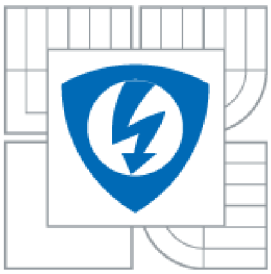
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV MIKROELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF MICROELECTRONICS

JEDNOTKY PRO ASYNCHRONNÍ PŘECHODY V OBVODECH FPGA

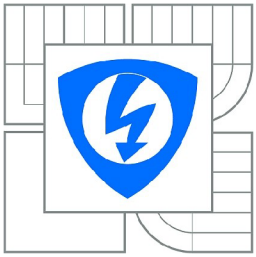
ASYNCHRONOUS COMMUNICATION INTERFACES IN FPGA

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — JAKUB CABAL
AUTHOR

VEDOUCÍ PRÁCE — ING. MAREK BOHRN
SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav mikroelektroniky

Bakalářská práce

bakalářský studijní obor
Mikroelektronika a technologie

Student:	Jakub Cabal	***ID:***	154688
Ročník:	3	***Akademický rok:***	2014/2015

NÁZEV TÉMATU:

Jednotky pro asynchronní přechody v obvodech FPGA

POKYNY PRO VYPRACOVÁNÍ:

Proveďte rozbor možností implementace jednotek pro řešení asynchronních přechodů v obvodech FPGA. Věnujte se především správnému řešení asynchronních přechodů pomocí asynchronních pamětí FIFO a metody handshake. Dále se zaměřte na aplikování správných omezujících podmínek (constraints).
Navržené jednotky pro řešení obecných asynchronních přechodů implementujte do obvodu FPGA a ověřte jejich funkčnost pomocí simulace. Implementované jednotky následně použijte pro realizaci asynchronních přechodů na proprietárních datových sběrnicích MI32, FLU, FL a DMA používaných sdružením CESNET.
V závěru práce vytvořte metodiku pro použití realizovaných asynchronních přechodů v obvodech s více hodinovými doménami. Uplatnění metodiky demonstrujte na případové studii v obvodu síťové karty vytvořeném pro akcelerační kartu COMBO-80G.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce.

Termín zadání:	10.2.2015	***Termín odevzdání:***	4.6.2015

Vedoucí práce: Ing. Marek Bohrn
Konzultanti bakalářské práce: Ing. Jiří Matoušek, CESNET

doc. Ing. Jiří Háze, Ph.D.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Cílem této práce je provést rozbor a implementaci jednotek pro řešení asynchronních přechodů v obvodech FPGA. Tyto přechody jsou nevyhnutelné ve složitějších obvodových návrzích a jejich nesprávná implementace může vést ke ztrátě nebo poškození dat. Dále se práce zaměřuje na aplikování správných omezujících podmínek (constraints). V praktické části této práce je popsána realizovaná knihovna asynchronních přechodů. Dále praktická část práce popisuje vytvořenou metodiku použití asynchronních přechodů, jejíž uplatnění je demonstrováno na případové studii v obvodu síťové karty vytvořeném pro akcelerační kartu COMBO-80G.

KLÍČOVÁ SLOVA

FPGA, VHDL, asynchronní přechod, FIFO, omezující podmínky

ABSTRACT

The aim of this thesis is to analyze the options for implementation of asynchronous modules for clock domain crossing in an FPGA circuit. Such crossings are inevitable in moderately complex firmware designs and can lead to data corruption or loss, if implemented incorrectly. Furthermore, the work deals with application of correct constraints. The practical part of this work describes an implemented library of clock domain crossing modules. Further, the practical part describes a created methodology for use of clock domain crossing modules, whose application is demonstrated in a case study of a network interface card circuit created for the acceleration card COMBO-80G.

KEYWORDS

FPGA, VHDL, clock domain crossing, FIFO, constraints

BIBLIOGRAFICKÁ CITACE

CABAL, J. *Jednotky pro asynchronní přechody v obvodech FPGA*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2015. 30 s. Vedoucí bakalářské práce Ing. Marek Bohrn.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma Jednotky pro asynchronní přechody v obvodech FPGA jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne 4. června 2015 .....

podpis autora

PODĚKOVÁNÍ

Děkuji vedoucímu bakalářské práce Ing. Marku Bohrnovi a odbornému konzultantovi Ing. Jiřímu Matouškovi za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování této práce.

V Brně dne 4. června 2015 .....

podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam symbolů, veličin a zkratek

BRAM	Block Ram
CPLD	Complex Programmable Logic Device
DIP	Dual In-line Package
DMA	Direct Memory Access
DSP	Digital Signal Processing
EEPROM	Electrically Erasable Programmable Read-Only Memory
FF	Flip Flop
FIFO	First In, First Out
FL	Frame Link
FLU	Frame Link Unaligned
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronics Engineers
LUT	Look-Up Table
MI32	Memory Interface 32 bit
MTBF	Mean Time Between Failures
PLCC	Plastic Leaded Chip Carrier
PLL	Phase-Locked Loop
QFP	Quad Flat Package
RAM	Random-Access Memory
ROM	Read-Only Memory
SPLD	Simple Programmable Logic Device
SSOP	Shrink Small-Outline Package
THS	Total Hold Slack
TNS	Total Negative Slack
VHDL	VHSIC Hardware Description Language
WHS	Worst Hold Slack
WNS	Worst Negative Slack

Úvod

Klesající ceny a pokračující vývoj obvodů FPGA (Field Programmable Gate Array) vedou k čím dál většímu využívání těchto obvodů v praxi. Vzniká tak široké spektrum aplikací vytvářených pomocí jazyků HDL (Hardware Description Language) a následně jejich implementace přímo do obvodů FPGA. Mezi typické aplikace realizované v obvodech FPGA patří například paměťové řadiče, dekodéry, síťové prvky a mnohé další.

Při implementaci rozsáhlých obvodů v FPGA je často nutné použít více hodinových domén, typicky při realizaci standardů Ethernet a PCI Express. Oba uvedené standardy mají pevně definovaný hodinový signál s rozdílnou frekvencí. V takovém případě je nutné, při konverzi dat mezi těmito standardy, bezpečně vyřešit asynchronní přechody signálů mezi jednotlivými hodinovými doménami.

Řešení bezpečných asynchronních přechodů mezi hodinovými doménami je obvykle implementováno odlišně pro jednobitové signály a pro vícebitové signály. Pro jednotlivé typy asynchronních přechodů je dále nutné aplikovat správné omezující podmínky (constraints). Špatně realizovaný asynchronní přechod může vést k poškození či ztrátě přenášené informace.

Tato práce se věnuje stručnému rozboru programovatelných logických obvodů, zejména architektury FPGA, dále jazyku VHDL a vývojovému prostředí Xilinx Vivado Design Suite. Dále je popsána problematika asynchronních přechodů včetně rozboru a popisu nežádoucích jevů a chyb, které mohou u asynchronního přechodu nastat.

V další části práce jsou popsány možnosti implementace jednotek pro řešení jednobitových a vícebitových asynchronních přechodů a popis správné aplikace omezujících podmínek ke každé jednotce. Dále následuje popis realizované knihovny asynchronních přechodů.

V závěru práce je popsána navržená metodika použití implementované knihovny asynchronních přechodů. Praktické využití vytvořené knihovny je následně demonstrováno na případové studii v návrhu síťové karty pro akcelerační kartu COMBO-80G [1].

1 Teoretický rozbor

V následujících kapitolách je vysvětleno, co jsou to programovatelné logické obvody a jaké druhy existují. Podrobněji je rozebrána architektura FPGA. Stručně je popsán jazyk VHDL a používané vývojové prostředí. Dále je teoreticky rozebrána problematika asynchronních přechodů a omezujících podmínek.

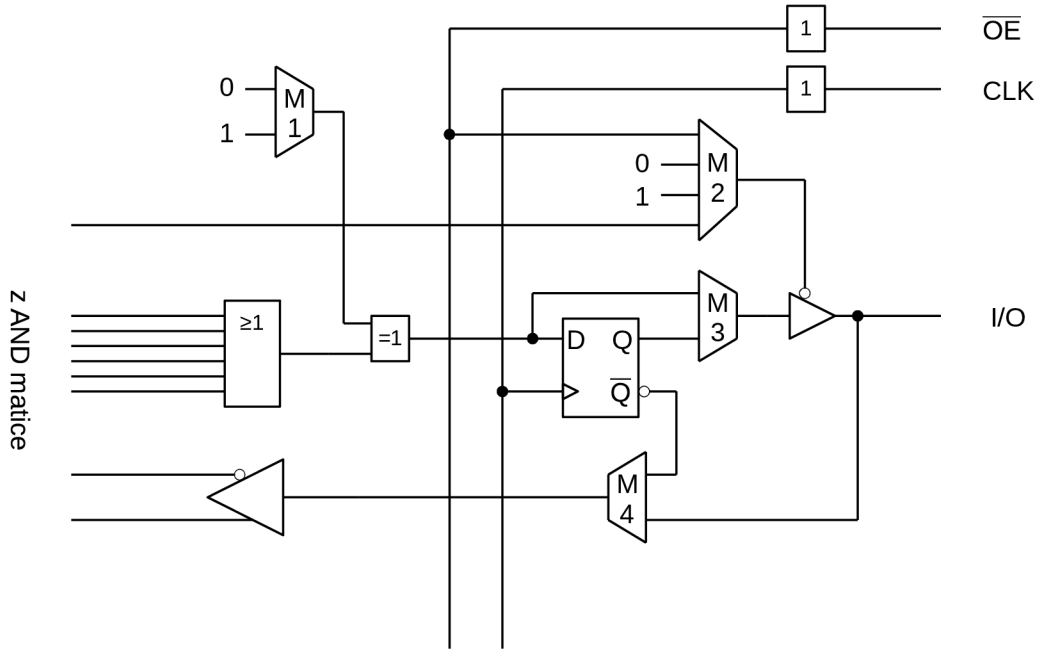
1.1 Programovatelné logické obvody

Programovatelné logické obvody jsou obvody, které lze naprogramovat tak, aby vykonávaly určenou logickou funkci. Existuje několik architektur programovatelných logických obvodů. Mezi tři nejznámější patří:

- SPLD (Simple Programmable Logic Device)
- CPLD (Complex Programmable Logic Device)
- FPGA (Field Programmable Gate Array)

Architektura SPLD (Simple Programmable Logic Device)

Obvody architektury SPLD jsou tvořeny velkou programovatelnou maticí hradel AND a spoustou tzv. makrobuněk, které obsahují programovatelné vícevstupové hradlo OR, programovatelné hradlo XOR, programovatelný klopný obvod a programovatelný vstup/výstupní obvod [2]. Schéma makrobuňky obvodu SPLD je zobrazeno na obrázku 1.



Obr. 1: Makrobuňka obvodu SPLD [2]

Architektura SPLD často používá k uložení konfigurace technologii EEPROM, díky čemuž je možné její konfiguraci smazat a opětovně ji naprogramovat. Počet programovacích cyklů má své limity, nejčastěji 100 až 1000 cyklů. Obvody SPDL dosahují velikostí v rozsahu 8 až 12 makrobuněk. Vyrábí se nejčastěji v pouzdrech typu DIP, PLCC a SSOP. Mezi nejčastější výrobce obvodů architektury SPLD patří firmy Atmel a Lattice Semiconductor. V dnešní době jsou obvody architektury SPLD považovány za zastaralé. [2]

Architektura CPLD (Complex Programmable Logic Device)

Obvody architektury CPLD jsou tvořeny čtyřmi programovatelnými prvky: vstupně-výstupní bloky, výstupním propojovacím polem, propojovacím polem (maticí) a programovatelnými funkčními bloky, které se skládají z programovatelné matice AND a několika makrobuněk s alokátory součinu. [3]

Obvody CPLD dosahují velikostí v rozsahu 32 až 1024 makrobuněk, což odpovídá počtu až 300 000 ekvivalentních hradel. Mezi největší výrobce CPLD obvodů patří firmy Lattice Semiconductor, Altera a Xilinx. Tyto obvody se vyrábějí například v pouzdrech typu PLCC a QFP. Stejně jako architektura SPLD je i CPLD již zastaralá, avšak stále používaná. [2]

1.2 Obvody FPGA

Obvody architektury FPGA typicky tvoří čtyři základní součásti:

- programovatelné logické bloky
- programovatelné vertikální a horizontální propojení
- programovatelné vstupě-výstupní bloky
- specializované bloky (DSP bloky, blokové paměti, …)

Programovatelné logické bloky jsou složeny z náhledových tabulek (LUT), klopného obvodu a lokálního propoje [2]. Zjednodušené schéma programovatelného logického bloku je zobrazeno na obrázku 2.

