ft256-5
 Number of Slices:                      29  out of   1920     1%
 Number of Slice Flip Flops:            37  out of   3840     0%
 Number of 4 input LUTs:                41  out of   3840     1%
 Number of bonded IOBs:                 15  out of    173     8%
 Number of GCLKs:                        1  out of      8    12%

Timing Summary:
---------------
Speed Grade: -5
   Minimum period: 4.744ns (Maximum Frequency: 210.781MHz)
   Minimum input arrival time before clock: 4.336ns
   Maximum output required time after clock: 6.216ns
   Maximum combinational path delay: No path found
```

4.3 Sériová linka

Pro vysílání a přijímání dat je zapotřebí znát přenosovou rychlost (Baud Rate - BR) a přesný formát dat, tedy zda se posílá parita, případně zda se jedná o lichou či sudou paritu. Oba následující moduly mají signály `BR` a `PAR`. Výpočet BR – vydělíme frekvenci hodin požadovanou přenosovou rychlostí. Např. pro 57 600 b/s – $50000000/57600 \doteq 868 \rightarrow$ "0000001101100100". Tudíž hodnota BR má význam počtu hodinových taktů spotřebovaných na přenesení 1 bitu.

Tabulka 4.2: Přepočet přenosové rychlosti na BR

Přenosová rychlost [b/s]	BR	BR binárně	Chyba
1200	41667	1010001011000011	$+0,0008\%$
2400	20833	0101000101100001	$-0,0016\%$
4800	10417	0010100010110001	$+0,0032\%$
9600	5208	0001010001011000	$-0,0064\%$
19200	2604	0000101000101100	$-0,0064\%$
38400	1302	0000010100010110	$-0,0064\%$
57600	868	0000001101100100	$-0,0064\%$
115200	434	0000000110110010	$-0,0064\%$

Tabulka 4.3: Formát parity

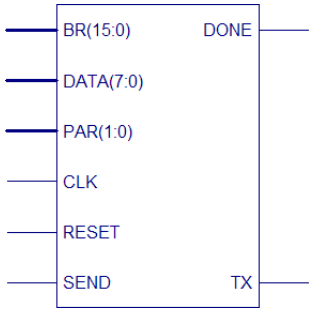
PAR	parita
0X	žádná
10	sudá
11	lichá

4.3.1 Sériová linka - vysílač s paritou

Rozhraní

Definice entity ve VHDL:

```
entity SERIALTX is port(
   CLK   : in std_logic;
   RESET : in std_logic;
   SEND  : in std_logic;
   DATA  : in std_logic_vector(7 downto 0);
   PAR   : in std_logic_vector(1 downto 0);
   BR    : in std_logic_vector(15 downto 0);
   DONE  : out std_logic;
   TX    : out std_logic);
end SERIALTX;
```

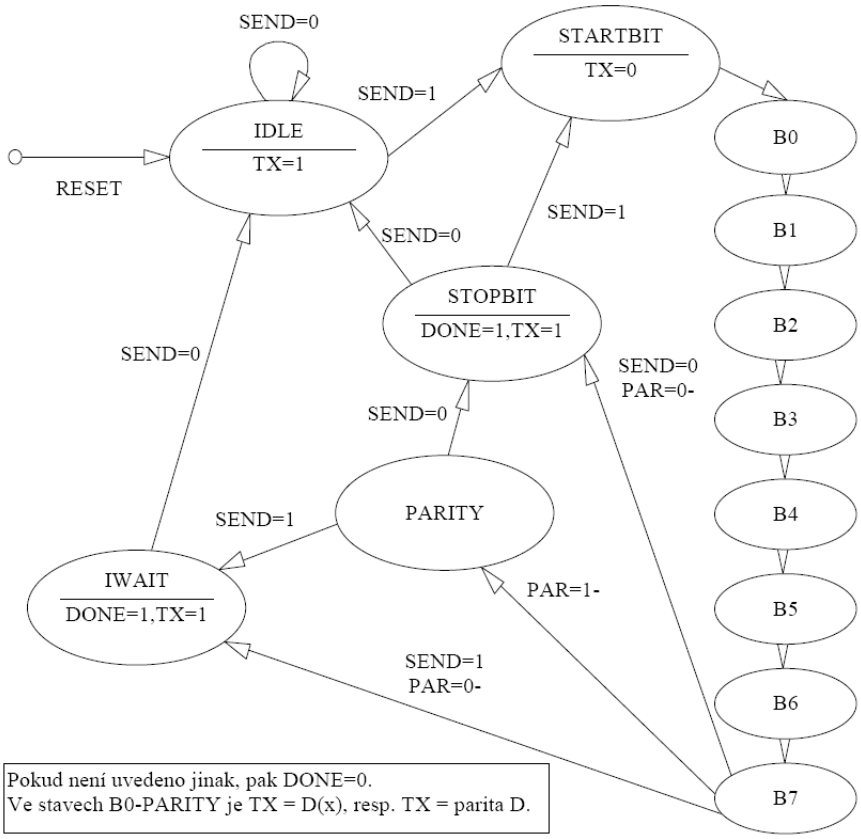


Obrázek 4.5: Entita SERIALTX

Popis

Jelikož v jednom rámci se přenáší 8 datových bitů, musíme vysílači dodat 8-bitový datový signál `DATA`. Přenos bude zahájen na vzestupnou hranu signálu `SEND`. Vysílač generuje výstupní signál `TX` a signál `DONE`, kterým oznamuje ukončení přenosu a očekává dostupnost dalších vstupních dat. Formát dat určují vstupy `BR` a `PAR`.

Řešení používá stavový automat a registr. Klidový stav automatu je stav Idle, na signál `SEND` začíná přenos, data se načtou do registru a automat přejde do stavu STARTBIT, kde setrvá po dobu, kterou určuje čítač Baud Rate. Obdobně projde dále automat stavy B0-B7, během kterých jsou vysílány datové bity. Potom automat přechází buďto do stavu PARITY, pokud je vysílání ve formátu s paritou, jinak do stavu STOPBIT či IWAIT. Pokud je signál SEND ve stavu jedna, očekáváme na jeho uvolnění a automat přejde do stavu IWAIT, ze kterého se dostate při tomto uvolnění. Jinak automat přejde do stavu STOPBIT, po kterém může rovnou přejít do stavu STARBIT a začít tak nový přenos, pokud přenos nenastává, přejde se do klidového stavu Idle.



Obrázek 4.6: Stavový automat vysílače

Výsledky syntézy XST

```
Synthesizing Unit <serialtx>.
    Related source file is "C:/Spartan/bakalarka/04_serialtx_example/serialtx.vhd".
    Found finite state machine <FSM_0> for signal <STAV>.
    -----------------------------------------------------------------------
    | States             | 13                                             |
    | Transitions        | 20                                             |
    | Inputs             | 3                                              |
    | Outputs            | 11                                             |
    | Clock              | CLK (rising_edge)                              |
    | Clock enable       | $n0003 (positive)                              |
    | Reset              | RESET (positive)                               |
    | Reset type         | asynchronous                                   |
    | Reset State        | idle                                           |
    | Power Up State     | idle                                           |
    | Encoding           | automatic                                      |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------
    Found 16-bit comparator equal for signal <$n0003> created at line 128.
    Found 1-bit xor8 for signal <$n0020> created at line 60.
    Found 16-bit up counter for signal <COUNTER>.
    Found 8-bit register for signal <D>.
    Summary:
inferred   1 Finite State Machine(s).
inferred   1 Counter(s).
inferred   8 D-type flip-flop(s).
inferred   1 Comparator(s).
inferred   1 Xor(s).
Unit <serialtx> synthesized.

