České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů

Diplomová práce

USB Logic analyzer

Vít Bernatík

Vedoucí práce: Ing. Pavel Kubalík

Studijní program: Elektronika a informatika, strukturovaný, navazující magisterský
Obor: Výpočetní technika (2612T051)
prosinec 2008

Poděkování

Rád bych poděkoval především všem lidem, kteří se podíleli na vzniku internetu, protože bez něj bych se nikdy nedostal včas a ve stejné míře k potřebným materiálům a literatuře. Dále bych chtěl poděkovat všem lidem, kteří vytvářejí neocenitelné odborné články, bez kterých by mnohé části nebylo možné v daném čase implementovat. Z odborných článků mě velice ovlivnily především ty od pana Cummings a Alfke, ale i mnohých jiných. Dále patří dík všem přispěvatelům na různá technická fóra a především pak Xilinx user community, kteří pomohli zdolat nástrahy vývojového prostředí EDK. Dík patří taky mému vedoucímu práce Pavlu Kubalíkovi, nejen za vytrvalou motivaci, ale především za podnětné rady ohledně řešení mnoha problémů v průběhu implementace. Také děkuji panu Janu Schmidtovi za vysvětlení EMI jevů v elektrických obvodech, které nás opomněli naučit v průběhu studia. V neposlední řadě děkuji mým rodičům, kteří mě živili po celou dobu studia a bez jejichž zázemí by také tato práce nikdy nevznikla. Dík patří všem ostatním, kteří mě v průběhu let, ať již přímo či nepřímo učili, ač zde není prostor abych je všechny jmenovitě uvedl.

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Havířově dne 1.1. 2009 ....

Abstract

This paper describes design and detail analysis of fast and relatively cheap logic analyzer. Achieved parameters of outcome product are sampling rate at 100MHz for 32 bits wide measuring port. Measuring and data collection is realized in dedicated hardware, powered by FPGA Virtex-4. Control and data interpretation is realized as a software solution on PC platform for Windows XP operating system. Interface between dedicated hardware and PC software application is via fast USB 2.0 interface. Main parts of work are foremost design of fast USB 2.0 interface framework, DDR interface solution and gain experience in extensive VHLD design especially in EDK tool from Xilinx.

Abstrakt

Tato práce obsahuje návrh a podrobnou analýzu rychlého a relativně levného logického analyzátoru. Parametry výsledného produktu jsou vzorkovací frekvence 100MHz pro 32 bitový měřící port. Měření a sběr dat je řešen dedikovaným hardwarem založeným na FPGA obvodu Virtex-4. Ovládání a interpretace dat pak je řešena softwarově na platformě PC v operačním systému Windows XP. Interface mezi měřícím zařízením a PC je realizován rychlou sběrnicí USB 2.0. Stěžejní části práce jsou především rychlé USB 2.0 rozhraní, rozhraní k DDR pamětem a získání zkušeností v navrhování komplexních zařízení pomocí jazyka VHDL a prostředí EDK od firmy Xilinx.

Obsah

1. Úvod 1
1.1 Možnosti dnešních špičkových logických analyzátorů 2
1.2 Možnosti levných logických analyzátorů 2
1.3 Vyvíjený analyzátor 2
2. Specifikace 3
2.1 Upřesňující požadavky na formu realizace systému 3
2.1.1 Softwarová část systému 3
2.1.2 Hardwarová část systému 3
2.2 Revize upřesňujících požadavků na realizaci systému 4
3. Analýza 5
3.1 USB IO deska 6
3.1.1 USB Konektor 7
3.1.2 USB elektrická norma 7
3.1.3 USB protokoly 8
3.1.3.1 USB Transakce 8
3.1.3.2 Typy přenosu 10
3.1.4 USB deskriptory 14
3.1.4.1 Device deskriptor 14
3.1.4.2 Konfigurační deskriptor 14
3.1.4.3 Interface deskriptor 15
3.1.4.4 Endpoint deskriptor 15
3.1.5 USB norma shrnutí 15
3.1.6 Výběr čipu pro IO desku 15
3.1.7 Podrobná analýza čipu ISP1583BS 16
3.1.7.1 Analýza hardwarového rozhraní čipu ISP1583 16
3.1.7.2 Analýza relevantních registrů čipu ISP1583 19
3.2 FPGA USB Driver 23
3.3 DDR Paměť 23
3.3.1 Analýza kapacity 23
3.3.2 Analýza rychlosti 23
3.4 DDR Controller 24
3.4.1 Rozhraní DDR pamětí MT46V16M16-6T 25
3.4.1.1 Deselect, No Operation 26
3.4.1.2 Activate 26
3.4.1.3 Read 26
3.4.1.4 Write 27
3.4.1.5 Burst terminate 27
3.4.1.6 Precharge 27
3.4.1.7 Auto Refresh 28
3.4.1.8 Self Refresh 28
3.4.1.9 Load Mode Register 28
3.4.2 Inicializace DDR paměti 29
3.4.3 Proces čtení dat 30
3.4.3.1 Problém zpoždění na vodičích 30
3.4.3.2 Problém rychlé efektivní frekvence 31
3.4.4 Zhodnocení DDR kontroleru 42
3.5 Blok jádra logického analyzátoru 42
3.5.1 Trigger 43

3.5.1.1 Nastavování pozice triggeru relativně k oknu platných dat........44
3.5.2 Přenosy dat mezi dvěma časovými doménami 44
3.5.2.1 Přenos malého objemu dat..45
3.5.2.2 Přenos velkého objemu dat..46
3.6 PC aplikace..51
3.6.1 USB driver a aplikační rozhraní...51
3.6.1.1 HID Driver...51
3.6.1.2 Cypress Driver 52
3.6.1.3 Aplikační rozhraní 53
3.6.2 GUI aplikace 53
4. Implementace 54
4.1 USB IO Deska...54
4.1.1 Redukce pro platformu ML402...56
4.2 FPGA..57
4.2.1 FPGA USB Driver 57
4.2.2 DDR Kontrolér..58
4.2.3 Blok jádra logického analyzátoru...59
4.2.4 Firmware 61
4.2.5 PC Aplikace..62
4.2.6 Bridge mezi DDR kontrolérem a řadičem obvodu ISP1583...........65
5. Testování 67
5.1 Testbenche pro moduly 67
5.2 Funkční test USB rozhraní 67
5.3 Funkční test DDR paměti...70
5.4 Funkční test celého systému...70
6. Zhodnocení..71
6.1 Zhodnocení využití obvodu FPGA 71
7. Závěr...73
8. Přílohy na CD..74
9. Reference...75
10. Přílohy 77
10.1 Schéma USB IO desky 77

1. Úvod

Logický analyzátor je elektronické zařízení, které slouží k měření a zobrazování rychlých elektrických signálu v digitálních zařízeních. Tyto signály mohou trvat pouze několik desítek nanosekund a není je proto možné měřit běžnými zařízeními jako jsou voltmetry.

Logické analyzátory mají obvykle několik desítek měřících kanálů které jsou vzorkovány a měřeny ve stejné a ekvidistantní časové okamžiky. U logických analyzátorů je důraz kladen právě na rychlost měření a maximální počet měřících kanálů.

Dá se říct, že logický analyzátor je speciální forma digitálního osciloskopu. Nevýhoda logických analyzátorů proti osciloskopu je schopnost rozeznávat pouze dvě napěťové úrovně, což je ovšem při měření digitálních obvodů dostačující při vhodném zvolení rozhodovací úrovně. Výhody logického analyzátoru oproti osciloskopu jsou pak daleko větší počet měřících kanálů. Typický osciloskop má pouze 2 až 4 měřící kanály. Typický logicky analyzátor má 32 až 128 měřících kanálů. Logické analyzátory jsou často schopny zaznamenávat podstatně delší úsek měřených vzorků než osciloskopy. Logický analyzátor má často velice sofistikovaný systém pro nastavení spouštěcí podmínky zahrnující stav na více kanálech a často i posloupnost stavů. Srovnáme-li spouštěcí podmínky typického osciloskopu, zjistíme, že je k dispozici obvykle jen napěťová úroveň na jednom kanálu. Lepší osciloskopy dokáží nastavit spouštěcí podmínku například na strmost hrany, ale opět obvykle jen jednoho kanálu.

V současné době jsou logické analyzátory vyráběny ve 3 typech provedení:

- Modifikovaný standardní počítač se standardním operačním systémem s přidaným vestavěným měřícím HW.
- Samostatný logický analyzátor se specifickým designem, ovládacími prvky a zobrazovacím zařízením. Obvykle má vlastní dedikovaný řídící firmware a nemá možnost rozšiřování a úprav.
- Externí hardware realizující obvykle sběr dat a interface ke standardnímu PC (obvykle USB, LPT nebo Ethernet). Software v PC slouží ke konfiguraci logického analyzátoru, vyčítání a interpretaci naměřených dat. Toto zařízení předpokládá, že uživatel již má k dispozici počítač, což je typický případ a náklady na logický analyzátor jsou pak mnohem menší, protože neobsahuje HW ke zobrazování.

Moderní logické analyzátory jsou schopné poskytovat další nadstandardní možnosti zobrazení naměřených dat. Například jsou schopné po přiřazení pinů k signálům standardní sběrnice dekódovat a zobrazovat stavy sběrnice. Popřípadě dekódovat a zobrazovat datový stav na sběrnici jako mnemotechnický název instrukce v assembleru.

Úskalím logických analyzátorů je mechanické řešení sond. Sondy slouží pro připojení logického analyzátoru k části digitálního systému kterou chceme měřit. Obvykle je nejlepší, když na testovací body myslíme už při návrhu prototypu. Moderní sondy jsou však mechanicky řešeny jako miniaturní pinzety a obvykle je možné je připojit přímo na pouzdra obvodů DIL nebo na kolíkové konektory. Pokud ovšem osazujeme moderní SMD (Surface-Mount Devices) součástky obzvláště BGA (Ball Grid Array) obvody, pak jsou měřící pady na prototypu nutností. Při opomenutí testovacích padů v návrhu plošného spoje prototypu se obvykle dostaneme při měření do velkých problémů.

1.1 Možnosti dnešních špičkových logických analyzátorů

Předními výrobci logických analyzátorů jsou firmy Agilent a Tektronix. Jako referenční logický analyzátor jsem vybral logický analyzátor 16801A 34-channel od firmy Agilent. Jedná se o logický analyzátor, který je modifikovaným počítačem. Parametry tohoto analyzátoru jsou:

- Vzorkování frekvencí 1GHz (1ns) pro 17 signálů.
- Vzorkování frekvencí 500MHz (2ns) pro 34 signálů.
- Hloubka paměti až 32M.
- Cena US$ 10000 (~200 000 CZK).

1.2 Možnosti levných logických analyzátorů

V katalogu Firmy GME jsem nalezl logický analyzátor PROG.SIGMA. Parametry tohoto logického analyzátoru jsou:

- Vzorkování frekvencí 200Mhz (5ns) pro 4 signály.
- Vzorkování frekvencí 100MHz (10ns) pro 8 signálů.
- Vzorkování frekvencí 50MHz (20ns) pro 16 signálů.
- Hloubka paměti až 32M.
- Cena 9 500 CZK.

1.3 Vyvíjený analyzátor

Logický analyzátor vyvíjený v rámci této práce má za cíl dosáhnout lepšího poměru ceny ku výkonu než výše uvedené logické analyzátory. Výkonem se rozumí rychlost měření * šířka měřícího portu. Součástí vyvíjeného logického analyzátoru bude rovněž aplikace pro jednoduchou a intuitivní obsluhu.

2. Specifikace

Cílem práce je navrhnout a realizovat logický analyzátor s minimální vzorkovací frekvencí 50MHz a minimální šířkou měřícího portu 8 bitů. Dalším cílem práce bude i případné zlepšení těchto parametrů v rámci možností použitého HW.

Přínosem této práce bude produkt, které bude optimalizovat poměr ceny ku výkonu a zaplní tak mezeru na trhu, který dnes ovládají velké firmy jako Agilent a Tektronix, které mají komplexní a špičkové produkty, avšak tyto se pohybují v cenové relaci od 200tisíc CZK výše. Výkonem zde rozumíme především počet měřicích kanálů a maximální vzorkovací frekvenci. Avšak zřetel je brán i na další parametry jako je snadná ovladatelnost, přehledné zobrazení naměřených výsledků a široké možnosti nastavení.

Logický analyzátor bude obsahovat obvyklé funkce. Jmenovitě tedy:

- Nastavení spouštěcí podmínky.
- Nastavení pozice spouštěcí události na začátek, střed a konec zobrazovaného úseku měřeného signálu.
- Podpora nastavení nižší vzorkovací frekvence.
- Možnost nastavení menší paměti pro ukládání vzorků (vhodné zejména pro nižší frekvence, aby měření netrvalo příliš dlouho).
- Uložení a načtení naměřených dat
- Přehledné grafické zobrazení s možností zmenšení a zvětšení měřítka časové osy

2.1 Upřesňující požadavky na formu realizace systému

Zařízení bude řešeno modulárně pomocí 2 standardních částí a pouze jedné specifické části. Standardní části jsou:

- Klasický počítač PC s rozhraním USB 2.0 a operačním systémem Windows XP
- Vývojový kit Spartan 3E Starter firmy Xilinx

Nově navrhnutá část pak bude představovat

- IO rozšiřující deska s rozhraním USB 2.0.

Tuto IO desku může pak uživatel používat i nezávisle k vývoji a testovaní svých aplikací používajících rychlé rozhraní USB 2.0.

Uživatel bude mít možnost si dokoupit pouze části systému které nevlastní. V ideálním případě pouze velmi levnou IO desku s interfacem USB 2.0.

2.1.1 Softwarová část systému

Řízení logického analyzátoru a zobrazování dat bude realizováno na straně standardního PC s operačním systémem Windows XP a rozhraním USB 2.0.

Aplikace bude poskytovat přehledný graficky uživatelský interface, který bude umožňovat veškerou konfiguraci zařízení, vyčtení dat a jejich přehledné zobrazení včetně referenční časové osy.

Součástí aplikace bude rovněž možnost otestovat správnou funkci zařízení

2.1.2 Hardwarová část systému

Hardwarové řešení předpokládá minimalizaci ceny. Proto se bude jednat o takzvaný „system on chip“, kde řídící mikroprocesor a maximální množství periferií je implementováno v jednom čipu – typicky FPGA.

Zadání předpokládá použití návrhového prostředí EDK a mikroprocesoru MicroBlaze v chipu FPGA Spartan 3E.

2.2 Revize upřesňujících požadavků na realizaci systému

Původní záměr realizovat řešení v čipu Spartan 3E s sebou neslo problémy při tvorbě DDR řadiče. Především tato platforma neobsahuje obvody pro programovatelné zpoždění IO bloků (takzvaný Input Delay Element). Tyto obvody jsou nezbytné pro synchronizaci dat přicházejících z DDR pamětí s lokálními hodinami FPGA obvodu. Zároveň platforma Spartan 3E neposkytovala dost prostoru pro firmware s ladícími knihovnami. Po domluvě s vedoucím práce bylo proto pro realizaci prototypu zvolena silnější platforma postavená na čipu Virtex 4E, a byl proto změněn development board ze Spartan 3E Starter Kit na development board ML402.

V dalším návrhu bude brán zřetel na budoucí možnost naportování funkčního a odladěného řešení zpět na platformu Spartan 3E. To bude zajištěno především přirozenou abstrakcí jazyka VHDL od použitého hardware a modulárním návrhem řešení, díky kterému bude možno snadno vyměnit moduly, které obsahují na hardware závislé konstrukce.

3. Analýza

V analýze by bylo vhodné srovnat návrh ostatních logických analyzátorů. Toto však není možné, protože výrobci neposkytují schémata ani kódy pro svá řešení, naopak tato jsou chráněna jako firemní tajemství. Možností by bylo zakoupit zařízení a provést reverzní inženýrství. To by však bylo patrně neetické a mimo jiné by škola neposkytla patřičné prostředky k zakoupení referenčních logických analyzátorů. Navíc moderní obvody jsou často chráněny proti přečtení konfigurací, nebo jsou natolik komplikované, že vyčtená data by bylo velmi obtížné a časově náročné interpretovat a analyzovat.

V této analýze vycházím z upřesňující specifikace, kdy jako platforma byl zvolen vývojový kit ML402. Postupně probírám jednotlivé požadavky na řešení a porovnávám, zda kit ML402 vyhovuje. Takto postupuji především proto, že tento vývojový kit byl k dispozici. Pokud bych měl větší možnosti se rozhodnout, pak by bylo patrně vhodnější, na základě jednotlivých požadavků, jako výsledek této analýzy vybrat i vývojovou platformu. Naopak pro definovanou platformu, je možné napsat kvalitnější analýzu všech potřebných aspektů systému. Na základě této analýzy však bude čtenář schopen snadno vybrat i další platformy, na kterých by navrhované řešení mělo jít uskutečnit.

Jelikož jsem neměl možnost porovnávat jiná řešení logických analyzátorů, navrhl jsem rovnou své řešení logického analyzátoru. Základní návrh má formu blokového schématu a je ukázán na obr. 1. Tento návrh vychází především z obecných požadavků pro funkce logických analyzátorů, ale částečně také z prostředků vývojového kitu ML402. Na ML402 je klíčový obvod Virtex-4E.

V tomto obvodu bude realizován mikroprocesor MicroBlaze, který bude provádět veškeré úlohy, které nejsou časově kritické včetně:

- inicializace a konfigurace periferních zařízení
- enumerace na USB sběrnici
- zpracovávání USB paketů zasílaných z řídící aplikace

V obvodu Virtex-4E je dále realizován DDR řadič zajištující komunikaci s DDR pamětí, která bude uchovávat navzorkovaná měřená data. Na vývojovém kitu ML402 se nachází dvě DDR paměti připojené k čipu Virtex-4E. Každá z nich má velikost 256Mbit. Můžeme tedy předpokládat, že vývojový kit nám poskytuje dostatek paměti pro ukládání dat bez použití jakékoliv přídavné paměti.

Dále bude v obvodu Virtex-4E periferie pro komunikaci s rozšiřujícím IO modulem realizujícím USB 2.0 interface. Tato periferie bude zajišťovat možnost procesoru MicroBlaze komunikovat s USB rozhraním a provádět tak enumeraci, posílání a přijímání dat.

Nejdůležitějším modulem v obvodu Virtex-4E je periferie realizující jádro logického analyzátoru. Jádro je rozděleno na dvě části schopné pracovat v jiných časových doménách. První část jádra logického analyzátoru běží na hlavní frekvenci FPGA a stará se o výběr dat z FIFO a jejich ukládání do DDR paměti. Zároveň zajišťuje bezpečné předání konfiguračních dat do druhé části jádra logického řadiče. Druhá část jádra logického řadiče vzorkuje data ze vstupního portu a ukládá je do FIFO vyrovnávacího bufferu. Tato druhá část bude schopna běžet na frekvenci nezávislé na zbytku obvodu a bude tak možno nastavit různé vzorkovací frekvence. Navzorkovaná data jsou testována na spouštěcí podmínku. Dojde-li ke spouštěcí podmínce, vynuluje se čítač a uloží přednastavený počet dat. Přednastavený počet dat bude závislý na nastavené hloubce paměti a na nastavené relativní pozici spouštěcí podmínky. Druhá část logického řadiče bude přímo propojena s měřícím portem.

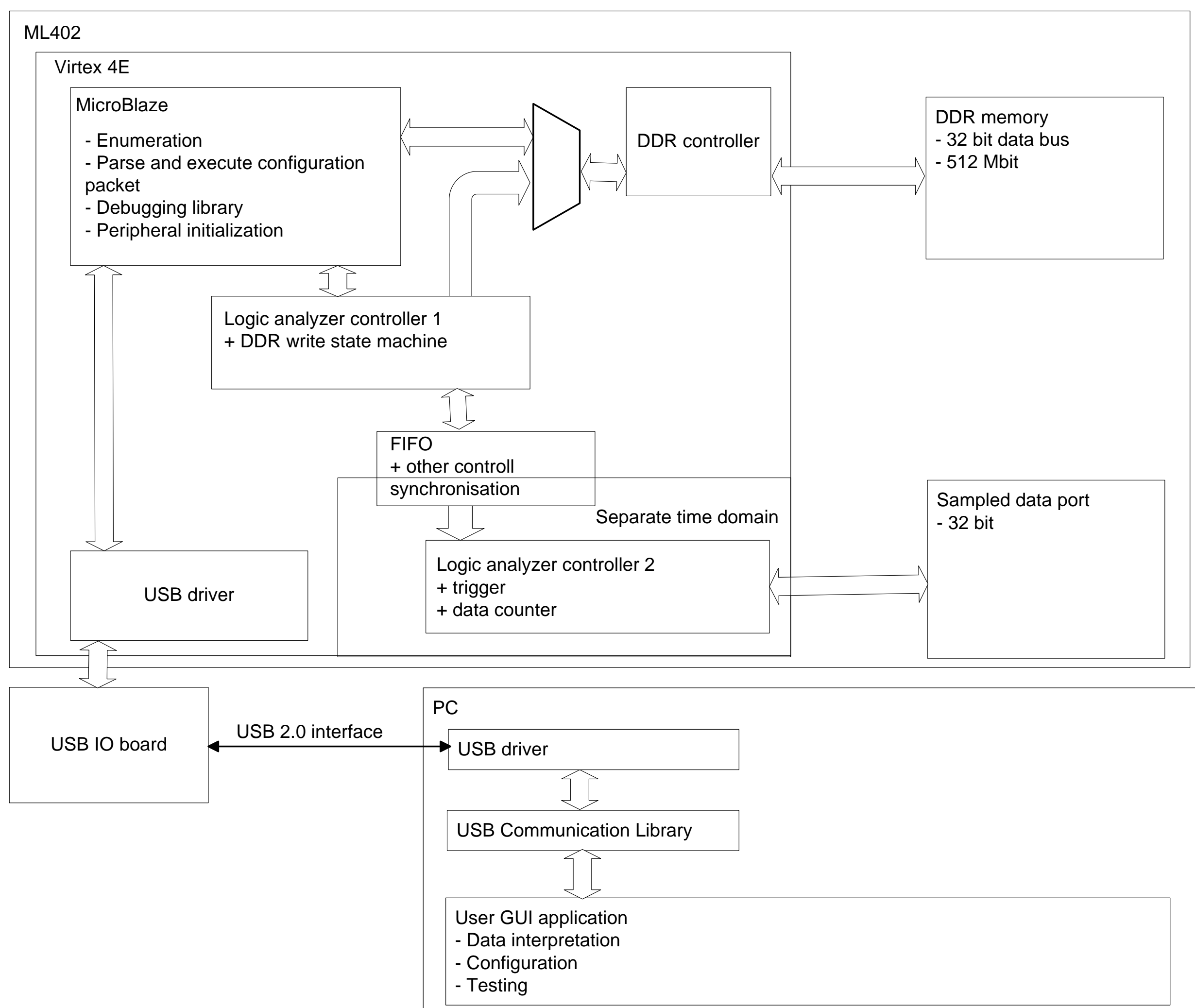
Rozšiřující IO deska realizující USB interface bude realizována levným čipem podporujícím přímé připojením k USB 2.0. Důraz bude kladen na rychlost a cenu řešení.

Na straně PC je pak klíčovým prvkem USB driver. Použijeme pokud možno generický USB driver.

Jako nadstavba bude napsána knihovna zajišťující potřebnou komunikaci mezi USB driverem a uživatelskou aplikací.

Uživatelská aplikace bude sloužit k celkové konfiguraci a ovládání logického analyzátoru, jakož i k interpretaci měřených dat a testování korektní funkce. Důraz bude kladen na přehlednost a na snadné a intuitivní ovládání.

V dalším textu se budeme podrobněji věnovat požadavkům na jednotlivé výše zmíněné komponenty.



obr. 1 – Náčrt systému logického analyzátoru.

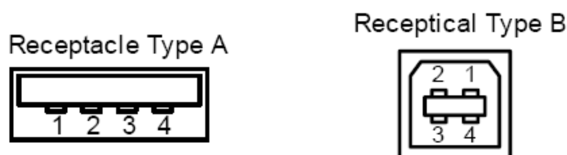
3.1 USB IO deska

Nyní se zaměříme na výběr vhodného čipu pro zajištění rozhraní mezi USB a mikroprocesorem. Zde je důležité rozumět jak USB funguje a jaké jsou jeho možnosti. Proto se v následující kapitole budeme letmo věnovat relevantním částem normy USB 2.0 [1]. Některé pasáže jsou pak převzaty z mé bakalářské práce [2] a zjednodušené normy v publikaci USB in a Nutshell [3].

3.1.1 USB Konektor

USB norma 2.0 [1] definuje přesné typy konektorů. Pro downstream porty je to konektor typu A (viz obr. 2 vlevo). Downstream port je určen pro host řadiče a huby.

Pro koncová zařízení slouží konektor typu B (viz obr. 2 vpravo). Je to takzvaný upstream port.



obr. 2- USB konektory

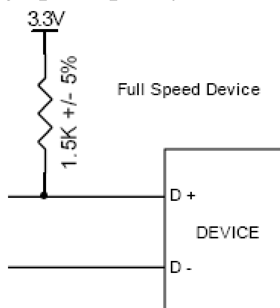
Existují i menší varianty mini-B pro mobilní zařízení.

Pro náš modul USB IO desky jsem se rozhodl použít standardní konektor typu B.

3.1.2 USB elektrická norma

USB kóduje data kódem NRZI pomocí rozdílových vodičů D- a D+. Logická 1 je kódovaná úrovní D+ větší než 2,8V a úrovní D- menším než 0,3V. Logická 0 je kódována úrovní D+ menší než 0,3V a D- větší než 2,8V. Přijímače rozeznávají logickou 1, když D+ je alespoň +200mV větší než D- a logickou 0, když D- je alespoň o 200mV větší než D+. Existuje ale i nerozdílový signál, takzvaná „single ended zero". Kóduje se úrovní D+ i D- menší než 0,3V. Je používaný například pro bus reset (délka musí být alespoň 10ms).

Zajímavostí je, že na „low speed" zařízení je Logická hodnota získaná ze signálů D+ a D- invertována, kdežto na „high" a „full speed" zařízeních invertována není. Proto pro obecné vyjádření stavu používáme označení „J" a „K". „J" je na „low speed" zařízeních diferenciální 0 a na „full" a „high speed" zařízeních diferenciální 1. Stav „K" je pak opačný.



obr. 3 – Deklarace rychlosti USB

Rychlost zařízení je určena připojením 1,5KΩ odporu (viz.obr. 3) k vodiči D+ nebo D-. Host řadič podle proudu přes odpor detekuje předně, že je zařízení vůbec připojeno. Podle toho, zda je odpor na vodiči D+ nebo D- detekuje, zda je zařízení „low" nebo „full/high speed".

USB zařízení mohou být napájeny přímo z host řadiče a nemusí tak mít vůbec vlastní zdroj. Pravidla tohoto napájení jsou však velmi přísná. Předně zařízení před nakonfigurováním nesmí odebírat více než 100mA. Po úspěšném nakonfigurování mohou požádat maximálně o 500mA. Hostující počítač je ale může odmítnout nakonfigurovat právě kvůli vysokým požadavkům na odběr. Napětí z USB řadiče je

definováno mezi 4,4V a 5,25V, pokud tedy si zařízení žádá přesné napájení, je složité jej napájet z USB host řadiče.

Hlavní obtíží je „suspend“ mód. V „suspend“ módu totiž zařízení nesmí odebírat více než 0,5mA. Každé zařízení je navíc povinno implementovat „suspend“ mód. Toto je velmi složité splnit. Například pouze standardní 3.3V stabilizátor spotřebovával řídící proud 10mA, což je 20x víc, než dovoluje norma.

V našem řešení se tedy velmi vyplatí použít napájení z vývojového kitu ML402. Vyhneme se tak řadě problémů a případnému zničení host řadiče v PC přílišným odběrem.

3.1.3 USB protokoly

USB narozdíl od RS-232 je tvořeno několika vrstvami protokolů. Na USB sběrnici se data přenášejí pomocí transakcí. Pro implementaci USB rozhraní je nutné správně zvolit typ přenosu, který je nejvíce vhodný pro zařízení.

3.1.3.1 USB Transakce

Group	PID Value	Packet Identifier
Token	0001	OUT Token
	1001	IN Token
	0101	SOF Token
	1101	SETUP Token
Data	0011	DATA0
	1011	DATA1
	0111	DATA2
	1111	MDATA
Handshake	0010	ACK Handshake
	1010	NAK Handshake
	1110	STALL Handshake
	0110	NYET (No Response Yet)
Special	1100	PREamble
	1100	ERR
	1000	Split
	0100	Ping

obr. 4 - PID paket list

Transakce se skládá ze tří typů packetů:

1. “token packet”, který uvozuje typ transakce
2. “data packet”, který přenáší aplikační data a je nepovinný
3. “status packet” (“handshake packet”), který potvrzuje výsledek transakce

Každý z těchto tří paketů má pole “packet ID” (PID), které určuje typ paketu a jaká (a jestli vůbec) data obsahuje. Přehled “packet ID” jsem shrnul na obr. 4.

3.1.3.1.1 ad 1) Token Packet

Sync	PID	ADDR	ENDP	CRC5	EOP

obr. 5 - Token paket struktura

„Token packet“ uvozuje typ transakce. Na obr. 5 je struktura „Token packet“. Přikládám stručný popis jednotlivých částí:

- Sync: protože USB je asynchronní protokol, je potřeba mít na obou stranách generátory hodin a je potřeba je synchronizovat. Pole Sync slouží právě k tomuto účelu a je součástí začátku každého paketu.
- PID je akronym pro „Packet ID“ a určuje typ paketu. Je proto nedílnou součástí každého paketu. „Token packet“ se týkají tyto ID:
  - IN: je důležité vědět, že USB je host centrická sběrnice. Proto zařízení nemůže vysílat data, kdy se mu zlíbí, ale pouze když je požádáno host řadičem. PID IN informuje USB zařízení, že host si od něj žádá data.
  - OUT: informuje USB zařízení, že mu host chce poslat data
  - Setup: informuje USB zařízení, že se budou posílat řídící a konfigurační informace
- ADDR: Protože na USB lze připojit až 127 zařízení, pole ADDR vybírá, se kterým zařízením se bude komunikovat. Adresa 0 slouží pro přidělení adresy nově připojeným zařízením.
- ENDP: USB rozhraní se skládá z několika bufferů. Těmto bufferům se říká „end points“ (EP). Každé zařízení musí implementovat minimálně EP0, které slouží pro konfiguraci. Pole ENDP vybírá, se kterým EP se bude pracovat. Endpointů je maximálně 16.
- CRC(x): Je kontrolní (x bitový) součet. Pomocí něj lze detekovat, zda paket dorazil v pořádku.
- EOP: signalizuje konec paketu a je opět součástí každého typu paketu. Kóduje se jako “single ended 0” (úroveň napětí menší než 0,3V na D+ i D-) po dobu 2 bitů a následuje úroveň ‘J‘.

3.1.3.1.2 ad 2) Data Packet

Sync	PID	Data	CRC16	EOP

obr. 6 - Data paket struktura

„Data packet“ přenáší datovou část transakce a není povinný u všech typů transakce. Stručný popis nových částí:

- PID: pro datové pakety jsou vymezeny ID:
  - Data0
  - Data1

  PID Data0 a Data1 se pravidelně střídají, aby bylo možno detekovat ztrátu celého jednoho datového paketu. Pokud např. po sobě přijdou Data0 a Data0 paket Data1 se ztratil.
- Data: toto pole obsahuje konečně užitečná data.

3.1.3.1.3 ad 3) Status packet (handshake packet)

Sync	PID	EOP

obr. 7 - Status paketu struktura

„Status packet“ potvrzuje výsledek transakce.
PID pro něj vyhrazené jsou:

  - ACK: Potvrzení, že transakce byla v pořádku provedena
  - NAK: Signalizace, že zařízení momentálně nemůže data přijmou/odeslat (např. data k odeslání nejsou ještě připravena)
  - STALL: Obvykle chybový stav, který si žádá vyřešení hostem.

3.1.3.1.4 Pojem „Endpoint”

Endpoint je v podstatě pouze pojem označující buffer v koncovém zařízení. K bufferu se přistupuje pomocí čísla. Každé USB zařízení musí mít buffer 0 (EP0), který slouží pro nakonfigurování. Celkově každé zařízení může mít až 16 bufferů (endpoints). Každý použitý endpoint je nastaven buď jako buffer pro příjem dat z host řadiče (EPx OUT), nebo buffer pro posílání dat do host řadiče (EPx IN).

3.1.3.1.5 Pojem „Pipe“

Je logická cesta pro přenos dat do endpointu. Každá pipe má nastavené potřebné parametry jako velikost koncového buffer, směr toku a typ přenosu.

3.1.3.2 Typy přenosu

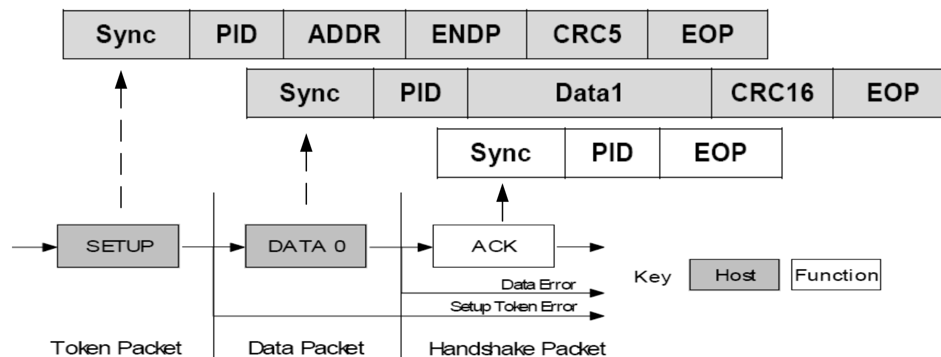
Existují čtyři základní druhy přenosu. Podle funkce zařízení se zvolí nejlépe vyhovující typ přenosu. Každý endpoint může mít nastaven jiný typ přenosu. V následujících podkapitolách uvádím přehled typů přenosu.

3.1.3.2.1 Control transfer

Je používaný k nakonfigurování zařízení. Typicky je určen pro datový tok, který není konstantní.

