VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

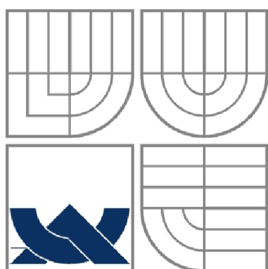
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ŘADIČ SBĚRNICE PCI PRO VÝVOJOVOU KARTU S OBVODEM FPGA

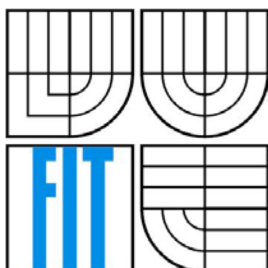
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Ľubomír Ilavský
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ŘADIČ SBĚRNICE PCI PRO VÝVOJOVOU KARTU S OBVODEM FPGA

PCI BUS CONTROLLER FOR DEVELOPMENT BOARD WITH FPGA

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Ľubomír Ilavský

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Václav Šimek

BRNO 2009

Abstrakt

Daná diplomová práce se zabývá problematikou komunikace na sběrnici PCI a návrhem řadiče PCI pro kartu s obvodem FPGA. Na úvod ukazuje funkčnost a struktura obvodů FPGA. Následně popisuje princip komunikace prostřednictvím sběrnice PCI. Po analýze PCI práce popisuje návrh řadiče pro cílovou kartu a seznamuje s jeho jednotlivými částmi. Při procesu implementace, důkladně rozebírá strukturu a činnost jednotlivých bloků PCI řadiče. V další části práce zachycuje proces, realizace a testování vytvořeného řešení na vzdělávací kartě s obvodem FPGA.

Klíčová slova

rekonfigurovatelné obvody FPGA, jazyk VHDL, sběrnice PCI, signály sběrnice PCI, řadič PCI, ovladače pro zařízení PCI,

Abstract

This thesis deals with the communication on the PCI bus and the design of controllers for the PCI card with FPGA circuit. The introduction shows the functionality and structure of FPGA circuits, followed by description of the principle of communication through the PCI bus. After an analysis of the PCI the thesis describes a design of controllers for a target card and lets the reader get acquainted with its different parts. In the process of implementation carefully examines the structure and operation of individual blocks of PCI controller. In the following part the thesis shows the process of implementation and testing of the final solution using the educational card with FPGA circuit.

Keywords

reconfigurable circuits FPGA, VHDL, PCI bus, PCI bus signals, PCI controller, PCI device drivers,

Citace

Ľubomír Ilavský: Řadič zběrnice PCI pro vývojovou kartu s obvodem FPGA, Brno,
FIT VUT v Brně, 2009

Řadič zběrnice PCI pro vývojovou kartu s obvodem FPGA

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Václava Šimka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……………………
Ľubomír Ilavský
22. 5. 2009

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu Ing. Václavovi Šimkovi za jeho odbornou pomoc při práci.

© Ľubomír Ilavský, 2009.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Obsah 1
1 Úvod 3
2 Konfigurovateľné obvody FPGA 4
2.1 Spartan II. 5
2.1.1 Štruktúra 5
2.2 VHDL 7
2.2.1 Popis obvodu pomocou VHDL 8
2.2.2 Syntéza 8
2.3 Vývojová karta Spartan – II PCI 11
3 Zbernica PCI 12
3.1 Komunikačný protokol PCI 13
3.2 Funkčné signály PCI 13
3.2.1 Systémové signály 14
3.2.2 Adresná a dátová zbernica 15
3.2.3 Kontrolné signály prenosu PCI 16
3.2.4 Arbitrážne signály 17
3.2.5 Signály pre požiadavky na prerušenie 17
3.2.6 Chybové signály 18
3.3 Príkazy PCI 18
3.4 Konfiguračný priestor 19
3.4.1 Povinné konfiguračné registre 19
3.4.2 Voliteľné konfiguračné registre 22
4 Návrh PCI radiča 23
4.1 Adresný dekodér 24
4.2 Príkazový dekodér 24
4.3 Blok pre výpočet parity 25
4.4 Arbiter funkcie TARGET zariadenia 25
4.5 Konfiguračné registre 27
4.6 Zhodnotenie návrhu 27
5 Čítací prenos 28
5.1.1 Analýza čítacieho prenosu 28
5.2 Popis implementácie čítacieho prenosu 31
5.2.1 Arbiter funkcie TARGET v čítacom prenose 31
5.2.2 Riadiaca činnosť arbitra 34

5.2.3 Implementácia arbitra vo VHDL 37
5.2.4 Adresný dekodér v čítacom prenose 38
5.2.5 Príkazový dekodér v čítacom prenose 39
6 Zapisovací prenos 41
6.1 Arbiter v zapisovacom prenose 44
6.2 Adresný dekodér v zapisovacom prenose 45
6.3 Blok pre výpočet parity 46
6.3.1 Čítanie zo zariadenia 46
6.3.2 Zápis do zariadenia 46
7 Konfiguračný prenos 47
7.1 Adresný priestor 47
7.2 Konfiguračné prenos typu NULA 48
7.2.1 Dátová fáza 48
7.3 Alokácia pamäti na PCI 49
7.4 Konfiguračná logika 51
7.5 Konfiguračný priestor zariadenia 51
8 Simulovanie a syntéza 53
8.1 Simulácia 53
8.2 Syntéza 54
8.3 Programovanie obvodu 55
9 Ovládač 56
9.1 Druhy ovládačov 56
9.2 Tvorba ovladača v prostredi Jungo WinDriver 58
10 Testovanie 61
10.1 Testovacia aplikácia 61
10.2 Test 62
11 Záver 63
Literatúra 65
Zoznam príloh 66

1 Úvod

Daná práca si kladie za svoj hlavný cieľ zoznámiť so zbernicou PCI a komunikačným protokolom na nej a priblížiť implementáciu radiča pre kartu s obvodom FPGA, pripojenú na túto zbernicu.

V prvej kapitole sa oboznámime s technológiou rekonfigurovateľných obvodov FPGA. Zameriame sa na spôsob ich rekonfigurácie, a vnútornú štruktúru týchto obvodov si priblížime na príklade obvodu Spartan II.. Čo je obvod, ktorý obsahuje karta ktorá je cieľovou platformou, pre navrhovaný radič. Ďalej sa v tejto kapitole oboznámime s jazykom VHDL, čo je jazyk pre popis číslicový obvodov a pomocou ktorého bude celý radič implementovaný. Na záver kapitoly si popíšeme vývojovú kartu *Spartan – II PCI*, ktorá je cieľovou platformou tohto projektu.

Ďalšia kapitola nám už priblíži PCI zbernicu, ako komunikačný prostriedok v hierarchií personálnych počítačov. Popíšeme si zaradenie tejto zbernice do tejto hierarchie, ukážeme si príklad komunikácie a popíšeme si potrebné signály pre riadenie komunikácie po tejto zbernici. Následne si popíšeme hierarchiu konfiguračného priestoru, ktorým musí disponovať každé zariadenie pripojené k zbernici PCI.

V nasledujúcej kapitole som popísal svoj návrh radiča PCI pre kartu s obvodom FPGA. Popisujem jednotlivé časti radiča, pričom som sa zameral na arbitra pre funkciu TARGET zariadenia. Jedna sa o najdôležitejšiu časť radiča, ktorá riadia celý proces komunikácie.

V jadre práce popisujem implementáciu navrhnutého PCI radiča. Rozoberám funkčnosť jednotlivých blokov a ich vzájomnú komunikáciu pomocou interných signálov radiča. V tejto časti sa hlavne zameriavam na implementáciu arbitra funkcie TARGET, ktorý riadi činnosť radiča. Popisujem jeho funkcie v procese čítacieho prenosu. V prípade zapisovacieho prenosu oboznamujem čitateľa s úpravami, ktoré som musel na arbitri vykonať, aby mohol riadiť aj túto činnosť radiča. Následne sa zaoberám druhou najdôležitejšou časťou radiča a to adresným dekodérom. Ten sa stará o zápis a čítanie dát z pamäte zariadenia a ich nastavovanie na AD zbernicu. V poslednom úseku jadra práce popisujem implementáciu konfiguračného prenosu a nastavenie hlavných konfiguračných registrov v konfiguračnom priestore zariadenia.

V poslednej časti zachytávam proces realizácie implementácie na cieľovom hardware, teda výukovej karte s obvodom FPGA SPARTAN II. a PCI konektorom. Popisujem proces syntézy VHDL kódu do konfiguračného súboru pre FPGA obvod a spôsob jeho nahrávania do pamäte zariadenia. Následne sa už venujem tvorbe ovládača pre výukovú kartu s PCI pripojením do počítača pre operačný systém Windows XP. Ten je dôležitý pre testovanie funkčnosti implementácie na karte zapojenej do PC zostavy s operačným systémom Windows XP. Testovanie a zhodnotenie dosiahnutých výsledkov uzatvára diplomovú prácu.

2 Konfigurovateľné obvody FPGA

Základným rysom rekonfigurovateľných zariadení je to, že je ich funkcia určená pomocou konfiguračného reťazca, ktorý je uložený v konfiguračnej pamäti, ktorá je obvykle súčasťou tohto zariadenia. Tento konfiguračný reťazec určuje funkciu jednotlivých konfigurovateľných elementov, ktoré sa nachádzajú v rekonfigurovateľnom zariadení, ich prepojenie, pripojenie k vstupom a výstupom obvodu a ďalšie vlastnosti rekonfigurovateľného zariadenia.

Najčastejšie jednotlivé bity konfiguračného reťazca priamo ovplyvňujú stav prepínačov (programovateľných prepojok, ktoré môžu byť realizované rôznymi spôsobmi). Pokiaľ je konfiguračná pamäť typu SRAM, je jej obsah možné obvykle meniť i behom činnosti rekonfigurovateľného zariadenia. Často existujú dve konfiguračné pamäte. Jedna definuje zapojenie aktuálne používaného obvodu, obsah druhej môže byť menený. Prehodením týchto pamätí (prepnutím kontextu) môže byť dosiahnutá rýchla rekonfigurácia celého systému. Dĺžka konfiguračného reťazca závisí na type rekonfigurovateľného zariadenia a pohybuje sa od stoviek bitov až po desiatky MB.

Konfiguračný reťazec sa vytvára najčastejšie pomocou vývojových nástrojov. Tento postup podrobnejšie popíšeme v kapitole 2.2. V niektorých prípadoch môže užívateľ konfiguračný reťazec bez pomoci vývojových nástrojov. Je však potrebné poznať formát konfiguračného reťazca. Dokumentácia však nie je obvykle výrobcom zverejnená. Spôsoby rekonfigurácie zariadení sú nasledovné [1]:

- **Statická rekonfigurácia** (static reconfiguration) je preprogramovanie obvodu behom jeho odstávky, keď obvod nevykonáva žiadne užitočné operácie. Typicky sa jedná o upgrade novej verzie HW.
- **Dynamická rekonfigurácia** (dynamic reconfiguration) je preprogramovanie obvodu v dobe, keď aspoň časť obvodu vykonáva užitočné operácie. Tu sa snažíme dynamicky (behom činnosti aplikácie) optimalizovať spôsob výpočtu, napr. s cieľom maximalizovať výkonnosť aplikácie.
- **Úplná rekonfigurácia** (partial reconfiguration) znamená, že meníme konfiguráciu celého obvodu naraz.
- **Čiastočná rekonfigurácia** (partial reconfiguration) znamená, že meníme konfiguráciu len určitej časti obvodu, bez toho aby sme zmenili funkciu ostatných časti obvodu. Podpora čiastočnej rekonfigurácie je dôležitá pre realizáciu aplikácií využívajúcich dynamickú rekonfiguráciu.
- **Interná rekonfigurácia** (internal reconfiguration) označuje situáciu, keď je rekonfigurácia riadenia zariadením, ktoré sa nachádza na rovnakom čipe ako rekonfigurovateľné zariadenie. Potom môže jedna skupina konfigurovateľných elementov meniť konfiguráciu inej skupiny konfigurovateľných elementov.

- **Externá rekonfigurácia** (external reconfiguration) označujeme situáciu, keď nejaké zariadenie, ktoré je umiestnené mimo rekonfigurovateľného zariadenia, mení konfiguráciu rekonfigurovateľného zariadenia.

Pre istú triedu úloh, ako napr. spracovanie signálov, kompresiu dát, šifrovanie, vyhľadávanie a pod. je ich realizácia pomocou rekonfigurovateľných hardwarových štruktúr v porovnaní s univerzálnym výpočtovými architektúrami (procesormi) výrazne výkonnejšia (10 – 100 krát). Taktiež výrazne klesá príkon a cena výsledného zariadenia.

FPGA (Field Programmable Gate Array – Programovateľné logické polia) boli zavedené firmou Xilix v roku 1984. Boli navrhnuté tak, aby mohli byť konfigurovateľné pri spustení systému privedením napájacieho napätia a umožňovali návrhárovi vylepšiť funkciu systému (opraviť chyby v návrhu) po tom, čo boli dodané užívateľovi. Tento fakt, spoločne s nižšou cenou v porovnaní s riešením využívajúce ASIC, viedol k ich úspešnému nasadeniu behom posledných 20 rokov. Ďalším faktorom, ktorý prispel k ich popularite v neskorších rokoch, bolo umožnenie dynamickej rekonfigurácie – možnosť zmeny HW behom činnosti aplikácie. V dnešnej dobe existuje rada výrobcov ponúkajúcich FPGA (Xilinx, Lattice Semiconductor, Altera, Atmel, Actel atd.). Firma Xilinx však na trhu stále dominuje. Architektúru FPGA si popíšeme na obvode Spartan II., na budeme implementovať danú úlohu.

2.1 Spartan II.

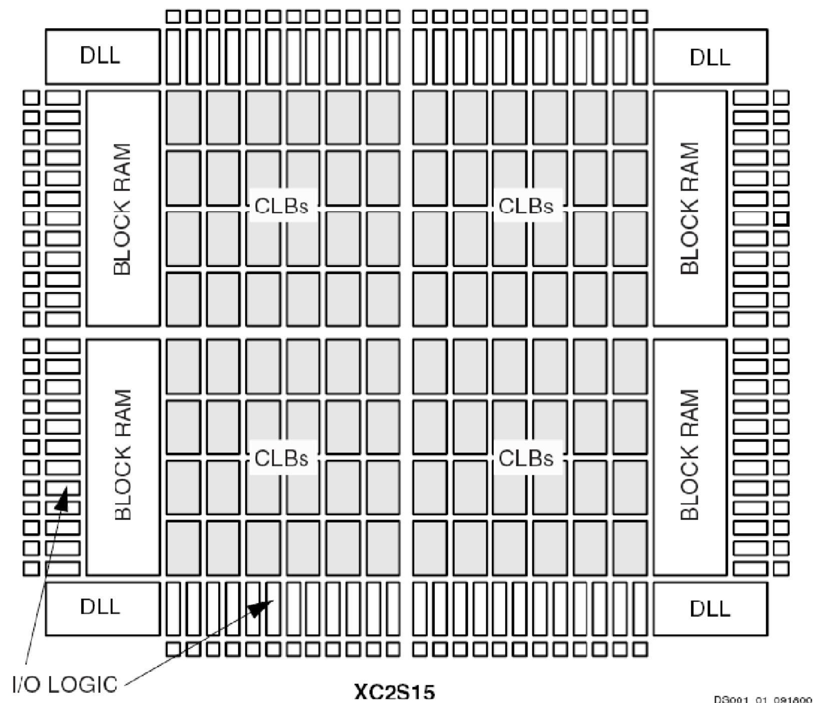
FPGA obvody rodiny Spartan II. sú vyrobené firmou Xilinx a boli uvedené v roku 2003. Jeden z členov tejto rodiny FPGA rekonfigurovateľných obvod obsahuje karta, ktorá je cieľovou platformou pre ktorú je vyvíjaný PCI radič, čo je hlavnou úlohou tejto práce. Preto si architektúru FPGA obvodov priblížime na tomto konkrétnom obvode.

Cely návrh radiča PCI sa potom bude odvíjať od toho cieľového čipu FPGA. Ostatné obvody FPGA môžu obsahovať i iné súčasti ako Spartan II., ako napríklad vstavané procesory, deličky, prídavné pamäťové bloky a pod. Vo svojej podstate však zachovávajú tú istú základnú štruktúru.

2.1.1 Štruktúra

Rekonfigurovateľný obvod Spartan II. sa skladá za nasledujúcich blokov [2].

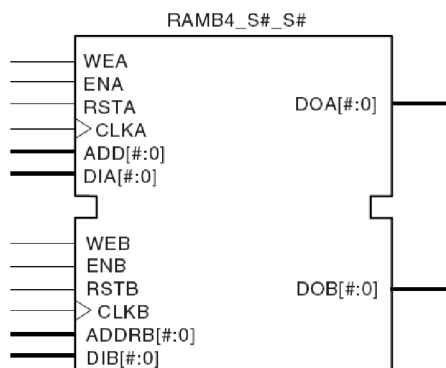
- CLB (rekonfigurovateľné logické bloky)
- Block Ram (blokové pamäte typu RAM)
- DLL bloky (Delay-Locked Loop – obvody zodpovedné za distribúciu hodinového signálu)
- I/O Logic (vstupno výstupné logické obvody)



Obr. č. 1. Štruktúra FPGA obvodu Spartan II. [2]

CLB (konfigurovateľné logické bloky) sú srdcom FPGA čipu. Každý CLB v obvode Spartan II. obsahuje dva obvody SLICE, čo sú menšie logické elementy. Ďalej CLB obsahuje rýchle pripojenie k susedným členom a pripojenie ku globálnej prepojovacej matici. V každom SLICE sa nachádzajú dva LUT (Look Up Table) obvody, ktoré môžu implementovať ľubovoľnú logickú funkciu a sú nastaviteľné konfiguračným reťazcom. Ďalej sú tu dve jednotky Carry a kontrolnej logiky, dva Latch registre a dva multiplexory MUXF5.

BLOCK RAM (blokové pamäte typu RAM) podporujú synchrónne čítanie i zápis. Sú organizované vo zväzkoch a sú v dvoj-portovom prevedení. Veľkosť pamäťových blokov závisí od jednotlivých členov rodiny Spartan II.



Obr. č. 2. Dvoj-portový **Block Ram** obvodu Spartan II. [2]

DLL bloky sú pripojené k I/O jednotkám. Slúžia na elimináciu asymetrií medzi vstupným hodinovým signálom a interným hodinovým signálom. DLL monitorujú vstupné a distribuované hodiny po čipe, a automaticky nastavujú hodinový oneskorovací prvok. Ďalšou funkciou ktoré realizujú obvody DLL okrem redukcie oneskorenia je frekvenčná syntéza. Jedná sa o násobenie a delenie hodinového signálu a tým jeho úpravu na požadovanú frekvenciu. Poslednou funkciou, ktorú DLL obvody umožňujú je fázový posun. Či už fixný alebo premenlivý.

I/O Logické členy sú pripojené ku každému pinu FPGA čipu. Umožňujú nakonfigurovať každý pin ako vstupný, výstupný alebo vstupno-výstupný. Podporujú jednotlivé vodiče i diferenciálne páry vodičov (väčšia odolnosť voči vonkajšiemu rušeniu). Ďalej musia podporovať širokú škálu technologických štandardov ako LVTTL, LVCMOS, PCI atď. I/O bloky musia okrem vstupno-výstupných jednotiek obsahovať podporné štruktúry pre pripojenie trojstavových zberníc(tri stavy – prijímanie dát, odosielanie dát a stav vysokej impedancie).

2.2 VHDL

VHDL jazyk patri do skupiny HDL(Hardware Description Language) jazykov. Jedná sa o jazyky pre popis logických obvodov. Tie vznikli ako odozva na stále sa zväčšujúcu zložitosť logických obvodov. Pre popis týchto obvodov schémy zapojenia už stratili svoje opodstatnenie, kvôli neprehľadnosti a značnej zložitosti. Pri návrhu pomocou HDL jazyka je možné obvod modelovať a simulovať, čo prispieva k zrýchleniu návrhu a rýchlemu odhaľovaniu chýb.

V praxi sa využívajú najmä dva jazyky tejto rodiny a to VHDL a Verilog. VHDL dominuje v Európe a Verilog v USA. Extrémny nárast počtu hradiel vedie k zavedeniu abstraktnejších jazykov rodiny HDL a to SystemC, HandelC, ImpulseC a ďalších. Čo prináša výrazné urýchlenie vývoja nových zariadení.

Názov VHDL pochádza z akronymu *VHSIC Hardware Description Language*. Kde VHSIC je skrátka pre *Very High Speed Integrated Circuit*. Pôvodne bol vyvinutý pre vojenské účely ku špecifikácií číslicových systémov. No v roku 1987 bol uznaný IEEE štandardom (IEEE Standard 1076-1987). Ako jazyk nie je zviazaný so žiadnou cieľovou technológiu. V dnešnej dobe existuje veľa nástrojov umožňujúcich syntézu alebo simuláciu obvodov popísaných jazykom VHDL.

Výhody jazyka VHDL sú [3]:

- je všeobecne prístupný
- podporuje rôzne návrhové metodológie a technológie
- je nezávislý na technológii
- poskytujú široké opisné možnosti
- umožňuje nezávislý návrh subsystémov
- podporuje návrh rozsiahlych systémov a opätovné použitie návrhu

2.2.1 Popis obvodu pomocou VHDL

Jazyk VHDL umožňuje tri druhy popisu obvodu [3]:

- **Behaviorálny popis** obsahuje procesy opísané na sekvenčnej úrovni, ktoré umožňujú popísať výstupy obvodu v diskrétnych okamihoch daných konkrétnymi vstupmi. Opis sekvenčných aj kombinačných obvodov na všetkých úrovniach abstrakcie.
- **Štrukturálny popis** opisuje funkciu obvodu spravidla pomocou externých komponentov, t.j. aké komponenty obsahuje a ako sú prepojené. Komponenty pritom môžu byť definované užívateľom alebo z knižníc dodávaných výrobcami obvodov.
- **Data-flow popis** je najmenej používaný. Modeluje obvod z hľadiska pohybu údajov v spojitom čase medzi komponentmi kombinačnej logiky (sčítačky, dekodéry, primitívne hradlá). Opis na úrovni RTL a hradiel - kombinačných obvodov.

V praxi sa používajú najmä prvé dve metódy popisu obvodu. Spravidla sa najprv popíšu jednotlivé obvody behaviorálne a následne sa štrukturálne prepoja do jedného celku.

Jednotlivé číslicové zariadenia VHDL popisuje pomocou tzv. **komponent**. Tá môže byť popísaná pomocou behaviorálneho ale štrukturálneho popisu. Komponenta pozostáva z dvoch častí. A to z entity a architektúry:

- **entita** popisuje rozhranie medzi komponentov a okolím. Skladá sa z signálov rozhrania a generických parametrov. Signály rozhrania môžu byť podľa smeru šírenia dát v móde IN, OUT a INOUT.
- **architektúra** definuje vnútornú funkciu komponenty. Je vždy zviazaná s entitou ktorá definuje iteráciu s okolím. Architektúra pozostáva z deklaračnej časti a sekcie paralelných príkazov. Deklaračná časť je vyhradená pre deklaráciu signálov, konštánt alebo typov použitých vnútri architektúry. V sekcie paralelných príkazov už popisujeme chovanie samotnej komponenty. Jej súčasťou môžu byť aj inštancie komponent alebo proces vzájomne prepojené signálmi.