Device utilization summary:
---------------------------
Selected Device : 3s200ft256-5
 Number of Slices:                       25  out of   1920     1%
 Number of Slice Flip Flops:             37  out of   3840     0%
 Number of 4 input LUTs:                 46  out of   3840     1%
 Number of bonded IOBs:                  31  out of    173    17%
 Number of GCLKs:                         1  out of      8    12%

Timing Summary:
---------------
Speed Grade: -5
   Minimum period: 7.492ns (Maximum Frequency: 133.478MHz)
   Minimum input arrival time before clock: 7.752ns
   Maximum output required time after clock: 13.047ns
   Maximum combinational path delay: 11.479ns
```

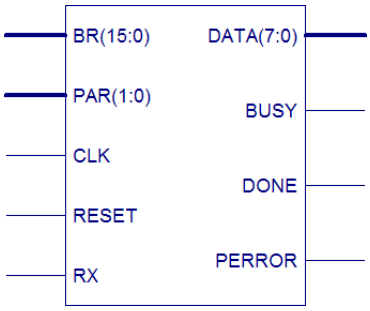
4.3.2 Sériová linka - přijímač s paritou

Rozhraní

Definice entity ve VHDL:

```
entity SERIALRX is port(
   CLK   : in std_logic;
   RESET : in std_logic;
   RX    : in std_logic;
   PAR   : in std_logic_vector(1 downto 0);
```

```
   BR    : in std_logic_vector(15 downto 0);
   BUSY  : out std_logic;
   DONE  : out std_logic;
   PERROR: out std_logic;
   DATA  : out std_logic_vector(7 downto 0));
end SERIALRX;
```



Obrázek 4.7: Entita SERIALRX

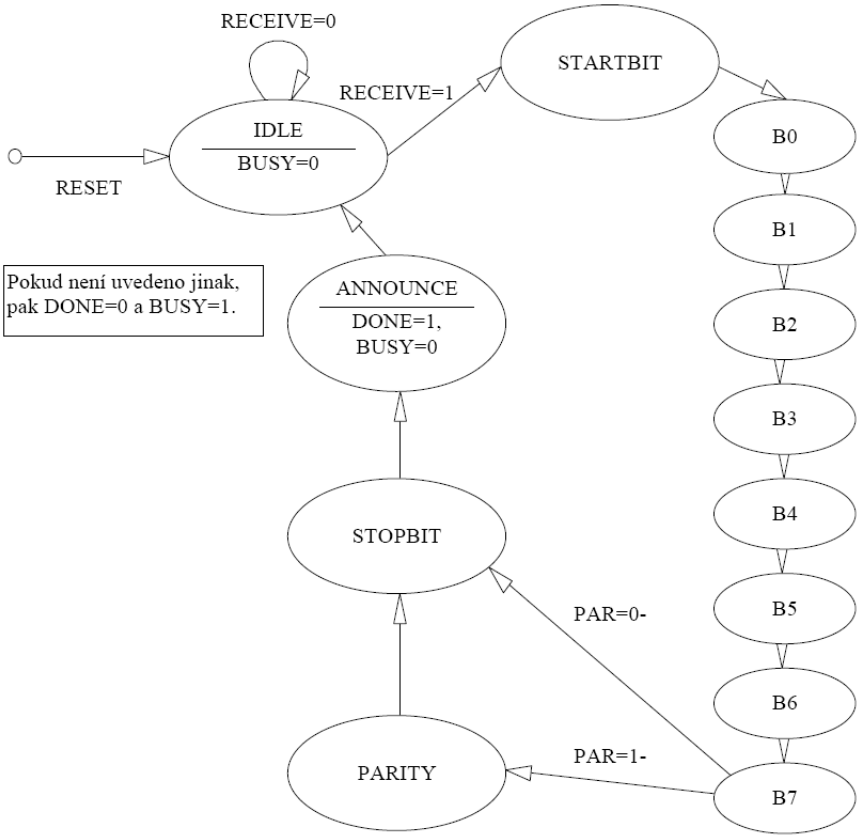
Popis

Datový přenos ovládá druhá strana a proto přenos může začít kdykoliv. Disponujeme informací o parametrech přenosu, ale nevíme kdy začne. Je proto zapotřebí detekovat počátek přenosu. Je vhodné vzorkovat příchozí signál na větší frekvenci, než je hodinová frekvence vysílače. Zvolil jsem frekvenci 8× rychlejší, čtení příchozích dat se tak odehrává přibližně v polovině datového bitu, kde je již ustálený.

Řešení používá opět stavový automat. Detekce začátku přenosu byla vydělena do samostatné entity DETECTOR. Při detekci přechází automat z klidového stavu Idle okamžitě do stavu STARTBIT, protože začátek přenosu je detekován s jistotou až přibližně v polovině startbitu. Přijetí dalších dat se odehrává v daných časových odstupech, které lze zjistit ze signálu `BR`. Tato činnost je zajišťována entitou COUNTER. Podle formátu `PAR` zkontrolujeme paritu dat, chybnou paritu oznamuje signál `PERROR`. Při přijetí stopbitu automat okamžitě přejde do stavu ANNOUNCE, ve kterém je přijetí oznámeno pulzem na signálu `DONE`, deserializovaná přijatá data jsou na výstupu `DATA`. Během přenosu je aktitován signál `BUSY`.

Výsledky syntézy XST

```
Synthesizing Unit <COUNTER>.
    Related source file is "C:/Spartan/bakalarka/05_serialrx_example/counter.vhd".
    Found 16-bit comparator equal for signal <$n0003> created at line 25.
    Found 16-bit up counter for signal <COUNTER>.
    Summary:
inferred   1 Counter(s).
inferred   1 Comparator(s).
Unit <COUNTER> synthesized.

Synthesizing Unit <DETECTOR>.
    Related source file is "C:/Spartan/bakalarka/05_serialrx_example/detector.vhd".
    Found 13-bit comparator equal for signal <$n0001> created at line 24.
    Found 13-bit up counter for signal <COUNTER>.
    Found 5-bit register for signal <REGISTR>.
    Summary:
inferred   1 Counter(s).
```



Obrázek 4.8: Stavový automat přijímače

```
inferred   5 D-type flip-flop(s).
inferred   1 Comparator(s).
Unit <DETECTOR> synthesized.

Synthesizing Unit <serialrx>.
    Related source file is "C:/Spartan/bakalarka/05_serialrx_example/serialrx.vhd".
    Found finite state machine <FSM_0> for signal <STAV>.
    -----------------------------------------------------------------------
    | States             | 13                                             |
    | Transitions        | 26                                             |
    | Inputs             | 4                                              |
    | Outputs            | 13                                             |
    | Clock              | CLK (rising_edge)                              |
    | Reset              | RESET (positive)                               |
    | Reset type         | asynchronous                                   |
    | Reset State        | idle                                           |
    | Power Up State     | idle                                           |
    | Encoding           | automatic                                      |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------
    Found 1-bit register for signal <PERROR>.
    Found 8-bit register for signal <DATA>.
    Found 16-bit comparator equal for signal <$n0004> created at line 115.
    Found 8-bit 4-to-1 multiplexer for signal <$n0025> created at line 196.
    Found 1-bit 4-to-1 multiplexer for signal <$n0026> created at line 196.
    Found 1-bit xor2 for signal <$n0031> created at line 198.
    Found 1-bit xor2 for signal <$n0032> created at line 201.
    Found 1-bit xor8 for signal <$n0033>.
    Found 1-bit 4-to-1 multiplexer for signal <BITL>.
    Found 9-bit register for signal <D>.
    Summary:
inferred   1 Finite State Machine(s).
inferred  18 D-type flip-flop(s).
inferred   1 Comparator(s).
inferred  10 Multiplexer(s).
inferred   1 Xor(s).
Unit <serialrx> synthesized.