Control transfer má tři fáze:

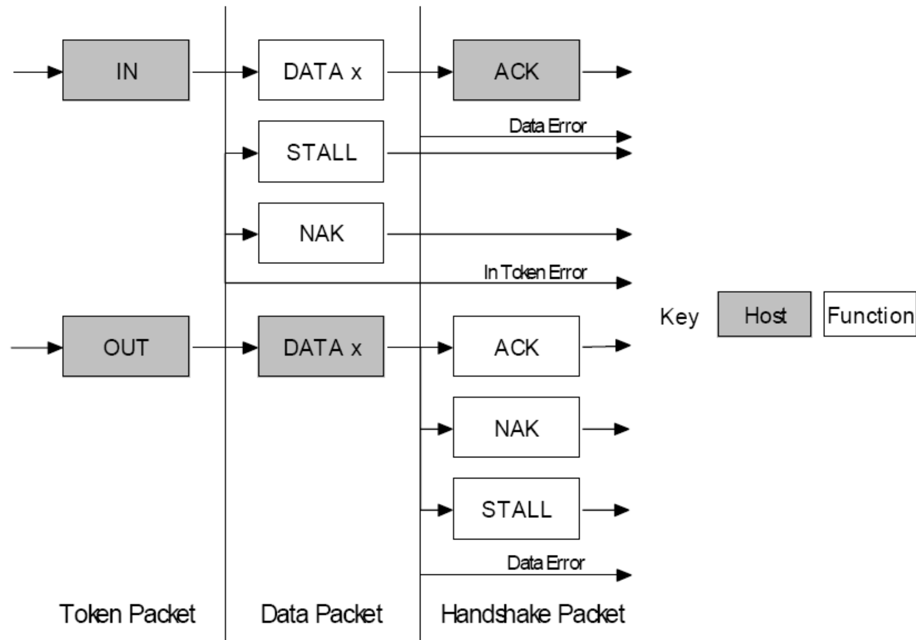
3.1.3.2.1.1 Fáze Setup



obr. 8 – Setup fáze struktura

Z obr. 8 je patrno, že fáze Setup dodržuje klasickou strukturu transakce, jak je popsána v kapitole 3.1.3.1. Míním tím posloupnost Token packet, Data Packet a Handshake Packet. Struktura jednotlivých paketů je podrobně popsána taktéž v kapitole 3.1.3.1 a pro vizuální představu je v horní části obr. 8 znázorněna. Ve spodní části obrázku pak je samotná struktura fáze setup. Text v obdélnících (SETUP, DATA0 a ACK) znamená PID jednotlivých paketů . Barva znázorňuje, kdo pakety posílá.

3.1.3.2.1.2 Fáze Data



obr. 9 – Data fáze struktura

Datová fáze se skládá z jednoho nebo více IN/OUT transakcí. Velikost dat je určena ve fázi Setup.

Průběh Datové fáze závisí na směru toku dat.

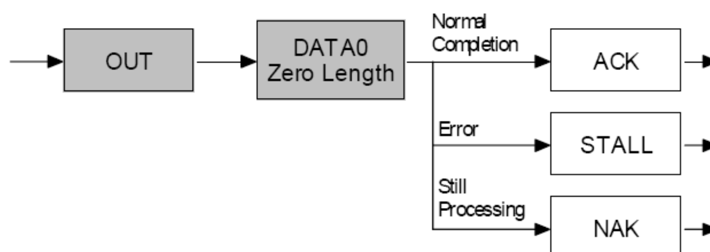
Pro posílání dat z USB zařízení do host řadiče host pošle IN token signalizující, že je připraven přijmout data. Pokud nesouhlasí CRC, zařízení nepošle žádnou odpověď. V opačném případě pošle data a host odpoví ACK, pokud je datový paket v pořádku. Pokud zařízení nemá data k dispozici, odpoví paketem NAK. Pokud je zařízení v chybovém stavu, odpoví paketem STALL.

Pro posílaní dat z host řadiče do USB zařízení host pošle paket OUT následovaný datovým paketem. Pokud zařízení uložilo data do EP, pak vrátí ACK. Pokud EP nebyl prázdný a zařízení data nemohlo uložit, pošle NAK. Pokud je zařízení v chybovém stavu, pošle STALL.

3.1.3.2.1.3 Fáze Status

Slouží k celkovému potvrzení control transferu. Potvrzení vždy posílá zařízení. Struktura potvrzení je však rozdělena podle směru toku dat v datové fázi:

- tok dat v datové fázi byl IN (zařízení→host)

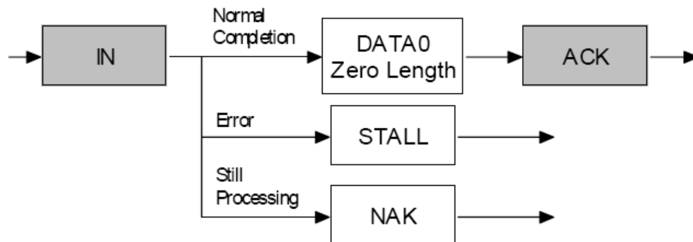


obr. 10 – Status fáze struktura, když Data fáze byla IN (function→host)

Host vyšle OUT token následovaný Datovým paketem s prázdnými daty. Zařízení odpoví paketem ACK, pokud celý přenos proběhl v pořádku. Pokud je zařízení

zaneprázdněno, odpoví paketem NAK, aby host opakoval fázi status později. Paketem STALL odpoví zařízení při chybě.

- tok dat v datové fázi byl OUT (host→zařízení)

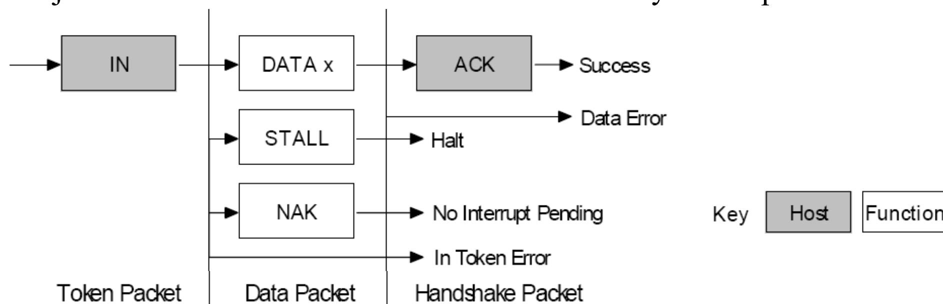


obr. 11 - Status fáze struktura když Data fáze byla OUT (host→function)

Host vyšle IN paket. Zařízení odpoví paketem DATA0 s nulovou délkou, pokud celý přenos byl v pořádku. Pokud zařízení ještě nezpracovalo data, odpoví paketem NAK, který žádá hosta, aby zopakoval fázi status později. Pokud je zařízení v chybovém stavu, odpoví paketem STALL.

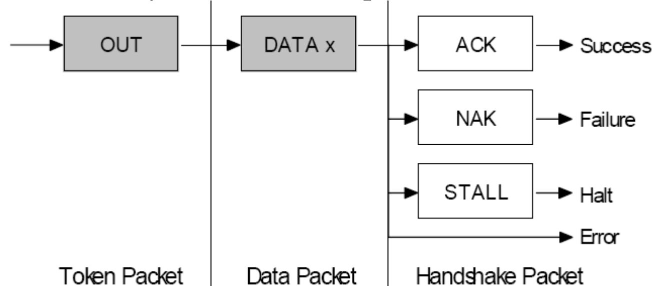
3.1.3.2.2 Interrupt transfer

Interrupt transfer navozuje představu, že zařízení přeruší host řadič v případě potřeby komunikace. To ale není možné, protože USB je navržena jako host centrická sběrnice, a proto host musí začít transakci. Ve skutečnosti tedy zařízení v tomto režimu sdělí hostovi, jak často si přeje, aby se host dotazoval, zda zařízení chce komunikovat. Tento čas je 1 až 256 ms a host v nastaveném intervalu vysílá IN paket.



obr. 12 - Interrupt transfer struktura, směr function→host

Na obr. 12 je znázorněn průběh komunikace. Když zařízení obdrží IN paket a má data pro hosta, pošle je v paketu DATAx; pokud data nejsou připravena, pošle paket NAK. Pokud je zařízení v chybovém stavu, pošle STALL. Host odpoví ACK po obdržení bezchybného DATAx paketu.

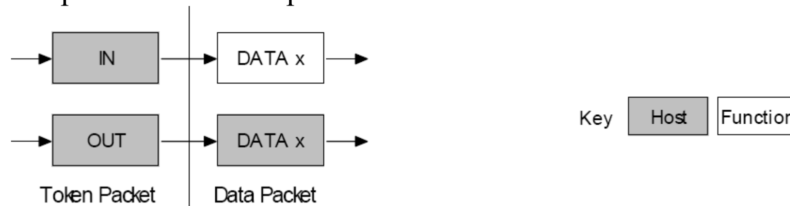


obr. 13 - Interrupt transfer struktura, směr host→function

Na obr. 13 je posílání dat směrem od hosta k zařízení. Host pošle OUT paket následovaný datovým paketem. Zařízení odpoví ACK, pokud data přijalo správně. Pokud byl buffer cílového EP již plný, pošle NAK paket. Pokud je zařízení v chybovém stavu, odpoví STALL paketem.

3.1.3.2.3 Isochronous transfer

Isochronous transfer je přenos vhodný pro souvislý periodický přenos dat (typicky proud obrazových nebo zvukových dat). Isochronní přenos garantuje šířku přenosového pásma a minimální prodlevu. Poskytuje jednocestný kanál a CRC detekci chyb bez možnosti požádání o znovuposlání dat.



obr. 14 - Isochronous transfer struktura.

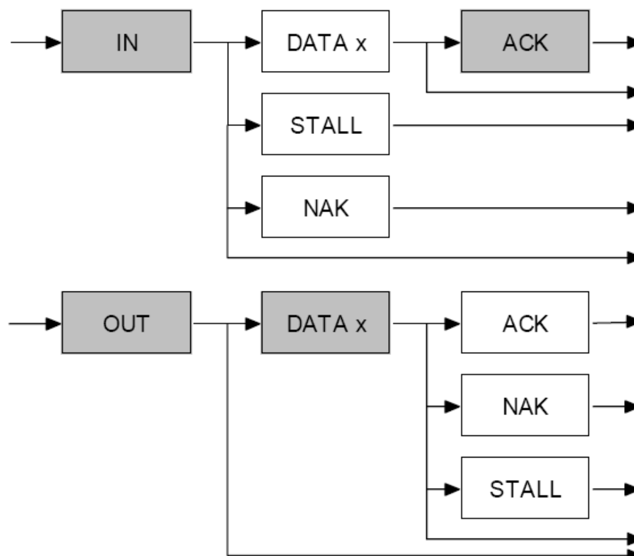
Isochronní přenos nemá potvrzovací část transakce (handshake paket), jak je patrno na obr. 14.

3.1.3.2.4 Bulk transfer

Bulk transfer je vhodný k velkým datovým přenosům, které se objevují neperiodicky. Například tisk na USB tiskárnu. Bulk transfer poskytuje detekci chyb pomocí CRC a znovuposlání chybných dat.

Bulk transfer má přiděleno pásmo, které zbývá na USB po odečtení pásem ostatních typů transakcí. Pokud tedy isochronní a interrupt transfer zabírají velké pásmo, je bulk transfer pomalý a je vhodný pouze pro časově nekritické operace.

Bulk transfer je podporován pouze full a high speed zařízeními.

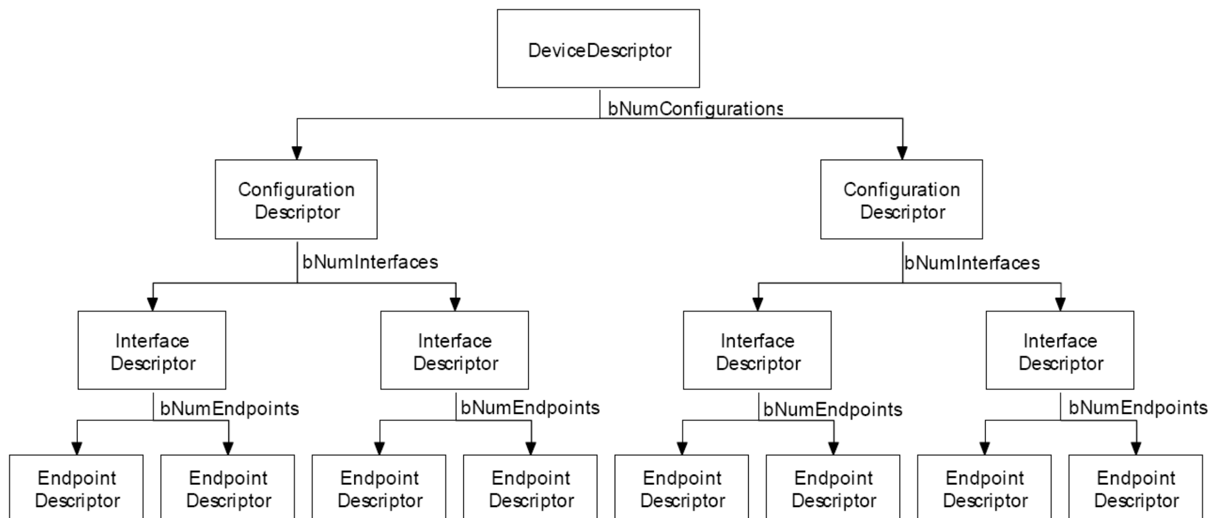


obr. 15 - Bulk transfer struktura.

Bulk přenos je strukturou podobný interrupt přenosu s tím rozdílem, že host neposílá IN tokeny periodicky, ale pouze při potřebě získání dat ze zařízení.

3.1.4 USB deskriptory

Při konfiguraci (obvykle po přidělení adresy k zařízení) si host vyžádá tabulky deskriptorů. Tyto tabulky mají hierarchickou strukturu (viz. obr. 16) a slouží k popisu zařízení.



obr. 16 - Hierarchie deskriptorů.

Všechny deskriptory mají společnou strukturu záhlaví.

- specifikace délky deskriptoru v bytech. (velikost 1B)
- specifikace typu deskriptoru (velikost 1B)

3.1.4.1 Device deskriptor

Deskriptor zařízení definuje globální parametry zařízení, jmenovitě:

- podporovanou verzi USB
- třídu zařízení
- podtřídu zařízení
- typ protokolu
- velikost EP0 (8/16/32/64)
- ID výrobce
- ID produktu
- verzi produktu
- ukazatel na string deskriptory popisující zařízení (výrobce , produkt, sériové číslo)
- počet konfiguračních deskriptorů

3.1.4.2 Konfigurační deskriptor

Konfigurační deskriptor předně deklaruje, zda je zařízení napájeno externě nebo z USB sběrnice. Je možné, aby zařízení mělo obě možnosti a může mít tedy více konfiguračních deskriptorů.

Přehled parametrů v konfiguračním deskriptoru:

- počet rozhraní (interfaces)
- ID aktuálního konfiguračního deskriptoru
- ukazatel na string deskriptor popisující daný konfigurační deskriptor
- Jakým způsobem je zařízení napájeno (USB/externě)
- Maximální spotřeba

3.1.4.3 Interface deskriptor

Popisuje jednu funkci zařízení. Sdružuje konfiguraci několika EP, které tuto funkci realizují. Pokud se jedná o multifunkční zařízení (například tiskárna se skenerem), může mít více deskriptorů rozhraní.

Přehled parametrů deskriptoru rozhraní:

- ID deskriptoru
- Počet EP, které využívá
- popis třídy (HID, Mass-storage ...)
- popis podtřídy
- typ protokolu (např. generický protokol pro myš, klávesnici ...)
- ukazatel na string deskriptor popisující daný interface

3.1.4.4 Endpoint deskriptor

Deskriptor popisující jednotlivé EP kromě EP0. EP0 je nakonfigurována standardně ještě předtím, než jsou vůbec posílány deskriptory zařízení. Na základě deskriptoru EP host zjistí potřebnou šířku pásna a typ přenosu.

EP deskriptor poskytuje tedy parametry:

- číslo EP
- typ přenosu (Bulk, Control, Isochronous, Interrupt)
- velikost EP v bytech
- interval požadování dat (pouze pro Interrupt a Isochronous přenos)

3.1.5 USB norma shrnutí

Po prostudování normy jsem se rozhodl najít obvod, který již poskytuje abstrakci od nejnižší elektrické vrstvy. Je schopen samostatně provádět bit-stuffing a kódování NRZI a ovládat diferenciální signály. Zároveň očekávám, že obvod bude poskytovat vlastní EP buffery, jelikož při rychlosti USB 480Mbps by mohl nastat problém s napsáním dostatečně rychlého automatu pro dodávání dat pro přenos. A při použití dat generovaných v mikroprocesoru bych patrně stejně musel realizovat buffery v FPGA a to by zabralo zbytečně prostředky FPGA obvodu.

Zároveň jsem se rozhodl použít obvod, který podporuje Bulk transfer, protože zařízení potřebuje přenášet data nárazově a v maximálním možném přenosovém pásmu (typicky přenos naměřených dat). Rozhodl jsme se použít pouze externí napájení, a tedy pouze jeden konfigurační deskriptor.

Ač zařízení poskytuje více funkcí, rozhodl jsme se vše zahrnout do jednoho komunikačního interface.

3.1.6 Výběr čipu pro IO desku

Největší přehled USB rozhraní jsem našel na stránkách [4] pana Jana Axelsona autora knihy USB Complete [5], která byla rovněž zdrojem cenných informací. Jako nejvhodnější jsem z uvedeného přehledu zvolil čip ISP1583. Z dokumentace obvodu ISP1583 [6] vyplývá, že tento čip splňuje přesně všechny požadavky stanovené výše v kapitole 3.1.5 a přitom neobsahuje samostatný procesor jako většina ostatních obvodů v této kategorii.

Předkládám seznam vlastností, pro které jsem se rozhodl použít čip ISP1583:

- Plně podporuje normu USB 2.0.
- Podporuje High Speed transfer.
- Má paralelní rozhraní vhodné pro přímé připojení k mikroprocesoru
- Napájecí napětí je 3.3V, tedy stejné jako napájecí napětí FPGA obvodu

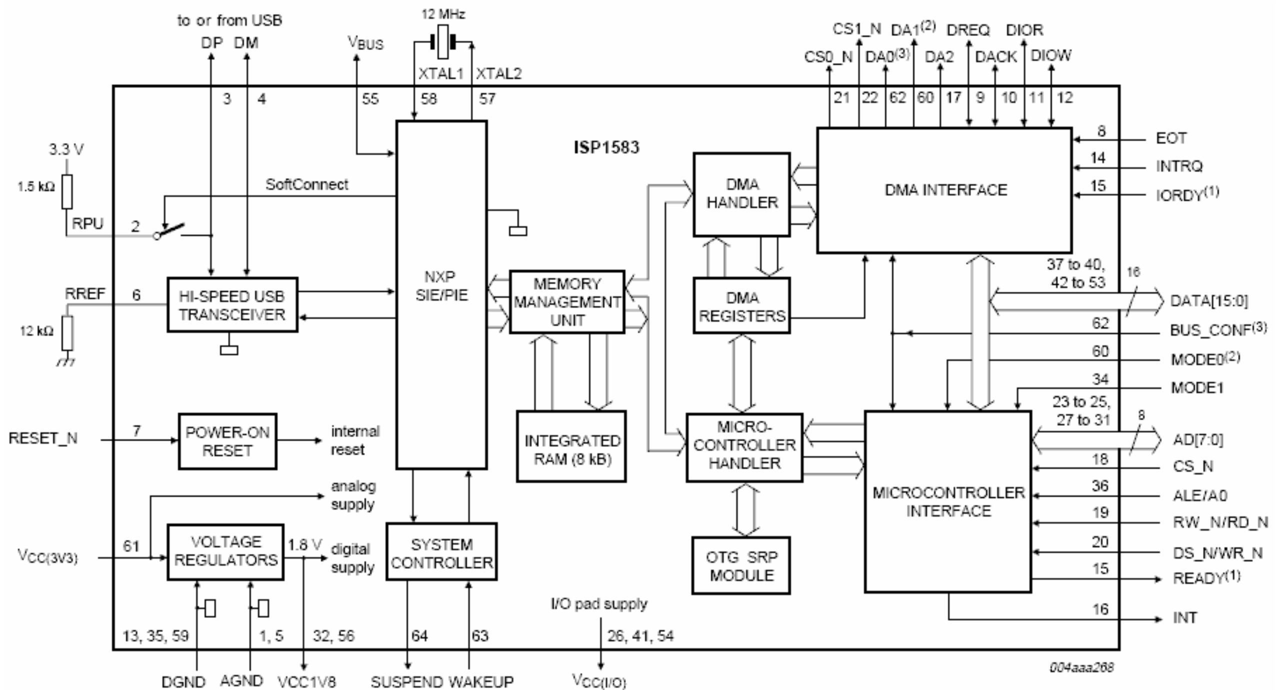
- Datové piny podporují od 1.65V do 3.6V, není proto potřeba konverze napěťových úrovní a lze přímo připojit k FPGA datovým pinům konfigurovaným jako LVCMOS33
- Podporuje softwarově konfigurovatelné odpojení od USB.
- Má až 7 IN a 7 OUT end pointů + konfigurační endpoint 0.
- Má 8KB paměti, kterou lze libovolně rozdělitelné mezi použité end pointy.
- Nízká spotřeba (při plném provozu maximálně 60 mA, při suspend režimu maximálně 160µA).
- Nízká cena ($3 Avnet, $5.4 DigiKey odběr po kusech (Prosinec 2009)).
- Jediné periferie, které jsou potřeba, jsou přesný 12MHz krystal, blokovací kondenzátory a samozřejmě USB konektor.
- Čip podporuje všechny 4 typy přenosů (Control, Bulk, Isochronous i Interrupt)

Podotýkám, že výběr čipu jsem prováděl více než před rokem, a proto mohou existovat některé nové obvody, které mají lepší požadované parametry.

3.1.7 Podrobná analýza čipu ISP1583BS

Tato analýza se opírá především o katalogový list obvodu ISP1583BS [6] a o dokumenty Application note: Firmware Programming Guide [7] a Application note: Frequently Asked Question [8].

3.1.7.1 Analýza hardwarového rozhraní čipu ISP1583



obr. 17 –ISP1583BS schéma

Ze schématu na obr. 17 vpravo je patrno, že obvod ISP1583 poskytuje dvě rozhraní. Předně poskytuje rozhraní k procesoru a dále poskytuje rozhraní k DMA řadiči. My se dále budeme zajímat pouze rozhraním k procesoru. Rozhraní k DMA řadiči nebudeme používat, protože by to znamenalo nutnost návrhu DMA řadiče, který není součástí zamýšleného řešení.

Jak je vidět z obr. 17 rozhraní k procesoru poskytuje následující piny:

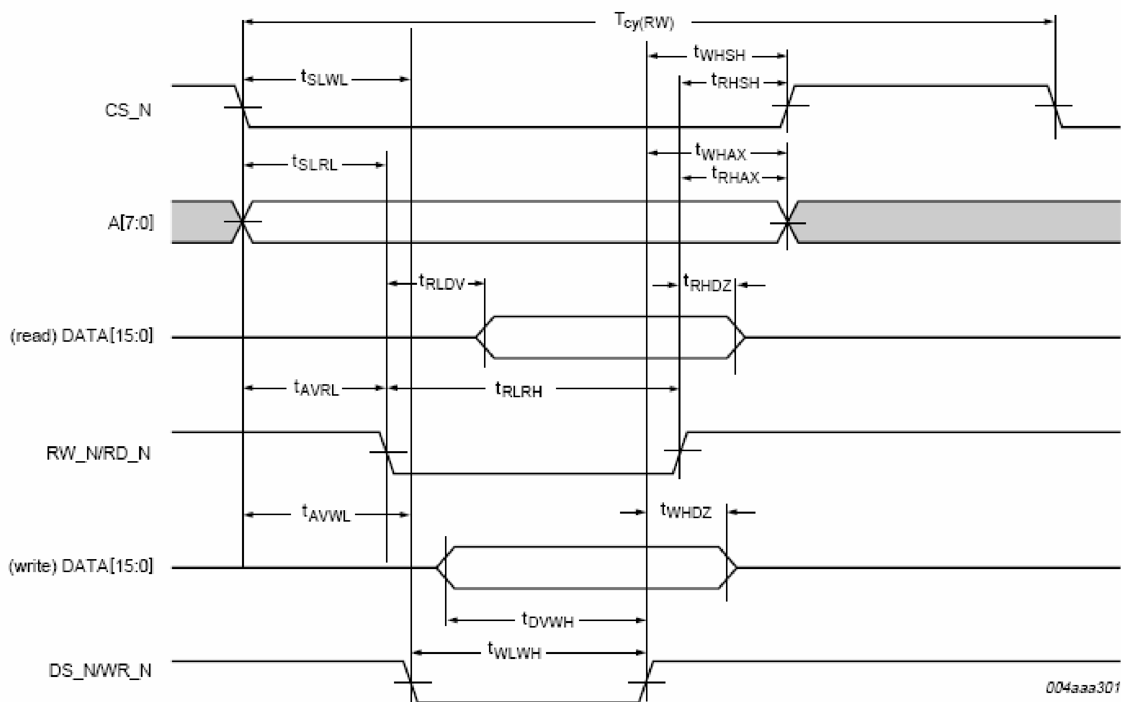
- DATA[15:0] - 16bitů širokou datovou sběrnici
- AD[7:0] - 8bitů širokou adresní sběrnici

- CS_N - chip select
- ALE/A0 používá se pouze ve split bus módů (závisí na pinu MODE1 a BUS_CONF)
    - BUS_CONF = 0 -> Split Bus mode
        - MODE1 = 0 -> Address Latch Enable
        - MODE1 = 1 -> A0 (address data level indicator)
    - BUS_CONF = 1 -> Generic processor mode
        - MODE1 should be 1
- pin RW_N/RD_N (závisí na pinu MODE0)
    - MODE0 = 0 -> Read or Write input
    - MODE0 = 1 -> Read input
- pin DS_N/WR_N (závisí opět na pinu MODE0)
    - MODE0 = 0 -> Data strobe input
    - MODE0 = 1 -> Write input
- READY – Signalizuje připravenost obvodu ISP1583 přijmout další data (1 = připraven)
- INT – interrupt signal, je programovatelný pro různé typy událostí

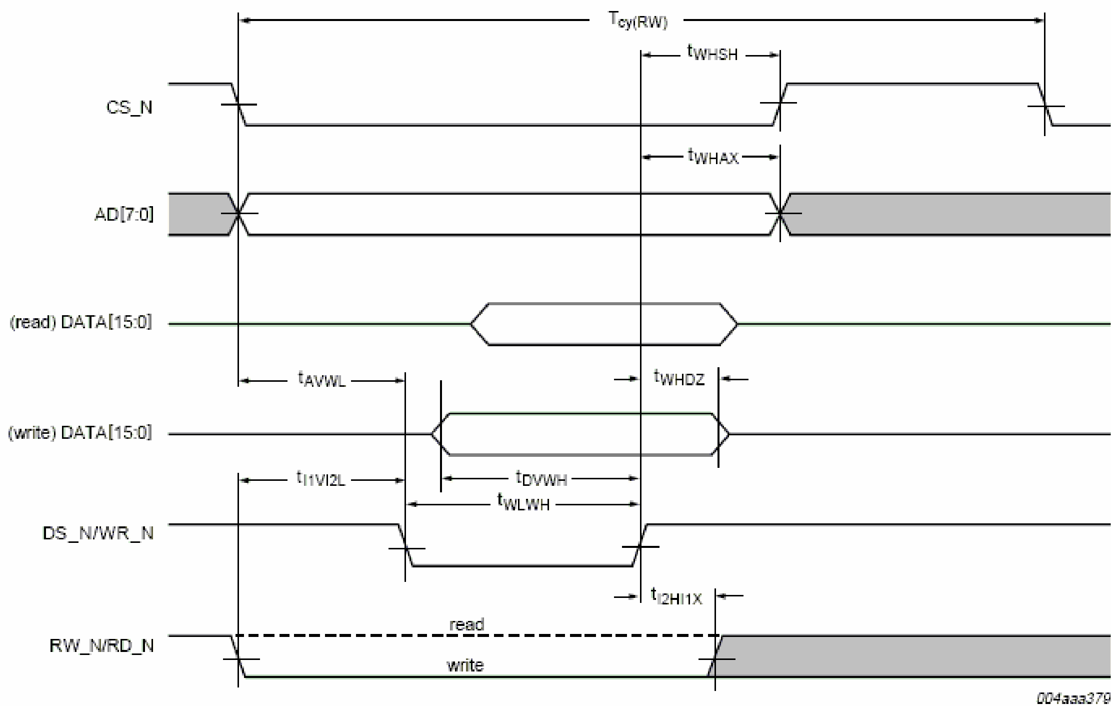
Z výše popsaného přehledu pinů a konfigurací vyplývá, že obvod ISP1583 je možno nastavit pro práci s různými typy rozhraní a procesorů.

Pro jednodušší návrh jsem se rozhodl použít „generic processor“ mód. To znamená, že datová a adresová sběrnice jsou oddělené (data jsou na pinech DATA a adresy jsou na pinech AD).Tento mód je nejjednodušší a je nejsnazší pro oživení, zvládnutí enumerace, komunikace a otestování maximální rychlosti. Nevýhoda tohoto módu je, že zabírá o 16 pinů více než Split Bus mód. Proto ve finální fázi projektu můžeme mód zaměnit za Split Bus mód. Tato operace vyžaduje mít možnost pomocí jumperu změnit logickou hodnotu na pinech MODE1 a BUS_CONFIG. Dále bude potřeba vyměnit nejnižší modul v FPGA, který zprostředkovává ISP1583 driver.

V generic procesor módu je možná podpora dvou typů procesorů. Jednak je zde podpora 8051 režimu (viz. obr. 18) a dále Freescale režimu (viz obr. 19).



obr. 18 – 8051 mód - rozdělená address a data bus



obr. 19 – Freescale mód – rozdělená address a data bus

Freescale režim a režim 8051 se od sebe liší pouze funkcí signálů DS_N/WR_N a RW_N/RD_N. Vzhledem k tomu, že pro řízení obvodu ISP1583 si hodlám vyrobit vlastní modul v obvodu FPGA, můžu si vybrat jak Freescale mód, tak 8051 mód. Nevidím jednoznačné výhody jednoho módu proti druhému. Pro další návrh jsem se rozhodl zvolit Freescale mód. Důvodem je, že RW_N signál (read/write) se dá nastavit spolu s CS_N signálem a dále je potřeba ovládat pouze DS_N signál (data_strobe).

Časování v tomto módu je následující:

$t_{WLWH} = 15$ ns

$t_{DVWH} = 11$ ns

$t_{WHDZ} = 5$ ns

$t_{CY(RW)} = 50$ ns

Bohužel v data sheetu k obvodu ISP1583 není uvedeno, zda vypnutí signálu CS_N je potřeba mezi jednotlivými přístupy k registrům obvodu. Budeme tedy dále předpokládat, že celá smyčka je potřeba přesně provést dle obr. 19. Teoretická maximální rychlost obvodu je pak:

$$throughput_{MAX} = \frac{bits}{T_{CY(RW)}} = \frac{16bits}{50ns} \approx 305Mbps$$

Je ale nutné si uvědomit, že uvedená rychlost 305Mbps je pouze teoretické maximum, kterého nelze dosáhnout. Například není možné pouze zapisovat data, ale je potřeba provádět různé jiné nastavovací operace (například nastavení čísla EP a potvrzení paketu). Zároveň je dobré podotknout, že vysílání dat probíhá až po obdržení požadavku z PC. Zde nastává velké zpoždění, než PC opravdu odešle paket s požadavkem, dále zařízení trvá nějakou dobu, než paket načte a vyhodnotí. Teprve po vyhodnocení je možné určit, jaká data mají být poslána a zahájit samotné posílání dat. Skutečná rychlost tedy bude velmi záležet na velikosti paketů s odpovědí. Další možné zpomalení představuje implementace řídícího automatu a rozhraní mezí procesorem a tímto automatem.

3.1.7.2 Analýza relevantních registrů čipu ISP1583

Obvod ISP1583 poskytuje především buffery a rozhraní mezi fyzickou vrstvou USB a procesorem. Provádí také nejnižší vrstvu USB protokolu jako je bit-stuffing, CRC výpočet, kontrola a potvrzování přijatých transakcí. Veškerá data jsou generována procesorem, tedy i celý enumerační proces a všechny tabulky deskriptorů. Procesor je zodpovědný i za nastavení adresy zařízení a provedení akcí souvisejících se stavy USB, jako je například „bus reset".

Přikládám seznam registrů a jejich částí, které budu využívat v mém řešení. Podotýkám, že jsem se omezil jen na bity, které budou potřebné k realizaci mého řešení. Pro kompletní seznam konfigurace nechť se laskavý čtenář podívá do katalogového listu obvodu ISP1583 [6]. Názvy registrů nepřekládám především proto, že by mohlo dojít k nejednoznačnosti při srovnávání této práce a odkazovaných dokumentů.

3.1.7.2.1 Address register (adresa: 00h)

- bit 7 = DEVEN – 1 -> Povoluje zařízení. Zařízení začne odpovídat na host pakety.
- bity [6:0] = DEVADDR – nastavuje adresu zařízení. Tato adresa má být 0 po připojení zařízení k USB. Poté při enumeračním procesu host řadič pošle paket s přidělenou adresou a procesor musí tento registr nastavit podle přidělené adresy.

3.1.7.2.2 Mode register (adresa: 0Ch)

používané bity budou:

- bit 8 = BUSSTAT – je logická 1 pokud detekuje napájecí napětí z USB host řadiče.
- bit 4 = SFRESET – logická 1 provede softwarový reset obvodu, tento reset je podobný hardwarovému resetu
- bit 3 = GLINTENA – provádí globální povolení signalizace nemaskovaných přerušení na pinu INT. Tento bit bude zakázán, protože MicroBlaze procesor je v návrhu zodpovědný pouze za komunikaci s USB a proto bude periodicky dotazovat příznakový interrupt registr.
- bit 0 = SOFTCT – logická 1 připojí 1.5kΩ pull-up rezistor na D+ pin a umožní tím host řadiči detekovat USB zařízení a provede tak tedy připojení zařízení k USB.

3.1.7.2.3 Interrupt Configuration register (adresa: 10h)

- bit [7:6] = CDBGMOD[1:0] – řídí generování přerušení pro control endpoint 0.
- bit [5:4] = DDBGMODIN[1:0] – řídí generování přerušení pro data endpointy 1-7 IN.
- bit [3:2] = DDBGMODOUT[1:0] – řídí generování přerušení pro data endpoint 1-7 OUT.
- bit 1 = INTLVL – řídí formu signalizace interrupu na pinu INT. 0 = úroveň, 1 = 60 ns puls.
- bit 0 = INTPOL - řídí polaritu signalizace interruptu na pinu INT. 0 = aktivní v 0, 1 = aktivní v 1.

Bity INTLVL a INTPOL nebudeme v našem řešením potřebovat, protože nepoužíváme pin INT. Bity CDBGMOD, DDBGMODIN a DDBGMODOUT budeme

potřebovat nastavit pro zajištění požadovaného generování příznaků v příznakovém interrupt registru. Možné nastavení těchto registrů je následující:

- 00h – Interrupt pro všechny ACK a NAK
- 01h – Interrupt pro všechny ACK.
- 1Xh – Interrupt pro všechny ACK a první NAK.