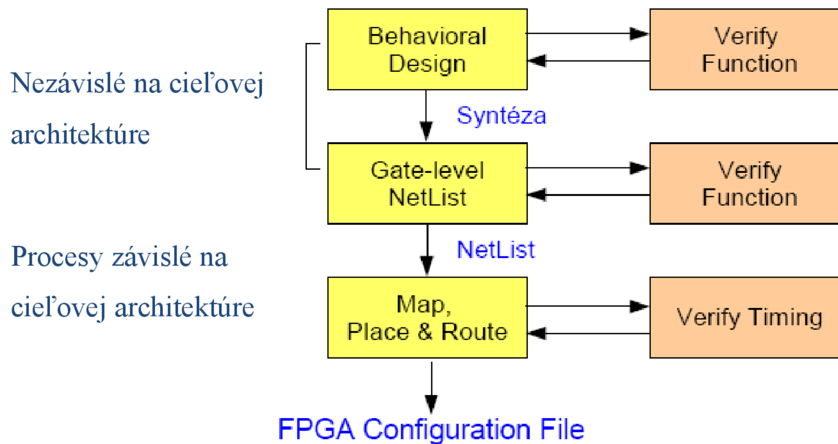
Po navrhnutí číslicového obvodu pomocou jazyka VHDL a odsimulovaní jeho funkčnosti pomocou simulačných nástroj, sa pristupuje k mapovaniu tohto popisu na cieľovú hardwarovú architektúru. Tento proces sa nazýva syntéza a priblížime si ho v nasledujúcej podkapitole.

2.2.2 Syntéza

Pod pojmom syntéza rozumieme transformáciu medzi rôznymi úrovňami popisu číslicového obvodu. Pri tejto transformácie je cieľom nielen zachovať základnú funkčnosť obvodu, ale aj vylepšiť parametre zadané užívateľom. Ako napríklad rýchlosť, spotreba, testovateľnosť atď. A taktiež je

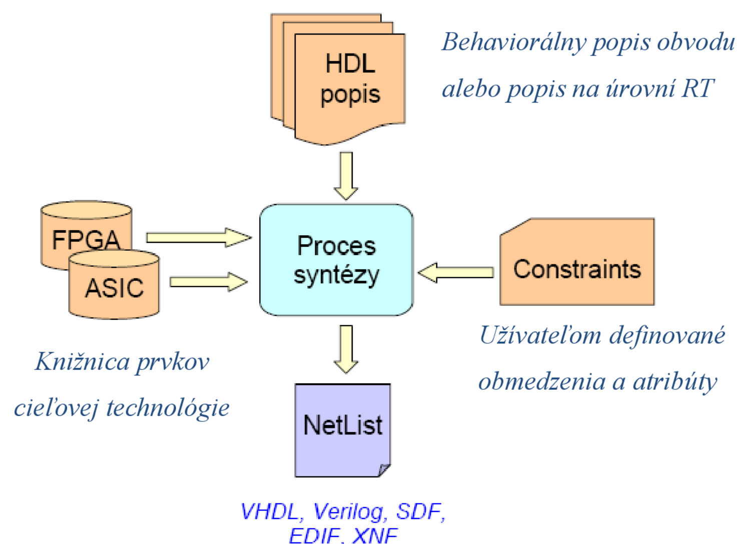
potrebné aby boli splnené požiadavkou, ktoré sú špecifikované užívateľom. Jedná sa o tzv. vstupné obmedzenia. Napríklad požadovaná frekvencia hodín, plocha na čipe atď.

Syntéza sa líši na základe cieľovej architektúry. Vyplýva to už so samotných rozdielov týchto architektúr(ASIC, FPGA). Ja tu popíšem syntézu na cieľovú architektúru FPGA nakoľko je to aj cieľová architektúra tohto projektu.



Obr. č. 3. Syntéza na cieľovú platformu FPGA [4]

Na obrázku č. 3. je znázornený proces na cieľovú architektúru FPGA. V každom bode procesu je potrebné verifikovať správnu funkčnosť popisu obvodu. Prvý krok syntézy, prevod na popis pomocou Gate-level NetListu, je platformovo nezávislí a prebieha pri syntéze na všetky cieľové platformy. V ďalšom kroku dochádza už k mapovaniu obvodu na prvky cieľovej architektúry a tým pádom sa jedna o čisto platformovo závislí proces.



Obr. č. 4. Vstupy a výstupy syntézy [4]

Na obrázku č. 4. sú zachytené vstupy a výstupy procesu syntézy. Hlavných vstupom je popis obvodu pomocou HDL jazyka. Ďalšími potrebnými vstupmi sú obmedzujúce podmienky, ktoré zadáva užívateľ. Môže sa jednať o požadovanú frekvenciu hodín, maximálna veľkosť spotrebovaného miesta na čipe a pod. Posledným nutným vstupom procesu syntézy je knižnica prvkov cieľovej architektúry. Jedná sa o presný popis použiteľných zdrojov ako aj spôsob ich využitia.

2.2.2.1 Fázy syntézy

Proces syntézy prebieha typicky v troch základných fázach [4]:

- prevod RTL popisu do booleovskej logiky
- booleovská optimalizácia
- mapovanie na hradlá cieľovej architektúry

Vo fáze booleovskej optimalizácie sú použité algoritmy pre optimalizáciu časovej náročnosti alebo priestorovej náročnosti. V základe idú tieto dva pohľady na optimalizáciu proti sebe. Keď sa budeme snažiť o maximálnu rýchlosť obvodu zákonite to povedie k paralelizácií obvodu a tým zabratiu veľkého množstva zdrojov. A naopak keď sa budeme snažiť o čo najmenšie využite priestoru na čipe, budem musieť v jednotlivých časových krokov využívať tie isté hardwarové zdroje, čo zákonite povedie k spomaleniu obvodu. Pre dosiahnutie maximálnej rýchlosti obvodu sa používajú algoritmy ako napríklad algoritmus **Flattening**, pri ktorom dochádza k prevodu booleovských funkcií na úplnú disjunktnú formu. Pri použití tohto algoritmu dochádza k problémom s veľkým dizajnom. Z toho vzniká problém z názvom *fanout* čo je vlastne nerovnomerná dĺžka použitých vodičov a tým aj nerovnomerné oneskorenie. Tým pádom je problém celý obvod synchronizovať na rovnakú frekvenciu. S opačným cieľom sa používa algoritmus **Factoring**, pri ktorom dochádza k redukcií logiky zdieľaním spoločných termov.

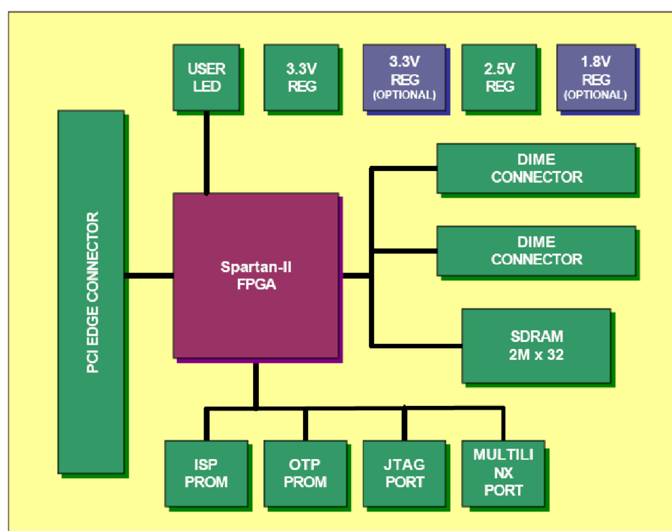
V pokročilejších metódach syntézy existuje mnoho algoritmov, pri ktorých sa popis obvodu prevádza do tzv. **CDFG** (Control Data Flow Grafu), ktorý reprezentuje operácie daného obvodu a ich časové závislosti. Nad CDFG sa následne vykonávajú optimalizácie s ohľadom na časové alebo priestorové obmedzenia.

Proces syntézy však dokáže pracovať len s podmnožinou jazyka VHDL. Medzi problematický syntetizovateľné konštrukcie jazyk VHDL patrí čítanie alebo zápis do súboru, nekonštantný počet iterácií, príkaz *wait* a príkazy pre overovanie funkčnosti komponent ako *assert* a *report*. Preto došlo k zavedeniu štandardu IEEE 1076.6 – 1999, ktorý zavádza syntax a sémantiku VHDL jazyka pre RT syntézu.

2.3 Vývojová karta Spartan – II PCI

Vývojový karta Spartan – II PCI je cieľovou platformou tohto projektu. Táto karta poskytuje jednoduché prostredie pre vývoj a testovanie FPGA aplikácií založených na PCI dizajne. Celá karta bola vytvorená zo zámerom pomôcť vývojárom v implementácií PCI jadra a rozhrania v Xilinx FPGA. Srdcom tejto vývojovej karty je FPGA rekonfigurovateľný obvod Spartan II, ktorého architektúru sme si priblížili v podkapitole 2.1.

Spartan II. plne podporuje implementáciu 32 bitového, 33 MHz PCI jadra pre komunikáciu zo strany INITIATORA a TARGET zariadenia. Pritom sa predpokladá, že samotný radič by mal zabrať niečo cez 10 % plochy na čipe. Tak ostane približne 90 % čipu voľného pre implementáciu užívateľskej aplikácie. Okrem rekonfigurovateľného FPGA obvodu Spartan II. karta obsahuje obvod Xilinx XC18V01 ISP PROM, dovoľujúci konštruktérovi opakovane sťahovať a ukladať nové revízie dizajnu.



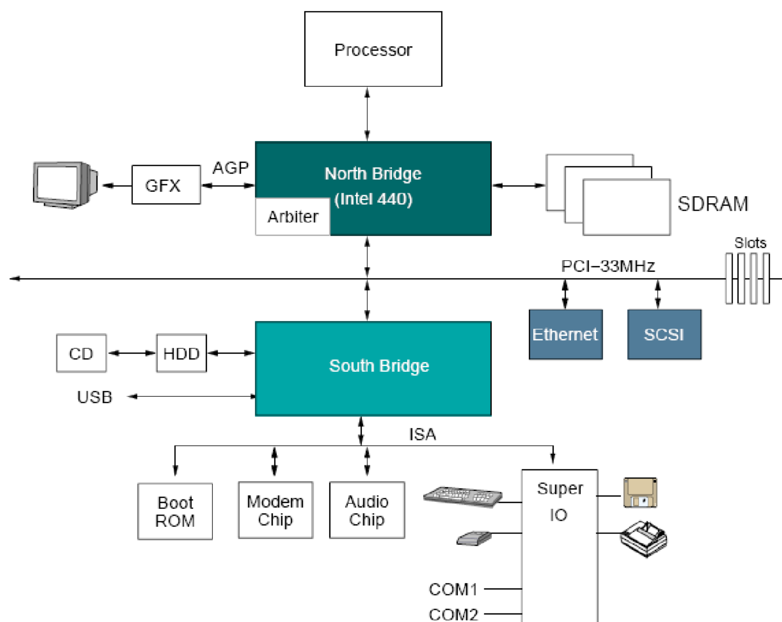
Obr. č. 5. Vstupy a výstupy syntézy [5]

Obrázok č. 5. zobrazuje blokovú schému vývojovej karty. Hlavnou časťou je už spomínaný FPGA obvod Spartan II., ktorý je pripojený k PCI konektoru. Okrem neho sa tu nachádzajú nasledujúce zariadenia:

- 2,5 V a 3,3 V regulátor
- XC18V01 ICP PROM
- SPROM soket
- JTAG Port
- MultiLINX Seriálový Port
- 64 Mb SDRAM

3 Zbernica PCI

Zbernice typu PCI tvoria v súčasnej dobe štandard pre pripojenie prídavných adaptérov a periférnych zariadení v rámci architektúry personálnych počítačov. Ich vývoj je koordinovaný organizáciu PCI-SIG (Peripheral Component Interconnect Special Interest Group), ktorá vydáva špecifikácie zberníc od fyzickej úrovne až po podrobný popis komunikačného protokolu.



Obr. č. 6. Model architektúry založenej na PCI [6]

Na obrázku č. 6 je zobrazený model architektúry pre systémy so zbernicou PCI. Architektúra je rozdelená do niekoľkých častí na základe toho, akú priepustnosť jednotlivé časti pre svoju funkciu potrebujú. Zariadenia vyžadujúce vysokú priepustnosť dát sú prepojené cez tzv. *North bridge*. Tu patrí predovšetkým procesor, systémová pamäť a grafický adaptér.

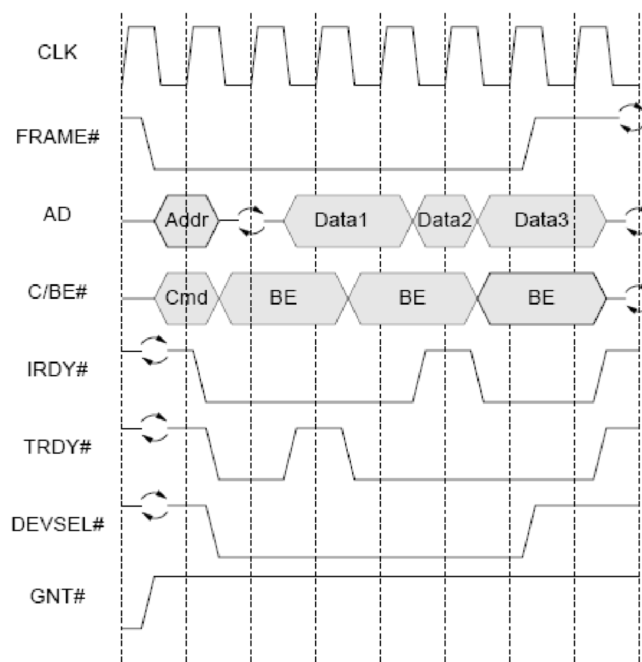
Aby bolo možné komunikovať s periférnymi zariadeniami je North bridge tiež pripojený na zbernicu PCI. Tu sa predpokladá pripojenie adaptérov s nižšími požiadavkami na priepustnosť(napr. sieťový adaptér, zvuková karta. atď). A zariadenia, ktoré pre svoju funkčnosť potrebujú nízku prenosovú kapacitu (myš, klávesnica), sú pripojené cez tzv. South bridge. V celkovom modely tak môže South bridgde vystupovať ako jedno z PCI zariadení. Je dôležité poznamenať, že sa jedná len o model. Reálne architektúry sa od tohto modelu môžu líšiť. Najmä implementácia North a South bridge je obvykle realizovaná v rámci jednej čipovej sady, kde skutočná realizácia a rozvrstvenie závisí na konkrétnom výrobcovi.

3.1 Komunikačný protokol PCI

Pokiaľ chcú ľubovoľné dve zariadenia na zbernici PCI komunikovať, musia dodržať definovaný protokol, Zariadenie, ktoré komunikáciu inicializuje je označované ako *INITIATOR*. Zariadenie, s ktorým je komunikácia naviazaná je označované ako *TARGET*(cieľ).

Komunikácia prebieha v nasledujúcich krokoch[5]:

1. INITIATOR vystaví na zbernicu AD adresu zariadenia, s ktorým chce komunikovať. Paralelne s adresou vystaví príkaz(command), ktorý špecifikuje typ transakcie a identifikáciu, či sa bude jednať o čítaciu alebo zapisovaciu operáciu.
2. Pokiaľ TARGET rozpozná na zbernici svoju adresu, aktivuje signál DEVSEL.
3. Začína sa prenos dát a na zbernici AD sú postupne vystavované dáta. Komunikácia môže byť v priebehu pozastavená ako zo strany IINITIATORU (signálom IRDY), tak zo strany TARGET (signálom TRDY).
4. Koniec prenosu je signalizovaný deaktiváciou signálu FRAME.

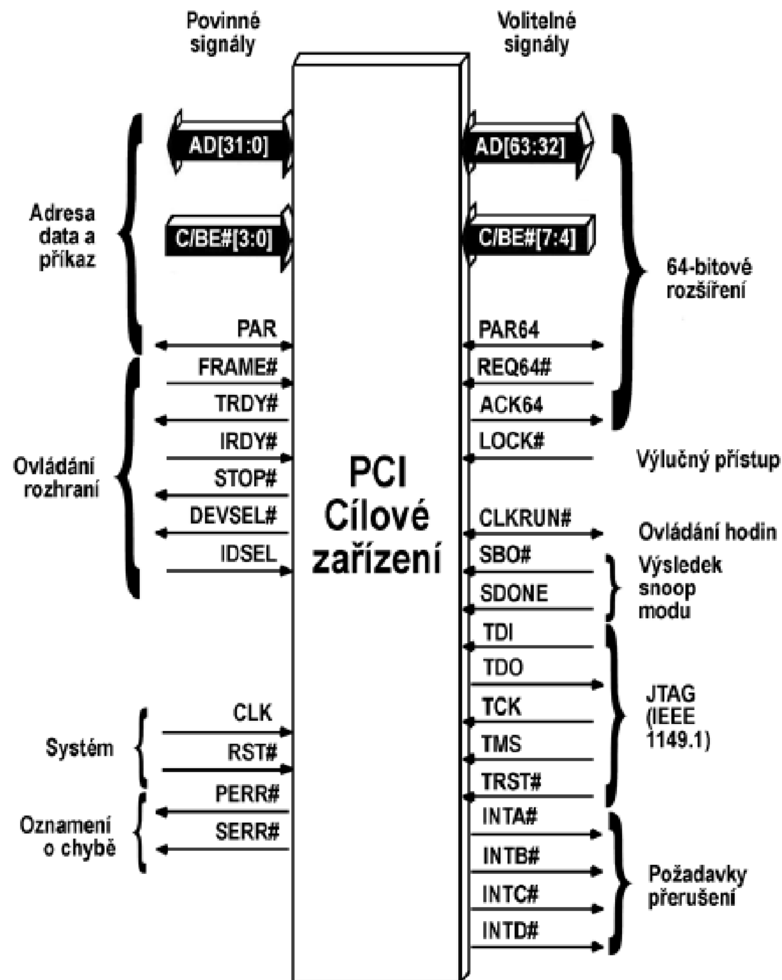


Obr. č. 7. Príklad komunikácie na zbernici PCI [6]

3.2 Funkčné signály PCI

V tejto kapitole si popíšeme PCI zbernicové signály. Obrázok č. 8. Ukazuje potrebné a voliteľné signály pre INICIATOR a TARGET zariadenie. PCI zariadenie, ktoré sa môže pri prenose správať ako INICIATOR má zabudované oba druhy signálov. V skutočnosti neexistuje zariadenie, ktoré sa

chová len ako INICIATOR a nikdy nie len ako TARGET. Minimálne sa zariadenie môže chovať ako TARGET s konfiguračným čítaním a zápisom.



Obr. č. 8. PCI signály [7]

3.2.1 Systémové signály

3.2.1.1 Signál PCI hodín (CLK)

CLK je vstupným signálom do všetkých PCI zariadení umiestnených na PCI zbernici. To zaisťuje časovanie pre všetky prenosy, vrátane arbitráže. Všetky vstupy do PCI zariadenia sú vzorkované s nástupnou hranou hodinového signálu CLK. Stavy všetkých vstupných signálov nie sú v ďalšom časovom okamihu sledované.

Všetky deje na PCI zbernici sú synchronizované PCI signálom CLK. Frekvencia signálu CLK môže byť v intervale 0 MHz až 33 MHz. Verzie 1.0 PCI špecifikácie uvádza, že všetky zariadenia musia podporovať činnosť od 16 do 33 MHz a striktne odporúča podporu pre operácie okolo 0 MHz pre statické krokovanie a málo výkonné operácie.

Frekvencia hodín sa môže meniť kedykoľvek pokiaľ [7]:

- hrana hodín zostáva bez chyby (edge remains clean)

- je zachovaná minimálna vzdialenosť vysokej a nízkej logickej úrovne signálu CLK
- LOCK# nie je nastavený

Signál CLK môže byť zastavený len v stave log. nuly (z dôvodu výkonu).

3.2.1.2 Signál Reset (RST#)

Pokiaľ je aktivovaný, tak signál prinúti všetky PCI konfiguračné registre, INICIÁTORA, TAFGET zariadenia, stavový automat a výstupné zariadenia prejsť do inicializačného stavu. RST# môže byť aktivovaný alebo deaktivovaný asynchrónne s PCI CLK hranou. Všetky výstupné signály musia byť riadené od počiatku.

Výnimku tvoria:

- SERR# ja plávajúci
- pokiaľ SBO# a SDONE nemôžu byť trojstavové, budú riadené od nuly
- zabezpečenie AD zbernice, C/BE zbernice a PAR signálov prechodu s plávajúceho stavu počas resetu musí byť riadený od nuly centrálnym prostriedkom počas resetu

3.2.2 Adresná a dátová zbernica

PCI zbernica požíva časove multiplexovanú adresnú a dátovú zbernicu.

Počas adresnej fázy prenosu:

- AD zbernica, AD[31:0] prenáša štartovaciu adresu. Adresa sa rozprestiera v hraničnom dvoj slove (adresa je deliteľná štyrmi) počas pamäťového alebo konfiguračného prístupu, alebo adresa je bytovo špecifikovaná (byte specifik adress) počas I/O čítacej a zapisovacej transakcie.
- Príkazová a BYTE ENABLE zbernica, C/BE[3:0], definuje typ prenosu.
- Paritný signál PAR je riadený INICIÁTOROM jednu periódu po dokončení adresnej fázy a po každej dátovej fáze zapisovacej transakcie. Je taktiež riadený práve adresovaným TARGET zariadením jednu periódu po dokončení dátovej fáze a po každom dokončení čítacej transakcie. Jednu periódu po dokončení adresnej fáze INICIÁTOR riadi PAR medzi log. jednotkou a nulou pre zaistenie párnej parity na adresnej zbernici AD[31:0], a štyroch vodičov C/BE[3:0].

Počas každej dátovej fázy:

- Dátová zbernica AD[31:0] je riadená INICIÁTOROM (počas zápisu) alebo TARGET zariadením (počas čítania).
- PAR signál riadený oboma účastníkmi. INICIÁTOROM počas zápisu a TARGET zariadením počas čítania jednu periódu po dokončení dátovej fázy pre zistenie párnej parity. Pokiaľ nie sú využité všetky štyri dátové cesty, musí účastník riadiaci dátovú

zbernicu zaistiť, aby boli platné dáta nastavené na všetky dátové cesty. Je to nutné, nakoľko PAR musí odrážať párnu paritu z AD aj C/BE# zbernice.

- Príkazová zbernica C/BE#[3:0] je riadená INICIÁTOROM k indikácií koľko a akých bitov sa bude týkať prenos v práve adresovanom dvoj slove. Tabuľka č. 1. Znázorňuje mapovanie signálov BYTE ENABLE dátových ciest a pozícií v danom dvoj slove. Každá kombinácia BYTE ENABLE sa berie v úvahu jej hodnota sa môže meniť medzi jednotlivými dátovými fázami.