Device utilization summary:
---------------------------
Selected Device : 3s200ft256-5
 Number of Slices:                       55  out of   1920     2%
 Number of Slice Flip Flops:             56  out of   3840     1%
 Number of 4 input LUTs:                 88  out of   3840     2%
 Number of bonded IOBs:                  32  out of    173    18%
 Number of GCLKs:                         1  out of      8    12%

Timing Summary:
---------------
Speed Grade: -5
   Minimum period: 6.249ns (Maximum Frequency: 160.019MHz)
   Minimum input arrival time before clock: 6.540ns
   Maximum output required time after clock: 7.999ns
   Maximum combinational path delay: No path found
```

4.4 VGA modul

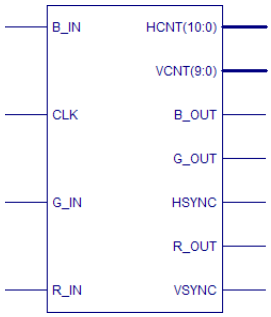
Modul umožňuje zobrazení v režimu VGA 640×480, který je blíže popsán na straně 5.

4.4.1 Obecný VGA modul

Rozhraní

Definice entity ve VHDL:

```
entity VGA is port(
   CLK   : in std_logic;
   R_IN  : in std_logic;
   G_IN  : in std_logic;
   B_IN  : in std_logic;
   HCNT  : out std_logic_vector(10 downto 0);
   VCNT  : out std_logic_vector( 9 downto 0);
   R_OUT : out std_logic;
   G_OUT : out std_logic;
   B_OUT : out std_logic;
   HSYNC : out std_logic;
   VSYNC : out std_logic);
end VGA;
```



Obrázek 4.9: Entita VGA

Popis

Vstupní hodinový signál je taktován na 50MHz, tedy s periodou odpovídající šířce poloviny pixelu při rozlišení 640×480 a překleslovací frekvenci 60Hz. Pixel sice trvá periodu 25MHz hodin, ale rozhodl jsem se raději použít hodiny rychlejší, protože jinak by to znamenalo zavedení druhé hodinové domény do celého designu, což není v tomto nepříliš komplikovaném případě odůvodněné.

Vstupy `RED_IN`, `GREEN_IN` a `BLUE_IN` jsou datové signály, které nesou informaci o barvách. Výstupy `RED_OUT`, `GREEN_OUT` a `BLUE_OUT` jsou připojeny na výstup – konektor VGA. Signály `HSYNC` a `VSYNC` jsou generovány modulem za použití čítačů. Horizontální čítač čítá půlpixely, vertikální čítač čítá řádky. Pokud se hodnota horizontálního čítače nachází v intevalu (0,191), je generována horizontální synchronizace (`HSYNC`=0). Obdobně pokud se hodnota vertikálního čítače nachází v intevalu (0,1), je generována vertikální synchronizace (`VSYNC`=0). Čítače (horizontální a vertikální) jsou vyvedeny i jako výstupy `HCNT` a `VCNT`, protože další jednotky jsou na nich závislé, podle jejich hodnoty připravují data, která budou dodávat VGA modulu.

Tabulka 4.4: Generování synchronizace

Hodnota hor. čít.	Fáze	Hodnota ver. čít.	Fáze
0-191	HSYNC pulz	0-1	VSYNC pulz
192-287	back porch	2-30	back porch
288-1567	obrazová část	31-510	obrazová část
1568-1599	front porch	511-520	front porch

Výsledky syntézy XST

```
Synthesizing Unit <vga>.
    Related source file is "C:/Spartan/bakalarka/06_vga/vga.vhd".
    Found 1-bit register for signal <HSYNC>.
    Found 1-bit register for signal <VSYNC>.
    Found 10-bit comparator less for signal <$n0004> created at line 60.
    Found 11-bit comparator less for signal <$n0007> created at line 52.
    Found 11-bit comparator greatequal for signal <$n0009> created at line 42.
    Found 11-bit comparator less for signal <$n0010> created at line 42.
    Found 10-bit comparator greatequal for signal <$n0011> created at line 42.
    Found 10-bit comparator less for signal <$n0012> created at line 42.
    Found 11-bit up counter for signal <HORIZONTAL_COUNTER>.
    Found 1-bit register for signal <OUTPUT_ENABLE>.
    Found 10-bit up counter for signal <VERTICAL_COUNTER>.
    Summary:
inferred   2 Counter(s).
inferred   3 D-type flip-flop(s).
inferred   6 Comparator(s).
Unit <vga> synthesized.

Device utilization summary:
---------------------------
Selected Device : 3s200ft256-5
 Number of Slices:                       29  out of   1920     1%
 Number of Slice Flip Flops:             24  out of   3840     0%
 Number of 4 input LUTs:                 41  out of   3840     1%
 Number of bonded IOBs:                  30  out of    173    17%
 Number of GCLKs:                         1  out of      8    12%