V našem řešení jsem se rozhodl používat pro všechny 3 registry (CDBGMOD, DDBGMODIN a DDBGMODOUT) nastavení 1Xh. Tím získáme příznaky přerušení na každý úspěšně odeslaný a přijatý paket. Zároveň získáme příznaky přerušení pro první neúspěšně přijatý paket (typický plný buffer) nebo pro první neúspěšně odeslaný paket (data nebyla připravena po obdržení paketu IN).

3.1.7.2.4 Interrupt Enable register (adresa: 14h)

- bit 25 = IEP7TX – Logická 1 povoluje přerušení od EP7 ve směru IN (device->host).
- bit 24 = IEP7RX – Logická 1 povoluje přerušení od EP7 ve směru OUT (host->device).
- bit 23 až bit 10 – Postupně IEP6TX, IEP6RX až IEP0TX a IEP0RX, funkce je obdobná jako u dvou výše popsaných bitů IEP7TX a IEP7RX.
- bit 8 = IEP0SETUP – logická 1 povoluje přerušení od příchozího SETUP paketu.
- bit 5 = IEHS_STA – Logická 1 povoluje přerušení při změně na full-speed.
- bit 0 = IEBRST – Logická 1 povoluje přerušení při detekci bus resetu.

V našem řešení budeme používat pouze EP0 IN a OUT který se bude používat pro enumeraci zařízení. Dále budeme používat EP1 OUT, který se bude používat pro přenos řídících a konfiguračních paketů. Naposled budeme využívat EP1 IN, který bude sloužit k odesílaní odpovědí (naměřená data, stav zařízení, potvrzení přijetí příkazu).

Dále je pro enumeraci potřebné povolit příznaky přerušení IEP0SETUP a IEBRST na tyto příznaky bude třeba reagovat parsováním příchozího SETUP paketu a inicializací obvodu po detekci bus resetu.

3.1.7.2.5 Interrupt (adresa: 18h)

Tento registr přesně odpovídá registru Interrupt Enable Registr popsanému v kapitole 3.1.7.2.4. Rozdílem je, že v Interrupt registru se zobrazují příznaky přerušení, které jsou povolené v Interrupt Enable Registr. Příznaky se mažou zápisem logické 1 do bitů, které zamýšlíme smazat.

3.1.7.2.6 Chip ID (adresa: 70h)

Toto je registr pouze pro čtení. Jeho hodnota je pevně definovaná jako verze čipu. Náš vybraný čip by měl vracet hodnotu 0015 8230h. Při inicializaci firmware ověří, zda čip odpovídá správným kódem a bude pokračovat pouze v případě, že tomu tak je.

3.1.7.2.7 Endpoint Index (adresa: 2Ch)

Tento registr je klíčový k výběru end pointu, nad kterým budeme pracovat, budeme-li přistupovat k následujícím registrům:

- Buffer length
- Buffer status
- Control function

- Data port
- Endpoint MaxPacketSize
- Endpoint type

Význam těchto registrů bude vysvětlen v následujících kapitolách. Endpoint Index registr se skládá z následujících bitů:

- bit 5 = EP0SETUP – tento bit určuje, zda budeme přistupovat k SETUP datům (při logické 1) nebo k datům vybraného end pointu (viz níže). Zde je vhodné zmínit, že SETUP data se ukládají do speciálního bufferu a bylo by tedy vhodné mluvit o speciálním end pointu SETUP. Příslušná dokumetnace k obvodu ISP1583 tak ovšem nečiní.
- bit [4:1] = ENDPIDX[3:0] – Vybere index end pointu, se kterým se bude pracovat použitím registrů „buffer length“, „buffer status“, „control function“, „data port“, „endpoint MaxPacketSize“, „endpoint type“. Pokud chceme přistupovat k SETUP datům, měla by hodnota této položky být 0.
- bit 0 = DIR – rozhoduje, zda se jedná o ednpoint IN či OUT
  - 0 – EP OUT (RX)
  - 1 – EP IN (TX)

3.1.7.2.8 Control Function register (adresa: 28h)

Tento registru slouží k ovládání end pointu. To, o který end point se jedná, je určeno hodnotou registru „Endpoint Index“ (viz kapitola 3.1.7.2.7). Možnosti nastavení end pointu jsou:

bit 4 = CLBUF – Logická 1 umožňuje vymazat RX buffer. Na TX buffer tento bit nemá vliv. RX buffer je sice automaticky vymazán při kompletním přečtení dat, ale vymazání bude užitečné při inicializaci zařízení.

bit 3 = VENDP – Logická 1 potvrdí data paket v TX (IN) bufferu. Buffer je sice potvrzen automaticky, když dosáhne velikosti pro něj přednastavené, toto je však cesta, jak odesílat menší pakety.

bit 2 = DSEN – Logická 1 povolí „DATA STAGE“ v Controll transferu (viz kapitola 3.1.3.2.1). Je na mikroprocesoru, aby rozhodl podle hodnoty dat ze SETUP paketu, zda pro příslušný control transfer je potřeba DATA STAGE.

bit 1 = STATUS – Tento bit se používá pouze pro end point 0 (controll end point). Logická 1 oznamuje, že procesor zpracoval control transfer. Až host vyšle paket pro ověření STATUS STAGE, je odpovězeno kladně (to, co je kladná odpověď viz kapitola 3.1.3.2.1.3).

bit 0 = STALL – Logická 1 nastaví end point do stavu STALL.

3.1.7.2.9 Data Port register (adresa: 20h)

Tento registr slouží k přístupu do end point bufferu. Tento buffer je koncipován jako FIFO buffer.

Pro IN endpoint se do registru postupně zapisují data. Při každé operaci se automaticky posune ukazatel na další položku bufferu. Po zapsaní všech dat (velikost je nastavena v EndPoint MaxPacketSize) se buffer automaticky potvrdí a pošle při první žádosti od host řadiče.

Pro OUT endpoint se postupně čtou data. Při každé operaci se automaticky posune ukazatel na další položku bufferu. Po přečtení všech dat se buffer automaticky označí za prázdný a je připraven pro zápis nových dat od host řadiče.

3.1.7.2.10 Buffer Length register (adresa: 1Ch)

Pro IN (TX) end point tento registr slouží k nastavení menší velikosti odesílaných dat, než je velikost „MaxPacketSize“. Data paket je automaticky nastaven jako validní, pokud se do „Data Port registr“ zapíše stejný počet dat, jako je nastaven v registru „Buffer length“.

Pro OUT (RX) end point je v registru uložena velikost přijatých dat.

3.1.7.2.11 Buffer Status register (adresa: 1Eh)

Tento registr je pouze pro čtení a určuje zda end point buffer je připraven. Při zápisu do IN (TX) bufferu je potřeba počkat 200 ns před čtením tohoto registru. Při přijetí přerušení z OUT (RX) end point bufferu je rovněž potřeba 200 ns počkat před čtením tohoto bufferu.

- bit [1:0] – BUF[1:0] – určují, zda buffer je volný
  - „00“ = buffer je volný
  - „01“ nebo „10“ = jeden buffer je plný (je možné end point nastavit jako dvojitý buffer)
  - „11“ oba buffery jsou plné

3.1.7.2.12 Endpoint MaxPacketSize register (adresa: 04h)

Tento register nastavuje maximální velikost paketu, a tedy bufferu (kromě bufferu end pointu 0 – ten je nastaven na pevnou hodnotu).

- bity [10:0] = FFOSZ[10:0] – nastavuje velikost end point bufferu v bytech.

Zde je vhodné upozornit, že celková velikost všech použitých end point bufferů včetně bufferu end pointu 0 a speciálního bufferu pro SETUP data nesmí přesáhnout 8192B.

Na tomto místě bych rád podotkl, že při studiu Application Note: Frequently Asked Question [8] jsem zjistil, že registr MaxPacketSize je potřeba nastavit ve specifickém pořadí, jinak celková paměť není přidělena správně a může docházet ke korupci dat. Toto pořadí je:

1. Zakázat veškeré end-pointy
2. Nastavit velikost MaxPacketSize u všech použitých end-pointů.
3. Nastavit typy všech použitých end-pointů a tyto posléze povolit.

V data sheetu obvodu ISP1583 [6] není jediná zmínka o této důležité informaci.

V mém řešení jsem se rozhodl pro velikosti end point bufferů co největších, aby zmírnily dopady režie nastavení obvodu a zvýšily tak rychlost. Oba použité buffery chci mít dvojitě bufferované (viz další kapitola 3.1.7.2.13). Maximální možná velikost každého bufferu by pak byla:

$$\frac{\text{VelikostPameti - EP0IN - EP0OUT - EPSETUP}}{\text{PocetEP_Bufferu * paralelniBuffery)}} = \frac{(8192\text{B - 64B - 64B - 8B})}{(2*2)} = 2014B$$

Dále aby velikost bufferu byla mocninou 2 rozhodl jsem se pro velikost 1024B.

3.1.7.2.13 Endpoint Type register (adresa: 08h)

Tento registr slouží k nastavení parametrů vybraného end pointu. Možná nastavení jsou následující:

- bit 3 = ENABLE – logická 1 povolí vybraný end point buffer. Zakázáním tohoto bitu a opětovným povolením se dá vynulovat Data Toggle bit.
- bit 2 = DBLBUF – Logická 1 povolí dvojitý bufferu pro příslušný end point.

- bit [1:0] = ENDPTYP[1:0] – Tyto bity nastaví typ end pointu. Možná nastavení jsou:
  - "00" – end-point není použit
  - "01" – Isochronous
  - "10" – Bulk
  - "11" – Interrupt

V našem řešení povolíme EP1 IN a EP1 OUT. Oběma povolíme dvojité bufferování. Tím se předpokládá velké zrychlení při přenášení více paketů za sebou, protože nebude potřeba čekat, než si počítač paket vyzvedne, ale bude možné již připravovat následující paket. Typ obou end pointů nastavíme na Bulk z důvodů již zmíněných v kapitole 3.1.5.

3.2 FPGA USB Driver

USB driver v FPGA bude modul řešící rozhraní mezi MicroBlaze procesorem a USB IO deskou. V předchozí kapitole (3.1.6) jsem vybral pro USB IO desku čip ISP1583. Řídící interface tohoto čipu je podrobně analyzován taktéž v předcházející kapitole. FPGA USB Driver modul bude řešit přístup k registrům obvodu ISP1583. Veškerá data a adresy registrů bude nastavovat firmware v procesoru MicroBlaze. V modulu USB Driveru bude tedy potřeba naimplementovat jednoduchý automat, který nastaví adresu a data zadané firmwarem a bude ovládat 3 řídící signály: CS_N, RW_N, DS_N ve sledu, jaký je popsán na obr. 19 v kapitole 3.1.7.1. Tento automat bude realizován jako mealy, protože ten je jednodušší k návrhu a je potřeba méně stavů k jeho zakódování. Bude však potřeba registrovat výstupy, aby nedocházelo k překmitům.

USB driver modul bude připojen k procesoru MicroBlaze přes PLB sběrnici. To je nejtypičtější typ připojení periferií k procesoru a je také nejjednodušší.

3.3 DDR Paměť

DDR Paměť je dána vybraným vývojovým kitem ML402. Podle dokumetnace [9] k tomuto kitu jsou osazeny dvě DDR paměti s označením Infineon HYB25D256160BT-7 (nebo kompatibilní). Při ověření skutečně osazeného čipu jsem zjistil, že jsou osazeny dvě paměti MT46V16M16-6T od firmy Micron.

3.3.1 Analýza kapacity

Paměť MT46V16M16-6T má 256Mbit. Pokud jsou osazené dva moduly, je na vývojovém kitu k dispozici 512Mbit. Při minimální požadované šířce měřícího portu 8 bitů jsme schopni uložit více než 67.1 milionů vzorků. Při minimální požadované frekvenci 50MHz to odpovídá délce měření více než 1.34 sekund. Tyto parametry jsem shledal více než dostatečnými. Srovnáme-li to například s profesionálním logickým analyzátorem Agilent 16801A, který stojí 200 000CZK, zjistíme, že tento je při maximální rychlosti schopen zaznamenat pouze 32ms (nutno ovšem podotknout, že měřící frekvence je 1GHz při měření 17 kanálů, což je 20x větší frekvence a 2x více kanálů, než jsou naše minimální požadavky). Kapacitu pamětí osazených na vývojovém kitu je tedy vyhovující pro naše řešení.

3.3.2 Analýza rychlosti

Paměť MT46V16M16-6T má v katalogovém listu [10] uvedeno, že maximální rychlost je DDR333. To znamená, že maximální frekvence je sice „pouze" 167MHz ale,

protože data jsou ukládána na obou hranách hodin, je efektivní frekvence 333MHz. Celková šířka datové sběrnice jednoho modulu je 16 bitů. Dva moduly mají tedy šířku datové sběrnice 32 bitů. Teoretická maximální rychlost ukládání dat je tedy více než 10.4Gbit/s, při minimální šířce měřícího portu 8 bitů je tedy maximální rychlost měření 1.3 Gsamples/s. Je nutné si ale uvědomit, že paměť je třeba refreshovat v pravidelných intervalech, to ubírá část efektivní rychlosti práce s pamětí. Za zmínku také stojí, že je potřeba několik taktů počkat na otevření banky paměti a stejně tak je potřeba čekat několik taktů na otevření řádku paměti a sloupce. Pokud je ovšem řadič DDR paměti napsán efektivně a k paměti je přistupováno sekvenčně (což při postupném ukládání vzorků není problém), lze většina zpoždění adresace vhodně maskovat adresací další paměťové buňky již při práci s aktuální paměťovou buňkou.

Celkovou režii práce s pamětí je v této fázi nemožné určit přesně. Je to především dáno kvalitou použitého DDR řadiče, ale i způsobem práce s pamětí. Přesných výsledků by se dalo dobrat extensivním testováním konkrétního řešení. V této chvíli je však bezpečné předpokládat, že při kvalitním DDR řadiči nebude režie paměti větší než 20% (předpokládám že to bude daleko méně) .

Pokud z maximální rychlosti 1.3Gsamples/s odebereme 20% rezervovaných pro režii práce s pamětí, získáme stále víc než 1Gsample/s. Tato vzorkovací frekvence je více než dostatečná. Srovnáme-li tuto vzorkovací frekvenci se vzorkovací frekvencí logického analyzátoru Agilent 16801A, zjistíme, že je stejná jako jeho maximální vzorkovací frekvence (nutno ovšem podotknout, že logický analyzátor Agilent 16801A touto frekvencí vzorkuje 2x více kanálů).

Rychlost samotných pamětí je tedy vyhovující. Rád bych však předeslal, že pro využití této rychlosti by ji musel podporovat celý systém a bohužel použitý obvod Virtex-4E není schopen zpracovávat signály frekvencí 1GHz.

3.4 DDR Controller

Tato část bude zajišťovat rozhraní mezi procesorem a DDR pamětí. Zároveň bude zajišťovat rozhraní mezi modulem jádra logického analyzátoru a DDR pamětí. Bude možno nastavit adresu v paměti a operaci (čtení/zápis).

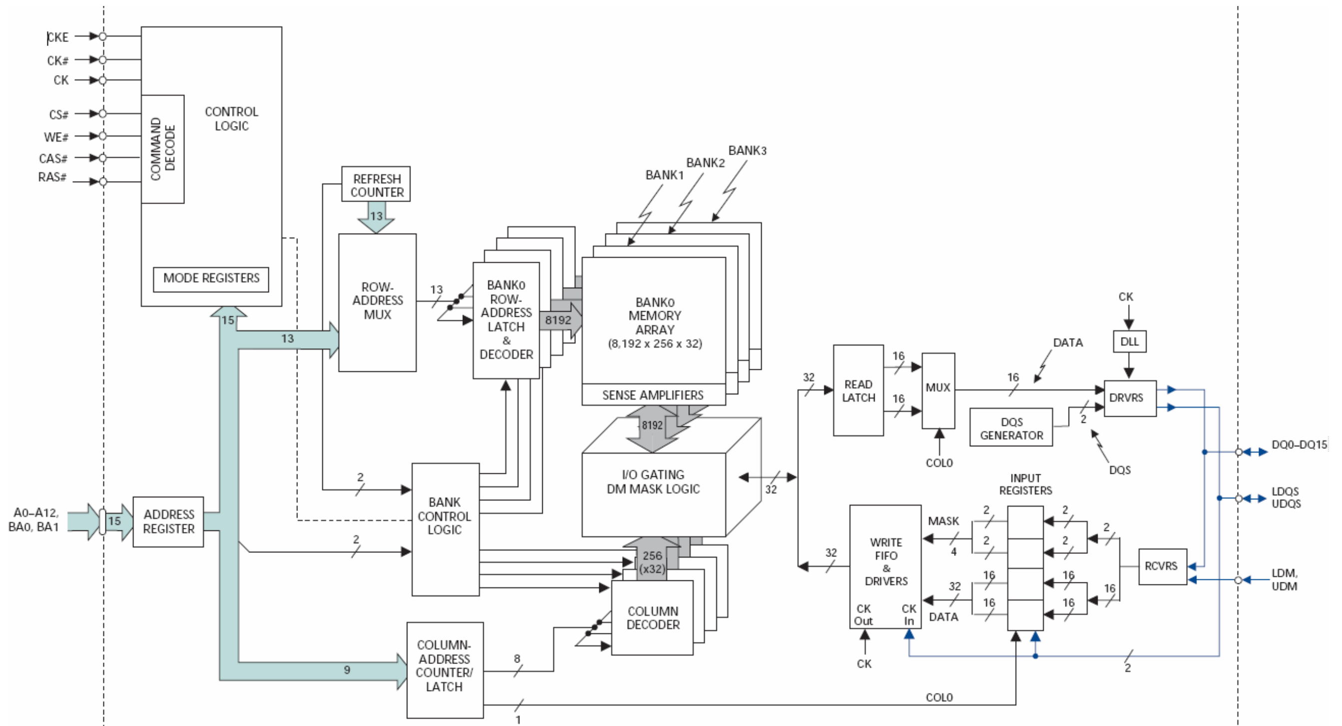
Pro operaci zápis bude poskytnut 64bitový datový registr (tedy 2x širší, než je šířka datového portu obou DDR SDRAM pamětí). Bude tedy možné data zapisovat poloviční rychlostí, než jsou vystavována na datový port připojený k DDR SDRAM obvodu. Pro maximální rychlost bude v modulu DDR řadiče implementován FIFO buffer pro vstupní data a příkazy (čtení/zápis), aby bylo dosaženo maximální propustnosti tohoto rozhraní.

Pro operaci čtení bude v modulu DDR řadiče implementován výstupní FIFO buffer, aby bylo dosaženo maximální propustnosti i pro vyčítání dat.

Modul DDR řadiče je důležitou součástí systému a jeho návrh bude relativně složitý. Jedním z hlavních problémů je potřeba vystavovat data na obě hrany hodinového signálu (tedy dvojnásobnou frekvencí, než je frekvence hodin v obvodu). Tento problém není možné vyřešit snížením frekvence DDR pamětí, protože minimální frekvence DDR pamětí podle katalogového listu [10] je ~77MHz, což znamená, že minimální frekvence zpracovávání dat bude 154MHz. Zároveň by se navíc snížila maximální možná frekvence ukládání měřených vzorků.

Pro návrh DDR řadiče je potřeba detailně prostudovat dokumentaci k DDR pamětem. Především rozhraní DDR paměti a způsob inicializace. V dalším textu proberu relevantní části dokumentace k použité DDR paměti, není však mým cílem jakkoliv suplovat katalogový list, a proto doporučuji čtenáři si daný katalogový list [10] prostudovat samostatně.

3.4.1 Rozhraní DDR pamětí MT46V16M16-6T



obr. 20 – 16M x 16 Funkční blokový diagram

Na obr. 20 je zobrazeno schéma paměti včetně všech vstupů a výstupů. Předně je důležité si uvědomit, jak je paměť adresovaná. Z obrázku je patrné, že se adresuje jedna ze 4 bank, řádek (ROW) a sloupec (COL), což udává přesnou pozici 32bitového slova. Z obrázku je patrno, že adresa banky má 2 bity. Adresa řádku (ROW) má 13 bitů. Adresa sloupce má 9 bitů, avšak nejméně významný bit pouze vybírá, které půlslovo se vyšle první. Proto jej budeme předpokládat vždy 0 a důležitých pro nás bude horních 8 bitů. Takto dokážeme naadresovat celých 256Mbitů paměti viz následující vzorec: $2^{ROW}*2^{COL}*2^{BA}*32 = 2^{13}*2^{8}*2^{2}*32 = 256$ Mbit. Uvědomme si, že předpona Kilo a Mega je v číslicových systémech 2^{10}, tedy 1024, ne 1000 jako u ostatních disciplín jako je matematika a fyzika.

Probereme si nyní postupně vstupní a výstupní piny:

- **A[12:0]** = Address[12:0] – Slouží především k adresaci řádku (ROW) a sloupce (COL). O tom, zda se jedná o ROW nebo COL adresu, rozhodují signály RAS a CAS viz níže. Při inicializaci slouží tento port k nastavení konfiguračních registrů.
- **BA[1:0]** = Bank address[1:0] – Slouží k výběru jedné ze 4 bank. Při inicializaci slouží k výběru konfiguračního registru.
- **CK a CK#** = Clock – Jedná se o vstup diferenciálního hodinového signálu.
- **CKE** = Clock enable – Logická 1 povolí hodinový signál.
- **CS#** = Chip Select – Logická 0 povolí dekodér příkazů.
- **(L/U)DM** = Input Data Mask – Logická 1 zabrání zápisu dat při operaci zápisu. DM je posíláno spolu s daty a je vzorkováno na obě hrany hodin. LDM je pro spodní byte dat a UDM je pro horní byte dat.
- **RAS#** = Row Address Select – Slouží k výběru řádku, ale i k jiným příkazům viz dále.
- **CAS#** = Column Address Select – Slouží k výběru sloupce, ale i k jiným příkazům viz dále.

- **WE#** = Write Enable – Slouží k výběru příkazu viz dále.
- **DQ[15:0]** = Data[15:0] – Obousměrný datový port.
- **(L/U)DQS** = Data Strobe – Obousměrný signál pro synchronizaci dat. Při zápisu do DDR paměti má být vysílán řadičem paměti a má být uprostřed platných dat. Při čtení DDR paměti je vysílán DDR pamětí a je zarovnán s hranou datového signálu. LDQS je pro nižší byte dat. UDQS je pro vyšší byte dat.

Z popsaných signálů a z obr. 20 je zřejmé, že pouze porty data (DQ) a data mask (DM) jsou vzorkovány na obě hrany hodin. Všechny ostatní signály včetně adresy jsou klasicky vzorkovány jen na náběžnou hranu hodin. Dokonce i samotná data a datová maska jsou sice navzorkovány na obě hrany hodinového signálu, dále jsou však uložena do FIFO bufferu dvojnásobné šířky a ve zbytku paměti jsou již zpracovávána standardně pouze na náběžnou hranu hodin.

Name (Function)	CS#	RAS#	CAS#	WE#	Address
DESELECT (NOP)	H	X	X	X	X
NO OPERATION (NOP)	L	H	H	H	X
ACTIVE (select bank and activate row)	L	L	H	H	Bank/row
READ (select bank and column and start READ burst)	L	H	L	H	Bank/col
WRITE (select bank and column and start WRITE burst)	L	H	L	L	Bank/col
BURST TERMINATE	L	H	H	L	X
PRECHARGE (deactivate row in bank or banks)	L	L	H	L	Code
AUTO REFRESH or SELF REFRESH (enter self refresh mode)	L	L	L	H	X
LOAD MODE REGISTER	L	L	L	L	Op-code

obr. 21 – Příkazy

Na obr. 21 jsem uvedl všechny možné příkazy do DDR paměti, které bude potřeba podporovat v DDR řadiči. Proberu je nyní podrobněji:

3.4.1.1 Deselect, No Operation

Tyto příkazy jsou funkčně ekvivalentní. Nevyžadují žádnou další operaci od DDR paměti a neovlivní jakoukoliv již prováděnou operaci.

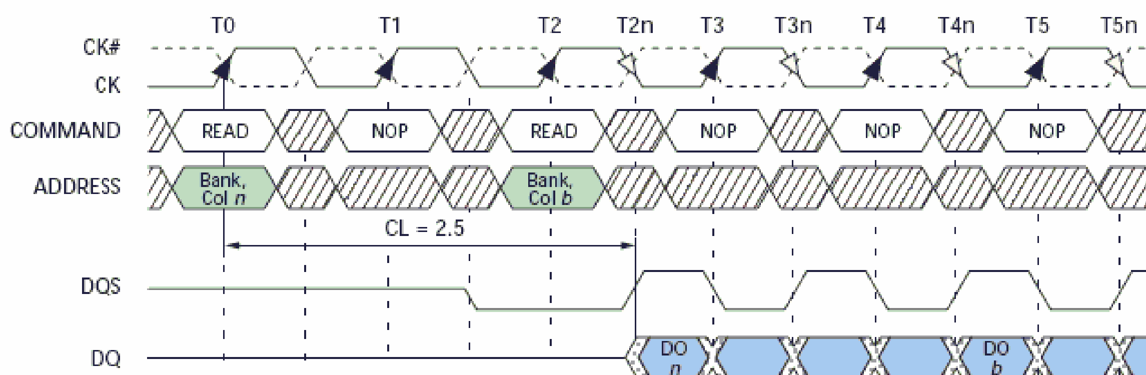
3.4.1.2 Activate

Příkaz „activate" je použit k otevření řádku ve vybrané bance. Adresa řádku je určena piny Address[12:0]. Banka je pak vybrána piny Bank_address[1:0]. Číst a zapisovat je možné pouze do předem otevřeného řádku nejdříve však po 15ns od příkazu activate. Řádek zůstane otevřen až do zavolání příkazu „precharge". Dalšímu volání příkazu „activate" do stejné banky musí předcházet příkaz „precharge". Následující příkaz „activate" nesmí být zavolán dříve než po 60ns od předchozího příkazu „activate"

3.4.1.3 Read

Příkaz „read" začne takzvané burst čtení. Burst čtení znamená, že se čte po sobě více dat – takzvaný blok dat. Velikost tohoto bloku je dána konfigurací paměti při inicializaci a obvykle může být 2, 4 nebo 8. Při příkazu „read" je zároveň potřeba nastavit adresu banky pomocí pinů Bank_address[1:0] a adresu sloupce (COL) pomocí pinů Address[8:0]. Pinem Address[10] se nastavuje takzvaný „auto precharge", to znamená automatické uzavření řádku. Pokud je Address[10]=1 dojde po přečtení bloku dat k automatickému uzavření řádku. Pokud je Address[10]=0 k automatickému uzavření řádku nedojde a je možné zavolat další příkaz "read" z libovolného sloupce

stejného řádku. Tento příkaz je možné zavolat již při probíhajícím čtení, a tím zamaskovat zpoždění, které je mezi posláním příkazu a vrácením platných dat z paměti – takzvaná CAS latency. Je však nutné dbát přesného vystavení příkazu “read”, aby nová data se nezačala číst dříve, než jsou všechna data z dříve požadovaného bloku dat přečtená, jinak data z nového požadavku budou vystavena místo konce původně požadovaných dat. Po sobě jdoucí korektní čtení, které maskuje CAS latenci, je znázorněno na obrázku obr. 22. Délka bloku je nastavena na 4. Pokud by délka bloku byla nastavena na 8, poslední 4 data ze sloupce „n“ by byly překryty prvními 4 daty ze sloupce „b“. Pokud by délka bloku byla nastavena jen 2, projevila by se část CAS latenci. Pro rychlejší navazování bloků dat o velikosti 2, by příkazy „read“ musely jít hned za sebou.



obr. 22 – Navazující READ příkaz

Podrobnosti ohledně časování a synchronizace procesu čtení dat z paměti najdete v kapitole 3.4.3.

3.4.1.4 Write

Příkaz „write“ slouží ke spuštění cyklu zápisu bloku dat. Je zároveň potřeba nastavit adresu banky pomocí pinů Bank_address[1:0] a adresu sloupce (COL) pomocí pinů Address[8:0]. Po přesném počtu taktů je potřeba začít vysílat na piny Data[15:0] platná data a na piny Data_stobe synchronizační signál zarovnaný se středem dat (typicky hodinový signál posunutý o 90°). Také je nutné souběžně s daty vysílat na piny Data_mask logickou hodnotu 0, aby došlo k zápisu dat. Naopak nechceme-li, aby některá data byla uložena, zapíšeme na piny Data_mask logickou hodnotu 1. Maskování dat je užitečné, protože velikost bloku zapisovaných dat je neměnná a určena napevno při inicializaci paměti. Když tedy nechceme uložit celý blok dat, ale pouze určitou část bloku, využijeme k tomu právě maskování pomocí signálu Data_mask.

3.4.1.5 Burst terminate

Slouží k předčasnému ukončení čtení posledního požadovaného bloku dat.

3.4.1.6 Precharge

Tento příkaz slouží k deaktivaci (uzavření) řádku. To je potřeba před otevřením jiného řádku nebo před obnovením paměti (takzvaný refresh paměti). Zároveň s příkazem „precharge“ musíme pomocí pinů Bank_address[1:0] nastavit adresu banky, ve které chceme deaktivovat řádek. Pinem Address[10] vybereme, zda chceme deaktivovat řádky ve všech bankách (Address[10]=1) nebo pouze ve vybrané bance (Address[10]=0).

3.4.1.7 Auto Refresh

Slouží k obnovení dat v paměti. Mělo by k tomu docházet postupnou aktivací řádků, přečtením a znovunahráním hodnoty. Tento příkaz je potřeba kvůli fyzickému provedené SDRAM DDR pamětí. Ty udržují uložené hodnoty v miniaturních kondenzátorech a tyto mají nepatrný svod a samovolně se vybíjejí. Bez obnovování by se tedy paměť po určité době vymazala. O těchto detailech však nejsou informace v použitém katalogovém listu a není potřeba je znát, uvádím je však, aby bylo zřejmé, proč je potřeba periodicky volat příkaz „Refresh“.

3.4.1.8 Self Refresh

Příkaz “Self Refresh” je obdobou příkazu “Auto Refresh”, pouze se dokáže vykonávat i se zakázaným signálem hodin (je dobré podotknout, že hodiny je potřeba stále generovat).

3.4.1.9 Load Mode Register

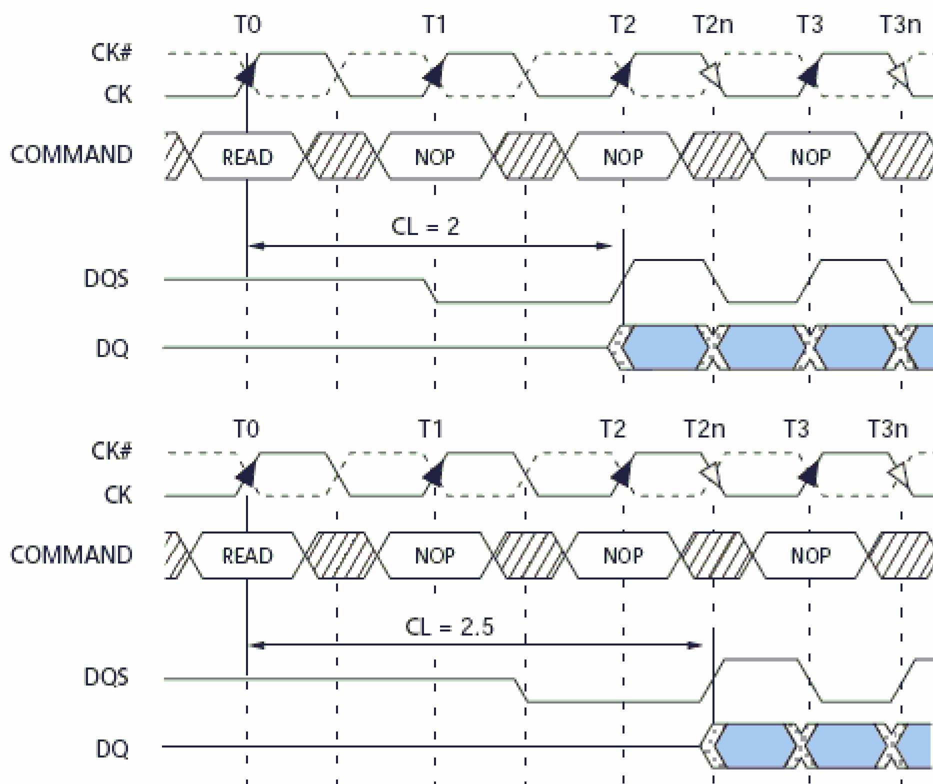
Příkaz “Load Mode Register” slouží k nahrání konfiguračních dat do DDR SDRAM paměti. Konfigurační registr je vybrán pomocí pinů Bank_address[1:0] a hodnota konfiguračního registru je přečtena z pinů Address[12:0]. Tento příkaz je možné použít, jen když jsou všechny banky ve stavu „Idle“. Uvádím strukturu obou konfiguračních registrů:

- BA[1:0] = “00” – Mode Register
    - Address[13:7] – nastaví režim
        - 0 = Normální
        - 2 = Normální + reset DLL
    - Address[6:4] – CAS Latency
        - 2 = CAS latency 2 cykly
        - 6 = CAS latency 2.5 cyklu
    - Address[3] – Burst Type
        - 0 = sekvenční čtení
        - 1 = prokládané čtení
    - Address[2:0] – Burst length:
        - 1 = délka 2
        - 2 = délka 4
        - 3 = délka 8
- BA[1:0] = “01” - Extended mode register
    - Address[13:1] - Drive Strength
        - 0 normální
        - 1 snížená
    - Address[0] - DLL – 0 enable, 1 disable

Pro svůj řadič jsem se rozhodl použít burst length délky 4. Vzhledem k tomu, že budu ukládat bloky dat, můžu bez problémů zvolit delší burst length. Krátký burst length se hodí v případě, že je potřeba přistupovat k jednotlivým bytům umístěným na různých adresách.

Burst type zvolím sekvenční. Naměřená data tedy budou ukládána v paměti jako sekvenční bloky dat.