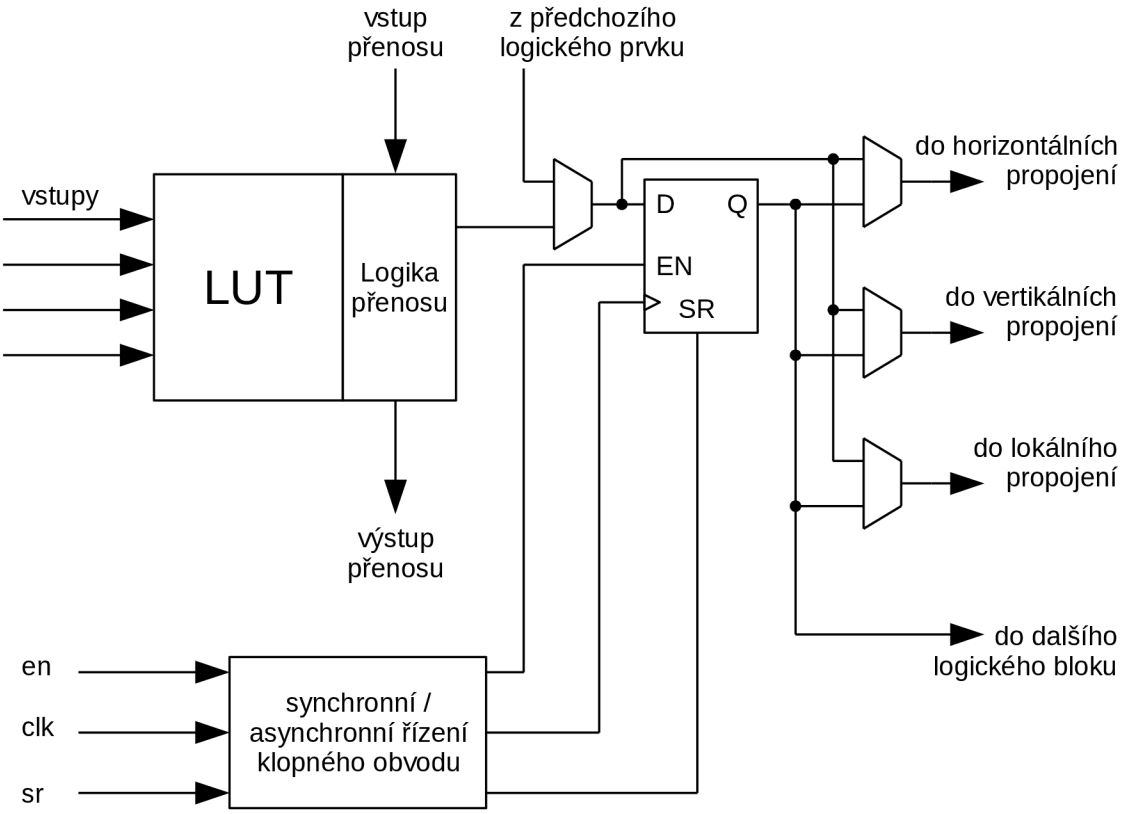
Náhledová tabulka je paměťová buňka, která slouží ke generování logické funkce o čtyřech nebo šesti vstupech a jednom výstupu. Princip fungování je následující, na adresové vodiče paměti jsou přivedeny vstupní proměnné logické funkce. Na výstupu se objeví v paměti uložená výstupní hodnota logické funkce, která odpovídá hodnotě vektoru vstupních proměnných logické funkce. Typická náhledová tabulka disponuje čtyřmi vstupy, ale současné moderní obvody FPGA již obsahují náhledové tabulky se šesti vstupy. Příklad náhledové tabulky s nastavenou konkrétní logickou funkcí je zobrazen na obrázku 3.

Vzhledem k tomu, že náhledové tabulky jsou paměťové buňky, tak je možné je využít také jako paměť. K tomuto účelu jsou náhledové tabulky vybaveny datovým vstupem, hodinovým vstupem a vstupem „povolení zápisu," který povoluje zápis do náhledové tabulky.

Horizontální a vertikální propoje obvodů FPGA slouží k propojení jednotlivých logických bloků mezi sebou, k propojení logických bloků se specializovanými bloky a také k propojení logických bloků se vstupě-výstupními bloky. Tyto propoje jsou rovnoměrně rozloženy po celé ploše obvodu FPGA. Pro hodinové signály jsou vyčleněny speciální propojovací vodiče,

které jsou navrženy tak, aby v různých místech FPGA bylo zpoždění hodinového signálu co nejmenší. [2]

Vstupně-výstupní bloky slouží k řízení přenosu dat mezi vnějšími piny obvodu FPGA a jeho vnitřními bloky. Obvody FPGA umožňují provozovat vstupně-výstupní bloky v různých napěťových úrovní. [3]



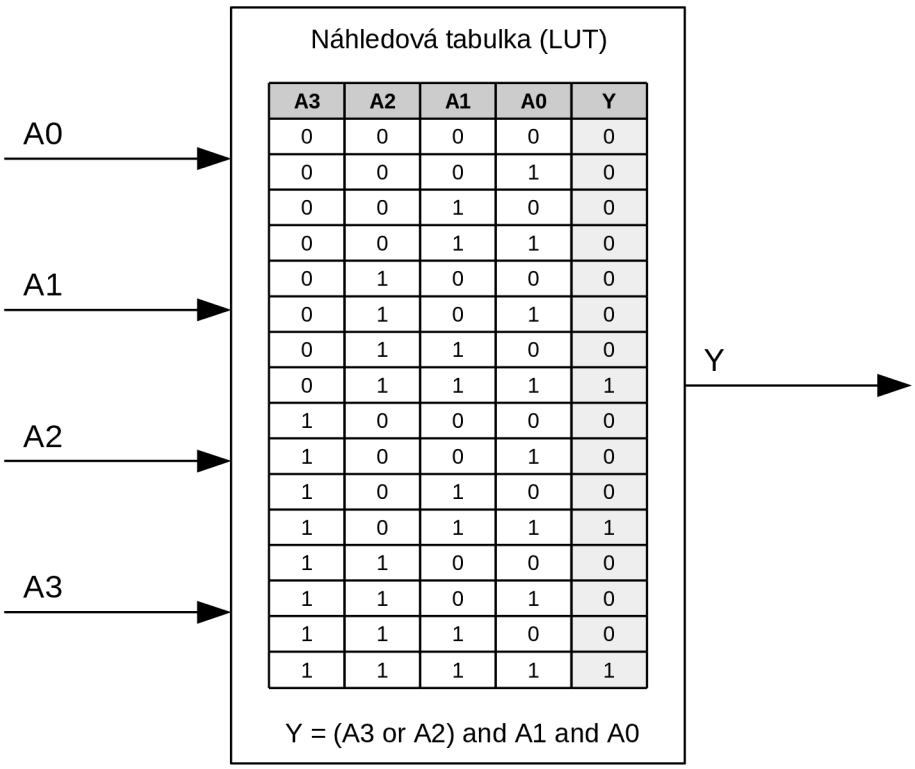
Obr. 2: Zjednodušené schéma programovatelného logického bloku FPGA [2]

Důležitou součástí moderních obvodů FPGA jsou také specializované bloky. Mezi tyto bloky patří nejčastěji: blokové paměti, DSP bloky a fázové závěsy (PLL).

Blokové paměti v moderních obvodech FPGA jsou ve většině případů dvouportové a používají se zejména pro realizaci pamětí, například typů ROM a RAM. Na rozdíl od pamětí vytvořených pomocí náhledových tabulek mají blokové paměti výrazně vyšší kapacitu. Typicky je kapacita blokových pamětí 18 Kb nebo 36 Kb. Datovou šířku blokových pamětí lze nastavit. Běžně povolené hodnoty datové šířky jsou: 1, 2, 4, 8, 16, 32, 64 bitů. Tyto hodnoty datové šířky lze obvykle rozšířit o paritní bity. Blokové paměti lze vzájemně sériově nebo paralelně propojovat, a tím zvýšit hloubku nebo datovou šířku paměti. [3]

DSP (Digital Signal Processing) bloky slouží k rychlému zpracování digitálních signálů. DSP bloky mohou vykonávat velmi mnoho funkcí, násobení, sčítání, odčítání, logické operace a další. Tyto bloky jsou často využívány zejména proto, že jejich použití nahrazuje velké množství programovatelných logických bloků. Další výhoda těchto bloků spočívá v jejich velké rychlosti, která je dána optimální implementací na tranzistorové úrovni. [2]

Fázové závěsy (PLL) umožňují vytvořit ze vstupního hodinového signálu několik různých hodinových signálů s různou fází a frekvencí. [2]



A3	A2	A1	A0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Obr. 3: Příklad náhledové tabulky (LUT) s nastavenou konkrétní logickou funkcí

1.3 Jazyk VHDL a vývojové prostředí Xilinx Vivado Design Suite

VHDL (VHSIC Hardware Description Language) je programovací jazyk určený pro návrh a simulaci digitálních obvodů. Jazyk VHDL byl v roce 1987 vydán jako standard IEEE, známý jako VHDL – 87. Od té doby jazyk VHDL přibližně každých pět let prochází pravidelnými revizemi. I když poslední vydanou revizí je standard VHDL – 2008, drtivá většina návrhových systémů stále podporuje pouze poměrně zastaralou revizi, která je známá jako standard VHDL – 93. [2]

Jazyk VHDL patří do skupiny imperativních strukturovaných jazyků. Značnou výhodou jazyka VHDL jsou prostředky pro popis paralelismu, konektivity a přesné vyjádření času. Kromě běžných stavů log. 1 a log. 0 umí jazyk VHDL také pracovat se stavem vysoké impedance, nedefinovaným stavem a s dalšími speciálními stavy.

Pro potřeby této práce je využíváno vývojové prostředí Vivado Design Suite, které představila společnost Xilinx v roce 2012. Vivado Design Suite se stalo hlavním vývojovým prostředím a nahradilo tak starší vývojové prostředí Xilinx ISE Design Suite. Vivado Design Suite disponuje veškerými potřebnými nástroji pro kompletní návrh, simulaci, syntézu a implementaci digitálního obvodu, například pro obvody FPGA.

1.4 Omezující podmínky

Omezující podmínky (constraints) jsou pokyny pro syntézní a implementační nástroje digitálních obvodů, které tyto nástroje musí splnit. Omezující podmínky ovlivňují různé vlastnosti navrženého digitálního obvodu, například cesty signálů, umístění komponent na čipu, zpoždění signálů a další. Omezující podmínky a způsob jejich zápisu se muže v jednotlivých syntézních a implementačních nástrojích lišit. [4]

Omezující podmínky je možné rozdělit na dvě základní skupiny:

- omezující podmínky aplikované při syntéze
- omezující podmínky aplikované při implementaci

Omezující podmínky se často používají k definování hodin, k nastavení požadovaného zpoždění na vstupech a výstupech, k nastavení minimálního či maximálního zpoždění konkrétních signálů, k ovlivnění rozmístění návrhu na ploše čipu apod.

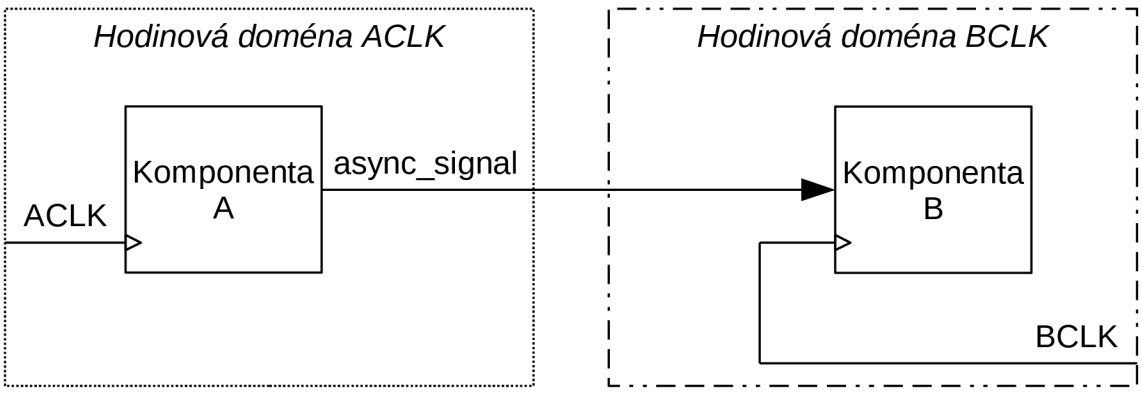
Ve vývojovém prostředí Vivado se omezující podmínky mohou aplikovat pomocí XDC souboru, do kterého se omezující podmínky zapisují. Některé omezující podmínky je možné zapsat i jako atribut přímo do VHDL kódu. [4]

Následuje příklad zápisu omezující podmínky *set_max_delay* pro vývojové prostředí Vivado, která nastaví maximální zpoždění 5 ns na cestu mezi FD1/C a FD2/D:

```
set_max_delay 5 -from [get_pins FD1/C] -to [get_pins FD2/D]
```

1.5 Asynchronní přechod mezi hodinovými doménami

Asynchronní přechod mezi hodinovými doménami lze definovat jako signál nebo skupinu signálů propojující dvě různé hodinové domény, které nejsou vzájemně synchronní, respektive každá doména je řízena vlastním nezávislým hodinovým signálem. [5] Příklad jednoduchého asynchronního přechodu zobrazuje obrázek 4.



Obr. 4: Příklad asynchronního přechodu mezi hodinovými doménami

Pokud není asynchronní přechod správně navržen, tak se můžou v místě asynchronního přechodu objevit různé chyby, jako například ztráta dat, metastabilní stav a další.