Timing Summary:
---------------
Speed Grade: -5
   Minimum period: 6.157ns (Maximum Frequency: 162.406MHz)
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: 7.761ns
   Maximum combinational path delay: 7.760ns
```

Kapitola 5

Testování – demonstrační designy

Všechny designy jsou ve složce Design.

Tabulka 5.1: Přehled demonstračních zapojení

Složka	Funkce	Využívá
01_ssegdisplay	Displej, přepínače	SSEGDISPLAY
02_ssegdecimal_example	Displej, přepínače	SSEGDISPLAY, DECIMAL
03_ps2_example	PS/2, displej	PS2, SSEGDISPLAY
04_serialtx_example	Sériová linka, přepínače, tlačítka	SERIALTX
05_serialrx_example	Sériová linka, displej, přepínače	SERIALTX, SSEGDISPLAY
06_vga	VGA, tlačítka	VGA
07_graphics_example	VGA, sériová linka, SRAM	VGA, SERIALRX
08_text_example	VGA, sériová linka, SRAM BlockRAM	VGA, SERIALRX
09_combined_example	VGA, sériová linka, SRAM BlockRAM	VGA, SERIALRX

5.1 Sedmisegmentové displeje

5.1.1 Hexadecimální displej

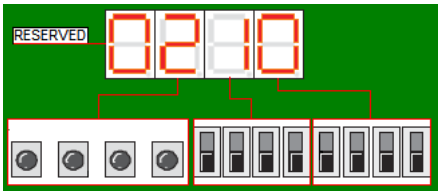
ISE projekt: 01_ssegdisplay

Na 4 displejích lze zobrazit šetnáctkové číslo v rozsahu (0000,FFFF). V jednoduchém demonstačním příkladu použijeme jako vstup přepínače a tlačítka. Přepínače SW7 – SW0 ovládají 2 nižší cifry, tlačítka BTN3 – BTN0 cifru vyšší. Jelikož máme k dispozici pouze 12 vstupů, jsme schopni ovládat pouze 3 nižší cifry. Nejvyšší cifra bude vždy 0.

5.1.2 Decimální displej

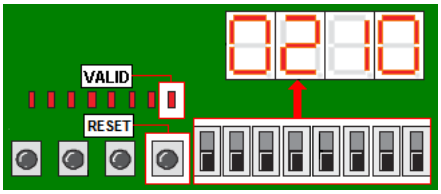
ISE projekt: 02_ssegdecimal_example

Tentokrát chceme zobrazovat na displeji desítkové číslo. Jako zdroj dat opět použijeme přepínače SW7 – SW0. Tím získáme nejvýše číslo 0xFF tedy desítkově 255. Před samotný řadič displeje



Obrázek 5.1: Hexadecimální displej

předřadíme BCD převodník. Jeho RESET je připojen na tlačítko BTN0, výstup VALID pak na LED diodu LD0. Převodník bude neustále převádět číslo na vstupu a to se zpožděním 64 taktů, což je ovšem při frekvenci 50MHz zanedbatelné. Jednou za 64 taktů nastane VALID = 1 a tím pádem bude dioda LD0 slabě svítit.



Obrázek 5.2: Decimální displej

5.2 PS/2

ISE projekt: 03_ps2_example

Jednoduchý příklad demonstruje funkci PS/2 přijímacího modulu. Poslední 2 bajty přijaté z klávesnice jsou hexadecimálně zobrazeny na displeji.

5.3 Sériová linka

5.3.1 Vysílač

ISE projekt: 04_serialtx_example

Demonstrační příklad odesílá 8-bitová data nastavená přepínači SW7 – SW0 přenosovou rychlostí 57 600 b/s. Paritu lze definovat stlačením tlačítek BTN3 – BTN2 dle tabulky 4.3 na straně 18. Pro povolení odeslání 1 znaku je nutné stisknout BTN0. Signál DONE je připojen diodu LD0.

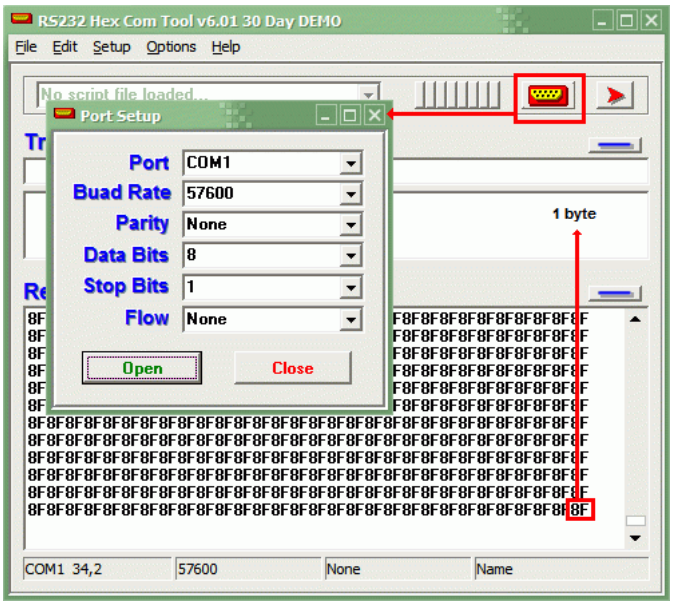
Jako přijímač na straně PC lze využít program HEX Com Tool ze složky Programy.

5.3.2 Přijímač

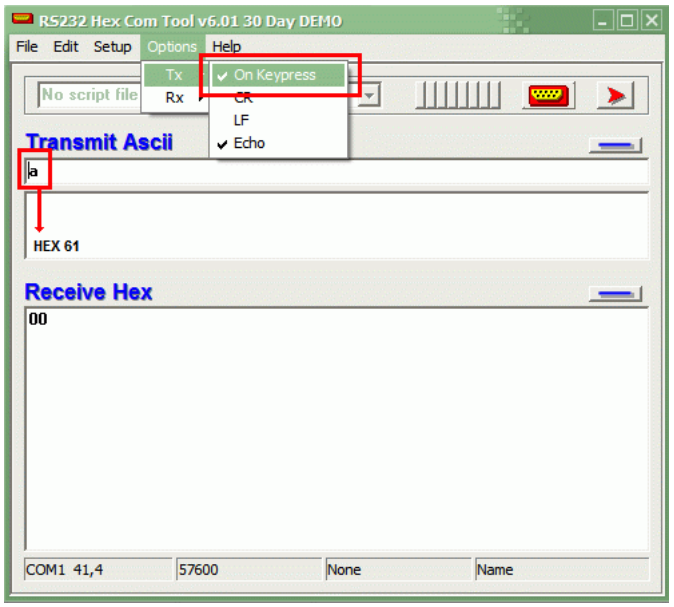
ISE projekt: 05_serialrx_example

Přijímač zobrazí poslední 2 přijaté bajty hexadecimálně na displeji. Předpokládá se přenosová rychlost 57 600 b/s. Paritu lze nastavit na přepínačích SW7 – SW6. Výstupy BUSY, PERROR a DONE na jsou zobrazeny na diodách LD2 – LD0.

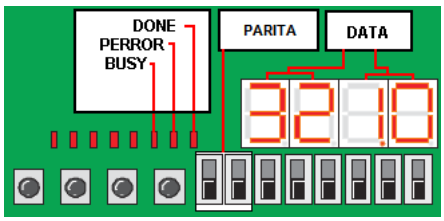
Jako vysílač na straně PC lze využít program HEX Com Tool ze složky Programy.



Obrázek 5.3: HEX Com Tool – příjem



Obrázek 5.4: HEX Com Tool – vysílání



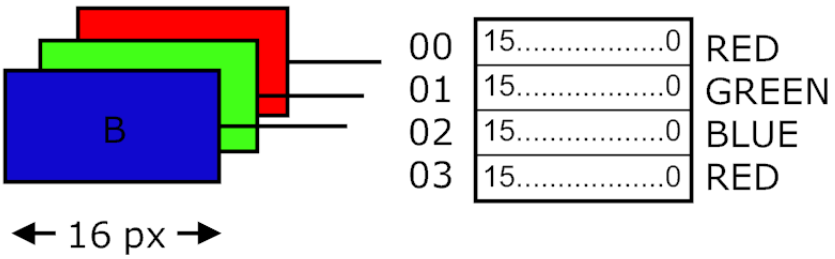
Obrázek 5.5: Sériový přijímač

5.4 VGA

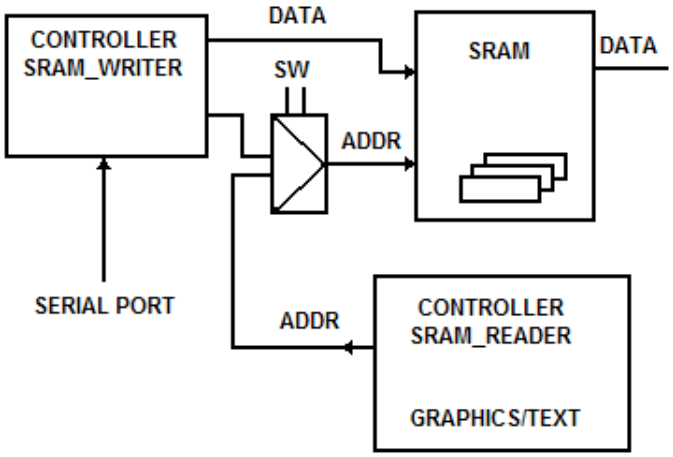
5.4.1 Obecný VGA modul

ISE projekt: 06_vga

Datové vstupy RED_IN, GREEN_IN, BLUE_IN připojíme na přepínače. Přepínač SW5 ovládá BLUE_IN, SW6 ovládá GREEN_IN, SW7 ovládá RED_IN. Jejich kombinací lze vytvořit 8 barev. Celá obrazovka bude mít zvolenou barvu.



Obrázek 5.6: Organizace paměti



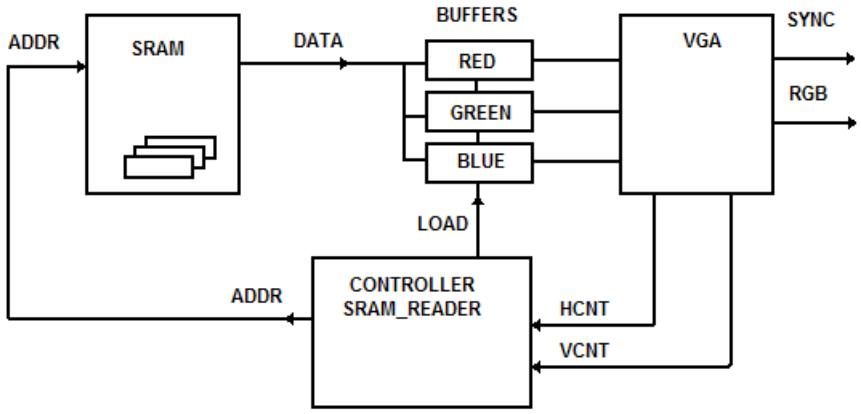
Obrázek 5.7: Schéma propojení, grafické/textové zobrazení - zápis

5.4.2 Grafické zobrazení, data uložena ve SRAM

ISE projekt: 07_graphics_example

V daném zapojení VGA režim disponuje 3 barevnými kanály - tedy celkem 8 barev. Jelikož je to celkem nestandardní počet barev na pixel, je nutné navrhout vlastní organizaci paměti. Pro každý pixel potřebujeme 3 bity (bit na barvu), je nevhodné ukládat pixely ve standardním formátu RGB/BGR, protože nastane plýtvání pamětí. V případě, že bychom četli data nezarovnaně na úrovni řádku SRAM, je nutná další logika. Proto je uložení dat následující – 16b R, 16b G, 16b B. Nejlevější pixel je na pozici MSb. Jelikož musíme znát všechny tři barevné kanály v okamžiku, kdy je chceme psát na obrazovku, musíme fázovat čtení ze SRAM. Nejdříve se přečte R, pak G a B. K uložení dat před jejich použitím je využito chytřejších posuvných registrů. Celkem data v paměti zaberou 115200B, tedy vejdou se do jedné banky SRAM. Pro zobrazení každých 16 pixelů musíme vstoupit třikrát do paměti.

Grafická data je nejprve nutné nahrát do paměti pomocí sériové linky. Zápis do paměti ovlivňuje přepínač SW7, pokud je v pozici 0, budou data zapisována ze sériové linky (rychlost 57600 b/s bez parity). Pokud bude v poloze 1, budou data čtena a zobrazena na monitoru. Pokud chceme nahrát nová data, je nutné zresetovat adresní čítač tlačítkem BTN0.



Obrázek 5.8: Schéma propojení, grafické zobrazení - čtení

Pro nahrání obrázku po sériové lince je potřeba:

1. Přepínač SW7 v poloze 0.
2. copy /b data com1 (trvá přibližně 20 sekund při 57 600 b/s bez parity)
3. Přepínač SW7 v poloze 1 - obraz na VGA.

Pozn. před odesláním je nutné otevřít port, který používáme pro přenos. Port lze snadno otevřít pomocí programu HEX Com Tool ze složky Programy. (Klikneme na ikonu portu (Port Setup), nastavíme přenosovou rychlost (57600 b/s) bez parity, klikneme na Open a celý program zavřeme. Nyní je port otevřen.)

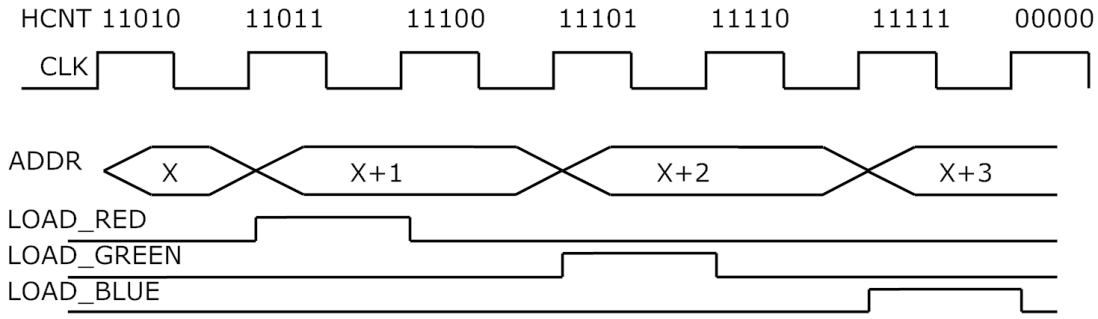
Konverze dat z formátu BMP

Pro konverzi dat je k dispozici program `color640.exe` ve složce Konverze/color640 (ve složce jsou i připravené obrázky). Jeho parametrem je soubor ve formátu BMP, velikosti 640×480 a ve 24-bitové barevné hloubce. Tento soubor musí být upraven posterizací 1bit/barva. Tím bude zaručeno, že je použito nejvýše 8 barev. Výstupem je soubor pevně pojmenovaný jako `data`, který lze odeslat pomocí sériové linky.

Formát BMP je obrazový formát používaný firmou Microsoft. Jedná se o bitovou mapu obrázku. Informace o formátu jsem čerpal z [3]. Soubor BMP obsahuje nejprve hlavičku s popisem vlastností a poté bitmapová data. Hlavička zabírá prvních 54 bajtů souboru. Pokud je BMP uložen ve 24-bitovém formátu, tedy 1 bajt na barvu, následují trojice bajtů pro jednotlivé pixely. BMP soubor velikosti 640×480 bude zabírat celkem 640*480*3+54=921 654 bajtů. Program `color640.exe` pro vnitřní reprezentaci používá 3 matice 640×480 – pro každou barvu. Při načtení bajtu na dané pozici zapíše do matice buďto 1, když má barva hodnotu 0xFF, jinak 0. Po načtení obrázku vypíše program do souboru hodnoty ve formátu 16/16/16. Tudíž přečte 16 hodnot z matice a udělá z nich dvojbajtovou hodnotu, každý bit znamená svítí/nesvítí pro konkétní barvu. Výstup bude 640*480*3/8=115 200 bajtů velký soubor.

Princip fázovaného čtení

Okamžiky, ve kterých musíme mít připravená data, zjistíme ze vstupů HCNT a VCNT. Předpokládejme, že v čase, kdy HCNT(4:0)=00000 začneme zobrazovat na výstup. Je nutné před tímto



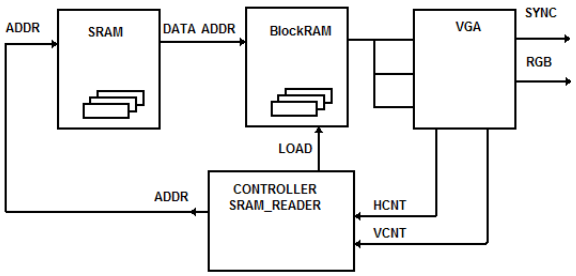
Obrázek 5.9: Časování přístupu do paměti, grafické zobrazení

časem přečíst 3 slova z paměti. 5 taktů před zobrazením vystavíme adresu, ze které chceme číst; 4 takty před zobrazením budou vybavena data v paměti a uložíme je do registru; 3 takty před zobrazením změníme adresu; 2 takty před zobrazením uložíme nová data do dalšího registru; 1 takt před zobrazením změníme adresu a pak v čase zobrazení data načteme do registru a rovnou začneme zobrazovat.

5.4.3 Textové zobrazení, data uložena ve SRAM

ISE projekt: 08_text_example

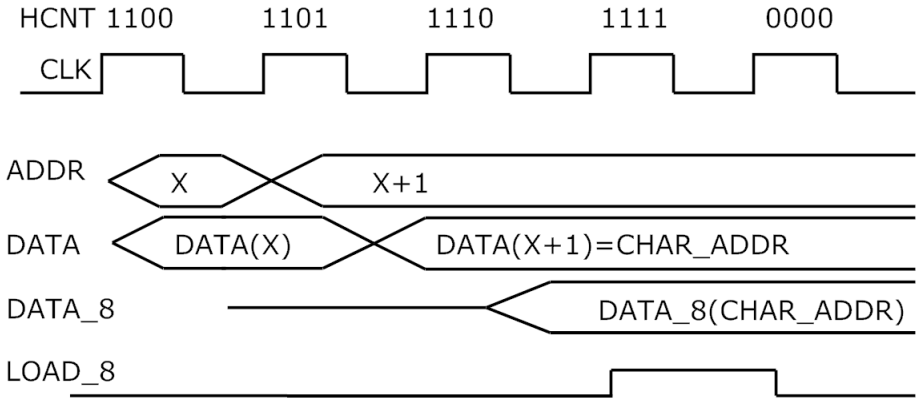
Pro textový režim použijeme stejný modul VGA, tedy rozlišení 640×480. Data jsou uložena v hlavní paměti SRAM, formát interpretujeme jako ISO8859-2 s českými znaky. Volba fontu vychází z použitého rozlišení a je 8×16 pixelů na znak. Rozlišení textového režimu je tedy 80×30 znaků. Data zabírají v paměti celkem 2400B. Pro uložení fontu bylo použito výhodného zapojení on-chip RAM. Tato pamět je organizována v bankách po 16Kb. K popisu fontu je zapotřebí dvou bank.



Obrázek 5.10: Schéma propojení, textové zobrazení - čtení

Nejprve přečeteme ze SRAM jeden bajt. Tento bajt vyjadřuje ASCII hodnotu znaku, použijeme jej jako součást adresy pro vyhledání řádku fontu v BlockRAM. Z BlockRAM dostaneme 1 bajt, každý bit znamená svítí/nesvítí pro jednotlivé pixely ve všech barvách, což odpovídá černobílemu

zobrazení. Na každých 8 pixelů musíme jednou přistoupit do paměti. Data lze nahrát po sériové lince (rychlost 57600 b/s bez parity), když je přepínač SW7 v poloze 0.



Obrázek 5.11: Časování přístupu do paměti, textové zobrazení

Font

Jeden znak má velikost 8×16. Tedy znak se rozprostírá na 16 řádcích a na jednom řádku zabírá 8 pixelů. Bitová reprezentace znaku:

```
// Character 216
Bitmap: ##--##-- \
        -####--- \
        -------- \
        ######-- \
        -##--##- \
        -##--##- \
        -##--##- \
        -#####-- \
        -####--- \
        -##-##-- \
        -##--##- \
        -##--##- \
        ###--##- \
        -------- \
        -------- \
        --------
```

Font bude uložen jako ROM v BlockRAM. ASCII kód je osmibitový – celkem je k dispozici 256 znaků. Font jednoho znaku zabere 16 bajtů. Celkem tedy potřebujeme 256*16=4096 bajtů paměti. Do jedné banky BlockRAM lze uložit 2048 bajtů, font bude využívat 2 banky BlockRAM. Organizace paměti – ASCII znak 0 bude uložen na prvních 16 bajtech, atd. Adresa bude 12-bitová, protože adresujeme 4096 bajtů. Horních 8 bitů odpovídá ASCII kódu, spodní 4 bity odpovídají žádanému řádku. V důsledku bankování paměti je prvních 128 ASCII znaků uloženo v jedné BlockRAM, druhých 128 znaků potom ve druhé BlockRAM. Nejvyšší bit adresy je tedy chip-enable.

Font jsem získal z balíku linuxových konzolových fontů, např. [5]. Jedná se o font `lat2-16.psf`. Za pomoci balíku PSFTools lze tento font zkonvertovat do formátu RAW FONT, který neobsahuje žádné hlavičky, pouze font-ROM data. Tento soubor má očekávanou velikost 4096 bajtů. Jelikož jsem se rozhodl použít Xilinx primitivy RAMB16_S9, je nutné font zkonvertovat do formátu odpovídajícího zápisu paměti BlockRAM. K tomuto účelu slouží program `font.exe`, jehož parametrem

je vstupní RAW FONT soubor. Výsledek je vypsán na standardní výstup. Další informace o BlockRAM jsem čerpal z [4]

5.4.4 Kombinované zobrazení, data uložena ve SRAM

ISE projekt: 09_combined_example

V tomto designu jsou obě předchozí jednotky, jejich výstup na obrazovku lze přepínat pomocí přepínače SW6 – hodnota 0 znamená text, hodnota 1 grafiku. Funkce přepínače SW7 je zachována, pro zobrazování musí být v poloze 1. Pokud je přepínač SW7 v poloze 0, jsou zapisována do paměti data přijatá po sériové lince.

Kapitola 6

Aplikace

6.1 Motivace

Jelikož se mi nepodařilo implementovat periferie do AVR jádra, dohodl jsem se s vedoucím práce na vytvoření komplexního testovacího designu a to hry Lloydova patnáctka. Hra bude využívat grafický a textový řežim VGA zobrazení, vstup z klávesnice a příjem dat po sériové lince.

6.2 Pravidla hry

Hra nese název po Samu Lloydovi, americkém luštiteli křížovek. Její princip je jednoduchý, na hracím plánu je 16 polí a 15 kamenů očíslovaných 1 až 15. Na začátku hry jsou kameny náhodně rozmístěny, pole 16 je volné. Hru řešíme postupným posouváním kamenů na prázdnou pozici – posun kamene nazýváme tahem. Cílem hry je srovnat zamíchané kameny do posloupnosti 1-15.

Tabulka 6.1: Lloydova patnáctka - řešení

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

6.3 Ovládání

Šipkami na klávesnici se přesouvají kameny na místo prázného pole. Tzn. po zmáčknutí šipky nahoru se kámen pod prázdným polem přesune nahoru a prázdné pole se přesune dolů. Klávesou F1 se kameny zamíchají. Přepínač SW7 ovládá příjem dat po sériové lince. Pokud je v poloze 1, data budou zapsána do paměti. V poloze 0 jsou data zobrazována na obrazovce. Tlačítko BTN0 zresetuje zapisovač dat do paměti, takže přístí zápis začne od adresy 0. Tlačítko BTN1 je globální reset. Dioda LD7 svítí pokud je hra vyřešena, LD6 svítí při stisku klávesy F1, LD2 oznamuje chybnou paritu na sériové lince, LD1 oznamuje přenost dat na sériové lince a LD0 oznamuje chybnou paritu na PS/2.

6.4 Design

ISE projekt: 10_lloyd_example

Celý návrh je rozčleněn do logických modulů, které obstarávají konkrétní funkcionalitu.

6.4.1 Automat

Rozhraní

Definice entity ve VHDL:

```
entity AUTOMAT is port(
   CLK      : in std_logic;
   RESET    : in std_logic;
   Xin      : in  std_logic_vector(1 downto 0);
   Yin      : in  std_logic_vector(1 downto 0);
   PointerX : in  std_logic_vector(1 downto 0);
   PointerY : in  std_logic_vector(1 downto 0);
   ValuePX  : out std_logic_vector(1 downto 0);
   ValuePY  : out std_logic_vector(1 downto 0);
   SOLVED   : out std_logic;
   EROW     : out std_logic_vector(1 downto 0);
   ECOL     : out std_logic_vector(1 downto 0));
end AUTOMAT;
```

Popis

Automat je matice hodnot o rozměrech 4×4. Každá hodnota označuje kámen, který leží na daných souřadnicích. Prázdné pole je kámen o hodnotě „0000“. Vstupy `Xin` a `Yin` jsou příští souřadnice prázdného pole. To znamená, že kámen, který je na těchto souřadnicích bude vyměněn s kamenem prázdné pole. Prázdé pole může změnit polohu pouze o jedno pole v jedné ose. Vstupy `PointerX` a `PointerY` jsou ukazatele na libovolné pole, výstupy `ValuePX` a `ValuePY` jsou souřadnice, na kterých by se měl správně nacházet kámen takto odkazovaný. Výstup `SOLVED` přejde do 1, když je hra vyřešena. Výstupy `ECOL` a `EROW` jsou souřadnice prázdného pole.

Funkce

Automat má vstupy, které umožňují provádět posuny kamenů. Kameny lze posouvat pouze na místo, kde se nachází prázdné pole. To je ekvivalentní s výměnou prázdného pole s některým vedlejším kamenem. Výstupem automatu jsou souřadnice prázdného pole, s jeho znalostí snadno dopočítáme, které kameny s ním lze vyměnit. Posun kamenů provádí buďto uživatel vstupem z klávesnice, nebo míchací automat, který pseudonáhodně generuje posuvy. Ukazatele na pole použijeme při zobrazení, kdy část adresy nahradíme hodnotu správných souřadnic. Tím pádem bude zobrazení reflektovat stav automatu. Abychom mohli použít tuto metodu změny adresy je nutné definovat novou organizaci grafické paměti.

6.4.2 Vstup uživatele

Rozhraní

Definice entity ve VHDL:

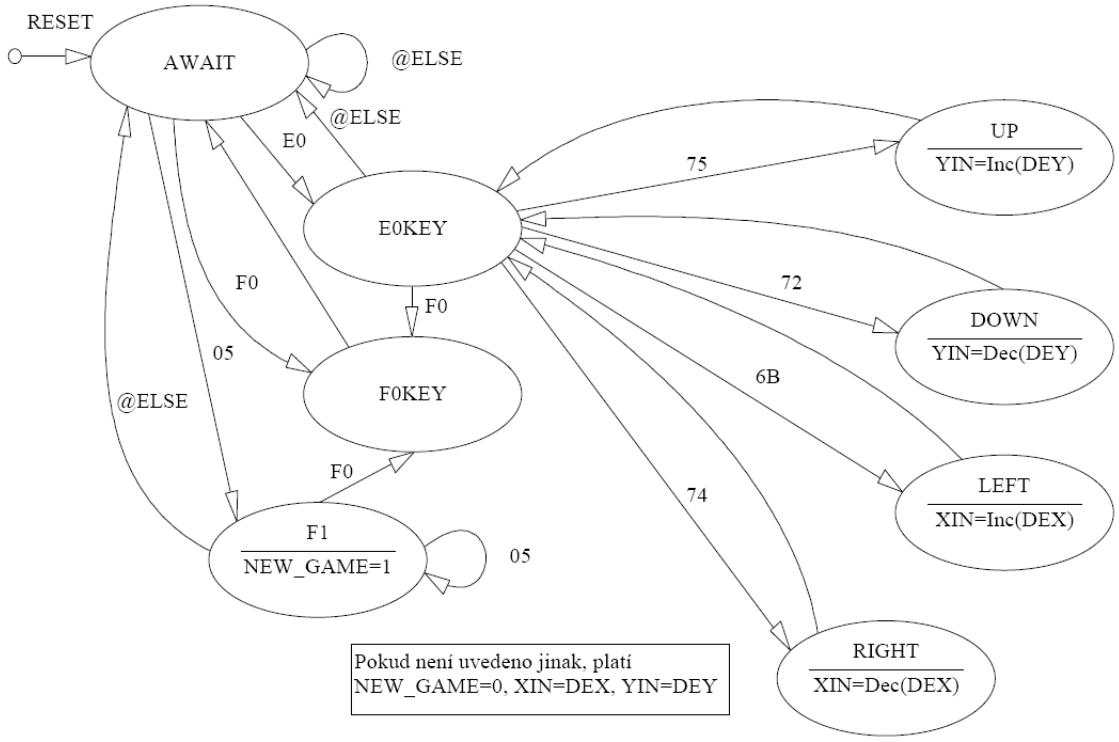
```
entity SWITCHER is port(
   CLK      : in std_logic;
   RESET    : in std_logic;
   CODE     : in std_logic_vector(7 downto 0);
   NEW_CODE : in std_logic;
   EX       : in std_logic_vector(1 downto 0);
   EY       : in std_logic_vector(1 downto 0);
```