Význam CAS latency je patrný z obr. 23. CAS latency zvolím co nejkratší, tedy 2, i když tento údaj není kritický pro propustnost, protože při správném návrhu řadiče a sekvenčním přístupu se dá beze zbytku maskovat zavoláním další operace čtení již v průběhu předchozí operace čtení. Při velkých blocích dat je proto CAS latency téměř irelevantní.



obr. 23 – CAS Latency

3.4.2 Inicializace DDR paměti

Před prvním použitím DDR paměti musí DDR řadič paměť inicializovat. Pro tento účel bude muset být napsán automat, který provede následující kroky v uvedeném pořadí:

1. Signál CKE se nastaví do logické 0
2. Začne se generovat stabilní hodinový signál na piny CK a CK#.
3. Počká se alespoň 200µs.
4. Signál CKE se nastaví do logické 1 a vystaví se alespoň jeden příkaz NOP.
5. Vystaví se příkaz "Precharge All".
6. Po dobu 15ns se přiloží příkaz NOP.
7. Vystaví se příkaz LMR a nastaví se "Extended mode registr" na hodnotu 0 (tím se povolí DLL, podrobnosti viz kapitola 3.4.1.9).
8. Po dobu 12ns se přiloží příkaz NOP.
9. Vystaví se příkaz LMR a nastaví se "Mode registr" na hodnotu 121h (tím se resetuje DLL a nastaví "CAS latency", "Burst Type" a "Burst Length" podrobnosti viz kapitola 3.4.1.9). Od tohoto kroku je potřeba počkat minimálně 200 hodinových taktů do prvního příkazu "read"
10. Po dobu 12ns se přiloží příkaz NOP.
11. Znovu se vystaví příkaz "Precharge All".
12. Po dobu 15ns se přiloží příkaz NOP.
13. Vystaví se příkaz "Auto Refresh".
14. Po dobu minimálně 72ns se vystaví příkaz NOP.
15. Znovu se vystaví příkaz "Auto Refresh".
16. Po dobu minimálně 72ns se vystaví příkaz NOP.
17. Vystaví se příkaz "LMR" a nastaví se "Mode registr" na hodnotu 021h (tím se vynuluje bit resetování DLL, podrobnosti viz kapitola 3.4.1.9).

18. Po dobu 12ns se přiloží příkaz NOP.
19. Nyní je paměť inicializovaná. Pokud je potřeba provést příkaz „read“, je potřeba zajistit, aby uplynulo alespoň 200 hodinových taktů od kroku 9.

3.4.3 Proces čtení dat

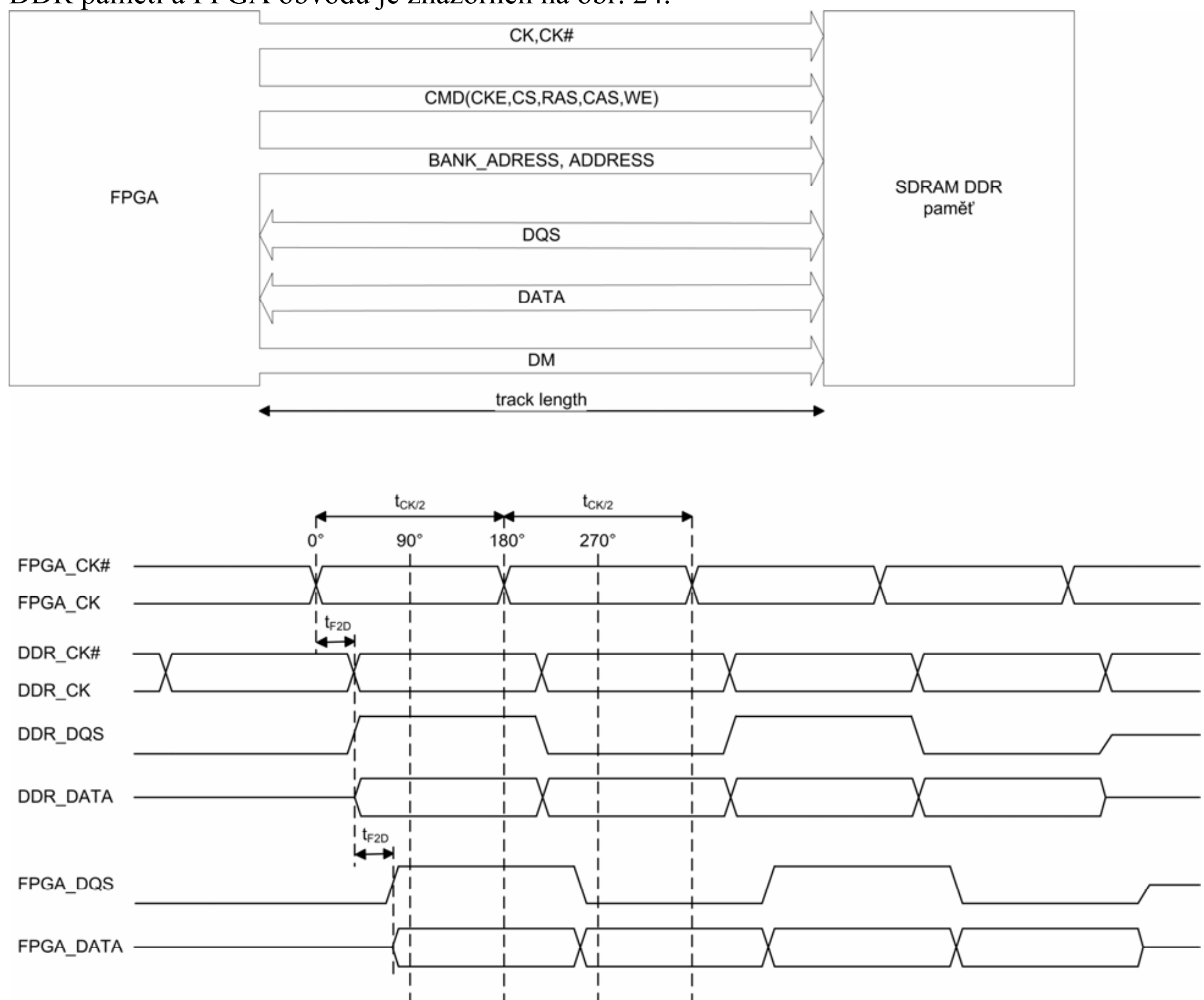
V tomto odstavci se budeme věnovat problematice čtení dat z DDR SDRAM. Hlavní problémy jsou:

1. Délka vodiče významně zpožďuje odpověď DDR pamětí
2. Frekvence je relativně rychlá, a proto doba vystavení dat je relativně krátká.

Kombinaci těchto problémů je v praxi velký problém vyřešit. Budeme hledat způsob, jak a hlavně kdy ukládat data čtená z DDR paměti na vstupu FPGA obvodu.

3.4.3.1 Problém zpoždění na vodičích

Problém se zpožděním čtených dat v závislosti na fyzické vzdálenosti obvodu DDR paměti a FPGA obvodu je znázorněn na obr. 24.



obr. 24 – Read data time shift

Význam jednotlivých signálu z obr. 24 je následující:

- signál FPGA_CK a FPGA_CK# je rozdílový hodinový signál tak, jak je generován v FPGA obvodu a na výstupu z něj.

- signál DDR_CK a DDR_CK# je hodinový signál, který je na vstupu DDR paměti. Je veden přímo z FPGA_CK(#), ale uplatní se na něm zpoždění na vodičích.
- DDR_DATA jsou čtená data na výstupu DDR paměti.
- DDR_DQS je synchronizační signál generovaný rovněž na výstupu DDR paměti.
- FPGA_DATA a FPGA_DQS jsou signály přímo vedeny ze signálu DDR_DATA a DDR_DQS, ale poté co se na nich uplatní zpoždění na vodičích.

Čas $t_{CK/2}$ je polovina periody hodinového signálu. Při maximální rychlosti použitých DDR pamětí (DDR333 = 167MHz) je $t_{CK/2}$ = 3ns.

Na obr. 24 je znázorněn posun o přibližně 1/5 z $t_{CK/2}$ pro jeden směr. To je přibližně 0.6ns zpoždění. Rychlost šíření signálu se vypočítá jako:

$$rychlost_signalu = \frac{rychlost_svetla}{\sqrt{k}} = \frac{300000000}{2} = 150000000 \frac{m}{s}$$

- k je koeficient odvozený od permitivity dielektrika v okolí vodiče. Toto je pro vodiče na deskách plošných spojů okolo 4.

Zpoždění o 1ns tedy vznikne při délce vodiče:

$$delka_vodice_{1ns} = rychlost_signalu \cdot t_{1ns} = 150000000 \cdot 1 \cdot 10^{-9} = 15cm$$

Vzdálenost obvodů z příkladu na obr. 24 je tedy pouhých:

$$delka_vodice_{0.6ns} = delka_vodice_{1ns} \cdot t_{zpozdeni} = 15 \cdot 0.6ns = 9cm.$$

Z uvedeného ilustračního příkladu se jeví jako nejlepší ukládat data do registru na hrany hodin s posunutím 180° a 0° v uvedeném pořadí. Pokud by ovšem vzdálenost vodičů obvodů byla 18cm, pak by patrně bylo nejlepší ukládat data na hrany hodin s fází 270° a 90°. Z uvedeného je doufám čtenáři patrné, že pro univerzální DDR řadič je minimálně potřeba mít možnost vhodně nastavit to, na kterou hranu hodin se budou data čtená z DDR paměti ukládat v FPGA vstupních bufferech. Možnosti tohoto nastavení jsou:

1. Parametr při syntéze DDR řadiče.
2. Dynamicky automatem, který bude součástí inicializačního automatu DDR řadiče.

První možnost bude jednodušší naimplementovat, avšak předpokládá přesnou znalost frekvence a vzdálenosti obvodů a bude třeba pečlivého výpočtu před syntézou.

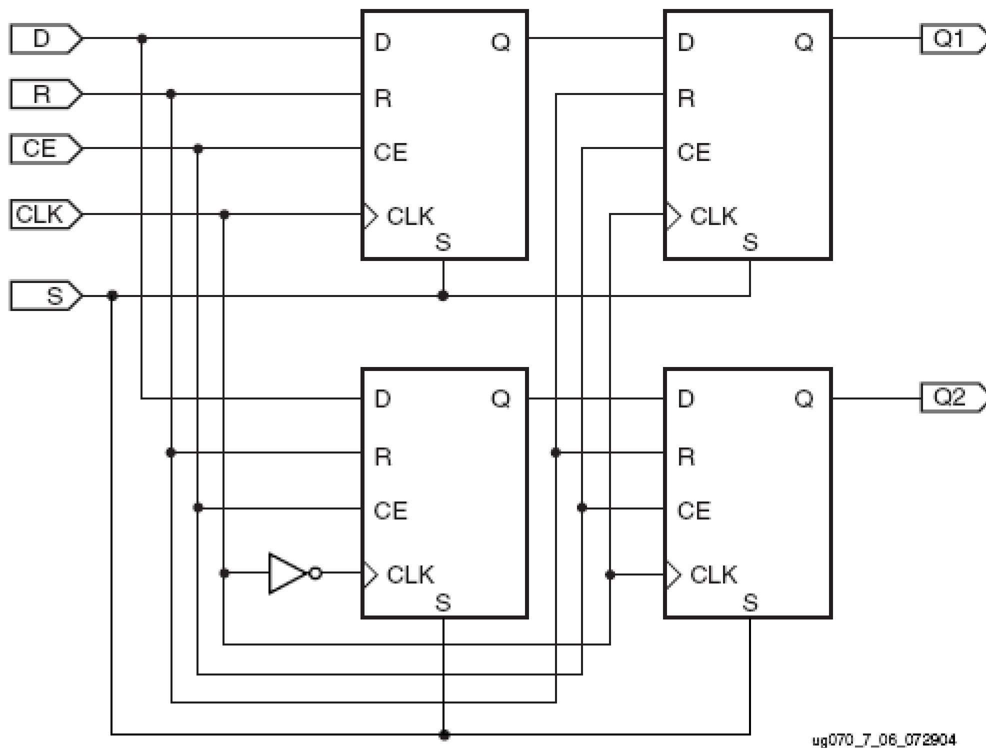
Oproti tomu druhá možnost je složitější na implementování, avšak modul DDR řadiče bude univerzálně použitelný pro jakýkoliv typ desky a nastavenou frekvenci.

3.4.3.2 Problém rychlé efektivní frekvence

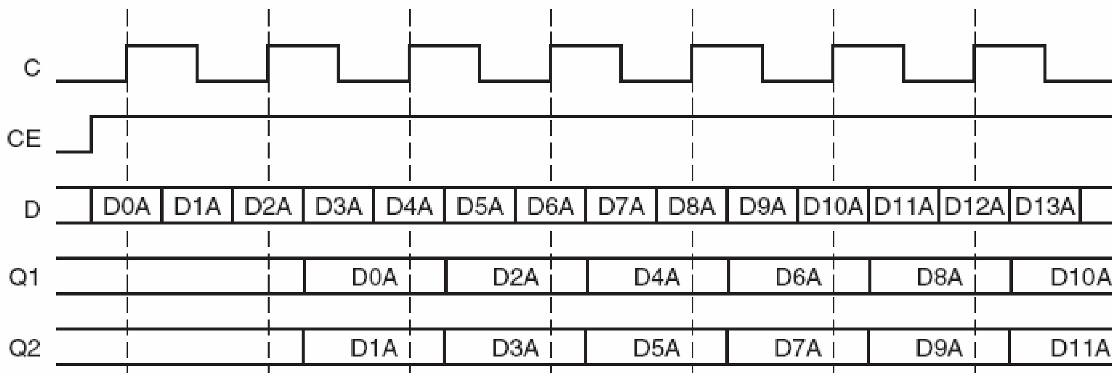
V předchozí kapitole jsme ukázali možnost řešení problému zpoždění dat na vodičích. V této kapitole ověříme, zda toto řešení bude dostatečné pro všechny kritéria obou obvodů. Kritéria myslím především požadavky na požadované časy setup a hold pro FPGA obvod a na jitter čtených dat z DDR paměti relativně k vstupnímu hodinovému signálu.

3.4.3.2.1 Strana FPGA obvodu

Studiem dokumentace k obvodu Virtex-4 (Virtex4 FPGA User Guide [11] a Virtex-4 FPGA Data Sheet: DC and Switching Characteristics [12]) jsem zjistil, že obvod přímo podporuje DDR (Double Data Rate) vstupní a výstupní registry. Pro čtení dat z DDR paměti nás bude zajímat v obvodu Virtex-4 blok IDDR (Input Double Data Rate register).

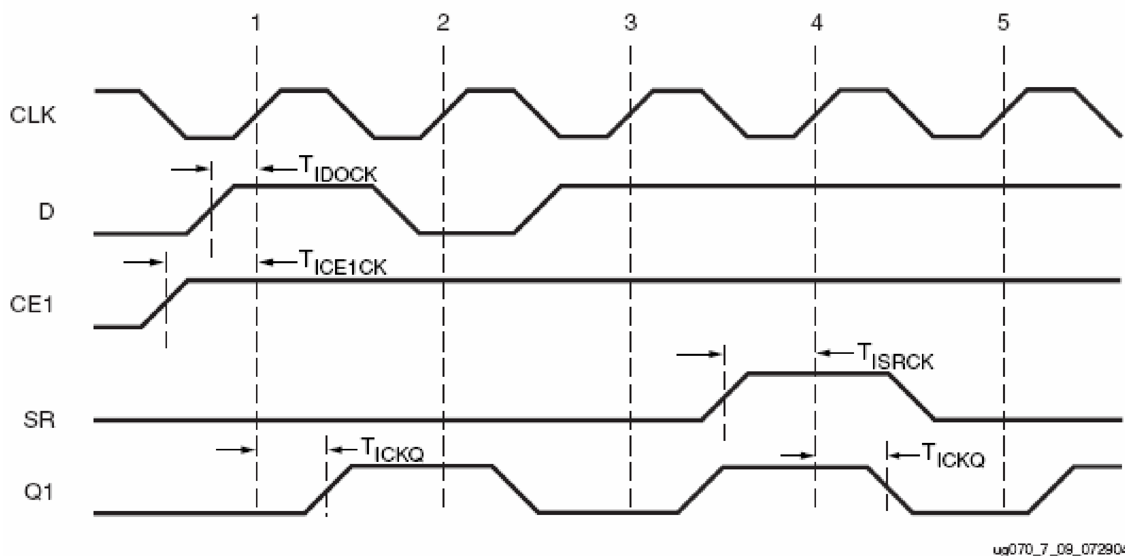


obr. 25 – Input DDR v SAME_EDGE_PIPELINED módu



obr. 26 – Input DDR Timing v SAME_EDGE_PIPELINED módu

Na obr. 25 je schéma IDDR bloku v SAME_EDGE_PIPELINED módu. První vrstva DFF obvodů (2 DFF obvody vlevo) slouží k uložení vstupního signálu z obou hran hodinového taktu CLK. Invertor před DFF obvodem vlevo dole slouží právě k ukládání vstupního signálu na spádovou hranu hodin CLK. Druhá vrstva DFF obvodů (2 DFF obvody vpravo na obr. 25) slouží k pipelinování uloženého signálu z první vrstvy DFF obvodů. Výstup z IDDR bloku se tedy objeví o jeden takt CLK signálu později, ale zato lze s daty pracovat jako s běžnými daty ukládanými pouze na náběžnou hranu hodin. Jeden vstup s dvojnásobnou efektivní frekvencí má pak dva výstupy s normální hodinovou frekvencí. Pipelinování a hodnoty obou výstupů jsou rozkresleny na obr. 26.

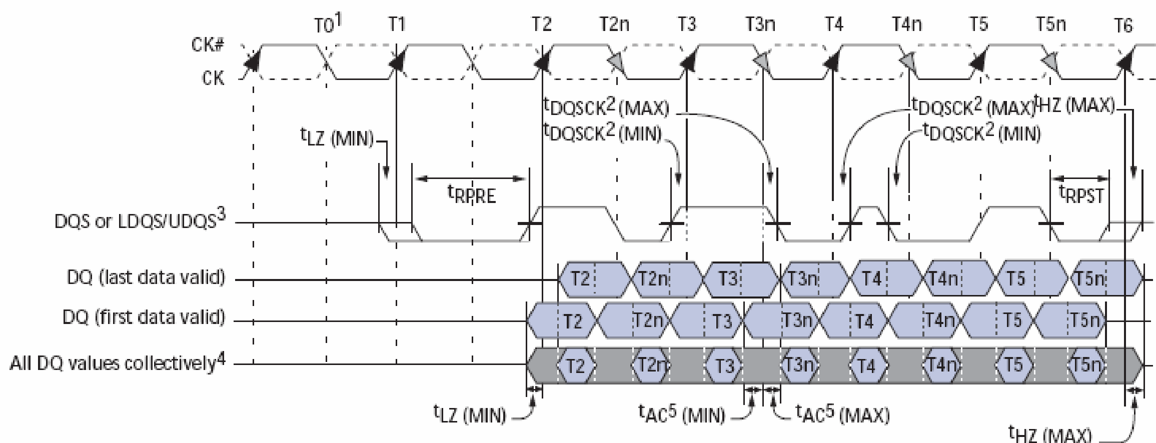


obr. 27 - ILOGIC input registr timing charakteristiky

Z katalogového listu obvodu Virtex-4 [12] jsem pak zjistil patřičné potřebné hodnoty času setup a hold viz obr. 27. Potřebný setup čas T_{IDOCK} je 0.34ns pro použitý obvod Virtex-4 s hodnotou speed grade „-10“. Potřebý hold čas je -0.10ns. Všimněme si, že v dokumentaci je uveden negativní hold čas. To by mělo znamenat, že je možné signál změnit dříve, než dojde k hraně hodin, na kterou má být tento signál uložen, a bude uložena správně stará hodnota signálu. Negativní hold time je vpodstatě maximální předstih, s jakým je možné přivést další hodnotu na vstup tak, aby byl původní vstup správně uložen.

3.4.3.2.2 Strana DDR SDRAM obvodu

DDR SDRAM obvod má určité tolerance na zpoždění, popřípadě předstih dat oproti hodinovému signálu. Možné varianty těchto posunutí signálů v čase jsou podrobně popsány v dokumentaci k DDR SDRAM paměti [10]. Pro nás nejdůležitější nepřesnosti signálů DQ (Data) a DQS (Data Strobe) uvádím na obr 28.



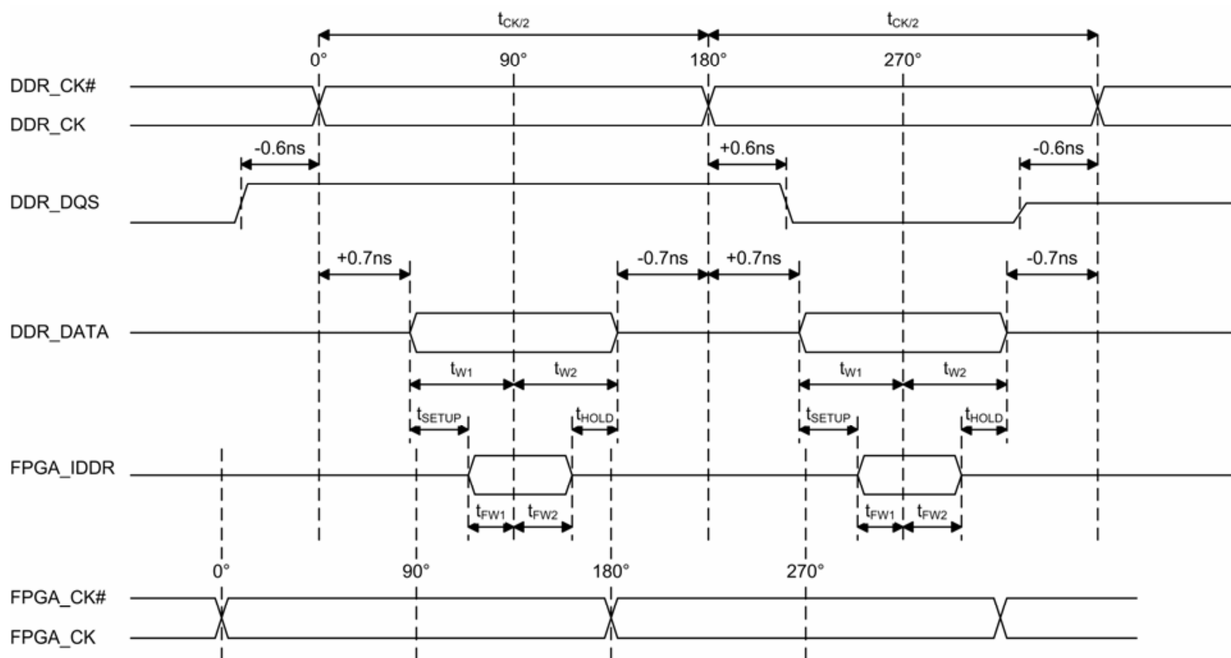
obr 28 – Data output timing

Hodnoty parametrů pro paměť MT46V16M16-6T jsou následující:

- t_{DQSCK} = <-0.6ns;+0,6ns>
- t_{AC} = <-0.7ns,+0,7ns>

Pokud je tedy DDR paměť použitá na maximální povolené frekvenci 167MHz, pak polovina periody hodinového signálu CK je 3ns. Potom okno platných dat je široké 1.6ns, tedy 0.8ns na každou stranu od signálu CK posunutého o 90°.

3.4.3.2.3 Kombinace požadavků FPGA a předpokladů DDR SDRAM



obr. 29 Combiovaný timing DDR SDRAM a Virtex-4.

Význam jednotlivých signálů z obr. 29 je následující:

- Signál DDR_CK a DDR_CK# je hodinový signál, který je na vstupu DDR paměti. Je veden přímo z FPGA_CK(#), ale uplatní se na něm zpoždění na vodičích.
- DDR_DATA jsou čtená data na výstupu DDR paměti.
- DDR_DQS je synchronizační signál generovaný rovněž na výstupu DDR paměti.
- FPGA_IDDR je signál platných dat po odečtené katalogové hodnoty časů SETUP a HOLD potřebných pro FPGA blok IDDR. Uvnitř znázorněných platných dat je možné kdykoliv data bezpečně uložit v IDDR bloku. Tato data jsou na vstupu FPGA v libovolném posunutí oproti hodinovému signálu FPGA_CK(#) a toto posunutí závisí především na fyzické vzdálenosti obou obvodů.
- Signál FPGA_CK a FPGA_CK# je rozdílový hodinový signál, tak jak je generován v FPGA obvodu a na výstupu z něj. Na obrázku je znázorněn příklad posunutí oproti signálu FPGA_IDDR.

Hodnota t_{SETUP} je 0.34ns a hodnota t_{HOLD} je -0.10ns, jak je uvedeno v kapitole 3.4.3.2.1. Tedy okno bezpečně připravených dat je

$$t_{OK} = t_{FW1} + t_{FW2} = t_{CK/2} - 0.7ns \cdot 2 - t_{SETUP} - t_{HOLD}$$

$$t_{OK} = 3ns - 0.7ns \cdot 2 - 0.34ns + 0.1ns = 1.36ns$$

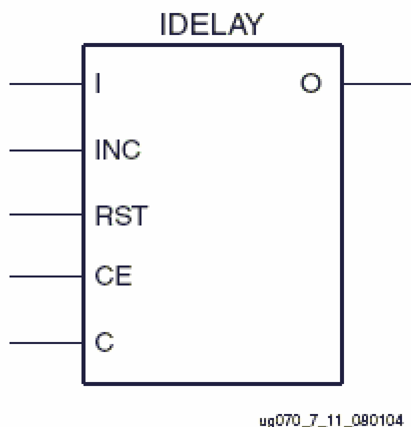
Okno 1.36ns je ale menší než ¼ periody hodinového signálu (6ns*1/4 je 1.5ns). To znamená, že pro obecný případ nebude stačit pouze zvolit hranu, na kterou se data budou ukládat v FPGA IDDR registru, jak jsme předpokládali v kapitole 3.4.3.1. Na obr. 29 je právě znázorněno posunutí, kde nelze vybrat hranu, na kterou by se data čtená z DDR paměti mohla v FPGA IDDR registru správně uložit. FPGA_CK posunuto o 90° je příliš brzy a FPGA_CK posunuto o 180° je příliš pozdě. Navíc by bylo dobré mít mechanismus nastavit čtení co nejvíce doprostřed platných dat. Proto postup výběru správné hrany navrhnutý pro řešení posunutí dat vlivem délky vodičů v kapitole 3.4.3.1 nebude v obecném případě použitelný.

3.4.3.2.4 Jemné nastavení posunutí dat

Jak vyplývá z předchozí kapitoly, je potřeba mít možnost nastavit okamžik ukládání dat do IDDR paměti jemněji než po 90° hodinové frekvence. Pro to budeme buď potřebovat speciálně posunuté hodiny, nebo speciálně posunutá vstupní data.

Řešení s posunutými hodinami je nešikovné z důvodu zavedení další časové domény a především nutnosti synchronizovat načtená data s hlavními hodinami systému. Nicméně tento postup je teoretický možný využitím jemně nastavitelného fázového posunu v DCM modulu FPGA obvodu. Výhodou tohoto řešení je, že je možné realizovat v původní platformě se Spartanem 3E. DCM se dá nastavit do VARIABLE_CENTER módu a lze dynamicky nastavovat za běhu systému. Je možný krok, který je 1/256 periody hodinové frekvence přivedené na vstup DCM bloku.

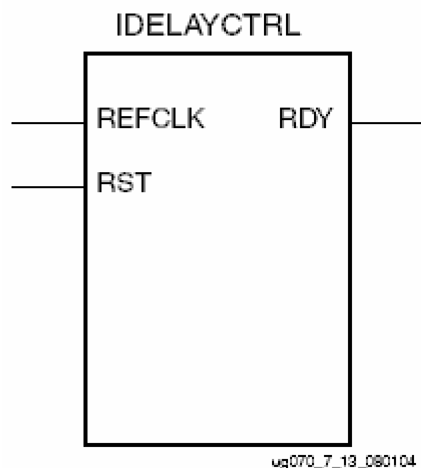
Jednoduší řešení je zpozdit vstupní data před přivedením do bloku IDDR tak, aby se dala zvolit hrana hodin, která bude přesně ve středu okna platných dat. Pro tento postup se výborně hodí „Input Delay Element" (IDELAY), který je součástí FPGA obvodu Virtex-4. Tento blok dokáže vložit 0 až 63 zpožďovacích hradel, které mají garantovanou dobu zpoždění. Nastavení lze provézt buď při syntéze, nebo dynamicky za běhu FPGA obvodu. Schéma bloku IDELAY je znázorněno na obr. 30.



obr. 30 – IDELAY Primitiv.

Na vstup I se přivedou vstupní data z IOB a na výstupu O jsou tato data zpožděna o nastavenou hodnotu. RST je reset vstup obvodu. Signály C, CE a INC jsou použity pouze při dynamickém nastavování zpoždění. Vstup C je pro hodinový signál. Vstup CE je povolení změny. Vstup INC rozhoduje, zda se zpoždění prodlouží nebo zkrátí. Pro podrobné informace o bloku IDELAY nechť se laskavý čtenář podívá do dokumentace obvodu Virtex-4 [11].

Za důležitou zmínku však stojí, že při použití bloku IDELAY je nutno použít blok IDELAYCTRL. Tento blok se stará právě o garanci přesného zpoždění zpožďovacích hradel bloku IDELAY. Zajišťuje, že hodnota je stabilní a v rozmezí, které udává katalogový list. Zároveň zajišťuje, že se toto zpoždění nemění za běhu vlivem změny teploty nebo jiných vlivů. Schéma tohoto bloku je znázorněno na obr. 31. Signál RST slouží k resetování bloku IDELAYCTRL. Signál REFCLK je referenční hodinový signál a má mít frekvencí 200MHz. Signál RDY je výstupní signál, který určuje, zda bloky IDELAY jsou již správně zkalibrované. Podrobnosti o tomto bloku nechť si laskavý čtenář dohledá opět v dokumentaci obvodu Virtex-4 [11].



obr. 31 – IDELAYCTRL Primitiv.

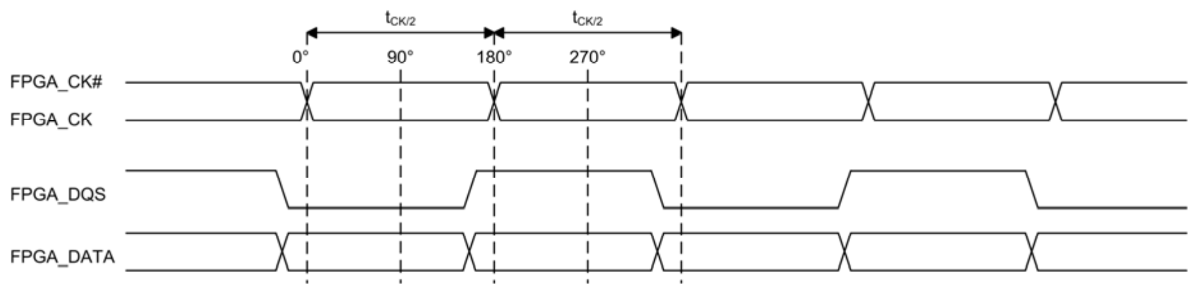
V katalogovém listu Virtex-4 FPGA Data Sheet: DC and Switching Characteristics [12] je přesně uvedena hodnoto zpoždění jednoho zpožďovacího hradla jako 75ps. Odchylka zpožďovacího hradla může být až ±7%. To je relativně hodně pro statické nastavení před syntézou, avšak při dynamickém nastavování nás tento fakt netrápí. Maximální nastavitelné zpoždění je 63*75ps = 4,725 ps. To nestačí pro libovolné zpoždění v celé periodě hodinového signálu DDR pamětí, která je 6ns. Nicméně to vyhovuje pro libovolné posunutí v ½ periody hodinového signálu. Je tedy možné přesně nastavit střed okna platných dat na jednu z hran hodinového signálu posunutých o 0° nebo 180° a potom určit, která hrana je platná pro první data a která pro druhá data.

Nyní jsme teoreticky zjistili použitelný způsob pro další implementaci DDR řadiče. Zbývá přijít na to, jak nastavit zpoždění dat. Jak jsem již uvedl, hledáme univerzální řešení, které nebude vyžadovat přesně spočítat parametr před syntézou, ale které DDR řadič dynamicky nastaví při inicializaci.

3.4.3.2.5 Zjištění středu okna platných dat

Paměť lze inicializovat před nastavením zpoždění IDDR bloku pro čtená data. Je to především proto, že pro inicializaci paměti nepotřebujeme žádná data číst. Potřebujeme pouze zapisovat příkazy pomocí řídících a adresních vodičů. Příkazy se navíc ukládají v DDR paměti pouze na náběžnou hranu hodinového signálu, a tedy na relativně pomalé frekvenci. Jak řídící příkazy, tak hodiny a data na adresních vodičích vycházejí při správném návrhu DDR řadiče v jednom momentu z FPGA obvodu. Podotýkám, že zde je velmi důležité registrovat řídící a adresní signály tak, aby v jejich cestě na výstup FPGA obvodu nebyla další logika, která by způsobila časové posunutí řídících signálů proti sobě nebo proti hodinám. Další nutnou podmínku je, že všechny vodiče mezi DDR pamětí a FPGA obvodem mají stejnou délku, řídící data se pak nebudou časově rozcházet proti sobě ani proti hodinovému signálu. Ukládání dat a příkazů je proto při dodržení výše zmíněných podmínek triviální záležitostí a není potřeba DDR řadič jakkoliv nastavovat pro plnění těchto operací. Fyzická vzdálenost obvodů je pro ukládání dat a příkazů irelevantní. Je však dobré se snažit tuto vzdálenost minimalizovat kvůli jiným elektromagnetickým jevům (EMI, kapacita vodičů, při velké délce vodičů odrazy a nutnost přizpůsobování).

Umíme tedy DDR paměť inicializovat. Nyní přistoupíme k hledání posunutí čtených dat.



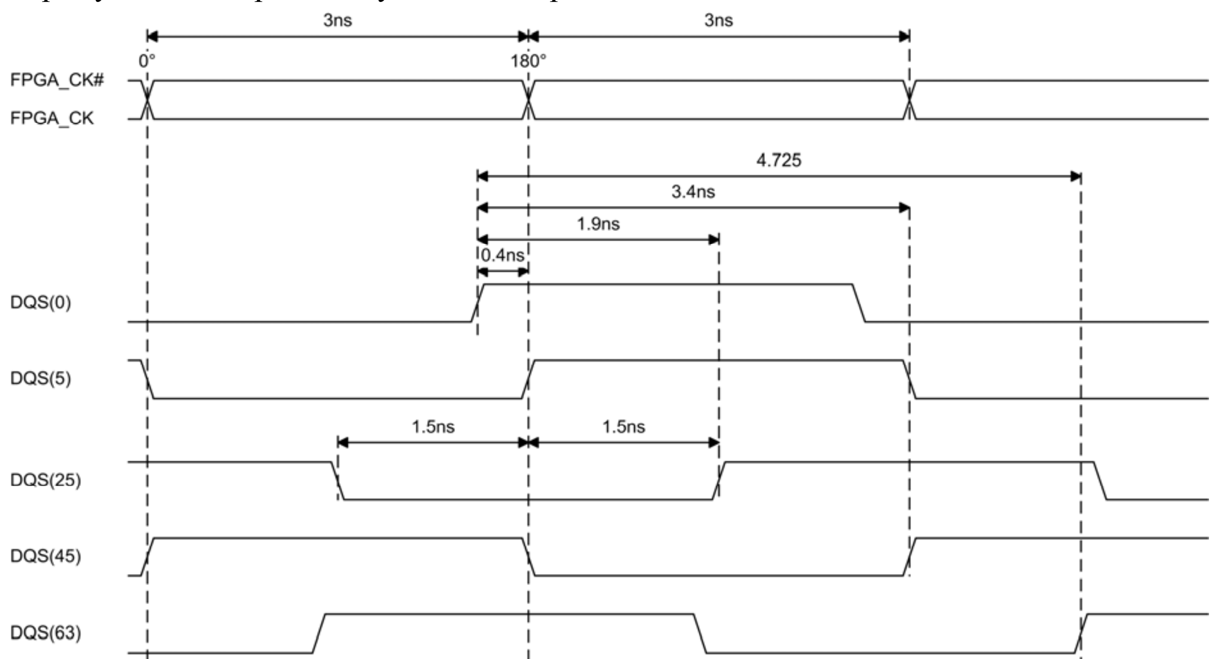
obr. 32 – DDR SDRAM navazující čtení

Pro hledání přesného posunutí využijeme signálu DQS, jehož hranu paměť generuje společně s hranou dat. Ideální pozice pro ukládání dat pak bude přesně mezi hranami signálu DQS. Celá situace průběžného čtení dat je znázorněna na obr. 32.