Aby tyto chyby nenastaly, je nutné správně navrhnout synchronizační obvody a také je nutné na tyto synchronizační obvody aplikovat správné omezující podmínky. [5]

Vhodný synchronizační obvod je nutné použít i v případě, kdy máme dvě nezávislé hodinové domény, které jsou řízeny hodinovými signály s teoreticky stejnými vlastnostmi (frekvence, fáze). Jelikož je každý z těchto hodinových signálů generován pomocí vlastního oscilátoru, můžou se vlivem parazitních parametrů oscilátorů výsledné vlastnosti hodinových signálů nepatrně lišit. Taková situace může typicky nastat na datové lince, kde probíhá asynchronní přenos dat. [6]

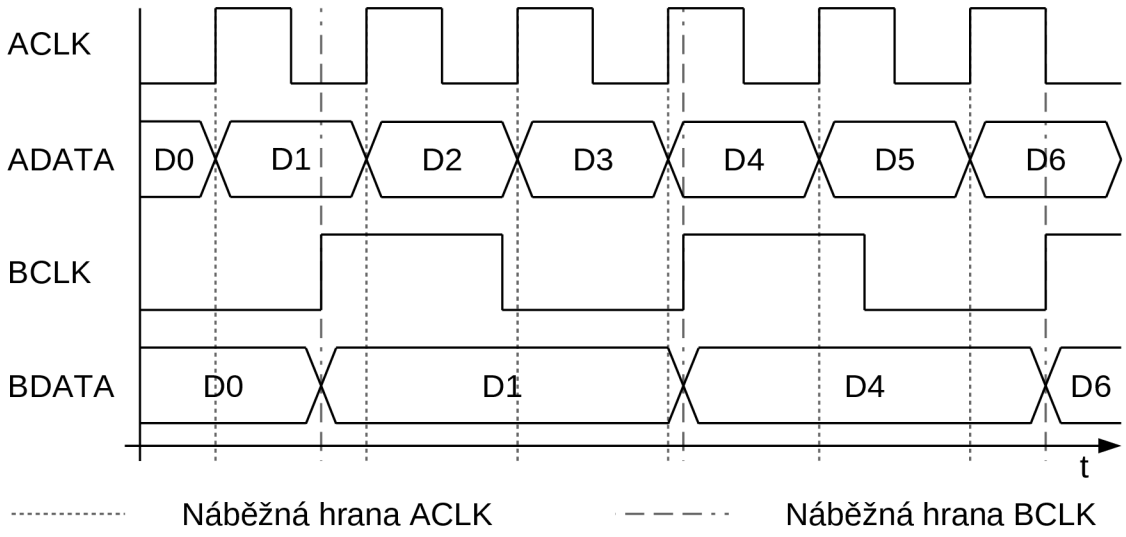
Řešení asynchronních přechodu navíc ztěžuje fakt, že dostupné simulační nástroje digitálních obvodů nedokáží simulovat metastabilní stav. Proto je velmi obtížné ověřit, zda asynchronní přechod funguje správně.

2 Možné chyby v asynchronních přechodech

Při přenosu informací mezi různými hodinovými doménami může typicky dojít k jedné ze čtyř základních chyb, případně i k jejich kombinaci. Při přechodu z rychlé do pomalé hodinové domény se mohou ztratit data. Při přechodu z pomalé do rychlé hodinové domény mohou být data zachycena vícekrát. Rychlou hodinovou doménou se rozumí tak, která má hodinový signál s vyšší frekvencí než hodinový signál z pomalé hodinové domény. V synchronizačním obvodu se může nastat metastabilní stav a nebo může dojít k poškození dat vlivem časového posunu.

2.1 Ztráta dat při přechodu z rychlé do pomalé hodinové domény

Při asynchronním přechodu z rychlé do pomalé hodinové domény může dojít ke ztrátě dat. Tuto situaci ilustruje obrázek 5, kde je vidět, že do pomalé hodinové domény se nepřenášejí všechna data ze signálu ADATA, který je v rychlé hodinové doméně. Datový signál ADATA je vzorkován na náběžnou hranu hodinového signálu BCLK, čímž je rovněž nastaven datový signál BDATA. Ztráta dat je způsobena tím, že vzorkování neprobíhá dostatečně často, a tak jsou některé data na signálu ADATA přenesena dříve, než je stihne zachytit vzorkování. [5]

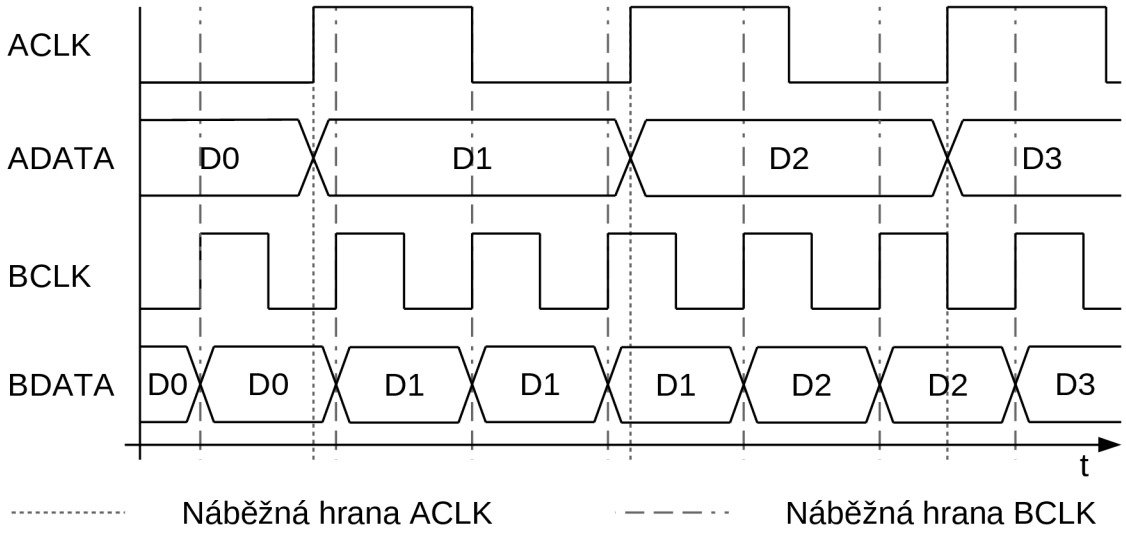


Obr. 5: Ztráta dat při přechodu z rychlé do pomalé hodinové domény

Této chyby se lze vyvarovat například tím, že data rychlé hodinové domény zůstanou zachována v nezměněné podobě minimálně po dobu 1,5 periody pomalého hodinového signálu. Toto řešení se používá zejména v případech, kdy je mezi hodinovými doménami přenášen řídící signál. [5]

2.2 Vícenásobné zachycení dat při přechodu z pomalé do rychlé hodinové domény

Při asynchronním přechodu signálů z pomalé do rychlé hodinové domény může nastat situace, kdy jsou data z pomalé hodinové domény na rychlé hodinové doméně zachycena dvakrát nebo i vícekrát. Tento jev ilustruje obrázek 6, kde je při každé náběžné hraně hodinového signálu BCLK zachycen signál ADATA a podle toho nastaven signál BDATA. Tento jev může být pro některé aplikace nežádoucí. [5]



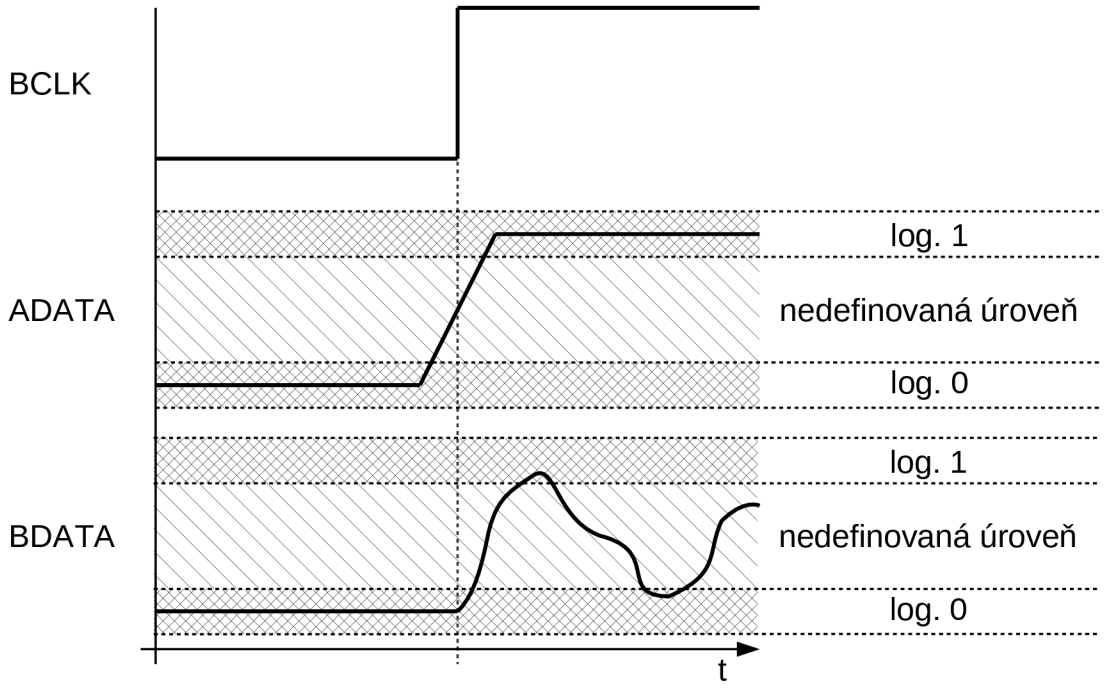
Obr. 6: Vícenásobné zachycení dat při asynchronním přechodu z pomalé do rychlé hodinové domény

2.3 Metastabilní stav

Metastabilní stav je stav obvodu, při kterém výstup obvodu kmitá mezi úrovněmi log. 1 a log. 0. Metastabilita je nežádoucí jev, který je způsobený obvodovou realizací hradel a klopných obvodů, které mají různé parazitní vlastnosti. [5]

Metastabilní stav je znázorněn na obrázku 7, kde můžeme vidět, že datový signál ADATA byl během toho, co se měnil ze stavu log. 0 do stavu log. 1, vzorkován na náběžnou hranu hodinového signálu BCLK. Signál ADATA během své změny procházel úrovní, která není nijak definována, a právě když se nacházel v nedefinované úrovni, byl právě navzorkován. Tím vznikl metastabilní stav, který se projevil na signálu BDATA.

Četnost metastability je možné odhadnout pomocí funkce MTBF (Mean Time Between Failure), což je střední doba mezi chybami [5]. V obvodu, kde je asynchronní signál synchronizován hodinovým signálem CLK lze funkci MTBF určit podle rovnice 1, kde f_{CLK} je frekvence zdrojového hodinového signálu CLK, f_{DATA} je frekvence vstupního asynchronního signálu a t_d je kritická doba, během které hrozí výskyt metastabilního stavu v obvodu [7].



Obr. 7: Znázornění metastabilního stavu

$$MTBF = \frac{1}{f_{DATA} \cdot f_{CLK} \cdot t_d} \tag{1}$$

Pro výpočet funkce MTFB je nutné znát hodnotu doby t_r, což je perioda hodinového signálu CLK zkrácená o dobu předstihu [7]. Výpočet této hodnoty ukazuje rovnice 2. Samotný výpočet funkce MTBF pro jeden konkrétní klopný obvod je uveden v rovnici 3. Pro výpočet byly zvoleny tyto hodnoty: $f_{CLK} = 20\ MHz$, $f_{DATA} = 10\ MHz$. Jako vzorový klopný obvod byl zvolen SN74ALS74, kterému odpovídají tyto parametry: $t_{SU} = 15\ ns$, $T_0 = 8{,}7\ \mu s$, $T = 1\ ns^{-1}$. Parametr t_{SU} udává dobu předstihu, parametr T_0 představuje kritické časové okno pro výskyt metastabilního stavu a parametr T představuje dobu ustálení metastabilního stavu.

$$t_r = \frac{1}{f_{CLK}} - t_{su} = \frac{1}{20 \cdot 10^6} - 15 \cdot 10^{-9} = 35\,ns \tag{2}$$

$$MTBF_{1FF} = \frac{e^{T \cdot t_r}}{f_{DATA} \cdot f_{CLK} \cdot T_0} = \frac{e^{1 \cdot 10^9 \cdot 35 \cdot 10^{-9}}}{10 \cdot 10^6 \cdot 20 \cdot 10^6 \cdot 8{,}7 \cdot 10^{-6}} = 911502\,s \tag{3}$$

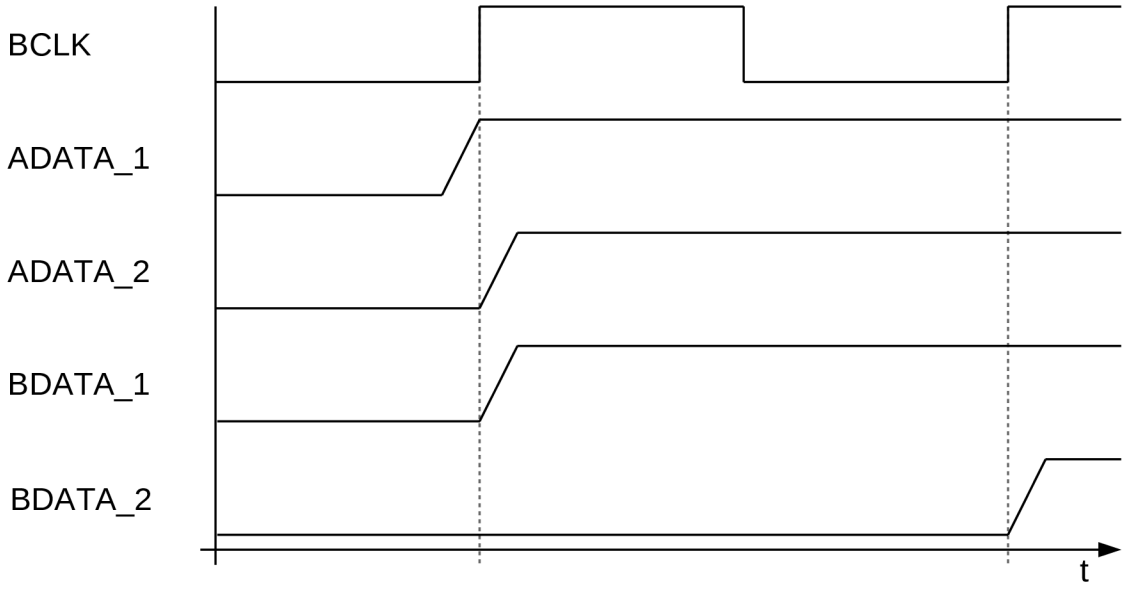
Z rovnice 3 vyplývá, že metastabilní stav nastane přibližně jednou za 911502 s, čili přibližně jednou za 11 dnů. Z rovnice 4 je možné určit hodnotu MTBF pro dva sériově zapojené klopné obvody.

$$MTBF_{2FFs} = \frac{e^{T \cdot t_r}}{\frac{1}{MTBF_{1FF}} \cdot f_{CLK} \cdot T_0} = \frac{e^{1 \cdot 10^9 \cdot 35 \cdot 10^{-9}}}{\frac{1}{911502} \cdot 20 \cdot 10^6 \cdot 8{,}7 \cdot 10^{-6}} = 8{,}31 \cdot 10^{18}\, s \qquad (4)$$

Toto zapojení se používá jako základní synchronizační obvod, který minimalizuje výskyt metastabilního stavu, což dokazuje rovnice 4, ze které vyplývá, že na výstupu druhého klopného obvodu nastane metastabilní stav přibližně jednou za $8{,}31*10^{18}$ s, tedy přibližně jednou za 263 miliard let.