```
   NEW_GAME : out std_logic;
   XIN      : out std_logic_vector(1 downto 0);
   YIN      : out std_logic_vector(1 downto 0));
end SWITCHER;
```

Popis

Vstupy `CODE` a `NEW_CODE` jsou připojeny na PS/2 přijímač. Podle příchozích kláves určujeme posun kamenu v automatu. K tomu potřebujeme znát současnou polohu prázdného pole – vstupy `EX` a `EY`. Příští souřadnice prázdného pole jsou na výstupech `XIN` a `YIN`. Výstup `NEW_GAME` označuje stisk klávesy F1 – zamíchání.

Automat



Obrázek 6.1: Vstup z klávesnice, přechody se odehrávají na přijetí scankódu (v hexa)

6.4.3 Míchání

Rozhraní

Definice entity ve VHDL:

```
entity RANDOMIZER is port(
   CLK      : in std_logic;
   RESET    : in std_logic;
   EX       : in std_logic_vector(1 downto 0);
   EY       : in std_logic_vector(1 downto 0);
   NEW_GAME : in std_logic;
```

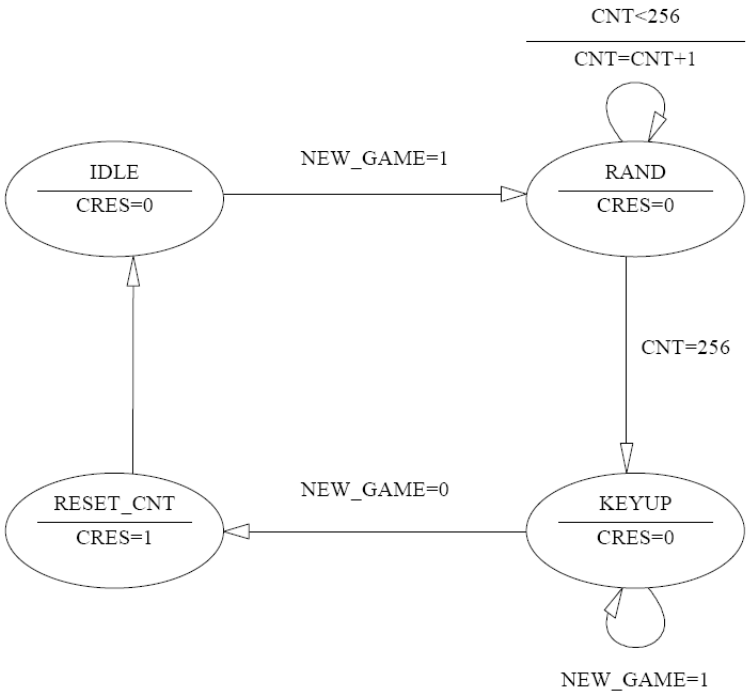
```
   BUSY     : out std_logic;
   CRES     : out std_logic;
   XIN      : out std_logic_vector(1 downto 0);
   YIN      : out std_logic_vector(1 downto 0));
end RANDOMIZER;
```

Popis

Vstupy `EX` a `EY` označují polohu prázdného pole. Vstup `NEW_GAME` spouští stavový automat, který generuje pseudonáhodně posuny na výstupech `XIN` a `YIN`. Během generování je výstup `BUSY` v jedničce. Výstup `CRES` resetuje hodnoty čítačů.

Automat

Pokud se automat nachází ve stavu RAND, ve BUSY=1 a automat mění rozložení kamenů podle dat LFSR.



Obrázek 6.2: Míchací automat

Vnitřní struktura

Jako zdroj dat je zvolen LFSR – linear feedback shift registrer o délce 16 bitů. Jelikož se data opakují po 65536 hodnotách, jsou pseudonáhodná. Ke generování posunu stačí 2 bity – pohyb ve vertikální a horizontální ose. Jsou použity 2 nejnižší bity LFSR. Ke spuštění LFSR slouží modul LATCH_PULSE, který vygeneruje 1 pulz a poté zůstává trvale v nule. Tento pulz zresetuje LFSR a ten potom běží samovolně. K zamíchání je použito 256 hodnot z LFSR, některé z nich nebudou mít efekt na automat, např. v případě, že kámen je u kraje či v rohu. Lze se ovšem domnívat, že velká část z nich nějaký pohyb způsobí. Po desítkách posunů již bude hra dostatečně zamíchána.

6.4.4 Zobrazení

Analýza

Chceme zobrazovat hru a další informace jako čas a skóre. Použijeme tedy grafické zobrazení pro zobrazení hry, data budou uložena ve SRAM ve vhodném formátu. Část obrazovky bude použita k zobrazení textových informací pomocí textového zobrazení. Data budou uložena v BlockRAM. Grafická data budou dodávána na pixelech 0-511, textová na pixelech 512-639 v každém řádku. Obrazová data jsou jednoduše slučována pomocí hradlel OR, protože v každé části obrazovky jsou data dodávána pouze jedním modulem.

Popis grafického formátu

Ve hře je použito 16 kamenů. Původní formát dat je nevhodný pro použití, protože data jsou uložena za sebou a bylo by nutné složitě dopočítávat další adresu. Proto je použit odlišný formát dat. Základní myšlenka spočívá v logickém rozdělení adresy. Kámen je široký 128 pixelů, pixely čteme z paměti po 16-bitových slovech ve 3 barvách. V jednom řádku tedy adresujeme 3*128/16=24 adres. K adresování 24 adres potřebujeme 5 bitů. Kámen zabírá 120 řádků na výšku na adresaci 120 řádků potřebujeme 7 bitů. Kamenů je celkem 16, k adresaci potřebujeme 4 bity. Celkem je tedy adresa 16-bitová.

Tabulka 6.2: Adresace

bity adresy	15-12	11-5	4-0
platné adresy	0000-1111	0000000-1110111	0000-10111
rozsah	0-15	0-119	0-23
počet	16	120	24

Tento formát adresy přináší možnost vyměnit nejvyšší 4 bity a zobrazovat stejnou část jiného kamene. Je zapotřebí přidat výpočet další adresy. Je také nutné přepsat konverzní program, nový program `color512.exe` převádí BMP obrázek 512×480 do tohoto formátu. Naleznete ho ve složce Konverze/color512 společně s připravenými obrázky.

6.4.5 Zjednodušené schéma zapojení

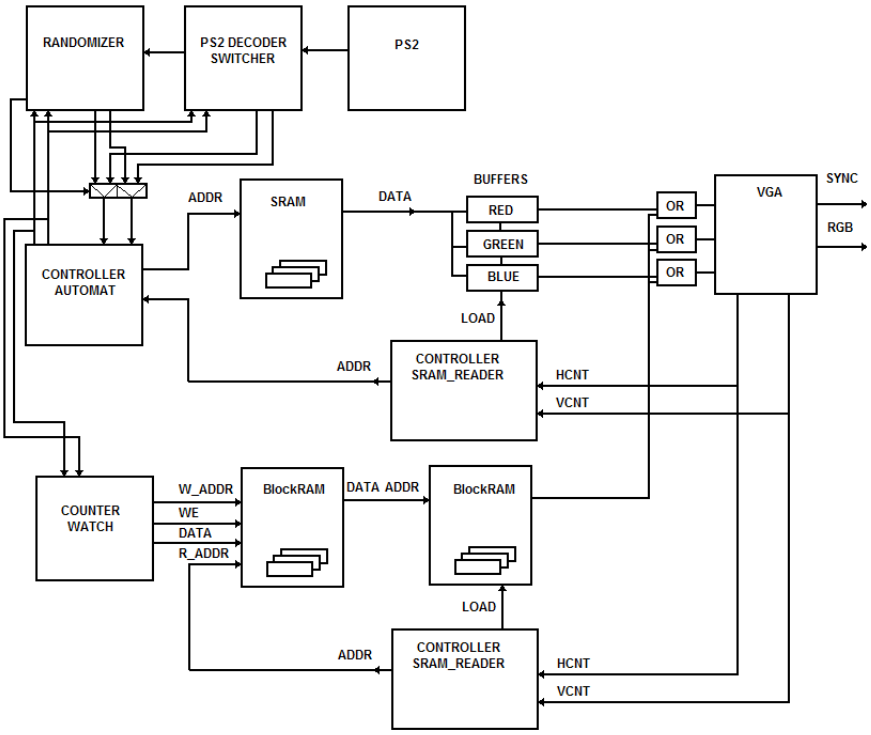
Schéma nezobrazuje přijímač dat ze sériové linky a automat pro zápis do paměti SRAM.

6.4.6 Textová paměť

Rozhraní

Definice entity ve VHDL:

```
entity RAM is
   generic(
   width : integer:=8;
   depth : integer:=512;
   addr  : integer:=9);
   port(
   CLK    : in std_logic;
   WE     : in std_logic;
   R_ADDR : in std_logic_vector(addr-1 downto 0);
   W_ADDR : in std_logic_vector(addr-1 downto 0);
   DI     : in std_logic_vector(width-1 downto 0);
```



Obrázek 6.3: Zjednodušené schéma zapojení

```
   DO     : out std_logic_vector(width-1 downto 0));
end RAM;
```

Popis

Pro zobrazení textu je vyhrazeno 128 pixelů. Šířka odpovídá 128/8=16 znakům. Na výšku se vejde 480/16=30 znaků. Celkem tedy zobrazujeme 16*30=480 znaků. Tyto znaky jsou uloženy v paměti BlockRAM. Entita RAM je popsána se synchronním čtením i zápisem, syntézní nástroj sám rozpozná BlockRAM.

6.4.7 Zápis grafických dat

Rozhraní

Je použit přijímač dat ze sériové linky – SERIALRX a automat pro zápis do paměti – SRAM_WRITER.

Popis

Grafická data zkonvertovaná programem `color512.exe` budou přijata po sériové lince a uložena v paměti. Zapisovací automat získá přístup ke SRAM, pokud je přepínač SW7 v poloze 0. Pokud chceme nahrát jiný obrázek, zapisovací automat zresetujeme tlačítkem BTN0.

6.4.8 Skóre

Rozhraní

Definice entity ve VHDL:

```
entity SCORE_INC is port(
   CLK : in std_logic;
   EROW : in std_logic_vector(1 downto 0);
   ECOL : in std_logic_vector(1 downto 0);
   SCORE_ENA : out std_logic);
end SCORE_INC;
```

Popis

Komponenta slouží k detekci změny pozice prázdného pole – došlo k tahu. Pokud dojde ke změně, objeví se na výstupu `SCORE_ENA` pulz v délce periody hodin. V komponentě čítačů to způsobí zvýšení skóre o 1.

6.4.9 Čas

Rozhraní

Definice entity ve VHDL:

```
entity CLOCK_INC is port(
   CLK : in std_logic;
   SOLVED : in std_logic;
   CLOCK_ENA : out std_logic);
end CLOCK_INC;
```

Popis

Komponenta slouží ke generování signálu povolení čítání pro herní hodiny. Obsahuje čítač do 50 milionů. Každou sekundu čítač přeteče a bude vygenerováno jedno povolení čítání. To způsobí v komponentě čítačů posunutí herních hodin o 1 sekundu. Vstup `SOLVED` potlačuje čítání a proto se hodiny zastaví.

6.4.10 Čítače

Rozhraní

Definice entity ve VHDL:

```
entity SIDEBAR is port(
   CLK   : in std_logic;
   RESET : in std_logic;
   HCNT_4: in std_logic_vector(3 downto 0);
   CHAR_ADR : in std_logic_vector(8 downto 0);
   SCORE_ENA: in std_logic;
   CLOCK_ENA: in std_logic;
   CHAR_WE  : out std_logic;
   CHAR_DI  : out std_logic_vector(7 downto 0));
end SIDEBAR;
```

Popis

Entita SIDEBAR obsahuje v interní struktuře čítač skóre a herní hodiny. Tyto údaje zapisuje do textové paměti, která se zobrazuje na obrazovce. Vstup `HCNT_4` je připojen na 4 nejnižší bity globálního horizontálního čítače. Vstup `CHAR_ADR` je adresa, ze které právě čte textový SRAM_READER. Vstupy `SCORE_ENA` a `CLOCK_ENA` jsou povolení čítaní skóre a herních hodin. Výstupy `CHAR_WE` a `CHAR_DI` ovládají zápisový port textové paměti, jako adresa je použita stejná adresa, ze které se právě čte. Čtecí adresa se neustále cyklicky mění, čehož jsme šikovně využili.

Kapitola 7

Závěr

Zabýval jsem se návrhem ovladačů periferií na vývojové desce Spartan-3 Starter Kit v jazyce VHDL. Navrhl jsem tyto ovladače, připravil jsem demostrační zapojení těchto ovladačů, které dokládá jejich funkcionalitu. V další části své práce jsem měl upravit stávající popis mikrořadiče AVR ve VHDL tak, aby využíval navržených ovladačů periferií. Nejprve jsem se seznámil s architekturou řadiče. Dále jsem upravil popis tak, aby využil on-chip paměť. Napsal jsem testovací program, který měl demonstrovat funkci V/V portů. Tento program nefungoval, snažil jsem se tedy zjistit proč. Chybu jsem hledal ve stávajícím popisu VHDL. Popis řadiče od autora není vybaven dostatečnou dokumentací a proto se mi nepodařilo zjistit, proč program nefunguje. Hlubší zkoumání stávajícího popisu by bylo časově náročné a pravděpodobně by přesahovalo rámec bakalářské práce.

Po dohodě s vedoucím jsem zanechal práce na AVR a soustředil jsem se na návrh komplexnějšího designu, který by demonstroval funkčnost jednotlivých ovladačů periferií. Jako vhodný demonstrační design byla zvolena hra Lloydova patnáctka. Využívá klávesnici, VGA, sériovou linku a paměť SRAM. Hru se mi podařilo implementovat. Při návrhu jsem používal popisu pomocí stavových automatů, takže výsledný design je soustava spolupracujících automatů.

Myslím si, že práce dobře poslouží jako informační zdroj při realizaci či návrhu jiných řadičů a procesorů na dané platformě. Pokud dojde k vývoji projektu AVR core, je možné použít výsledky této práce k úspěšné implementaci ovladačů periférií.

Literatura

[1] Xilinx
Spartan-3 Starter Kit Board User Guide UG130 (v1.1) May 13, 2005
`http://www.xilinx.com/bvdocs/userguides/ug130.pdf`

[2] Adam Chapweske
PS/2 Mouse/Keyboard Protocol
`http://www.computer-engineering.org`

[3] Don Lancaster
Exploring the .BMP File Format
`http://www.tinaja.com/glib/expbmp.pdf`

[4] Xilinx
Xilinx XAPP463 Using Block RAM in Spartan-3 Generation FPGAs
`http://www.xilinx.com/bvdocs/appnotes/xapp463.pdf`

[5] Conectiva
Console tools fonts Package
`ftp://fr2.rpmfind.net/linux/conectiva/snapshot/ppc/RPMS.main/console-tools-fonts-0.3.3-21699cl.ppc.rpm`

[6] Atmel
ATmega103 manual
`http://www.atmel.com/dyn/resources/prod_documents/Doc0945.pdf`

[7] Ruslan Lepetenok
Stránka projektu AVR core
`http://www.opencores.com/projects.cgi/web/avr_core/overview`

Příloha A

Obsah přiloženého CD

Přiložené CD obsahuje tyto dokumenty:

- DESIGN – jednotlivé projekty pro Xilinx ISE 7.1.04i
- KONVERZE – programy pro konverzi z BMP, konverze fontu, konverze HEX2VHD; předpřipravené obrázky pro aplikaci – hru Lloydova patnáctka
- PROGRAMY – PSFtools – konverze fontů, HEX Com Tool – program pro komunikaci po sériové lince
- TEXT – elektronická verze publikace (ve formátu PDF)
- ZDROJE – kopie elektronických zdrojů

Seznam obrázků