- FPGA_CK(#) je hodinový signál FPGA obvodu
- FPGA_DQS je DQS signál generovaný DDR pamětí zakreslený v okamžiku vstupu do FPGA obvodu. Kvůli délce vodičů může být v libovolné pozici k signálům FPGA_CK(#)
- FPGA_DATA je signál DQ (DATA), generovaný DDR pamětí a uvažovaný v okamžiku vstupu do FPGA obvodu.

Při zjišťování hran DQS signálu nás hodnota dat nezajímá. Vyčítáme z paměti neinicializovaná data jen proto, ať nám paměť generuje DQS signál.

Vezmeme-li v úvahu to, že hodnoty t_{SETUP} a t_{HOLD} vstupního bloku IDDR registru nejsou symetrické, ale hodnota t_{SETUP} je mírně větší, pak ideální pozice pro ukládání dat je mírně za středem signálu DQS. Bohužel hodnoty t_{SETUP} a t_{HOLD} pro blok IDDR jsou různé pro různé nastavení bloku IDELAY a není tedy obecně možné je určit. Rozvážení hodnot t_{SETUP} a t_{HOLD} je však velmi malé. Například při nastavení IDELAY_TAP=0: t_{SETUP}=1.09ns a t_{HOLD}=-0.63ns. Ideální posunutí středu platného signálu je tedy o $(-t_{SETUP}+t_{HOLD})/2$ a je tedy -0.86ns. Při hodnotě zpoždění jednoho zpoždovacího hradla 75ps bychom měli přidat k výsledku -11 zpožďovacích hradel.



obr. 33 – Hledání DQS hran

Na obr. 33 je znázorněn proces hledání hran DQS signálu. Tento proces si nyní podrobně popíšeme. Signál DQS(x) z obr. 33 je signál DQS, který prošel blokem IDELAY, hodnota “x“ reprezentuje počet použitých zpožďovacích elementů. Budeme

muset sestrojit automat, který bude načítat DQS signál. Já jsem se rozhodl načítat DQS signál při fázi 180°. Automat vždy zjistí hodnotu DQS, přidá zpoždění a znova zjistí hodnotu DQS signálu. V momentě, kdy automat detekuje změnu signálu DQS od poslední hodnoty, znamená to, že jsme se právě posunuli přes hranu DQS signálu. Uložíme hodnotu počtu použitých zpožďovacích elementů. V příkladu z obr. 33 by to byla hodnota 5. Pokračujeme dalším přidáváním zpoždění. V momentě, kdy dojde k další změně signálu DQS, opět uložíme nastavené zpoždění. V příkladu z obr. 33 by to byla hodnota 45. Ideální hodnota zpoždění tedy bude:

$X=(X_2-X_1)/2+X_1-11$

- X1 je uložený počet zpožďovacích elementů při první detekci hrany (=5)
- X2 je počet zpožďovacích elementů při druhé detekci hrany (=45)
- -11 je z důvodů rozvážení t_{SETUP} a t_{HOLD} IDDR bolu. Tato hodnota je přibližná

Optimální posunutí X tedy bude (45-5)/2+5-11 = (25-11) = 14. Optimální posunutí by tedy bylo z obr. 33 DQS(25), avšak od něj je ještě odečtena hodnota 11 vlivem parametrů IDDR bloku.

Uvažujme však ještě případ, kdy se v celém rozsahu nepodaří najít dvě změny hrany. Představme si, že na příkladu v obr. 33 by hrana signálu DQS(0) byla těsně za FPGA_CK# hranou. Pak bychom patrně v celém rozsahu nastavitelného zpoždění (0 až 63) nalezli pouze jednu hranu, patrně někde při hodnotě zpoždění 40. V případě, že detekujeme pouze jednu hranu, nastavíme hodnotu zpoždění podle následujícího pravidla:

Jestliže hodnota zpoždění při detekci hrany je menší nebo rovna 64/2 (polovina rozsahu), pak nastavíme pak nastavíme $X_2=64$ a použijeme stejný vzorec pro výpočet ideální hodnoty zpoždění $X=(X_2-X_1)/2+X_1-11$.

Jestli-že hodnota zpoždění při detekci hrany je větší než 64/2, pak použijeme vzorec pro výpočet ideální hodnoty $X=X_1/2-11$.

V našem uvažovaném případě se tato situace stát sice nemůže, ale uvažujme, že nedetekujeme žádnou hranu. To se může stát, používáme-li DDR paměti na nižší frekvenci, třeba 100MHz, pak je celý rozsah nastavení zpoždění (4.725ns) menší než půlperioda hodinové frekvence (5ns). V takovém případě nastavíme ideální hodnotu zpoždění jako X=64/2-11.

Máme nyní navržen automat pro detekci hran. Zamysleme se ještě nad samotným procesem detekce hrany. Automat hranu zjišťuje z dat uložených v jednom registru. Pak posune zpoždění o pouhých 75ps. A znovu uloží signál DQS do registru. 75ps je velice krátký časový úsek a vlivem jitteru DQS signálu nebo rychlejší/pomalejší reakcí vstupního registru může náš automat omylem detekovat 2 nebo více hran za sebou. Například můžeme dostat sekvenci 1111010000. Tomu je potřeba předejít. První způsob, jak tento problém vyřešit, je použít více registrů přes více taktů. Například 4 registry a každý ukládat v následujícím taktu. Automat by pak akceptoval hranu až tehdy, pokud je ve všech registrech stejná hodnota a jsme tedy těsně za hranou. Například bychom tedy dostali sekvenci:

1111 při zpoždění 4
1111 při zpoždění 5
0110 při zpoždění 6
0000 při zpoždění 7
0000 při zpoždění 8

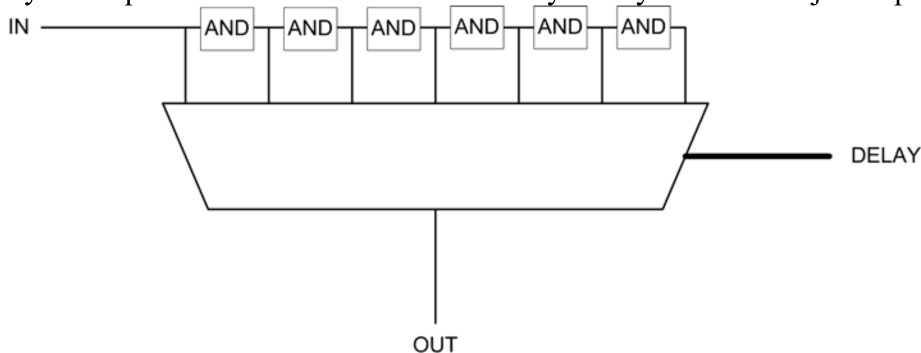
Hranu by náš automat uznal při zpoždění 7 a provedl by posunutí zpoždění na hodnotu 8. Při zpoždění 8 je pak velmi nepravděpodobné, že by se objevila vlivem jitteru nebo rychlejší reakcí DFF opět hodnota 1111. I přesto tento případ nelze vyloučit

a je vhodné více registrů zkombinovat ještě z inhibičním čítačem. Tento čítač se nastaví po detekci hrany a nedovolí další detekci hrany po dobu několika smyček přidání hodnoty zpoždění. Vzhledem k tomu, že hranu bychom měli detekovat co 3ns, pak musíme čítač pro zakázání detekce hrany nastavit na hodnotu menší než 3ns. Jitter DQS signálu by měl být menší než 1.2ns podle dokumentace k DDR paměti. Rozsah hodnot t_{HOLD} a t_{SETUP} DFF obvodu Virtex-4 by měla být menší než 0.46ns. Délka zákazu hrany by měla být větší než větší z těchto dvou údajů. Proto by mělo být bezpečné zvolit délku zákazu detekce hrany 1.5ns. Při kroku 75ps můžeme nastavit inhibiční čítač na hodnotu 1.5ns/0.075ns=20.

Nyní jsme si ukázali, jakým způsobem je možné změřit posunutí signálu a vypočítat ideální hodnotu dalšího zpoždění. Toto zpoždění pak nastavíme pro blok IDDR registrů, které ukládají načítaná data z DDR pamětí. Tím bychom měli zajistit, že čtená data z DDR paměti budeme ukládat právě ve středu platných dat.

Nyní zbývá poslední věc, a to určit s kolika taktovým zpoždění DDR paměť data vystavuje. Při nastavení parametru CL=2 by zpoždění mělo být 2 taky (viz kapitolu 3.4.1.3). K tomuto zpoždění musíme přičíst zpoždění na vodičích a zpoždění IDELAY bloku. Pokud je paměť dostatečně blízko (méně než 22.5cm), pak bychom za 2.5 taktu měli obdržet první data a pak po půl taktech data následující (v závislosti na nastavené délce bloku dat). Pro obecné řešení je však nutné i tuto dobu změřit. Toto bude snadné provést automatem, který uloží nějaký definovaný blok dat. Například 0x1122,0xAABB (pro blok délky 2). Poté se automat data ze stejné adresy pokusí vyčíst. Data bude ukládat každý takt po příkazu čtení. Zároveň bude počítat kolik taktů uplynulo. Při shodě načtených dat s ukládanými uloží hodnotu čítače taktů do konfiguračního registru DDR řadiče. Každé další čtení pak bude předpokládat platná data po stejném počtu taktů od příkazu „read“.

Nyní bych rád vzpomněl platformu Spartan-3E. Spartan-3E nemá IDELAY blok. Výše uvedený postup by však mohl být realizovatelný tak, že místo IDELAY bloku bychom použili vlastní IDELAY blok vyrobený ve VHDL jako specifické zapojení.



obr. 34 – IDELAY blok substituce

Na obr. 34 je takto vyrobený IDELAY blok. Jde o sériově zapojenou řadu hradel, například AND a z této jsou paralelně vyvedeny po celé délce výstupy. Je zřejmé, že signál, který projde více hradly, bude více zpožděn než signál, který projde méně hradly. U tohoto postupu je však potřeba dát pozor na to, jak je kód skutečně sesyntetizován. Zároveň je potřeba, aby MUX měl stejné zpoždění pro všechny vstupy. V neposlední řadě ve výčtu problémů je fakt, že u obecných hradel není garantována rychlost odezvy, a tato se navíc může v čase měnit, popřípadě být jiná pro přechody z 0 do 1 než z 1 do 0. V případě platformy Spartan-3E bych tento postup nedoporučoval a pokusil bych se lépe analyzovat variantu vytvoření další časové domény s patřičným posunutím fáze pomocí DCM bloku.

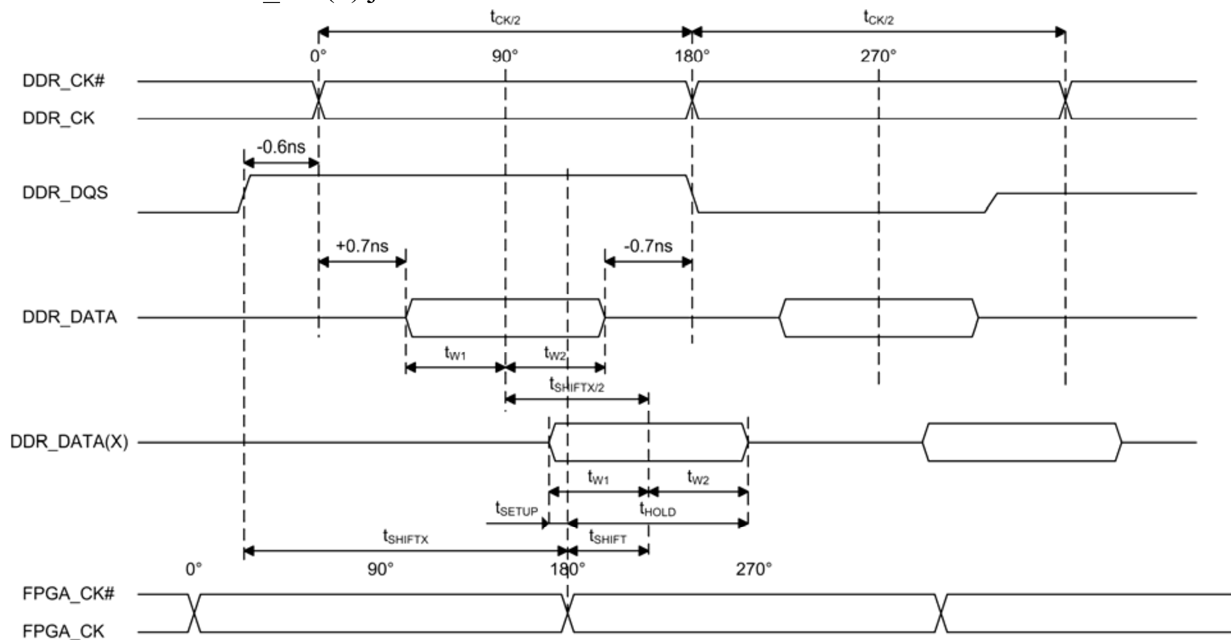
3.4.3.2.6 Ověření časování

Ověřit 100% časování je v podstatě nemožné. Je to především proto, že v řešení obecného DDR řadiče je příliš mnoho faktorů a tyto faktory mají příliš mnoho možných hodnot nebo dokonce nekonečno možných hodnot v případě spojitých veličin (například vzdálenost obvodů). Jiné parametry závisí na syntéze a nastavení IDELAY bloku (například t_{SETUP} a t_{HOLD} bloku IDDR je ovlivněn nastavením bloku IDELAY, které je provedeno dynamicky za chodu – nemám proto teoreticky možnost tento parametr ověřit). Pokusím se sestavit nejhorší možný příklad pro odhad maximální použitelné frekvence. Tento nejhorší případ je však zároveň velmi nepravděpodobný.

Základním předpokladem pro možnost ukládat data z DDR paměti je to, aby velikost okna platných dat byla větší než rozsah hodnot t_{SETUP} a t_{HOLD} IDDR bloku. Pro DDR333 je okno platných dat v nejhorším případě široké 1.6ns. Hodnoty t_{SETUP}+t_{HOLD} bloku IDDR při použití IDELAY bloku je 1.09ns-0.63ns=0.46ns (viz katalogový list obvodu Virtex-4 [12]). Tento předpoklad je tedy splněn.

Nyní vezmeme v úvahu funkci automatu a také ostatní nepřesnosti DDR pamětí. Na obr. 35 jsem rozkreslil případ, o kterém se domnívám, že je nejhorší možný.

- Signál DDR_CK(#) je hodinový signál zakreslen se zpožděním, jako by se vrátil na vstup FPGA obvodu (aby mohl být zarovnaný se signály DDR_DQS a DDR_DATA)
- DDR_DQS je DQS signál na vstupu FPGA obvodu.
- DDR_DATA je DQ (DATA) signál na vstupu FPGA obvodu.
- DDR_DATA(X) představuje signál posunutý blokem IDELAY na vstupu IDDR registru.
- FPGA_CK(#) je clock uvnitř FPGA obvodu



obr. 35 – DDR kontroler – případ nejhoršího timing

Signál DDR_DQS generovaný DDR pamětí je maximálně vychýlen na jednu stranu. Při jeho detekci se navíc vlivem nevhodné vzdálenosti obvodů podaří zachytit pouze jednu hranu signálu DDR_DQS. Zvolený střed proto nebude ideální a bude vypočítán podle popsané funkce inicializačního automatu v kapitole 3.4.3.2.5. Nejhorší případ nastane, když zpoždění na vodičích bude co nejblíže ¼ periody, ale tak aby se detekovala pouze jedna hrana a zároveň aby detekce této první hrany proběhla při zpoždění větším než polovina rozsahu IDELAY bloku.

Předpokládejme:

- DDR333 a tedy $t_{CK/2}$=3ns.

- Minimální větší polovina rozsahu IDELAY bloku je t_{SHIFTX}=(64/2+1)*0.075ns=2.475ns. Protože $t_{CK/2}$+0.6ns-t_{SHIFTX}<$t_{CK/2}$/2 (jinak řečeno pokud automat detekuje hranu při minimální větší polovině rozsahu IDELAY bloku ještě pořád to neznamená, že posunutí je větší než 1/4t_{CK}) a protože chceme posunutí co nejblíže 1/4t_{CK} a další zvětšování hodnoty t_{SHIFTX} má za následek vzdálení od tohoto posunutí, je nejhorší možná hodnota t_{SHIFTX} právě minimální větší polovina rozsahu IDELAY bloku (2.475ns).
- $t_{SHIFTX/2}$ se pak vypočítá podle vzorce popsaného v kapitole 3.4.3.2.5. $t_{SHIFTX/2}$=(X_1/2-11)*$t_{ELEMENTU}$==(33/2-11)*0.075ns=0.375ns
- t_{SHIFT} je posun středu dat od hrany, na kterou jsou ukládána, a v našem případě je t_{SHIFT}=0.6ns+$t_{CK/2}$/2+$t_{SHIFTX/2}$-t_{SHIFTX} = 0.6ns+1.5ns+0.375ns-2.475ns = 0ns. V našem nejhorším případě je střed dat paradoxně přesně zarovnán s hranou hodin. Uvědomme si však, že toto není ideální posunutí, protože kvůli relativně velké hodnotě t_{SETUP} a záporné hodnotě t_{HOLD} bloku IDDR je potřeba data posunout relativně výrazně před hranu hodinového signálu.
- t_{W1} = $t_{CK/2}$/2-0.7 = 0.8ns
- t_{SETUP} = t_{W1}-t_{SHIFT}=0.8ns-0ns=0.8ns.

V katalogovém listu IDDR obvodu je však napsáno, že t_{SETUP} musí být alespoň 1.09ns při použití IDELAY bloku s nastaveným zpožděním 0. Bohužel v katalogovém listu nejsou hodnoty pro ostatní hodnoty zpoždění, ale předpokládám, že tato hodnota nebude menší. Proto v nejhorším případě nesplňujeme potřebný čas t_{SETUP} přibližně o 0.29ns. Jak už jsem uvedl tato situace je velmi nepravděpodobná. Může ji vyřešit restartování obvodu, kdy nedojde k maximální výchylce DQS signálu.

Zmenšíme tedy frekvenci na DDR200 (tedy 100MHz) a zkusme znova vypočítat časování pro nejhorší možný případ.

Předpokládejme:

- DDR200 a tedy $t_{CK/2}$=5ns.
- Podmínka, že se bude detekovat jen jedna hrana, je splněna vždy při délce půlperiody větší než je rozsah IDELAY bloku (4.725ns). Pro zpoždění na vodičích rovném ¼ periody bude: t_{SHIFTX}=0.6ns+$t_{CK/2}$/2=0.6ns+2.5ns=3.1ns.
- Detekce hrany by pak proběhla při zpoždění X_1=t_{SHIFTX}/0.075=42, což splňuje podmínku, aby detekce hrany nastala ve více než polovině rozsahu zpoždění IDELAY bloku.
- $t_{SHIFTX/2}$ se pak vypočítá podle vzorce popsaného v kapitole 3.4.3.2.5. $t_{SHIFTX/2}$=(X_1/2-11)*$t_{ELEMENTU}$==(42/2-11)*0.075ns=0.75ns.
- t_{SHIFT} je posun středu dat od hrany, na kterou jsou ukládána, a v našem případě je t_{SHIFT}=0.6ns+$t_{CK/2}$/2+$t_{SHIFTX/2}$-t_{SHIFTX} = 0.6ns+2.5ns+0.75ns-3.1ns = 0.75ns.
- t_{W1} = $t_{CK/2}$/2-0.7 = 1.8ns
- t_{SETUP} = t_{W1}-t_{SHIFT}=1.8ns-0.75ns=1.05ns.

V katalogovém listu IDDR obvodu je však napsáno, že t_{SETUP} musí být alespoň 1.09ns při použití IDELAY bloku s nastaveným zpožděním 0. Bohužel v katalogovém listu nejsou hodnoty pro ostatní hodnoty zpoždění, ale předpokládám, že tato hodnota bude podobná. Proto v nejhorším případě nesplňujeme potřebný čas t_{SETUP} přibližně o 0.04ns. V modelovém nejhorším možném případě je takto navržený DDR řadič schopen fungovat na frekvenci 95MHz.

V našem řešení zkusíme použít DDR paměť na 100MHz a budeme předpokládat, že nedojde k nejhorší modelové situaci. V případě problémů při návrhu snížíme frekvenci DDR řadiče.

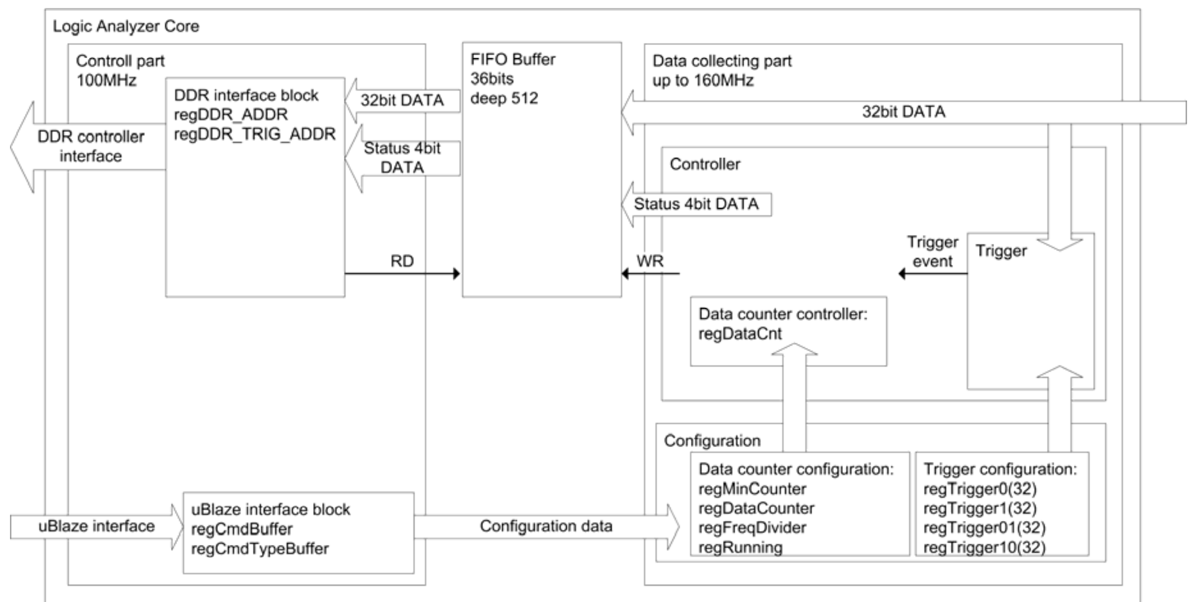
3.4.4 Zhodnocení DDR kontroleru

Z uvedených kapitol je zřejmé, že automat realizující DDR kontrolér bude velmi obtížné navrhnout nejen proto, že je to komplexní automat realizující mnoho funkcí. Ale především proto, že je zároveň kritický na splnění velmi přísných časování. Navíc od něj požadujeme vysokou propustnost dat.

Ještě bych rád uvedl jeden zajímavý nápad jak realizovat nastavení posunutí čtených data. Místo toho, aby automat zkoumal signál DQS, zkoumal by přímo signál DQ (DATA). Tímto postupem bychom posun mohli vyladit přímo na data a vyhnuli bychom se tak jitteru DQS signálu. Problém však je, že podle katalogového listu DDR SDRAM [10] platná data mohou mít ještě é 0,1ns větší posun, než je jitter DQS signálu. V nejhorší situaci by pak mohl nastat případ, kdy bychom šum načetli jako platná data a podle toho nastavili posunutí. Popřípadě by DDR paměť měla pro některé data platné okno širší než pro jiné. Toto je sice velmi nepravděpodobný případ, nicméně obdobně nepravděpodobný jakože inicializace automatu proběhne při maximální výchylce signálu DQS a detekci pouze jedné hrany. Případ falešně rozpoznaných platných dat bychom mohli řešit složitějším inicializačním automatem, který by testoval více hodnot dat a snižoval by tak šanci, že jsou platná data načtena mimo skutečné okno platných dat. Myslím, že nastavování přímo z dat by bylo obecně kvalitnější, nicméně velmi nepravděpodobná nejhorší možná situace by vyžadovala nižší maximální frekvenci. Proto budeme nastavení posunutí dat realizovat zkoumáním signálu DQS.

3.5 Blok jádra logického analyzátoru

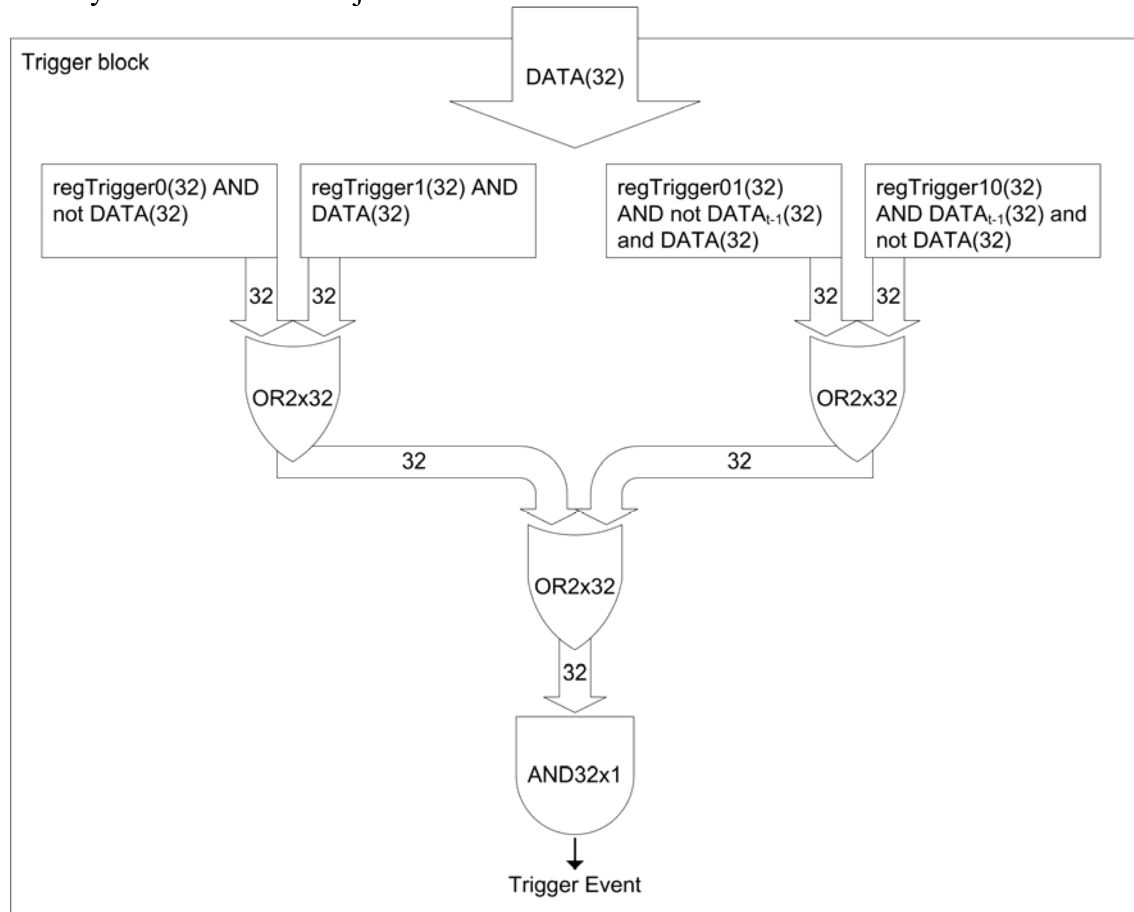
V této kapitole se zaměřím na jádro logického analyzátoru. Toto jádro bude realizovat základní funkci zařízení. Paradoxně při minimálních požadavcích na rychlost (50MHz) a šířku měřícího portu (8) bude tento blok daleko jednodušší pro návrh než blok DDR řadiče. Analýzou DDR řadiče jsme zjistili, že bude schopen běžet na efektivní frekvenci 200MHz a šířka dat je 32 bitů. Jak jsme uvedli v kapitole 3.3.2 není možné využít 100% teoretické maximální rychlosti DDR paměti. Je potřeba určitou část věnovat na obnovovaní DDR paměti a také příkazy mají určitou latenci. Odhadem je tedy možné využít 80% maximální rychlosti. To by znamenalo, že při použití dostatečně velkého FIFO bufferu bychom mohli ukládat 32 bitů dat rychlostí 160MHz. Protože procesor a periferie běží na frekvenci 100MHz, bude nutné jádro logického analyzátoru rozdělit na dvě části. První část bude komunikovat s procesorem a DDR řadičem a bude běžet na frekvenci 100MHz. Druhá část bude sbírat data a ukládat je do FIFO bufferu a bude běžet obecně na libovolné frekvenci až na maximálních 160MHz. FIFO buffer bude tedy zároveň zajišťovat bezpečné předání dat mezi dvěma časovými doménami. Zároveň je potřeba navrhnout blok triggeru, který bude určovat, zda došlo ke spouštěcí podmínce. Tento jsem se rozhodl realizovat rovněž v druhé časti jádra logického analyzátoru. Je možné jej však navrhnout i v první části logického analyzátoru. Zjednodušené schéma bloku jádra logického analyzátoru je znázorněno na obr. 36.



obr. 36 – Logický analyzér – schéma jádra

3.5.1 Trigger

Trigger jsem se rozhodl realizovat pro jakoukoliv kombinaci úrovní a hran všech měřených kanálů. Schéma je znázorněno na obr. 37.



obr. 37 – Trigger blok

Trigger je v zásadě pouze velmi jednoduchý blok logických hradel. Jeho funkce je porovnávat vstupní 32 bitový signál DATA a 32 bitový signál $DATA_{t-1}$ s 32 bitovými

konfiguračními registry regTrigger0, regTrigger1, regTrigger01 a regTrigger10. Na základě jejich hodnot pak nastaví bit „Trigger Event“ do hodnoty 1 v případě, že nastala spouštěcí událost a vice versa. Význam konfiguračních registrů je následný:

- regTrigger0 – obsahuje logickou 1 na pozicích, kde požadujeme, aby DATA byla 0 pro splnění spouštěcí podmínky.
- regTrigger1 – obsahuje logickou 1 na pozicích, kde požadujeme, aby DATA byla 1 pro splnění spouštěcí podmínky.
- regTrigger01 - obsahuje logickou 1 na pozicích, kde požadujeme, aby $DATA_{t-1}$ byla 0 a DATA byla 1 pro splnění spouštěcí podmínky.
- regTrigger10 - obsahuje logickou 1 na pozicích, kde požadujeme, aby $DATA_{t-1}$ byla 1 a DATA byla 0 pro splnění spouštěcí podmínky.
- $DATA_{t-1}$ představuje DATA uložená do registru v předchozím taktu
- DATA představuje DATA uložená do registru v tomto taktu

Jediný háček v tomto bloku je, že trigger by měl být schopný pracovat na relativně vysokých frekvencích (až 160MHz). Logický blok triggeru je relativně dlouhý, a proto by mohlo docházet k problémům s časováním. Toto budeme řešit vložením pipeline registrů. Vhodné místo pro vkládání registrů budou vrstvy tak, jak jsou znázorněny na obr. 37. Výsledkem toho bude, že se dozvíme až o několik taktů později, že došlo ke spouštěcí podmínce. Tento fakt nám nevadí, protože budeme ukládat data průběžně mimo jiné proto, aby bylo možné libovolně nastavovat pozici triggeru relativně k oknu měřených dat (viz kapitola 3.5.1.1).

3.5.1.1 Nastavování pozice triggeru relativně k oknu platných dat

Nastavování pozice triggeru relativně k oknu platných dat nám jednoduše řečeno umožní, abychom si při konfiguraci zvolili jeden z modelových případů:

- Chceme vidět co nejdelší úsek dat, která nastala před spouštěcí podmínkou
- Chceme vidět co největší usek dat, která nastala za spouštěcí podmínkou
- Chceme vidět stejně dlouhé úseky dat před spouštěcí podmínkou jako i za spouštěcí podmínkou

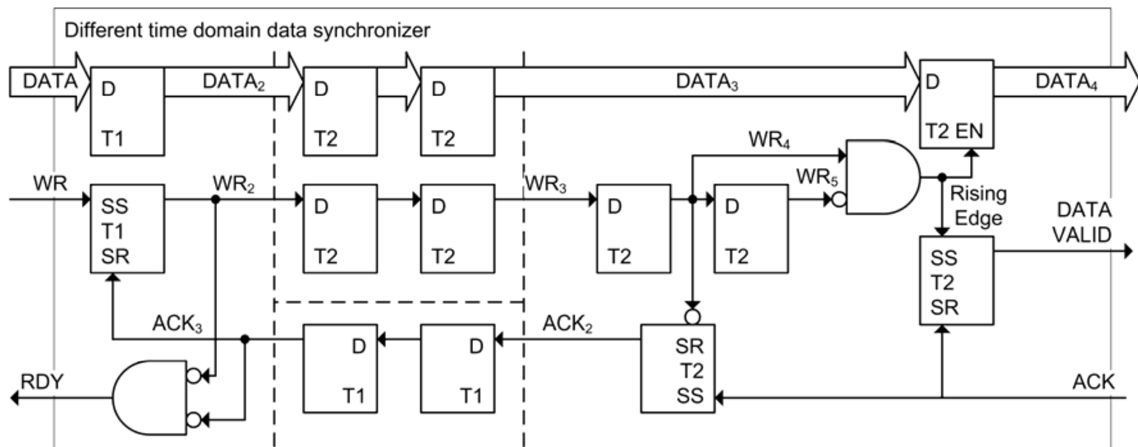
Této zdánlivě složité funkce lze dosáhnout relativně jednoduše. Od začátku měření budeme ukládat data do DDR paměti. Pokud se DDR paměť zaplní, začneme přepisovat nejstarší data od začátku paměti. Pokud dojde ke spouštěcí podmínce, vynulujeme čítač naměřených dat a načteme přednastavený počet měřených dat. Pokud bude tento přednastavený počet dat blízký velikosti paměti, většina měřených dat bude pocházet z úseku po spouštěcí podmínce. Pokud bude naopak tento přednastavený počet daleko menší, než je velikost paměti, pak bude většina měřených dat pocházet z úseku před spouštěcí událostí. Vhodnou změnou čítače bude tedy možné libovolně měnit relativní pozici spouštěcí podmínky, vzhledem k oknu měřených dat. Oknem měřených dat se rozumí velikost paměti. Pro zajištění načtení minimálního množství dat před spouštěcí podmínkou, bude možné nastavit ještě jeden čítač, který bude spuštěn na začátku měření a zabraní detekci spouštěcí podmínky tak, aby byla přednačtena požadovaná velikost dat.

3.5.2 Přenosy dat mezi dvěma časovými doménami

V mém řešení budu potřebovat dva typy přenosů. První typ bude přenos konfiguračních dat. Jedná se o přenos malého objemu informací, který probíhá typicky pouze před spuštěním logického analyzátoru. Pro tento přenos tedy nebude nutné implementovat buffer pro zvýšení propustnosti. Druhý typ přenosu budou měřená data. Pro přenos měřených dat potřebujeme velmi rychlé rozhraní. Bude proto potřeba

implementovat FIFO buffer, který zajistí potřebnou propustnost. Zpoždění, které vzniká synchronizací časových domén, se díky bufferu neprojeví na propustnosti.

3.5.2.1 Přenos malého objemu dat



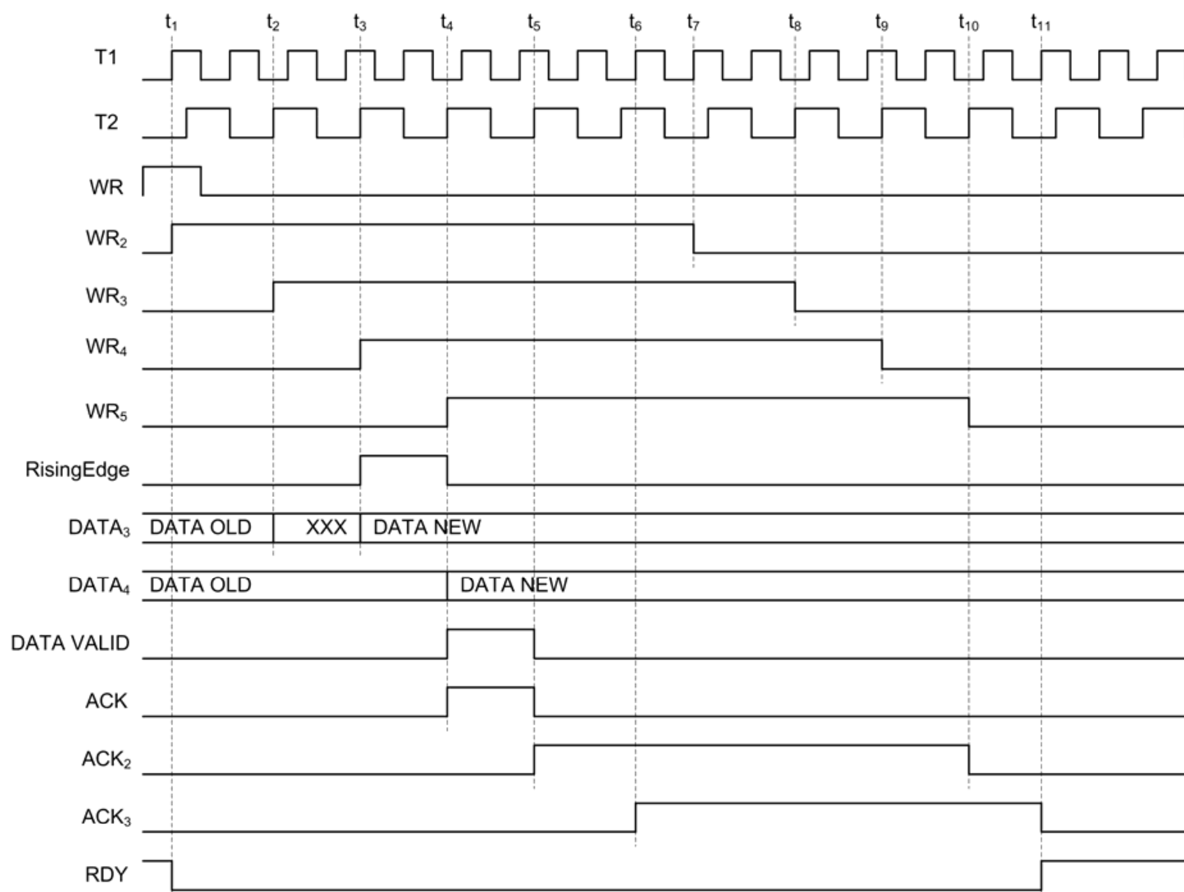
obr. 38 – Rozdílné časové domény - data synchronizace

Kanál, který nepotřebuje velkou propustnost, stačí řešit obvodem zapojeným dle obr. 38. Oproti FIFO bufferu je tento obvod daleko méně náročný na prostředky FPGA čipu. Význam jednotlivých bloků je následující:

- Blok D je klasický D flip-flop, T1 nebo T2 určuje časovou doménu ke které je synchronní. Případný vstup EN je povolení uložení dat. Pokud je EN=0, obvod si pamatuje naposledy uloženou hodnotu.
- Blok SS/SR slouží k synchronnímu nastavení logické hodnoty. Vstup SS znamená „Synchronous Set“, vstup SR znamená „Synchronous Reset“, Pokud je SS=SR=0, pak si obvod pamatuje předešlou hodnotu. T1 nebo T2 opět určuje, ke kterým hodinám je blok synchronní.

DFF uzavřené mezi čárkovanými čarami slouží jako základ synchronizace mezi časovými doménami. Standardně se používají páry DFF registrů, aby nedocházelo k metastabilnímu výstupu. K tomu by mohlo dojít, když obvod v jedné časové doméně mění výstup a obvod v druhé časové doméně se ho ve stejném okamžiku snaží uložit. Pokud je hodnota na vstupu DFF mezi platnými úrovněmi logických signálů, může dojít na výstupu tohoto DFF k pomalejšímu překlopení do platné úrovně. Během jedné periody by se však měla úroveň ustálit a na vstupu druhého DFF by již měla být přivedena stabilní úroveň. Z výstupu druhého obvodu by již měl vycházet logický signál se standardním zpožděním.

Dalším typickým problémem při synchronizaci je, že když se paralelní signály v jedné časové doméně překlopí v okamžik, kdy druhá časová doména se data snaží uložit, může se stát, že z některých signálů je uložena ještě stará hodnota, kdežto na jiných už je uložena nová hodnota. Řešení tohoto problému je jednoduché, avšak mnoho návrhářů jej opomene, nebo neprovede úplně korektně. Tento typ chyb je velice obtížné najít při simulacích, a dokonce i při testování obvodu se nemusí špatné zapojení projevit. Věřím, že mé řešení předvedené na obr. 38 je korektní. Jeho funkci si podrobně popíšeme. Pro ilustraci je průběh jednotlivých signálů z obr. 38 znázorněn na obr. 39.



obr. 39 – Signály pro synchronizaci dat mezi dvěma časovými doménami

Obvod v časové doméně T1 bude chtít zapsat data do časové domény T2. Nastaví tedy DATA a signál WR=1. Data jsou uložena do vstupního DFF spolu s žádostí WR. Zároveň je nastaven signál RDY=0. Žádost WR společně s daty se přenese do T2 přes synchronizační dvojici DFF registrů. V T2 je detekována náběžná hrana signálu WR a nová data jsou uložena do výstupního DFF registru. Zároveň je nastaven signál DATA_VALID. Do cesty signálu WR je přidán jeden DFF registr navíc, aby bylo zajištěno, že data nahraná ve stejný časový okamžik jako signál WR jsou již ustálena (viz fáze mezi úseky t_2 a t_3, kdy WR_3 je již 1, ale data se nemusela všechna stihnout uložit a může zde být mix starých a nových dat). V doméně T2 jsou data přečtena a je potvrzeno jejich přijetí signálem ACK. Signál ACK se šíří zpět do domény T1 přes synchronizační registry. Doméně T1 je tak dáno najevo, že data se již stihla přenést. V tento moment bychom mohli již vrátit signál RDY zpět do 1 a smazat WR_2 nastavením na 0 . Problém by ovšem mohl nastat, kdyby T1 byla mnohem rychlejší než T2. Pak by se mohlo v T1 požádat okamžitě o nový zápis nastavením signálu WR_2=1, ale T2 by ještě nestihla uložit signál WR_2=0. Pak by nebyla nikdy detekována náběžná hrana hodin. Musíme tedy signál WR_2=0 nechat bezpečně přenést do domény T2. Jakmile je zjištěn signál WR_4=0 v doméně T2, je vrácen signál ACK_2=0. Tato změna se přesune do T1 a až tehdy je nastaven signál RDY zpět na 1.

3.5.2.2 Přenos velkého objemu dat

Pro přenos velkého množství dat mezi dvěma asynchronními časovými doménami je vhodné použít FIFO buffer. Velice kvalitní článek k podrobnějšímu studiu tohoto tématu naleznete v publikaci „Simulation and Synthesis Techniques for Asynchronous FIFO Design“ [13]. Z tohoto materiálu budu čerpat při analýze návrhu rozhraní pro přenos velkých objemů dat mezi asynchronními časovými doménami. Doporučuji

laskavému čtenáři si tento materiál prostudovat, nejen protože je vhodnou prerekvizitou této kapitoly, ale především proto, že je to jeden z nejlepších materiálů, který jsem četl na toto téma. Dále bych rád doporučil článek [14], který je od stejných autorů a zabývá se dalším typem řešení přenosu dat mezi dvěma asynchronními časovými doménami. Druhé řešení však zde analyzovat nebudu, protože podle mého názoru přináší malé výhody a je méně přehledné, o to však více zajímavé.

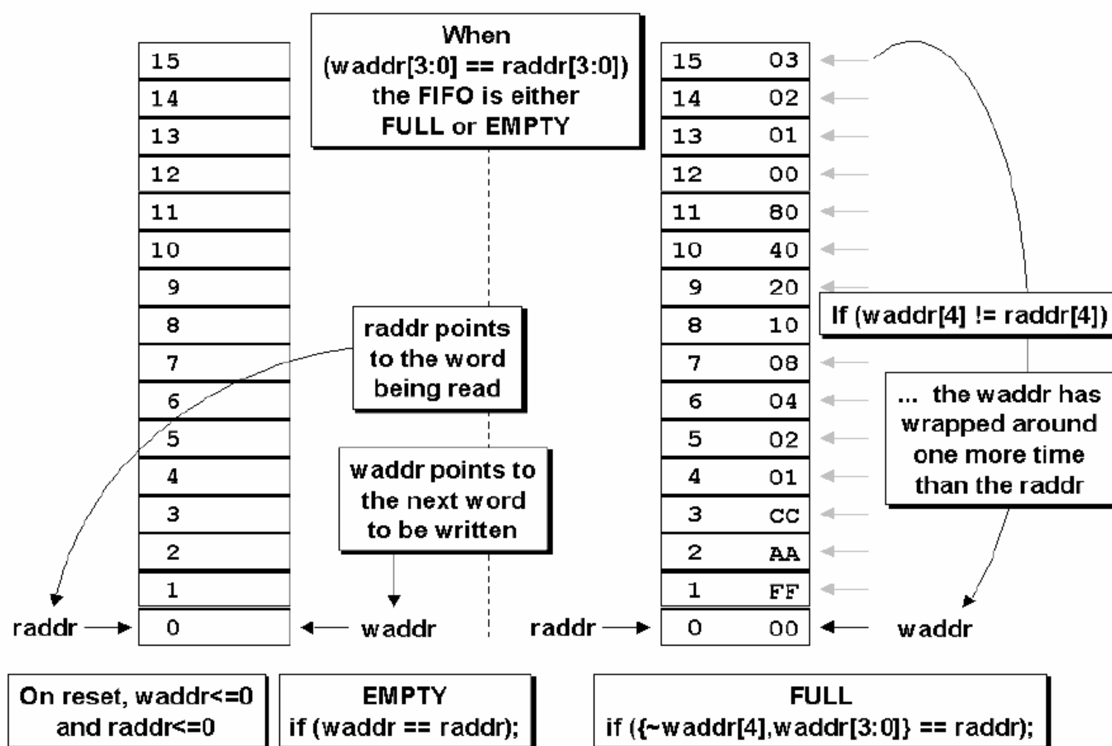
Problém je opět stejný jako v předchozí kapitole 3.5.2.1. Při změně vícebitového signálu se může stát, že tento se změní v blízkosti hrany druhé časové domény a v této jsou data nekonzistentní a to tak, že část může být stará a část nová. Nekonzistentní data pak například ve smyslu čítače dávají nesprávný kontext a mohou mít za následek špatnou funkci FIFO buffer (ať už přepsaní dat, nebo načtení neinicializovaných dat).

V této kapitole se zároveň stane problémem, že k synchronizaci, jak je popsána v předchozí kapitole 3.5.2.1, je potřeba více taktů, a proto je potřeba tuto vlastnost vhodně maskovat, abychom mohli dosáhnout požadované velké propustnosti.

Řešení rychlého rozhraní pro přenos dat mezi dvěma časovými doménami spočívá v použití dvouportové paměti v režimu kruhového FIFO bufferu.

3.5.2.2.1 Klasický dvouportový kruhový FIFO buffer

Klasický dvouportový kruhový FIFO buffer funguje tak, že má dvouportovou paměť a dva čítače. První je CntWr, ten slouží jako ukazatel na volnou pozici pro ukládaná data. Druhý je CntRd, ten slouží jako ukazatel na první data. Paměť však může být prázdná, pro tento případ existuje signal Empty. Zároveň paměť může být plná, pro tuto situaci existuje signal Full. Pokud dojde k žádosti o zápis dat (Wr), pak jsou data uložena na pozici CntWr a tento je posléze inkrementován. Pokud dojde k žádosti o čtení dat (Rd), pak jsou data načtena z pozice CntRd a tento je posléze také inkrementován. Data lze číst, pouze pokud je signál Empty=0 a zapisovat, pouze pokud je signál Full=0. Otázka zní jak nastavovat příznaky Empty a Full. Pokud jsou CntWr a CntRd různé, pak Empty=Full=0. Paměť není ani plná, ani prázdná. Na začátku jsou oba čítače CntWr=CntRd a paměť je prázdná a příznak Empty=1. Pokud se ukládají data rychleji, než jsou čtena, dojde k tomu, že CntWr dožene CntRd. Tento stav nastane tak, že po posledním platném zápisu se CntWr inkrementuje a opět dojde k situaci, že CntWr=CntRd. Tentokrát však tento stav znamená, že paměť je plná. Je tedy potřeba rozlišit plnou a prázdnou paměť. Elegantní řešení uvedeno v [13] je přidat čítačům CntWr a CntRd jeden nejvýznamnější bit CntWrMsb a CntRdMsb. Pak pokud by měl standardní CntWr přetéct, dojde k překlopení nejvýznamnějšího bitu CntWrMsb. Pak tedy pokud CntWr dožene CntRd, znamená to, že CntWr musel přetéct přesně jedenkrát víc než CntRd. Proto se nejvyšší přidané bity CntWrMsb a CntRdMsb liší. Pokud tedy nastane situace, kdy CntWr=CntRd, ale přidané CntWrMsb!=CntRdMsb, nastavíme bit Full=1. Celá situace je znázorněna na obr. 40.



obr. 40 – FIFO - full a empty situace

3.5.2.2.2 Specifikace dvouportového kruhového FIFO bufferu pro funkci ve dvou různých časových doménách

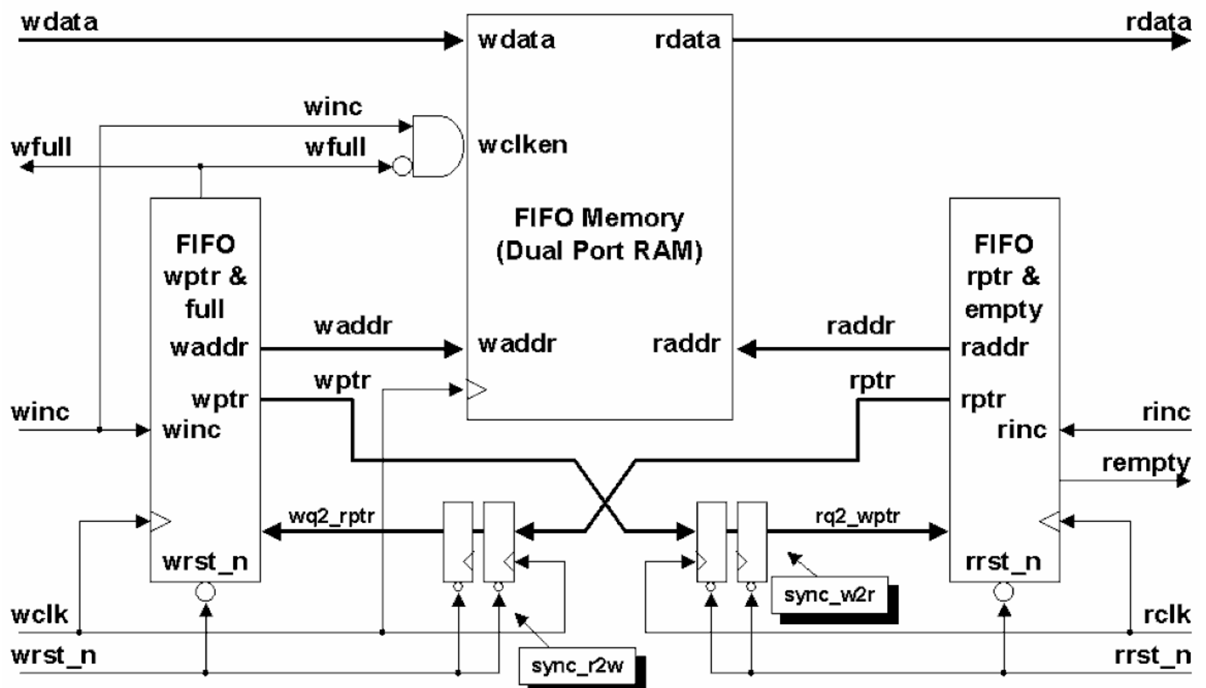
V předchozí kapitole 3.5.2.2.1 jsme uvedli funkci klasického FIFO bufferu. Základem byly čítače CntWr a CntRd, které se vzájemně porovnávaly pro určení, zda je paměť plná nebo prázdná. Problémem zápisu z jedné časové domény a čtení z druhé časové domény je, že i čítače jsou nastavovány v různých časových doménách, a nelze je proto okamžitě porovnat.

První časová doména (T1) bude ukládat data a posouvat čítač zapsaných dat CntWrT1. Druhá strana bude v další časové doméně (T2) a bude data číst a posouvat čítač čtených dat CntRdT2. Otázka nastává, jak zjistit, že v paměti jsou již uložena platná data. Jak jsme již uvedli, nemůžeme přímo porovnávat CntWrT1 a CntRdT2, protože jsou ve dvou různých časových doménách. Musíme tedy synchronizovat CntWrT1 do domény T2 (pak jej budeme nazývat CntWrT2) a CntRdT2 do domény T1 (pak jej budeme nazývat CntRdT1). Proces synchronizace zabere několik taktů a může být proveden například tak, jak byl popsán v předchozí kapitole 3.5.2.1.

Zbývá ukázat, že při práci se staršími hodnotami čítačů nedojde k nesprávné funkci bufferu. Strana pro čtení tedy bude mít několik taktů starou hodnotu CntWrT1 (synchronizovanou CntWrT2). Tato hodnota bude buď stejná nebo menší než aktuální hodnota CntWrT1 (menší bude, pokud došlo k dalším zápisům během procesu synchronizace). Při čtení je signál Empty generován, když CntRdT2 dosáhne hodnoty CntWrT2, protože tato hodnota je buď stejná, nebo menší než reálný počet skutečně uložených dat, pak nemůže dojít k načtení neplatných dat. Navíc to, že CntRdT2 došel se zpožděním, zajistí, že avizovaná data jsou již v dvouportové paměti stabilně uložena a připravena ke čtení. Strana pro zápis bude mít naopak starou hodnotu čítače CntRdT2 (synchronizovanou jako CntRdT1). Tato hodnota bude buď stejná, nebo menší než aktuální hodnota CntRdT2 (menší bude, pokud došlo ke čtení během procesu synchronizace). Při zápisu je hodnota Full generována, když čítač, CntWrT1=CntRdT1

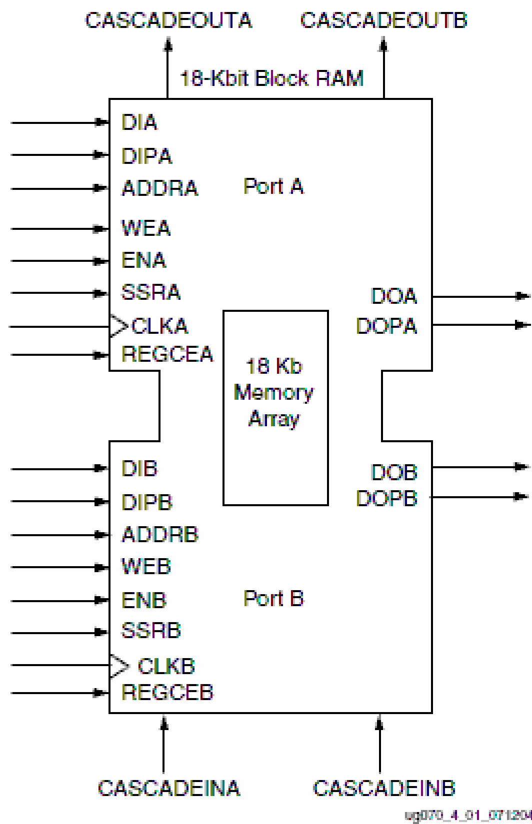
a MSB bity jsou různé. Protože hodnota CntRdT1 je buď menší, nebo stejná jako skutečně přečtená data, nedojde nikdy k přepsání nepřečtených dat.

Předchozí řešení synchronizace čítačů tak jak bylo popsané v kapitole je 3.5.2.1 je možné, avšak v [13] je uveden lepší postup synchronizace. Problém, který jsme řešili, byl, že jsme vícebitový signál potřebovali přenést korektně a tak, aby se nemohlo stát, že přijde jako mix starých a nových dat. Protože ale hodláme přenášet čítače, je tyto čítače možné udělat tak, aby se v jeden okamžik změnil pouze jeden bit. Toho docílíme, použijeme-li čítače v Gray kódu. Při vícenásobném zápisu/čtení dojde sice ke změně více bitů, ale tyto změny se dějí v různých taktech a jsou proto dost daleko od sebe. Nemůže se proto stát, že bychom proto získali mix změněných bitů. Pro operace inkrementace v Gray kódu pak použijeme převod do binárního kódu a po provedení inkrementace převedeme výsledek zpět do Gray kódu. Popřípadě použijeme dvojici čítačů. Jeden binární a tento budeme převádět do čítače v Gray kódu.



obr. 41 – FIFO buffer se synchronizovanými pointry

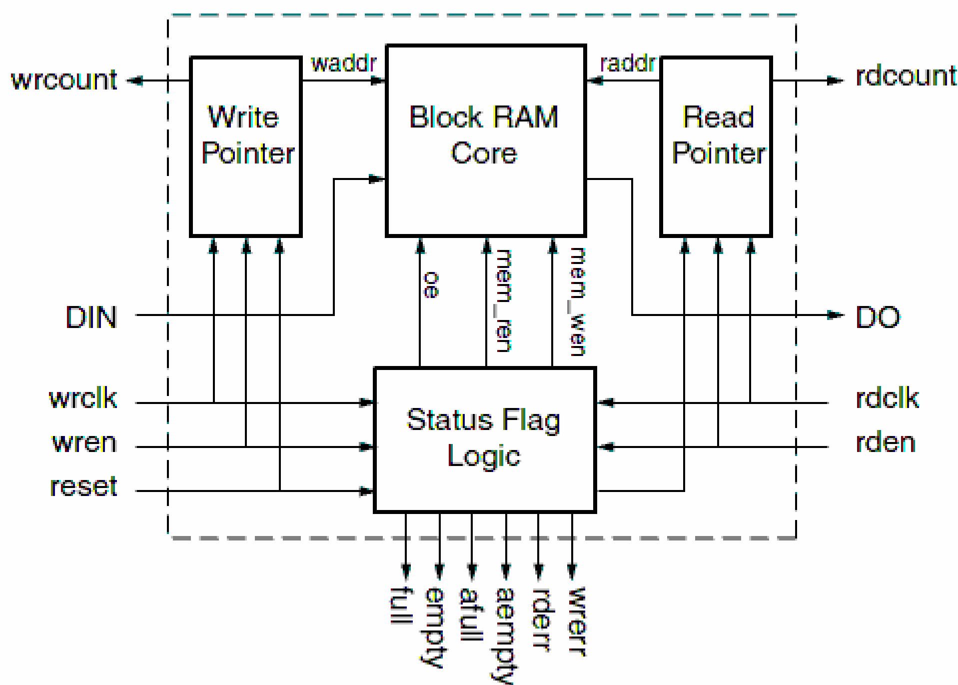
Na obr. 41 je znázorněno schéma celého FIFO bufferu. Signály, které jsou synchronní s hodinami v části pro zápis, mají prefix „w“. Signály, které jsou synchronní s hodinami, v části pro čtení mají prefix „r“. Na obrázku je čítač pro zápis označen jako wptr a čítač pro čtení jako rptr. Protože wptr a rptr jsou v Gray kódu, lze je snadno převézt do jiné časové domény. Pojmem snadno je myšleno, že není potřeba hodnoty zamykat, posílat signál WR a čekat na odpověď ACK tak, jak bylo uvedeno v kapitole 3.5.2.1 pro přenos obecného vícebitového signálu. Stále je však potřeba dvojice synchronizačních DFF registrů. Ty slouží k tomu, aby se případně metastabilní signál ustálil mezi prvním a druhým synchronizačním DFF a z druhého synchronizačního DFF již vycházel signál s definovaným zpožděním (ať již starší nebo novější). Použitím čítačů v Gray kódu tedy nejen ušetříme pár taktů při synchronizaci, ale zároveň zjednodušíme návrh celého FIFO bufferu.



obr. 42 – Dual-port blok RAM

Ukázal jsem, jakým způsobem vytvořit FIFO buffer vhodný pro synchronizaci dat mezi dvěma časovými doménami. Klíčovým prvkem je dvouportová paměť. Studiem dokumentace k obvodu Virtex-4 [11] jsme zjistili, že takováto paměť je v obvodu k dispozici. Schéma dvouportové paměti je znázorněno na obr. 42. Uvedené řešení je tedy možné použít pro zvolenou platformu.

Dalším studiem dokumentace jsem zjistil, že obvod Virtex-4 přímo podporuje vestavěnou FIFO paměť. Její výstupy jsou znázorněné obr. 43. Využitím vestavěného FIFO bloku bychom si ušetřili práci s implementací vlastního FIFO modulu. Bohužel dalším studiem této cesty jsme zjistili, že ve vestavěném FIFO bufferu je chyba, která se může projevit při jisté posloupnosti operací a konfiguraci čítačů. Podrobný popis chyby naleznete v dokumentaci k obvodu Virtex-4 [11]. Tímto se potvrzuje to, co jsem uvedl na začátku, že navrhnout kvalitní FIFO buffer pro různé časové domény je náchylné na chyby v návrhu, které jsou špatně odhalitelné při testování. Jakákoliv nedbalost v návrhu se nám může vymstít jako renomované firmě Xilinx. Dokumentace popisuje cesty, jak problém vyřešit oklikou, avšak tyto okliky spočívají ve vytvoření dalšího malého FIFO bufferu. Pro nás nejvhodnější cesta tedy bude sestavit přímo fungující FIFO buffer pomocí dvouportové paměti.



obr. 43 – FIFO v blok RAM

3.6 PC aplikace

Nyní se zaměříme na analýzu poslední části, a tou je PC aplikace, která bude sloužit k interpretaci dat a ovládání logického analyzátoru. Pro snadnou použitelnost bude potřeba realizovat grafický uživatelský interface (GUI). Přestože ovládání logického analyzátoru by bylo možné provést z příkazové řádky, GUI je stále potřeba pro přehlednou interpretaci signálů. Důležitou částí bude ovladač k USB rozhraní. Dále bude nutné implementovat aplikační rozhraní k USB ovladači. Jako cílový operační systém jsem zvolil Windows XP, jednak proto, že je zatím nejrozšířenějším operačním systémem (2008) a jednak proto, že aplikace pravděpodobně bude pracovat i pod jeho nástupcem Windows Vista.

3.6.1 USB driver a aplikační rozhraní

Napsat vlastní ovladač pro OS Windows XP je úkol časově přesahující plán na tuto diplomovou práci. Byla by potřeba dokonalá znalost systémového programování a Windows Driver Modelu, o kterém základní knihy mají rozsah přes tisíc stran. Ač by bylo patrně možné najít fungující příklad a ten pak vhodně rozšířit, časová dotace neumožňuje napsat kompletní a především kvalitní driver. Proto jsem se rozhodl najít již napsaný driver a zbytek systému mu přizpůsobit.

3.6.1.1 HID Driver

Jako nejjednodušší řešení se zdálo přizpůsobit systém HID driveru. Tento driver vyžaduje po zařízení pouze přidat delší deskriptor popisující HID rozhraní. HID driver je navíc standardně dostupný ve všech moderních operačních systémech. Po nainstalovaní WDK (Windows Driver Development Kit) a použití knihovny hidsdi.h je snadné použít HID driver v systému Windows XP. K získání kontextu zařízení a k otevření zařízení pro zápis a čtení se používají funkce SetupDiGetClassDevs, SetupDiEnumDeviceInterfaces, SetupDiGetDeviceInterfaceDetail a CreateFile. Dokumentace k těmto funkcím je

k dispozici na stránkách MSDN [15]. Detaily ohledně zmíněných funkcích nechť si laskavý čtenář dohledá buď přímo na stránkách MSDN [15], nebo přímo v přiloženém kódu na CD k této práci.

Bohužel při prvních testech bylo zjištěno, že HID driver není schopen dodat potřebnou propustnost. HID driver znamená Human Interface Device a byl napsán tak, aby podporoval „Interrupt transfer“ a jeho cílem je přenášet malé množství informací s garantovanou nejdelší odezvou. I když byla odezva nastavena na nejkratší možnou a posílané informace na největší možné, rychlost HID driveru se nepodařilo dostat přes ~19 Mbps. To je pouze zlomek rychlosti, kterou USB 2.0 nabízí, a která je až 480 Mbps. Proto jsme pro náš systém museli hledat jiný driver.

3.6.1.2 Cypress Driver

Nalezl jsme proprietární driver od firmy Cypress, který je volně dostupný. Tento driver podporuje „Bulk transfer“ a je zkompilován pro platformu Windows XP. Existuje pro něj dobrá dokumentace včetně návodu, jak driver zavést do systému. Je dodáván vzorový inf file, který je snadné upravit. Především lze zvolit vlastní GUID (Globální unikání identifikátor) driveru. Pod tímto GUID je driver poprvé otevřen z uživatelské aplikace. Je zároveň možné přejmenovat název souboru driveru. V neposlední řadě předností tohoto driveru je, že výrobce poskytuje „wrapper“ třídu, která umožňuje snadno s driverem pracovat na úrovni jazyka C++. Je možné se tak úplně vyhnout voláním příkazů jako jsou SetupDiXxxx a DeviceIoControl. Tato třída je rovněž velmi dobře zdokumentovaná a její použití ušetří při vývoji spoustu času a práce.

Driver samotný umožňuje daleko více operací, než je potřeba. Například implementuje speciální protokol pro automatické zavedení firmware. Tyto funkce jsou ale dedikované právě pro obvody firmy Cypress, jako je například obvod cy7c68014a. Tato část analýzy byla prováděna přibližně rok od analýzy a výběru USB obvodu a nebylo již možné vyměnit použitý USB čip. Avšak při této dodatečné analýze obvodu cy7c68014a (především z materiálů [16] a [17]) a celkově větším přehledu o navrhovaném systému, jsem zjistil, že by se obvod cy7c68014a daleko více hodil pro naše řešení. Především kvalitou driveru mnohonásobně převyšuje řešení s obvodem ISP1583, pro který jeho výrobce nedodává žádný driver. Obvod ISP1583 byl původně vybrán s ohledem na nízkou cenu a s předpokladem, že mikroprocesor bude realizován samotným FPGA obvodem. Při komplexnějším pohledu na systém a znalostmi možností obvodu cy7c68014a jsem však zjistil, že cena systému může být nižší, i když použijeme dražší obvod cy7c68014a. Obvod cy7c68014a totiž podporuje naprogramování z host systému po připojení k USB a byl by rovněž schopen s patřičným FW naprogramovat i FPGA obvod. Tím bychom ušetřili za obvod nonvolatilní paměti, která je v současném systému potřeba a zároveň za obvod, který se stará o programování FPGA obvodu z této paměti. Navíc bychom získali velice versatilní způsob distribuování nového FW a konfigurace FPGA obvodu. Uživatel by si stáhl pouze novou verzi ovládací aplikace a ta by standardně obsahovala FW a konfiguraci FPGA obvodu. Tyto by byly nahrány při každém připojení zařízení. Současné řešení už je navrženo a otestováno a bylo by kontraproduktivní a časově nemožné jej v této fázi měnit. Avšak pro vývoj další verze tohoto zařízení bude zřejmě v budoucnu použit právě čip cy7c68014a.

Cypress driver je možné a snadné použít i se stávajícím řešením založeném na obvodu ISP1583. Jediná nevýhoda je, že Cypress driver lze pro komerční účely legálně použít pouze s obvody od firmy Cypress (jako například cy7c68014a). Tento fakt nám zatím nevadí a v dalším řešení použijeme právě Cypress driver.

3.6.1.3 Aplikační rozhraní

Díky výběru Cypress driveru máme již zajištěno i aplikační rozhraní. Jako aplikační rozhraní bude sloužit právě „wrapper“ třída napsaná přímo firmou Cypress. Její dokumentace je dodávaná jako součást balíku SuiteUSB 1.0 [18].

3.6.2 GUI aplikace

Pro demonstraci správné funkce zařízení a jeho použitelnosti v praxi je potřeba napsat GUI aplikaci zajišťující konfiguraci a ovládání logického analyzátoru. Důležitou funkcí pak bude načtení a interpretace naměřených dat.

Je tedy potřeba zvolit vhodnou knihovnu, která zajistí aplikační framework. Standardní knihovna pro GUI aplikace od firmy Microsoft je knihovna MFC (Microsoft Foundation Class). Tato knihovna je robustní a bohatě postačující pro náš záměr, avšak rozhodl jsem se ji nepoužít. Především proto, že práce s knihovnou MFC vyžaduje rozsáhlé znalosti této knihovny a není snadné ji použít. Další nevýhodou jsou přísné nároky na stavbu programu. Úvody do práce s knihovnou MFC čítají mnoho set stránek a jednoduché problémy si často vyžádají mnoho času na řešení.

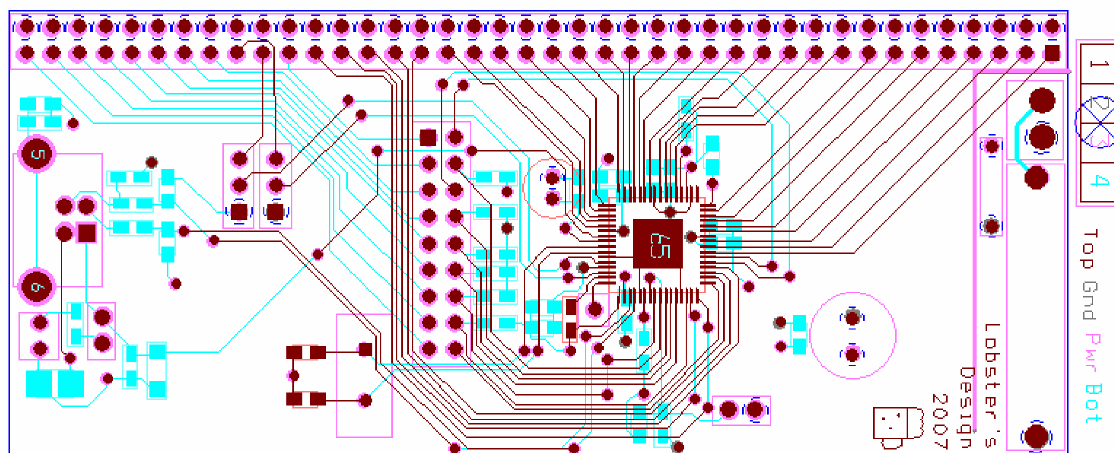
Další kandidát pro vytvoření aplikačního framework byly knihovny QT od firmy Trolltech. Tyto knihovny jsou v základní verzi zadarmo. Jejich obrovská výhoda je, že je možné je použít jak v operačním systému Windows, tak v operačních systémech Linux a Apple Mac OS. Navíc QT má jednu z nejlepších dokumentací, jakou jsem kdy viděl, a skvělý systém tutoriálů. S QT mi přišlo nejen mnohonásobně snazší pracovat, ale především mnohonásobně snazší se jej naučit. Vyzkoušení napsání jednoduché aplikace v QT a v MFC jednoznačně ukázalo, že aplikaci v QT dokážu napsat mnohem rychleji a s menším množstvím kódu než aplikaci v MFC.

4. Implementace

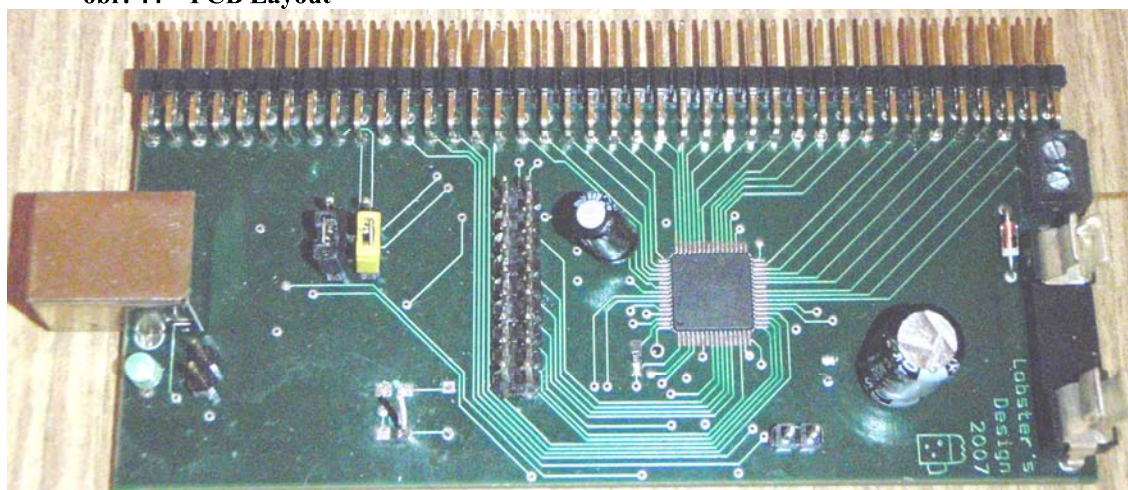
Hardwarové produkty jsou specifické nutností dokonalé analýzy. Při výběru součástek s ohledem na celkovou cenu zařízení je nutné zvolit co nejlevnější součástky, ale tak, aby byly schopny řešit daný úkol. Proto je potřeba nejen pečlivě přečíst dokumentaci, ale především již navrhnout algoritmus a odhadnout, zda je použitý obvod dostatečně výkonný pro jeho realizaci. Proto je velká část implementačních detailů již obsažena v předchozí kapitole 3. (Analýza). V tomto odstavci se tedy zaměříme na implementační detaily, které nejsou zřejmé již z analýzy. Také se zaměříme na změny, které bylo nutné vykonat při implementaci oproti původnímu návrhu.

4.1 USB IO Deska

Deska byla navržena v systému OrCAD 15.7. Byly použity nástorje OrCAD Capture, pro nakreslení schématu zapojení USB IO Desky, a OrCAD Layout, pro nakreslení šablon pro vyrobení plošného spoje USB IO Desky. Schéma USB IO desky je v příloze. Šablona pro výrobu plošného spoje USB IO Desky je zobrazena na obr. 44. Fotka vyrobené desky je na obr. 45



obr. 44 – PCB Layout



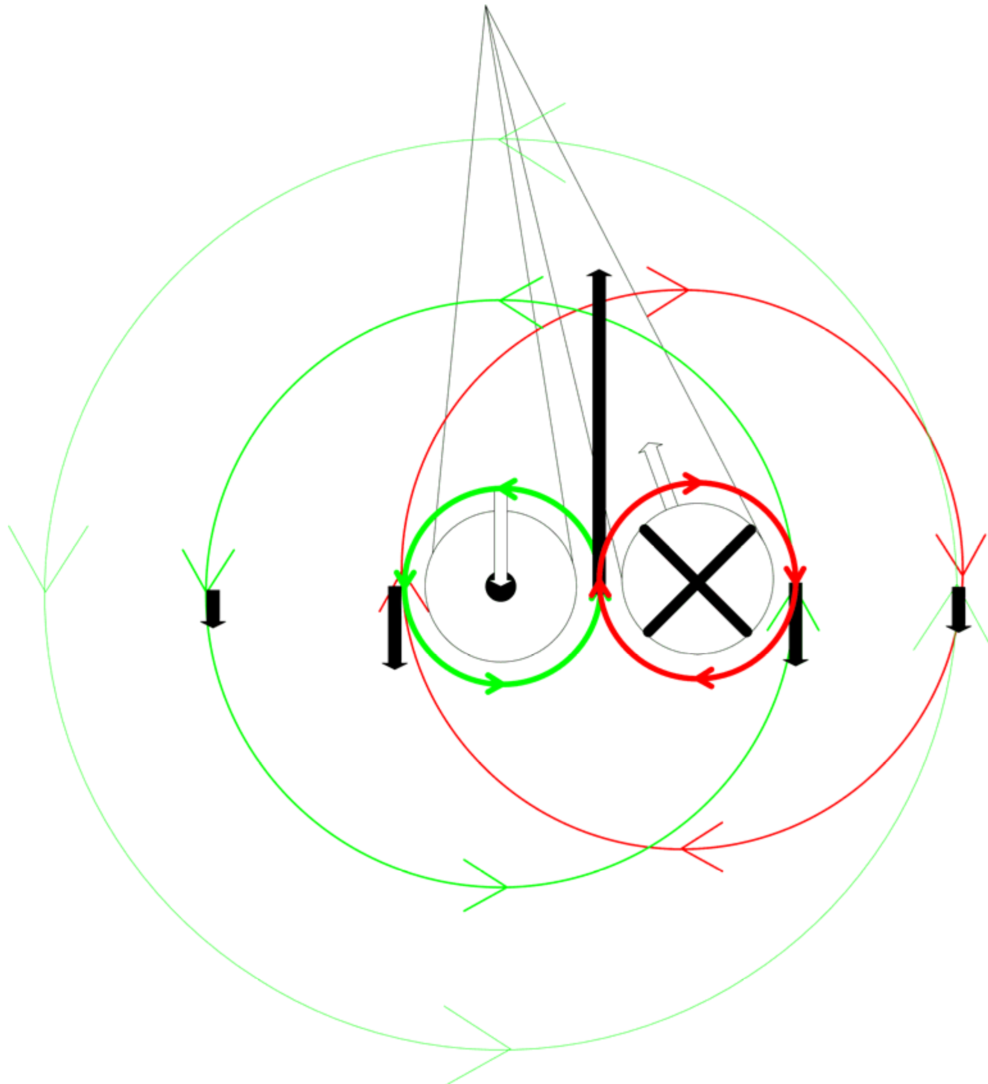
obr. 45 – PCB

Jednotlivé součástky byly vybírány s ohledem na dostupnost v České republice. Deska byla původně navrhována pro vývojový kit Spartan 3E. Problém byl s dostupností protikusu k jeho IO konektoru Hirose FX2-100P-1.27DS. Vhodný protikus je pouze konektor FX2-100S-1.27DS. Tento konektory je vyráběn společností

Hirose, která má zastoupení především v Americe a v Číně. Pro nás nejbližší dealer byl v Německu. Konektory se nakonec podařilo poptat přes firmu RS (Alfatronic). USB IO Deska má klasický kolíkový konektor a k desce Spartan3E se připojuje přes redukci.

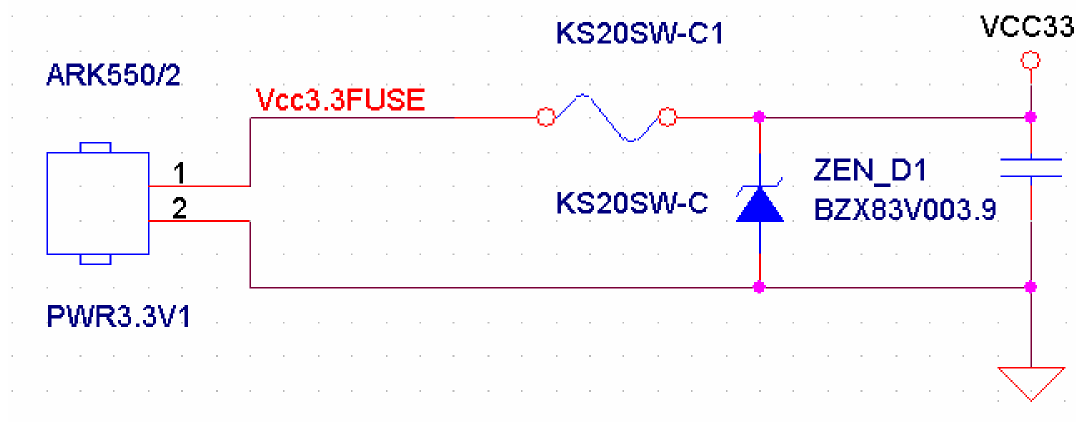
USB IO Desku jsem navrhl s dodržením obecných pravidel pro návrh DPS:

- Především je deska 4-vrstvá s vnitřními vrstvami rozlité země a rozlitého napájení (3.3V). Rozlitá země zajišťuje nízké vyzařování, jak je zobrazeno na obr. 46. Kdykoliv dojde na výstupu jednoho obvodu ke změně signálu z „0“ do „1“, dojde k proudovému impulzu na tomto signálu. Proudový impuls je zapříčiněn tím, že vstupy každého připojeného obvodu mají jistou kapacitu, která právě při změně logické úrovně představuje na krátký okamžik zkrat. Takto vzniklý proud se uzavírá smyčkou zpět přes zemní vodič. Čím více je signálový vodič blíž k zemnícímu vodiči, tím více se elektromagnetické vyzařování navzájem tlumí. Viz obr. 46 – elektromagnetické vyzařování je znázorněné magnetickými siločarami kolem vodičů a jeho směr je znázorněn šipkami. K jeho zesílení dochází právě v mezeře mezi oběma vodiči, to je znázorněno tlustou černou šipkou. K jeho tlumení pak na vnější straně vodičů. Protože vyzařování se vzdáleností klesá, je i jeho tlumení efektivnější, pokud jsou vodiče dostatečně blízko u sebe.



obr. 46 – EMI signálového a zemního vodiče tvořící proudovou smyčku

- Vodiče D+ a D- rozdílového USB signálu jsou vedeny co nejblíže u sebe. Ze stejného důvodu jako v předchozím bodě. Zároveň doplním, že nejenže vodiče samy méně září, ale zároveň jsou i méně náchylné k zašumění okolním zářením. Šum rozdílového signálu se generuje především změnou magnetického toku mezi vodiči.
- Rozlitá vrstva země a napájecího napětí vytváří také malý kondenzátor. Kondenzátor je vytvořen, protože dochází k polarizaci dielektrika (nevodivé desky mezi plošnými spoji).
- USB čip ISP1583 má zablokované přívody napájecího napětí sadou kondenzátorů. To je z důvodů špičkových odběrů, aby se nešířily přes celou desku plošného spoje až ke zdroji, ale aby se lokálně zásobily nábojem na blokovacích kondenzátorech. Kondenzátory se pak pomaleji dobijí ze zdroje. Celkové EMI je tak podstatně menší, navíc nehrozí, že při velké indukčnosti přívodu napájecího napětí obvod bude mít špičkově nedostatek energie.



obr. 47 – Ochrana proti přepětí a přepólování

Na obr. 47 je zobrazeno zapojení napájecího konektoru. Zenerova dioda spolu s tavnou pojistkou slouží jednak pro ochranu proti přepólování a jednak na ochranu proti připojení příliš velkého napájecího napětí. V obou případech se uzavře přes zenerovu diodu velký proud, který by měl mít za následek přepálení tavné pojistky. K tomuto jevu nesmí za normálních okolností docházet. Při opakovaném chybném zapojení může dojít ke zničení zenerovy diody. Ta pak nebude plnit ochranu obvodu a při chybném zapojení může dojít k poškození obvodu ISP1583.

4.1.1 Redukce pro platformu ML402

Protože v průběhu vývoje zařízení byla platforma Spartan-3E změněna za platformu ML402, bylo potřeba udělat redukci mezi konektorem USB IO Desky a kolikovým konektorem na desce ML402. Protože jsem potřeboval rychle udělat dočasnou redukci, vynechal jsem podzemnění signálových vodičů. Během dalšího vývoje a testování jsem zjistil, že FPGA někdy obdrží z USB rozhraní špatný paket. K tomu by nemělo docházet, protože USB protokol má vlastní CRC a čip ISP1583 by měl potvrzovat a ukládat pouze správně přijaté pakety.

Problémem jsem se zdržel přes dva týdny a nakonec bylo zjištěno, že na vině je právě EMI, které vzniklo použitím narychlo udělaného konektoru. Vynecháním podzemnění datových vodičů se proudové impulzy ze signálových vodičů šířily přes společnou zem USB IO desky. Fyzické vzdálenosti mezi vodičem společné země a signálovými vodiči byly řádově jednotky centimetrů. Vznikala tak proudová smyčka, která emitovala elektromagnetické záření, které patrně bylo dost velké k rušení obvodu

ISP1583. Po vytvoření nového konektoru s podzemněnými datovými vodiči vše začalo fungovat jak má a ani po několikahodinových testech (cca 16 hodin) nedošlo k obdržení chybného paketu. Předpokládal jsem, že při testovací rychlosti 6MHz EMI nebude problém. Domnívám se, že problém vznikl především kvůli rychlým hranám řídících signálů generovaných z FPGA obvodu. Ty patrně způsobily, že se zarušil obvod ISP1583 a uložil si do registru nesprávnou hodnotu.

4.2 FPGA

Jádrem celého systému je FPGA obvod. V FPGA obvodu je realizován procesor MicroBlaze. FW procesoru zajišťuje ovládání celého zařízení. Maximum potřebných funkcí systému je rovněž implementováno přímo v obvodu FPGA. Tyto funkce jsou napsány jako VHDL moduly a jsou připojeny k procesoru MicroBlaze jako periferie přes sběrnici PLB. Takovému řešení se říká „System on chip“, protože maximum operací je prováděno jedním obvodem (FPGA) a na desce je minimum podpůrných obvodů. Toto řešení je populární zejména díky své nízké výrobní ceně, malé velikosti plošných spojů a menšímu prostoru pro chyby vlivem osazení nebo studených spojů.

Pro vývoj FW a samotné konfigurace FPGA obvodu bylo použito prostředí EDK 9.1 a posléze EDK 9.2. Jako procesor byl použit softcore procesor MicroBlaze 7.0b. K vývoji FW pro procesor MicroBlaze, poskytuje prostředí EDK nástroje pro kompilaci standardního jazyku C++. Po přidání knihovny libstdc++ je dokonce možné využívat i operátory new a delete a používat plnohodnotný jazyk C++ včetně tříd a polymorfizmu a dokonce i šablon. K připojování vlastních HW periferií umožňuje prostředí EDK vytvořit VHDL modul s rozhraním na PLB sběrnici. Tento modul je pak možné rozšířit o libovolnou funkcionalitu.

4.2.1 FPGA USB Driver

Jádro FPGA USB Driveru byl Mealy automat který generoval řídící signály pro obvod ISP1583, tak jak je popsáno v analýze v kapitole 3.1.7.1. Aby byly hrany řídících i datových signálu zarovnané, byly všechny výstupy registrované. Samotný automat je velice jednoduchý a odkazuji čtenáře na přiložený kód a soubor „my_ips1583.vhd“. Rozhraní modulu je následující:

- clk – Hodinový signál, předpokládaný na 100MHz
- rst – Resetovací signál, je potřeba přiložit před použitím modulu alespoň na jeden platný takt.
- in_rd – Signál startující smyčku pro přečtení hodnoty z obvodu ISP1583 z nastavené adresy.
- in_wr - Signál startující smyčku pro zápis hodnoty z obvodu ISP1583 do nastavené adresy.
- in_adr – 8mi bitový signál pro nastavení adresy, se kterou se bude pracovat.
- in_data – 16ti bitová data pro zápis do obvodu ISP1583.
- resp_data – Výstupní 16ti bitový signál načtených dat z obvodu ISP1583.
- resp_rdy – Výstupní signál, který oznamuje dokončení poslední operace (read/write).

Rozhraní pro procesor je pomocí PLB sběrnice a prvních dvou slov od bázové adresy. Proces uložení dat do registrů je standardní tak, jak je generován pomocníkem pro vytvoření periferií pro PLB sběrnici. Jediná úprava spočívá v tom, že bity sloužící k odstartování čtení a zápisu jsou automaticky nulovány další takt. Mohlo by se totiž stát, že by je nebylo možné stihnout nulovat procesorem a mimo to by to pro relativně

pomalý procesor MicroBlaze byla zbytečně práce navíc. Podrobná dokumentace k rozhraní mezi procesorem MicroBlaze a periferii USB Driveru je přímo v hlavičce souboru „user_logic.vhd“. Uvádím zde část této dokumentace týkající se USB Driveru v podobě, ve které se nachází v hlavičce souboru:

- BASEADDR[0] WR:
  - 15 downto 0 : DATA for WRITE operation
  - 23 downto 16 : ADR to ISP1583 registers (see ISP1583 documentation)
  - 24 <= 1 write to ISP1583
  - 25 <= 1 read from ISP1583 (DATA are ireleveant)
- BASEADDR[1] RD:
  - 15 downto 0: DATA from READ operation
  - 16 == 1 => ISP ready for nxt command and data ready (if last operation was read)

Periferie sama neparsuje pakety a nerozumí komunikaci s obvodem ISP1583. Za tyto úkony je zodpovědný firmware v procesoru MicroBalze. Automat pro enumeraci USB zařízení je příliš složitý pro implementaci přímo v HW.

4.2.2 DDR Kontrolér

DDR kontrolér byl první implementován pro platformu Spartan-3E. Implementace však skončila neúspěchem. Problémem byla kombinace mých malých zkušeností s psaním kvalitního VHDL kódu a velmi vysoká efektivní frekvence DDR pamětí a s ní spojené přísné požadavky na časování. Časování bylo tak přísné, že bylo potřeba počítat se vzdáleností obvodů a zpožděním signálů na této trase, které bylo relevantní, i když tyto se šíří rychlosti blízké rychlosti světla. Podrobně se tomuto problému věnuji v analýze v kapitole 3.4. Pro řešení problémů s časováním jsem potřeboval mít možnost ladit zpoždění signálů na vstupních vodičích. Tato potřeba mě donutila nakonec použít platformu Virtex-4, která obsahuje blok IDELAY určený právě k řízenému zpoždění vstupních signálů.

Nicméně ani na platformě Virtex-4 se mi napoprvé nepovedlo implementovat DDR kontrolér. Chybu jsem objevil v tom, že výstupní signály nebyly registrované, proto docházelo k jejich posunutí vlivem nestejného počtu kombinační logiky ve výstupních signálech. Tento posun byl kritický, protože frekvence byla příliš rychlá. Tuto chybu jsem v podstatě hledal několik týdnů. Chyby v časování se špatně hledají při simulacích, navíc simulace nemají 100% přesné časování, tak jako skutečný FPGA obvod. Nakonec se mi podařilo napsat vlastní DDR kontrolér, který umí nahrát a zpět načíst data z DDR paměti. Jemné doladění zpoždění jsem provedl ručně a to tak, že jsem provedl několik syntéz s různě nastaveným zpožděním čtených dat. Pak jsem zkoušel, kdy se uložená data načtou správně.

Zvládl jsem tedy kritický krok s časováním a podařilo se mi ukládat a zpět načítat data z DDR paměti. Napsat však kompletní kvalitní DDR kontrolér je velmi obtížná práce. A domnívám se, že může být předmětem i samotné diplomové práce. Další kroky by vyžadovaly napsat relativně složitý automat, řídící obnovování pamětí, otevírání a uzavírání řádků paměti, podporu blokového zápisu a toto celé při vysoké propustnosti a efektivnosti. Rozhodl jsem se tedy, že raději zkusím najít DDR řadič již vyrobený a otestovaný. Naštěstí program ISE je dodáván s nástrojem CORE Generator. Ten obsahuje MIG 1.72 (Memory Interface Generator), který dokáže generovat DDR kontrolér pro různé typy pamětí, mimo jiné i pro můj typ MT46V16M16-6T. DDR řadič z MIG 1.72 pro paměť MT46V16M16-6T je podle dokumentace schopný tuto paměť provozovat až na frekvenci DDR333 (167MHz). Výstupem z tohoto nástroje je přímo

VHDL kód DDR kontroléru. Tento kód jsem prostudoval a nabyl jsem mnoho užitečných konstrukcí pro řešení jednotlivých problémů, které jsem řešil ve svém DDR řadiči. Například v DDR kontroléru z MIG 1.72 je univerzální řešení pro hledání velikosti posunutí dat z DDR paměti vlivem vzdálenosti obvodů. Většinu z nabitých zkušeností jsem uvedl již v analýze.

Vygenerované kódy pro DDR kontrolér jsem integroval do své periferie s názvem „logan_v1_00_a", která zatím obsahovala jen USB rozhraní. Napsal jsem pak vlastní rozhraní mezi tímto DDR kontrolérem a řídícím procesorem MicroBlaze.

Rozhraní k procesoru MicroBlaze je opět popsáno v hlavičce souboru „user_logic.vhd". Uvádím část této dokumentace relevantní k DDR kontroléru v podobě, ve které se nachází v hlavičce souboru:

- BASEADDR[2] WR:
  - 23 downto 0 : ADDRESS in DDR memory
  - 24 <= '1' : Write DATA and MASK to FIFO buffer of DDR controller
  - 25 <= '1' : WR to DDR - Write ADDDRESS to FIFO buffer and start DDR controller to write data to DDR memory
  - 26 <= '1' : RD from DDR - Write ADDDRESS to FIFO buffer and start DDR controller to read data from DDR memory
- BASEADDR[2] RD:
  - 31 == 1 : Data valid (is synchronnously cleared after read [bit 26==1] cmd)
  - 30 == 1 : DDR controller inited and ready to use
- BASEADDR[3] WR
  - 32b DATA LOW (written with BASEADDR[2].24 == 1)
- BASEADDR[3] RD
  - 32b 1st DATA LOW (check Data valid bit before read)
- BASEADDR[4] WR
  - 32b DATA HIGH (written with BASEADDR[2].24 == 1)
- BASEADDR[4] RD
  - 32b 1st DATA HIGH (check Data valid bit before read)
- BASEADDR[5] WR
  - 7 dowtno 0 DATA MASK for DATA HIGH + DATA LOW (written with BASEADDR[2].24 == 1)
- BASEADDR[5] RD
  - 32b 2nd DATA LOW (check Data valid bit before read)
- BASEADDR[6] WR
  - no function
- BASEADDR[6] RD
  - 32b 2st DATA HIGH (check Data valid bit before read)

4.2.3 Blok jádra logického analyzátoru

S návrhem samotného bloku logického analyzátoru nebyly větší problémy. Návrh byl v zásadě pouze přepsání analýzy do jazyku VHDL. Navržený blok logického analyzátoru se skládá z následujících podbloků:

- my_logan.vhd – toto je top modul, který zprostředkovává rozhraní se zbytkem periferie „logan_v1_00_a". Rozhraní umožňuje přenášet konfigurační a stavové registry. Zároveň umožňuje spustit a zastavit logický analyzátor. Stará se zároveň o komunikaci mezi jednotlivými

podmoduly, především obsahuje automat pro ukládání měřených dat do synchronizační FIFO a automat, který v další časové doméně tuto FIFO vybírá a ukládá ji pomocí DDR kontroléru do pamětí DDR.

- trigger.vhd – Tento podmodul pracuje na frekvenci sběru dat a stará se o generování signálu trigger a uchovávání konfiguračních registrů.
- fifo64.vhd – Tento podmodul realizuje FIFO buffer, který slouží pro synchronizaci měřených dat mezi dvěma časovými doménami. V první časové doméně jsou sbírána data a ukládána do této FIFO paměti. Zároveň s daty je ukládán příznak trigger. V druhé časové doméně jsou data z tohoto FIFO bufferu načítána a ukládána do DDR paměti. Tento FIFO buffer je navrhnut pro velkou propustnost a jeho základ byl převzat z nástroje CORE Generátor.
- synchro.vhd – Tento podmodul slouží pro synchronizaci konfiguračních dat, které je potřeba ze standardní časové domény přenést do časové domény, ve které jsou načítána měřená data. Tento blok je navrhnut podle analýzy synchronizace dat, která nejsou kritická na propustnost (viz kapitola 3.5.2.1).
- hwtb.vhd – tento podmodul slouží jako rozhraní k DDR kontroléru vygenerovaném v nástroji CORE Generator. Tento blok přidává FIFO pro rychlé čtení dat z DDR kontroléru. FIFO pro zápis dat bylo již součástí DDR kontroléru tak, jak byl vygenerován z nástroje CORE Generátor.

Rozhraní s procesorem MicroBlaze je opět podrobně popsáno v hlavičce souboru „user_logic.vhd“. Uvádím zde část dokumentace relevantní pro blok logického analyzátoru v podobě, ve které se nachází v hlavičce souboru:

- BASEADDR[7] WR
  - b0 = 1 = log. an. start
  - b1 = 1 = log. an. stop
- BASEADDR[7] RD
  - b0 = 1 = log. an. is running
  - b1 = 1 = FIFO underrun when DDR modul was getting data
  - b2 = 1 = FIFO overrun when log.an. modul was putting data
- BASEADDR[8] WR
  - 32b configuration data for logic analyzer module.
- BASEADDR[8] RD
  - 32b status data from logic analyzer module
- BASEADDR[9] WR
  - b[3 downto 0] - select type of configuration data
    - X"0" = no valid data
    - X"1" = store trigger configuration 00
    - X"2" = store trigger configuration 01
    - X"3" = store trigger configuration 10
    - X"4" = store trigger configuration 11
    - X"8" = store minimum counter loops before trigger detection is started
    - X"9" = store maximum counter loops after trigger event
    - X"10" = b0 = 0 - debug mode / 1 - normal mode
    - X"11" = store period configuration 0 = 10ns 1=20ns 2=30ns ... relevant bits are (26:0), bits(31:27) should be set as 0 for future compatibility
- BASEADDR[9] RD

- b0 = if 1 then logic analyzer is ready to receive configuration data
- b1 = if 1 then required status data are valid

- BASEADDR[10] WR :
  - select type of required status data
    - 0=no operation,
    - 1= b31 = 1=triggered b(23..0)=trigger DDR address (its approx +3..+10 cycles from real trigger)
    - 2= b31 = 1=DDR address roll over b(23..0)= end ddr address (last data are at address-1)

4.2.4 Firmware

Firmware v procesoru MicroBlaze má především za úkol komunikaci s USB kanálem a obsloužení USB požadavků. Procesor se ve smyčce dotazuje modulu „logan_v1_00_a“, zda obvod ISP1583 dostal z host řadiče nějaký požadavek. Pokud je detekován příznak obdržení požadavku, pak jsou vyčtena další případná data a provedena patřičná akce. Ve firmware je realizován kompletní automat pro enumeraci. Ve firmwaru jsou také napevno zakódované deskriptory zařízení, které jsou vráceny host řadiči po jejich vyžádání. Jednotlivé deskriptory jsou podrobně popsané v části analýzy v kapitole 3.1.4. Obsah deskriptorů nechť si laskavý čtenář dohledá v přiloženém kódu v souboru „devDescr.cpp“.

Pro ladění enumerace zařízení byly implementovány funkce, které logovaly komunikaci mezi FW a obvodem ISP1583 do DDR paměti. Prostředí EDK poskytuje i příkazy pro komunikaci s debugovacím modulem procesoru MicroBlaze. Je mimo jiné možné i vyčítat data uložená v paměti DDR (toto je možné pouze s originálním DDR řadičem generovaným prostředím EDK). Pro práci s ladícím modulem procesoru MicroBlaze je v prostředí EDK použit jazyk TCL. V tomto jazyku byl napsán skript pro načtení a parsování dat z DDR paměti. Výstupem byl pak textový soubor s analyzovaným přenosem. V podstatě jsem dostával výstup obdobný standardnímu výstupu z USB analyzátoru. Nevýhodou však bylo, že jsem neviděl všechny stavy USB sběrnice, ale pouze stavy které byly zpracovány obvodem ISP1583 a pouze v době, kdy jsem zavolal příslušnou komunikační funkci. Mohlo se třeba stát, že jsem načetl a zalogoval příznak příchozích dat a přitom příchozích dat bylo více od posledního smazání tohoto příznaku. Někdy proto tato cesta ladění USB enumerace nebyla úplně jednoduchá. V normálních firmách se k účelu ladění USB komunikace používá dedikovaný USB analyzátor. Na takovém zařízení jsou vidět nejen všechna data komunikace, ale i časy, hlavičky paketů, chybové stavy sběrnice a další užitečné informace. Nakonec se mi však i s mým ladícím systémem podařilo napsat knihovnu pro funkční enumeraci zařízení.

Dále jsou ve FW implementovány funkce pro ovládání modulu logického analyzátoru. Tyto funkce jsou volány po obdržení příkazového paketu na EP1 OUT. První 2 byty paketu obsahují identifikaci požadované funkce. Další byty paketu pak obsahují parametry pro danou funkci. Rozpoznávané pakety a jejich význam je následující:

- 0x0001 - Get version number.
- 0x000A - Write to DDR SDRAM (address, size, data).
- 0x000B - Read from DDR SDRAM (address, size).
- 0x000C - Start logic analyzer.
- 0x000D - Stop logic analyzer.
- 0x000E - Get logic analyzer runing status
- 0x000F - Get logic analyzer detail status

- 0x0010 - Set logic analyzer configuration(configuration_data)

Pro ladění firmwaru jsem napsal dále modul pro práci s LCD displejem. Byl jsem potom schopný zobrazovat hodnoty proměnných přímo za chodu na displeji. Popřípadě jsem si zobrazoval aktuální pozici v programu.

Při vývoji FW rovněž docházelo k zaseknutí programu při určitých datech. Několikadenním laděním jsem zjistil, že k tomu docházelo tak, že některá pole nebyla dost velká a při určitých datech se tato uložila za rozsah polí. Tím se pravděpodobně přepsal vlastní kód a aplikace se začala chovat neočekávaně. Tato chyba se velmi těžko hledala, protože typicky se projevila po delší době a ne hned, když k přepsání kódu došlo. Navíc při úpravách kódu, které byly provedeny při hledání chyby, se někdy problém vyřešil patrně tím, že byl přepisován méně používaný úsek kódu. Při dalších úpravách se pak problém znovu objevil. Nakonec byl tento problém vyřešen tak, že veškerá pole byla nahrazena třídami, které kontrolovaly, zda index není mimo rozsah a případný index mimo rozsah okamžitě vypisovaly na displej. Díky těmto třídám se snad podařilo opravit veškeré rozsahy polí.

Dále byly ve FW implementovány inicializační testy. Test na DDR paměti uloží hodnotu čítače do části paměti a tu pak přečte, pokud se uložená a načtená data shodují, je proveden další test. Další test vyčte ID obvodu ISP1583, pokud je toto ID shodné s definovaným ID, provede se test na modul logického analyzátoru. Tento je nastaven na testovací režim a zapnut. Pokud naměřená data odpovídají generovaným, pak přejde firmware do čekací smyčky, kde kontroluje USB komunikaci. V této smyčce zajišťuje enumeraci a vykonávání paketů s příkazy.

4.2.5 PC Aplikace

PC aplikace byla napsána s pomocí frameworku QT od firmy Trolltech. Pro rozhraní k USB ovladači byla použita „wrapper“ třída dodávaná spolu s USB ovladačem od firmy Cypress.

Aplikace umožňuje:

- Spustit a zastavit logický analyzátor.
- Nakonfigurovat spouštěcí podmínku.
- Nakonfigurovat měřící frekvenci.
- Nakonfigurovat minimální množství naměřených dat před začátkem detekce spouštěcí podmínky.
- Nakonfigurovat měřené množství dat po detekci spouštěcí podmínky.
- Načtení naměřených dat a údaje o jejich velikosti a pozici spouštěcí podmínky.
- Uložit a načíst naměřená data do/z XML souboru.
- Přehledně graficky zobrazit naměřená data.
- Zobrazená data je možné zvětšovat a zmenšovat.

Veškeré ovládání a konfigurace je řešena triviálně posláním paketu do zařízení do EP1. První 2 byty paketu obsahují identifikaci požadavku, další data paketu pak obsahují specifické parametry požadavku. Zařízení posílá potvrzení (popřípadě i s odpovědí, pokud je vyžadována) o přijetí každého paketu.

Pro ukládání a načítaní dat z/do XML souboru byly použity třídy prostředí QT pro práci s XML soubory. Do XML souboru jsou ukládána nejen data, ale i veškerá relevantní konfigurace. Především pozice triggeru a měřicí frekvence. Jediný problém nastal při ukládání měřených dat. Tato mohla teoreticky obsahovat sekvenci, která by v kontextu XML znamenala konec bloku dat. Bylo tedy potřeba takové sekvence detekovat a zakódovat jinak. Byl použit escape znak „\“. Pomocí tohoto escape znaku

se upravila ukládaná data tak, aby neobsahovala ukončovací sekvenci bloku dat. Při zpětném načítaní dat byl tento znak zase odebrán tak, aby načtená hodnota dat byla stejná jako původní ukládaná.

Nejsložitější část implementace bylo zobrazování dat. Měřená data totiž bylo potřeba zobrazovat v různém měřítku a posunutí časové osy. Možná měřítka časové osy byla zvolena jako počet naměřených dat postupně celočíselně dělený dvěma až na minimální počet dat 4. Při měřítku časové osy, kdy na jeden pixel obrazovky připadalo více měřených hodnot, byly všechny tyto hodnoty analyzovány a zobrazen jeden ze 3 stavů:

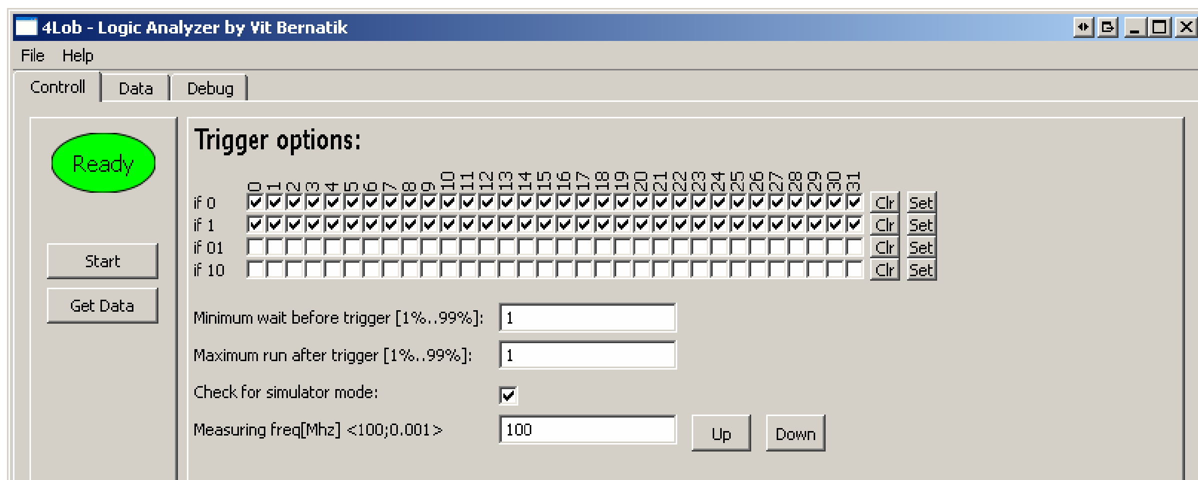
- Pixel v horní části linky, pokud všechny měřené hodnoty byly rovny 1.
- Pixel v dolní části linky pro všechny měřené hodnoty byly rovny 0.
- Svislá čárka v lince, pokud se alespoň jedna hodnota lišila od jiných.

Při měřítku časové osy kdy naopak na jednu měřenou hodnotu odpovídalo více pixelů, byla tato hodnota zobrazena jako podélná čára přes všechny tyto pixely. Pokud měřená hodnota byla 0, pak tato čára byla zobrazena v dolní části linky. Pokud tato hodnota byla 1, tato čára byla zobrazena v horní části linky. Pokud předcházející hodnota byla různá, byla zároveň vykreslena svislá čárka na rozhraní těchto dvou hodnot.

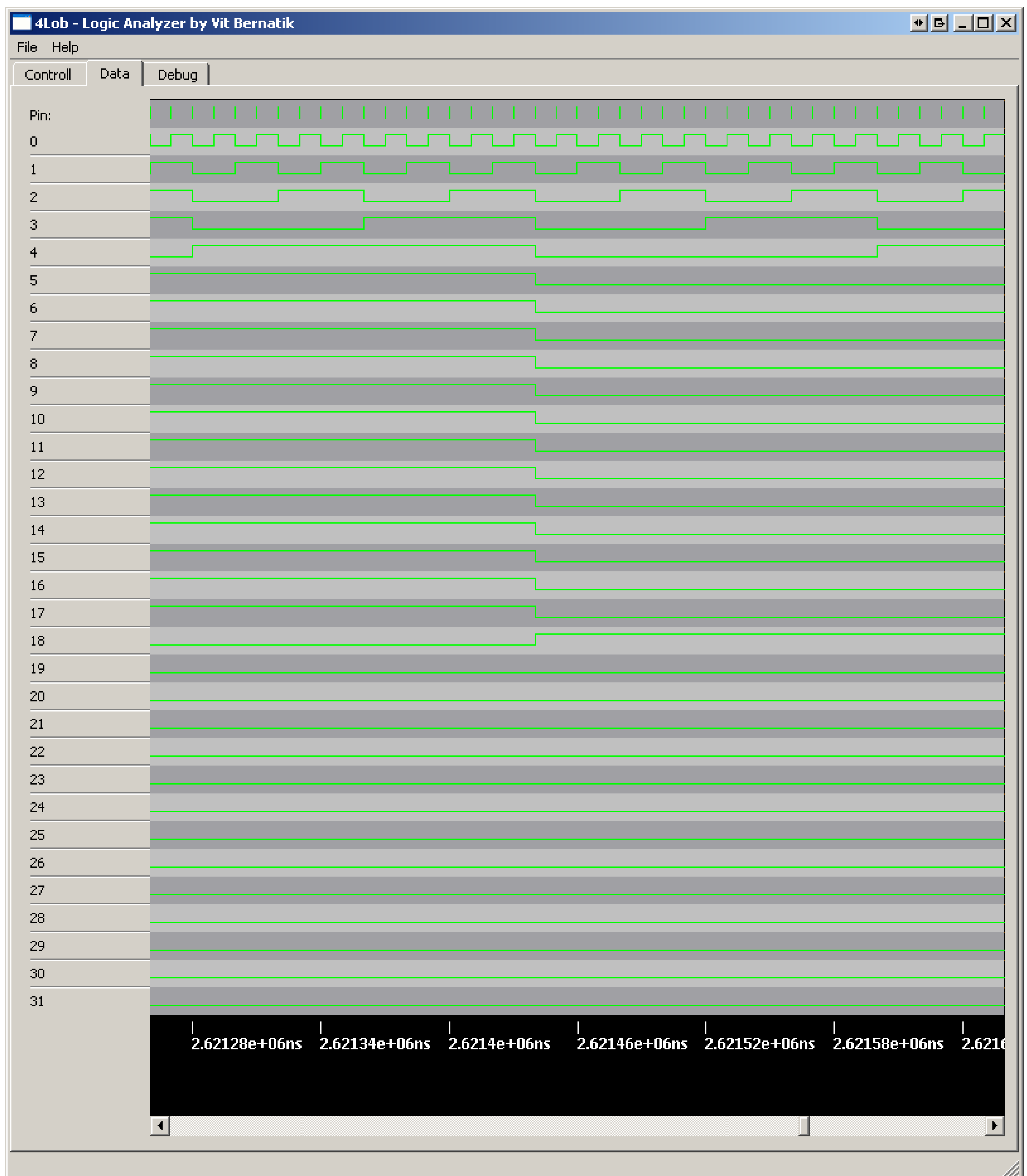
Při zvětšování a zmenšování časové osy je respektována pozice kurzoru myši v datech. To znamená, že při zmenšování či zvětšování se data, na která ukazuje kurzor myši, neposunují, ale zůstávají na stejném místě obrazovky. Tato funkce je implementována tak, že při každém zvětšení a zmenšení je provedena korekce posunutí na časové ose. Tato korekce funguje tak, že je vypočten počet dat od levého okraje zobrazovaných dat až po kurzor myši. Při zvětšení časové osy se přičte polovina této hodnoty k posunu. Při zmenšení časové osy se odečte celá tato hodnota od hodnoty posunu. Celý výpočet se mírně různí pro různá zvětšení a podmínky. Pro kompletní implementaci této funkce nechť se laskavý čtenář podívá přímo do zdrojového kódu PC ovládací aplikace.

Při zvětšování a zmenšování časové osy je zároveň přepočítáváno zobrazení časová osy tak, aby byl vždy přehledně zobrazen relativní čas pořízení měřených dat. Čas 0 je nastaven pro první zobrazovaná data. Pro další data se pak čas inkrementuje v závislosti na měřicí frekvenci. Zobrazená číselná hodnota se snaží mít maximální počet 0 na nejméně významných pozicích. Rozestup zobrazených časových značek je dán napevno body na obrazovce, při jakémkoliv zvětšení nebo zmenšení časové osy je tedy zobrazeno přibližně stejné množství časových značek. Pro kompletní funkci výpočtu časové osy nechť se laskavý čtenář opět podívá přímo do kódu PC ovládací aplikace.

Design aplikace je zobrazen na obr. 48 a obr. 49.



obr. 48 – Aplikace Logický analyzátor – záložka Controll



obr. 49 – Aplikace Logický analyzátor – záložka Data

4.2.6 Bridge mezi DDR kontrolérem a řadičem obvodu ISP1583

Procesor MicroBlaze nebyl dost rychlý k načítání dat z DDR SDRAM paměti a přeposíláním je na USB rozhraní. Maximální rychlost byla menší než 5Mbit. Poslání celé paměti pak trvalo téměř 2 minuty. Proto jsem oproti návrhu řešení v analýze přidal HW automat, který načítal data z DDR pamětí a přímo jej bez pomoci procesoru MicroBlaze zapisoval do rozhraní ISP1583 řadiče. Procesor MicroBlaze pouze zapínal posílaní paketu o velikosti 1024B z nastavené adresy DDR paměti. V podstatě je tento blok specifický DMA řadič. S použitím tohoto bloku je možné dosáhnout přenosové rychlosti ~155Mbps a přenést tak celou paměť za méně než 3,3 sekundy. To je

pro aplikaci logického analyzátoru více než dostatečné. Tento blok je realizován jako relativně jednoduchý Moore automat.

5. Testování

V průběhu několikaměsíčního vývoje byla použita spousta různých testů tak, jak byla potřeba při ověřování různých částí projektu. Mnoho z těchto testů bylo v průběhu pokračující implementace postupně odstraněno z řešení, především protože zabíraly prostředky FPGA obvodu. Není proto pro mě bohužel možné uvést veškeré dílčí mezivýsledky testování, uvádím tedy alespoň přehled testů, které byly provedeny a podrobně proberu testy, které jsou součástí finální verze.

5.1 Testbenche pro moduly

Každý navržený modul má svůj vlastní testbench (někdy i více) ve své projektové složce včetně skriptů pro simulaci v programu ModelSim.

Například nejrozsáhlejší testbench pro testování DDR řadiče a bloku logického analyzátoru obsahuje i skript pro dekódovaní stavů DDR signálů a jejich textový popis přímo v simulaci. Tento testbench obsahuje přes 140 zkoumaných signálů. Jeho funkce je inicializovat paměť, nalézt a nastavit korektní jemné posunutí a posléze posunutí v taktech pro čtená data z DDR paměti. Dále zapíše a opětovně přečte blok dat do DDR paměti. Pak nastaví všechny konfigurační registry bloku logického analyzátoru a tento odstartuje. Po detekci příznaku připravenosti bloku logického analyzátoru přečte první blok naměřených dat. Využívá přitom vytvořené procedury „bus_rd“ a „bus_wr“, které slouží k simulaci zápisu dat procesorem MicroBlaze na sběrnici PLB. Za zmínku stojí, že paměť je simulována pomocí simulačních souborů v jazyce Verilog dodávaných přímo výrobcem paměti. Je tedy možné porovnávat přímo načítaná data z DDR paměti a nemusíme zkoumat pouze správnost řídících signálů. Simulační model DDR paměti je zároveň schopen nahlásit nedodržení správného časování pro jednotlivé signály a příkazy.

Popsat do podrobnosti funkci a výsledné průběhy všech vytvořených testbenchů by zabralo příliš mnoho místa a odkazuji proto laskavého čtenáře přímo do zdrojových kódů. Všechny moduly jsou ve složce projektu EDK*\pcores. V jednotlivých modulech je pak složka devl\projnav, kde se nachází projekt pro program ISE. Po otevření projektu ISE pak většina testbenchů má předponu „tb_“. Pro samotnou simulaci pak uživatel musí mít nástroj ModelSim SE (nejlépe verze 6.2c). Při spuštění simulace z prostředí ISE by se automaticky měly spustit skripty pro přidání a zobrazení zkoumaných signálů.

Podotýkám, že simulovány byly jen klíčové momenty. Přestože jsem simulaci prováděl na relativně rychlém procesoru Intel Core 2 duo s frekvencí 3.16GHz, simulace ukládání dat do celé 32MB DDR paměti by trvala několik hodin. Ověřit veškeré konfigurace by pak nebylo možné ani během celého roku simulací na daném stroji. Bohužel simulační nástroj ModelSim nedokázal využít druhé jádro procesoru, a simulace tak trvaly podstatně déle.

5.2 Funkční test USB rozhraní

USB kanál byl testován velmi extenzivně. Především, když byly zjištěny chyby při přenosu dat, bylo napsáno mnoho testů pro různé vzorky dat. Vzpomínám si, že jsem implementoval testy pro přenos hodnot:

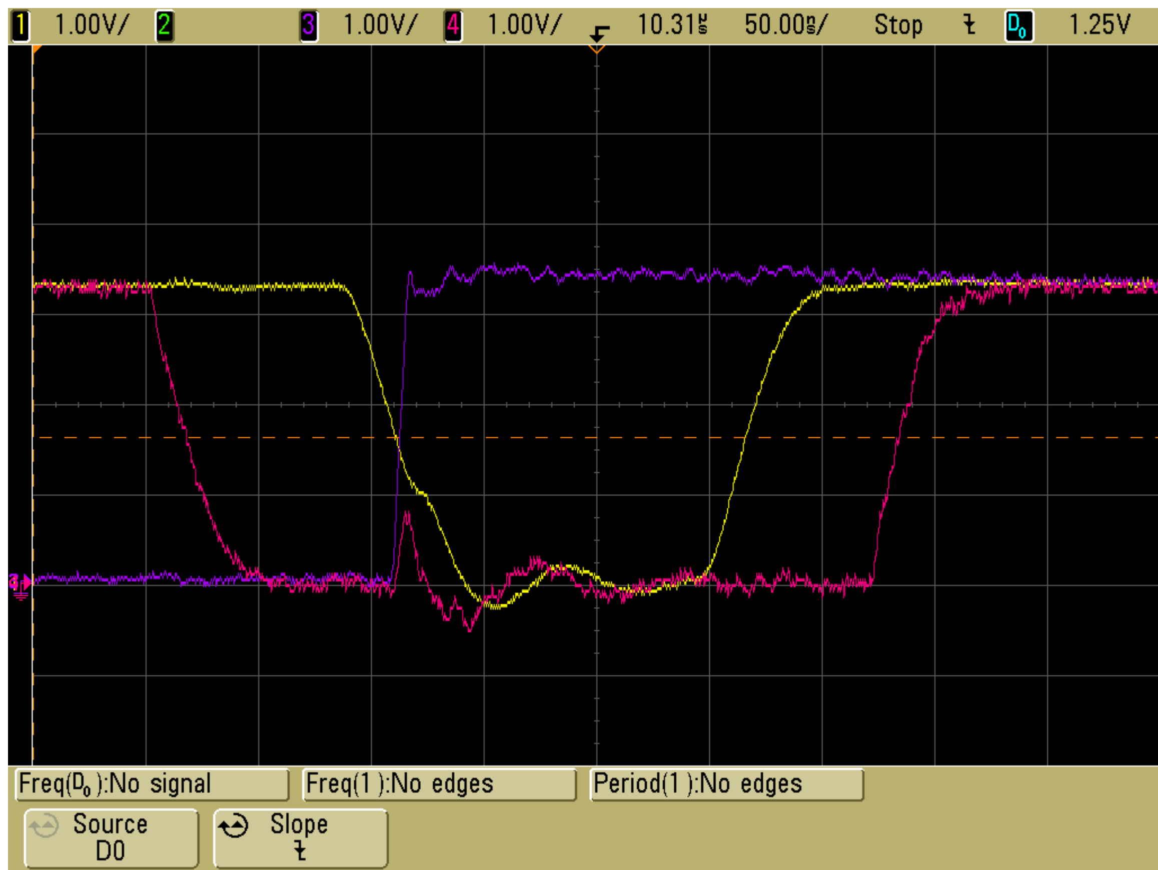
- Střídaní samých nul a samých jedniček.
- Pochodující 1.
- Pochodující 0.
- Všechny kombinace po sobě jdoucích 8mi bitových hodnot.

Testy bylo zjištěno, že některé pakety se náhodně kazí. K této situaci by nemělo docházet mezi host řadičem a USB obvodem, protože USB protokol poskytuje CRC pro každý paket a obvod ISP1583 implementuje základní vrstvu USB protokolu a měl by ignorovat pakety, u kterých nesouhlasí CRC. Dlouhou dobu jsem předpokládal, že problém je v návrhu a je chybou modulu pro komunikaci s obvodem ISP1583, nebo chybou FW v procesoru MicroBlaze. Několikrát jsem přečetl dokumentaci k rozhraní obvodu ISP1583 včetně FAQ a několika application notes, podařilo se mi při tom najít a opravit některé drobné chyby, avšak problém s chybnými pakety přetrvával. Obvykle bylo chybných méně než 5% paketů. Podařilo se mi však najít testovací data, pro které byla chybovost až 50%.

Jako rychlou záplatu jsem navrhl vlastní vrstvu protokolu, která posílaným datům přidávala CRC16. Toto CRC bylo testováno a data posílána ve smyčce zpět. Tato záplata během celé doby testování odhalila korektně všechny poškozené pakety. Toto řešení však nebylo úplně korektní a zhoršovalo skutečnou propustnost USB rozhraní.

Začal jsem tedy hledat chybu v HW. Chyba se objevovala na různých pinech, avšak na některých pinech s vyšší pravděpodobností než na jiných. Zkoušel jsem snižovat rychlost komunikace mezi FPGA a obvodem ISP1583. Tento postup nepřinesl žádné znatelné zlepšení. Ověřil jsem, zda na cestě nejsou studené spoje. Poté jsem měřil průběhy řídících signálů na osciloskopu, jejich průběhy uvádím na obr. 50 a obr. 51.

obr. 50 – Čtení 0



obr. 51 – Čtení 1

Význam signálů na obr. 50 a obr. 51 je následovný:

- Červená čára = signál ChipSelect.
- Žlutá čára = signál DataStrobe.
- Modrá čára = signál Data.

Z obrázků je patrné, že signál Data se překlopí při detekci hrany signálu DataStrobe. Signál Data se pak postupem času pozvolna vybíjí. Signál Data je stabilní a není proto možné, aby chyba při čtení dat vznikala problémem se signálem Data. Nicméně řídící signály nejsou zcela v pořádku. Na signálu ChipSelect lze vidět glitch při překlápění signálu DataStrobe. Tento glitch vznikl špatným návrhem modulu pro komunikaci s obvodem ISP1583. Řídící automat byl navržen tak, že výstupní signály byly měněny kombinační logikou na základě vnitřního stavu. Při změně stavu tak docházelo ke glitchům na řídících signálech. Tento problém byl vyřešen registrováním výstupních signálů z automatu. Nicméně na poměr chybných paketů tato oprava neměla vliv.

Nakonec byly na řídící signály připojeny malé kapacity (22pF), aby došlo k přizpůsobení vodičů, ač délka byla jen asi 15cm. Tento postup velmi pomohl zlepšit poměr špatných paketů, avšak problém zcela nevyřešil. Navíc korektní přizpůsobení se implementuje pomoci kapacitoru a rezistoru v sérii.

Problém byl nakonec vyřešen tak, že se přidalo každému signálovému vodiči podzemnění. Tato úprava byla podrobně popsána v kapitole o redukci mezi USB IO deskou a přípravkem ML402 (4.1.1).Chybovost pak i pro 16 hodinový test byla 0. Domnívám se tedy, že problém byl způsoben elektromagnetickým rušením (EMI), kdy řídící signály rušili funkci obvodu ISP1583. Snižování rychlosti signálů nepomohlo, protože problémem byly ostré hrany signálů. Při změně hrany z 0 do 1 se uzavřela proudová smyčka přes kapacitu vstupních pinů obvodu ISP1583 a tato se šířila přes společnou zem zpět ke zdroji. Fyzický tvar smyčky (přesněji její velká

vnitřní plocha) způsobil patrně dostatečně velké rušení pro obvod ISP1583 a tento nefungoval správně. Domnívám se, že původně přidané malé kapacity na řídící vodiče zmenšily strmost hran a změnily tvar proudové smyčky a částečně tak snížily procento chybných paketů.

Maximální měřená rychlost mého USB rozhraní byla větší než 190Mbps (23,75MBps). Data byla generovaná čítačem v FPGA obvodu a zapisována do modulu pro komunikaci s obvodem ISP1583. Velikost testovaného bloku dat byla 16MB.

5.3 Funkční test DDR paměti

Podobnými testy jako USB rozhraní prošlo i rozhraní s DDR pamětí. Byly naimplementovány testy, které ukládaly různé hodnoty do DDR paměti a ty posléze četly a porovnávaly. Konkrétně byly implementovány testy pro ukládání hodnot:

- Střídaní samých nul a samých jedniček.
- Pochodující 1.
- Pochodující 0.

Tyto testy běžely opět déle než 10 hodin bez chyb.

5.4 Funkční test celého systému

Ve finální verzi zůstal jen zlomek testů. V inicializační části FW je krátký test pro DDR paměť, modul logického analyzátoru a obvod ISP1583. Tyto testy jsou popsány v kapitole o návrhu FW (4.2.4).

V aplikaci samotné pak je možné kdykoliv provést test, který testuje všechny bloky zařízení. Napřed se spustí sběr měřených dat v simulačním módu. Data jsou těsně za vstupem uměle generována čítačem. Tato jsou přenášena do další časové domény, kde jsou ukládána do DDR paměti. Po skončení měření jsou uživatelem načtena přes USB rozhraní. V aplikaci je pak třetí záložka „Debug“, ve které je tlačítko „TestData“. To ověří, zda jsou naměřená data přesně výsledkem čítače. Pokud tomu tak je, test projde úspěšně, pokud tomu tak není, vypíše se chybová hláška s pozicí chybných dat. Za celou dobu zkoušení finální verze a mnohačetným ověřováním dat nikdy nedošlo k chybě.

Aplikace zároveň měří rychlost přenosu naměřených dat. Výsledky závisí především na samotné velikosti přenášených dat. Při maximální velikosti (tedy velikosti DDR paměti = 64MB) je přenosová rychlost 155Mbps (19,4MBps) a přenos celé paměti zabere méně než 3,3s.

6. Zhodnocení

Podařilo se navrhnout a vyrobit logický analyzátor, který několikrát převyšuje požadavky zadání. Původně jsme požadovali osciloskop s měřící frekvencí alespoň 50MHz. Navržený logický analyzátor je schopen pracovat až na 100MHz. Zároveň jsme požadovali 8 měřících kanálu. Navržený logický analyzátor je schopen měřit až 32 kanálů. Jsme tedy schopni 2x rychleji měřit 4x více kanálů oproti původnímu zadání.

Zároveň byla k řešení napsána přehledná ovládací aplikace s grafickým rozhraním. Byly implementovány i pokročilé funkce. Nejen základní nastavení spouštěcí podmínky, ale i možnost měnit měřicí frekvenci a pozici spouštěcí podmínky v rámci okna měřených dat. Naměřená data jsou přehledně zobrazena spolu s časovou osou, pozicí triggeru a zobrazením pozice vzorkovacích úseků.

Původně se předpokládalo, že zařízení bude realizováno na vývojovém kitu Spartan-3E. Toho se nepodařilo dosáhnout. Překážkou byly především přísné nároky na časování DDR řadiče. Zařízení bylo tedy vyvinuto pro platformu Virtex-4. Domnívám se však, že nyní, když je vše odladěno a otestováno, by řešení šlo naporotvat zpět na platformu Spartan-3E. Bohužel mi nezbyl čas, abych se o to pokusil.

Bohužel mi nezbyl čas ani na konečnou fázi dalšího zvyšování měřicí frekvence systému. Domnívám se, že na platformě Virtex-4 bych teoreticky mohl docílit měřicí frekvence až 160MHz. Zároveň jsem chtěl zkusit implementovat řešení pro méně měřicích kanálů a větší frekvenci. Na to bohužel také nezbyl čas.

V porovnání ceny ku výkonu jsem však dosáhl konkurenceschopného produktu, což si nemyslím, že by dokázala každá diplomová práce. Firma GME nabízí logický analyzátor s názvem PROG.SIGMA za 9500CZK. Tento logický analyzátor dokáže 16 kanálů vzorkovat pouze frekvencí 50MHz. Moje řešení je tedy 4x výkonnější a jeho pořizovací cena je 11500CZK za vývojový kit ML402 a přibližně 1000CZK za rozšiřující USB IO desku. Celková cena je tedy 12500CZK. To je sice 1,32x více než za PROG.SIGMA ale za 4x výkonnější zařízení. Dalším porovnávaným logickým analyzátorem je profesionální logický analyzátor 16801A od firmy Agilent. Tento logický analyzátor má vzorkovací frekvenci 500MHz pro 34 signálů. Je tedy přibližně 5.32x výkonnější než můj logický analyzátor, jeho cena je však 200000CZK, což je 16x více, než je cena mého logického analyzátoru. V poměru ceny ku výkonu se tak můj logický analyzátor ukázal lepší než oba konkurenční analyzátory. Navíc je dosavadní řešení pouze první verze, a mnohé vylepšení jsem ještě nestihl do uzávěrky této diplomové práce. Domnívám se, že kdybych měl další půlrok pro vývoj mého logického analyzátoru, převyšoval by poměr ceny ku výkonu ještě více než tato první verze.

6.1 Zhodnocení využití obvodu FPGA

Přikládám tabulku využití prostředků obvodu, tak jak je generována ve vývojovém prostředí EDK:

Number of BSCANs	1 out of 4	25%
Number of BUFGs	5 out of 32	15%
Number of DCM_ADVs	2 out of 8	25%
Number of DSP48s	3 out of 192	1%
Number of FIFO16s	7 out of 192	3%
Number of IDELAYCTRLs	2 out of 16	12%
Number of ILOGICs	85 out of 448	18%
Number of External IOBs	154 out of 448	34%

Number of LOCed IOBs	154 out of 154	100%
Number of OLOGICs	96 out of 448	21%
Number of RAMB16s	32 out of 192	16%
Number of Slices	7855 out of 15360	51%
Number of SLICEMs	475 out of 7680	6%

7. Závěr

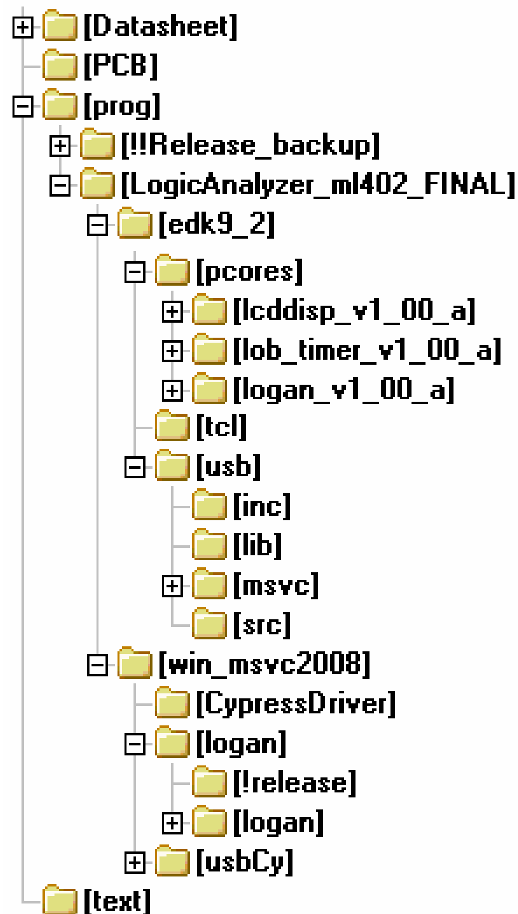
Podařilo se vyrobit fungující logický analyzátor. Maximální měřící frekvence je nyní 100MHz a je možné při této frekvencí najednou měřit všech 32 kanálů. Je možné nastavovat spouštěcí podmínky pro jakoukoliv kombinací stavů na všech měřících kanálech. Při maximální frekvenci 100MHz je možné naměřit úsek dat o velikosti 167ms. Je možné snižovat měřicí frekvenci pro získání delší doby měření. Frekvence se dá snížit až na 1KHz a pak lze naměřit přes 4,6hodin dat. Načtení celé paměti (tedy 64MB dat) zabere méně než 3.3 sekundy a probíhá rychlostí vetší než 155Mbps (19.5MBps). Pro srovnání relativně rychlý flashdisk Voyager od firmy Corsair dokáže dodávat data pouze rychlostí 18,8MBps. Je možné libovolně nastavovat pozici spouštěcí podmínky v rámci měřeného okna dat.

Byla vytvořena ovládací aplikace, která je velmi přehledná a rychle použitelná. V aplikaci byly pro zvýšení užitnosti v praxi implementovány funkce pro uložení naměřených dat do XML souboru a zpětné načtení a zobrazení dat z XML souboru. Naměřená data jsou přehledně graficky znázorněna a je možné libovolně zvětšovat a zmenšovat měřítko časové osy.

Výsledkem této práce je fungující použitelný produkt, který poměrem cena/výkon překoná mnohé komerční řešení dosud na trhu. Navíc pokud uživatel již vlastní vývojový kit ML402 od firmy Xilinx a chce ho upgradovat na logický analyzátor, stačí si pouze pořídit modul rozhraní pro komunikaci s USB, který byl vyvinut v rámci této práce. Tento modul se dá vyrobit za řádově 1000CZK. Uživatel tak má možnost si pořídit relativně výkonný logický analyzátor za zlomek ceny logických analyzátorů na trhu.

8. Přílohy na CD

Obsah CD:



- Použité katalogové listy
- Schéma a layout PCB USB IO desky
- Veškeré kódy systému
- Zálohy starších verzí
- Kompletní finální kódy pro celý systém
- HW + FW
- Všechny navržené moduly
- Modul pro LCC Display
- Modul pro přesné časování
- **Modul pro log. an**. (i s DDR a USB řadičem)
- Skripty pro ladění USB rozhraní
- **FW pro MicroBlaze**
- Hlavičkové zdrojové soubory
- Knihovní soubory
- Projekt v MSVC (ne pro kompilaci)
- Zdrojové kódy
- PC aplikace
- **Použitý driver pro Windows XP**
- **GUI aplikace Logický analyzátor**
- Zkompilovaná verze pro Windows XP
- Zdrojové kódy a projekt
- Konzolová testovací aplikace
- **Text samotné diplomové práce**

9. Reference

[1] USB 2.0 Specification
http://www.usb.org/developers/docs/usb_20_122208.zip

[2] Bernatík, V.: TempLab – tester sítě PROFINET (bakalářská práce)
ČVUT, Praha 2006

[3] USB in a Nutshell
http://www.beyondlogic.org/usbnutshell/usb-in-a-nutshell.pdf

[4] Axelson, J.: USB chips
http://www.lvr.com/usbchips.htm

[5] Axelson, J.: USB Complete
Lakeview Research, 2005, ISBN# 1-931448-02-7

[6] Data sheet ISP1583
http://www.nxp.com/acrobat/datasheets/ISP1583_7.pdf

[7] ISP1583 Firmware Programming Guide (AN10039_4)
http://www.nxp.com/acrobat/applicationnotes/AN10039_4.pdf

[8] ISP1583 Frequently Asked Question (AN10046_8)
http://www.nxp.com/acrobat/applicationnotes/AN10046_8.pdfa

[9] ML402 User Guide
http://www.xilinx.com/support/documentation/boards_and_kits/ug080.pdf

[10] Data Sheet paměti DDR SDRAM MT46V16M16P-6T
http://download.micron.com/pdf/datasheets/dram/ddr/256MbDDRx4x8x16.pdf

[11] Virtex4 FPGA User Guide
http://www.xilinx.com/support/documentation/user_guides/ug070.pdf

[12] Virtex-4 FPGA Data Sheet: DC and Switching Characteristics
http://www.xilinx.com/support/documentation/data_sheets/ds302.pdf

[13] Clifford E. Cummings and Peter Alfke, “ Simulation and Synthesis Techniques for Asynchronous FIFO Design ”
SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002), March 2002,
www.sunburst-design.com/papers

[14] Clifford E. Cummings and Peter Alfke, “Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons,”
SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) March 2002,
www.sunburst-design.com/papers

[15] Microsoft Developer Netvrok
http://msdn.microsoft.com

[16] EZ-USB FX2LP™ USB Microcontroller High-Speed USB Peripheral Controller
http://www.cypress.com/MyAccount/index.cfm?type=qtp&target=fx2lp_fx1_at2lp_family__c8q_3r__fab4__cy7c68xxx__4.pdf

[17] EZ-USB® Technical Reference Manual
http://download.cypress.com.edgesuite.net/design_resources/technical_reference_manuals/contents/ez_usb_r___technical_reference_manual__trm__14.pdf

[18] Cypress CyAPI Programmer's Reference
Tato dokumentace je součástí balíku SuiteUSB 1.0:
http://app.cypress.com/portal/server.pt?space=CommunityPage&control=SetCommunity&CommunityID=285&PageID=552&drid=80195&shortlink=&r_folder=&r_title=&ref=drs

10. Přílohy

10.1 Schéma USB IO desky