2.4 Vliv malého časového posunu mezi signály na asynchronní přechod

Při řešení asynchronního přechodu více signálů může nastat nežádoucí jev, kdy mezi signály vznikne malý časový posun. Takový jev ilustruje obrázek 8, kde na náběžnou hranu hodinového signálu BCLK vzorkujeme signály ADATA_1 a ADATA_2 a podle nich nastavujeme signály BDATA_1 a BDATA_2.



Obr. 8: Vliv malého časového posunu na asynchronní přechod

Mezi signály ADATA_1 a ADATA_2 je malý časový posun, vlivem čehož nedojde u obou k navzorkování log. 1, ale u signálu ADATA_2 dojde k navzorkování log. 0. Tento nežádoucí jev se následně projeví na signálu BDATA_2. Stručněji řečeno, je navzorkována část starých a část nových hodnot. [6]

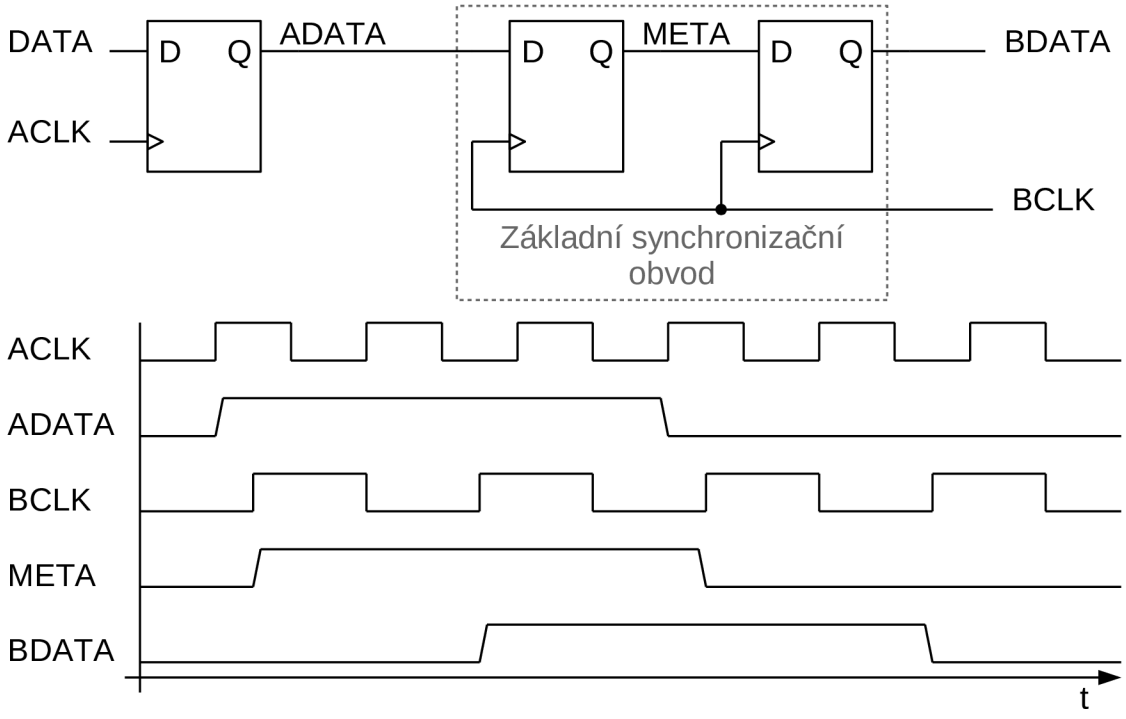
3 Jednotky pro jednobitové asynchronní přechody

V technické praxi jsou využívány dvě varianty jednobitového asynchronního přechodu mezi hodinovými doménami. První možností je použití základního synchronizačního obvodu, které však pro správnou funkčnost vyžaduje dodržení několika podmínek. Druhou možností je synchronizační obvod se zpětnou vazbou, který navíc poskytuje potvrzení o přenosu dat do cílové hodinové domény. Speciálním obvodem je pak synchronizační obvod pro reset.

3.1 Základní synchronizační obvod

Za základní synchronizační obvod pro jednobitový signál lze považovat obvod složený ze dvou za sebou zapojených klopných obvodů typu D, které jsou řízeny na náběžnou hranu cílového hodinového signálu. Toto zapojení synchronizuje vstupní signál do cílové hodinové domény. [5]

Na schématu obvodu z obrázku 9 je navíc zobrazen předřadný klopný obvod typu D. Který odděluje asynchronního přechodu od výstupu kombinační logiky. Pokud by asynchronní přechod nebyl oddělen, mohlo by docházet k průběžným změnám hodnoty signálu a tyto změny by mohly dále vyvolat vznik metastabilních stavů.



Obr. 9: Princip funkce základního synchronizačního obvodu

Pokud první klopný obvod synchronizačního obvodu navzorkuje vstupní signál v průběhu jeho změny, tak se na jeho výstupu objeví metastabilní stav. Z toho důvodu je za ním připojen další klopný obvod, který má za úkol tento stav odstranit a na svůj výstup nastavit korektní

logickou hodnotu. V případě, že obvod běží na vysokých frekvencích, existuje šance, že metastabilní stav projde i přes druhý klopný obvod. Jak často tento jev nastane lze odhadnout podle funkce MTBF. V případě, že obvod provozujeme na vysoké frekvenci je vhodné použít tři namísto dvou klopných obvodů. Tím snížíme pravděpodobnost výskytu metastabilního stavu na výstupu celého synchronizačního obvodu. [5]

Aby při přechodu z rychlejší do pomalejší hodinové domény nedocházelo k chybě popsané v kapitole 2.1, je nutné dodržet podmínku, aby vstupní pulzy byly dlouhé alespoň 1,5 periody cílového hodinového signálu. [5]

Naopak při přechodu z pomalejší do rychlejší hodinové domény je nutné, aby frekvence rychlejšího hodinového signálu byla alespoň 1,5x větší než frekvence pomalejšího hodinového signálu [5]. Nicméně při přechodu z pomalejší do rychlejší hodinové domény se může objevit jev popsaný v kapitole 2.2.

Tento synchronizační obvod by neměl být používán pro přenos vícebitových signálů. Je sice technicky možné poskládat paralelně větší počet těchto obvodů tak, aby každý jeden obvod přenášel jeden bit vícebitového signálu, ale v takovém případě hrozí poškození dat vlivem jevu popsaného v kapitole 2.4. Toto paralelní zapojení můžeme použít pouze ve výjimečném případě, kdy přenášíme data v Grayově kódu, která mohou zvyšovat nebo zmenšovat svou hodnotu pouze o jedničku. Tím je zaručeno, že se vždy změní pouze hodnota jednoho bitu. Často se tento způsob využívá k synchronizaci výstupu z čítače, který je v Grayově kódu. Zmíněné řešení se typicky používá u asynchronní paměti FIFO. [8]

3.1.1 Aplikace omezujících podmínek

Na základní synchronizační obvod je nutné aplikovat několik omezujících podmínek. Nejprve je nutné zajistit, aby se při implementaci obvodu použily klopné obvody a nikoliv posuvný registr. Při používání vývojového prostředí Vivado to lze zajistit nastavením atributu *shreg_extract* na hodnotu *„no“*. Dále je nutné informovat Vivado o tom, že konkrétní propojené klopné obvody budou asynchronní. Toho lze docílit nastavením atributu *async_reg* na hodnotu *„true“*. [9] Vivado díky této informaci umístí klopné obvody v FPGA tak, aby vzdálenost mezi nimi byla co nejmenší. Tím se zajistí malé zpoždění mezi nimi a sníží se pravděpodobnost výskytu metastabilního stavu na výstupu. Atributy *shreg_extract* a *async_reg* lze aplikovat přímo pomocí VHDL popisu nebo v prostředí Vivado.

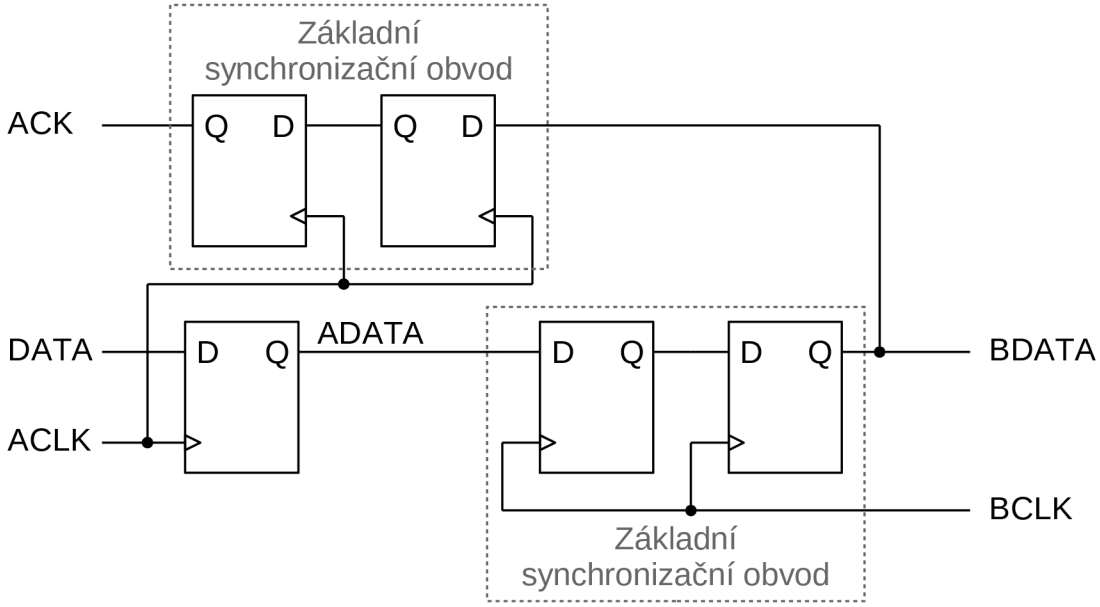
Nakonec je nutné zajistit, aby Vivado neanalyzovalo časování na datové cestě z předřadného klopného obvodu do prvního synchronizačního klopného obvodu. To lze zajistit aplikováním výjimky z časové analýzy *set_false_path* na cestu z předřadného klopného obvodu do prvního synchronizačního klopného obvodu. [4]

Pokud je tento synchronizační obvod používán pro přenos vícebitových dat v Grayově kódu, která mohou zvyšovat nebo zmenšovat svou hodnotu pouze o jedničku, je navíc nutné aplikovat časovou podmínku *set_max_delay -datapath_only*. Tuto podmínku je nutné aplikovat s hodnotou menší nebo rovnou jedné periodě hodinového signálu ze zdrojové hodinové domény a to na cestu z prvního do druhého synchronizačního klopného obvodu. [8]

3.2 Synchronizační obvod se zpětnou vazbou

Synchronizační obvod se zpětnou vazbou je především složen ze dvou základních synchronizačních obvodů. Pomocí prvního synchronizačního obvodu se vstupní signál synchronizuje do cílové hodinové domény a pomocí druhého synchronizačního obvodu

se synchronizuje zase zpět do zdrojové hodinové domény. Signál, který je synchronizován zpět do zdrojové hodinové domény, pak slouží jako potvrzení toho, že došlo k úspěšné synchronizaci do cílové hodinové domény. Nevýhodou tohoto řešení je zpoždění potvrzení synchronizace [5]. Zapojení jednobitového synchronizačního obvodu se zpětnou vazbou zobrazuje obrázek 10.