Sig. BYTE ENABLE	Mapovanie
C/BE3#	Dátová cesta 3, AD[31:24], štvrtá pozícia v práve adresovanom dvoj slove
C/BE2#	Dátová cesta 2, AD[23:16], tretia pozícia v práve adresovanom dvoj slove
C/BE1#	Dátová cesta 1, AD[15:8], druhá pozícia v práve adresovanom dvoj slove
C/BE0#	Dátová cesta 0, AD[7:0], prvá pozícia v práve adresovanom dvoj slove

Tab. č. 1. Mapovanie signálu BYTE ENABLE [7]

3.2.3 Kontrolné signály prenosu PCI

FRAME# (INICIATOR–IN/OUT, TARGET-IN) – je riadení INICIÁTOROM vlastniacom zbernicu a indikuje počiatok (t.j. kedy je nastavený prvý krát) a dobu trvania prenosu (doba jeho nastavenia). Ďalej indikuje, kedy je zbernica predaná do vlastníctva. INICIÁTOR musí oba signály (FRAME#, IRDY#) vzorkovať zároveň s nástupnou hranou CLK hodín. Transakcia môže obsahovať jednu alebo viacero dátových fázy medzi INICIÁTOROM a práve adresovaným TARGET zariadením. Signál FRAME# je deaktivovaný, keď je INICIÁTOR pripravený dokončiť poslednú dátovú fázu.

TRDY# (INICIATOR–IN, TARGET-OUT) – Značí, že je cieľové zariadenie pripravené. Tento signál je nastavovaný práve adresovaným TARGET zariadením. Je nastavený pokiaľ je TARGET zariadenie pripravené k dokončeniu prebiehajúceho dátového prenosu. Dátová fáza je dokončená, keď TARGET aktivuje TRDY# a INICIÁTOR aktivuje IRDY# s nástupnou hranou CKL hodín. Počas čítania aktívny signál TRDY# indikuje, že TARGET zariadenie nastavilo platné dáta na AD zbernicu. Počas zápisu aktívny signál TRDY# indikuje, že TARGET zariadenie akceptuje dáta od INICIÁTORA. Čakací stav je vložený do prebiehajúcej dátovej výmeny, pokiaľ je TRDY# aj IRDY# nastavení.

IRDY# (INICIATOR–IN/OUT, TARGET-IN) – Značí, že INICIÁTOR je pripravený. Tento signál je riadený aktívnym INICIAÁTOROM. Počas zápisu aktívny IRDY# indikuje, že INICIÁTOR nastavil platné dáta na AD zbernicu. Počas čítania aktívny IRDY# indikuje, že je pripravený akceptovať dáta

od TARGET zariadenia. Ďalej indikuje, že vlastník bol pridelený. INICIÁTOR musí vzorkovať neaktívny FRAME# a IRDY# s každou nebežnou hranou PCI hodín.
STOP# (INICIATOR–IN, TARGET-OUT) – TARGET zariadenie aktivuje STOP# k indikácií, že INICIATOR požaduje zastavenie prenosu v počas prebiehajúceho dátového prenosu.

IDSEL# (INICIATOR–IN, TARGET-IN) – Inicializácia vybraného zariadenie.

LOCK# (INICIATOR–IN/OUT, TARGET-IN) – je používaný INICIÁTOROM k uzamknutiu práve adresovanej pamäti TARGET zariadenia, počas automatickej transakcie (t.j. počas semaforových operácií ako čítanie, úprava, zápis).

DEVSEL# (INICIATOR–IN, TARGET-OUT) – Signál značí, že zariadenie je vybrané. Signál je nastavený TARGET zariadením. Jedná sa o vstup do aktívneho INICIÁTORA a o odpočítavajúci dekodér v rozšírenom zbernicovom moste. Pokiaľ INICIÁTOR inicializuje transakciu a nedetekuje aktívny DEVSESL# po šiestich periódach, musí predpokladať, že TARGET zariadenie nemôže odpovedať, alebo že adresa nie je povolená. Výsledkom je ABORT INICIÁTORA.

3.2.4 Arbitrážne signály

Každý INICIÁTOR má dva arbitrážne vodiče, ktoré ho prepojujú priamo so zbernicovým arbitrom. Pokiaľ INICIÁTOR potrebuje použiť PCI zbernicu, aktivuje svoj **REQ#** signál k arbitrovi. Pokiaľ arbiter zistí, že požiadavka INICIÁTORA bola vybavená, aktivuje signál **GNT#** (grant) [6]. Na PCI zbernici sa arbitráž môže uskutočniť iným INICIÁTOROM, ktorý stále kontroluje zbernicu. Takýto prípad sa nazýva **skrytá arbitráž**.

Pokiaľ INICIÁTOR dostane signál o pridelení zbernice z arbitra, musí čakať na aktívneho INICIÁTORA, pokiaľ nepredá zbernicu do čakacieho stavu a potom môže inicializovať vlastný prenos. To je splnené až potom, čo je signál FRAME# (indikuje počiatok poslednej fázy) neaktívny a po ďalšej perióde aj signál IRDY# (indikuje dokončenie poslednej dátovej fázy) prejde do neaktívneho stavu. Tento stav signálov naznačuje, že práve prebiehajúci prenos je dokončený a zbernica sa vrátila do čakacieho stavu.

Pokiaľ je nastavený RST# signál, všetci INICIÁTORI musia mať svoj výstup REQ# nastavený v treťom stave a musí ignorovať GNT# vstupy.

3.2.5 Signály pre požiadavky na prerušenie

PCI účastník, ktorý musí generovať požiadavky na služby, môže využiť jeden z vodičov prerušenia: INTA#, INTB#, INTC# alebo INTD#.

3.2.6 Chybové signály

Do tejto skupiny patria dva signály a to **PERR#** a **SERR#**. PCI zbernica je zabezpečená v adresných a dátových fázach prenosu dátovou paritou. Bit párnej parity **PAR** zabezpečuje AD[31:0] a C/BE[3:0] zbernice. PCI zariadenie, ktoré riadi AD zbernicu počas adresnej fázy alebo počas každej dátovej fázy zodpovedá za výpočet a potvrdenie bitu parity pre každú fázu. Nastavuje sa jeden cyklus po adresnej alebo dátovej fáze. Výpočet bitu parity, ktorý má byť nastavený na **PAR** signál, musí byť nastavený, alebo deaktivovaný tak, aby pole 37 bitov zloženého z adresnej zbernice AD[31:0] a príkazovej C/BE[3:0] a PARR# signálu obsahovalo práve párny počet jednotkových bitov.

Zaistenie správnej parity umožňuje každé PCI zariadenie. Všetky PCI zariadenia musia generovať párnu paritu AD, C/BE a PAR pre adresnú a dátovú fázu. Signál **PERR#** je OUT signálom TARGET zariadenia a IN/OUT signálom INICIÁTORA. INICIÁTOR prenosu zodpovedá za detekciu programovej chybovej dátovej parity. Z tohto dôvodu musí sledovať signál PERR# počas zápisovej dátovej fázy k zisteniu, kedy TARGET zariadenie detekovalo chybnú paritu.

SERR# signál sa používa pre oznámenie vážnych chýb. Malé opraviteľné chyby môžu byť signalizované inou cestou. Signál systémovej chyby SERR# môže byť nastavovaný PCI zariadením, ktoré oznamuje chybnú paritu adresy, chybnú paritu dát počas špeciálneho cyklu a kritické chyby ďalších parít. SERR# je nutný pre všetky prídavné karty PCI, ktoré vykonávajú kontrolu adresnej parity alebo sledujú ďalšie vážne chyby nastavením SERR# signálu.

3.3 Príkazy PCI

Keď je INICIÁTOROVI pridelená zbernica, môže inicializovať jeden typ prenosu z tabuľky č. 2.. Počas adresnej fázy prenosu je C/BE zbernica použitá k indikácii príkazu alebo k definícií typu prenosu. Tabuľka č. 2. definuje kombinácie, ktoré sú nastavované INICIÁTOROM počas adresnej fázy.

I/O príkazy čítania a zápisu sú používané k prenosu dát medzi INICIÁTOROM a práve adresovaným I/O zariadením. TARGET zariadenie musí dekódovať vstupnú 32 bitovú adresu.

PCI špecifikácia definuje päť príkazov používajúcich prístup k pamäti [7]:

1. príkaz čítania pamäti
2. príkaz riadkového čítania pamäti
3. príkaz viacnásobného čítania
4. zápis do pamäti
5. príkaz zápisu a zrušenia platnosti (*write and invalidate*)

C/BE[3:0]	typ príkazu
0000	potvrdenie prerušenia
0001	špeciálny cyklus
0010	I/O čítanie
0011	I/O zápis
0100	vyhradené
0101	vyhradené
0110	čítanie z pamäti
0111	zápis do pamäti
1000	vyhradené
1001	vyhradené
1010	konfiguračné čítanie
1011	konfiguračný zápis
1100	viacnásobný pamäťové čítanie
1101	dvojitý adresný cyklus
1110	lineárne pamäťové čítanie
1111	pamäťový zápis a zrušenie platnosti

Tabuľka č. 2. Tipy PCI príkazov [7]

3.4 Konfiguračný priestor

Každé funkčné PCI zariadenie má pridelený blok 64 konfiguračných dvoj slov rezervovaných pre implementáciu svojho konfiguračného registru. Formát a použitie prvých 16 dvoj slov je predefinovaných špecifikáciou PCI. Táto oblasť sa odkazuje na hlavičku konfiguračných oblastí zariadenia. Špecifikácia súčasne definuje dva formáty hlavičky, hlavičku typu NULA a typu JEDNA. Hlavička typu JEDNA je definovaná pre PCI-to-PCI mosty (bridge), zatiaľ čo hlavička typu NULA je používaná všetkými ostatnými zariadeniami.

Obrázok č. 9. zobrazuje formát hlavičky konfiguračného priestoru. Zostávajúcich 48 dvoj slov konfiguračného priestoru špecifikuje vlastné zariadenie. Modrou farbou sú vyznačené povinné konfiguračné registre.

3.4.1 Povinné konfiguračné registre

Register výrobcu (Vednor ID) informuje, kde bolo zariadenie vyrobené, Hodnota je natvrdo uložená v registri, ktorý je možné len čítať. Toto číslo je prideľované.

ID register zariadenia (Devide ID) je 16 bitová hodnota prideľovaná výrobcom a identifikuje typ zariadenia.

3	2	1	0	DW
Device ID		Vendor ID		00
Status Register		Command Register		01
Class Code			Revision ID	02
BIST	Header Type	Latency Timer	Cache Line Size	03
Base Address 0				04
Base Address 1				05
Base Address 2				06
Base Address 3				07
Base Address 4				08
Base Address 5				09
CardBus CIS Pointer				10
Subsystem ID		Subsystem Vendor ID		11
Expansion ROM Base Address				12
Reserved			Capability Pointer	13
Reserved				14
Max Gnt	Min Gnt	Interrrupt Pin	Interrrupt Line	15

Obr. č. 9. Formát hlavičky konfiguračného registru [7]

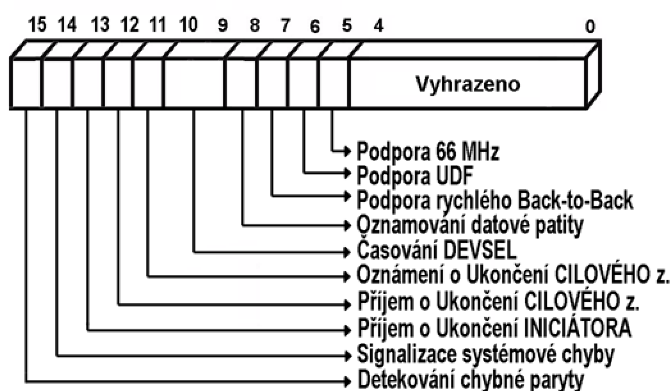
Príkazový register (Command register) zisťuje kontrolu nad schopnosťou zariadenia odpovedať a vykonávať PCI prístup. Jedná sa o 16 bitový register. Následne si popíšeme funkciu bito 9 až 0[7]:

- **0 bit – I/O prístup povolený.** Ak je bit nastavený na log. 1 je I/O povolený. Pri log. 0 je I/O zakázaný. Primárna hodnota je log. 0.
- **1 bit – Pamäťový prístup povolený.** Ak je bit nastavený na log. 1 zariadenie odpovedá pamäťovým prístupom. Pri log. 0 zakázaný. Primárna hodnota je log. 0.bit –
- **2 bit - INICIÁTOR povolený -** Ak je bit nastavený na log. 1 umožňuje sa zariadeniu chovať ako INICIÁTOR (ak je k tomu zariadenie vybavené). Pri log. 0 je toto chovanie zakázané. Primárna hodnota je log. 0.
- **3 bit** – Možnosť použitia špeciálneho cyklu. Pokiaľ je bit nastavený na log. 1. zariadenie môže sledovať špeciálny cyklus. Log. 0. Ignoruje špeciálny cyklus. Primárna hodnota je log. 0.
- **4 bit – Možnosť pamäťového zápisu a zrušenia platnosti.** Ak je bit nastavený na log. 1 zariadenie môže generovať príkazy pamäťového prístupu a zrušenia platnosti. Pri log. 0 zariadenie prednostne používa pamäťový zápis. Tento bit musí mať

implementovaný INICIÁTOR, ktorý je schopný generovať príkaz pamäťového zápisu a zrušenia platnosti. Reset nuluje tento bit.

- **5 bit – Povolenie VGA palete snoop.** Ak je bit nastavený na log. 1 nariaďuje VGA zariadeniu vykonávanie *palette snooping*. Pri zariadeniach, ktoré nepoužívajú VGA zobrazenie, reset nastavuje bit na log. 1. čo nastavuje daný mód. Pri zariadeniach VGA kompatibilných reset tento bit nastavuje na log. 0.
- **6 bit - Odozva na chybnú paritu.** Ak je bit nastavený na log. 1 zariadenie môže sledovať chybnú paritu. Pri log. 0 zariadenie ignoruje hlásenie o chybnej parite. Resetom je bit vynulovaný.
- **7 bit – Povolenie čakacieho stavu.** Sleduje, či zariadenie krokuje adresu či dáta. Zariadenie, ktoré nepoužíva krokovanie, musí mať tento bit natvrdo nastavený na log. 1. Zariadenie, ktoré pracuje v oboch prípadoch, musí mať tento bit prepisovateľný a inicializuje ho po resete do log. 1.
- **8 bit – Povolenie systémových chýb.** Pokiaľ je bit nastavený do log. 1., zariadenie môže riadiť signálový vodič SERR#. Log. 0. toto riadenie zakazuje. Stav bitu po resete je log. 0. Tento bit a šiesty bit musia byť nastavené súhlasne.
- **9 bit. – Povolenie Fast BACK-to-BACK.** Jedná sa o voliteľný bit pre INICIÁTORA. Pokiaľ je INICIÁTOR schopný vykonávať Fast BACK-to-BACK (prenos s rôznym TARGET zariadením), je tento bit nastavený podľa tejto schopnosti.

Stavový register sleduje udalosti na PCI zbernici. Zariadenie musí mať implementované bity, ktoré sa týkajú jeho funkčnosti. Na obrázku č. 10. Sú popísané významy jednotlivých bitov stavového registra.

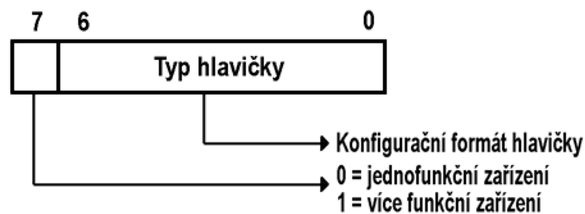


Obr. č. 10. Významy bitov stavového registra [7]

Register kódu triedy (Class code) je 24 bitový register určený pre čítanie. Je rozdelený na tri pod registre: bazová trieda, podtrieda a programovacie rozhranie. Tie definujú základné funkcie zariadenia a v niektorých prípadoch register špecifikuje programové rozhranie. Horný byt definuje

základný typ triedy, prostredný byt podtriedu v základnej triede a najnižší byte definuje programové rozhranie.

Register typu hlavičky (header type) zobrazuje formát tohto registra. Bity [6:0] definujú formát štvrtého až pätnásteho dvoj slova v konfiguračnom priestore. Ak je siedmi bit nastavený na log. 0. tak je zariadenie jedno funkčné, ak na log 1. tak je viacfunkčné.

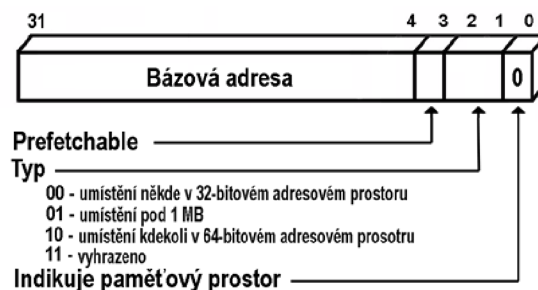


Obr. č. 11. Register typu hlavičky [7]

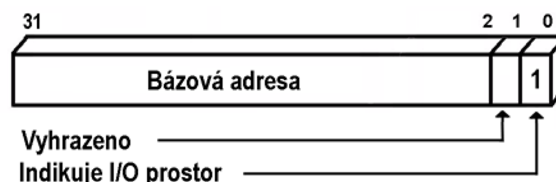
3.4.2 Voliteľné konfiguračné registre

Voliteľné konfiguračné registre sú potrebné pri konštrukcií zariadení, ktoré podporujú tzv. afektované (predstierané, nevlastné) funkcie

Register bázovej adresy sa nachádza v štyroch až deviatich dvoj slovách hlavičky konfiguračného priestoru zariadení umožňujúcich jeho premiestnenie. Každý register je 32 bitový a je využitý ako programovateľný pamäťový alebo I/O adresný dekodér.



(a)



(b)

Obr. č. 12. (a) Pamäťový adresný bázový register (b) I/O adresný bázový register [7]

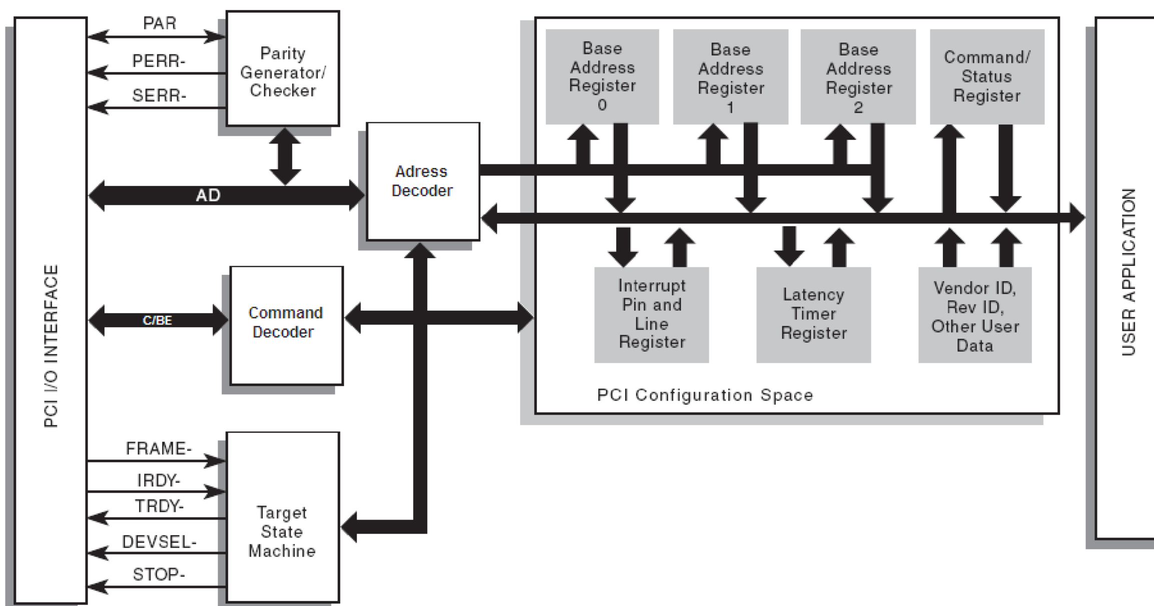
Obrázok č. 12. Ukazuje dva prípady adresného konfiguračného registra. Pokiaľ je bit nula vrátený ako log. 0., jedná sa o pamäťový adresný dekodér, ak ako Log. 1. ide o dekodér I/O adresy.

4 Návrh PCI radiča

PCI radič pre výukovú kartu s obvodom FPGA som po predchádzajúcej analýze PCI protokolu a potrebnej PCI špecifikácie navrhol nasledovným spôsobom. Celý radič bude zostavený z jednotlivých blokov implementujúcich funkcie potrebné pre komunikáciu karty pomocou zbernice PCI.

Jedná sa o kartu ktorá sa bude správať v hierarchií PCI ako INICIÁTOR komunikácie a TARGET zariadenie. Preto musí radič obsahovať dva bloky obsluhujúce komunikáciu obidvoch typov. Celý radič by mal pozostávať z nasledujúcich funkčných blokov:

- Adresný dekodér
- Príkazový dekodér
- Blok pre výpočet parity
- Arbiter funkcie TARGET zariadenia
- Konfiguračné registre



Obr. č. 13. Návrh blokovej schémy PCI radiča

Bloková schéma radiča je ako som ju navrhol zachytená na obrázku č. 13. Skladá sa vyššie z uvedených blokov. K adresnému dekodéru je privedená adresná zbernica, na ktorej daný blok neustále kontroluje stav a dekóduje adresu vystavenú na túto zbernicu, podľa veľkosti pridelenej pamäte a bázovej adresy pridelenej zariadeniu, ktorá je zapísaná v bázovom registri konfiguračného

priestoru daného zariadenia. Táto hodnota privedená zbernicou, ktorá je napevno pripojená k registru s bázovou adresou. Pri rozpoznaní adresy daného zariadenia, následne adresný dekodér riadi zápis a čítanie z adresovanej pamäte.

K adresnej zbernici je pripojený aj blok zodpovedný za výpočet parity a jej nastavovanie na signál PAR, alebo v prípade jej kontrolu s týmto signálom a nastavenie chybových signálov SERR a PERR. K tomuto bloku je tiež privedená zbernica C/BE, ktorá je tiež zahrnutá v procese výpočtu párnej parity. Táto činnosť neprebieha neustále ale je riadená internými signálmi z inštrukčného dekodéra a z arbitra funkcie TARGET zariadenia.

K arbitrážnym blokom funkcie TARGET a INICIÁTOR zariadenia sú pripojené signály pre riadenie prenosu so zbernice. Funkčnosť týchto jednotiek je riadená pomocou stavových automatov, ktorých návrh je bližšie špecifikovaný v nasledujúcom texte. Obe jednotky riadia činnosť ostatných blokov a teda podmieňujú výpočet parity a jej nastavovanie na zbernicu, dekódovanie príkazov a zápis a čítanie dát z a do pamäti.

Posledným blokom je príkazový dekodér, ktorý určuje druh vykonávanej operácie podľa hodnoty príkazu na zbernici C/BE počas adresnej fázy. Návrh funkčnosti všetkých blokov popisujem v nasledujúcich častiach tejto kapitoly.

4.1 Adresný dekodér

Tento blok zachytáva a nastavuje dáta z a na Adresnú zbernicu AD[31:0] a prepína ich na vnútorné zbernice DATA[31:0] a ADR[31:0]. Jeho hlavnou úlohou je rozpoznať pridelenú adresu danému zariadeniu a podľa nej indexovať zápis alebo čítanie do príslušného vnútorného registra zariadenia.

Adresa je dekódovaná na základe pamäťového priestoru, ktorý je pridelený zariadeniu pri úvodnom konfiguračnom prenose. Adresný priestor zariadenia je uložený v registroch bázovej adresy opisovaných v kapitole 3.4.2. Dekódovanie prebieha porovnávaním adresy získanej zo zbernice a adresného priestoru prideleného zariadeniu. Adresa je zachytená v čítači a ten je automatický inkrementovaný o 4 byty pre podporu blokového prenosu dát. V blokom prenose dát, sa daný blok stará o synchrónne posielanie dát a adresy, na ktorú majú byť dané dáta zapísané do pamäte zariadenia. Dáta sú posielané v takom stave ako sú posielané po zbernici, teda v blokoch o veľkosti štyroch bytov. Ďalšou funkciou, ktorú ma na starosti adresný dekodér je povolenie zápisu a čítania dát na základe stavu zbernice C/BE. Stav zbernica C/BE určuje, ktoré z aktuálnych bytov na zbernici sa majú zapísať, alebo ktoré byty chce zariadenie čítať.