Obr. 10: Jednobitový synchronizační obvod se zpětnou vazbou

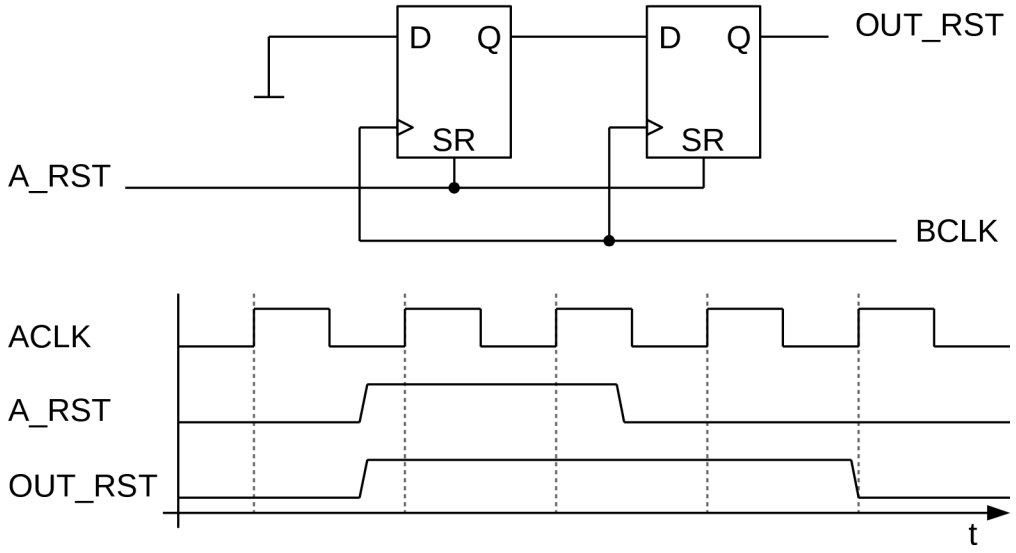
Pokud by bylo toto zapojení provozováno na vysokých frekvencích, je opět vhodné použít tři namísto dvou klopných obvodů u obou základních synchronizačních obvodů, ze kterých se zapojení synchronizačního obvodu se zpětnou vazbou skládá.

3.2.1 Aplikace omezujících podmínek

Protože se zapojení jednobitového synchronizačního obvodu se zpětnou vazbou skládá ze dvou základních synchronizačních obvodů, stačí, aby byly správně aplikovány omezující podmínky právě na tyto dva obvody. Jak správně aplikovat omezující podmínky na základní synchronizační obvod je popsáno v kapitole 3.1.1.

3.3 Synchronizační obvod pro reset

Synchronizační obvod pro reset je složen ze dvou nebo tří klopných obvodů. Tento obvod slouží k synchronizaci reset signálu. Zapojení a funkci tohoto obvodu ilustruje obrázek 11. Vstup prvního klopného obvodu je připojen na log. 0. Pokud je vstupní reset signál neaktivní (log. 0), je i výstup obvodu v log. 0. Jakmile se vstupní reset signál nastaví, tak se i všechny klopné obvody ihned přenastaví do log. 1. Funkcí obvodu je ihned propagovat vstupní reset na výstup, ale jeho konec (sestupnou hranu) zarovnat na náběžnou hranu hodinového signálu cílové hodinové domény. Pokud obvod provozujeme na vysoké frekvenci, je vhodné použít tři namísto dvou klopných obvodů.



Obr. 11: Zapojení a princip funkce synchronizačního obvodu pro reset

3.3.1 Aplikace omezujících podmínek

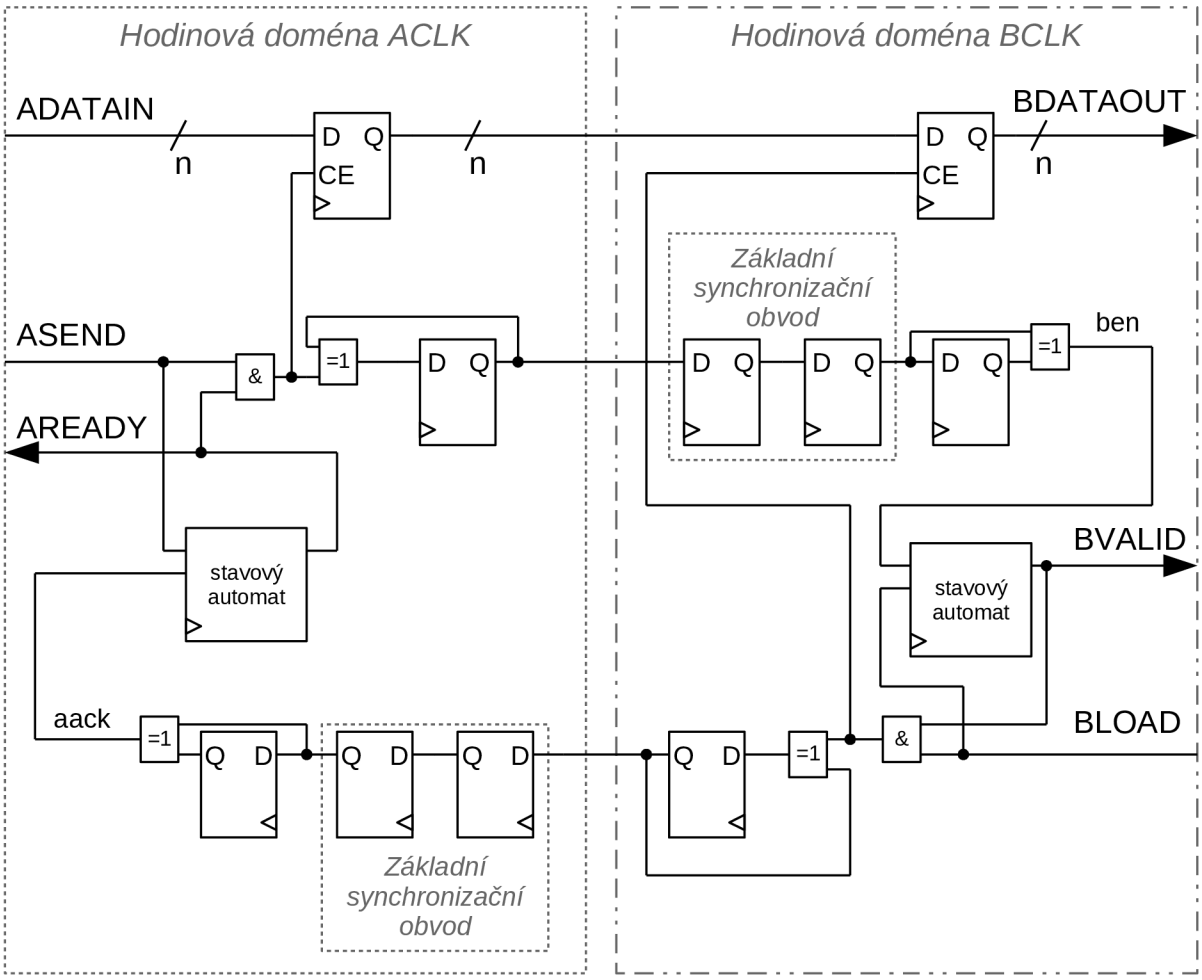
První omezující podmínka, kterou je nutné aplikovat na tento obvod, je výjimka z časové analýzy pro cestu procházející vstupním portem A_RST. Ve vývojovém prostředí Vivado toho lze docílit aplikováním příkazu *set_false_path*. Dále je nutné zamezit nechtěným optimalizací klopných obvodů, které může Vivado provést. Toho lze docílit nastavením atributu *dont_touch* na hodnotu *„true“*. [4] A stejně jako u základního synchronizačního obvodu je ještě nutné aplikovat atributy *shreg_extract* a *async_reg* tak, jak je to popsáno v kapitole 3.1.1.

4 Jednotky pro vícebitové asynchronní přechody

V technické praxi jsou využívány dvě základní metody pro řešení vícebitových asynchronních přechodů. První možností, jak synchronizovat vícebitový datový signál, je využít metody handshake. Druhou možností synchronizace vícebitového signálu, je použití asynchronní paměti FIFO.

4.1 Synchronizační obvod využívající handshake metody

Synchronizační obvod využívající metody handshake bude spolehlivý, ale také poměrně pomalý. Proto se toto řešení příliš nepoužívá v rychlých datových sběrnicích [5].



Obr. 12: Schéma synchronizačního obvodu využívajícího handshake metody

Tento synchronizační obvod je zobrazen na obrázku 12. Obvod funguje následovně: když je výstupní signál AREADY nastaven do log. 1, znamená to, že synchronizační obvod je připraven přijmout data. V takovém případě lze na vstup ADATAIN vložit přenášená data. Při nastavení dat je zároveň nutné nastavit vstupní signál ASEND do log. 1, který znamená,

že data na vstupu ADTAIN jsou platná. Nastavením signálu ASEND zároveň dojde k nastavení signálu AREADY do log. 0.

Následně se data a řídící pulz začnou přenášet do cílové hodinové domény. Jakmile je řídící pulz přenesen, nastaví se výstupní signál BVALID do log. 1. To znamená, že data jsou připravena na vyčtení a nastavením vstupního signálu BLOAD do log. 1 se přenášená data vyčtou na výstup BDATAOUT. Nastavení signálu BLOAD zároveň vyvolá přenos dalšího řídícího pulzu do zdrojové hodinové domény. Jakmile je tento pulz přenesen, dojde k nastavení signálu AREADY do log. 1 a celý proces může začít znovu. Nastavování signálů AREADY a BVALID řídí dva stavové automaty. Diagramy těchto stavových automatů jsou znázorněny na obrázku 13.

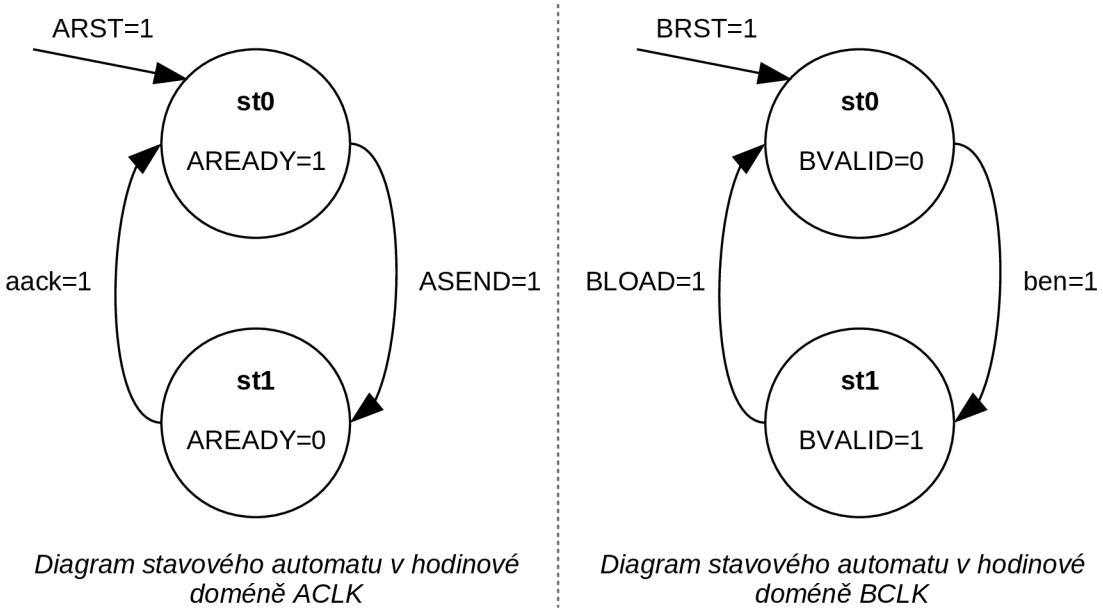


Diagram stavového automatu v hodinové doméně ACLK

Diagram stavového automatu v hodinové doméně BCLK

Obr. 13: Diagramy stavových automatů synchronizačního obvodu „handshake“

4.1.1 Aplikace omezujících podmínek

Synchronizační obvod využívající metody handshake obsahuje základní synchronizační obvody jako podkomponenty. Na tyto dva základní synchronizační obvody je nutné aplikovat omezující podmínky tak, jak to popisuje kapitola 3.1.1.

Dále je u tohoto obvodu nutné aplikovat výjimku z časové analýzy *set_max_delay -datapath_only* na cestu mezi dvěma vícebitovými klopnými obvody, které slouží pro přenos dat. Tuto podmínku je nutné nastavit s hodnotou, která je rovná jedné periodě hodinového signálu z cílové hodinové domény. [8]

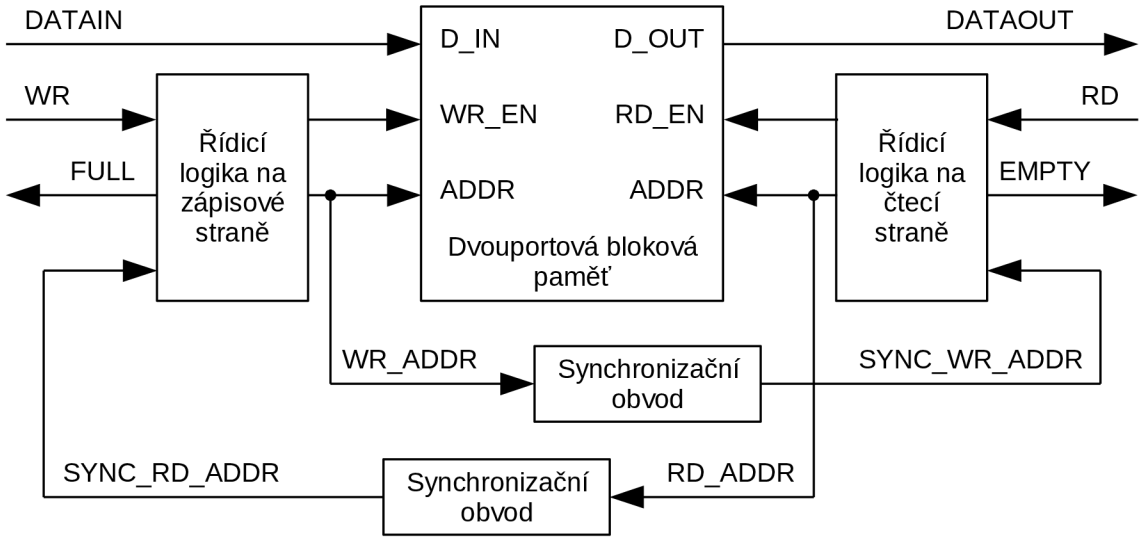
4.2 Synchronizační obvod využívající asynchronní FIFO

Synchronizační obvod využívající asynchronní paměť FIFO je na rozdíl od handshake metody podstatně rychlejší, proto je vhodné ho použít na rychlé datové sběrnice. Pro přenos dat se využívá asynchronní paměti FIFO, která přímo oddělí jednotlivé hodinové domény. [5]

Asynchronní FIFO lze v současných obvodech FPGA implementovat poměrně jednoduše za pomoci dvouportových blokových pamětí. Každý port blokové paměti dokáže pracovat s vlastním nezávislým hodinovým signálem. Pro použití blokové paměti jako asynchronní FIFO je nutné k ní doplnit řídicí logiku s generátory příznaků plné a prázdné paměti.

Obvody FPGA řady 7 Series od společnosti Xilinx obsahují moderní blokové paměti, které již potřebnou logiku obsahují, a díky tomu mohou pracovat přímo v režimu asynchronní paměti FIFO [10].

Pokud blokové paměti potřebnou řídicí logiku nemají, tak je nutné ji implementovat. Generování příznaku plné paměti probíhá ve zdrojové hodinové doméně a generování příznaku prázdné paměti probíhá v cílové hodinové doméně. Aby generování obou příznaků mohlo fungovat, je nutné synchronizovat zapisovací a čtecí adresu mezi oběma hodinovými doménami. Blokové schéma tohoto synchronizačního obvodu, který využívá asynchronní paměť FIFO, zobrazuje obrázek 14.



Obr. 14: Blokové schéma synchronizačního obvodu využívajícího asynchronní FIFO

Synchronizaci adresových signálů lze zajistit pomocí synchronizačního obvodu využívajícího metodu handshake, který je popsán v kapitole 4.1. Druhou možností je převést zapisovací a čtecí adresy do Grayova kódu a následně provést synchronizaci adres pomocí základního synchronizačního obvodu v paralelním zapojení, který je popsán v kapitole 3.1.

4.2.1 Aplikace omezujících podmínek

Jestliže je k synchronizaci zapisovacích a čtecích adres využíván synchronizační obvod využívající metodu handshake, je nutné aplikovat požadované omezující podmínky podle popisu uvedeného v kapitole 4.1.1.

Pokud jsou zápisové a čtecí adresy převedeny do Grayova kódu a pro jejich synchronizaci je využit základní synchronizační obvod v paralelním zapojení, je nutné aplikovat omezující podmínky podle kapitoly 3.1.1.

5 Realizace knihovny asynchronních přechodů

Knihovna asynchronních přechodů, vytvořená v rámci řešení práce, obsahuje obecné synchronizační obvody a z nich odvozené synchronizační obvody pro proprietární datové sběrnice MI32, FL, FLU a DMA. Tyto proprietární datové sběrnice jsou využívány sdružením CESNET. Obecné synchronizační obvody byly podrobeny simulaci. Proprietární datové sběrnice MI32, FL, FLU a DMA prošly již připravenou verifikací.

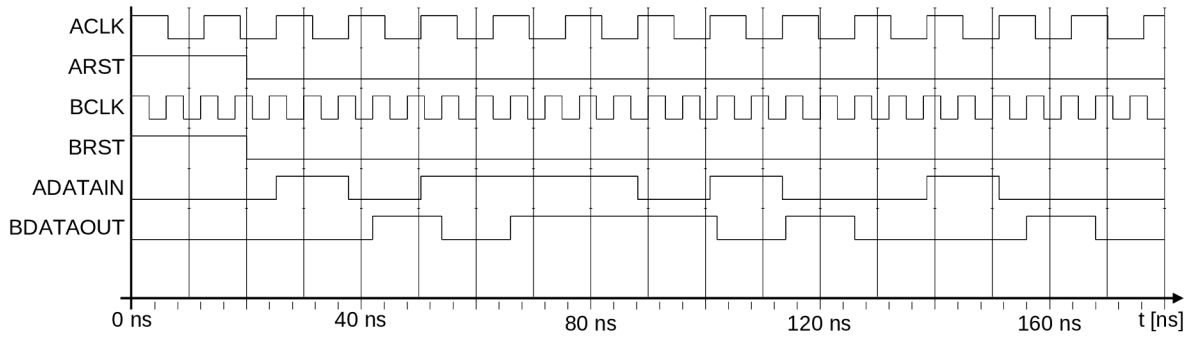
Knihovna asynchronních přechodů byla vytvořena tak, aby bylo její použití velmi snadné. Ke každému implementovanému synchronizačnímu obvodu je přiložený XDC soubor, který obsahuje požadované omezující podmínky. Pokud chce návrhář digitálních obvodů použít nějaký synchronizační obvod z knihovny asynchronních přechodů tak stačí, aby vybraný obvod použil na zvolené místo. Jakmile je celý návrh načten do vývojového prostředí Vivado, načtou se i omezující podmínky k použitým synchronizačním obvodům. K tomu je využíván mechanismus constraint scoping, který umožňuje automaticky aplikovat zvolené omezující podmínky na všechny komponenty s konkrétním názvem entity v celém návrhu digitálního obvodu bez znalostí a ovlivnění vyšších úrovní návrhu [4].

5.1 Obecné synchronizační obvody

V rámci knihovny asynchronních obvodů bylo realizováno celkem osm obecných synchronizačních obvodů. Tyto realizované synchronizační obvody mohou být použity pro řešení asynchronních přechodů u řídicích signálů, reset signálů a vícebitových signálů.

5.1.1 ASYNC_OPEN_LOOP a ASYNC_OPEN_LOOP_SMD

Základní synchronizační obvod byl implementován tak, že lze parametricky nastavit, zda budou použity dva nebo tři synchronizační klopné obvody. Dále lze parametricky zapnout či vypnout předřadný vstupní klopný obvod. Tento obvod byl implementován ve dvou variantách, které se liší připravenými omezujícími podmínkami. Varianta tohoto obvodu, kterou lze použít pro synchronizaci jednotlivých signálů vícebitového Grayova kódů, je označena názvem entity ASYNC_OPEN_LOOP_SMD. Druhá varianta pro běžné použití má název entity ASYNC_OPEN_LOOP.

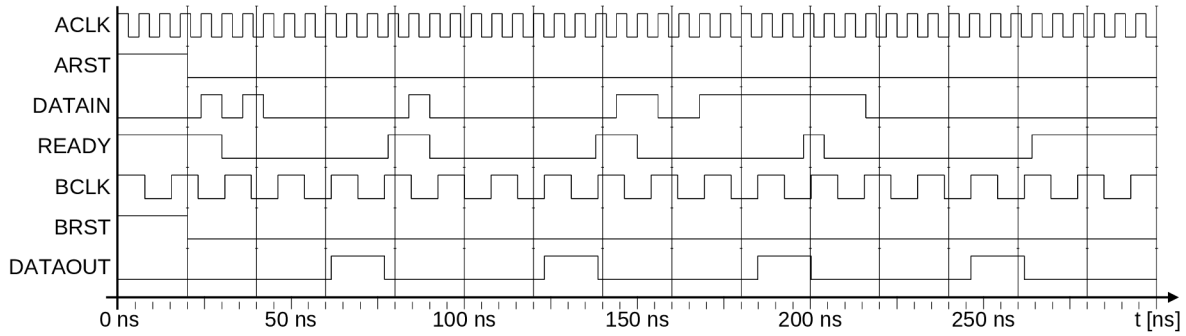


Obr. 15: Simulace základního synchronizačního obvodu (ASYNC_OPEN_LOOP)

Obrázek 15 zobrazuje výstup simulace obvodu ASYNC_OPEN_LOOP při parametricky vypnutém předřadném klopném obvodu a s parametricky nastavenými třemi synchronizačními klopnými obvody. V simulace je možné vidět, že na vstup ADATAIN postupně přichází kratší pulz, pak delší pulz a následně zase dva krátké pulzy. Tyto pulzy jsou synchronizovány do hodinové domény BCLK a na výstupu BDATAOUT se objevují se zpožděním dvou taktů od zachycení na náběžnou hranu hodinového signálu BCLK.

5.1.2 ASYNC_GENERAL

Synchronizační obvod se zpětnou vazbou byl implementován s názvem entity ASYNC_GENERAL. Je složen ze dvou synchronizačních obvodů ASYNC_OPEN_LOOP. Opět lze parametricky nastavit, zda budou používány dva nebo tři synchronizační klopné obvody. Dále lze parametricky nastavit, zda bude obvod zachytávat log. 1, nástupnou hranu, sestupnou hranu či nástupnou i sestupnou hranu. Na výstupu obvodu jsou pulzy o šířce jedné periody cílové hodinové domény. Jeden výstupní pulz odpovídá jednomu zachycení například nástupné hrany.



Obr. 16: Simulace synchronizačního obvodu se zpětnou vazbou (ASYNC_GENERAL)

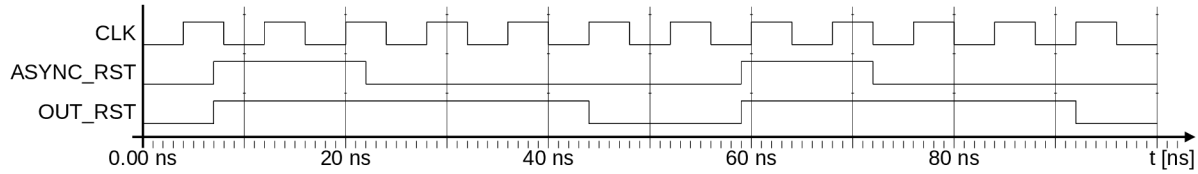
Obrázek 16 zobrazuje výstup simulace obvodu ASYNC_GENERAL při parametricky nastaveném zachytávání log. 1 a s parametricky nastavenými třemi synchronizačními klopnými obvody. Tyto parametrické volby jsou pro tento obvod výchozí. Na simulaci lze vidět několik různě dlouhých pulzů, které postupně přicházejí na vstup DATAIN. Pokud je signál READY v log. 1, tak jsou tyto pulzy zachyceny a na výstupu DATAOUT se ke každému zachycení vygeneruje pulz o šířce jedné periody hodinového signálu BCLK.

5.1.3 ASYNC_RESET

Synchronizační obvod s název entity ASYNC_RESET je určen pro řešení asynchronního přechodu reset signálu. Princip funkce tohoto obvodu byl popsán v kapitole 3.3. Opět lze parametricky nastavit, zda budou používány dva nebo tři synchronizační klopné obvody. Dále lze parametricky aktivovat výstupní klopný obvod, a pokud je aktivní tak, lze parametricky výstupní klopný obvod replikovat. Každý z replikovaných klopných obvodů lze umístit do jiné části FPGA, a tím umožnit rozšíření výstupního reset signálu do více částí FPGA s menším zpožděním signálu.

Obrázek 17 zobrazuje výstup simulace obvodu ASYNC_RESET při výchozím nastavení parametrických vlastností, tedy s nastavenými třemi synchronizačními klopnými obvody a s deaktivovaným výstupním klopným obvodem a jeho replikováním. Ze simulace vyplývá, jak na vstup ASYNC_RST postupně přichází dva asynchronní resetovací pulzy. Tyto pulzy

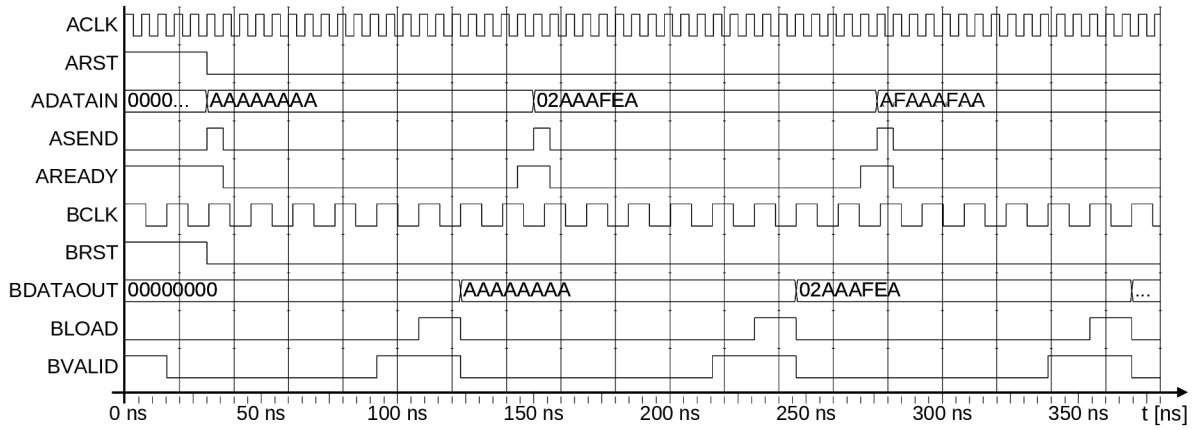
jsou okamžitě generovány i na výstup, ale konec těchto pulzů je prodloužen přibližně o dva takty a zarovnán na náběžnou hranu hodinového signálu CLK.