4.2 Príkazový dekodér

V tomto bloku sa dekóduje príkaz nastavený počas adresnej fázy na C/BW[3:0] zbernici. Po dekódovaní sú dáta prístupné počas celého dátového prenosu. Dáta sú zachytené na základe signálu

udávajúceho platnosť príkazu na C/BE zbernici. Dekodér rozpoznáva PCI príkazy popísané v kapitole 3.3 a následne podľa nich nasledovné nastavuje signály, ktoré riadia činnosť ostatných častí radiča.

Je potrebné zabezpečiť, aby bol dekodér aktívni len počas adresnej fázy prenosu. Hlavnou úlohou dekodéru je rozoznať druh operácie a zaradiť ju do príslušnej kategórie. Operácie som rozdelil do nasledovných kategórií:

- Konfigurácia – jedná sa o nejakú konfiguračnú operáciu
- Čítanie – počas operácie sa bude vykonávať čítanie z daného zariadenia
- Zápis – počas operácie sa bude vykonávať zápis do daného zariadenia
- Pamäťová operácia – bude sa jednať o operáciu čítania, alebo zápisu z pamäte zariadenia
- I/O operácia - bude sa jednať o operáciu čítania, alebo zápisu z vstupno/výstupného priestoru zariadenia

4.3 Blok pre výpočet parity

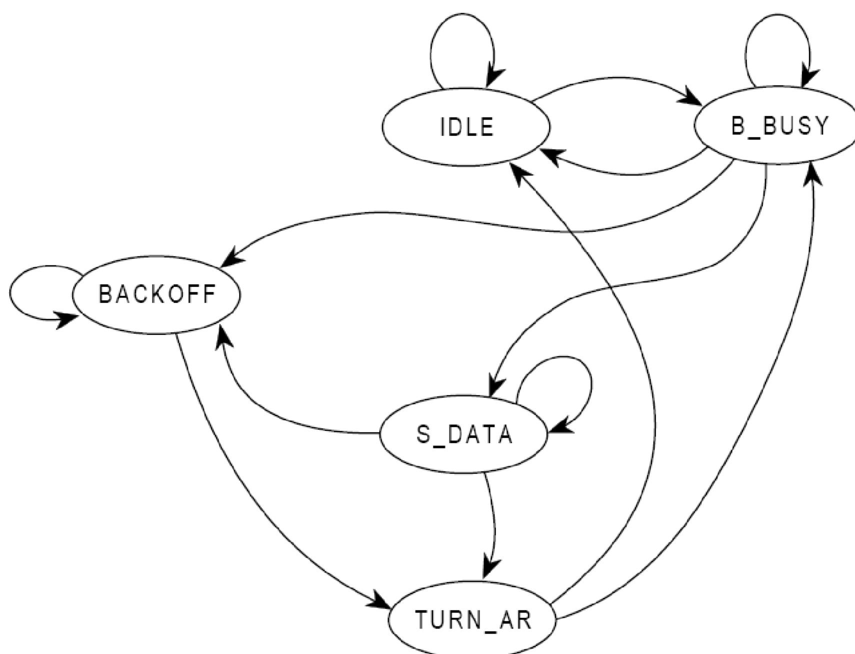
Tento blok, pokiaľ sa jedná o zápisu do TARGET zariadenia, kontroluje nastavenú paritu od INICIÁTORA nastavenú na signálovom vodiči PAR s vypočítanou paritou z adresy zachytenou z adresnej zbernice a dát zachytených zo zbernice C/BE. Kontrolná parita je vypočítaná pomocou troch stupňov XOR blokov s piatimi vstupmi. Ďalej je porovnávaná s dodanou paritou (musia byť zhodné) a na základe výsledku je v adresnej fáze generovaný signál SERR# udávajúci chybnú adresu a v dátovej fáze sa generuje signál PERR# udávajúci platnosť dát. Tieto dva signály sú oneskorené za adresou a dátami, z ktorých boli vypočítané o dve PCI periódy hodín a za PAR signálom o jednu periódu PCI hodín.

4.4 Arbiter funkcie TARGET zariadenia

Je takisto navrhnutý ako arbiter INICIÁTORA ako stavový automat. Na obrázku č. 15. Je zobrazený tento automat, zodpovedný za riadenie zariadenia vo funkcií TARGET. Automat je prevzatý s publikácie PCI Local Bus Specification [7], a v reálnej implementácií sa môže od daného automatu líšiť.

Automat pozostáva z nasledujúcich piatich stavov:

1. IDLE – Nečinný stav na zbernici.
2. TURN_AR – Transakcia na zbernici je kompletná.
3. B_BUSY – Nezapojený do aktuálnej transakcie.
4. S_DATA – Zariadenie akceptuje požiadavku a odpovedá.
5. BACKOFF – Agent je zaneprázdnený na odpoveď v tomto čase.



Obr. č. 16. Stavový automat riadenia funkcie TARGET [7]

Stavy IDLE a TURN_AR majú rovnaké výstupné pravidlá. Preto by mohli byť teoretický spojené do jedného stavu pre zjednodušenie automatu. Ale v tomto prípade každý stav nastavuje iné signály a tým je toto zjednodušenie znemožnené. Prechody medzi stavmi automatu sú v PCI Local Bus Specification [7] definované podľa nasledovných podmienok:

IDLE or TURN_AR	
goto IDLE	if **FRAME#**
goto B_BUSY	if !**FRAME#** * !Hit
B_BUSY	
goto B_BUSY	if (!**FRAME#** + !D_done) * !Hit
goto IDLE	if **FRAME#** * D_done + **FRAME#** * !D_done * !**DEVSEL#**
goto S_DATA	if (!**FRAME#** + !**IRDY#**) * Hit * (!Term + Term * Ready) * (FREE + LOCKED * L_lock#)
goto BACKOFF	*if (!**FRAME#** + !**IRDY#**) * Hit * (Term * !Ready + LOCKED * !L_lock#)
S_DATA	
goto S_DATA	if !**FRAME#** * !**STOP#** * !**TRDY# * IRDY#** + !**FRAME#** * **STOP#** + **FRAME#** * **TRDY#** * **STOP#**
goto BACKOFF	if !**FRAME#** * !**STOP#** * (**TRDY#** + !**IRDY#**)
goto TURN_AR	if **FRAME#** * (!**TRDY#** + !**STOP#**)
BACKOFF	
goto BACKOFF	if !**FRAME#**
goto TURN_AR	if **FRAME#**

Rovnako ako v prípade arbitra pre INICIÁTOR mód aj tu je potrebný ďalší automat pre určenie, či môže dané zariadenia v danom momente využívať komunikáciu prostredníctvom PCI zbernice. Jedna sa v podstate o to, aby zariadenie nezasahovalo do komunikácie iných dvoch

zariadení. Teda aby nevykonávalo dekódovanie adresy v dátach, ktoré sú zasielané v komunikácií, do ktorej nie je zariadenie zapojené. V opačnom prípade, by mohlo dôjsť k rozpoznaniu adresy v týchto dátach a následnému zapojeniu sa do komunikácie a kolízií na zbernici. Tento stavový automat je zobrazený na obrázku č. 17 [7].

Obr. č. 17. Stavový automat stavu zariadenia v režime TARGET [7]

Prechody medzi stavmi sú podmienené nasledovnými podmienkami[7]:

FREE
- goto LOCKED — if !**FRAME#** * **LOCK#** * Hit * (IDLE + TURN_AR) + L_lock# * Hit * B_BUSY)
- goto FREE — if ELSE

LOCKED
- goto FREE — if **FRAME#** * **LOCK#**
- goto LOCKED — if ELSE

4.5 Konfiguračné registre

Konfiguračné registre implementujú konfiguračný priestor ako som ho popísal v kapitole 3.4. Sú dôležitou súčasťou celého radiča a určujú funkcie celého zariadenia. Skladajú sa zo sady registrov. Z ktorých je niekoľko nastavených na pevné hodnoty a iné sa nastavujú pri úvodnej konfiguračnej komunikácií a pri RESETE.

4.6 Zhodnotenie návrhu

V procese návrhu som rozdelil radič do funkčných blokov, z ktorých má každý na starosti istú časť komunikácie prostredníctvom zbernice PCI. Celkovo som navrhol šesť častí radiča. Pri návrhu funkčnosti blokov som vychádzal z poznatkov získaných o princípoch komunikácie prostredníctvom zbernice PCI a signáloch, ktoré sú zodpovedné za prenos dát a riadenie komunikácie. Tieto teoretické vedomosti som zhrnul v kapitole tri danej práce.

V prípade blokov arbitrov oboch funkcií radiča som oba automaty navrhol ako ich popisuje publikácia *PCI Local Bus Specification* [7]. Je možné, že v priebehu implementácie radiča môže dôjsť k nutnosti pozmeniť funkčnosť nejakého bloku, alebo dokonca k úplnému odstráneniu, prípadne pridaniu nového bloku.

Samozrejme radič nemôže fungovať ako samostatné zriadenie. Sám radič negeneruje dáta na prenos, ani nie je cieľovou adresátom komunikácia na zbernici PCI. Pre otestovanie a demonštráciu funkcie PCI radiča je potrebné ho pripojiť, k nejakému funkčnému bloku, ktorý by generoval a prijímal dáta z PCI zbernice. Preto po implementácií PCI radiča budem musieť k radiču PCI pripojiť nejaké aplikačné zariadenie, ako napríklad sadu registrov grafického adaptéra.

5 Čítací prenos

Radič PCI zbernice pre výukovú kartu s obvodom FPGA som implementoval v jazyku VHDL. Vychádzal som z návrhu vykonanom v predchádzajúcej kapitole. Návrh som sa snažil dodržať, hlavne čo sa týka blokového rozdelenia radiča a popisovanej funkčnosti jednotlivých blokov.

Ako prvé som implementoval čítací prenos s pohľadu TARGET zariadenia. Pred implementáciou som vykonal dôkladnú analýzu tohto prenosu. Tento prenos popíšem v nasledujúcej kapitole a následne vykreslím detaily implementácie, ktorá sa stará o riadenie tohto prenosu.

V ďalšej časti sa budem venovať analýze zapisovacieho prenosu a jeho implementácií. Následne vykonám dôkladný rozbor implementovaných častí radiča. Bližšie poukážem hlavne na rozdiely v návrhu a implementácií.

V tejto kapitole podrobne popisujem čítací prenos prostredníctvom PCI zbernice. Prenos rozoberám a popisujem na vzorovom príklade. V podstate je dosiahnutie takéhoto priebehu cieľom implementácie.

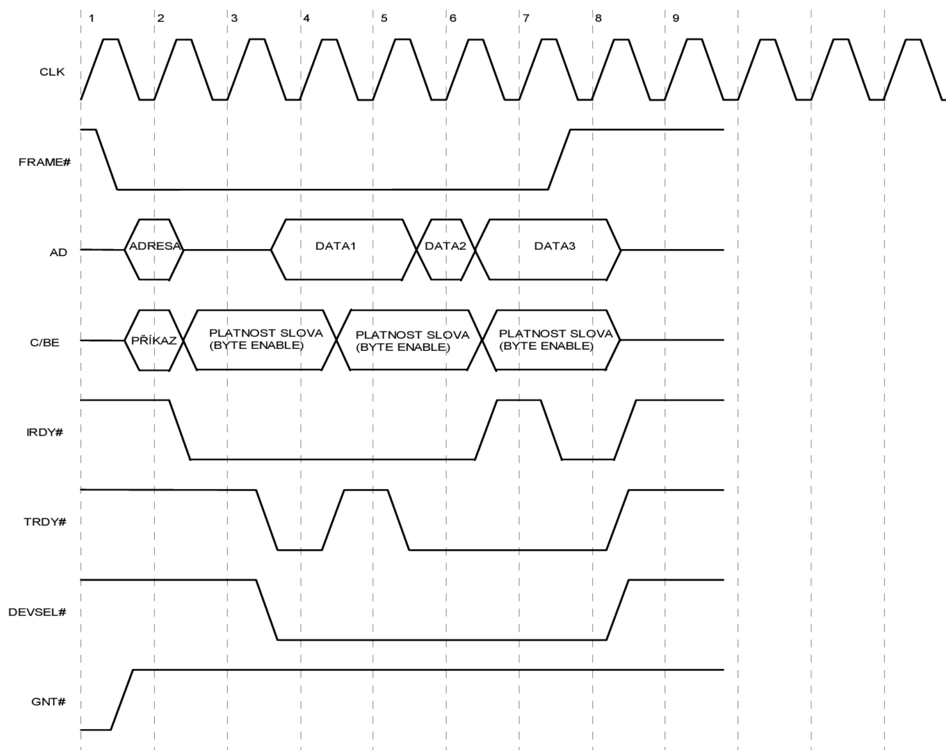
5.1.1 Analýza čítacieho prenosu

Nasledujúci popis sa vzťahuje na obrázok č. 18. Každá perióda je očíslovaná pre jednoduchšie odkazovanie. Predpokladajme, že INICIÁTOR získal zbernicu pre prenos. Následne musí INICIÁTOR čakať na stav, keď zbernica prejde do čakacieho stavu. To sa uskutoční vzorkovaním stavu signálov FRAME# a IRDY# s každou nástupnou hranou hodín. Pokiaľ sú oba signály v neaktívnom stave (prvá hrana hodín), zbernica je v čakacom stave a môže sa inicializovať prenos daným INICIÁTOROM.

Na začiatku prvej periódy hodín INICIÁTOR aktivuje signál FRAME#, čo indikuje, že prenos môže začať a platná štartovacia adresa spoločne s príkazom sú nastavené na príslušných zberniciach. Signál FRAME# musí byť nastavený až do konca okamihu prenosu, keď je INICIÁTOR pripravený dokončiť poslednú dátovú fázu. Adresa, príkaz a signál FRAME# musia byť nastavené počas prvého cyklu hodín.

Tzv. vratný cyklus (turn-around, alebo tiež mŕtvy cyklus) je potrebný pre všetky signály, ktoré majú byť riadené viac ako jedným PCI zbernicovým účastníkom (bus agent). Tento cyklus je potrebný k zamedzeniu kolízie, pokiaľ je jeden účastník v stave prerušenia riadenia výstupu a ďalší účastník začne riadiť ten istý signál.

Na začiatku druhej periódy INICIÁTOR prestane riadiť AD zbernicu. To dovolí TARGET zariadeniu prevziať kontrolu nad zbernicou k nastaveniu prvých požadovaných dát. Počas čítania je druhý cyklus definovaný ako vratný (turn-around), je to doba, keď sa mení vlastník AD zbernice, z INICIÁTORA na adresované TARGET zariadenie. TARGET musí počas tejto periódy udržať aktívny signál TRDY#.



Obr. č. 18. Čítací prenos na PCI zbernici (odvodené z [7])

Na počiatku druhej periódy INICIATOR takisto prestane riadiť C/BE ako príkazovú zbernicu a použije ju k indikácií, koľko bitov bude prenesené práve adresovanom dvoj slove. Obvykle bude INICIÁTOR aktivuje počas čítania všetky vodiče na C/BE zbernici. INICIÁTOR taktiež v druhom takte aktivuje signál IRDY#, čím indikuje, že je pripravený k prijatiu prvého dátového bloku z TARGET zariadenia. Po aktivácií IRDY# INICIÁTOR nedeaktivuje signál FRAME#, to znamená, že sa nejedná o poslednú dátovú fázu. Pokiaľ by išlo o poslednú dátovú fázu, INICIÁTOR aktivuje IRDY# a súčasne deaktivuje FRAME#. Touto kombináciou signálov naznačuje, že je pripravený dokončiť poslednú dátovú fázu.

Počas tretieho cyklu, TARGET zariadenie:

1. Aktivuje signál DEVSEL#, čím potvrdí rozpoznanie svojej adresy a to, že sa zúčastní nasledovného dátového prenosu.
2. Začne nastavovať prvé dáta (medzi jedným, až štyrmi bytmi, podľa toho, ako je nastavená C/BE zbernica) na AD zbernicu a aktivuje signál TRDY#, čím indikuje prítomnosť platných dát.

Pokiaľ INICIÁTOR a adresované TARGET zariadenie vzorkujú aktívne signály TRDY# a IRDY# s nástupnou hranou štvrtého hodinového cyklu, je prvý dátový blok čítaný INICIÁTOROM z dátovej zbernice a prvá dátová fáza je ukončená.

Pravidlom je, že INICIÁTOR musí vždy nastaviť správny hodnotu na zbernicu C/BE po každom úspešnom dátovom prenose. Pokiaľ z nejakého dôvodu INICIÁTOR nevie, ako bude BYTE

ENABLE nastavený pre ďalšiu dátovú fázu, mal by podržať signál IRDY# v neaktívnom stave a nemal by pripustiť ukončenie prebiehajúcej dátovej fázy. To bude trvať tak dlho, pokiaľ nebude vedieť čo bude ďalej.

V tomto prípade INICIÁTOR po vstupe do druhej dátovej fázy drží IRDY# v aktívnom stave, ale nedeaktivuje FRAME#. To znamená, že INICIÁTOR je pripravený čítať druhý dátový blok, bezprostredne po prvom, ale nejedná sa o poslednú dátovú fázu.

Pri viacnásobnom prenose dát je zodpovednosťou TARGET zariadenia zachytiť počiatočnú adresu v čítači adries. Po dokončení jednej dátovej fázy by malo TARGET zariadenie automaticky zvýšiť adresu uloženú v čítači o hodnotu štyri pre prenos ďalšieho dvoj slova. Následne dôjde ku kontrole nastavenia BYTE ENABLE od INICIÁTORA pre zistenie počtu prenášaných bytov v práve adresovanom dvoj slove.

V tomto príklade TARGET zariadenie potrebuje určitú dobu pre privedenie druhého požadovaného dátového bloku, preto deaktivuje TRDY#, čím vkladá do prenosu čakací stav do druhej dátovej fázy(časový úsek päť). S nástupnou hranou piateho cyklu INICIÁTOR vzorkuje neaktívny signál TRDY#, čím sa indikuje neprítomnosť dát na zbernici na AD zbernici. Pokiaľ INICIÁTOR vzorkuje aktívne signály IRDY# a TRDY# s nástupnou hranou hodín, prebieha čítanie druhého dátového bloku zo zbernice a zároveň je druhá dátová fáza ukončená. Druhá dátová fáza je zhrnutá časovými úsekmi štyri a päť.

Na počiatku tretej dátovej fázy INICIÁTOR nastaví BYTE ENABLE, čím informuje, koľko dát sa bude prenášať v ďalšom dvoj slove. Súčasne deaktivuje IRDY#, čo znamená, že bude potrebovať viac ako jednu periódu na prípravu prijatia dát. Počas šiestej periódy TARGET zariadenie drží aktívny signál TRDY# a vystaví tretie blok požadovaných dát na AD zbernicu. V tomto prípade INICIÁTOR požaduje viac času pokiaľ bude schopný dáta prečíta, preto je vložený do tretej dátovej fázy čakací stav. TARGET zariadenia však musí požadované dáta stále držať na AD zbernici aj počas tohto čakacieho stavu(časový úsek sedem).

Počas siedmej periódy INICIÁTOR aktivuje signál IRDY#, čím dáva najavo, že je pripravený akceptovať tretí dátový blok s nasledujúcou nástupnou hranou hodín. INICIÁTOR tiež deaktivuje signál FRAME#, pretože sa jedná o poslednú dátovú fázu. Vzorkovaním aktívnych signálov IRDY# a TRDY# s nástupnou hranou ôsmeho hodinového taktu INICIÁTOR číta tretí dátový blok zo zbernice.

Časové úseky sedem a osem zahrňujú tretiu dátovú fázu. Vzorkovaní deaktivovaného signálu FRAME#, TARGET zariadenie zistí, že sa jedná o posledný dátový blok. Celkový blokový prenos sa skladá z troch dokončených dátových fázy. INICIÁTOR deaktivuje signál IRDY#, čím vracia zbernicu do čakacieho stavu (s nástupnou hranou deviateho časového cyklu) a TARGET zariadenie deaktivuje TRDY# a DEVSEL#.

5.2 Popis implementácie čítacieho prenosu

Pre zabezpečenie čítania z pohľadu TARGET zariadenia, som musel implementovať nasledovné bloky:

- Adresný dekodér
- Príkazový dekodér
- Arbiter funkcie TARGET
- Pamäť

Všetky spomenuté bloky som implementoval, ako entity v jazyku VHDL. Pre popis ich funkčnosti som využil behaviorálny popis architektúry. Celý PCI radič pre výukovú kartu som popísal entitou PCI_core, ktorú som implementoval štruktúrnym popisom. Teda vnútri tejto entity som popísal vzájomne prepojenie implementovaných blokov. V nasledujúcich podkapitolách popíšem implementáciu jednotlivých blokov, ako aj ich vzájomne prepojenie.

5.2.1 Arbiter funkcie TARGET v čítacom prenose

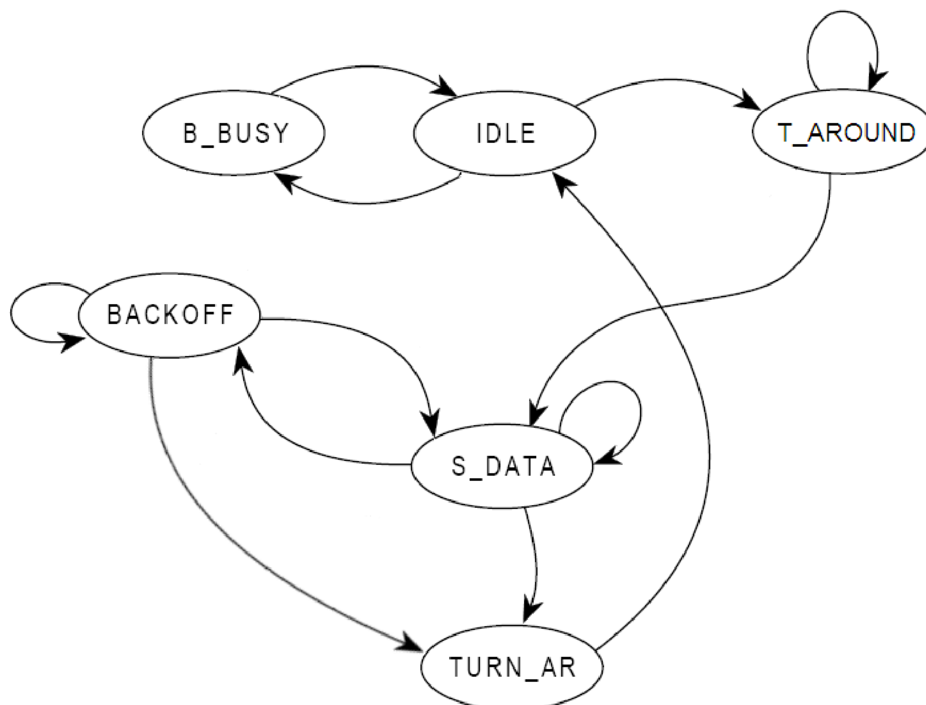
Arbiter funkcie TARGET srdcom celého PCI radiča implementujúceho funkciu TARGET zariadenia. Jedná sa o riadiacu jednotku, ktorá ma na starosti pomocou interných signálov riadiť činnosť ostatných jednotiek na základe signálov zo zbernice ako aj na základe stavu interných signálov, ktoré indikujú stav v jednotlivých blokoch.

Arbiter je implementovaný ako stavový automat. Nakoľko som sa pri jeho implementácií odklonil od toho ako som ho navrhol v procese návrhu radiča, tak tu následne popíšeme jeho implementáciu pre čítací prenos. Na rozdiel od návrhu som implementoval stavový automat pomocou nasledujúcich šesť stavov:

1. IDLE
2. B_BUSY
3. T_AROUND
4. S_DATA
5. BACKOFF
6. TURN_AR

IDLE – Je počiatočný stav celého automatu. V tomto stave zariadenie očakáva adresovanie na AD zbernici a všetky nastavené adresy dekóduje.

B_BUSY – Stav v ktorom sa dané zariadenie neúčastní aktuálnej komunikácia na zbernici. V tomto stave zariadenie nevykonáva žiadnu operáciu. Očakáva len ukončenie aktuálne komunikácie, následne sa opäť prepína do stavu IDLE.



Obr. č. 19. Stavový automat pre riadenie čítania TARGET zariadenia

T_AROUND – Stav, ktorý zabezpečuje tzv. vratný stav (turn-around). Jedna sa o stav, v ktorom sa čaká jeden cyklus pre prevzatie AD zbernice od INICIÁTORA. Po ubehnutí jednej periódy sa prechádza do stavu S_DATA.

S_DATA – Stav, v ktorom dochádza k vystaveniu dát, na AD zbernicu a inkrementácií adresy pre nasledujúci blok dát. Z tohto stavu sa môže prejsť do stavu BACKOFF v prípade, že je potrebné pozastaviť plynulý priebeh komunikácie, alebo priamo do stavu TURN_AR ak bude nasledujúci blok dát posledným čítaným z TARGET zariadenia.

BACKOFF – Stav, počas ktorého je pozastavený prenos dát na AD zbernici. Prenos dát môže pozastaviť cieľové aj zdrojové zariadenie. V prípade čítania je zariadenie TARGET zdrojom dát, v prípade zápisu je cieľovým zariadením zápisu dát. Po indikácií pripravenosti oboch zariadení pre prenos dát sa automat vracia do stavu S_DATA, a v prípade že sa bude jednať o posledný prenášaný blok dát do stavu TURN_AR

TURN_AR – Koncový stav komunikácie. Stav v ktorom sa dochádza k vystaveniu posledného bloku dát na zbernicu. Následne je prenos ukončený a automat prechádza sa do počiatočného stavu IDLE.

Celý automat by fungoval nesprávne, bez presne definovaných prechodových pravidiel. V nasledujúcom texte popíšem presné pravidlá, ktoré som použil v implementácií arbitra funkcie TARGET zariadenia, pre čítací prenos. Význam kontrolných signálov PCI zbernice sú popísané v kapitole 3.2.3 a význam interných signálov PCI radiča ktoré som definovál sám popíšem v nasledujúcej pokapitole.

IDLE

goto T_AROUND

Hit = '1' and FRAME = '0' and

(Operation = "000" or Operation = "101")

goto B_BUSY

Hit = '0' and FRAME = '0' then

B_BUSY

goto IDLE

FRAME = 'Z' and IRDY = 'Z' and TRDY = 'Z'

T_AROUND

goto S_DATA

(FRAME = '1' or FRAME = '0') and IRDY = '0'

goto T_AROUND

(FRAME = '1' or FRAME = '0') and IRDY /= '0'

S_DATA

goto BACKOFF

IRDY = '1' or TRDY = '1'

goto TURN_AR

FRAME = '1'and IRDY = '0' and TRDY = '0'

BACKOFF

goto BACKOFF

IRDY = '1' or TRDY = '1'

goto S_DATA

FRAME = '0' and IRDY = '0' and TRDY = '0'

goto TURN_AR

FRAME = '1'and IRDY = '0' and TRDY = '0'

TURN_AR

goto IDLE

IRDY = '0' and TRDY = '0'

Interné signály, ktoré nepatria do skupiny kontrolných signálov PCI zbernice a ich hodnota podmieňuje prechody medzi stavmi stavového automatu arbitra sú nasledovné dva signály:

1. **Hit** – jedná sa o jednobitový signál, ktorý je nastavovaný v adresnom dekodéry. Jeho nastavenie na log. 1 značí, že došlo k rozpoznaniu adresy daného zariadenia počas adresnej fázy na zbernici AD. A teda bude zariadenie zdrojom, alebo cieľom dát v nasledujúcej transakcií.

2. **Operation** – signál, ktorý určuje aký druh operácie sa bude vykonávať v nasledovnej transakcií. Hodnota signálu je nastavovaná v príkazovom dekodéry, podľa hodnoty príkazu na zbernici C/BE počas adresnej fázy na zbernici. Hodnota signálu "000" značí, že sa bude vykonávať čítanie dát z pamäte zariadenia a hodnota "101" indikuje čítací konfiguračný prenos. Tento prenos si bližšie popíšem v kapitole Konfiguračný prenos.

5.2.2 Riadiaca činnosť arbitra

Hlavnou úlohou arbitra funkcie TARGET je riadiť ostatné bloky v procese komunikácie prostredníctvom zbernice PCI. Túto svoju funkciu vykonáva pomocou interných signálov radiča. Ich hodnoty nastavuje podľa aktuálneho stavu riadiaceho stavového automatu. V každom stave nastaví hodnoty signálov tak, aby bloky ovládané arbitrom vykonávali správnu činnosť potrebnú pre daný úsek komunikácie na PCI zbernici.

Následne popíšem signály, pomocou nastavovania ktorých som implementoval riadiacu činnosť arbitra. Teda sa jedná o signály, ktoré prepájajú blok arbitra s ostatnými blokmi. Ich hodnotu môže zmeniť jedine arbiter. Ostatné bloky podľa ich hodnôt vykonávajú vlastné operácie.

1. **ADDR_inc_tar** – signál, ktorý riadi inkrementáciu adresy v adresnom dekodéry. Využíva sa pri blokovom dátovom prenose, keď dochádza k prenosu viac ako jedného bloku dát. Nastavením signálu do log. 1 arbiter káže adresu inkrementovať, v opačnom prípade k inkrementácií nedochádza.
2. **Decode_tar** – signál, ktorého hodnota určuje, či bude v adresnom dekodéry dochádzať k dekódovaniu adresy na zbernici AD a v príkazovom dekodéry k dekódovaniu príkazov na zbernice C/BE. Teda určuje čí práve prebieha v procese komunikácie na PCI zbernici adresná fáza. Ak je signál nastavený na log. 1 dochádza k dekódovaniu v opačnom prípade sa nedekóduje.
3. **Do_operation_tar** – signál, prikazujúci vykonávať adresnému dekodéru operáciu. Druh operácie určuje hodnota interného signálu Operation, ktorú nastavuje príkazový dekodér po dekódovaní príkazu na zbernici C/BE počas adresnej fázy. To znamená, že signál určuje, či sa budú dáta z príslušnej adresy nastavovať na AD zbernicu(v prípade čítacieho prenosu), alebo sa budú dáta zapisovať z AD zbernice na adresované pamäťové miesto(v prípade zapisovacej transakcie). V prípade, že arbiter nastaví signál na hodnotu log. 1, dochádza k vykonávaniu operácie, v prípade, že je signál nastavený na log. 0. k operácií nedochádza.
4. **ParEnable_tar** – signál určujúci, či bude v bloku pre výpočet a kontrolu parity dochádzať k výpočtu parity a následne jej kontrole so signálom PAR(v prípade zapisovacej transakcie), alebo k nastavovaniu signálu PAR, podľa vypočítanej hodnoty

parity. V prípade, že je signál nastavený na log. 1. Dochádza ku výpočtu parity, v opačnom prípade, ak je signál nastavený na log. 0. k výpočtu nedochádza. To či sa bude porovnávať so signálom PAR alebo sa bude tento signál nastavovať vypočítanou paritou rozhoduje nastaveným signálom príkazový dekodér.

Okrem týchto interných signálov je úlohou arbitra funkcie TARGET nastavovať kontrolný signál DEVSEL#, ktorým TARGET značí IINICIÁTOROVI transakcie, že sa našlo zariadenie, ktoré bolo adresované ako TARGET danej transakcie. V nasledujúcej časti popíšem, ako som v jednotlivých stavoch stavového automatu nastavoval riadiace signály a vysvetlím prečo je potrebné nastaviť tieto signály práve takto aj v analógií na obrázok č. 18. Pomocou ktorého som analyzoval čítací prenos. Keď signál nastavujem na stav 'Z', znamená to že sa nastavuje do stavu vysokej impedancie. Túto anotáciu som prebral priamo z jazyka VHDL, ktorým som celý radič implementoval.

IDLE

- DEVSEL# Z
- ADDR_inc_tar 0
- Decode_tar 1
- Do_operation_tar 0
- ParEnable_tar Z

V stave IDLE dochádza na zbernici k adresnej fázy, (obr. č. 18 – prvá perióda hodín). Nevykonáva sa operácia, ani sa nepočíta parita, ani sa nesignalizuje INICIÁTOROVI, nájdenie TARGET zariadenia. Jediné čo PCI radič vykonáva je dekódovanie adresy na AD zbernici a príkazu na C/BE zbernici(Decode_tar - 1).

B_BUSY

- DEVSEL# Z
- ADDR_inc_tar 0
- Decode_tar 0
- Do_operation_tar 0
- ParEnable_tar Z

V stave B_BUSY prebieha na zbernici komunikácia do ktorej nie je dané zariadenie zapojené. Signály sú nastavené rovnako, až na to, že nedochádza ani k dekódovaniu adresy na AD zbernici a príkazu na C/BE. Keby k dekódovaniu dochádzalo, hrozila by kolízia na zbernici. Signál Decode_tar je teda nastavený na log. 0.

T_AROUND

- DEVSEL# Z
- ADDR_inc_tar 0
- Decode_tar 0

- Do_operation_tar 0
- ParEnable_tar 1

Daný stav implementuje tzv. čakací stav(turn-around). Zariadenie v tomto stave už vie, že bude účastníkom v tejto komunikácií no ešte sa nenastavuje signál DEVSEL, pomocou ktorého zariadenie indikuje INICIÁTOROVI, že bolo rozpoznané adresované zariadenie. Viď. druhý hodinový takt na Obrázku č. 18. Signál DEVSEL má teda stále hodnotu 'Z'. V tomto stave už dochádza k výpočtu parity adresy a jej porovnaniu so signálom PAR. Preto je signál ParEnable_tar už nastavený na log. 1. Počas čakacieho stavu, neprebieha prenos, preto je signál Do_operation_tar nastavený na log. 0 a taktiež signál ADDR_inc_tar na log. 0. Keďže nemôže dochádzať ani k inkrementácií adresy.

S_DATA

- DEVSEL# 0
- ADDR_inc_tar 1
- Decode_tar 0
- Do_operation_tar 1
- ParEnable_tar 1

Stav S_DATA modeluje stav zariadenia, kedy dochádza k prenosu dát prostredníctvom PCI zbernice. Adresované dáta sú počas čítacieho prenosu nastavené s dobežnou hranou na AD zbernicu, a čítač adries v adresnom dekodéry je inkrementovaný o hodnotu 4. Teda signál Do_operation_tar a ADDR_inc_tar sú nastavené na log. 1, čím sa dáva znamenie adresnému dekodéru, že prebieha operácia a je potrebné inkrementovať čítač, pre vystavenie dát na AD zbernicu v nasledujúcom takte. Počas tohto stavu, už dochádza k indikácií, že je vybrané TARGET zariadenie v danej komunikácií, a to tým, že sa signál DEVSEL nastaví na log. 0. Daný stav modeluje tretí hodinový takt, na obrázku č. 18. Počas prenos dát nedochádza k dekódovaniu adresy a a príkazu, preto je Decode_tar nastavený na log. 0. Dochádza však k výpočtu parity. Signál ParEnable_tar je nastavený na log. 1.

BACKOFF

- DEVSEL# 0
- ADDR_inc_tar 0
- Decode_tar 0
- Do_operation_tar 1
- ParEnable_tar 1

Daný stav modeluje situáciu v čítacom prenose, keď buď INICIÁTOR nie je schopný prijať dáta v danom takte, alebo TARGET nie je schopný vystaviť platné dáta na AD zbernicu. Túto situáciu signalizuje INICIÁTOR nastavením signálu IRDY na log. 1. a TARGET nastavením signálu TRDY taktiež na log. 1. Viď piaty a ôsmy hodinový takt na obrázku č. 18. Zariadenie je stále vybrané, preto je signál DEVSEL stále nastavený na log. 0. Priebeh prenosu je pozastavený ale operácie nie je zrušená, preto je signál Do_operation_tar ponechaný na hodnote log. 1. Počas

čakacieho stavu však nemôže dochádzať k inkrementácií adresného čítača, nakoľko bude adresa dáta rovnaká i v nasledujúcom hodinovom takte. Preto je signál ADDR_inc_tar nastavený na log. 0., čím prikazuje adresnému čítaču, aby v tomto takte nevykonal inkrementáciu adresy. Keďže nie je prerušená operácie, nemôže byť prerušený ani výpočet parity. Signál ParEnable_tar má teda hodnotu log. 1. Signál Decode_tar je ponechaný na úrovni log. 0. nakoľko stále nedochádza k dekódovaniu adresy a príkazu.

TURN_AR

- DEVSEL# Z
- ADDR_inc_tar 0
- Decode_tar 0
- Do_operation_tar 0
- ParEnable_tar 1

Tento stav indikuje, že daný cyklus prenosu je posledným v danom spojení. Znamená to, že už netreba inkrementovať adresu v čítači. Signál ADDR_inc_tar sa teda v tomto stave nastavuje na hodnotu log. 0. Takisto sa už prestáva vykonávať prebiehajúca operácie a preto je signál Do_operation_tar taktiež nastavený na log. 0. Stále je však potrebné vykonávať výpočet parity a podľa nej nastavovať signál PAR(v danom prípade čítacieho prenosu), teda signál ParEnable_tar je ponechaný na úrovni log. 1. Práve prebiehajúci prenos dát sa končí, a teda od nasledujúceho cyklu bude musieť byť, signál DEVSEL nastavený do stavu vysokej impedancie 'Z'.

V tejto kapitole som popisoval, ako som implementoval arbiter funkcie TARGET v čítacom prenose. Jedná sa o riadiacu jednotku ktorá riadi prenos dát prostredníctvom PCI zbernice. O samostatný zápis dát na zbernicu sa v procese čítania zo zariadenia stará adresný dekodér. Jeho implementáciu popisujem v nasledujúcej kapitole.

5.2.3 Implementácia arbitra vo VHDL

Arbiter funkcie TARGET je stavový automat. V jazyku VHDL existuje prístup, ktorý implementuje stavové automaty, ktoré by mali syntetizačné nástroje rozpoznať a optimalizovane namapovať na hardwarové prostriedky v FPGA integrovanom obvode. Táto tzv. šablóna pozostáva z troch hlavných procesov, ktoré implementujú celú funkčnosť stavového automatu:

1. Popis aktuálneho stavu
2. Výpočet nasledujúceho stavu
3. Výstupná logika

Proces popisujúci aktuálny stav, nie je nič iné ako register, ktorý v každom hodinovom cykle priradí do signálu cur_state, ktorý predstavuje hodnotu aktuálneho stavu, hodnotu signálu next_state.

Proces pre výpočet nasledujúceho stavu popisuje kombinačnú logiku, ktorá na základe aktuálneho stavu a hodnoty vstupných signálov arbitra a podľa prechodových podmienok, ktoré som

popísal v kapitole 5.2.1., vpočíta hodnotu nasledujúceho stavu a túto hodnotu priradí signálu next_state.

Výstupná logika je proces, ktorý taktiež implementuje kombinačnú logiku. Úlohou tohto procesu je nastaviť výstupné signály, ako som ich popísal v predchádzajúcej kapitole. Pomocou tohto procesu arbiter riadi činnosť ostatných častí radiča.

Celý stavový automat by nefungoval bez zakódovaných stavov. Automat pozostáva zo šiestich stavov, ktoré som zakódoval na troch bitoch. Zápis kódu stavov som implementoval pomocou konštánt, ktoré podporuje jazyk VHDL pomocou kľúčového slova constant.

5.2.4 Adresný dekodér v čítacom prenose

Adresný dekodér v čítacom prenose dve hlavné úlohy:

1. **Dekódovať adresu**
2. **Zaistiť zápis dát na AD zbernicu počas samostatného čítania**

V prvom prípade adresný dekodér kontroluje stav AD zbernice počas adresnej fázy prenosu a zisťuje, či aktuálne vystavená adresa na zbernici, nespadá do adresného priestoru prideleného zariadeniu. K tomu je potrebné, aby adresné dekodér vedel aká bázová adresa je pridelená zariadeniu. Prideľovanie bázovej adresy zariadeniu budem podrobne popisovať v kapitole o implementácií konfiguračného prenosu. Do adresného dekodéru je teda napevno privedený signál z aktuálnou hodnotou bázového registru s konfiguračného priestoru zariadenia pomocou signálu **Base_dec.** Ďalšou informáciou, ktorú musí adresný dekodér poznať, je veľkosť pamäte zariadenia. Túto hodnotu som už zapísal natvrdo do procesu starajúceho sa o dekódovanie adresy.

Samotný proces dekódovanie je podmienený hodnotou signálu **Decode_dec**, ktorého hodnotu nastavuje arbiter a určuje tak, či sa zbernica momentálne nachádza v adresnej fáze, alebo či na zbernici prebieha nejaký prenos dát a tým pádom ich adresný dekodér nemôže dekódovať.

Ak dôjde počas adresnej fázy k rozpoznaniu adresy pridelenej zariadeniu, adresný dekodér nastaví signál **Hit** na log. 1. Čím dá najavo arbitrovi, že bola rozpoznaná adresa pridelená zariadeniu a môže sa začať prenos dát. Zároveň to, že je signál Hit nastavený na log. 1. podmieňuje zápis dekódovanej adresy do adresného čítača pre čítanie z pamäte.

Počas plnenia druhej funkcie(zápisu dát na AD zbernicu), číta adresný dekodér dáta z adresovanej pamäte a podľa hodnoty C/BE zbernice ich nastavuje na AD zbernicu. Celý tento proces je podmienený hodnotou signálu Do_operation_dec, ktorého hodnotu nastavuje arbiter a dáva tak najavo dekodéru, že je potrebné vykonávať aktuálnu operáciu. O to akú operáciu rozhoduje príkazový dekodér podľa príkazu nastaveného na zbernicu C/BE INCIÁTOROM počas adresnej fázy. Bude sa jednať o čítanie alebo zápise s pamäte. Implementáciu zápisu do pamäte popíše v kapitole popisujúcou implementáciu zapisovacieho prenosu.

Ak má teda signál Do_operation_dec hodnotu log. 1. začne adresný dekodér čítať dáta z adresy uloženej v adresnom čítači a nastavovať ich na AD zbernicu. To, že bude čítať dáta zo správnej adresy je tá skutočnosť, že nastavenie signálu Do_operation_dec na log. 1. je podmienené tým, že dôjde k rozpoznaniu adresy na AD zbernici a jej nastaveniu do adresného čítača.

Ak je povolená operácia a jedná sa o čítanie zo zariadenia tak proces **output** s každou dobežnou hranou hodín nastaví dáta podľa adresy v čítači na AD zbernicu. V každom takte dochádza k nastaveniu štyroch bytov. Nastavenie každého bytu je podmienené hodnotou príslušného bitu na C/BE zbernici. V podstate proces modeluje kombinačnú logiku pozostávajúcu zo štyroch registrov prenášajúcich vstupný byte na výstup s povoľovacím vstupom, na ktorý je privedený príslušný bit zo zbernice C/BE.

To že na vstup procesu **output** prídu dáta adresované v danom takte zabezpečuje správne nastavená adresa v adresnom čítači. Inkrementácia tohto čítača je podmienená signálom **ADDR_inc_dec**(log. 1.). V každom takte keď dôjde k úspešnému prenosu dvoj slova z TARGET zariadenia do INICIÁTORA dôjde aj ku inkrementácií dresy v adresnom čítači o hodnotu štyri. V prípade, že nie je INICIÁTOR v danom takte prijať aktuálne nastavené dáta na AD zbernicu, nedochádza k inkremntácií, aby nedošlo k preskočeniu dvoj slova v prenose. Tento stav rozpozná arbiter funkcie TARGET a nastaví hodnotu signálu **ADDR_inc_dec** na log. 0. V prenášaní dát sa bude pokračovať aj naďalej, len v danom takto hodín nedôjde k ich zmene na AD zbernici.

K adresnému dekodéru je okrem riadiacich signálov z arbitra a AD a C/BE zbernice pripojená cieľová zariadenie a teda pamäť. Tá je prepojená s dekodérom pomocou dvoch zbernic na čítanie a zápis dát, adresnej zbernice a pomocou signálu **Read_Write_dec**, ktorý určuje či sa bude čítať alebo zapisovať do pamäte. Tento signál sa nastavuje v registri data_output, ktorý jeho hodnotu podľa dekódovanej operácie z príkazového dekodéra.

5.2.5 Príkazový dekodér v čítacom prenose

Príkazový dekodér dekóduje príkaz na C/BE zbernici počas adresné fázy. Podľa rozpoznaného príkazu nastavuje signál, určujúci o akú operáciu sa jedná. Implementoval som operáciu čítania a zápisu do pamäte a konfiguračný zápis a čítanie.

Príkazový dekodér som implementoval ako veľký CASE, ktorého činnosť je podmienený hodnotou signálu Decode, ktorý nastavuje arbiter a indikuje tak, že práve prebieha na zbernici adresná fáza. V CASE sa podľa vstupnej hodnoty C/BE nastaví hodnota výstupného signálu Memory_RW_ins, ktorý môže nadobúdať nasledujúcich päť hodnôt:

1. Čítanie z pamäte
2. Zápis do pamäte
3. Konfiguračné čítanie

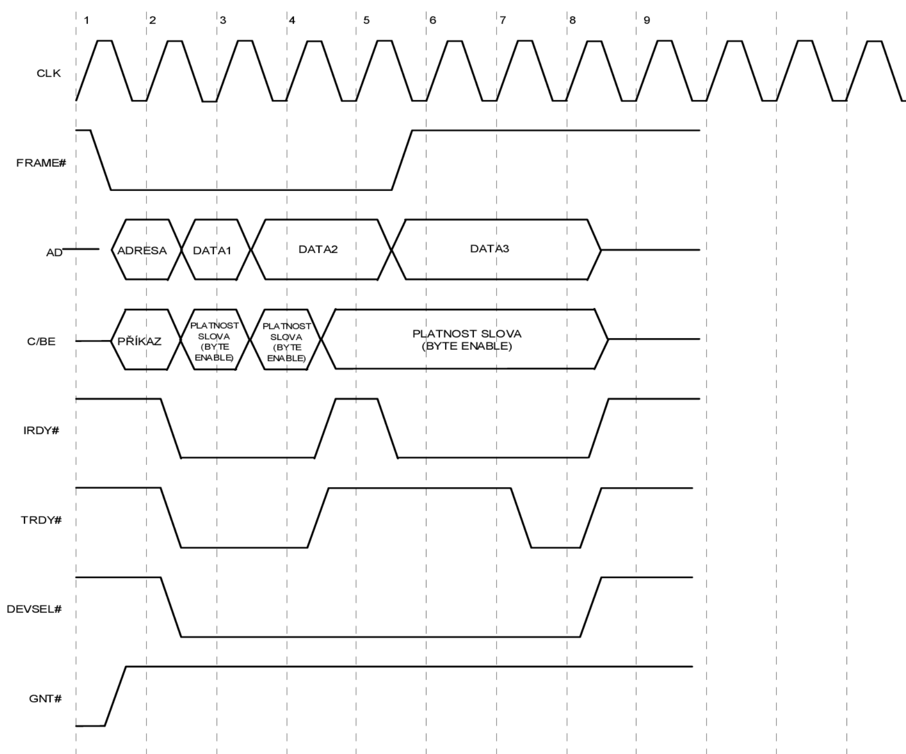
4. Konfiguračný zápis
5. Žiadna operácia

Podľa tohto signálu riadia svoju činnosť ostatné bloky. Adresný dekodér určuje či bude zapisovať alebo čítať z pamäte, arbiter určuje činnosť automatu a taktiež blok pre výpočet parity určuje čo sa bude vykonávať s vypočítanou paritou.

6 Zapisovací prenos

Počas tohto popisu zapisovacieho prenosu sa budeme odkazovať na Obr. 5. Predpokladajme, že INICIÁTORU bola arbitrom pridelená zbernica. INICIÁTOR musí čakať, až zbernica prejde do čakacieho stavu. To je splnené navzorkovaním signálov FRAME# a IRDY# s každou nástupnou hranou hodín. Pokiaľ sú oba signály neaktívne (s nástupnou hranou prvého cyklu), zbernica je v čakajúcom stave a prenos môže byť inicializovaný INICIÁTOROM, ktorého povolávací signál je aktivovaný od zbernicového arbitra.

Na počiatku prvého cyklu INICIÁTORA aktivuje FRAME#, čím oznamuje, že prenos začal a že platná adresa a príkaz sú prítomné na zberniciach. FRAME# zostáva nastavený až do okamihu kedy, je INICIÁTOR pripravený dokončiť poslednú dátovú fázu. V ten samý okamih, kedy INICIÁTOR aktivuje FRAME#, riadi počiatočnú adresu na AD zbernici a typ prenosu na C/BE zbernici. Adresa a typ prenosu sú nastavované na zbernici behom prvého cyklu hodín.



Obr. č. 20. Zapisovací prenos na PCI zbernici (odvodené z [7])

Spätný cyklus je potrebný pre všetky signály, ktoré sú riadené viac než jedným PCI účastníkom. Táto perióda je nutná aby sa zabránilo kolízii, ktorá vzniká v okamihu kedy jedno zariadenie pripojuje svoje výstupy a druhé zariadenie ich od tohto signálu odpojuje. Behom prvej periódy nie sú signály IRDY# , TRDY# a DEVSEL# riadenie (pripravujú sa na predanie medzi novým INICIÁTOROM a CIEĽOVÝM zariadením).

Na počiatku druhého cyklu INICIÁTOR vymenuje informácie, ktoré sú nastavené pre CIEĽOVÉ zariadenie cez AD zbernicu. Behom zapisovacej transakcie INICIÁTOR riadi AD zbernicu medzi adresnou a dátovou fázou. Od tejto chvíle sa predáva AD zbernice CIEĽOVÉMU zariadenie, ako je tomu pri čítaní prenosu, vratný cyklus je tu zbytočný. INICIÁTOR môže začať riadiť prvý dátový blok na AD zbernici s počiatkom druhého cyklu hodín. Ďalej behom tohto cyklu INICIÁTOR použije C/BE zbernicu k indikácií, koľko bytov bude prenesených v práve adresovanom dvoj slove a ktorá dátová cesta bude pre prvú dátovú fázu použitá.

Na počiatku druhého cyklu INICIÁTOR nastaví dáta na AD zbernicu a aktivuje BYTE ENABLE k indikácií dátovej cesty, ktorá prenesie platné dáta. Taktiež aktivuje IRDY# k indikácií prítomnosti platných dát na zbernici. INICIÁTOR nedeaktivuje FRAME#, dokiaľ je aktivované IRDY# (pretože sa nejedná o poslednú dátovú fázu).

Je nutné zdôrazniť, že INICIÁTOR nemôže aktivovať IRDY# bezprostredne po vstupe do dátovej fázy, fáza potrebuje určitý časový úsek, aby pripravil prvé zdrojové dáta (pretože jeho buffer je prázdny). Viac - menej INICIÁTOR nemôže držať IRDY# neaktívne ďalej než osem hodinových cyklov behom dátovej fázy.

Počas druhého časového úseku CIEĽOVÉHO zariadenia dekóduje adresu, príkaz a aktivuje DEVSEL# k vyžiadaniu prenosu. Ďalej aktivuje TRDY#, čím dáva na vedomie svoju pohotovosť prijať prvý dátový blok.

S treťou nástupnou hranou hodín INICIÁTOR a práve adresované TARGET zariadenie vzorkuje aktívne signály TRDY# a IRDY#, toto nastavenie oboch signálov informuje o pripravenosti dokončiť prvú dátovú fázu. Jedná sa o nulový čakací stav prenosu. TARGET zariadenie príjme prvý dátovú blok zo zbernice s treťou nástupnou hranou PCI hodín (a vzorkuje BYTE ENABLE aby zistil, ktoré byty sa budú zapisovať) a dokončiť prvú dátovú fázu.

Počas tretieho časového úseku INICIÁTOR nastavuje druhý dátový blok na AD zbernici a BYTE ENABLE k indikácií, aké byty budú prenášané a dátovú cestu, ktorá bude použitá pre prenos druhej dátovej fázy. Zároveň je držaný IRDY# v aktívnom stave a nedeaktivuje sa FRAME#, čím sa dá najavo pripravenosť k dokončeniu druhej dátovej fázy, a tým i to, že sa nejedná o poslednú dátovú fázu. Aktívny IRDY# indikuje, že zapisované dáta sú nastavené na zbernicu.

S nástupnou hranou štvrtých hodín INICIÁTOR a práve adresované TARGET zariadenie vzorkuje aktívne signály TRDY# a IRDY#, čím dávajú najavo pripravenosť dokončiť druhú dátovú fázu. Jedná sa o nulový čakací stav dátovej fázy. TARGET zariadenie príjme so štvrtou nástupnou hranou hodín druhý dátový blok zo zbernice (vzorkuje BYTE ENABLE) a dokončuje sa druhá dátová fáza.

INICIÁTOR vyžaduje viac času predtým, než začne riadiť ďalší dátový blok na AD zbernicu. Na počiatku štvrtého časového úseku sa vkladá čakací stav do tretej dátovej fázy deaktivácie IRDY#. To dovoľuje INICIÁTOROVI omeškať nové dáta o jeden cyklus PCI hodín, ale s počiatkom štvrtého časového úseku musí nastaviť BYTE ENABLE pre tretiu dátovú fázu.

V tomto príklade TARGET zariadenie vyžaduje viac času predtým, než bude pripravené akceptovať dátový blok. Požiadavka na vyčkanie nastavuje TARGET zariadenie behom štvrtého časového úseku deaktiváciou TRDY#. Pokiaľ INICIÁTOR a TARGET zariadenie vzorkuje neaktívny IRDY# a TRDY# je vložený čakací stav s piatou nástupnou hranou hodín do tretej dátovej fázy.

Počas piateho časového úseku, kým INICIÁTOR ešte nemá prístupné dáta k riadeniu, musí riadiť stabilnú vzorku na dátovej ceste, inak spôsobí, že sa zbernica dostane do plávajúceho stavu. Špecifikácia neprikazuje, aby bola vzorka udržiavaná behom tejto periódy. Väčšinou sa pokračuje v riadení predchádzajúceho dátového bloku. TARGET zariadenie nebude akceptovať znovu prítomné dáta v týchto dvoch prípadoch:

- Po deaktivácii TRDY# to znamená, že nie je pripravený vziať v úvahu dáta.
- Po deaktivácií IRDY# dáva INICIÁTOR najavo, že sa nejedná o ďalší dátový blok pre TARGET zariadenie.

V priebehu piatej periódy INICIÁTOR aktivuje IRDY# a riadi posledný dátový blok na zbernicu. Taktiež deaktivuje FRAME#, aby informoval o poslednej dátovej fáze. TARGET zariadenie drží TRDY# neaktívny, čo naznačuje , že nie je pripravený akceptovať tretí dátový blok. S nástupnou hranou šiestej periódy INICIÁTOR vzorkuje aktívny IRDY#, oznamuje tak, že dáta sú pripravené, ale TRDY# je neustále neaktívny (pretože TARGET zariadenie nie je pripravené akceptovať dátový blok). TARGET zariadenie taktiež vzorkuje neaktívny FRAME#, čo naznačuje, že dátová fáza pokračuje. Tento stav je iba zdanlivo ukončená dátová fáza (TARGET drží neaktívny TRDY# až do okamihu, kedy je pripravený akceptovať poslednú dátovú fázu). Odpoveď na vzorkovanie neaktívneho TRDY# so šiestou nástupnou hranou hodín TARGET zariadenie a INICIÁTOR vkladajú druhý čakací stav (šiesty úsek) do tretej dátovej fázy. Behom druhého čakacieho stavu INICIÁTOR pokračuje v riadení tretieho dátového bloku na AD zbernici a udržuje nastavený BYTE ENABLE. TARGET zariadenie drží TRDY# neaktívne, čo naznačuje, že nie je pripravené. So siedmou nástupnou hranou hodín TARGET zariadenie a INICIÁTOR vzorkuje aktívny IRDY#, čo indikuje, že má INICIÁTOR stále nastavené dáta, ale TRDY# je neustále deaktivované. Odozvu TARGET zariadenia a INICIÁTORA je vložené do tretieho čakacieho stavu (siedma perióda) do tretej dátovej fázy.

Počas tretieho čakacieho stavu INICIÁTOR pokračuje v riadení tretieho dátového bloku na AD zbernici a udržuje nastavený BYTE ENABLE. TARGET zariadenie aktivuje TRDY#, čím dáva na vedomie, že je pripravené k dokončeniu poslednej dátovej fázy. S ôsmou nástupnou hranou TARGET zariadenie a INICIÁTOR vzorke oba aktívne signály IRDY# a TRDY#. Toto nastavenie signálov indikuje, že oba ako INICIÁTOR tak TARGET zariadenie sú pripravené k dokončeniu tretej a poslednej dátovej fázy. Odpoveďou je dokončenie tretej dátovej fázy s ôsmou nástupnou hranou hodín. TARGET zariadenie akceptuje tretí dátový blok z AD zbernice. Tretia dátová fáza sa skladá zo štyroch periód hodín (prvý časový úsek začína v časovej bunke štyri plus tri periódy čakacieho stavu).

Počas časovej periódy číslo osem INICIÁTOR prestal riadiť dáta na AD zbernicu, prestanie riadiť príkaz na C/BE zbernici a deaktivuje IRDY# (vracia zbernicu do čakacieho stavu). TARGET zariadenie deaktivuje TRDY# a DEVSEL#.

6.1 Arbiter v zapisovacom prenose

Po analýze zapisovacieho prenosu som začal implementovať tento prenos tak, že som doplnil už implementovaný arbiter, ktorý obsluhoval čítací prenos, tak aby mohol naraz obsluhovať oba typy prenosu.

Stavový automat obsluhujúci len zapisovací prenos by mal päť stavov:

1. IDLE
2. B_BUSY
3. S_DATA
4. BACKOFF
5. TURN_AR

Ako vidno tento automat má úplne tie isté stavy ako automat obsluhujúci čítací prenos, len mu chýba stav T_AROUND, nakoľko nie je potrebné v tomto prípade modelovať čakací cyklus, keďže nedochádza k odovzdaniu riadenia AD zbernice do rúk TARGET zariadenia a tým pádom nehrozí, kolízia na zbernici.

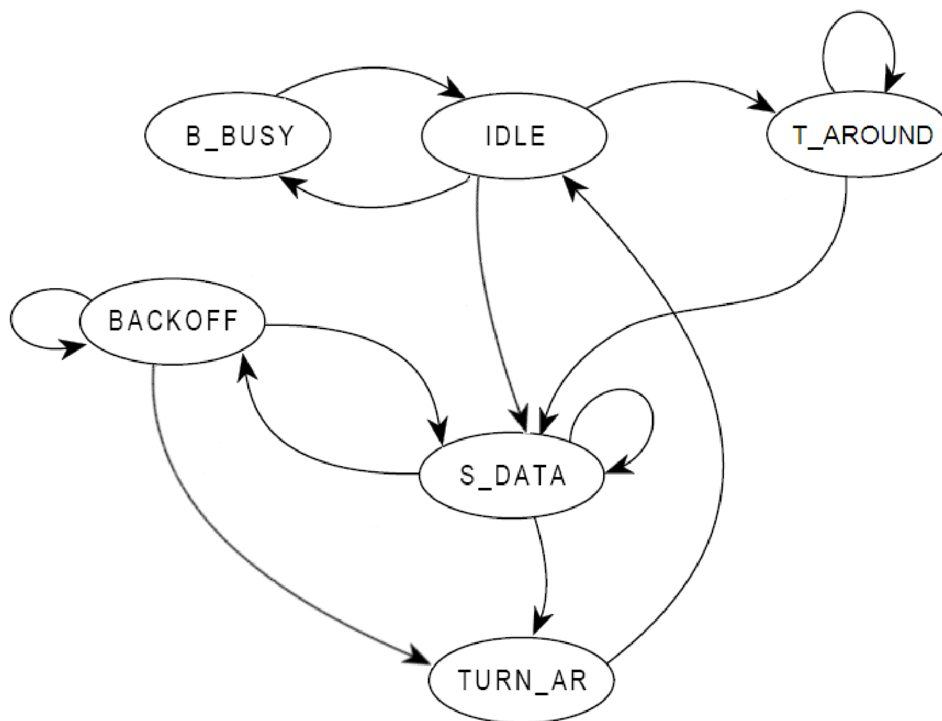
Celý automat ako som ho implementoval je na obrázku č. 21. Ako vidno je tu ešte jedna podstatná zmena a to prechod zo stavu IDLE do stavu S_DATA. Je to logická zmena nakoľko zo stavu IDLE do stavu S_DATA sa v automate implementujúcom čítací prenos dalo dostať len cez stav T_AROUND. V tomto prípade sa po detekovaní adresy zariadenia počas adresnej fázy na zbernici prechádza priamo do stavu S_DATA.

Pravidlo pre prechod zo stavu IDLE do stavu S_DATA má nasledovný tvar:

IDLE

```
goto S_DATA
        Hit = '1' and FRAME = '0' and
        (Operation = "001" or Operation = "110")
```

Zo stavu IDLE sa prechádza do stavu S_DATA v prípade, ak dôjde k rozpoznaniu adresy pridelenej danému zariadeniu počas adresnej fázy, čo indikuje hodnota signálu Hit nastavená na logickú 1.. A indikáciu prenosu pomocou nastaveného signálu FRAME na úroveň log. 0. Ďalšou nutnou podmienkou priameho prechodu do stavu S_DATA je to, že sa musí jednať o zapisovací prenos. To indikuje nastavenie signálu Operation na hodnotu “001” (zápis do pamäte), alebo “110”(konfiguračný zápis).



Obr. č. 21. Konečný stavový automat pre riadenie TARGET zariadenia

6.2 Adresný dekodér v zapisovacom prenose

Adresný dekodér v zapisovacom prenose implementuje inverznú operáciu ako pri čítacom prenose. Teda okrem rozpoznania adresy zariadenia sa stará o zápis dát z AD zbernice na príslušné adresované miesto pamäte zariadenia.

Zápis jednotlivých bytov s aktuálneho dvoj slova na AD zbernici je taktiež podmienení hodnotou príslušných bitov na C/BE zbernici. Kombinačnú logiku starajúcu sa o zápis príslušných bytov som implementoval v procese input. Táto kombinačná logika sa taktiež skladá zo štyroch zapisovacích blokov. Na ktorých tzv. povolovací vstup je privedení príslušný bit C/BE. Teda príslušný byt AD zbernice sa zapíše na vstupný signál pamäte zariadenia, len v prípade, že má potrebný bit na zbernici C/BE hodnotu log. 1.

V adresnom dekodéri som implementoval dva čítače adries. Zvlášť pre operáciu zápisu a zvlášť pre operáciu čítania. Nakoľko som potreboval použiť rozdielnu synchronizáciu zápisu a čítania do pamäte zariadenia. Dôležité je tie upozorniť, že počas zápisu do pamäte sa musí nastaviť povolovací signál zápisu Read_Write_dec na hodnotu log. 1. Tento signál sa nastavuje v procese **data,** na základe hodnoty signálu udávajúceho druh vykonávanej operácie.

6.3 Blok pre výpočet parity

Blok pre výpočet parity sa stará o zabezpečenie dát párnou paritou. Počíta teda párnu paritu pre zbernice AD a C/BE a nastavuje alebo kontrolu signál PAR. Samotný výpočet parity som implementoval ako blok za sebou idúcich XOR operácií. Jedná sa o operáciu tzv. logickej nezhody. V prípade že sa nezhodujú vstupy logického hradla XOR bude výstup nastavený na log. 1. V prípade, že sa budú zhodovať, bude výstup nastavený na log. 0.

Postupným prevedením tejto operácie na všetky bity oboch zberníc dôjdeme k hodnote 1 v prípade, že je na oboch zberniciach nepárny počet jednotiek a v prípade, že je počet jednotiek párny na hodnotu log. 0.

Správanie tohto bloku záleží od vykonávanej operácie na PCI zbernici. A teda či sa vykonáva zápis, alebo čítanie zo zariadenia.

6.3.1 Čítanie zo zariadenia

V prípade čítania zo zariadenie musí blok zabezpečiť dve operácie

1. Vypočítať párnu paritu a porovnať je so signálom **PAR** pre adresnú fázu. A v prípade, že dôjde k chybe musí blok nastaviť signál **SERR,** ako znak že došlo k chybe počas adresnej fázy. Pričom signál **PAR** je oneskorený o jeden hodinový takt od dát adresy a signál **SERR** o dva hodinové takty.
2. Počas prenosu dát musí blok počítať párnu paritu a nastavovať hodnotu signálu **PAR** na hodnotu párnej parity, ale oneskorenú o jeden hodinový signál oproti dátam, pre ktoré je parita počítaná.

Párna parita sa počíta pre všetky dáta nastavene na AD z C/BE, ktoré sa tikajú daného zariadenia. Rozdiel je len v tom čo sa s vypočítanou paritou urobí. Správnu funkciu bloku zabezpečuje výstupná logika. Tá v adresnej fáze porovnáva vypočítanú paritu so signálom PAR a nastavuje signál SERR, a v dátovej fáze nastavuje signál PAR. Nakoľko sa nastavuje signál PAR oneskorený o jednu periódu za dátami, použil som na indikáciu povolenia nastavovania čítač, ktorý počíta periódy po dobehnutí operácie.

6.3.2 Zápis do zariadenia

V tomto prípade sa musí blok taktiež riešiť dva prípady operácií:

1. Ako v prvom prípade porovnávať paritu so signálom **PAR** a nastavovať signál **SERR**.
2. Počas zápisu dát musí počítanú paritu porovnávať s hodnotou signálu **PAR** a nastavovať hodnotu signálu **PERR,** pre indikáciu chyby počas prenosu dát. Signál PERR je takisto oneskorený o dva hodinové cykly.

Pre prípad zápisu som použil ten istý druh výstupnej logiky.

7 Konfiguračný prenos

Počas návrhu PCI radiča som sa zameral len na implementáciu prenosu dát prostredníctvom PCI zbernice. Počas implementácie som však narazil na nutnosť implementovať blok, ktorý by mal na starosti konfiguračný zápis a čítanie. Počas konfiguračného čítania a zápisu prideľuje systém zariadeniu pamäť z adresného priestoru PC. Pre lepšie pochopenie funkčnosti konfiguračného bloku je treba vysvetliť pojmy adresný priestor, a konfiguračný zápis a čítanie.

V PCI norma sa uvádza niekoľko druhov konfiguračného prenosu. Ja som implementoval tzv. konfiguračný prenos typu NULA, ktorý sa používa pri prístupe ku konfiguračnému registru vo vnútri PCI zariadenia. Toto zariadenie sa nachádza na PCI zbernici a prevádza transakciu.

Systémovú konfiguráciu prevádza konfiguračný program prevádzaný hosťujúcim priestorom. To znamená, že hosťujúci procesor musí mať prístup k inštrukciám host/PCI mostov *(host/PCI bridge)* k prevádzaniu čítania a zapisovania transakcie na PCI zbernici.

7.1 Adresný priestor

Procesory Intel x86 majú schopnosť ovládať dva odlišné adresné priestory, a to pamäťový a I/O. K týmto priestorom u PCI sa pripojil ešte priestor konfiguračný. Táto časť priestoru je reprezentovaná konfiguračným registrom, ktorý musí byť inicializovaný pri štarte pre zariadenia pracujúce s I/O a pamäťovým priestorom. Pamäťový a I/O priestor PCI môže mať veľkosť až 4GB. Tento priestor je rozdelený na oddelené adresné priestory pre každú funkciu zariadenia.

Po zapnutí zariadenia musí konfiguračný program prejsť PCI zbernicou alebo zbernicami a detekovať, aké zariadenia sú pripojené a akú konfiguráciu vyžadujú. Tento proces sa nazýva skenovanie *(scaning)*, prechádzanie *(walking),* alebo sondovanie *(probing),*zbernice. Niekedy sa taktiež môžeme stretnúť s výrazom objavovanie *(discovery)*.

Na zjednodušenie tohto procesu musia mať všetky zariadenia PCI implementovaný konfiguračný register špecifikovaný normou PCI ako som ho popísal v kapitole 3.4. Tento register závisí od druhu operácií, preto zariadenie môže mať implementované ďalšie potrebné pracovné konfiguračné registre. Vedľa tohto špecifikácia definuje počet prídavných konfiguračných priestorov pre implementáciu zariadením definovaných konfiguračných registrov.

Konfiguračný program číta pod časťou konfiguračného registru zariadením, aby zistil, aké zariadenia sú pripojené ku zbernici a akého sú typu. Po zistení prítomnosti zariadenia program sprístupní ďalšie konfiguračné registre a zistí, koľko pamäťových blokov alebo aký I/O priestor zariadenie vyžaduje.

7.2 Konfiguračné prenos typu NULA

Hlavnou zmenou v konfiguračnom prenose typu NULA, oproti normálnemu prenosu dát je zmena adresnej fázy. Počas PCI konfiguračného prenosu všetky PCI zariadenia na zbernici zachytávajú nasledujúce informácie až do ukončenia adresnej fázy:

- obsah AD zbernice
- stav FRAME# signálu (nastavenie indikuje, že bola nastavená platná adresa a príkaz na zbernicu
- stav IDSEL signálov. Fyzické zariadenia, ktoré vzorkuje svoj aktívny IDSEL, je CIEĽOVÉ fyzické zariadenie.
- príkaz na C/BE 3:0 indikuje, že sa jedná o prenos konfiguračného zápisu/čítania. Typ príkazov je odvodený od typu prístupu hosťujúcich procesorov na dátový konfiguračný port host/PCI mostov. I/O čítanie je prevedené na konfiguračné čítanie a I/O zápis na konfiguračný zápis.

"00" binárne na AD 1:0 indikuje, že ide o konfiguračný prenos typu NULA s cieľom v jednom zo zariadení na tejto PCI zbernici. Fyzické zariadenie, ktoré vzorkuje svoj aktívny IDSEL, je cieľom konfiguračného prenosu. AD zbernica behom konfiguračného prenosu typu NULA predáva informácie a indikuje nasledujúce:

- pokiaľ AD 1:0 sú „00" binárne, znamená to, že ide o konfiguračný prenos typu NULA s cieľom v jednom zo zariadení na tejto PCI zbernici
- AD 7:2 udáva cieľové konfiguračné dvoj slovo
- AD 10:8 udáva cieľovú funkciu z fyzického puzdra
- AD 31:11 sú rezervované a nie sú využívané žiadnym zariadením

Cieľové fyzické číslo zariadenia je nastavované na bitoch 15:11 konfiguračného adresného portu, je dekódované a použité k nastaveniu jedného z výstupných signálov IDSEL radiča *(bridge)*. Radič je vybavený oddeleným IDSEL výstupom pre každé fyzické puzdro na PCI zbernici, vrátane jedného pre každý rozšírený PCI konektor. Nastavenie špecifického IDSEL počas adresnej fázy konfiguračného prístupu typu NULA je použité k výberu fyzického puzdra, ktoré je cieľom konfiguračného prístupu.

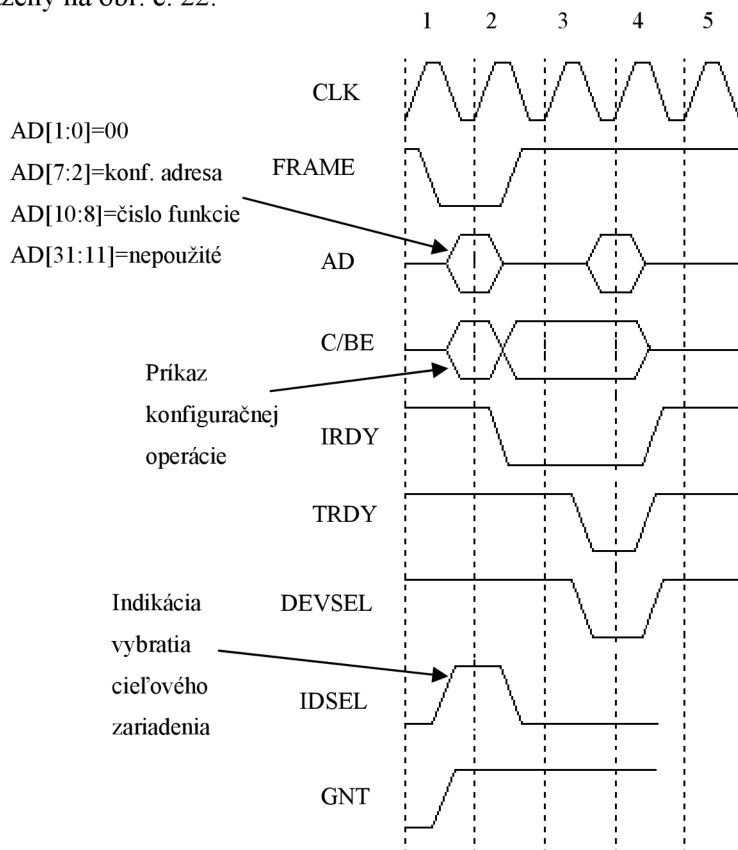
7.2.1 Dátová fáza

Na počiatku dátovej fázy je stav vstupného bytu hosťujúcich procesorov kopírovaný na PCI príkazovou zbernicou CB/E. Vstupné byty BE môžu byť nastavený na jednu z týchto hodnôt:

- Nastavenie BE0# a BE1# sú nastavené a indikujú prenos dvoch bytov cez nižšie dve dátové cesty s bytmi nula a jedna vo vybranom konfiguračnom dvoj slove.

- BE0# BE1# BE2# a BE3# sú nastavené a indikujú prenos štyroch bytov cez všetky štyri dátové cesty s bytmi nula v troch vybraných konfiguračných dvoj slovách.

Na počiatku dátovej fázy PCI zariadenia prevádzajú dekódovanie adresy, aby zistili, ktorí z nich je cieľom konfiguračnej transakcie. Zariadenie, ktoré vzorkujú svoje IDSEL vstupy jeho zhodením (deaktiváciou) na konci adresnej fázy ignorujú prenos. Fyzické zariadenie, ktoré vzorkuje aktívny IDSEL, dekóduje funkčné číslo nastavené na AD (10:8) a zistí či funkcia existuje v danom puzdre. Predpokladajme, že existuje. Potom puzdro nastaví DEVSEL#, čím žiada o prenos. Zvyšok prenosu je rovnaký ako u prenosu pamäťového alebo I/O. Celý priebeh konfiguračného zápisu je zobrazený na obr. č. 22.



Obr. č. 22. Príklad konfiguračného prenosu (odvodené z [7])

Konfiguračný zápis prebieha úplne rovnakým spôsobom, len s tým rozdielom, že sa vypúšťa spätný cyklu pre prevzatie AD zbernice TARGET zariadením. Teda dáta, ktoré sa majú zapísať, do príslušného konfiguračného registra, sú na AD zbernicu vystavené hneď v nasledovnom cykle po adresnej fáze.

7.3 Alokácia pamäti na PCI

Každé PCI zariadenie, ktoré sprístupňuje určitú internú pamäť alebo riadiace registre, musí požiadať, prostredníctvom registrov bázových adries vo svojom konfiguračnom priestore, o pridelenie

potrebného pamäťového priestoru v rámci celého systému. Prideľovanie pamäti prebieha práve v rámci konfiguračného prenosu typu NULA.

Výrobca PCI zariadenia špecifikuje svoju požiadavku na pamäť ta, že napevno nastaví spodný bit registra bázovej adresy. Najnižší bit vyjadruje typ pamäte, o ktorú zariadenie žiada. Ďalej nasleduje oblasť, v ktorej je špecifikovaná veľkosť požadovanej pamäte. Posledná časť registra bázovej adresy zostáva nenastavená a slúži pre zápis adresy prideleného priestoru zo strany systémového software. Postup pri obsluhe požiadavku na alokáciu pamäti prebieha v nasledujúcich krokoch:

1. Systémový software najprv zapíše do neinicializovaného registra bázovej adresy samé jednotky. Bity, ktoré sú napevno nastavené výrobcom nebudú týmto zápisom ovplyvnené.
2. Systémový software si prečíta hodnotu registra bázovej adresy a zistí tak, požadovaný typ a množstvo pamäti, ktoré adaptér potrebuje. Po získaní všetkých požiadaviek software pridelí zariadeniu pamäťový priestor.
3. V poslednom kroku vykoná systémový software zápis hodnoty do registru bázovej adresy, ktorá zodpovedá bázovej adrese prideleného priestoru.

31 20 4 3 2 1 0
1) xxxxxxxxxxxx 0000000000000000 1 0 0 0
31 20 4 3 2 1 0
2) 111111111111 0000000000000000 1 0 0 0
31 20 4 3 2 1 0
3) 100000000000 0000000000000000 1 0 0 0
0=Non-prefetchable
1=Prefetchable
00=32 bit address
10=64 bit address
0=Memory request
1=I/O request

Obr. č. 23. Príklad alokácie 1MB pamäte [6]

Obrázok č. 23. zobrazuje príklad pridelenia 1 MB pamäti zariadeniu. Zariadenia požaduje adresný 32 bitový priestor pamäte. To že žiada práve veľkosť 1 MB určuje počet natvrdo nastavených bitov na hodnotu log. 0. V tomto prípade sú to pozície [19:4] registra, teda zariadenie žiada o pamäť veľkosti 2^20 = 1 MB. V poslednom kroku pridelí systémový software zariadeniu pamäť o veľkosti 1MB s bázovú adresou hodnoty 8000 0000 hexa(2 GB).

7.4 Konfiguračná logika

Konfiguračnú logiku som implementoval ako blok obsluhujúci konfiguračný prenos. Jej činnosť pozostáva z dvoch hlavných úloh:

1. Rozpoznať, že sa v adresnej fáze snaží systémový software o konfiguračný prenos a adresovaným zariadením je dané zariadenie.
2. Zabezpečiť zápis a čítanie v konfiguračnom priestore zariadenia.

V úlohách sa konfiguračná logika podobá na blok adresný dekodér. V podstate je jeho činnosť rovnaká, len s rozdielnym cieľovým blokom zápisu a čítania. Preto som pre riadenie tohto bloku využil blok, ktorý riadi aj adresný dekodér, arbitra funkcie TARGET. Arbiter riadi konfiguračnú logiku pomocou tých istých signálov ako adresný dekodér. Celý proces riadenia som popísal v kapitole 5.2.2 Riadiaca činnosť arbitra.

To, že obe jednotky sú riadené rovnakými signálmi a ich činnosť nebude prebiehať synchrónne. Teda sa nebude naraz zapisovať do konfiguračnej logiky aj do pamäte zariadenia zabezpečuje signál **Operation**, ktorý je nastavovaný v inštrukčnom dekodéry. Tento signál je nastavený podľa operácie, ktorá je deklarovaná na C/BE zbernici počas adresnej fázy. A teda jasne rozlišuje aká operácia sa bude v nasledujúcom prenose vykonávať.

Konfiguračná logika počas adresnej fázy vzorkuje svoj IDSEL signál a AD zbernicu. Ak dôjde k detekcií aktívneho IDSEL signálu a identifikácií konfiguračného prenosu typu NULA a identifikácií prevádzkovanej funkcie zariadením, konfiguračná logika nastaví adresované dvoj slovo z konfiguračného priestoru podľa hodnoty AD zbernice na bitoch 7 až 2 do adresného čítača. Adresný čítač sa v tomto prípade inkrementuje o hodnotu jedna. Zabezpečuje sa tak možnosť blokového čítania alebo zápisu do konfiguračnej pamäte.

Implementáciu napevno nastavených častí konfiguračného priestoru, ako napríklad pevne nastavených registrov ID zariadenia a ID výrobcu, som implementoval v procese zabezpečujúcom čítanie a zápis do konfiguračnej logiky. Ak proces rozpozná adresu s pevne nastaveným registrom, automatický nastaví jeho hodnotu na AD zbernicu, alebo v prípade zápisu do daného registra zapíše prednastavenú hodnotu.

7.5 Konfiguračný priestor zariadenia

Konfiguračný priestor som implementoval ako register obsahujúci 16 dvoj slov. Dvoj slová sú adresované pomocou 5 bitového adresného signálu. Pre adresovanie by bolo možné použiť aj 4 bity, ale v konfiguračnom prenose typu NULA sa používa 5 bitové adresovanie (kapitola 7.2).

Následne popíšem aké hodnoty dom priradil je jednotlivých registrov, z ktorých sa skladá konfiguračný priestor.

Register ID výrobcu som nastavil na hodnotu 0001 hexa. Na takú istú hodnotu som nastavil aj register udávajúci ID zariadenia. Status register som nastavil nasledovne

- 5 bit – log. 0. Zariadenie pracuje na 33 MHz.
- 6 bit – log. 0. Zariadenie nepodporuje užívateľom definované konfigurácie
- 7 bit – log. 0. Zariadenie nepodporuje FastBack-to-Back prenos.
- 8 bit – log. 0. Vyžadované INICIÁTOROM.
- bity 9 a 10 – hodnota "10"bin. Pomalé nastavenie signálu DEVSEL.
- 11 bit – log. 0. Nemožnosť indikácie Target-Abort.
- 12 bit – log. 0. Vyžadované INICIÁTOROM.
- 13 bit – log. 0. Vyžadované INICIÁTOROM.
- Bity 14 a 18 – log. 1. Detekcia chybnej systémovej a dátovej parity.

Príkazový register som nastavil nasledovne:

- 0 bit – log. 0. Nepodporuje I/O prístup.
- 1 bit – log. 1. Podporuje pamäťový prístup.
- 2 bit – log. 0. Zariadenie nie je INICIÁTOROM.
- 3 bit – log. 0. Zriadenie nesleduje špeciálne cykly.
- 4 bit – log. 0. Zariadenie nepodporuje Memory Write and Invalidate.
- 5 bit – log. 0. Zariadenie nepodporuje VGA snooping.
- 6 bit – log. 1. Zariadenie sleduje chybu parity.
- 7 bit – log. 0. Zariadenie nevkladá čakací stav do každej dátovej fázy.
- 8 bit – log. 1. Zariadenie riadi signál SERR.
- 9 bit – log. 0. Vyžadované INCIÁTOROM.
- bity 10 až 15 sú rezervované a nastavené na log. 0.

Hodnotu registru opráv som zvolil tak isto ako v prípade registrov ID výrobcu a ID zariadenia na hodnotu 0001 hexa.

Zariadenie som vybavil jedným 32 bytovým prístupovým registrom. Preto musí mat prvý register bázovej adresy celkovú prednastavenú hodnotu FFFFFFE8 hexa. Pomocou tejto hodnoty zariadenie žiada o pridelenie 32 bytov systémovej pamäte pre adresovanie tohto registra. Prvých 5 bito je natvrdo nastavených na hodnotu "01000". Ostatné bity môžu biť prepísané počas konfiguračného prenosu ako som to popísal v kapitole 7.3 Alokácia pamäti na PCI.

8 Simulovanie a syntéza

Ako vývojové prostredie pre realizáciu PCI radiča som zvolil programový produkt od firmy Xilinx ModelSim XE III 6.3c. Toto vývojové prostredie ponúka pre vývojára okrem tvorby VHDL kódu možnosť otestovať správnosť svojho kódu prostredníctvom simulácií. Pre simuláciu v ModelSime sa využíva tzv. test bench súbory.

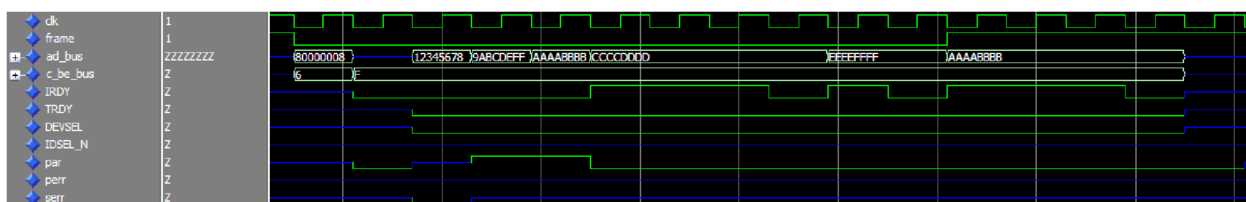
8.1 Simulácia

Test bench súbory predstavujú akési okolie modelovaného zariadenia ku ktorému sa zariadenie pripojí a nevyhnutne obsahuje nasledujúce časti:

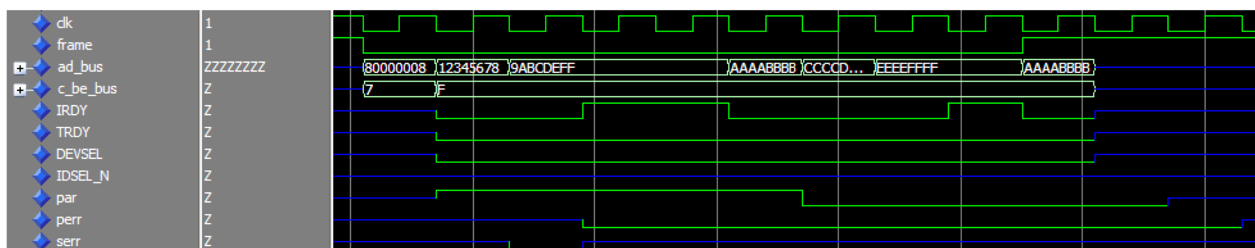
- Inštanciu vyvíjanej komponenty označenou ako UUT (Unit Under Test)
- Generátorov testovacích vektorov
- Monitorovanie a overovanie reakcií UUT

V samotnom test bench súbore **PCI_core_tb.vhd** som simuloval komunikáciu prostredníctvom PCI zbernice s vyvíjaným PCI radičom. Nadefinoval som adresovanie zariadenia a signály nastavované INICIÁTOROM počas komunikácie ako aj počas konfiguračného prenosu typu NULA. Testovacie vektory, ktoré sa pripájajú k testovanej komponente som nastavoval v čase pomocou signálu **wait** a priradenia hodnoty vstupných signálov v jednotlivých časových cykloch.

Simulácia sa spúšťa vo vývojovým prostredím ModelSimu v adresári **sim** v projektovom adresári spustením testovacieho skriptu **PCI_core.fdo** (príkaz: do PCI_core.fdo). V tomto skripte sú nadefinované sledované simulované signály jednotlivých častí vyvíjanej komponenty a čas simulácie.



(a)



(b)

Obr. č. 24. Výstup simulácie – (a) čítací prenos, (b) zapisovací prenos

Obrázok č. 24. zachytáva výstup procesu simulácie. Po spustení simulácie v ModelSime vyskočí okno **Wawe**, kde sa vykreslia definované signály v skripte **PCI_core.fdo**. Na časti obrázku (a) je zachytená simulácia čítacieho prenosu. Po adresnej fázy a rozpoznaní adresy zariadenia nasleduje nastavenie dát zariadením na AD zbernicu po vratnom cykle. Daný priebeh zapisovacieho prenosu zodpovedá vzorovému priebehu zobrazenému na obrázku č. 18. Ktorý vlastne slúžil ako vzor mojej implementácií. N časti obrázku (b) je zachytená simulácia zapisovacieho prenosu prostredníctvom PCI zbernice.

8.2 Syntéza

Pre proces syntézy VHDL kódu vyvíjaného PCI radiča, do konfiguračného reťazca pre FPGA obvod som zvolil vývojové prostredie od firmy Xilinx ISE 10.1. Jedná sa o vývojové prostredie, ktoré poskytuje užívateľovi proces automatickej syntézy.

Pred štartom syntézy je potrebné zvoliť:

- vstupné VHDL súbory
- cieľové zariadenie FPGA
- obmedzujúce podmienky syntézy

Ako cieľový FPGA obvod proces syntézy som zvolil obvod, ktorým je osadená karta pre ktorú som vyvíjal PCI radič. Teda obvod od firmy Xilinx **Spartan II. xc2S150-5Qpq208**. Časovú obmedzujúcu podmienku som zvolil optimalizáciu na 33 MHz. Pre obmedzujúcu podmienku rozmiestnenia vstupných a výstupných pinov na čipe, som musel vytvoriť súbor, v ktorom som presne priradil jednotlivé signály k pinom FPGA, tak aby sedeli s PCI špecifikáciu. Presné priradenie pinov FPGA obvodu k pinom PCI konektora na vývojovej karte, je popísané v manuály výukovej karty. Súbor popisujúci priradenie pinov, musí mať v ISE vývojovom prostredí príponu .ucf. Celý súbor som nazval **PCI_core.ucf**.

Syntéza prebehla celkovo úspešne. Na obrázku č. 25. Je zachytená časť výstupného logovacieho súboru popisujúceho percentuálne využite jednotlivých častí FPGA obvodu, pre implementáciu vyvíjaného PCI radiča.

```
Device utilization summary:
---------------------------

Selected Device : 2s150pq208-5q

 Number of Slices:                    1374  out of   1728    79%
 Number of Slice Flip Flops:          1172  out of   3456    33%
 Number of 4 input LUTs:              2188  out of   3456    63%
 Number of IOs:                         47
 Number of bonded IOBs:                 47  out of    140    33%
 Number of GCLKs:                        3  out of      4    75%
=========================================================================

Process "Synthesis" completed successfully
```

Obr. č. 25. Výsledok procesu syntézy – využitie obvodu FPGA

Výstupnou časťou procesu syntézy je binárny programovací súbor, určená pre presný typ FPGA obvodu. Tento súbor má príponu .bit. Celý súbor má názov **pci_core.bit**.

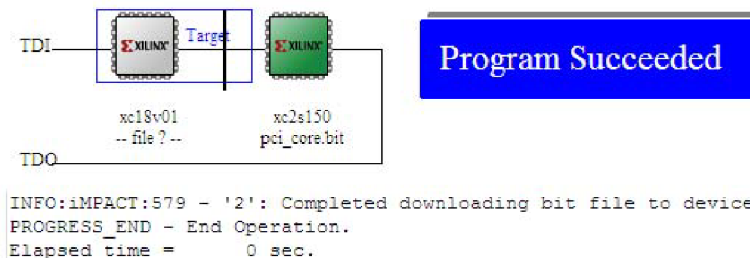
8.3 Programovanie obvodu

Pre programovanie obvodu FPGA na výukovej karte, bolo treba ku karte pripojiť programátor s rozhraním JTAG. Jedná sa o programovanie rozhranie, ktorá sa okrem programovania FPGA obvodov používa aj pre programovane mikroprocesorov.

Okrem neho som musel na kartu priviesť externé napájanie 5 V, aby bolo možné kartu vôbec naprogramovať. Pripojenie JTAG programátora je zachytený na obrázku č. 26.

Obr. č. 26. Programovanie obvodu FPGA prostredníctvom rozhrania JTAG

Ako software pre programovanie som zvolil program iMPACT 10.1 od firmy Xilinx, ktorý je priamo súčasťou ISE 10.1. K JTAG programátoru sa iMPACT pripája prostredníctvom USB rozhrania. Software iMPACT pracuje v grafickom režime. Na obrázku č. 27. je zachytené úspešné naprogramovanie FPGA obvodu na vývojovej karte v prostredí iMPACT.



Obr. č. 27. Programovanie obvodu FPGA v software iMPACT

9 Ovládač

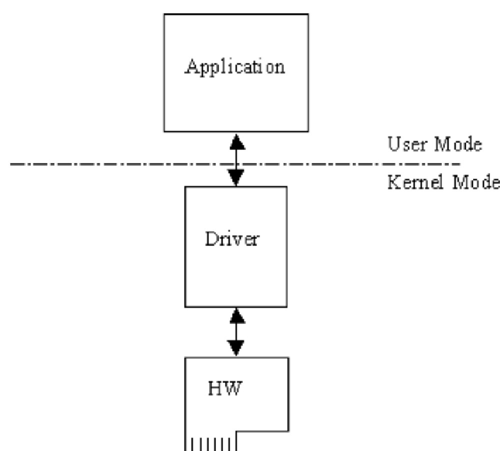
Ovládač zariadenia je softwarový segment, ktorý poskytuje rozhranie medzi operačným systémom a špecifickým hardwarovým zariadením, ako napríklad, terminály, disky, video karty, sieťové média atď. Ovládač umožňuje uviesť zariadenie do prevádzky, alebo ho vypnúť, nastaviť hardwarové parametre na zariadení, preniesť dáta zo zariadenia do jadra a naopak a spravovať chyby na zariadení. Ovládač sa správa ako prekladateľ medzi zariadením a programom, ktorý využíva dané zariadenie. Každé zariadenie má svoju vlastnú sadu špecializovaných príkazov, ktoré pozná len ovládač. Na rozdiel od ovládačov, väčšina programov sprístupňuje zariadenia použitím všeobecných príkazov. Ovládač preto musí byť schopný prijať všeobecné príkazy z užívateľských aplikácií a následne ich preložiť na špecializované príkazy zariadenia.

9.1 Druhy ovládačov

Poznáme niekoľko druhov ovládačov. My si priblížime jeden druh. tzv. monolitický ovládač. Okrem monolitických ovládačom poznáme ešte [8]:

- vrstvené ovládače (Layered drivers)
- miniportové ovládače (Miniport drivers)

Monolitické ovládače sú ovládače, ktoré stelesňujú všetku funkcionalitu potrebnú pre podporu hardwarového zariadenia. Monolitický ovládač je sprístupnení jednej, alebo viacerým užívateľským aplikáciám a priamo riadi hardware. Ovládač komunikuje s aplikáciou prostredníctvom I/O riadiacich inštrukcií a riadi hardware použitím volaní funkcií differentWDK, ETK, DDI/DKI. Monolitické ovládače sú podporované vo všetkých operačných systémoch.



Obr. č. 28. Monolitický ovládač [8]

Okrem tohto delenia ovládače delíme aj z pohľadu operačného systému, pre ktorý je ovládač vyvíjaný [8].

WDM ovládače

WDM (Windows Driver Model) ovládače sú ovládače v tzv. kernel móde v rámci rodín operačných systémov Windows NT a Windows 98. Rodina operačných systémov Windows NT zahrňuje tiež systémy Windows Vista /Server 2008/Server 2003/XP/NT 4.0. Rodina Windows 98 zahrňuje Windows 98 a Windows Milenium.

WDM pracuje pomocou tzv. kanálov z ovládača zariadenia do častí kódov, ktoré sú integrované do operačného systému. Tieto časti operačného systému pracuje na spodnej úrovni práce so správou vyrovnávacích pamäti, vrátane DMA a Plug-and-Play (PNP) zariadení. WDM ovládače sú PNP ovládače, ktoré podporujú tzv. silové riadiace protokoly, a zahrňujú monolitické, vrstvené aj miniportové ovládače.

VxD ovládače

Tieto ovládače sú vo Windows 95/98/Me nazývané virtuálne ovládače. VxD zdedili meno po prípone .VxD, ktorou sú označované. VxD ovládače sú monolitického typu. Poskytujú priami prístup k hardware v privilegovanom móde operačného systému.

Unixové ovládače

V klasickom Unixovom modely sú zariadenia členené do jednej z troch kategórií [8]:

- znakové zariadenia
- blokové zariadenia
- sieťové zariadenia

Ovládače pre tieto zariadenia sú pomenované podľa druhu zariadenia, pre ktorý sú určené. Unixové ovládače predstavujú kódové jednotky spojené s jadrom systému, ktoré bežia v tzv. privilegovanom kernel móde. Prístup k ovládačom z užívateľského režimu aplikácií prebieha pomocou systémových súborov. Inými slovami, zariadenia sa zdajú aplikáciám, ako špeciálne prístrojové súbory, ktoré môžu byť otvorené. Unixové ovládače sú tvorené ako monolitické alebo vrstvené.

Linuxové ovládače

Ovládače pre operačný systém Linux sú založené na modely klasických Unixových ovládačov. Prinášajú však zo sebou aj nové charakteristické vlastnosti. Pod Linuxom, môžu byť blokové zariadenia sprístupnené ako znakové, ako v Unixe, ale tiež majú blokovo orientované rozhranie, ktoré je neviditeľné v užívateľskom alebo aplikačnom móde. V Unixe sú ovládače spojene s jadrom a systém musí byť po nainštalovaní nového ovládača reštartovaný. V Linuxe sa zaviedol pojem dynamicky načítateľný ovládač, tiež nazývaný modul, ktorého načítanie alebo odstránenie nevyžaduje

reštart systému. V Linuxe môže byť ovládač napísaný a staticky nalinkovaný, alebo napísaný formou modulu a dynamicky načítaný. Vďaka tomu je využitie pamäte efektívnejšie, nakoľko moduly môžu byť napísané pre vlastný hardware a v prípade, že sa zariadenie nenájde môžu byť modul dynamicky odobratý. Ako Unix aj Linux podporuje monolitické a vrstvené ovládače.

Každý ovládač musí mať tzv. vstupný bod. Niečo ako funkcia main() v C. Vo Windows sa vstupný bod nazýva **DriverEntry()** a v Linuxe **init_module().** Keď operačný systém načítava ovládač zariadenia, tak volá tento vstupný bod. Operačné systémy sa líšia v tom ako linkujú ovládač zariadenia do systému. Vo Windows sa linkovanie vykonáva pomocou tzv. **INF** súboru, ktorý registruje zariadenie pre prácu s ovládačom. Táto asociácia sa vykonáva pred načítaním **DriverEntry()**. V Linuxe je linkovanie medzi zariadením a ovládačom definované v rutine **init_module()**. Tá zahrňuje callback funkcie, ktoré definujú pre aký hardware je ovládač vytvorený.

Ovládač môže vytvoriť inštanciu zaridenia, a tak umožní aplikáciám otvoriť tzv. handle, cez ktorý môžu aplikácie komunikovať so zariadením. Aplikácie komunikujú s ovládačom pomocou prístupovej API (Application Program Inteface). Tie umožňujú zariadeniam vytvárať handly na zariadenia, a následne pracovať zo zariadením. Teda otvoriť zariadenie, čítať zo zariadenia, zapisovať do zariadenia a nakoniec prístup ku zariadeniu korektne ukončiť zatvorením zariadenia.

9.2 Tvorba ovladača v prostredi Jungo WinDriver

Ovládač pre výukovú kartu v prostredí operačného systému Windows XP som vytvoril vo vývojovom prostredí Jungo WinDriver v10.01. Jedná sa o vývojový nástroj, ktorý uľahčuje vývoj ovládačov pre monolitické zariadenia. WinDriver zahrňuje grafické vývojové prostredie, diagnostické a ladiace obslužné programy, ktoré umožňujú rýchlo vyvinúť vysoko výkonný ovládač.

Jungo WinDriver zabezpečuje nasledujúce funkcie:

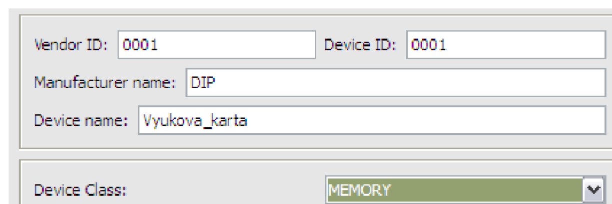
- Okamžitý prístup na hardware. Sprístupnenie hardware s použitím grafického užívateľský režimu. S automatickým vygenerovaním kódu.
- Generovanie špecifického kódu pre každý hardware. Umožňuje namieru upraviť a presne špecifikovať požiadavky hardware na ovládač.
- Výkonovú optimalizáciu. Jadro pluginu umožňuje prechod z užívateľského režimu to režimu jadra, pre dosiahnutie optimálneho výkonu.
- Multipodpora operačných systémov. Windows 2000/XP/Wista, Linux IA64, Solaris atď.
- Prenositeľnosť medzi operačnými systémami.

WinDriver umožňuje užívateľovi presne špecifikovať vlastnosti hardwaru, pre ktorý je určený vyvíjaný ovládač. Ako prvé je potrebné zvoliť typ štandardu cez ktorý sa bude dané zariadenie do systému pripojovať.

WinDriver podporuje tvorbu ovládačov pre nasledovné štandardy [9]:

1. PCI (špecifikácia 2.2)
2. PCI-X
3. PCI-Expres
4. PICMG 2.0 R3.0
5. USB

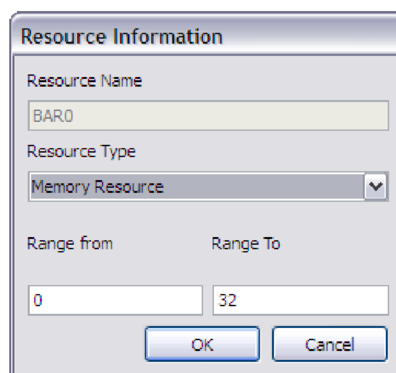
Po zvolení štandartu PCI som zvolil ID výrobcu a ID zariadenia tak ako som ich nakonfiguroval v konfiguračnom priestore vo VHDL kóde. Viď. obrázok č. 29. Ďalším vstupným parametrom tvorby ovládača je druh zariadenia, pre ktoré je ovládač vytváraný. V mojom prípade som implementoval PCI radič s 32 bytovým registrom. Preto som zvolil triedu zariadenia ako pamäťové zariadenie.



Obr. č. 29. Konfigurácia vo WinDriver

Po zvolení predchádzajúcich nastavení dôjde k vygenerovaniu tzv. .inf súboru. Následne je už len potrebné zvoliť pamäť alebo I/O priestor, ktorý zariadenie vyžaduje. V mojom prípade som len pridal už spomínanú pamäť o veľkosti 32 bytov. Viď. obrázok č. 30.

Po zadaní všetkých vlastností, pamäťových a I/O priestorov je už len potrebné pristúpiť ku generovaniu kódu ovládača. Pri generovaní kódu je potrebné zadať, aký zdrojový kód sa bude generovať. Ja som zvolil ANSI C. Po zvolení výstupného jazyka, dôjde k vygenerovaniu zdrojového kódu ovládača aj s príslušným Makefile súborom.



Obr. č. 30. Zadanie pamäte zariadenia

Vygenerovaný ovládač pozostáva z funkcií pre:

- Inicializáciu zariadenia
- Otvorenie zariadenia
- Zatvorenie zariadenia
- Pre Plug and Play a správu napájania
- Prístup k adresnému priestoru
- Čítanie a zápis do pamäte pridelenej zariadeniu

Pre správnu funkcionalitu musí ovládač vytvorení v prostredí WinDriver obsahovať hlavičkový súbor **windrvr.h**, ktorý už obsahuje hlavičkové súbory **wdc_lib.h** a **wdc_defs.h**, ktoré umožňujú užívateľovi pracovať s **WDC**. WDC – „WinDriver Card" je API, ktoré poskytuje vhodný užívateľskú obálku pre základný rozhranie WinDriver PCI/ISA/PCMCIA/CardBus WD, ktoré je na nižšej vrstve.

Knižnica pre prácu zo zariadením sa inicializuje vo funkcií **LibInit()**. V nej sa pomocou funkcie WDC_SetDebugOptions() nastavia tzv. ladiace možnosti, ktoré sa predajú do WDC. Následne nasleduje volanie funkcie **WDC_DriverOpen()**, ktorá vytvorí tzv. handle na zariadenie a inicializuje WDC knižnicu. Handle si môžeme predstaviť ako štruktúru popisujú zariadenie. Zo zariadením sa bude pracovať prostredníctvom handlu. Vo funkcií **LibUninit()** sa volá inverzná funkcia **WDC_DriverClose()**, ktorá uzavrie knižnicu WDC a tiež hnadle na zariadenie.

V ďalšej časti už nasleduje funkcia **DeviceOpen()**, ktorá otvára hande pre prácu zo zariadením, stará sa o alokáciu pamäte potrebnej pre komunikáciu zo zariadením. Inverzná funkcia, ktorá sa stará o korektné uzavretie handlu nesie názov **DeviceClose()**.

Následné sú implementované funkcie **WDC_EventHandler()** a **WDC_EventRegister()**. Sú to callback funkcie volané pre registráciu udalostí a užívateľskom režime. Nasleduje funkcia **GetAddrSpaceInfo()**, ktorá pracuje z konfiguračným priestorom zariadenia. Prehľadáva a vypisuje informácie uložené v tomto priestore. Ako posledné sú implementované funkcie čítania a zápisu do pamäte zariadenia **ReadMemmory()** a **WriteMemmory().**

Všetky vytvorené funkcie používajú pre komunikáciu so zariadením pomocou WDC funkcií z WDC API, čo ukazuje aj nasledovná ukážka kódu funkcie pre čítanie z pamäte zariadenia. Tá využíva funkciu **WDC_REadAddr32()**, ktorá vráti prečítanú 32 bitovú hodnotu zo zariadenia identifikovaného pomocou handlu. Následne potrebuje funkcia ukazateľ na pamäť a veľkosť offsetu pre určenie presnej pozície, z ktorej má čítať.

```
UINT32 DRIVER_ReadMemmory (WDC_DEVICE_HANDLE hDev)
{
    UINT32 data;
    WDC_ReadAddr32(hDev, DRIVER_Memmory_SPACE, DRIVER_Memmory_OFFSET, &data);
    return data;
}
```

10 Testovanie

Testovanie realizácie PCI radiča na vývojovej karte s obvodom FPGA som vykonal za pomoci počítača s voľným PCI slotom. Počítač musel byt vybavený operačný systémov Windows XP, nakoľko ovládač, ktorého tvorbu som popísal v predchádzajúcej kapitole, bol určený práve pre tento operačný systém. Následne som musel pre testovanie funkčnosti zariadenie vytvoriť aplikáciu, ktorá by pomocou ovládača komunikovala zo zariadením a tak otestovala jeho funkčnosť.

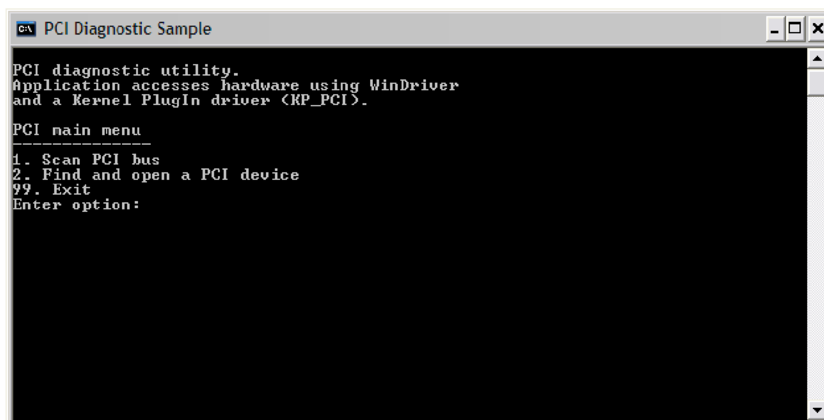
10.1 Testovacia aplikácia

Vývojové prostredie Jungo WinDriver umožňuje okrem vývoja ovládačom pre zariadenia pripájané k počítaču prostredníctvom PCI rozhrania aj tvorbu aplikácií testujúcich funkčnosť daných zariadení. Túto možnosť som využil ja a spolu z ovládačom som vytvoril aplikáciu, ktorá otestuje komunikáciu s vývojovou kartou s obvodom FPGA prostredníctvom PCI rozhrania.

Aplikácia pracuje v príkazovom riadku. Umožňuje nasledovné operácie:

- Postupne prejsť všetky zariadenia pripojené k PCI zbernici na danom počítači a postupne vypísať nasledovné položky konfiguračného priestoru zariadenia: obsah registru ID výrobcu, obsah registra ID zariadenia, slot, ktorý zariadene používa, počet funkcií zariadenia, prerušenie ak ich zariadenie podporuje a rozsah pamäti, ktorá bola zariadeniu pridelená. Ďalej vypíše rozsah pridelenej I/O pamäti, ak je zariadeniu nejaká pridelená.
- Podľa zadaného obsahu registra ID výrobcu a ID zariadenia vyhľadať dané zariadenie, pripojiť sa k nemu a vypísať: obsah registru ID výrobcu, obsah registra ID zariadenia, slot, ktorý zariadene používa, počet funkcií zariadenia, prerušenie ak ich zariadenie podporuje a rozsah pamäti, ktorá bola zariadeniu pridelená a I/O priestor pridelený zariadeniu.

Zdrojový kód testovacej aplikácie je vytvorený, ako v prípade ovládača v jazyku C. Aplikácia používa pre komunikáciu zo zariadením ovládač, ktorý bol vyvinutý pre testované zariadenie.

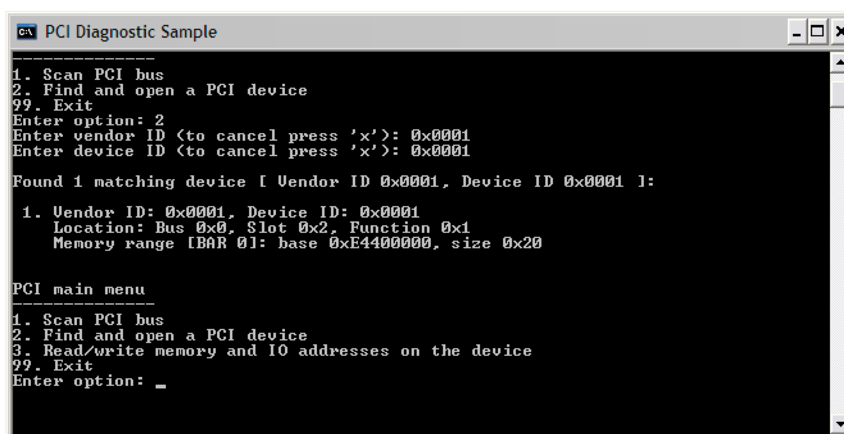


Obr. č. 31. Rozhranie testovacej aplikácie

10.2 Test

Samotný test funkčnosti implantovaného radiča PCI v jazyku VHDL som vykonal za pomoci testovacej aplikácie v prostredí operačného systému Windows XP. Po pripojení karty na PCI zbernicu a nainštalovaní implementovaného ovládača, som spustil testovaciu aplikáciu.

Na obrázku č. 31. je zobrazený priebeh testovania. Z uvedených možností testovania, ktoré aplikácia ponúka, som zvolil možnosť nájsť zariadenie podľa obsahu registra ID výrobcu a registra ID zariadenia a vypísať na štandardný výstup hodnoty popísané v predchádzajúcej podkapitole 10.1.



Obr. č. 32. Test implementácie PCI radiča, za pomoci testovacej aplikácie

Obrázok č. 31. zachytáva úspešné prevedenie testu funkčnosti. Teda testovacia aplikácia otvorí hľadané zariadenie a prečíta obsah požadovaných registrov konfiguračného registra. Ako je vidno zariadeniu bola pridelená bázová adresa E4400000 hexa. Program ďalej vypísal aj veľkosť pamäte pridelenej zariadeniu 20 hexa čo je v desiatkovej sústave 32 bytov. To presne odpovedá implementácií, ktorú som vykonal v jazyku VHDL. Týmto testom som potvrdil funkčnosť implementácie radiča PCI pre komunikáciu s vývojovou kartou s obvodom FPGA.

11 Záver

Hlavným cieľom tejto diplomovej práce je navrhnúť, implementovať a otestovať PCI radič pre vývojovú kartu s obvodom FPGA. K celkovému úspešnému splneniu cieľov práce som musel postupovať v nasledujúcich krokoch.

V prvej časti som naštudoval technológie pre implementáciu projektu, či sa už jedná o rekonfigurovateľné obvody FPGA alebo jazyk popisujúci číslicové obvody VHDL.

V ďalšej časti som sa zameral na analýzu komunikácie na PCI zbernici a PCI zbernicu samotnú. Hlavne, čo sa týka zariadenia pripojeného k tejto zbernici. Každé zariadenie musí dodržiavať presne špecifikovaný komunikačný protokol, riadený pomocnými riadiacimi signálmi a samé musí generovať potrebné signály.

Celú obsluhu takejto komunikácie má na starosti radič PCI rozhrania, ktorým musí byť vybavené každé zariadenie pripojené k zbernici PCI. Po analýze som vykonal návrh takéhoto radiča. Navrhol som blokovú štruktúru PCI radiča, ktorý by plnil funkciu tzv. TARGET zariadenia pripojeného k PCI zbernici. Jedná sa o zariadenie, ktoré je len cieľom operácií na zbernici. Teda zariadenie, ktoré nemôže vyvolať operácie prenosu prostredníctvom zbernice, môže byť len cieľovým zariadením operácií. Pravdaže jednalo sa o predbežný návrh tohto zariadenia a vo fáze implementácie došlo ku zmenám v tomto návrhu na základe novo vzniknutých skutočností a možno aj opomenutých požiadaviek.

V jadre práce sa zaoberám implementáciou PCI radiča v jazyku VHDL. Najprv som sa zameral na čítací prenos z pohľadu TARGET zariadenia a dôkladne som ho rozanalyzoval. Priebeh komunikácie pomocou signálov zbernice PCI, bol v podstate cieľ, ktorý som chcel implementáciou docieliť. Ďalej rozoberám podrobne implementáciu jednotlivých blokov radiča, pričom sa zameriavam hlavne na arbitra funkcie TARGET, ktorý má na starosti riadenie ostatných blokov a celej komunikácie. Arbiter je implementovaný ako stavový automat, ktorého prechody medzi jednotlivými stavmi, sú podmienené stavmi riadiacich signálov PCI zbernice a internými signálmi z ostatných blokov radiča. Následne popisujem činnosť ostatných blokov radiča v čítacom a neskôr aj v zapisovacom prenose. Ako cieľový zariadenie som k radiču implementoval register o veľkosti 32 bytov. Zariadenie umožňuje čítať a zapisovať do tohto registra.

V záverečnej časti popisujem priebeh testovania implementovaného radiča na cieľovom hardware. Pre testovanie bolo potrebné najprv vykonať syntézu implementovaného VHDL kódu na konfiguračný bitový súbor pre FPGA obvod a tento súbor nahrať do pamäte tohto zariadenia. Následne som implementoval za pomoci aplikácie Jungo WinDriver ovládač pre kartu a aplikáciu, ktorá by kartu otestovala. Ovládač som implementoval pre operačný systém Windows XP.

Ako koncový bod celej práce som pristúpil k testovaniu implementovaného radiča. Testovanie som vykonal prostredníctvom počítača s voľným slotom PCI a operačným systémom Windows XP.

Po nainštalovaní implementovaného ovládača a testovacej aplikácie som vykonal test. Testovanie prebehlo úspešne nakoľko karta odpovedala testovacej aplikácií. Teda implementovaný radič riadil komunikáciu prostredníctvom PCI zbernice správne. Tým sa splnil hlavný cieľ práce a môžem teda povedať, že práca bola úspešne dokončená.

Literatúra

[1] Sekanina L. Rekonfigurotavelná výpočetní zařízení. Biologií inspirované počítače, Přednáška 3, FIT VUT v Brně, 2008

[2] Xilinx. Spartan-II FPGA Data Sheet, dokument dostupný na url: http://www.xilinx.com/support/documentation/data_sheets/ds001.pdf, Online navštívené 23. 12. 2008

[3] Jelemenská K. Základy jazyka VHDL, FIT STU Bratislava, dokument dostupný na url: http://www.kme.elf.stuba.sk/kme_new/buxus/docs/predmety/PNIO/VHDL_Danka_Durackova.ppt, Online navštívené 23. 12. 2008

[4] Kořenek J., Marínek T. Syntéza, Pokročilé číslicové systémy, FIT VUT v Brně, 2008

[5] Insighnt MEMEC, SPARTAN – II PCI DEVELOPMENT KIT, ISBN I-888-488-4133 Ext.205 2001

[6] Kořenek J., Marínek T. Návrh externích adaptérů a vestavěných systémů NAV – Studijní opora, FIT VUT v Brně, 2008

[7] PCI Local Bus Specification, Revision 2.2, Dokument dostupný na url: http://www.ece.mtu.edu/faculty/btdavis/courses/mtu_ee3173_f04/papers/PCI_22.pdf, Online navštívené 2. 1. 2009

[8] Jungo. WinDriver PCI 10.01 užívateľský manuál, dokument dostupný na url: http://www.jungo.com/st/support/support_windriver.html, Online navštívené 3. 5. 2009

[9] Jungo. WinDriver PCI 10.01 Data Sheet, dokument dostupný na url: http://www.jungo.com/st/support/support_windriver.html, Online navštívené 3. 5. 2009

Zoznam príloh

Príloha 1: CD so zdrojovými textami