Obr. 17: Simulace synchronizačního obvodu pro reset (ASYNC_RESET)

5.1.4 ASYNC_BUS_HANDSHAKE

Vícebitový synchronizační obvod využívající metody handshake byl implementován a označen názvem entity ASYNC_BUS_HANDSHAKE. Princip funkce tohoto obvodu byl popsán v kapitole 4.1. U tohoto obvodu je možné parametricky nastavit datovou šířku a opět i počet synchronizačních klopných obvodu ve dvou obvodech ASYNC_OPEN_LOOP, které jsou součástí celého tohoto obvodu.



Obr. 18: Simulace synchronizačního obvodu využívajícího metody handshake (ASYNC_BUS_HANDSHAKE)

Obrázek 18 zobrazuje výstup simulace obvodu ASYNC_BUS_HANDSHAKE při výchozím nastavení parametrických vlastností, tedy s parametricky nastavenými třemi synchronizačními klopnými obvody a s parametricky nastavenou datovou šířkou 32 bitů. V simulací je vidět, jak po resetu jsou na vstup ADATAIN přivedena vícebitová data a zároveň je nastaven signál ASEND do log. 1. Jelikož je do log. 1 nastaven i signál AREADY, tak ihned započne synchronizace a přenos dat ze vstupu ADATAIN na výstup BDATAOUT. Po určité době, potřebné k synchronizaci dat, se do log. 1 nastaví signál BVALID, což znamená, že přenesená data jsou připravena k vyčtení na výstup BDATAOUT. K vyčtení dat na výstup BDATAOUT dojde takt po nastavení signálu BLOAD do log. 1.

5.1.5 ASFIFO, ASFIFO_BRAM a ASFIFO_BRAM_7SERIES

Vícebitový synchronizační obvod využívající asynchronní paměť FIFO, jehož princip byl popsán v kapitole 4.2, byl implementován celkem ve třech variantách. Jednotlivé varianty se od sebe liší především použitým typem asynchronní paměti FIFO. První varianta s názvem entity ASFIFO využívá asynchronní paměti FIFO sestavené z distribuované paměti.

Druhá varianta s názvem entity ASFIFO_BRAM využívá asynchronní paměti FIFO sestavené z blokové paměti. Třetí varianta s názvem entity ASFIFO_BRAM_7SERIES využívá blokovou paměť v režimu asynchronní paměti FIFO, která je k dispozici v obvodech FPGA řady 7 Series od společnosti Xilinx.

5.2 Synchronizační obvody pro proprietární datové sběrnice

V rámci knihovny asynchronních přechodů bylo implementováno třináct vícebitových synchronizačních obvodů pro proprietární sběrnice využívané sdružením CESNET. Konkrétně jde o datové sběrnice MI32, FL, FLU a DMA. Pro každou z těchto sběrnic byly implementovány tři varianty synchronizačního obvodu, které jako základní podkomponentu používají synchronizační obvody založené na asynchronní paměti FIFO. První varianta proprietárních sběrnic využívá synchronizační obvod ASFIFO, druhá varianta ASFIFO_BRAM a třetí varianta ASFIFO_BRAM_7SERIES. U datové sběrnice MI32 byla implementována navíc ještě čtvrtá varianta, která je postavená na synchronizačním obvodu ASYNC_BUS_HANDSHAKE. Tabulka 1 zobrazuje kompletní přehled realizovaných synchronizačních obvodů pro proprietární sběrnice MI32, FL, FLU a DMA.

Název datové sběrnice	Název entity synchronizačního obvod	Využívaný obecný synchronizační obvod
MI32	MI32_ASYNC_FIFO	ASFIFO
	MI32_ASYNC_FIFO_BRAM	ASFIFO_BRAM
	MI32_ASYNC_FIFO_BRAM_7SERIES	ASFIFO_BRAM_7SERIES
	MI32_ASYNC_HANDSHAKE	ASYNC_BUS_HANDSHAKE
FL	FL_ASFIFO_LUT	ASFIFO
	FL_ASFIFO_BRAM	ASFIFO_BRAM
	FL_ASFIFO_7SERIES	ASFIFO_BRAM_7SERIES
FLU	FLU_ASFIFO_LUT	ASFIFO
	FLU_ASFIFO	ASFIFO_BRAM
	FLU_ASFIFO_7SERIES	ASFIFO_BRAM_7SERIES
DMA	DMA_ASFIFO_LUT	ASFIFO
	DMA_ASFIFO_BRAM	ASFIFO_BRAM
	ASFIFO_DMA_BUS	ASFIFO_BRAM_7SERIES

Tab. 1: Přehled realizovaných synchronizačních obvodů pro proprietární sběrnice MI32, FL, FLU a DMA, včetně jimi využívaného obecného synchronizačního obvodu.

Tyto synchronizační jednotky pro proprietární sběrnice MI32, FL, FLU a DMA již vlastní omezující podmínky v přiloženém souboru XDC nemají. To je dáno tím, že tyto synchronizační obvody jsou realizovány jako obálky obecných synchronizačních obvodů, které potřebné omezující podmínky již mají a aplikují je zcela automaticky.

6 Metodika použití asynchronních přechodů

Jedním z cílů této práce bylo vytvořit metodiku použití asynchronních přechodů, která usnadní jejich používání návrhářům digitálních obvodů. Způsobů, jak vytvořit vhodnou metodiku, je několik. Například bylo možné vytvořit podrobnou technickou dokumentaci použití asynchronních přechodů. Cílem však nebylo přidat vývojářům digitální obvodů další obsáhlou dokumentaci ke čtení, ale zefektivnit a zjednodušit jejich práci. Proto byla metodika použití asynchronních přechodů vypracována v grafické formě jako rozhodovací strom.

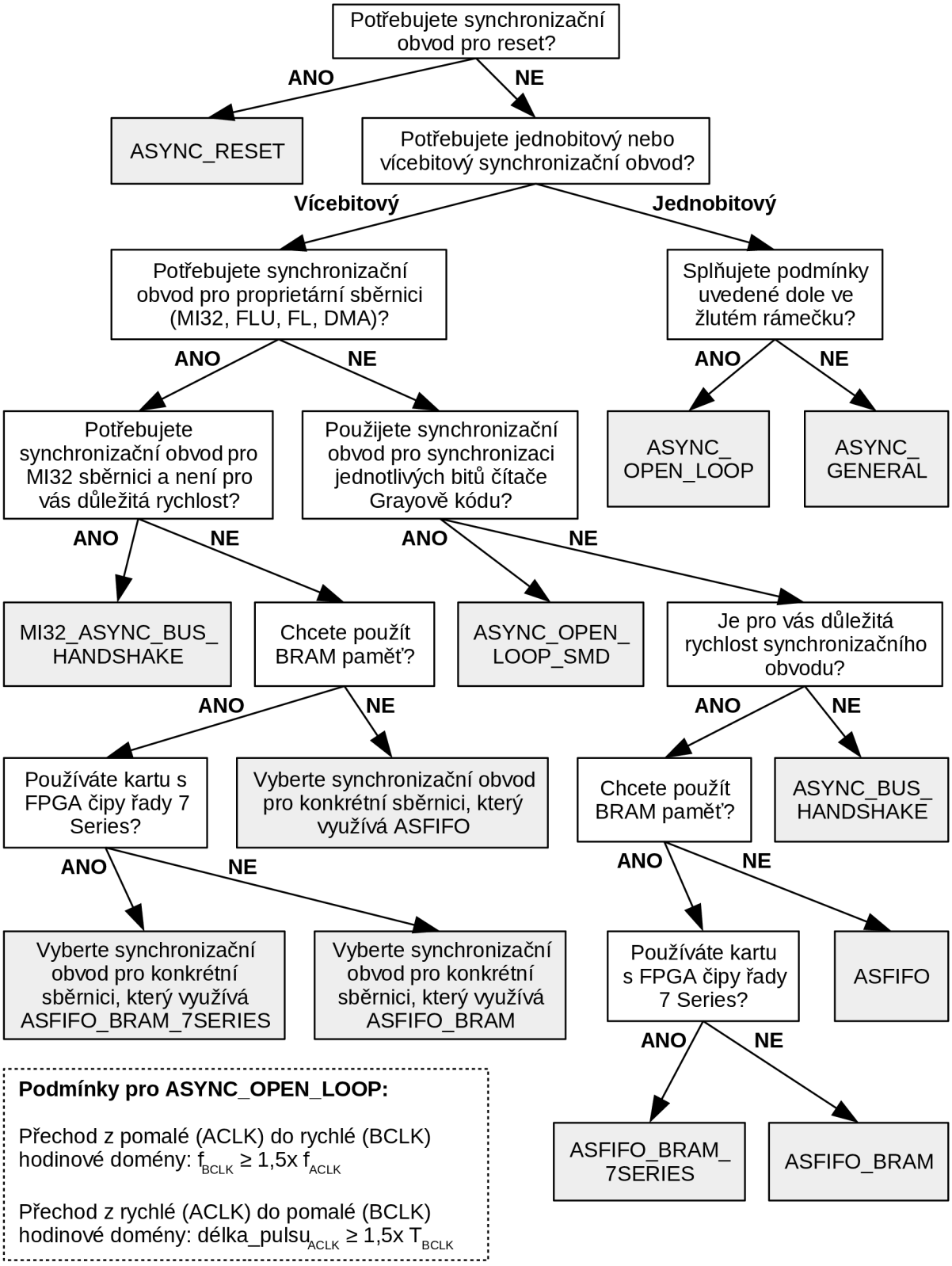
6.1 Rozhodovací strom a použití vybraného synchronizačního obvodu ve VHDL

Grafický rozhodovací strom je zobrazen na obrázku 19. Tento rozhodovací strom obsahuje takové otázky, aby bylo možné konkrétně určit vhodný synchronizační obvod na ošetření požadovaného asynchronního přechodu v obvodu. Rozhodovací strom byl vytvořen tak, aby jej bylo možno vytisknout na papír formátu A4.

Při použití rozhodovacího stromu návrhář digitálních obvodů projde otázky obsažené v rozhodovacím stromě a na konci získá informaci o tom, jaký použít synchronizační obvod z knihovny asynchronních přechodů, která byla popsána v kapitole 5.

Jakmile je zvolen vhodný synchronizační obvod, je nutné ho správně použít v návrhu celého obvodu. Použití synchronizačních obvodů z knihovny asynchronních přechodů je velice jednoduché. Zvolený synchronizační obvod stačí vložit do VHDL kódu, stejně jako jakoukoli jinou komponentu. Následující příklad ukazuje použití synchronizačního obvodu z realizované knihovny asynchronních přechodů ve VHDL kódu.

```
architecture full of obvod_abc is
begin
      muj_sync_obvod: entity work.ASYNC_OPEN_LOOP
      generic map(
            IN_REG  => false,  -- vypnutý vstupní klopný obvod
            TWO_REG => false   -- tři synchronizační klopné obvody
      )
      port map(
            ACLK     => ACLK,  -- hodinový signál ACLK
            BCLK     => BCLK,  -- hodinový signál BCLK
            ARST     => ARST,  -- reset signál z hod. domény ACLK
            BRST     => BRST,  -- reset signál z hod. domény BCLK
            ADATAIN  => adata, -- vstupní signál, hod. doména ACLK
            BDATAOUT => bdata  -- výstupní signál, hod. doména BCLK
      );
```



Obr. 19: Rozhodovací strom usnadňující výběr synchronizačního obvodu

7 Případová studie použití asynchronních přechodů v návrhu síťové karty pro akcelerační kartu COMBO-80G

V rámci řešení projektu byla provedena případová studie použití asynchronních přechodů v návrhu síťové karty pro akcelerační kartu COMBO-80G. Cílem bylo analyzovat stav asynchronních přechodů v návrhu sítové karty a špatně ošetřené asynchronní přechody opravit s využitím již připravených jednotek z knihovny asynchronních přechodů. Na závěr případové studie byly shrnuty dosažené výsledky a zhodnocen celkový přínos úprav asynchronních přechodů.

7.1 Analýza asynchronních přechodů

Návrh sítové karty pro akcelerační kartu COMBO-80G existuje ve dvou variantách. První varianta se označuje 10G8 a umožňuje komunikaci 8 x 10 Gb/s. Druhá varianta se označuje 40G2 a umožňuje komunikaci 2 x 40 Gb/s. Analýza obou variant návrhů sítové karty pro akcelerační kartu COMBO-80G ukázala, že v obou variantách návrhu síťové karty je přibližně desítka špatně ošetřených či neošetřených asynchronních přechodů.

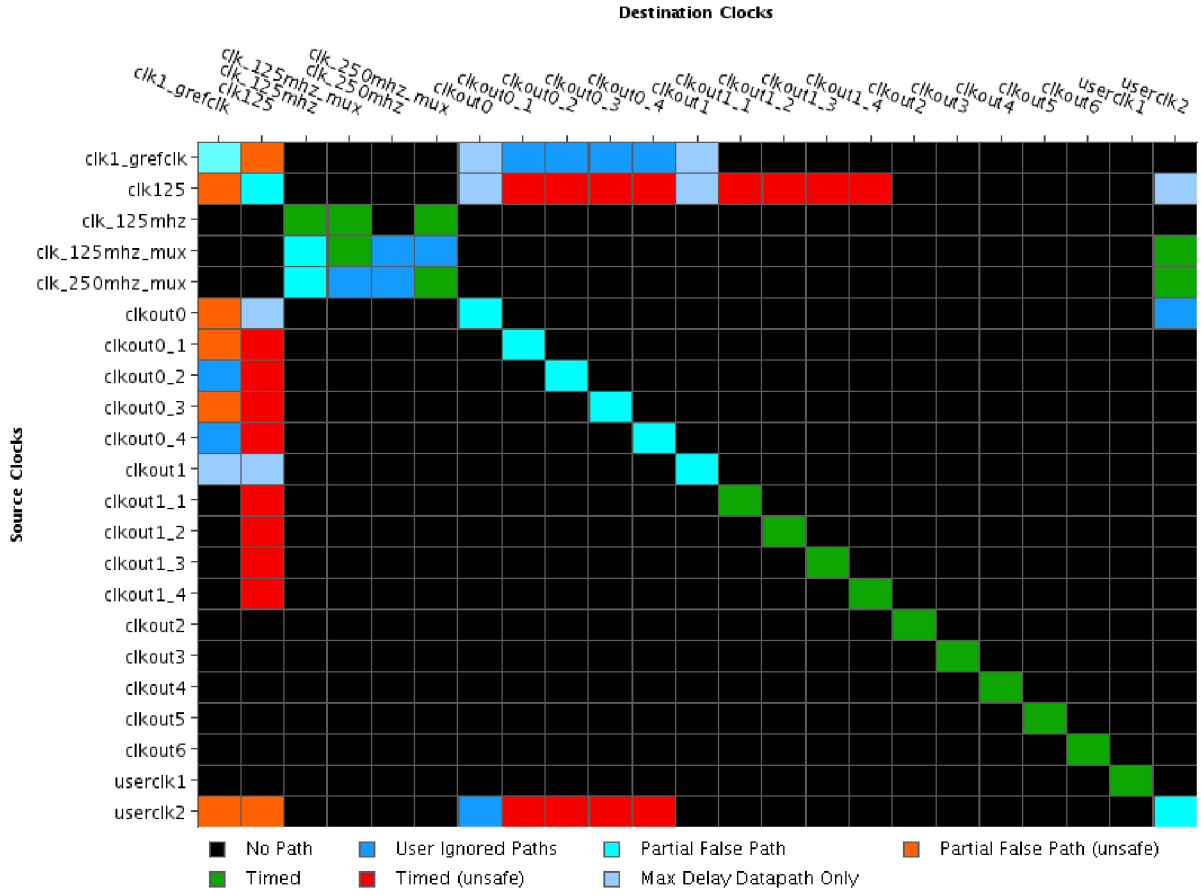
Zmíněné asynchronní přechody nebyly na první pohled viditelné, protože je skryla omezující podmínka *set_clock_group*. Tato použitá omezující podmínka způsobila to, že všechny přechody mezi různými hodinovými doménami byly vyjmuty z časové analýzy. Následkem toho se všechny asynchronní přechody ve vývojovém prostředí Vivado jevily jako správně ošetřené, i když například neobsahovaly žádný synchronizační obvod. Zmíněné špatně ošetřené asynchronní přechody mohly způsobit metastabilitu, a tím poškodit přenášená data.

7.2 Vyhledání a oprava neošetřených asynchronních přechodů

Postup pro nalezení špatně asynchronních přechodů se skládal z několika dílčích kroků. Prvním krokem bylo odstranění omezující podmínky *set_clock_group*, která špatně ošetřené asynchronní přechody skrývala. Dalším krokem bylo provedení syntézy celého návrhu síťové karty a následné zobrazení analýzy interakcí hodinových domén (Report Clock Interaction) ve vývojovém prostředí Vivado. Tato analýza zobrazuje, zda existují spojení mezi konkrétními hodinovými doménami a zda tato spojení splňují časování nebo jsou ošetřena omezujícími podmínkami. Ukázka analýzy interakcí hodinových domén z vývojového prostředí Vivado je zobrazena na obrázku 20.

Z analýzy interakcí hodinových domén lze získat výpis všech spojení respektive asynchronních přechodů mezi vybranými dvěma hodinovými doménami. Z tohoto výpisu lze vyfiltrovat například všechny neošetřené asynchronní přechody a ty dále zobrazit ve schématu. Vývojové prostředí Vivado v jisté míře umožňuje označit komponentu ve schématu a následně přejít na její VHDL kód. Díky tomu se výrazně usnadní hledání problémových míst ve VHDL kódu.

Jakmile je nalezen špatně ošetřený asynchronní přechod, tak je vybrán a použit vhodný synchronizační obvod z knihovny asynchronních přechodů pro tento konkrétní případ. Omezující podmínky budou aplikovány zcela automaticky.



Obr. 20: Analýza interakcí hodinových domén ve vývojovém prostředí Vivado Design Suite před úpravou 40G2 varianty síťové karty

Pod tímto odstavcem je uvedena část VHDL kódu z 40G2 varianty návrhu síťové karty pro akcelerační kartu COMBO-80G, která obsahuje špatně ošetřený asynchronního přechod. Ačkoliv na první pohled nejde odhalit, že jde o asynchronní přechod, při podrobné analýze celého návrhu lze zjistit, že signál „*repeater_ctrl*“ patří do hodinové domény XL_X_GMII_CLK, zatím co signál „*repeater_ctrl_netmod*“ patří do hodinové domény MI_CLK. Při propojení těchto signálů vzniká neošetřený asynchronní přechod, který by mohl způsobit například vznik metastability, a tím poškodit přenášená data.

```
repeater_ctrl_realign_gen : for i in 0 to 7 generate
      repeater_ctrl_netmod(i*2+0) <= repeater_ctrl(ifcnum_tr(i)*2+0);
      repeater_ctrl_netmod(i*2+1) <= repeater_ctrl(ifcnum_tr(i)*2+1);
end generate;
```

Tento konkrétní případ špatně ošetřeného asynchronního přechodu byl upraven tak, že signál „*repeater_ctrl*“ patřící do hodinové domény XL_X_GMII_CLK byl připojen k synchronizačnímu obvodu ASYNC_BUS_HANDSHAKE, který byl vybrán podle vypracovaného rozhodovacího stromu. Tento synchronizační obvod synchronizuje signál

„*repeater_ctrl*“ z hodinové domény XL_X_GMII_CLK do hodinové domény MI_CLK, kde je vyveden jako výstupní signál „*sync_repeater_ctrl*“. Tento signál byl následně připojen k signálu „*repeater_ctrl_netmod*“. Omezující podmínky se na použitý synchronizační obvod aplikují automaticky. Těmito kroky byl nalezený asynchronní přechod ošetřen, a tak bylo zabráněno vzniku chyb či poškození přenášených dat. Následující VHDL kód zobrazuje asynchronní přechod po úpravě.

```
sync_repeater_ctrl_i: entity work.ASYNC_BUS_HANDSHAKE
generic map (
      DATA_WIDTH => 32
)
port map (
      --! A clock domain
      ACLK       => mi_clk,
      ARST       => mi_reset,
      ADATAIN    => repeater_ctrl,
      ASEND      => '1',
      AREADY     => open,
      --! B clock domain
      BCLK       => xl_x_gmii_clk,
      BRST       => xl_x_gmii_rst,
      BDATAOUT   => sync_repeater_ctrl,
      BLOAD      => '1',
      BVALID     => open
);

repeater_ctrl_realign_gen : for i in 0 to 7 generate
      repeater_ctrl_netmod(i*2+0) <= sync_repeater_ctrl(ifcnum_tr(i)*2+0);
      repeater_ctrl_netmod(i*2+1) <= sync_repeater_ctrl(ifcnum_tr(i)*2+1);
end generate;
```

7.3 Shrnutí a zhodnocení vlivu úprav asynchronních přechodů na návrh síťové karty

Z tabulky 2, která obsahuje získané hodnoty časové analýzy implementace před a po úpravách návrhu sítové karty, je vidět, že u obou verzí návrhu síťové karty se úprava asynchronních přechodů projevila mírně lepšími výsledky časové analýzy. Tyto výsledky časové analýzy však můžeme brát v úvahu jen orientačně, protože vývojové prostředí Vivado implementuje návrh pro FPGA jen do doby, než výsledná implementace splňuje časování, a dál již lepší varianty implementace nehledá.

	Varianta 10G8		**Varianta 40G2**	
	Před úpravou	**Po úpravě**	**Před úpravou**	**Po úpravě**
WNS [ns]	0,010	0,074	0,000	0,009
TNS [ns]	0,000	0,000	0,000	0,000
WHS [ns]	0,017	0,017	0,017	0,017
THS [ns]	0,000	0,000	0,000	0,000

Tab. 2: Přehled výsledků časové analýzy před a po úpravě asynchronních přechodů

Z tabulky 3 je patrné, že se implementované úpravy asynchronních přechodů projevily jen velmi minimálně na celkové spotřebě zdrojů. To je ale zcela logické, protože asynchronní přechody mají oproti celému návrhu sítové karty naprosto zanedbatelné využití zdrojů.

	Varianta 10G8		**Varianta 40G2**	
	Před úpravou	**Po úpravě**	**Před úpravou**	**Po úpravě**
LUT [%]	27,34	27,37	20,92	20,86
FF [%]	12,95	13,00	8,07	8,13
BRAM [%]	24,52	24,52	8,54	8,54
DSP [%]	9,75	9,75	2,42	2,42

Tab. 3: Přehled využití zdrojů před a po úpravě asynchronních přechodů

Závěrem lze tedy říct, že správné ošetření asynchronních přechodů nemusí být na první pohled jasně patrné, ale jeho zásadním přínosem je zabránění vzniku nebezpečných a náhodných chyb, jako je například metastabilita, které by mohly způsobit poškození přenášených dat.

Závěr

V rámci předložené práce byl proveden rozbor problematiky asynchronních přechodů mezi hodinovými doménami v obvodech FPGA. Byly analyzovány chyby, které mohou v nesprávně implementovaném asynchronním přechodu nastat. Dále byly rozebrány možnosti implementace jednobitových a vícebitových synchronizačních obvodů včetně popisu aplikace omezujících podmínek, které jsou velmi často zanedbávané.

Na základě výsledků rozboru byly v rámci knihovny asynchronních přechodů realizovány obecné synchronizační obvody a z nich odvozené synchronizační obvody pro proprietární sběrnice MI32, FL, FLU a DMA využívané sdružením CESNET, z. s. p. o., které bylo zadavatelem této práce. Zásadním přínosem realizovaných synchronizačních obvodů je jejich jednoduché použití v praxi spolu s automatickou aplikací připravených omezujících podmínek ve vývojovém prostředí Vivado Design Suite od společnosti Xilinx. Tyto synchronizační obvody z knihovny asynchronních přechodů se již používají v projektech realizovaných v rámci sdružení CESNET, z. s. p. o., například při vývoji firmwaru pro zmiňovanou akcelerační kartu COMBO-80G. Nasazení realizovaných synchronizačních obvodů v praxi přináší zejména zvýšení spolehlivosti, respektive snížení doby ladění a snížení možnosti chyb nebo překlepů vůči ručnímu zápisu omezujících podmínek.

Dalším důležitým výstupem předložené bakalářské práce je navržená metodika výběru a použití synchronizačních obvodů, která byla vypracována ve formě rozhodovacího stromu. Tato metodika usnadňuje použití realizované knihovny asynchronních přechodů. V závěru práce byla provedena případová studie použití asynchronních přechodů v návrhu síťové karty pro akcelerační kartu COMBO-80G. Případová studie ověřila funkčnost implementovaných synchronizačních obvodů v praxi.

Literatura

[1] COMBO-80G FPGA karta. *INVEA-TECH* [online]. 2015 [cit. 2015-05-29]. Dostupné z: https://www.invea.com/cs/produkty-sluzby/fpga-karty/combo-80g

[2] PINKER, Jiří a Martin POUPA. *Číslicové systémy a jazyk VHDL*. 1. vyd. Praha, ČR: BEN - technická literatura, 2006, 349 s. ISBN 80-730-0198-5.

[3] KOLOUCH, Jaromír. *Programovatelné logické obvody a hradlová pole* [online]. 2006 [cit. 2014-12-14]. Dostupné z: http://www.urel.feec.vutbr.cz/~kolouch/pld/

[4] XILINX INC. *Vivado Design Suite User Guide: Using Constraints - UG903*. v2014.3. San Jose, USA, October 2014. Dostupné z: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_4/ug903-vivado-using-constraints.pdf

[5] CUMMINGS, Clifford. SUNBURST DESIGN. *Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog*. Boston, USA, 2008. Dostupné z: http://www.sunburst-design.com/papers/CummingsSNUG2008Boston_CDC.pdf

[6] MUSIL, Vladislav a Jaromír KOLOUCH. *Návrh digitálních integrovaných obvodů VLSI a jazyk VHDL*. Brno, ČR, 28.11.2007. Dostupné z: http://www.umel.feec.vutbr.cz/BDIO/skripta/Navrh_digitalnich_integrovanych_obvodu_a_jazyk_VHDL_S.pdf

[7] HASELOF, Eilhard. TEXAS INSTRUMENTS. *Metastable Response in 5-V Logic Circuits*. Dallas, USA, February 1997. Dostupné z: http://www.ti.com/lit/an/sdya006/sdya006.pdf

[8] XILINX INC. *Vivado design suite advanced XDC and STA for ISE software users*. San Jose, USA, 2012.

[9] XILINX INC. *Vivado Design Suite User Guide: Synthesis - UG901*. v2014.4. San Jose, USA, November 2014. Dostupné z: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_4/ug901-vivado-synthesis.pdf

[10] XILINX INC. *7 Series FPGAs Memory Resources: User Guide - UG473*. San Jose, USA, November 2014. Dostupné z: http://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf