VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

APLIKOVANÉ VYUŽITÍ DSP BLOKŮ V INTEL FPGA
APPLIED USE OF DSP BLOCKS IN INTEL FPGA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Daniel Kondys

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. David Smékal

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: Daniel Kondys ***ID:*** 203254
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Aplikované využití DSP bloků v Intel FPGA

POKYNY PRO VYPRACOVÁNÍ:

Cílem bakalářské práce je navrhnout vhodné využití DSP bloků, které jsou dostupné u moderních FPGA čipů, k implementaci čítače a komparátoru. Seznamte se s jazykem VHDL a s architekturou moderních FPGA čipů. Podrobně nastudujte možnosti DSP bloků v FPGA Intel Stratix 10. Navrhněte a realizujte implementaci čítače a komparátoru využívající DSP bloky dostupné na FPGA. Funkčnost implementace ověřte simulací a následně implementací na FPGA Intel Stratix 10. Závěrem zhodnoťte dosažené výsledky.

DOPORUČENÁ LITERATURA:

[1] INTEL Corporation, 2018. Intel® Stratix® 10 Variable Precision DSP Blocks User Guide: UG-S10-DSP [online]. 18.1. Dostupné z: https://intel.ly/2kpvP9W

[2] PINKER, Jiří, Martin POUPA. Číslicové systémy a jazyk VHDL. 1. vyd. Praha: BEN - technická literatura, 2006, 349 s. ISBN 80-7300-198-5.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: Ing. David Smékal
Konzultant: Ing. Jakub Cabal (CESNET, z. s. p. o.)

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Tato bakalářská práce se zabývá využitím DSP bloků zejména na FPGA Stratix 10 DX 2800 pro implementaci čítače a komparátoru. V teoretické části je zběžně popsán protokol Ethernet, vysvětlena souvislost síťových karet s FPGA čipy, následně je popsána jejich struktura a obecný postup při návrhu digitálního obvodu pro FPGA. V poslední podkapitole této části jsou podrobně rozebrány DSP bloky na FPGA Virtex UltraScale+ XCVU7P a Stratix 10 DX 2800. V praktické části je popsána realizace čítače a komparátoru, od jejich návrhu přes vlastní implementaci až po testování. U těchto komponent byly následně měřeny a vyhodnoceny parametry týkající se spotřeby zdrojů FPGA a maximální frekvence. Nakonec byly tyto komponenty zapojeny do obvodů, které jsou součástí firmwaru pro síťovou kartu COMBO-400g1, a také zde byly měřeny a vyhodnoceny změny ve spotřebě zdrojů a maximální frekvence.

KLÍČOVÁ SLOVA

CESNET, čítač, DSP, Ethernet, FPGA, komparátor, Quartus, síťová karta, Stratix 10, UltraScale+, VHDL

ABSTRACT

This bachelor's thesis explores the utilization of DSP blocks located particularly on FPGA Stratix 10 DX 2800 for the implementation of a counter and a comparator. In the theoretical part, topics such as the protocol Ethernet, the FPGA technology and its relation with Network Interface Controllers are explained, followed by a description of general design flow for digital circuits on FPGAs and a detailed insight on DSP blocks in the Virtex UltraScale+ XCVU7P and Stratix 10 DX 2800 FPGAs. The practical part focuses on the design, implementation and testing of the counter and comparator, followed by measurements of their impact on FPGA's resource utilization and maximum frequency. Lastly, it describes the integration of these components into modules that are part of the COMBO-400g1 Network Interface Connector firmware and analyzes their impact on FPGA's resource utilization and maximum frequency.

KEYWORDS

CESNET, comparator, counter, DSP, Ethernet, FPGA, Network Interface Controller, Quartus, Stratix 10, UltraScale+, VHDL

KONDYS, Daniel. *Aplikované využití DSP bloků v Intel FPGA*. Brno, 2020, 72 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. David Smékal

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Aplikované využití DSP bloků v Intel 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 autorského 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.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Davidovi Smékalovi za ochotu ke konzultacím a přínosné rady. Dále bych rád poděkoval odbornému konzultantovi panu Ing. Jakubovi Cabalovi za neúnavnou podporu i trpělivost, věcné rady a připomínky ke zlepšení této práce.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V dnešní době stále narůstá požadavek na vyšší přenosovou rychlost v digitálních komunikacích. Jedním z vhodných řešení je využití FPGA (Field Programmable Gate Array – programovatelná hradlová pole) čipů. Použití těchto čipů například v síťových kartách umožní zrychlit zpracování dat, a tím výrazně snížit zatížení procesoru. Nároky na větší přenosovou rychlost mají za následek větší spotřebu zdrojů na FPGA, a proto je důležité co nejefektivněji využít všechny dostupné prostředky, které dané FPGA obsahuje.

Jedním z takových prostředků jsou DSP (Digital Signal Processing – digitální zpracování signálů) bloky. Standardně se tyto bloky využívají pro různé matematické operace, například násobení. Taková využití nejsou vždy zapotřebí, a proto je pro tyto DSP bloky vhodné nalézt i jiná uplatnění. Vhodným uplatněním může být například čítač nebo komparátor, které jsou využívány častěji. Zejména při větších šířkách datových slov může takové řešení ušetřit mnoho zdrojů, které lze využít jinak.

Jedna z možných aplikací, kde je možné využít výsledky této práce, jsou síťové karty, které dnes běžně využívají FPGA čipů. Tyto karty využívají statistických čítačů například pro počítání přijatých a odeslaných rámců a komparátorů pro různá porovnání například MAC (Media Access Control – řízení přístupu k médiu) adres. Takové čítače a komparátory mohou být implementovány v DSP blocích nacházejících se v daném FPGA.

V teoretické části je nastíněn protokol Ethernet, jakožto jeden z nejrozšířenějších protokolů používaných v síťových komunikacích. Poté je vysvětlena souvislost mezi síťovými kartami a FPGA čipy, jejichž architektura je následně popsána. Dále je vysvětlen postup návrhu digitálního obvodu pro FPGA a nakonci této kapitoly jsou podrobně rozebrány DSP bloky na FPGA čipech Stratix 10 DX 2800 a Virtex UltraScale+ XCVU7P. Praktická část se skládá z realizace DSP čítače a DSP komparátoru. U obou komponent byl popsán jejich návrh, implementace i ověření funkčnosti. Obě také byly zapojeny do vybraných obvodů, jež jsou součástí firmware síťové karty COMBO-400g1 připravované sdružením CESNET.

1 Teoretická část

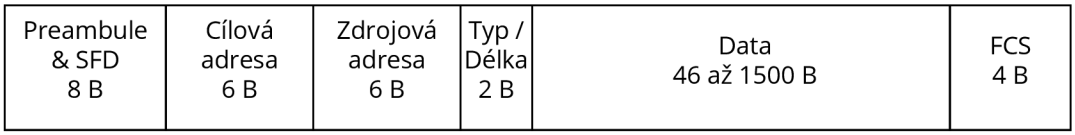
V této kapitole jsou probírány teoretické záležitosti a nastudované poznatky, které jsou důležité k pochopení podstaty této práce a byly potřeba k její realizaci. Jsou zde zmíněny základní poznatky o Ethernetu, více pak o síťových kartách, ve kterých bude výstup této práce aplikován, na což navazuje kapitola o technologii FPGA, která je v těchto kartách použita. Následuje obecný popis návrhu digitálního obvodu pro FPGA a nejdůležitější kapitola je zaměřena na DSP bloky nacházející se v FPGA Stratix 10 DX2800 a Virtex UltraScale+ XCVU7P.

1.1 Ethernet

Ethernet je nejrozšířenější technologií pro propojování zařízení v lokálních i rozsáhlých sítích. Umožňuje propojeným zařízením spolu komunikovat přes sadu pravidel zvaných protokol [1].

Popularita Ethernetu má své odůvodnění. Je levný, a to nejen z pohledu nízké ceny ethernetových zařízení a jejich příslušenství, ale také kvůli jednoduchosti instalace, údržby a odstraňování problémů, díky čemu není potřeba tolika znalých, zkušených a tudíž i nákladných techniků. Další výhodou je jeho flexibilita, a to jednak z pohledu topologie zapojení – Ethernet je možné použít ve všech topologiích – a jednak z pohledu efektivního využití dostupné šířky pásma [2].

Ethernet byl poprvé zveřejněn roku 1980 s rychlostí až 10 Mb/s a o pět let později byl standardizován Institutem pro elektrotechnické a elektronické inženýrství (IEEE – Institute of Electrical and Electronics Engineers) jako IEEE 802.3 - 1985. Do rodiny IEEE 802.3 dnes spadá mnoho standardů lišících se většinou v rychlostech, případně přidaných protokolech. V běžných lokálních sítích dnes většinou narazíme na Ethernet o rychlosti 100 Mb/s, který je dnes často nahrazován 1 Gb/s Ethernetem. V páteřních sítích jsou však požadovány mnohem vyšší rychlosti. Nejrychlejší současné síťové karty zvládají až 200 Gb/s, karta zvládající 400 Gb/s je právě vyvíjena sdružením CESNET. Obě zmíněné rychlosti sdílejí standard IEEE Std 802.3bs [3].



Obr. 1.1: Základní ethernetový rámec [4]

Přes Ethernet se posílají data ve formě rámců. Typická struktura takového rámce je na obrázku 1.1. Ethernetový rámec se skládá ze tří hlavních částí: hlavičky (header), datového pole (payload) a patičky (trailer) [4]. Mezi jednotlivými rámci musí být určitá mezera, kdy se přes dané komunikační médium neposílají žádná data. Mezirámcová mezera umožní přijímací stanici připravit se na příjem dalšího rámce, nebo třeba poskytne čas stanici, která právě dovysílala, přepnout z vysílacího módu na přijímací. Při nedodržení mezery mezi rámci by mohlo dojít k tomu, že by stanice minula data, která pro ni byla určena [5]. Standardní mezera mezi vysílanými rámci je 96 b, pro vyšší rychlosti Ethernetu je možná mezera jen 8 b[6]. Počítá se od posledního bitu FCS (Frame Check Sequence – kontrolní součet rámce) jednoho rámce po první bit preambule následujícího rámce[7].

Zařízení komunikující přes protokol Ethernet vyšle úvodní 7B posloupnost střídajících se 1 a 0 známou jako preambuli (Preamble), která umožní přijímací stanici(ím) se s ním synchronizovat a připravit se na příjem. Za ní je 1B oddělovač začátku rámce (SFD – Start of Frame Delimiter), který navazuje na posloupnost 1 a 0, ale je zakončen dvěma jedničkami. Na to navazuje hlavička rámce. V prvním políčku rámce se nachází MAC adresa zařízení, kterému je rámec určen, tzv. cílová MAC adresa. Jedná se o 6 B dlouhou adresu, která jednoznačně identifikuje cílové zařízení v rámci dané sítě. Podobně je to s dalším políčkem, kde se nachází 6B MAC adresa zařízení, které rámec vysílá, tzv. zdrojová MAC adresa. Poslední částí hlavičky je 2B políčko nesoucí informaci o délce rámce (Length) nebo typu protokolu vyšší vrstvy (EtherType – zkráceně jen Type), záleží na typu ethernetového rámce. V současné době se při zmínce ethernetového rámce myslí rámec typu Ethernet II, kde v hlavičce za MAC adresami je v políčku EtherType [4]. Případně lze délku od typu rozlišit tak, že pokud je číslo v tomto poli menší nebo rovno než 1500, indikuje délku rámce, a pokud je větší nebo rovno 1536, indikuje protokol vyšší vrstvy [8].

Následuje část datového pole, které v sobě má hlavičky protokolů vyšších vrstev[1] a hlavně data uživatele. Jeho velikost musí být mezi 46 a 1500 B. Existuje výjimka pro „jumbo“ rámce, které mohou být až 9000 B dlouhé a používají se například pro přenos souborů [4].

Ethernetový rámec končí 4B patičkou, ve které je kontrolní součet rámce (FCS – Frame Check Sequence). Kontrolní součet se spočítá z části rámce (například hlavičky) nebo z celého rámce (vyjma preambuli a SFD). Přijímací stanice také vypočítá FCS přijatého rámce a pokud není stejný, jako FCS v patičce, identifikuje jej za poškozený a definovaným způsobem s ním naloží [9].

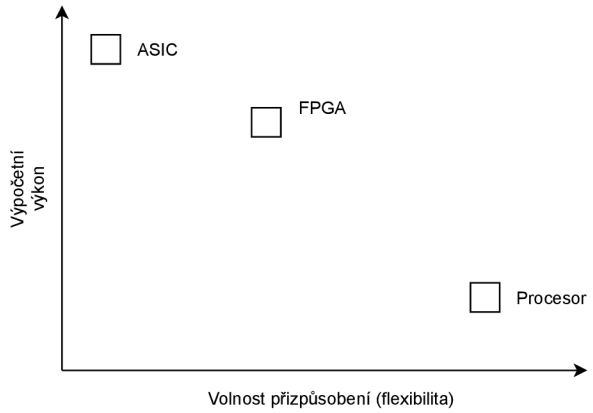
1.2 Síťové karty

Síťové karty jsou zařízení zpracovávající data ze sítě, což u vyšších rychlostí, kdy je potřeba zpracovávat více dat za méně času, není výpočetně jednoduchá záležitost. V této podkapitole budou rozebrány výhody a nevýhody technologií, které se pro zpracování síťových dat používají, a budou také uvedeny reálné příklady síťových karet využívající technologii FPGA.

1.2.1 Porovnání technologie ASIC, FPGA a procesoru

Procesory jsou nejběžnější výpočetní jednotky v elektronických zařízeních. Mají obecnou strukturu, čímž umožňují téměř neomezené využití [10]. To znamená, že činnost zpracování paketů zvládnou, ale jelikož na to nejsou optimalizovány (co se hardwaru týče), jejich zpracování tak není příliš efektivní. To ovšem není problémem v lokálních sítích, kde rychlost internetu většinou nepřesahuje 1 Gb/s. Zato však v aplikacích, kde nejdůležitějším parametrem je efektivita a rychlost zpracování paketů (například páteřní sítě, datová centra) se využívají speciální síťové karty. Síťová karta je hardwarový prvek, pomocí kterého se zpracovávají síťová data, obecně nazývaná jako pakety. Zmíněné speciální síťové karty zpravidla využívají hardwarové akcelerace, tedy že určité úkony jsou vykonávány v hardwarových prvcích síťové karty namísto v procesoru. Těmito hardwarovými prvky většinou bývají čipy ASIC (Application-Specific Integrated Circuit – integrovaný obvod pro konkrétní aplikaci) nebo FPGA. Tyto čipy se liší ve výkonu, flexibilitě a složitosti návrhu, a tudíž mají i různá uplatnění.

[1] Žádné jiné další vrstvy patičku nemají, pouze hlavičku.



Obr. 1.2: Porovnání technologie ASIC, FPGA a procesoru [11]

Obrázek 1.2 znázorňuje porovnání technologií ASIC a FPGA s procesorem podle výkonu a flexibility. Jedná se zejména o ilustrativní obrázek, nelze tedy říci, že například FPGA je v poměru asi 4× výkonnější než procesor atp. Takové poměry hodně závisí na aplikaci dané technologie, například ve zdroji [12], kde jsou porovnávány tyto a další technologie pro hardwarové urychlení binárních neuronových sítí (BNNs – Binarized Neural Networks), lze z porovnání vyčíst, že v daných měřeních bylo FPGA průměrně 70× výkonnější než procesor.

ASIC je speciální obvod, který je navržen pro jednu specifickou aplikaci, což ústí ve velmi vysoký výkon. Je to tedy integrovaný obvod skládající se z elektrických součástek, které jsou pak určitým způsobem zapojeny na destičce z křemíku nebo jiného polovodičového materiálu. Tento určitý způsob zapojení závisí na požadované funkcionalitě obvodu ASIC. Nedostatkem technologie ASIC je jeho nízká, takřka nulová flexibilita, která vychází z jeho principu. Elektrické součástky jsou zapojeny takovým způsobem, aby vykonávaly jednu určitou aplikaci. Úprava funkce ASIC je tudíž takřka nemožná a často bývá jednodušší začít s návrhem od začátku. Návrh a výroba takového čipu bývá velmi pracná a zdlouhavá, a tím pádem také drahá. Proto jsou čipy ASIC vytvářeny pro aplikace s velkým důrazem na rychlost, malými rozměry, nízkou spotřebou a obrovským využitím (velkosériová výroba).

Technologie FPGA podle již zmíněného obrázku 1.2 sice není na takové výkonnostní úrovni, jako je ASIC, ale jejich výhodou je nižší cena a větší flexibilita. Je značně rychlejší navrhnout obvod na FPGA, otestovat jej a případně vyladit nějaké chyby než návrh, vytvoření masky a verifikace obvodu ASIC. Po vytvoření FPGA čipu jsou do něj nahrána konfigurační data (blíže popsáno v podkapitole 1.4) a v případě změny požadavků aplikace se stávající nastavení FPGA přepíše novým, upraveným a otestovaným návrhem. Tato technologie je proto vhodná pro malosériovou výrobu a rychlý vývoj [11, 13].

1.2.2 Příklady síťových karet

Jako reálné příklady síťových karet jsou zde uvedeny dva, prvním je karta COMBO-200g2ql, která byla vyvinuta sdružením CESNET v nedávné době, druhým je karta COMBO-400g1, která je sdružením CESNET právě vyvíjena. V obou těchto kartách bude možné použít čítač, komparátor bude však

možné použít pouze pro kartu COMBO-400g1, protože nezahrnuje implementaci komparátoru pro Xilinx FPGA, což je blíže vysvětleno v podkapitolách 2.2 a 3.2.

Porovnání základních parametrů těchto karet je uvedeno v tabulce 1.1, fotografie karty COMBO-200g2ql zapojené v serveru je na obrázku 1.3. Jak bylo již zmíněno, karta COMBO-400g1 je ve vývoji, a tudíž její parametry nejsou oficiální a ještě se mohou mírně změnit [14].

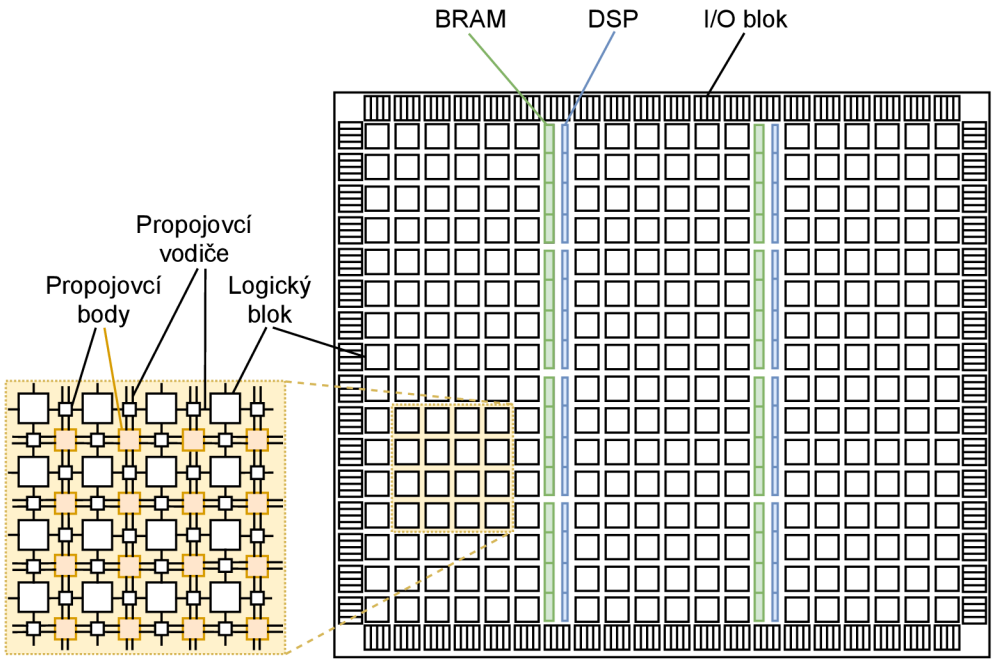
Tab. 1.1: Tabulka porovnání síťových karet [14]

Parametr	Síťová karta	
Název	COMBO-200g2ql	COMBO-400g1
Propustnost	200 Gb/s	400 Gb/s
FPGA	Virtex UltraScale+ XCVU7P	Stratix 10 DX 2800
PCI Express	2 × PCI Express Gen3 × 16	2 × PCI Express Gen4 × 16
Optické transcievery	2 × QSFP28	1 × QSFPDD
Externí paměti	3 × QDRIIIe SRAM	2 × sloty pro DDR4

Obr. 1.3: Síťová karta COMBO-200g2ql vyvinutá sdružením CESNET

1.3 Technologie FPGA

FPGA jsou integrované obvody postavené na 2D matici zejména logických bloků, ve kterých lze implementovat různé kombinační či sekvenční obvody, dále blokových pamětí typu RAM (Random Access Memory – paměť s přímým přístupem) neboli zkráceně BRAM (Block Random Access Memory), DSP bloků využívaných pro různé matematické operace, vstupních/výstupních (I/O) bloků, které propojují vnitřek FPGA se zbytkem desky plošných spojů, a dalších bloků například pro řízení frekvence hodinového signálu. Všechny tyto bloky jsou propojeny propojovacími vodiči. Zmíněné prvky FPGA jsou vyznačeny na obrázku 1.4 [11, 15].

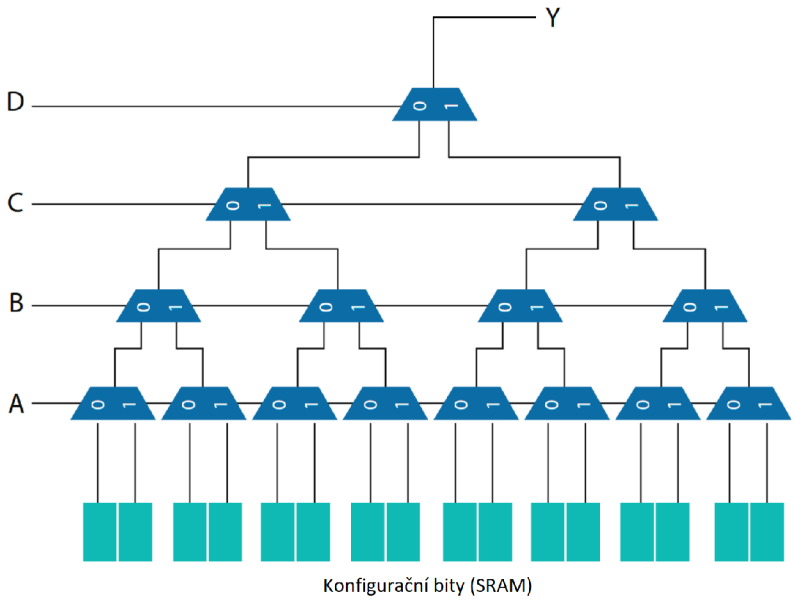


Obr. 1.4: Obecná architektura FPGA [11]

1.3.1 Obecný popis architektury FPGA

Základním prvkem vykonávající logické funkce v FPGA je vyhledávací tabulka zvaná LUT (Look-Up Table), jejíž vnitřní struktura je ukázána na obrázku 1.5. Podle konfigurace FPGA uložené v konfiguračním souboru (podrobněji v 1.4), podle kterého bude navržený obvod v FPGA vytvořen, jsou vstupům spodních multiplexorů přiděleny bitové hodnoty. Tyto hodnoty jsou zde uloženy v buňkách SRAM (Static RAM – statická paměť s přímým přístupem) do té doby, dokud není stávající obvod FPGA přepsán jiným (je nahrán jiný konfigurační soubor), případně – protože se jedná o energeticky závislou paměť – dokud nedojde k přerušení napájení [15]. Logická funkce vyhledávací tabulky může

být tedy libovolná (limitovaná pouze maximálním počtem jejích vstupů), je určena pouze konfiguračními bity, jenž se přes multiplexory dostanou na výstup vyhledávací tabulky podle vstupních signálů A, B, C a D. Vyhledávací tabulka se tak nazývá proto, že její výstup je vybrán vyhledáním správného bitu a jeho provedením sítí multiplexorů řízených vstupními signály vyhledávací tabulky [16, 17].

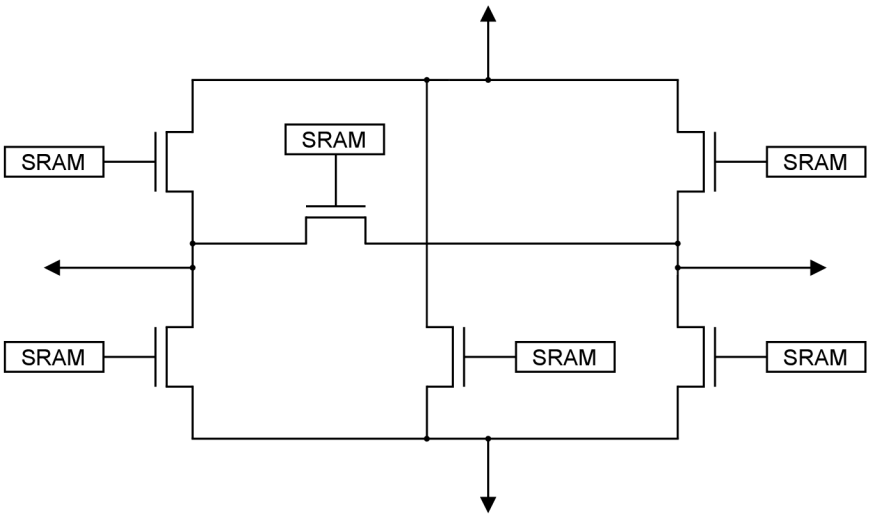


Obr. 1.5: Vyhledávací tabulka se čtyřmi vstupy a jedním výstupem [16]

V blízkém okolí vyhledávací tabulky se nachází registr/y (záleží na typu FPGA), pomocí nichž lze v kombinaci s vyhledávací tabulkou vytvářet sekvenční obvody, dále zde bývá logika pro přenos z okolních a k okolním logickým prvkům a jednotky pro vykonávání jednoduchých matematických operací, například sčítačky. Vyhledávací tabulka s výše vyjmenovanými prvky v okolí utváří blok, jenž se v jistých ohledech liší dle výrobce a jsou také jinak nazývány. Firma Xilinx pojmenovává tyto bloky Slice, Intel ALM (Adaptive Logic Module – přizpůsobivý logický modul), viz následující část.

Propojovací síť zabírá majoritní část FPGA. Dlouhé propojovací vodiče vedou mezi sloupci a řádky logických bloků (viz obrázek) a tvoří hlavní propojovací síť. Používají se zejména k propojením vzdálených logických bloků nebo pro sběrnice. Svislé a vodorovné vodiče se spolu propojují v přepojovacích maticích (na obrázku 1.4 zobrazeny jako béžové čtverečky). Kromě nich se v FPGA nachází přímé propoje, které propojují poblíž nacházející se logické bloky. Přímé propoje jsou s hlavní propojovací sítí spojeny přes propojovací body, na obrázku 1.4 zobrazeny jako menší bílé čtverečky. Bod, ve kterém dochází k přepojení svislého a vodorovného vodiče, je detailněji zobrazen na obrázku 1.6. Tranzistory jsou také zde řízeny paměťovou buňkou SRAM, jejíž hodnota opět pochází z konfiguračního souboru blíže rozebraného v části 1.4 [11, 17, 18].

Na okrajích FPGA se nachází vstupní/výstupní bloky (IOB – Input/Output Blocks). Každý takový blok může být konfigurován jako vstup, výstup nebo obojí. Dále podporuje množství různých standardů, včetně těch, které používají diferenciální signály (v tom případě musí být použity dva sousední



Obr. 1.6: Propoj svislého a vodorovného vodiče [17]

I/O bloky). Vstupní/výstupní bloky obsahují pull-up rezistory pro možnost zakončení signálů a tím šetří potřebné zdroje na desce.

Možnosti využití FPGA jsou velmi rozsáhlé. Kromě síťových karet se může jednat například o letecký průmysl, kde se využívají FPGA tolerantní vůči radiaci, o automobilový průmysl například jako asistent při řízení, v lékařství pro zpracování různých obrázků a snímků, monitorování pacientů a také na výpočty například rychlé Fourierovy transformace, dále se uplatňují třeba v zabezpečovacích systémech a lze pomocí jich vytvářet prototypy pro obvody ASIC [19, 20].

Přední výrobci FPGA jsou již zmíněné firmy Xilinx a Intel (dříve Altera), dalším takovým je třeba firma Lattice, jenž se zaměřuje na výrobu menších, úspornějších a cenově dostupnějších FPGA, které nejsou podstatné pro tuto práci. Dále budou zmiňovány jen FPGA od firem Xilinx a Intel, zejména modely Virtex UltraScale+ XCVU7P a Stratix 10 DX 2800.

1.3.2 Porovnání FPGA od firmy Xilinx a Intel

V této části se nebude jednat o obecné srovnání výrobků firem Xilinx a Intel, nýbrž přímo o určité modely používané v síťových kartách. Jsou to již zmíněné modely FPGA Virtex UltraScale+ XCVU7P od firmy Xilinx a Stratix 10 DX 2800 od firmy Intel. Pro jednoduchost budou dále používány zkrácené názvy modelů, Virtex UltraScale+ XCVU7P bude nazýván jako UltraScale+ a model Stratix 10 DX 2800 jako Stratix 10.

V FPGA UltraScale+ jsou logické bloky, viz obrázek 1.4, nazývány jako CLB (Configurable Logic Block - konfigurovatelný logický blok). Ve starších architekturách CLB obsahovaly více sliců (2–4), které jsou v architektuře UltraScale+ spojeny v jeden pro možnost vytváření složitějších (vícevstupých) funkcí. Obyčejný slice je nazýván SLICEL, druhým typem je SLICEM, který může být použit nejen

jako vyhledávací tabulka, ale také jako distribuovaná 64b paměť typu RAM, případně jako 32b posuvný registr. V obou typech sliců se nachází 8 šestivstupých vyhledávacích tabulek, což pro SLICEM znamená, že může sloužit jako až 512b paměť. Co se logických funkcí týče, každá z nich může být použita pro 1 šestivstupou funkci nebo 2 pětivstupé se společnými vstupními signály. Dále se v jednom slicu nachází 8b propoje pro přenos mezi vyhledávacími tabulkami, 16 paměťových prvků (registrů) a multiplexory právě pro vytváření více než šestivstupých funkcí [21]. Xilinx uvádí počty zdrojů na FPGA pomocí tzv. systémových logických buněk, což je počet vyhledávacích tabulek vynásobených jistým koeficientem (pro UltraScale+ se používá 1,6), který by měl kromě samotné tabulky brát v potaz i nápomocnou okolní logiku [22].

Intel své logické bloky (opět viz obrázek 1.4) nazývá LAB (Logic Array Block – blok logických polí). Každý LAB obsahuje 10 ALM nad sebou ve sloupci, které jsou propojeny tzv. místními propoji spojující také LAB se sousedními prvky, ať už se jedná o další LAB, blokovou paměť, nebo třeba DSP blok. Jeden ALM obsahuje dvě kombinační ALUT (Adaptive LUT – přizpůsobivá vyhledávací tabulka) dohromady s 8 vstupy, pomocí kterých lze vytvořit až osmivstupou funkci v tzv. rozšířeném módu vyhledávací tabulky. Také přináší více možností použití, kromě 2 pětivstupých funkcí, kde musí být alespoň 2 vstupy stejné (protože 10 vstupů do prvku s 8 vstupy dostat nelze), je možné vytvořit například 2 na sobě nezávislé funkce o čtyřech a čtyřech vstupech nebo o pěti a třech vstupech atd. ALUT v paměťových typech ALM mohou být navíc nastaveny jako 64b paměti typu RAM. ALM v jednom LABu jsou vždy jednoho typu, pokud se tedy jedná o ty paměťové, logický blok se nazývá jako MLAB (Memory LAB – paměťový blok logických polí) a dohromady může mít kapacitu až 640 b. Dále se v ALM nachází logika pro přenos (mezi ALM v jednom LABu nebo i mezi ALM sousedících LABů), 2 hardwarové sčítačky a 4 registry a k tomu logika na jejich takřka libovolnou kombinaci. Intel své logické bloky přepočítává na tzv. logické elementy (LE – Logic Element). Logický element je starší verzí ALM, má 1 standardní čtyřvstupou LUT, 1 registr a neobsahuje hardwarové sčítačky, viz například FPGA Intel Cyclone 10 LP [23]. [24]

Tab. 1.2: Tabulka porovnání zdrojů v FPGA UltraScale+ a Stratix 10 [25, 26]

Parametr	FPGA	
Výrobce	Xilinx	Intel
Model	Virtex UltraScale+ XCVU7P	Stratix 10 DX 2800
Logické bloky [-]	1 724 000	2 753 000
DSP bloky [-]	4 560	5 760
Distribuovaná paměť [Mb]	24,1	15
Bloková paměť [Mb]	230,6	229
I/O piny [-]	832	816
Maximální rychlost transcieverů [Gb/s]	32,75 (GTY)	57,8 (GXE)
Celkový počet transcieverů [-]	80 (GTY)	4 (GXE) + 76 (GXP)

Položka logické bloky uvedená v tabulce 1.2 se v tomto případě vztahuje na počty logických buněk u UltraScale+ a na počty logických elementů ve Stratix 10. I když je nelze stoprocentně srovnat, protože u každého výrobce mají kvůli jiné technologii trochu jiný význam, vždy se nějakým způsobem jedná o vyhledávací tabulku a okolní logiku. Je tedy zřejmé, že ve Stratix 10 se těchto zdrojů nachází značně více.

DSP bloky jsou realizovány jako vestavěné bloky. Využívají se pro náročnější výpočty, které by implementací pomocí logických bloků zabraly mnoho zdrojů a spotřebovaly více energie. Stratix 10 má vestavěných DSP bloků více, i když je třeba zmínit, že tyto DSP bloky jsou jednodušší než ty, které se nachází na UltraScale+. DSP bloky jsou podrobně rozebrány v kapitole 1.5.

Distribuovaná paměť je tvořená pomocí sliců, respektive LABů typu M. Tyto paměti RAM v UltraScale+ mají více možností nastavení, viz dokument [21], oproti možnostem ve Stratix 10 viz [24].

Bloková paměť je v FPGA realizovaná jako vestavěný blok podobně jako DSP. V UltraScale+ se nachází dva typy těchto pamětí, Block RAM s kapacitou 36 kb (s maximální šířkou jedné položky 72 b) a UltraRAM s kapacitou 4 kb × 72 b [27]. Protože se v obou případech jedná o blokové paměti, v tabulce 1.2 byly pro jednoduchost celkové kapacity těchto pamětí sečteny. Ve Stratix 10 se jedná o M20K – 20kb paměť s maximální šířkou jedné položky 40 b [28].

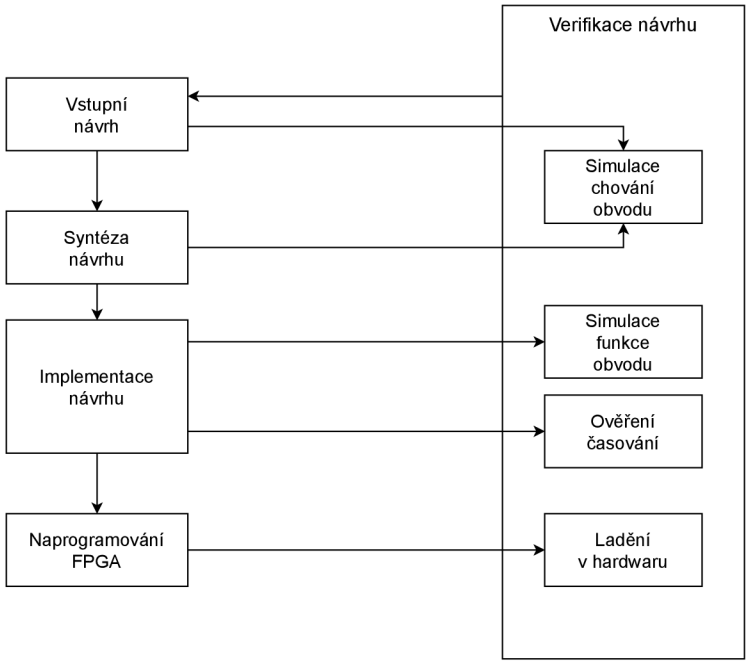
Vstupní/výstupní piny jsou v obou FPGA podobné jak z hlediska provedení, tak početně. Další položka srovnává transcivery, což jsou obvody, který mohou pracovat jako přijímač i jako vysílač na vysokých rychlostech. Přijímač paralelizuje vstupní sériový tok s určitým kódováním. Paralelizovaný výstup bude mít nižší frekvenci podle poměru šířky vstupu ku výstupu. Vysílač naopak vstupní paralelní data serializuje a zakóduje. Transciever GTY (na Ultrascale+) podporuje kódování NRZ (Non Return to Zero – bez návratu k nule) s maximální rychlostí 32,75 Gb/s [27]. Transciever GXE na Stratix 10 provádí jak kódování NRZ s maximální datovou rychlostí 28,9 Gb/s, tak i PAM4 (Pulse Amplitude Modulation – pulzně amplitudová modulace) kódování s datovou rychlostí až 57,8 Gb/s (tedy dvojnásobek) [29]. To je klíčovým parametrem FPGA v síťové kartě zvládající přenosovou rychlost 400 Gb/s po Ethernetu.

1.4 Návrh digitálního obvodu na FPGA

Digitální obvod pro FPGA bývá zpravidla vytvářen podle postupu skládajícího se z několika kroků, které jsou znázorněny na obrázku 1.7. Na začátku je vstupní soubor, kde je nějakým způsobem popsán navržený obvod. Ten je vysyntetizován (co je syntéze je vysvětleno níže), přiřazen k fyzickým prvkům FPGA a pak převeden na formát, který je nakonec do FPGA nahrán. Navržený obvod je důležité důkladně testovat, což se děje pomocí průběžných verifikací, viz pravá část obrázku 1.7. Po neúspěšné verifikaci je upraven vstupní návrh a postup vytváření obvodu se opakuje [30].

Vstupním návrhem nejčastěji bývají soubory obsahující návrh obvodu buď ve formě schématu nebo HDL (Hardware Description Language – jazyk popisující hardware) kódu [30]. Návrh pomocí schématu má výhodu v tom, že návrhář má celkový přehled o vytvářeném obvodu, navíc také přímo souvisí s hardwarem, takže je přímo vidět prvky, ze kterých se navrhovaný obvod skládá, a lze jednodušeji poznat jeho funkci. Navíc pro vytvoření schématu není zapotřebí žádných programovacích dovedností. Oproti tomu je návrh za pomocí HDL jazyku výhodný v tom, že umožňuje parametrizovat obvody – příklad vztažený k této práci je šířka výstupu čítače – dále je jednodušší takovýto návrh optimalizovat a syntetizovat (z pohledu nástrojů), je to také vhodné pro vytváření různých algoritmů a konečně pro napsání HDL kódu je potřeba pouze textový editor [31].

Nejrozšířenější HDL jazyky jsou VHDL, Verilog a SystemVerilog. První dva jmenované slouží primárně k popisu digitálních obvodů, zatímco SystemVerilog, jenž je nadstavbou Verilogu, je přizpůsoben pro tvorbu verifikací. Jazyk VHDL vychází z programovacího jazyka Ada a to z pohledu jeho konceptu



Obr. 1.7: Postup vytváření obvodu na FPGA [30, 32]

i syntaxe. Verilog je spíše založen na programovacím jazyku C. Praktická část této práce byla napsána v jazyce VHDL, jenž je sdružením CESNET pro návrh digitálních obvodů používán. [33]

Nad vstupními soubory se následně provádí syntéza. Tuto činnost provádí syntézní nástroj. Výrobci FPGA mají své vlastní syntézní nástroje – Xilinx má Vivado Design Suite [34] a Intel Quartus® Prime [35] – ale lze použít i nástroje třetích stran. Tyto nástroje však neznají různé knihovny a jiná duševní vlastnictví (IP – Intellectual Property) výrobců, jako jsou například DSP bloky.

Při syntéze dojde ke kontrole ucelenosti návrhu a správnosti syntaxe. Také se zde provádí různé minimalizace a optimalizace, například dochází k vyřazení nadbytečné logiky. Mohou zde být také definována různá omezení, jako třeba frekvence hodin nebo přiřazení k pinům FPGA. Výstupem procesu syntézy je tzv. netlist, což je síť složená z hradel, registrů, propojů a dalších základních prvků FPGA. Před samotnou syntézou může být provedeno ověření chování obvodu, tedy zda-li se navržený obvod chová tak, jak je zamýšleno (viz blok simulace chování obvodu na obrázku 1.7) [35, 36].

Po syntéze následuje krok implementace, kdy jsou prvky netlistu přiřazeny k fyzickým prvkům FPGA s ohledem na definovaná omezení. Jedná se o umístění do logických bloků a jejich propojení, jinak běžně nazýváno jako „Place and Route“. Výsledek se ověřuje jednak z pohledu funkčnosti (viz simulace funkce obvodu na obrázku 1.7) a jednak z pohledu časování, kdy jsou analyzovány jednotlivé zpoždění signálů a jsou nalezeny všechny kritické cesty (na obrázku 1.7 se jedná o simulaci časování) [32, 35, 37].

Aby takto vytvořený obvod mohl být nahrán do FPGA, musí být výstup přeformátován. Toto se děje v posledním kroku vytváření digitálního obvodu pro FPGA. Výstup z implementace (Place and Route) je převeden na proud bitů a následně uložen do konfiguračního souboru, který lze poté

nahrát do FPGA, a podle něj pak dojde k fyzickému vytvoření obvodu. Po konfiguraci zařízení je možné provést verifikaci v hardwaru pomocí nástrojů, pomocí kterých je možné nahlédnout do vnitřku FPGA. Je potřeba mít předem jasno v tom, které signály mají být sledovány. Během syntézy je do vytvářeného obvodu připojena speciální část, která bude sbírat data. Tato data se poté přehrají do počítače a zobrazí se pomocí uživatelského rozhraní [30, 32, 36, 38].

Pro simulace se může používat vestavěný simulátor v syntézním nástroji, pokud je v něm obsažen, nebo simulační program třetí strany, jako je například ModelSim, který je podporován výrobci Intel i Xilinx. Do ModelSimu je možné přidat knihovny těchto výrobců, takže funkci obvodu je možné ověřit i se všemi dílčími prvky daného FPGA. Ke spuštění simulace je kromě testovaného obvodu také potřeba tzv. „testbench“, kde jsou vytvořeny testovací signály, které se připojí na vstupy daného obvodu. Výstupem simulace bývá časový diagram, ve kterém jsou veškeré signály testovaného obvodu, a podle nich je určována správnost jeho chování [37, 39].

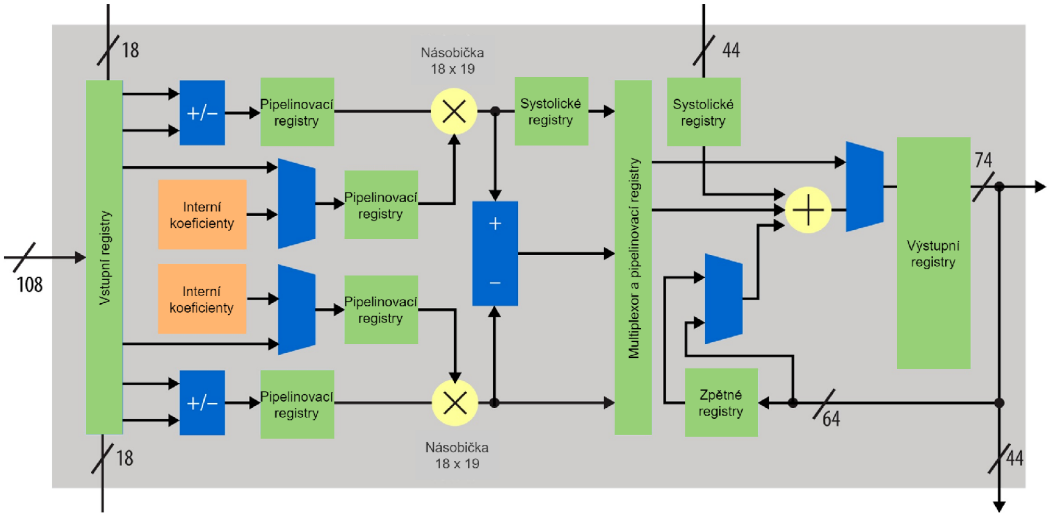
1.5 DSP

FPGA jsou efektivní zařízení pro zpracování digitálních signálů, protože umožňují implementace plně paralelizovaných algoritmů na míru požadovaným aplikacím. Často používané operace, jako jsou násobení a akumulování, je vhodné implementovat vyhrazenými prvky, které jsou na to přizpůsobeny. V FPGA jsou takovými vyhrazenými prvky DSP bloky [40]. Jsou to tedy dedikované hardwarové celky, které dále zlepšují parametry zpracování digitálních signálů v FPGA. Jak již bylo naznačeno v kapitole 1.3, jejich využitím je možné ušetřit energii, mnohé zdroje a plochu FPGA a dosáhnout vyšších rychlostí výpočtů. DSP bloky je možné za běhu přenastavit, aby v každém taktu prováděly jiný výpočet [41]. Příklady digitálního zpracovaní signálů jsou FIR (Finite Impulse Response – konečná impulzní odezva) filtry [42], kde se využívá násobení komplexních čísel, a FFT (Fast Fourier Transform – rychlá Fourierova transformace), dalšími možnými aplikacemi DSP bloků jsou široké posuvné registry a multiplexory nebo generátory adres pro paměti.

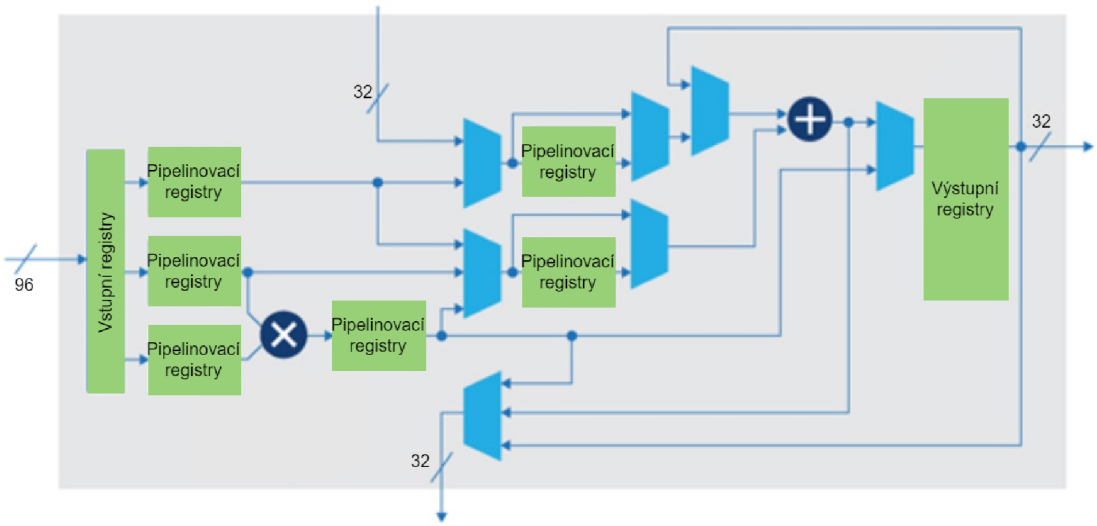
Syntézní nástroje výrobců automaticky využívají DSP bloky integrované v daném FPGA, i když limitovaným způsobem. Za správně napsaného HDL kódu dokáže syntézní nástroj využít dané DSP bloky, jinak spíše použije logické prvky. Návrhář může instancovat DSP blok jako komponentu, nastavit si ji podle potřeb a poté zapojit jako komponentu do obvodu.

1.5.1 DSP ve Stratix 10

Veškeré informace z této části pochází z oficiální příručky od firmy Intel včetně obrázků, viz [42]. DSP s proměnlivou přesností („variable-precision DSP“) nacházející se na Stratix 10 dokáže pracovat jak s pevnou řádovou („fixed-point“) čárkou, tak i s plovoucí („floating-point“). Pro tyto dvě implementace existují dvě různé architektury DSP, viz obrázky 1.8 a 1.9. Na obrázku 1.8 je zobrazeno schéma DSP bloku se standardní přesností („standard-precision“) pro „fixed-point“ implementaci, konkrétně pro nastavení 18×19. V nastavení 27×27 jsou spodní násobička a odpovídající předčítačka vynechány a vstupy násobičky dosahují šířky až 27 b. Na obrázku 1.9 je zobrazeno schéma pro „floating-point“ implementaci.



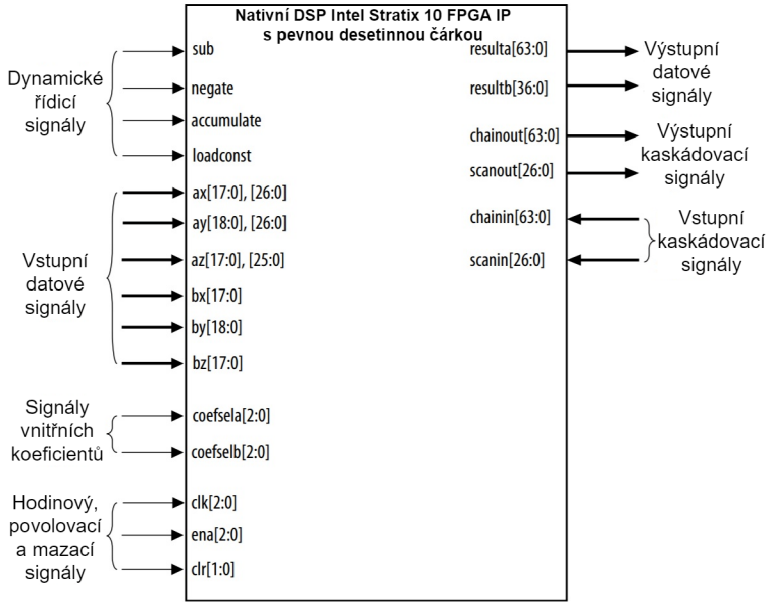
Obr. 1.8: Základní schéma DSP bloku na Stratix 10 pro „fixed-point“ implementaci [42]



Obr. 1.9: Základní schéma DSP bloku na Stratix 10 pro „floating-point“ implementaci [42]

Vstupní a výstupní signály DSP

Implementace „fixed-point“ a „floating-point“ se liší už vstupními signály. Ve „fixed-point“ implementaci DSP bloku jsou k dispozici řídicí signály *sub*, *negate*, *accumulate*, *loadconst*, které je možné za běhu měnit, viz obrázek 1.10. Ve „floating-point“ implementaci (až na obrázku 1.12) se řídicí signály *sub*, *negate* a *loadconst* nenachází.



Obr. 1.10: Blokově zobrazené DSP ve „fixed-point“ implementaci [42]

Signál *sub* rozhoduje, jestli se výstupy z násobiček budou sčítat (logická nula) nebo odčítat (logická jednička). Signál *negate* mění příchozí hodnoty na opačné tak, že z nich udělá dvojkový doplněk. Tím se tedy určí, zda-li se příchozí data s těmi ze zřetězení nebo akumulování budou sčítat (*negate* v logické nule) a nebo odčítat (*negate* v logické jedničce). Signál *accumulate* bude již podle názvu řídit akumulaci dat, a to tím způsobem, že data z výstupu přivádí na jeden ze vstupů zřetězovací sčítačky/akumulátoru (na obrázku 1.8 jako „+“). Když je nastaven do logické jedničky, data z výstupu (případně z registru pro dvojitou akumulaci) jsou přiváděna na vstup zřetězovací sčítačky/akumulátoru. Je-li nastaven do logické nuly, do zřetězovací sčítačky/akumulátoru je přiváděna hodnota konstanty, která je nulová tehdy, když signál *loadconst* je také nastaven do logické nuly (nebo když je hodnota konstanty nulová). Signál *loadconst* nastavený do logické jedničky načte do zřetězovací sčítačky/akumulátoru ručně nastavenou konstantu už při vytváření obvodu. Konstanta je nastavena libovolným číslem N v rozmezí od 0 do 64. Hodnota konstanty je pak číslo 2^N. A protože výstup konstanty je 64b číslo, bude hodnota konstanty nula, když $N = 64$. *Scanin* (skenovací vstup) i *scanout* (skenovací výstup) slouží pro řetězení vstupů DSP bloků. V případě potřeby zřetězení vstupů DSP bloků je *scanin* jednoho DSP připojen na *scanout* předchozího DSP a *scanout* tohoto DSP připojen na *scanin* dalšího. Signál *scanin* pak nahrazuje vstupní data *ay*, případně i *by*. Výstupy z DSP bloků se řetězí podobně jako vstupy. Používají se na to signály *chainin* (zřetězovací vstup) a *chainout* (zřetězovací výstup). Na *chainin* se připojí

chainout předchozího DSP a *chainout* se připojí na *chainin* dalšího DSP. Signály *ay*, *az* a *ax*, případně i *by*, *bz* a *bx* nesou vstupní data a *resulta* (výsledek a) a *resultb* (výsledek b) jsou výstupy z DSP mající určitou datovou šířku. Na obrázku 1.10 jsou šířky vstupů i výstupů označeny v hranatých závorkách hned za názvy signálů. V případě „fixed-point“ architektury existuje již dříve zmíněné nastavení DSP 27×27, kde vstupy mají šířku 27 b a výstup 64 b.

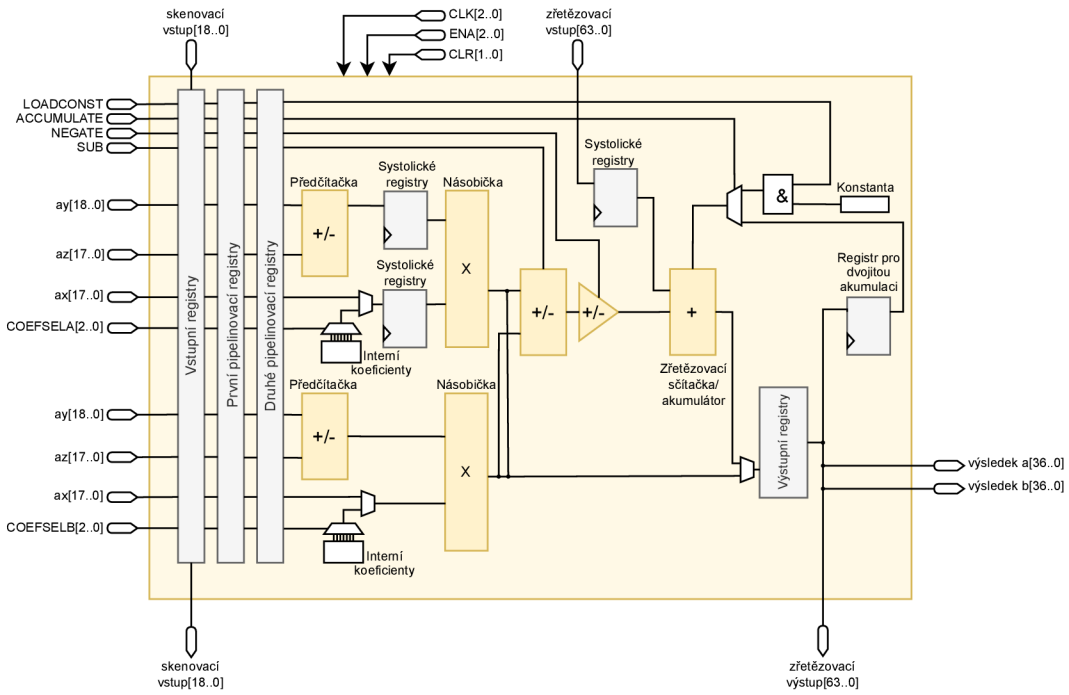
Ve „floating-point“ implementaci DSP bloku (až na obrázku 1.12) se také nachází signály *chainin* a *chainout* s tím, že že mohou být nastaveny i jako *scanin* a *scanout*, ale mohou být nastaveny jen jako jedno nebo druhé, nikoliv obojí naráz. Vstupní a výstupní datové signály mají odlišnou šířkou, ale jinak jsou stejné jako v předchozí implementaci. V případě zřetězení vstupů DSP bloků (tedy *scanin*) bude deaktivován vstup *ax*, stejně jako ve „fixed-point“ implementaci. Navíc zde oproti „fixed-point“ implementaci existují signály, které jsou schopny samostatně identifikovat a zahlásit chybu (tzv. „exception handling“) u výsledků násobiček a sčítaček. Chybové signály jsou v logické nule v případě, že se o danou výjimku nejedná, nebo v jedničce, když se o výjimku jedná. Detekované výjimky jsou: *mult_overflow*, *mult_underflow*, *mult_inexact*, *mult_invalid*, *adder_overflow*, *adder_underflow*, *adder_inexact* a *adder_invalid*. *Mult_overflow* oznamuje přetečení výsledku násobičky (tj. výsledná hodnota je větší než maximální vyjádřitelná) a výsledkem je maximální vyjádřitelná hodnota. U výjimky *mult_underflow* se jedná o podtečení výsledku násobičky (tj. výsledek je menší než je možné vyjádřit) a výsledek je pak vyjádřen nulou. Logická jednička signálu *Mult_inexact* signalizuje nepřesné vyjádření výsledku, tedy v případě přetečení, podtečení, nebo zaokrouhlení výsledku násobičky. Výjimka *mult_invalid* hlásí nedefinovanou operaci násobičky, při které vyšel neplatný výsledek. Ve výsledku pak bude hodnota qNaN (Quiet Not A Number – není číslo). Zbylé čtyři výjimkové signály mají stejnou funkci, jako ty předchozí, jenom dané výjimky hlásí podle výsledku ze sčítaček (případně odčítaček). Další kontrolní signály, které jsou k nalezení ve fixed-point implementaci (například *loadconst*), se zde neuplatňují.

Registry v DSP jsou řízeny signály *clk*, *ena* a *clr*. Všechny registry reagují na náběžnou hranu a mohou být řízeny libovolně až třemi různými hodinovými signály – *clk*[0], *clk*[1] A *clk*[2]. Pro vypuštění registru z obvodu se jeho hodinový signál nastaví na „none“. Povolení a zákaz zápisu do registrů je řízeno *ena* (enable – povolit) signálem. Signál *ena*[0] zakazuje zápis registrům běžících na *clk*[0], *ena*[1] registrům běžícím na *clk*[1] a *ena*[2] registrům na *clk*[2]. Signál *clr* (clear – vymazat) maže data uložená v registrech a to buď synchronně, tedy že daný *clr* signál musí být v logické jedničce v okamžiku náběžné hrany hodinového signálu, anebo asynchronně, kdy vymazaní může proběhnou kdykoliv, nezávisle na hodinovém signálu. Signál *clr*[0] se vztahuje na vstupní registry a *clr*[1] na pipelinovací (neboli zpožďovací) a výstupní registry.

Operační módy s pevnou desetinnou čárkou

Na obrázku 1.10 je zobrazena „fixed-point“ implementace DSP jako černá skříňka se vstupními a výstupními signály. Podle různých nastavení řídících signálů může DSP blok operovat v několika módech popsaných dále. Architektura DSP bloku ve „fixed-point“ implementaci je podrobně znázorněna na obrázku 1.11.

V módu „nezávislá násobička“ jsou obě dvě násobičky v DSP využívány zvlášť pro obecné násobení. Každá z násobiček má vlastní nezávislé vstupy a výstupy. Vstupy mohou být jak typu signed (se znaménkem) tak i unsigned (bez znaménka). Výstupní signál *resulta* pak obsahuje výsledek z první násobičky a *resultb* výsledek druhé násobičky, viz rovnice 1.1 a 1.2. V případě 27×27 se jedná o rovnici



Obr. 1.11: Architektura „fixed-point“ implementace DSP bloku [42]

1.1, rozdíl je jen v šířce vstupních a výstupních dat.

$$ax \times ay = resulta, \tag{1.1}$$

$$bx \times by = resultb. \tag{1.2}$$

Mód „sčítání součinů“ je mód, při kterém jsou výsledky z násobiček sečteny, případně odečteny v závislosti na řídícím signálu *sub*. Výstupem je jen signál *resulta*. Rovnice 1.3 bude platit, když *sub* bude v logické nule, jinak bude platit rovnice 1.4.

$$(ax \times ay) + (bx \times by) = resulta \tag{1.3}$$

$$(ax \times ay) - (bx \times by) = resulta \tag{1.4}$$

Mód „komplexní násobení“ využívá dva DSP bloky nastavené do módu „sčítání součinů“. Vyplývá to z principu násobení komplexních čísel:

$$(a + \mathrm{j}b) \times (c + \mathrm{j}d) = [(a \times b) - (c \times d)] + \mathrm{j}[(a \times d) + (b \times c)]. \tag{1.5}$$

V prvním DSP se provede výpočet imaginární části, tedy $(a \times d) + (b \times c)$, a v druhém výpočet reálné, tedy $(a \times b) - (c \times d)$.

Dalším módem je „násobení a součet“, kde první dva vstupy *ax* a *ay* vstupují do násobičky a jejich součin je poté sečten s (nebo odečten od) třetím, až dvojnásobně širším vstupem. Tento třetí vstup (označen jako *bx*) se skládá z dvou normálních vstupů, *bx* a *by*, a je rozdělen tak, že vstup *by* pokrývá

první část datového slova ($by[17..0]$) a vstup bx pokrývá druhou ($bx[35..18]$). Rovnice pro tento mód závislé na řídícím signálu sub jsou:

$$ax \times ay + bx = resulta, \quad \text{když } sub = 0, \text{ nebo} \tag{1.6}$$

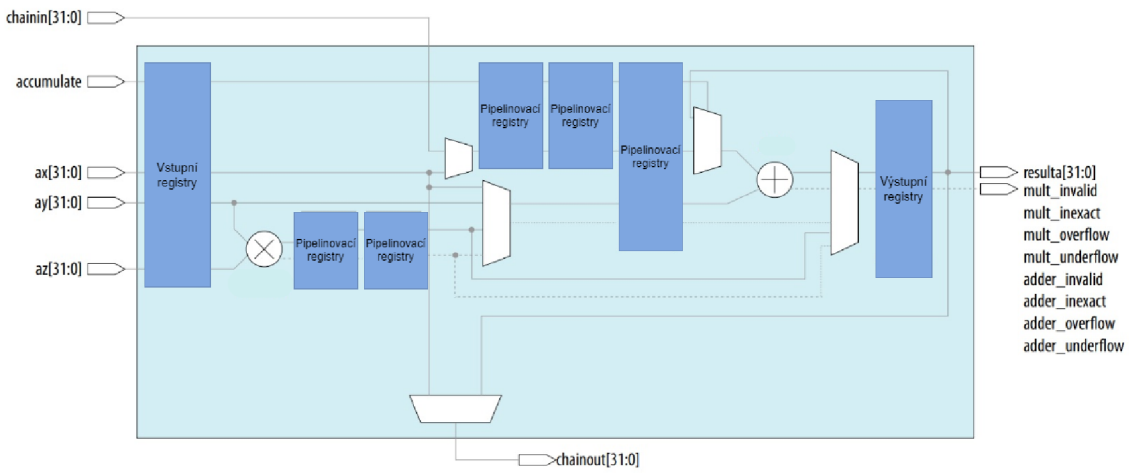
$$ax \times ay - bx = resulta, \quad \text{když } sub = 1. \tag{1.7}$$

Posledním módem je „systolický FIR“ mód, který se používá pro filtry s konečnou impulzní odezvou (FIR filtry). U těchto filtrů se jedná o několik postupných násobení s následným sečtením, viz rovnice 1.8. V tomto módu vstupy násobiček mohou pocházet kromě dynamických vstupů také z předčítaček nebo ze zásobníku interních koeficientů, který uchovává až 8 různých koeficientů, mezi kterými se dá za běhu přepínat. Hodnoty koeficientů se ale nastavují před vytvářením obvodu. Ačkoliv je „systolický FIR“ mód poměrně rozsáhlý, nebude zde podrobněji rozebírám, protože pro tuto práci není důležitý.

$$y[n] = \sum_{i=1}^{k} w_i[n-k+i] + w_1[n-k+2], \quad \text{kde } w_i[n] = c_i x[n-2i+2]. \tag{1.8}$$

Operační módy s plovoucí desetinnou čárkou

Funkční schéma architektury DSP bloku pro „floating-point“ implementaci je na obrázku 1.12. V této implementaci se módy dělí na dvě skupiny podle počtu užitých DSP bloků (jedno použité DSP nebo více použitých DSP).



Obr. 1.12: Funkční schéma floating-point DSP [42]

Jedno samostatné DSP ve „floating-point“ implementaci dokáže operovat ve třech módech. Prvním je „násobící“ mód. Tento mód vykonává základní násobení s plovoucí desetinnou čárkou, jak ukazuje rovnice 1.9. Detekují se výjimky *mult_overflow*, *mult_underflow*, *mult_inexact* a *mult_invalid*, viz výše.

$$ay \times az = resulta \tag{1.9}$$

Druhý je „sčítací nebo odčítací“ mód. Podle přednastavení sčítačky se data ze vstupů buď sečtou, nebo odečtou, viz rovnice 1.10. Jsou detekovány výjimky *adder_overflow*, *adder_underflow*, *adder_inexact* a *adder_invalid*, které byly zmíněné již výše.

$$ax \pm ay = resulta \tag{1.10}$$

Třetím módem je „násobení s akumulací“, při kterém jsou vstupní data vynásobena a poté buď sečtena (či odečtena) s předchozím výsledkem, to nastane v případě, že řídící signál *accumulate* je v logické jedničce a výstup je pak podle rovnice 1.11, nebo rovnou převedena na výstup (když je *accumulate* v logické nule). Výstup je pak podle rovnice 1.12. Zde se detekují všechny dříve zmíněné výjimky.

$$ay \times az \pm predchozi\_hodnota = resulta \tag{1.11}$$

$$ay \times az = resulta \tag{1.12}$$

Při využiti více DSP bloků je na výběr z pěti módů. Prvním módem je „sečtení součinů nebo odečtení součinů“. V tomto módu není výjimečně nutné propojení s dalšími DSP bloky. Při zakázaném zřetězovacím vstupu *chainin* se výsledek násobičky sečte (nebo odečte podle nastavení sčítačky) se třetím vstupem podle rovnice 1.13. Naopak při povoleném zřetězovacím vstupu je výsledek násobičky sečten (nebo odečten) se zřetězovacím výstupem předchozího DSP bloku. Tento případ popisuje rovnice 1.14. Také zde se detekují všechny vyjmenované výjimky.

$$ay \times az \pm ax = resulta \tag{1.13}$$

$$ay \times az \pm chainin = resulta \tag{1.14}$$

Druhý mód se nazývá „vektor jedna“. Výsledek násobičky se sečte (nebo odečte) se zřetězovacím vstupem, pokud je povolen, viz rovnice 1.15. Jinak se ponechá tak, jak je (viz rovnice 1.16). V každém případě je vstup *ax* připojen na zřetězovací výstup *chainout*, viz rovnice 1.17. Opět jsou detekovány všechny druhy výjimek.

$$ay \times az \pm chainin = resulta \tag{1.15}$$

$$ay \times az = resulta \tag{1.16}$$

$$ax = chainout \tag{1.17}$$

Mód s názvem „vektor dva“ posílá výstup z násobičky rovnou na zřetězovací výstup *chainout*. Výstup DSP závisí na skutečnosti, zda-li je, anebo není, povolen zřetězovací vstup *chainin*. Pokud je povolen, sčítá (případně odčítá) se se vstupem *ax*, jak je znázorněno rovnicí 1.19. Pokud není povolen, vstup *ax* se rovnou přenáší na výstup DSP (viz rovnice 1.20. Jsou detekovány všechny druhy výjimek.

$$ay \times az = chainout \tag{1.18}$$

$$ax \pm chainin = resulta \tag{1.19}$$

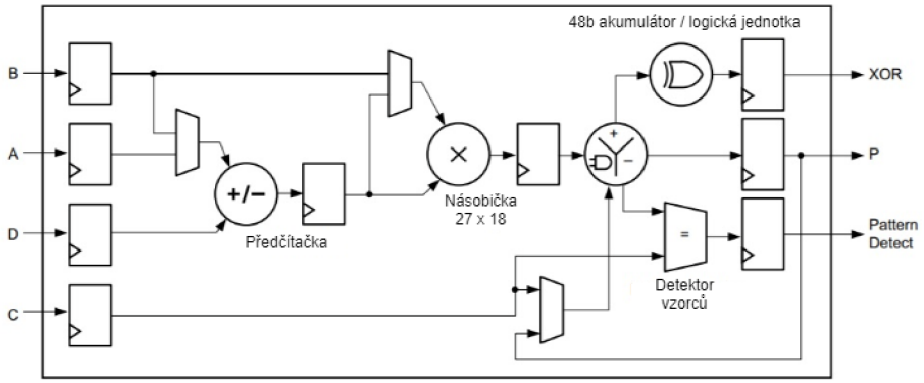
$$ax = resulta \tag{1.20}$$

Dalším módem je „direct vector dot product“. Je realizován zapojením několika DSP ve výše zmíněných módech: „sečtení nebo odečtení součinů“ s povoleným zřetězovacím vstupem, „vektor jedna“ a „vektor dva“. Detekují se všechny druhy výjimek.

Posledním módem je „komplexní násobení“, neboli násobení komplexních čísel. Princip je úplně stejný, jako v dříve zmíněném módu komplexního násobení s pevně danou desetinnou čárkou. Rozdíl je v realizaci, kde pro násobení komplexních čísel s plovoucí desetinnou čárkou jsou potřeba dva DSP bloky na výpočet reálné části a dva na výpočet imaginární části. Je to z toho důvodu, že DSP ve „floating-point“ implementaci má jen jednu násobičku a pro výpočet jak reálné, tak imaginární části jsou potřeba dvě násobení, viz rovnice 1.5.

1.5.2 DSP v UltraScale+

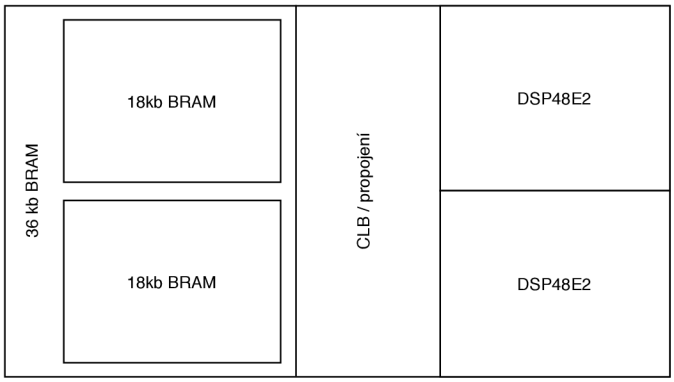
Veškeré informace z této části pochází z oficiální příručky od firmy Xilinx, viz [40]. V FPGA UltraScale+ se nachází DSP bloky s názvem DSP48E2. Jeho funkční schéma je na obrázku 1.13. zdejší DSP



Obr. 1.13: Funkční schéma DSP bloku na UltraScale+ [40]

blok nemá dvě různé implementace pro „floating-point“ a „fixed-point“ aritmetiku, jak tomu bylo u DSP na Stratix 10, tato jediná implementace dokáže počítat s obojím.

DSP bloky jsou umístěny v tzv. DSP tile (DSP dlaždicích), viz obrázek 1.14. V jedné takové dlaždici



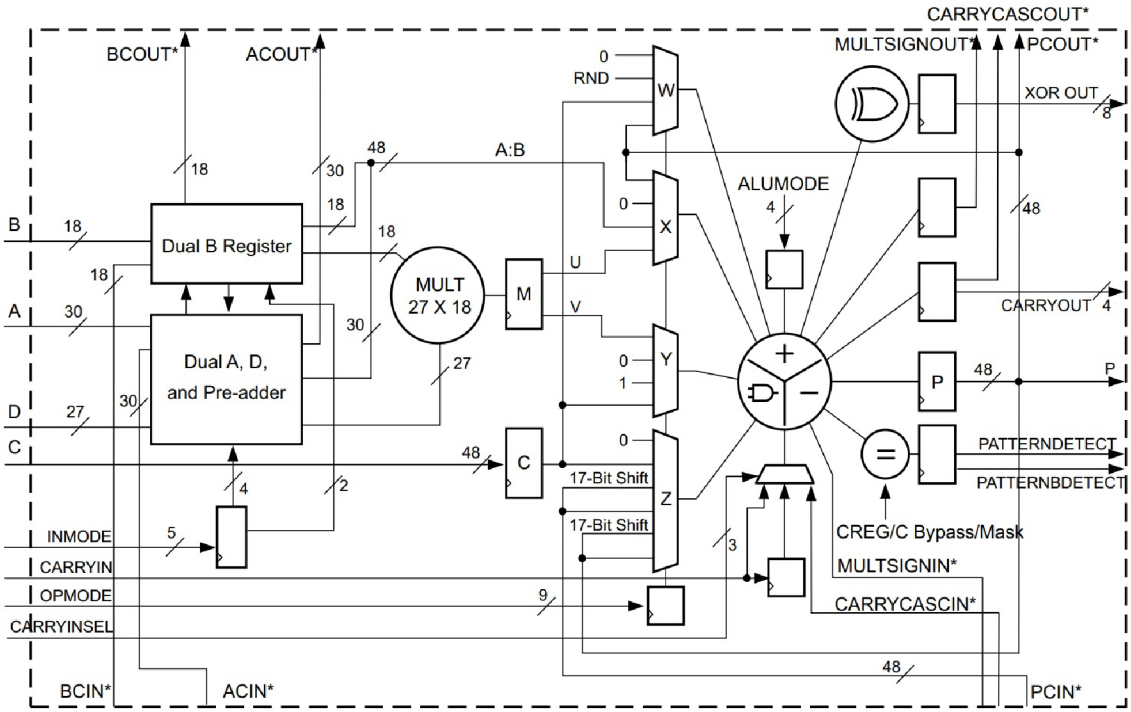
Obr. 1.14: DSP dlaždice v UltraScale+ [40]

se nachází 2 DSP bloky s vyhrazeným propojením například k BRAM, CLB nebo dalšímu DSP. Má stejnou šířku jako 5 CLB nebo jedna 36kb BRAM. BRAM paměti jsou v DSP dlaždicích rozděleny na dvě 18kb a jsou zarovnané naproti DSP blokům, což ústí ve skvělé propojení mezí těmito zdroji. DSP dlaždice se v FPGA řadí svisle pod sebe, viz obrázek 1.4.

V UltraScale+ je až 12 DSP dlaždic v jednom regionu hodinového signálu. DSP bloky je možné řetězit i napříč regiony hodinových signálů, ale pak nelze zajistit maximální výkon.

Vstupní signály

Detailnější znázornění DSP48E2 je na obrázku 1.15. Má 4 přímé datové vstupy: A, B, C a D s maximálními šířkami znázorněnými na obrázku.



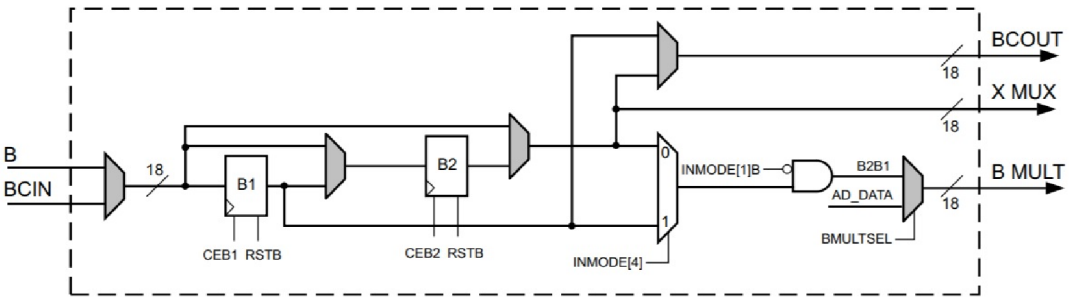
Obr. 1.15: Detailní funkční schéma DSP bloku na UltraScale+ [40]

Pokud jsou vstupní signály A a B operandy násobičky, je maximální šířka signálu A 27 b (místo 30 b). Tyto vstupy společně s nezávislým vstupem C mohou provádět operace jako jsou součin s následným sečtením, součin s následným odečtením a součin se zaokrouhlením.

Z DSP lze vytvořit dvojvstupou 48b sčítačku nebo odčítačku tím způsobem, že se spojí vstupy A a B, čímž vytvoří jeden 48b vstup, druhým vstupem sčítačky nebo odčítačky je pak vstup C, který je už ve výchozím nastavení 48b. Signál A obsahuje vrchních 30 b a signál B spodních 18 b, tedy $A[47..18]$ a $B[17..0]$. Na obrázku 1.15 je toto spojení označeno jako $A{:}B$. Aby vstupy A a B mohly být takto spojeny, musí být vyřazena násobička.

Signály *ACIN* a *BCIN* jsou vstupními zřetězovacími vstupy, pomocí nichž lze zapojit do kaskády více DSP bloků. Tato možnost je vhodná pro aplikace využívající komplexní násobení, FIR filtry, násobení s velkou přesností a komplexní MACC (Multiply And Accumulate – násobení s akumulací).

Vstupy *A*, *B*, *ACIN* a *BCIN* je možné pipelinovat, což se provádí v blocích *Dual B Register* a *Dual A, D, and Pre-adder* na obrázku 1.15. Blok *Dual B Register* je dále podrobněji rozkreslen na obrázku 1.16, druhý zmíněný blok je až na obrázku 1.17, v části, kde je podrobněji popsána funkce sčítačky. Počet pipelinovacích registrů je možné dynamicky nastavovat řídícím signálem *INMODE*.



Obr. 1.16: Detailní pohled na blok Dual B Register [40]

Vyšší počet znamená lepší výkon DSP (dokáže pracovat ve vyšších frekvencích) na úkor většího zpoždění (větší počet taktů od vstupu dat do DSP po opuštění DSP). Uvnitř těchto bloků je to provedeno tak, že se daný signál rozdělí na dva a jeden prochází pipelinovacím registrem a poté na jeden vstup multiplexoru, zatímco ten druhý vede přímo na druhý vstup multiplexoru, a jedním bitem z řídícího signálu *INMODE* se rozhodne, zda-li se na výstup multiplexoru dostane signál z registru nebo ten druhý, neproregistrovaný. Ostatní statické řídící signály vybírají data například pro vstup do předčítačky (signál *D* je možné sečíst se signálem *A* nebo se signálem *B*) nebo pro výstupní signály z těchto bloků, tj. signály *ACUOT*, *X MUX* a *A MULT* z bloku *Dual A, D, and Pre-adder*, respektive *BCOUT*, *X MUX* a *B MULT* z bloku *Dual B Register*.

Vstup *C* je obecný vstup, který vede do multiplexorů *W*, *Y* a *Z*, přičemž je možné jej vést přes jeden registr. Účastní se různých operací, jako je sčítání, odčítání a logické operace. Při potřebě zaokrouhlování vstupuje do prvku rozpoznávající vzorce, který je na obrázku 1.15 označen jako „=“, anglicky „Pattern Detector“.

Výstupní signály

Standardním výstupem DSP48E2 je 48b výstup *P*. Pro zřetězení více DSP bloků za sebou se používá výstup *PCOUT*, který je připojen k vstupu *PCIN* sousedního DSP.

Výstupy *CARRYOUT* a *CARRYCASCOUT* přenášejí přetečená data. *CARRYOUT* je 4 b široký a je možné jej vést do okolní logiky. Pokud je prvek sčítačka/odčítačka/logická jednotka, který je na obrázku 1.15 mezi multiplexory *W*, *X*, *Y*, *Z* a výstupními registry, nastaven pro vykonávání jedné 48b operace, pak je pro přenos použit pouze nejvýznamnější bit signálu *CARRYOUT* (tj. *CARRYOUT*[3]). Tento prvek ovšem může fungovat také v módu „SIMD“ (Simple Instruction Multiple Data – jednoduchá operace prováděná na vícero datech současně), který je podrobněji popsán níže. Jednoduše

řečeno, místo jedné 48b operace, jako je třeba sčítání, může provádět v jednom taktu dvě a to tak, že se rozdělí na dvě nezávislé sčítačky s užšími vstupy. Může se také rozdělit na čtyři nezávislé sčítačky s ještě užšími vstupy. Z tohoto prvku tedy mohou vést až čtyři různé výstupy a pro každý je určen jeden bit z *CARRYOUT* signálu. Tento signál není platný při nastavení prvku sčítačka/odčítačka-/logická jednotka jako tří- nebo čtyřvstupá sčítačka (například $A{:}B + C + P$), dvoj- nebo třívstupý akumulátor nebo při použití násobičky.

CARRYCASCOUT přenáší *CARRYOUT* signál do sousedního DSP. Tento signál je ovšem 1b, přenáší jen nejvýznamnější bit *CARRYOUT* signálu (tj. *CARRYOUT*[3]). Pomocí tohoto signálu lze realizovat širší funkce, například sčítačky, akumulátory apod. Navíc je *CARRYCASCOUT* přiveden zpět na vstup daného DSP a to přes kaskádovací multiplexor řízený signálem *CARRYINSEL*.

MULTSIGNOUT je 1b signál, který se používá jen pro vytvoření 96b MACC funkce. Má reprezentovat nejdůležitější bit výstupu z násobičky.

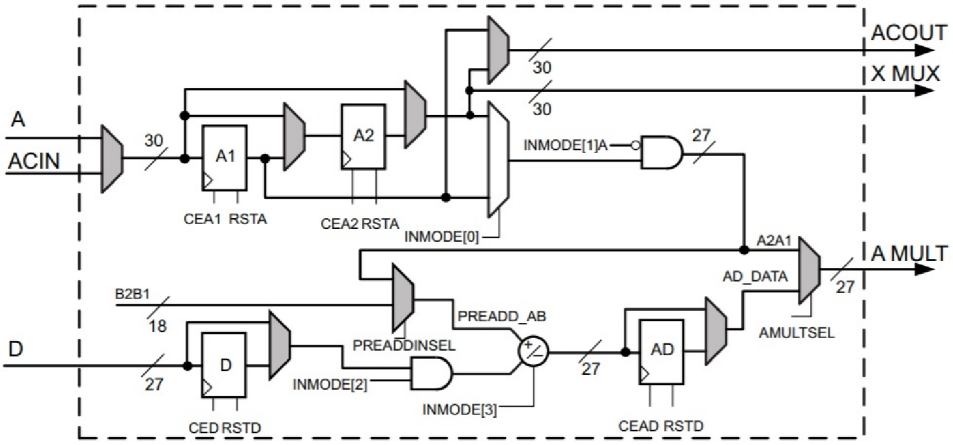
Signály *PATTERNDETECT* a *PATTERNBDETECT* jsou výstupy z prvku rozpoznávajícího vzorce. Tento prvek detekuje shodu výstupu P se zadaným vzorcem (*Pattern*), který může přijít ze vstupu C nebo být zadán maskou. Při přímé shodě výstupu P se vzorcem nastaví signál *PATTERNDETECT* do logické jedničky. Při shodě výstupu P s invertovaným vzorcem nastaví do logické jedničky signál *PATTERNBDETECT*. Pro detekci shody se po bitech se provede logická operace $((P == Pattern) \,||\, mask)$ a nad všemi bity výsledného signálu se udělá AND. Slovně se to dá vysvětlit tak, že pokud nesouhlasí jakýkoliv z bitů signálu P s odpovídajícím bitem signálu *Pattern* a zároveň není překryt maskou, na výstupu *PATTERNDETECT* bude logická nula. Z toho vyplývá, že maska má tedy jedničky na pozicích těch bitů, na kterých nezáleží (mají být maskovány). *PATTERNBDETECT* je počítán obdobně, logická operace AND všech bitů se provede na výsledek operace $((P == Pattern) \,||\, !mask)$, kde $!mask$ značí převrácenou masku (jedničkový doplněk).

Detekce přetečení (výstup *OVERFLOW*) nebo podtečení (výstup *UNDERFLOW*) využívá také prvek rozpoznávající vzorce. Detekuje se, jestli výstup P přetekl přes *P[N]* bitů, kde N může nabývat hodnot od 1 do 46. Při využití detekce přetečení nebo podtečení musí být v obvodu DSP povolen registr pro výstup P. Tato funkce je podrobněji popsána níže.

Vestavěné funkce, mód „SIMD“, rozpoznávání vzorců a široký XOR

V dokumentaci k DSP od firmy Xilinx [40] nejsou tak podrobně popsány jednotlivé módy DSP bloků, jak tomu je v dokumentaci od firmy Intel [42]. Jeden mód, který je podrobněji popsán, je mód „SIMD“, ostatní jsou zmíněny v [40] v tabulkách číslo 2-12 až 2-17. Místo toho jsou zde blíže vysvětleny vestavěné funkce, mezi které patří předčítačka, násobička pracující s čísly ve dvojkovém doplňku a prvek sčítačka/odčítačka/logická jednotka. Dále je podrobněji vysvětlena logika pro rozpoznávání vzorců, se kterou úzce souvisí logika přetečení a podtečení, a nakonec široká logická operace XOR.

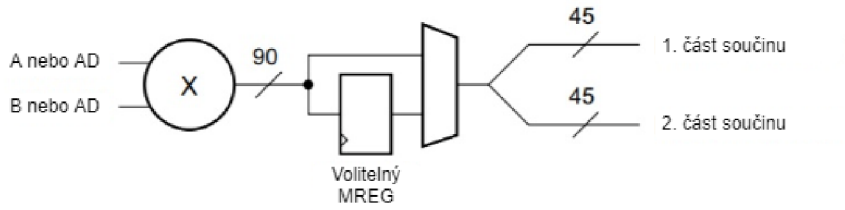
Blok předčítačky je zobrazen na obrázku 1.17. Jeho funkcí je umožnit sečtení nebo odečtení vstupních signálů předtím, než přijdou do násobičky. Vstupy do předčítačky (na obrázku 1.17 je čítací prvek vyobrazen jako „+/-“) jsou datové vstupy A nebo B (případně *ACIN*, resp. *BCIN*) a D. O tom, zda-li se vezme vstup A nebo B rozhoduje multiplexor s řídícím signálem *PREADDINSEL*. Je na návrhářovi, aby výsledek předčítačky nebyl moc široký a nedošlo k přetečení či podtečení, není zde žádná logika, která by to hlídala. Je možné předčítačku obejít, v takovém případě bude vstup D druhým vstupem násobičky. Pokud vstup D není použit, je možné vstup A nebo B znegovat před vstupem do násobičky. Tento blok je velmi flexibilní, dokáže operovat až v 15 módech, včetně mocnění na druhou.



Obr. 1.17: Podrobnější pohled na blok předčítačky v DSP48E2 [40]

Jak již bylo zmíněno, násobička nacházející se na DSP48E2 pracuje s čísly (vstupy i výstupy) v dvojkovém doplňku, což je srovnatelné s typem signed v DSP od firmy Intel. Pokud je požadována práce se signály pouze typu unsigned, nejdůležitější bit se nastaví na nulu.

Podrobnější schéma násobičky je na obrázku 1.18, kde jsou naznačeny některé možné vstupy násobičky (*AD* je výsledek z předčítačky), které mají datovou šířku 18 a 27 b. Výsledný součin je 90b a po volitelném proregistrování (průchod dat přes registr) je rozdělen na dva dílčí 45b výsledky. Pro dosažení součinů větších datových šířek se násobičky řetězí za pomoci 17b kaskádovacího výstupu s posunem vpravo. Posun vpravo zarovná dílčí výsledky násobičky požadovaným počtem bitů. Tento kaskádovací výstup poté vstupuje do multiplexoru *Z*, který je připojen ke sčítačce/odčítačce v sousedním DSP bloku.



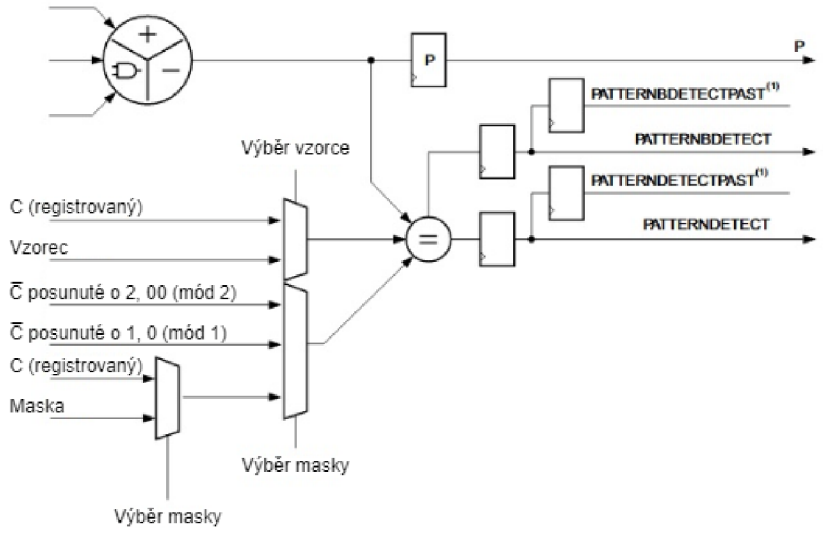
Obr. 1.18: Podrobnější pohled na násobičku 18 × 27 v DSP48E2 [40]

Další vestavěnou funkcí je prvek nastavitelný jako sčítačka, odčítačka nebo logická jednotka. Jeho funkce se vybírá signálem *ALUMODE* a vstupní data pro se volí pomocí řídících signálů *OPMODE* a *CARRYINSEL*. Výstup tohoto prvku je tedy funkcí signálů *ALUMODE*, *OPMODE* a *CARRYINSEL*. Při nastavení jako logická jednotka musí být násobička vyřazena z obvodu.

Mód „SIMD“ umožňuje prvek sčítačka/odčítačka/akumulátor rozdělit na dva nebo čtyři stejné prvky vykonávající tu stejnou operaci a podle toho se dělí i datové šířky vstupů a výstupů. V případě,

že jsou místo jedné 48b sčítačky požadovány dvě 24b, sčítačka se rozdělí na dvě samostatné tím způsobem, že se zamezí přenosu přetečení z bitu 23 na 24 (tedy v půlce). Spodní bity [23..0] dvou vstupních signálů budou sčítány v jedné sčítačce a výsledek bude na pozici [23..0] výstupního signálu. Horní bity [47..24] vstupních signálů budou sečítány v druhé sčítačce a výsledek bude na pozici [47..24] výstupního signálu. Obdobně tomu bude při požadavku na čtyři 12b sčítačky. Přetečení se zamezí mezi bity 11 a 12, 23 a 24 a 35 a 36. Jednotlivé výsledky budou ve výstupním signálu v rozmezí po 12 b. Každá dílčí část, v tomhle příkladě sčítačky, má svůj výstup pro přetečení ($CARRYOUT$), jak již bylo zmíněno výše.

Funkcí logiky pro rozpoznávání vzorců je srovnávat vstupní data. Vstupní data pro srovnání jsou brána z výstupu P (před výstupním registrem), druhým vstupem může být přednastavený signál $Pattern$, dynamický vstup C, nebo maska, která může být daná přednastaveným signálem ($MASK$) nebo také dynamickým vstupem C, který může být invertován a posunut o jeden nebo dva bity. Schéma prvku pro rozpoznávání vzorců je na obrázku 1.19. Porovnání výstupu P s jinou hodnotou se provádí

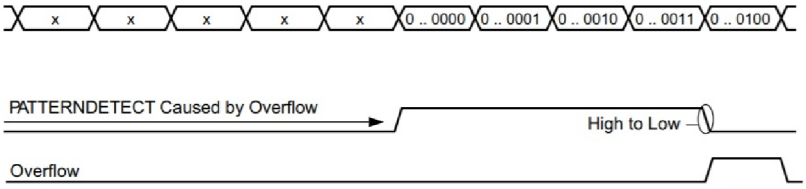


Obr. 1.19: Schéma porovnávacího prvku v DSP48E2 [40]

v tomtéž taktu, kdy je výstup P vypočítán, takže nedochází žádnému zpoždění. Nicméně je to logika navíc, což způsobí určité snížení rychlosti DSP. Kromě detekce vzorce na výstupu P tento prvek může srovnávat hodnoty dynamických vstupů $A{:}B$ a C nebo $A \times B$ a C, hlídat přetečení/podtečení/saturaci nad bitem P[46], vyhodnotit automatický reset aj. Pokud žádného takového srovnání není třeba, může se tento prvek využít například pro duplikaci některého z výstupních pinů nebo třeba invertování jednoho bitu.

Logika pro přetečení nebo podtečení se používá při akumulaci buď po násobení (MACC) nebo po sčítání (čítač). Využívá se přitom prvku rozpoznávacího vzorce, kde se vzorec ($Pattern$) nastaví ve většině případů na 0..0 (samé nuly) nebo 1..1 (samé jedničky) a maskou se řídí, v jakém místě se bude jednat o přetečení/podtečení. V datovém slově by měl být alespoň jeden ochranný („guard“) bit. Například když je požadována detekce přetečení prvních dvou bitů (tj. když výsledek přesáhne hodnotu

3), bude *Pattern* nastavena na 0..0 a maska na 0..11. Obrázek 1.20 odpovídá uvedenému příkladu. Spodní dva bity jsou kvůli masce vynechány z porovnání a srovnávají se P[47..2] a *Pattern*[47..2]. Až



Obr. 1.20: Časový průběh signalizace o přetečení [40]

výsledek P přesáhne číslo 3, bude tedy 0..100, výsledek P[47..2] a *Pattern*[47..2] se nebudou rovnat. V tu chvíli se shodí signál *PATTERNDETECT* z logické 1 do 0, na což okamžitě (ve stejném taktu) zareaguje signál *Overflow* tím, že se přepne do logické 1.

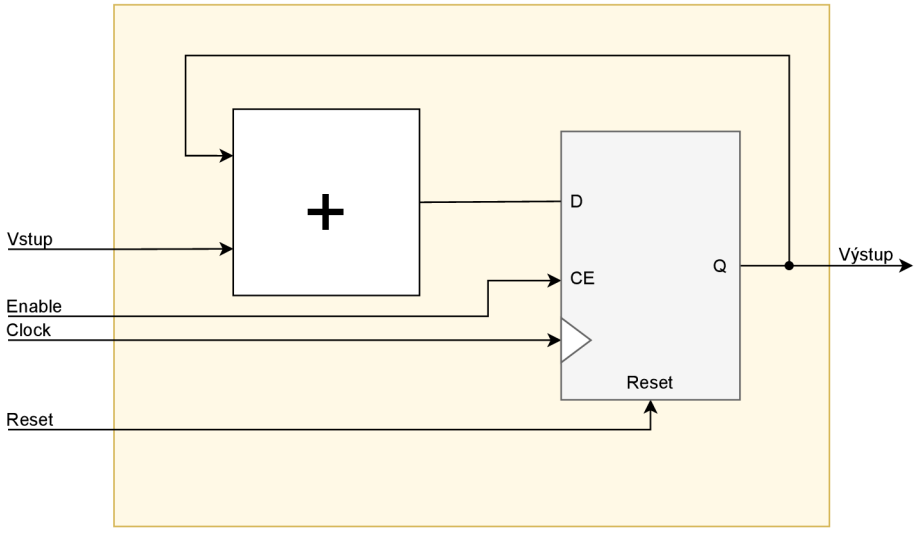
Novou funkcí DSP48E2 je až 96 b široká logická funkce XOR. Používají se výstupy z multiplexorů X, Y a Z, multiplexor W má na výstupu trvale samé nuly. Z toho vyplývá, že funkce XOR může mít až 3 vstupy. V tom případě je na výstup multiplexoru Y vybrán signál C, jinak když je požadován pouze dvouvstupý XOR, je na výstup multiplexoru Y vybrán signál s nulami. Tato funkce podporuje mód „SIMD“, takže může probíhat více XOR operací zároveň. To zlepšuje výkonnostní vlastnosti při opravě přijatých rámců neboli FEC (Forward Error Correction), nebo při výpočtu kontrolního součtu CRC (Cyclic Redundancy Check), viz [43].

2 Realizace DSP čítače

Tato kapitola je věnována způsobu zpracování úkolu využití DSP bloků na FPGA Stratix 10 na implementaci čítače. Nejprve byl v syntézním nástroji Intel Quartus® Prime nastaven DSP blok (také nazýván jako atom) tak, aby fungoval jako čítač. Nastavený DSP blok byl poté testován, aby se ověřila funkce čítání. Následně se vzal DSP blok jako primitivum a instancoval se v behaviorálním popisu, kde byl nastaven stejně jako předtím v nástroji Quartus. Nad touto instancí byly postaveny dvě obálky. V první obálce je kromě zapojení instancovaného DSP vytvořena druhá varianta čítače, ve které je pro implementaci použita volně dostupná logika na FPGA místo DSP bloků. Navíc je zde možné detekovat přetečení čítače a zastavit jej na své maximální hodnotě. Druhá obálka zastřešuje implementace DSP čítače pro FPGA UltraScale+ a Virtex 7 od firmy Xilinx a pro Stratix 10 od firmy Intel, a tvoří tak společné rozhraní. Obě zmíněné obálky byly samostatně testovány, což je popsáno v předposlední části této kapitoly. Nakonec byl čítač zapojen do vybraných komponent, které pak byly také otestovány, a byla na nich provedena měření, která měla zjistit, zda-li a jak moc je využití čítačů implementovaných pomocí DSP bloků výhodné.

2.1 Návrh

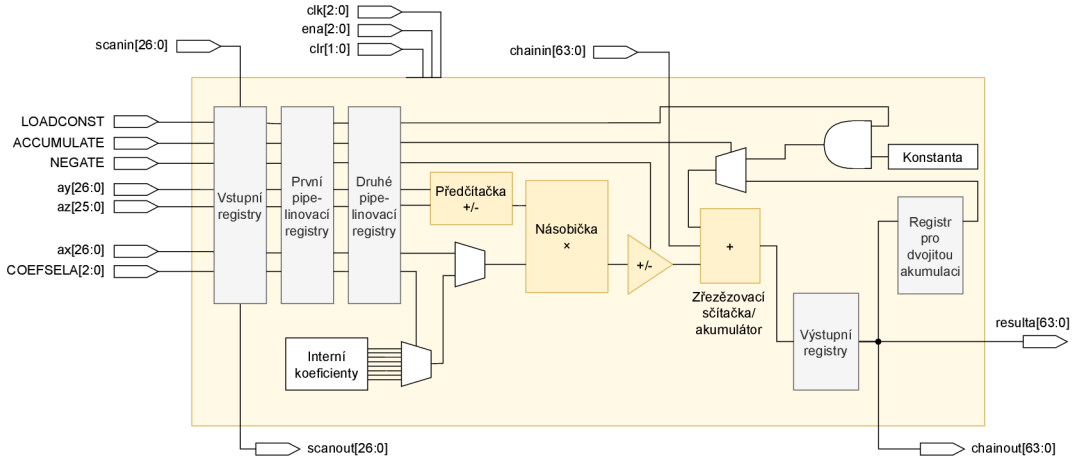
Čítač je prvek, který má standardně jeden datový vstup i výstup, dále vstup pro hodinový signál (*Clock*) a zpravidla také vstup pro resetování (*Reset*, někdy zvaný *Clear*) a povolení čítaní (*Enable*). Takový čítač je schématicky znázorněn na obrázku 2.1. Skládá se ze dvou jednotek, první jednotka sčítá dva signály na jejím vstupu a druhá jednotka si součet zapamatuje, tj. registr. Do sčítací jednotky vstupuje jednak již zmíněný datový signál ze vstupu čítače a jednak výstup z registru. V DSP bloku bylo tedy potřeba vyhledat tyto dvě jednotky a správně je propojit.



Obr. 2.1: Detailní schématické znázornění čítače

2.1.1 Volba nastavení DSP bloku

V syntézním nástroji Quartus v katalogu IP bloků je možné vybrat si některý z DSP bloků a ten si podle vlastních potřeb nastavit. V tomto případě se jednalo se o fixed-point variantu, protože čítat se vždy budou čísla s pevnou desetinnou čárkou. Z části 1.5.1 bylo jasné, že pro co nejširší čítač (zejména výstup čítače) je potřeba zvolit nastavení 27 × 27. Funkční schéma DSP bloku v nastavení 27 × 27 je na obrázku 2.2.



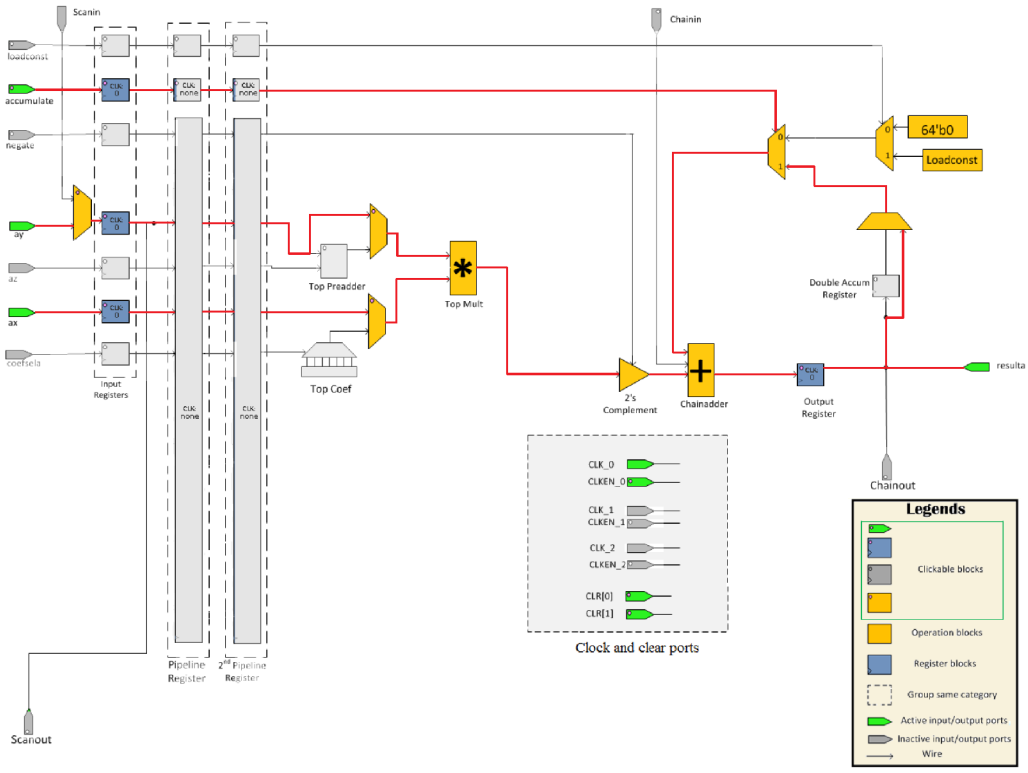
Obr. 2.2: Funkční schéma DSP bloku na na Stratix 10 v nastavení 27 × 27 [42]

Sčítací jednotku čítače tvoří prvek zřetězovací sčítačka/akumulátor, který bude dále nazýván jen jako akumulátor, protože zřetězovací vstup nebude používán. Jeho výstup je přímo připojen na výstupní registr, který je zapamatovávací jednotkou čítače. Nastavení DSP bloku tedy vypadalo tak, jak je ukázáno na obrázku 2.3.

V prvním kroku byly nastaveny vstupy do akumulátoru. Aby se výstup z výstupního registru stal také jedním ze vstupů akumulátoru, bylo potřeba nastavit řídící signál *ACCUMULATE* do logické jedničky (na obrázku 2.3 svítí zeleně, stejně jako ostatní aktivované vstupy či výstupy). Druhým vstupem akumulátoru je výstup prvku, který podle nastavení řídícího signálu *NEGATE* vytváří dvojkový doplněk signálu z násobičky. V základní verzi čítače je požadavek pouze na čítání nahoru (hodnota čítače a nová vstupní hodnota se sčítají), takže *NEGATE* je ponechán v logické nule. Před prvkem vytvářející dvojkový doplněk se nachází násobička, kterou v DSP na Stratix 10 nelze vypustit z obvodu, jak tomu je v případě DSP na UltraScale+. Ten hlavní (datový) ze vstupů násobičky bude obsahovat data, která poputují až do akumulátoru, a vedlejší vstup násobičky musí být trvale nastaven na hodnotu jedna, aby nedošlo ke změně čítaných dat. Vstupy *ay* a *ax* jsou oba široké 27 b a je tedy jedno, který bude hlavním vstupem a který vedlejším. Jako datový vstup čítače byl vybrán vstup *ay* a vstup *ax* tedy bude trvale nastaven na hodnotu jedna. Tento signál po průchodu vstupními registry vstupuje do předčítačky, kterou také není možné obejít, proto je třeba nastavit druhý vstup předčítačky *az* na nulovou hodnotu, respektive deaktivovat jej (opět z důvodu, aby nedošlo ke změně čítaných dat).

Druhou variantou by bylo dát na druhý vstup násobičky vnitřní koeficient, vybraný řídícím signálem *COEFSELA*, a vstup *ax* by se pak nenastavoval. Vybraný koeficient by samozřejmě také musel mít

hodnotu jedna. Toto řešení nebylo použito, protože to znamenalo trošku více nastavování (nastavení vnitřního koeficientu a poté signálu *COEFSELA*).



Obr. 2.3: DSP blok v konfiguraci 27 × 27 nastavený jako čítač

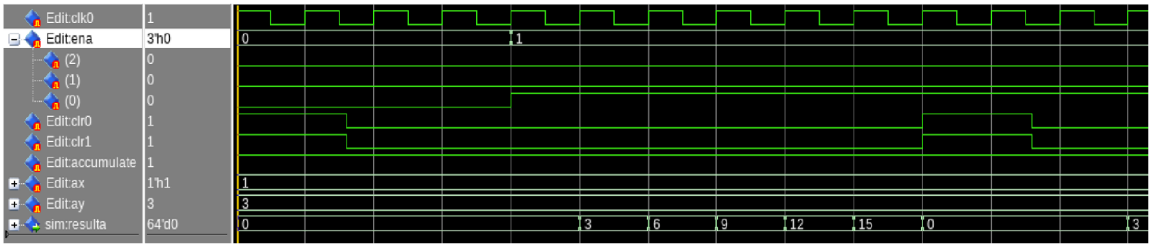
Vstupní i výstupní registry jsou řízené hodinovým signálem *CLK_0* a povolují se signálem *CLKEN_0*. Signál *CLR* maže data uložená v registrech. Jedná se o synchronní mazání, což znamená, že hodnota signálu *CLR* je brána v potaz až při vzestupné hraně hodinového signálu. *CLR*[0] maže vstupní registry, *CLR*[1] maže výstupní a pipelinovací registry.

Ostatní signály jako *LOADCONST* a *COEFSELA* v tomto případě nebyly podstatné, a tudíž byly ponechány ve výchozím stavu jako neaktivní.

2.1.2 Simulace DSP bloku

Po nastavení DSP bloku v nástroji Quartus byla ověřena jeho funkčnost, zda-li opravdu funguje jako čítač. Quartus nabízí možnost si vygenerovat VHDL (nebo Verilog) kód takto nastaveného DSP bloku a k tomu i simulační soubory. Pomocí těchto souborů lze v nástroji ModelSim spustit simulaci, kde je možné nastavit vstupní signály a poté ve výstupním časovém diagramu pozorovat hodnoty na výstupu. Výsledný časový diagram takové simulace je na obrázku 2.4.

Signál *clk0* byl nastaven jako hodinový signál s periodou 5 ns. Signál *ena*, který povoluje nebo zakazuje zápis do všech registrů, je 3b vektor. Jak již bylo zmíněno, v DSP lze využít až tři různé hodinové signály a pro každý existuje jeden povolovací *ena* signál. Protože v tomto nastavení DSP je



Obr. 2.4: Simulace DSP bloku nastaveného jako čítač

použit pouze hodinový signál *clk0*, je zde důležitý pouze *ena*[0]. Mazací (resetovací) signály *clr0* a *clr1*, byly nastaveny identicky, protože se předpokládá, že vstupní i výstupní registry budou resetovány současně a oba *clr* signály budou připojeny na tentýž resetovací signál. Jsou nastaveny do logické jedničky hned od začátku kvůli zbavení registrů od předchozích hodnot. Dále signál *accumulate* byl po celou dobu držen v logické jedničce (aby docházelo k čítání) stejně jako signál *ax*, datový signál *ay* byl pak nastaven na hodnotu 3, protože i když bylo možné jej nastavovat náhodnými čísly, generovaly se v tomto případě velká čísla, což mělo za následek náročné ověření správnosti výsledku, a navíc konstantní hodnota 3 na vstupu čítače pro ověření jeho funkčnosti plně dostačuje.

Výstupním signálem čítače je signál *resulta* (v podkapitole 1.5.1 označován jako *vysledek a*). Od začátku je jeho hodnota nulová, protože signál *clr1* resetující výstupní registr je aktivní. Poté, co spadne, zůstává signál *resulta* na nulové hodnotě kvůli tomu, že registry ještě další dva takty nemají povolený zápis – signál *ena*[0] je v logické nule. V okamžiku, kdy při vzestupné hraně hodinového signálu se na *ena*[0] dostane logická jednička, se vstupní data (s hodnotou 3) dostanou na výstup vstupních registrů (na obrázku 2.3 přední registry). Data projdou kombinační logikou, ve které se stávající hodnota výstupního registru (v tomto okamžiku 0) sečte s příchozími daty (s hodnotou 3), a dostanou se až na vstup výstupního registru. Odtud se s další vzestupnou hranou hodinového signálu dostanou na jeho výstup, což je výstup celého obvodu. V dalším taktu se na výstup dostane součet předchozí výstupní hodnoty, tedy 3, se vstupními daty (pořád 3), tudíž 6. Stejným způsobem se na výstup dostávaly další hodnoty, vždy o číslo 3, které je konstantně na vstupu čítače, větší než předchozí. Tím byla ověřena funkce nastaveného DSP jakožto čítače. Navíc byla otestováno správná reakce DSP na *clr* signály. Tyto signály jsou k registrům připojeny napřímo (nejsou vedeny přes jiné registry), proto je ve výstupním signálu *resulta* hodnota 0 od toho okamžiku, kdy se při vzestupné hraně hodinového signálu dostane na signály *clr* logické jednička. Skutečnost, že výstupní signál *resulta* má hodnota 0 od okamžiku, kdy jsou *clr* signály v logické jedničce zároveň se vzestupnou hranou hodinového signálu, značí, že se jedná o synchronní reset, což bylo v DSP bloku také nastaveno, ale na obrázku 2.3 to nelze vidět. Po návratu *clr* signálů do logické nuly opět dva takty trvá, než vstupní data ovlivní výstup čítače.

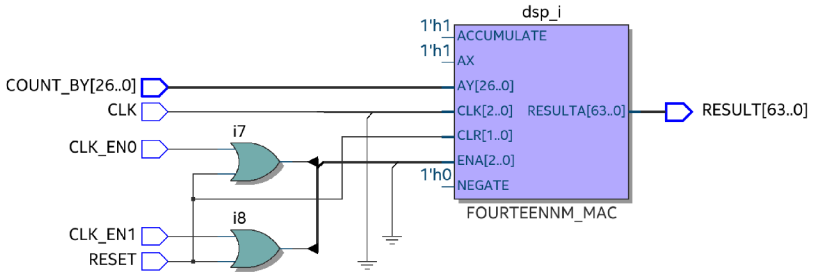
2.2 Implementace

V předchozí podkapitole bylo navrženo a otestováno nastavení DSP bloku jako čítače. V této podkapitole je DSP blok zapojen do behaviorálního popisu, kde je možné genericky nastavit některé parametry

a připojit vnější signály. Nad touto komponentou byla vytvořena obálka, kde by v případě, že použité FPGA nepodporuje tyto DSP bloky (resp. nadřazenou knihovnu), bylo možné použít pro realizaci čítače obecné zdroje nacházející se v daném FPGA. Zároveň je zde čítači přidána funkce zvaná „auto reset“, která je schopná detekovat přetečení a zastavit čítač na své maximální hodnotě. Nad touto obálkou byla vytvořena ještě jedna, která utváří společné rozhraní jednak pro tento DSP čítač a jednak pro DSP čítač na UltraScale+, který byl návrháři ze sdružení CESNET již dříve vytvořen a nasazen do firmwaru síťové karty COMBO-200g2ql.

2.2.1 Zapojení DSP bloku

DSP blok jako primitivum byl vložen do komponenty DSP_COUNTER_STRATIX_10_ATOM pomocí knihovny „fourteennm“, ve které je pod názvem „fourteennm_mac“. Tato nadřazená komponenta zjednodušuje rozhraní a umožňuje uživateli jednodušší manipulaci s čítačem. Byly na ni namapovány generické hodnoty a vstupní a výstupní porty. Schéma zapojení DSP bloku do této komponenty je na obrázku 2.5.



Obr. 2.5: DSP blok zapojený v komponentě DSP_COUNTER_STRATIX_10_ATOM

Nastavení, která definovala funkci čítače, byla DSP bloku nastavena napevno a nejsou na obrázku 2.5 vidět. Jednalo se o nastavení módu na 27 × 27, typu mazacího signálu na synchronní a o změnu typu hodinového signálu pro výstupní registry na *clk1* z výchozího *clk0*.

Signál pro akumulaci byl trvale nastaven na hodnotu jedna, stejně jako signál *ax*, což je druhý vstup násobičky uvnitř DSP. Další dva signály jsou připojeny k odpovídajícím vstupům komponenty DSP_COUNTER_STRATIX_10_ATOM. Datový vstup čítače *ay* je připojen na vstup *COUNT_BY* („čítej po“), hodinové signály *clk0* a *clk1* jsou připojeny na *CLK*. Tyto hodinové signály mají stejný zdroj, takže by bylo možné pro celé DSP použít jen jeden z nich (např. *clk0*). Důvodem, proč to takhle nebylo provedeno, je umožnit uživateli povolování zápisu do vstupních a výstupních registrů nezávisle na sobě. Do registrů, které běží na určitém hodinovém signálu, například *clk0*, je možné povolit zápis jen a pouze odpovídajícím povolovacím signálem, tj. *ena0*. Aby bylo možné výstupním registrům povolovat zápis jiným povolovacím signálem než vstupním registrům, musí běžet i na rozdílných hodinových signálech, které mohou mít, a v tomto případě také mají, stejný zdroj. Signály *clk2* a *ena2* byly napevno nastaveny na hodnotu 0, protože nejsou požívány (na obrázku 2.5 jsou přivedeny na zem). Povolovací signály *ena0* a *ena1* jsou výstupem z logických členů OR, jejichž vstupy jsou jednak odpovídající vstupní povolovací signály *CLK_EN0* a *CLK_EN1* a jednak mazací signál *RESET*. Důvod k tomu

je ten, že registry v DSP bloku prioritně reagují na povolovací signál, tedy když je čítač zastaven, nereaguje už na povel k mazání. To znamená, že v případě, kdy bude čítač pozastaven (povolovací signály budou v logické nule) a mezitím přijde povel k mazání (mazací signál v logické jedničce), který bude trvat kratší dobu, než skončí pozastavení čítače, na výstupu čítače bude stále stejná hodnota, nedojde k jeho vynulování. Povolovací signály *ena0* a *ena1* jsou tedy pomocí členu OR upraveny tak, aby byly v logické jedničce pokaždé, když přijde povel k mazání. Mazací signály *clr0* i *clr1* jsou připojeny k jednomu stejnému vstupu *RESET*, protože vstupní i výstupní registry budou vždy resetovány současně.

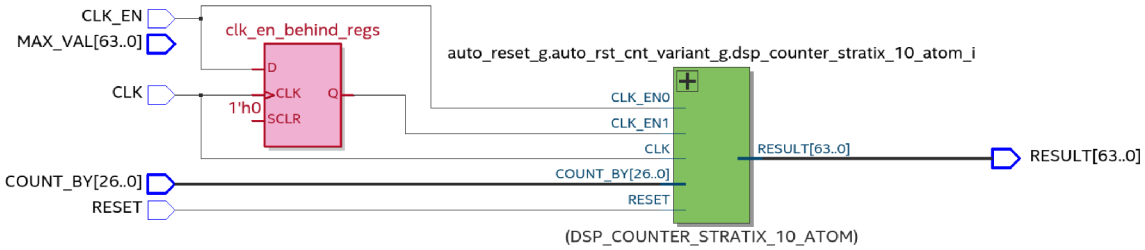
Nakonec vstupní signál *NEGATE*, kterým se nastavuje, zda-li akumulovací prvek bude vstupní data sčítat nebo odčítat od aktuálního výsledku, je možné nastavovat v DSP dynamicky, jeho hodnota (a funkce celého čítače) se tedy může za běhu měnit. Nicméně v této aplikaci se nepočítá s tím, že bude za běhu potřeba změna z čítání nahoru na čítání dolů, proto v komponentě DSP_COUNTER_STRATIX_10_ATOM je zmiňovaný port nastaven generickou hodnotou, jeho funkce bude tedy jasně stanovená už při vytváření obvodu v FPGA. Když bude *NEGATE* v logické nule, čítač bude čítat nahoru, když bude v logické jedničce, bude čítat dolů a to tím způsobem, že udělá dvojkový doplněk vstupních dat (což vyústí ve změnu znaménka), která se následně sečtou s aktuální hodnotou čítače, tedy ve skutečnosti se odečtou. Protože není možné nastavit výchozí hodnotu čítače, po resetu se začíná odečítat vždy od hodnoty 0, tudíž čítač hned s první nenulovou hodnotou podteče. Na obrázku 2.5 je nastaven na výchozí hodnotu 0, čítá tedy nahoru. Dalšími generickými nastaveními DSP bloku uživatel ovlivňuje vlastnosti čítače, jako například šířku vstupu (maximálně 27), šířku výstupu (maximálně 64) a aktivaci či deaktivaci vstupních registrů, které případně poběží na hodinovém signálu *clk0*.

2.2.2 DSP čítač pro Stratix 10

Tato obálka DSP čítače vznikla zejména z toho důvodu, aby jej bylo možné použít i v jiných FPGA, kde tento druh DSP bloků (tj. „fourteennm_mac“) není. Místo DSP bloků totiž umožňuje vytvořit čítač pomocí obecné logiky FPGA. To je výhodné i v případě, že v daném FPGA není dostatek DSP bloků na vytvoření čítačů využívající DSP bloky. Pokud by existoval nějaký návrh obvodu, jehož součástí by byl i tento čítač, a daný obvod by bylo potřeba použít i pro jiné FPGA (nebo by nedostačoval počet DSP bloků), pouhou změnou jedné generické hodnoty by se dalo docílit toho, že čítač bude implementován v obecné logice. Kdyby toto možné nebylo, všechny DSP čítače by se musely ručně nahradit jinými, buď těmi využívající DSP bloky daného FPGA (pokud dané FPGA DSP bloky obsahuje) nebo obecnou logiku.

Provedení čítače se tedy nastavuje genericky. Pokud je nastaveno provedení pomocí DSP, je přeportována komponenta DSP_COUNTER_STRATIX_10_ATOM. Schéma takto nastavené první obálky je na obrázku 2.6.

Dalším generikem lze nastavit čítač tak, aby se v okamžiku, kdyby přicházející hodnota způsobila přetečení čítače (jedná-li se o čítání nahoru), automaticky resetoval, nebo aby se zastavil a na výstupu pak byla maximální hodnota. Maximální hodnotou nyní může být pouze vektor bitů o šířce výstupu plný logických jedniček. Detekce přetečení nebo podtečení je provedena pomocí nejvýznamnějšího bitu (MSB – Most Significant Bit) výstupního signálu (*RESULT*). Tento bit se každý takt uloží do registru a v následujícím taktu se provede jeho porovnání. K přetečení dojde vždy, když v minulém taktu měl MSB hodnotu 1 a nyní má hodnotu 0. Tohle by nefungovalo, kdyby bylo možné nastavit maximální hodnotu libovolně. Podtečení se detekuje obráceně, tedy když v minulém taktu měl MSB hodnotu



Obr. 2.6: První obálka čítače

0 a nyní má hodnotu 1. Zde je ale navíc potřeba vynechat první podtečení, ke kterému dojde hned s první nenulovou odečítanou hodnotou, protože se čítá od 0. Ke špatné detekci přetečení by navíc mohlo dojít, když by v jednu chvíli byl výsledek 01..11 a přičetla by se hodnota větší než 10..00 (kde by výsledek i vstupní hodnota měla stejnou bitovou šířku). Toto by mělo z následek okamžité přetečení, a jelikož předchozí hodnota MSB výsledku nebyla 1 ale 0, k detekci přetečení by nedošlo. V podobném případě by k této chybě mohlo dojít také při čítání dolů. Tento problém ale není potřeba řešit, protože k tomu nikdy nemůže dojít. Maximální šířka vstupních dat je totiž 27 b (limit DSP bloku) a to na takové přetečení/podtečení nestačí. Při využití maximální šířky čítače 64 b je při čítání nahoru maximální hodnota $2^{64}-1$, což je zhruba $18,4\times10^{18}$. Je to tedy obrovské číslo, kterého se jen tak nedá dosáhnout. Když se bude jednat o čítač čítající dolů, půjde o detekci podtečení čítače a maximální hodnota musí být 0 (název „maximální hodnota“ zůstává, přestože se zde jedná o minimální hodnotu). Výchozím chováním čítače je klasické přetečení/podtečení.

Maximální hodnota čítače se nastavuje hodnotou vstupního signálu *MAX_VAL*. Důvodem, proč je *MAX_VAL*[63..0] deklarován jako signál a ne jako generická hodnota, je opět čítač na Ultrascale+. Zde tento signál vstupuje do DSP (pokud je čítač realizován pomocí DSP) přes vstup *C* (viz obrázek 1.15) do prvku rozpoznávajícího vzorce (*Pattern Detector*, viz obrázek 1.19). Je to tedy kvůli vzájemné kompatibilitě, jinak se samozřejmě nepočítá s tím, že by bylo potřeba měnit maximální hodnotu čítače za provozu.

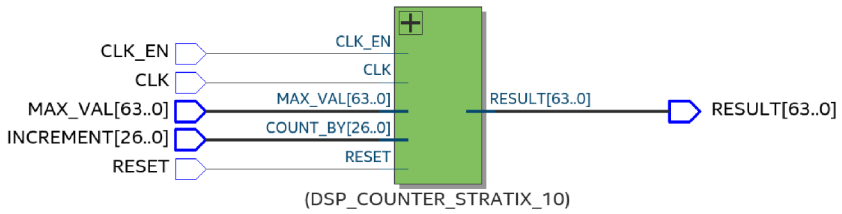
Rozhraní generik se tedy rozrostlo o možnost automatického resetu a možnost použití DSP bloků pro realizaci čítače. Možnost čítání dolů nebo nahoru byla uživatelsky zjednodušena, zde se nastavuje booleovskou hodnotou, tedy *true* nebo *false*, která se také vytvořenou funkcí převádí na požadovaný formát (logická 0 nebo 1).

Většina signálů byla napřímo propojena, jmenovitě se jednalo o signály *CLK*, *COUNT_BY*, *RESET* a *RESULT*. Jedinou změnou prošel povolovací signál CLK_EN, který je na obrázku 2.6 veden přes registr. Zde se jedná o variantu čítače s aktivovanými vstupními registry. Tento vstupní registr o jeden takt zpozdí signál povolující zápis do výstupního registru čítače. Ve chvíli, kdy přijde zákaz zápisu registrů (tj. *CLK_EN* bude v logické nule), se tato zpráva dostane do výstupních registrů o takt později, takže dojde ještě k jednomu sečtení a na výstupu tak bude aktuálnější hodnota. V případě, kde by vstupní registry aktivovány nebyly, se *CLK_EN* připojí přímo na *CLK_EN1*. Stojí za zmínku, že v této obálce existuje jen jeden povolovací signál, kterým se povoluje zápis do vstupních i výstupních

registrů současně. Pokud uživatel vyžaduje separátní řízení zápisu vstupních a výstupních registrů, musí si zapojit nižší komponentu DSP_COUNTER_STRATIX_10_ATOM zmiňovanou dříve. Jde opět o snahu sjednotit rozhraní tohoto čítače s čítačem na Ultrascale+.

2.2.3 Obecný DSP čítač

V druhé obálce čítače je obecné rozhraní sjednocující čítač na UltraScale+[1] a čítač na Stratix 10, resp. jeho první obálku, která je popsaná výše. Zapojení, kde zadaným typem FPGA je Stratix 10, se nachází na obrázku 2.7.



Obr. 2.7: Druhá obálka čítače se sjednoceným rozhraním

Protože už první obálka byla navrhována s úmyslem sjednocení, nebylo zde potřeba výrazných úprav. Přibyla možnost genericky nastavit název FPGA pro implementaci čítače, kde je na výběr Stratix 10 od firmy Intel a UltraScale+ a 7 Series od firmy Xilinx. Jednou další výraznější změnou je odebrání možnosti pro odčítání a vypnutí automatického resetování čítače. Během průběžných simulací a inspirace autora této práce čítačem na Xilinx FPGA bylo zjištěno, že tyto funkce zde nefungují tak, jak bylo očekáváno. Z tohoto důvodu čítač trvale čítá nahoru a automaticky se resetuje (přeteká). Ostatní generické položky jsou stejné a veškeré signály byly přeportovány napřímo, jen signál *COUNT_BY* byl v této obálce přejmenován na *INCREMENT* („navýšení“), viz obrázek 2.7.

2.3 Simulace a testování

Správný přístupem k testování nejen digitálních obvodů je, že testy jsou prováděny jiným člověkem, něž je návrhář daného obvodu. Nicméně zadáním této práce je ověřit funkčnost implementovaného čítače, a proto autor této práce je i autorem simulací dále popsaných níže.

Soubor, jenž definuje, kdy a který signál bude mít jakou hodnotu, se nazývá „testbench“. Základní testbench obsahuje zapojení zkoušeného obvodu, nastavení jeho generických hodnot a generování signálů, které se připojí na jeho vstupy. Pro generování signálů se používají procesy. Například pro generování hodinového signálu se použije proces, kde se určí, jak dlouho bude mít hodinový signál hodnotu jedna a jak dlouho bude mít hodnotu nula. Většinou se předtím vytvoří konstanta s názvem *perioda*, takže hodinový signál bude mít hodnotu jedna po dobu *perioda*/2, stejně tak pro hodnotu

[1] Tento čítač lze použít i pro FPGA 7 Series od výrobce Xilinx, protože DSP bloky na těchto dvou FPGA jsou vzájemně kompatibilní.

nula. Ostatní signály se generují obdobně: nastaví se na určitou hodnotu a nějakým způsobem (možností je mnoho) se stanoví čas, po který budou takhle nastaveny. Toto nastavení se zopakuje několikrát po sobě s různými hodnotami, aby se testovaný obvod co nejlépe prozkoušel.

Pokročilejší soubory testbench využívají náhodných čísel pro nastavování datových i řídicích signálů. Komplexní soubory testbench pak mohou obsahovat i popis celého obvodu podle požadovaného chování, do kterého se také přivedou generované signály, a kontroluje se shoda testovaného obvodu a obvodu popsaného v souboru testbench.

Pro tuto práci byly vytvořeny dva takové soubory, v principu dost podobné, jeden pro první obálku čítače, tj. komponentu DSP_COUNTER_STRATIX_10, a druhý pro druhou obálku, tj. komponentu DSP_COUNTER. V obou těchto souborech byly signály vytvářeny pomocí náhodných čísel. Oba také obsahují instanci čítače pro srovnání výstupů.

V souboru testbench pro testování první obálky čítače byl vytvořen hodinový signál již dříve popsaným způsobem. Datový signál *cnt_by* nesoucí hodnotu, o kterou se bude hodnota čítače navyšovat nebo snižovat, byl generován náhodnými čísly vytvářenými interní funkcí sdružení CESNET. Náhodná čísla byla generována v rozmezí od nuly až po maximální možnou hodnotu signálu, která je daná jeho šířkou. V případě povolovacího signálu *clk_ena* a mazacího signálu *rst* se náhodným číslem neovlivňuje jejich hodnota nýbrž čas, po který budou signály dané hodnoty (0 nebo 1) nabývat. V procesu se signály nastavily na nějakou hodnotu a proces byl pozastaven na dobu, která byla dána součinem periody hodinového signálu a náhodně vygenerovaného čísla. Signál *clk_ena* byl měněn častěji než signál *rst*, protože více než nulování čítače je zajímavé pozorovat jeho přetékání/podtékání, k čemuž by při častém nulování nedocházelo.

V tomto souboru testbench byl také vytvořen prvek, který má simulovat funkci čítače. Jeho výsledek se za běhu simulace porovnává s výsledkem komponenty DSP_COUNTER_STRATIX_10. Porovnání výsledků slouží k rychlému zhodnocení funkčnosti čítače. Vytvoření porovnávacího signálu je zobrazeno ve výpisu kódu níže.

```
result_ok <= '1' when sim_result = cnt_result else '0';
```

Význam tohoto úryvku kódu je následující. Pokud se výsledek instance čítače v souboru testbench *sim_result* a výsledek testovaného čítače *cnt_result* rovnají, porovnávací signál *result_ok* bude mít hodnotu jedna, jinak nula. Při prohlížení časového diagramu, který je výstupem dané simulace, pak není nutné zkoumat všechny signály a určovat, zda-li výstup je v souladu se vstupními daty. Stačí zkontrolovat, že srovnávací signál je vždy v logické jedničce.

Tu ovšem nastává riziko, že testovaný obvod a jeho instance v souboru testbench budou obsahovat tu samou chybu, srovnávací signál tedy bude ukazovat shodu, ale obvod nebude plnit očekávanou funkci. Pravděpodobnost takového případu je o to větší, když testovací soubory vytváří a samotné testování provádí tentýž člověk, který obvod navrhl, což, jak bylo již zmíněno, je právě tento případ. Riziko této chyby se dá zmírnit tím, že i když srovnávací signál po celou dobu ukazuje shodu, tak jsou podrobně zkoumány ostatní signály (tedy postup, jako kdyby tam srovnávací signál nebyl).

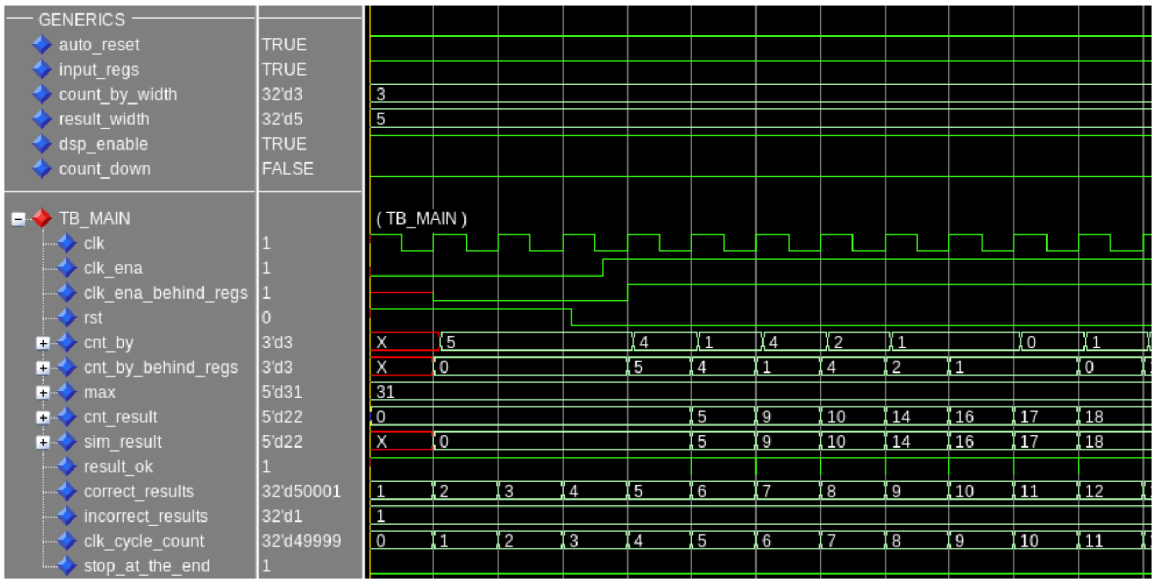
Pomocí srovnávacího signálu lze i mimo jiné jednoduše vypsat shody či odlišnosti do výpisového okna. Takový výpis může vypadat jako na obrázku 2.8, který se vztahuje k úspěšnému testu komponenty DSP_COUNTER_STRATIX_10. Tento test trval 50 tisíc taktů, při čemž byly na jeho vstupy vysílány signály s náhodně generovanými hodnotami, jak bylo popsáno výše. Simulace však může probíhat mnohem déle než 50 tisíc taktů. Pro shrnutí úspěšnosti proběhlé simulace byl vytvořen proces,

který na základě počtu chyb (případů, kdy *result_ok* nebyl v logické jedničce) určí, jestli simulace proběhla v pořádku (vypíše: „Success.“) nebo ne (vypíše: „There were some incorrect results.“).

```
# Results match. Iteration: 5000
# Results match. Iteration: 10000
# Results match. Iteration: 15000
# Results match. Iteration: 20000
# Results match. Iteration: 25000
# Results match. Iteration: 30000
# Results match. Iteration: 35000
# Results match. Iteration: 40000
# Results match. Iteration: 45000
# Results match. Iteration: 50000
#  Success.
```

Obr. 2.8: Výpis při testování první obálky čítače

Část výstupního časového diagramu z této simulace je na obrázku 2.9. Jak je vidět, signál *result_ok* je po celou dobu v logické jedničce, stejně je tomu i mimo úsek ukázaný na daném obrázku. K chybě tedy mohlo dojít pouze v případě, pokud by byla stejná chyba testovaném čítači i v simulaci. Pro zjištění takové chyby je potřeba prozkoumat všechny ostatní signály a stanovit, zda-li se chovají podle očekávání, nebo nikoliv.



Obr. 2.9: Výstupní časový diagram po simulaci první obálky čítače s použitými DSP bloky

Generické hodnoty jsou nastaveny tak, že *auto_reset* je aktivovaný (čítač tedy přirozeně přetéká), dále jsou aktivní vstupní registry, šířka vstupních dat jsou 3 b a šířka výstupních dat jsou 4 b. Později byly otestovány i maximální šířky vstupů a výstupů, ale pro prvotní otestování a ověření, že signál *result_ok* podává správné údaje, to bohatě stačí. Maximální hodnota, do které kdy čítač při simulaci dočítal, bylo nějaké 50b číslo (při nastavení šířky výstupu na 64 b). Více nebylo možné otestovat kvůli

selhání nástroje ModelSim, který při takových pokusech zamrzl a bylo třeba jej restartovat. Poslední dvě generická nastavení říkají, že čítač byl realizován pomocí DSP bloků a že se čítalo nahoru.

Nejprve je důležité si všimnout, že při prvních třech vzestupných hranách hodinového signálu je obvod resetován, což bývá zvykem, aby se vymazaly všechny registry, paměti apod. Proto je při těchto taktech na výstupu čítače (*cnt_result*) hodnota 0, kromě prvního taktu, kdy na výstupu čítače je neznámá hodnota „X“. Signál *sim_result* má už i v prvním taktu nulovou hodnotu, což je správné chování – takový byl úmysl. Aby v tomto případě nebyla hlášena chyba (protože ačkoliv $0 \neq \mathrm{X}$, je to v souladu s očekávaným chováním), je první porovnání těchto signálů ignorováno.

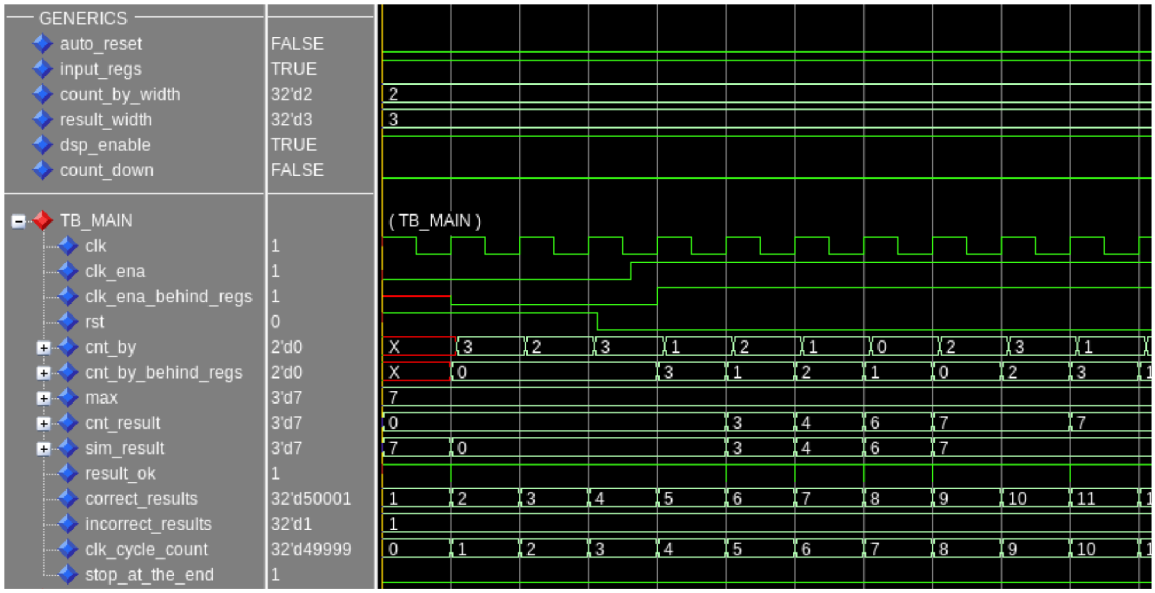
Ve stejném taktu, kdy *rst* přechází z 1 do 0, signál povolující zápis do registrů, tj. *clk_ena*, nabývá hodnoty 1 (čtvrtý takt). Hodnota datového signálu čítače *cnt_by* se tedy v dalším pátém taktu objeví na výstupu vstupních registrů, což ukazuje signál *cnt_by_behind_regs* (signál „čítej po“ za vstupními registry – pokud vstupní registry nejsou povoleny, je stejný jako *cnt_by*). V tomto pátém taktu se také dostane povolení zápisu v podobě *clk_ena_behind_regs* do výstupního registru, takže od této chvíle budou na výstupu čítače v následujících taktech data ze sčítačky.

Na obrázku 2.9 je vidět, že první data s hodnotou 5 jsou na vstupu čítače od druhého taktu, nicméně čítač je začne zpracovávat až ve čtvrtém taktu – až dostane povolení. V následujícím pátém taktu je hodnota 5 na výstupu vstupních registrů (tedy v signálu *cnt_by_behind_regs*), kde se okamžitě sečte s předešlým výstupem čítače (s hodnotou 0). Jelikož $5+0 = 5$, hodnotu 5 nese v šestém taktu jak signál *cnt_result*, tak i *sim_result*. Tyto signály jsou kromě prvního taktu shodné, a proto je signál *result_ok* v logické jedničce (v prvním taktu je na hodnotu jedna nastaven napevno). Shody těchto signálů, tedy počet taktů, kdy *result_ok* má hodnotu 1, počítá signál *correct_results* a po každé n-té shodě (v tomto případě $n = 5000$) je do výpisového okna vypsána věta oznamující shodu a dosavadní počet shodných výsledků, viz obrázek 2.8. Signál *incorrect_results* pak počítá neshody a je dále využíván pro finální verdikt o úspěšnosti celé simulace (výpis „Success.“). Signály *clk_cycle_count* a *stop_at_the_end* se používají ke stanovení konce simulace, kdy je verdikt vypsán.

Na obrázku 2.10 se nachází časový diagram, který je výsledkem simulace při vypnutí automatického resetování čítače. Rozdílů oproti předchozímu časovému diagramu je pár. Prvním je samozřejmě změna generické hodnoty *auto_reset* na *false*. Pak aby bylo možné vidět na tomto malém úseku časového diagramu zastavení čítače na maximální hodnotě v okamžiku jeho přetečení, byly změněny šířky vstupů a výstupů na 2, resp. 3 b. Protože byla změněna šířka vstupu na 2 b, hodnoty *cnt_by* se pohybují od 0 do 3. Maximální hodnota udávaná signálem *max* je v tomto případě 7, protože 3b číslo plné logických jedniček dá v desítkové soustavě hodnotu 7. Jinak čítač opět začne čítat až po čtvrtém taktu, kdy signál *rst* spadne na 0 a *clk_ena* naopak nabude hodnoty 1. Nejdůležitějším rozdílem je, že když čítač napočítá hodnoty 7 (zde to náhodou vyšlo přesně na maximální hodnotu), jeden takt ještě přičte další číslo, protože tou je 0 a nedetekuje se tak přetečení, a v následujícím taktu už je výstupu vnitřní logikou přiřazena hodnota 7 a zároveň je čítači zakázán další zápis trvající až do té doby, kdy přijde povel k mazání čítače (což už v časovém diagramu nelze vidět).

Testbench pro simulaci komponenty DSP_COUNTER, stejně jako výsledek simulace, vypadal velmi podobně. Simulace obou zmíněných obálek byly navíc spouštěny na delší dobu a hlavně pro všechny různé varianty nastavení generických hodnot.

Obě tyto simulace byly navíc přidány do „sbírky“ automatických testů, které se spouští denně. Jedná se o skript, který spouští vytvořené simulace/verifikace jednu po druhé, každou několikrát, podle toho, kolik definovaných kombinací nastavení obsahuje (většinou se testují všechny smysluplné kom-



Obr. 2.10: Výstupní časový diagram po simulaci první obálky čítače bez automatického resetování

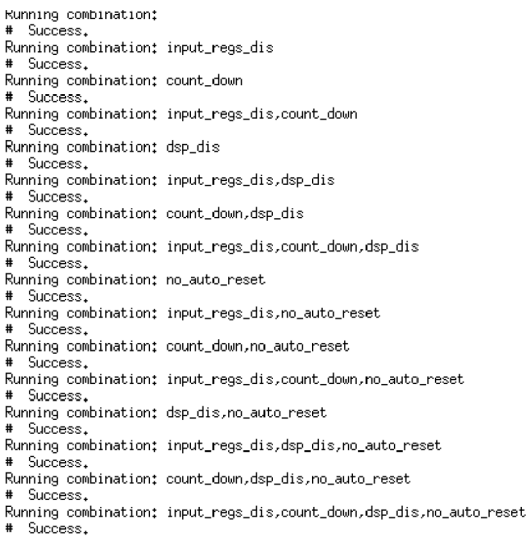
binace generik). Na konci každé spuštěné simulace o určitém nastavení skript hledá výpis „Success“. Pokud ho nenajde, zahlásí chybu a pokračuje další simulací s jiným nastavením. Až projde všechny definované kombinace, přejde na další simulaci/verifikaci.

Skript pro otestování jedné jediné komponenty lze spustit i ručně, výsledek takto spuštěného skriptu je na obrázku 2.11. Za textem „Running combination:“ se nachází nějaké nastavení. Prázdnota znamená výchozí nastavení a ostatní položky jsou pak změny oproti výchozímu nastavení. Ve výchozím nastavení se čítá nahoru, jsou aktivovány vstupní registry, jsou použity DSP bloky a dochází k automatickému resetování. Takže například při druhém průběhu simulace (třetí řádek na obrázku 2.11) se jedná o výchozí nastavení změněné položkou „input_regs_dis“, což znamená, že jsou deaktivovány vstupní registry. V dalším průběhu se testuje čítání dolů, ale už s opět aktivovanými vstupními registry. Následuje kombinace obou předchozích variant. Dalšími položkami jsou „dsp_dis“, kde jsou zakázány DSP bloky, a posledním nastavením je „no_auto_reset“, které deaktivuje automatické resetování čítače. Simulace tedy proběhne pro všechny varianty nastavení, a pokud někde nastane chyba, nedojde k výpisu „Success“, a při automatickém spouštění těchto testů (zde se jednalo pouze o ruční spuštění) je autorovi, jehož komponenta neprošla testem, zaslán e-mail se základní hláškou o chybě.

Jedním z důležitých výsledků získaných pomocí těchto simulací bylo nalezení chyb v čítači na Xilinx FPGA, které se projevují jen ve zřídka používaných nastaveních.

2.4 Integrace a výsledky měření

Jedním z cílů této práce bylo vytvořit čítač, který bude schopen využívat DSP bloky, integrovat jej do zvolených komponent a analyzovat spotřebu zdrojů na FPGA. Tabulka 2.1 zobrazuje rozdíly ve využití zdrojů a maximální možné frekvenci s a bez použití DSP bloků v samostatné komponentě



Obr. 2.11: Výpis ručně spuštěného verifikačního skriptu pro první obálku čítače

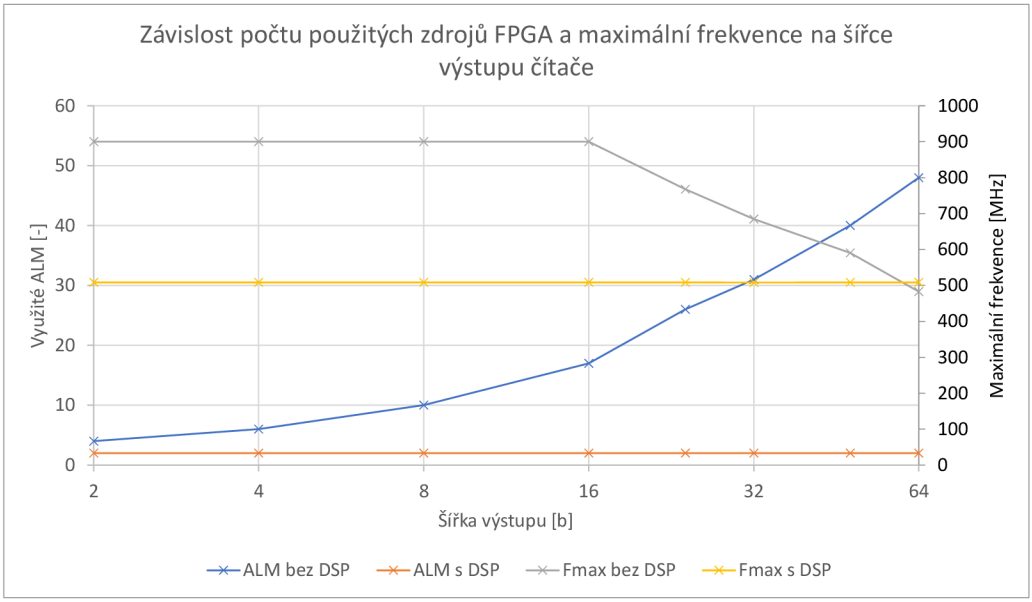
DSP_COUNTER. Při měření byl tento čítač nastaven pro použití na FPGA Stratix 10, vstupní šířka byla nastavena na 27 b a výstupní šířka na 64 b, byly aktivovány vstupní registry a DSP bloky byly v jednom případě zakázány a v druhém povoleny. Záporná znaménka ve sloupci rozdílů poukazují na skutečnost, že se jedná o úbytek parametrů, kladná znaménka pak značí nárůst.

Tab. 2.1: Tabulka porovnání zdrojů a maximální frekvence v DSP_COUNTER

Název měřeného parametru	Parametry bez použití DSP	Parametry s použitím DSP	Rozdíl [%]
ALUT [-]	66	2	$-97,0$
Registry [-]	92	1	$-98,9$
ALM [-]	48	2	$-95,8$
LAB [-]	5	1	$-80,0$
DSP [-]	0	1	–
Maximální frekvence [MHz]	482,86	508,39	$5,3$

Z tabulky 2.1 lze vyčíst, že celý čítač implementovaný pomocí DSP využívá právě jeden DSP blok, dvě vyhledávací tabulky ALUT a jeden registr. Dvě zmíněné ALUT realizují funkce OR mazacího signálu s oběma povolovacími signály. Použitý registr zpožďuje CLK_EN1 v první obálce čítače, viz obrázek 2.6. Pokud by nebyly použity vstupní registry DSP čítače, pak by ani tento registr nebyl zapojen. Zmíněný registr a dvě ALUT byly umístěny do dvou ALM, který se nachází v jednom LAB (viz podkapitola 1.3), proto je tedy jeden celý LAB započítán jako využitý, i když Quartus dokáže do jeho prázdných a částečně využitých ALM implementovat jiné obvody nebo jejich části. Nárůst maximální frekvence o 5,3 % není příliš výrazný, ale potvrzuje to, co bylo avizováno v kapitole 1.5, tedy že pomocí DSP bloků je možné dosáhnout vyšší maximální frekvence.

Dále bylo provedeno měření, při kterém se zjišťovala úroveň výhod při využití DSP bloků s různými šířkami výstupu čítače. Opět byla sledována maximální frekvence a spotřeba zdrojů, která je zde měřena jako počet použitých ALM. Výsledný graf z tohoto měření je na obrázku 2.12. Je vidět, že



Obr. 2.12: Graf znázorňující závislost počtu užitých ALM a maximální frekvence na šířce výstupu čítače při a bez využití DSP

úspora zdrojů je značná v případě 64b čítače, při nižších šířkách je však možné uvažovat o tom, jestli o něco vyšší spotřeba zdrojů není v daném případě výhodnější s tím, že je značně vyšší maximální frekvence. Skutečností ale je, že čítače budou reálně používány v obvodech s frekvencí hodinového signálu maximálně 400 MHz, takže srovnávat maximální frekvence nad touto hodnotou nemá smysl. V tom případě se implementace čítačů pomocí obecné logiky nevyplatí, a to ani u nižších šířek výstupu. Další výhodou čítače s DSP je možnost ponechat šířku výstupu na 64 b, a to ve všech možných zapojeních bez následku poklesu maximální frekvence nebo nárůstu zdrojů, poněvadž ty jsou po celou dobu konstantní. Pokud by se totiž při užití logiky šetřilo na zdrojích a zvolila by se moc malá výstupní šířka čítače, mohl by přetéct (nebo se zastavit) moc brzy, takže až do resetování celého obvodu by neukazoval potřebná data (například počet přijatých rámců by zůstal třeba jen na hodnotě 1023).

Po otestování byla komponenta DSP_COUNTER zapojena do statistických jednotek nacházejících se v komponentách RX_MAC_LITE a TX_MAC_LITE. Rx (přijímací) jednotka se stará o příjem ethernetových rámců, kontroluje jejich správnou velikost, ověřuje CRC apod. Tx (vysílací) jednotka se stará o vysílání rámců a vklad mezirámcových mezer, kontrolu minimální délky vysílaných rámců, počítá a vkládá CRC, ale celkově obsahuje méně statistických čítačů. V těchto komponentách nahradil DSP_COUNTER čítač na UltraScale+, který zde byl implementován pomocí obecných zdrojů FPGA.

Výsledné spotřeby zdrojů uvedené v tabulkách 2.2 a 2.3 pochází ze statistických jednotek zapojených v již zmíněných komponentách RX_MAC_LITE a TX_MAC_LITE.

Tab. 2.2: Tabulka porovnání zdrojů a maximální frekvence v RX_MAC_LITE s DSP čítači

Název měřeného parametru	Parametry bez použití DSP	Parametry s použitím DSP	Rozdíl [%]
ALUT [-]	1741	321	$-81,6$
Registry [-]	3708	1628	$-56,1$
ALM [-]	2124	1301	$-38,7$
LAB [-]	188	81	$-56,9$
DSP [-]	0	22	–
Maximální frekvence [MHz]	412,54	494,80	$19,9$

Tab. 2.3: Tabulka porovnání zdrojů a maximální frekvence v TX_MAC_LITE s DSP čítači

Název měřeného parametru	Parametry bez použití DSP	Parametry s použitím DSP	Rozdíl [%]
ALUT [-]	377	121	$-67,9$
Registry [-]	606	286	$-52,8$
ALM [-]	286	172	$-39,9$
LAB [-]	30	23	$-23,3$
DSP [-]	0	4	–
Maximální frekvence [MHz]	354,23	508,13	$43,4$

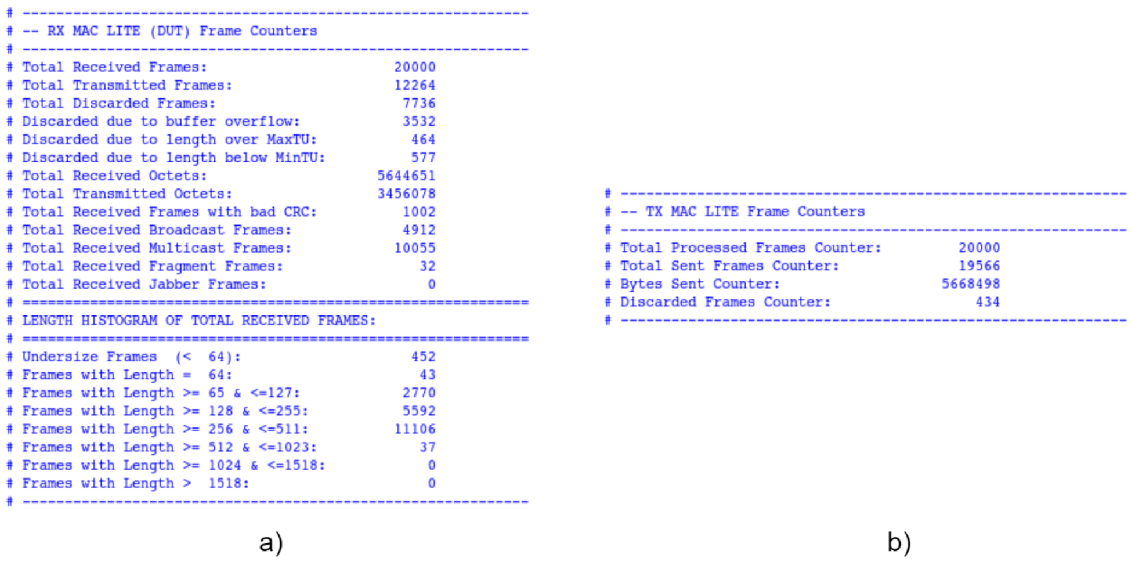
Protože v komponentě TX_MAC_LITE není zapotřebí tolik čítačů jako v RX_MAC_LITE, ale jinak je podobně velká, procentuální rozdíly při a bez použití DSP v tabulce 2.3 nejsou tak velké, jako v tabulce 2.2, alespoň co se počtu zdrojů týče. Naopak při použití DSP čítačů byl v komponentě TX_MAC_LITE mnohem výraznější nárůst maximální frekvence, a to o 43,4 %.

Úbytek základních zdrojů (ALUT a registrů) při užití DSP je výrazně větší, než úbytek nadřazenějších bloků (ALM i LAB), což je očekávané, protože, nadřazené bloky se započítávají, i když nejsou využity celé.

Dále je dobré si uvědomit, že bez ručního použití DSP bloků (tedy jejich instancování v behaviorálním popisu) je syntézní nástroj nedokázal pro vytvoření čítačů využít. To dokazuje nula ve všech tabulkách u položky využitých DSP v případě, kdy se jednalo o čistě behaviorální popis čítače. Jak bylo zmíněno výše, statistická jednotka v komponentě TX_MAC_LITE je mnohem menší než ta v komponentě RX_MAC_LITE, což dokazuje i počet použitých DSP bloků. Na jeden čítač byl použit vždy jeden DSP blok, takže v prvním případě bylo použito 22 čítačů a v druhém jen 4.

Pro ověření zachování funkčnosti obvodů, do kterých byl DSP čítač integrován, byly využity připravené verifikace. Výpisy získané po dokončení verifikací jsou ukázány na obrázku 2.13.

V první části verifikace komponenty RX_MAC_LITE (na obrázku 2.13 a) jsou od vrchu vypsány celkové počty přijatých, dál přeposlaných a zahozených rámců, dále počty zahození rámců kvůli: přeplnění vyrovnávací paměti, rámec větší než je maximální povolená velikost nebo menší než je minimální



Obr. 2.13: Výpis verifikace komponent RX_MAC_LITE a TX_MAC_LITE se zapojenými čítači

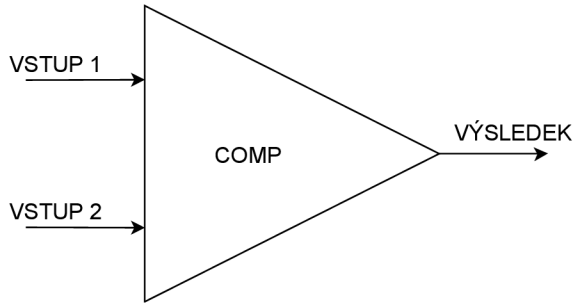
povolená velikost, potom jsou přijaté a dál přeposlané oktety (bajty) a nakonci první části jsou vypsány počty rámců podle určitých vlastností: rámce s nesprávným CRC, všesměrové a vícesměrové rámce, fragmentované rámce a tzv. jabber rámce, což jsou rámce větší než maximální povolená velikost a s nesprávným CRC [44]. V druhé části je histogram přijatých rámců podle jejich délek. Podle výpisu verifikace komponenty TX_MAC_LITE (obrázek 2.13 b) zde byly počítány zpracované a odeslané rámce, odeslané bajty a zahozené rámce. Rámce byly zahazovány při nedostatečné délce, tj. když byly kratší než 64 B. Jelikož verifikace prošly bez chyby, znamená to, že integrací čítače, který je výstupem této práce, nebyla narušena funkčnost těchto komponent.

3 Realizace DSP komparátoru

V této kapitole je rozebráno využití DSP bloků na FPGA Stratix 10 pro implementaci komparátoru. V části návrhu jsou popsány komplikace využití DSP bloků na Stratix 10 pro implementaci komparátoru a nalezení nového způsobu porovnávání dvou hodnot. Díky zkušenostem získaným při návrhu DSP čítače mohlo být nastavování a testování DSP bloku přeskočeno. Po návrhu řešení byly potřebné DSP bloky instancovány a nastaveny v behaviorálním popisu. Nejprve byla vytvořena nejjednodušší funkční verze komparátoru, která byla posléze vylepšena. Podobně jako u čítače, i nad komparátorem byly vytvořeny dvě obálky. První obálka obohacuje komparátor tím, že přidává možnost implementovat jej pomocí obecné logiky FPGA místo DSP bloků. V druhé obálce je prozatím zapojen pouze tento komparátor (přesněji jeho první obálka) a obsahuje funkci, která z generické položky specifikující model FPGA povolí nebo zakáže využití DSP bloků pro implementaci komparátoru. Poté proběhlo důkladné testování vytvořeného komparátoru a následně byl integrován do vybraného obvodu.

3.1 Návrh

Komparátor je prvek typicky se dvěma datovými vstupy a jedním výstupem, viz 3.1. Jeho úkolem je porovnávat hodnoty na svých vstupech a určovat, jestli jsou si rovny nebo ne, případně jestli je jedna hodnota větší nebo menší něž ty druhá. Komparátor, jenž je součástí této práce, dokáže obojí.

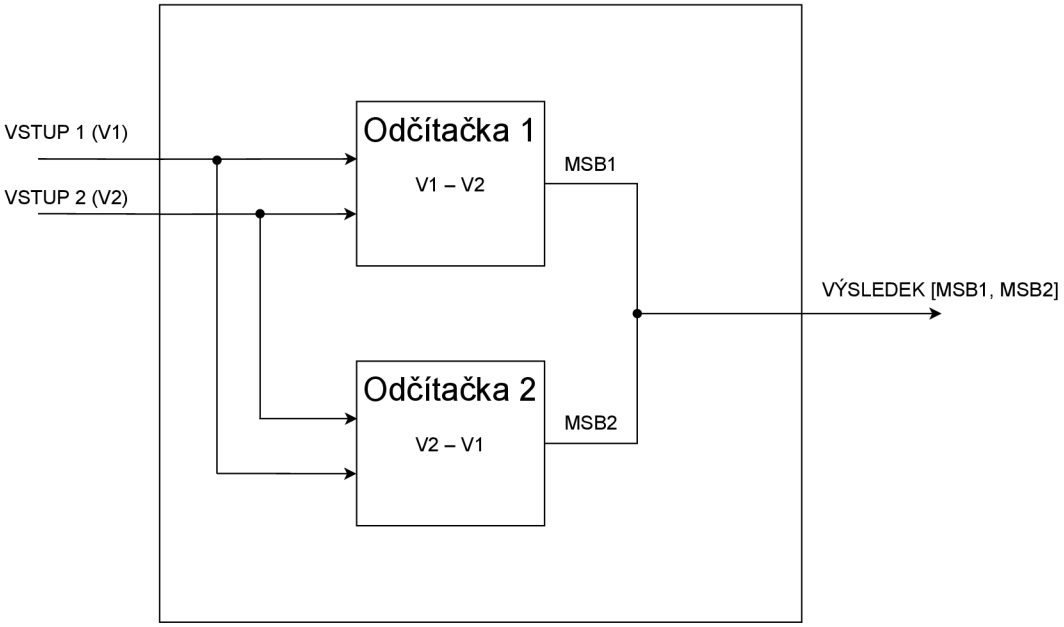


Obr. 3.1: Obecné schéma komparátoru

Při zadávání tohoto úkolu nebylo stanovené, v jakém formátu by výsledek porovnání měl být. Protože mohou nastat 3 různé situace (<, >, =), dávalo smysl, aby výsledný signál měl šířku 2 b. V případě, že se bude jednat o rovnost vstupních hodnot, bude mít výstupní signál hodnotu b'00' (b pouze značí, že se jedná o binární soustavu). Když hodnota na vstupu 1 bude větší než na vstupu 2, výsledek bude b'01', což po převodu do desítkové soustavy dá číslo 1. Má tak vyjadřovat skutečnost, že *první* vstup obsahuje větší hodnotu. A naopak když bude větší hodnota na vstupu 2, výsledek bude ukazovat hodnotu b'10', což znamená 2 v desítkové soustavě, a tím tedy říká, že *druhý* vstup obsahuje větší hodnotu. Výsledek by nikdy neměl obsahovat hodnotu b'11', pokud se tak stalo, muselo někde dojít k chybě.

Zatímco DSP blok v FPGA UltraScale+ přímo takový prvek na porovnávání signálů obsahuje (detektor vzorců), v DSP blocích na Stratix 10 nic takového není. Proto musel být nalezen nový způsob porovnávání dvou hodnot signálů, který by k tomu využíval některých prvků obsažených v DSP bloku.

Řešením je použít dva DSP bloky, ve kterých se budou vstupní signály od sebe odečítat, v každém DSP bloku se budou odečítat v opačném pořadí. Výstup je pak dán výběrem MSB z výsledného rozdílu každého DSP bloku. Tento návrh je znázorněn na obrázku 3.2.

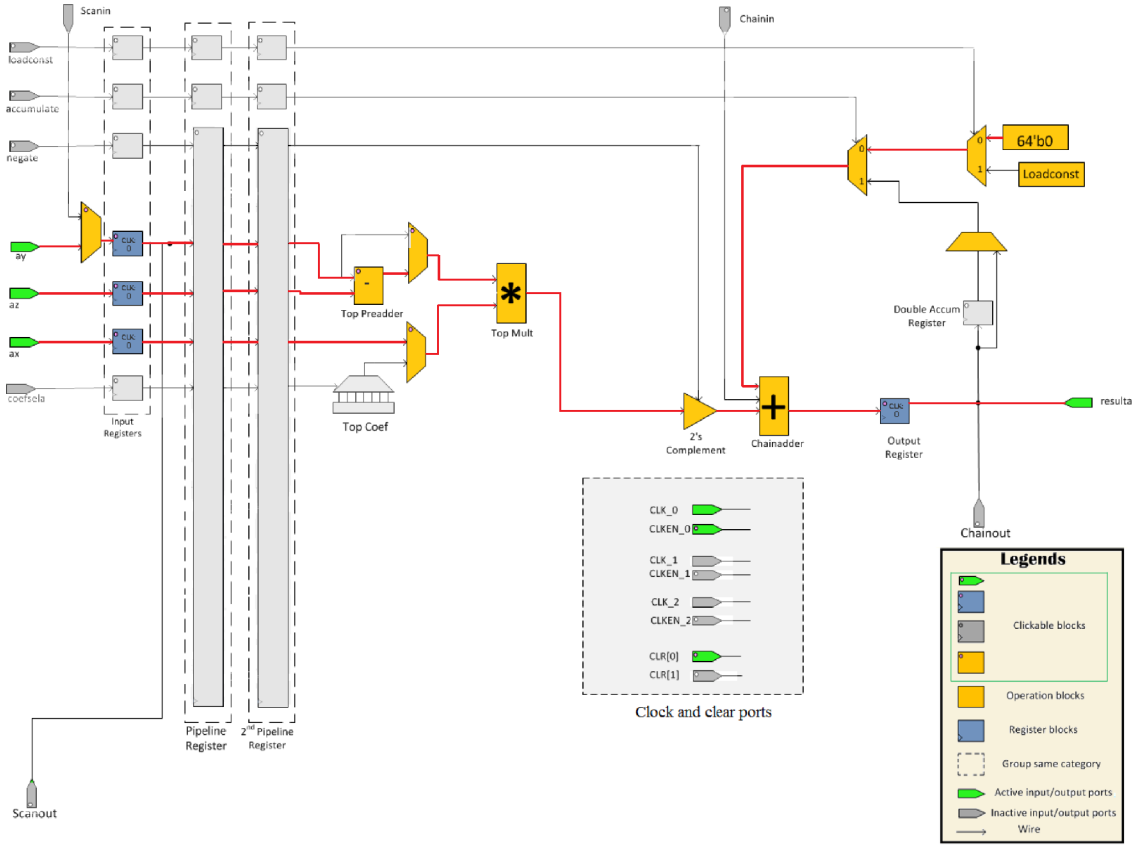


Obr. 3.2: Schéma navrženého komparátoru využívajícího DSP bloky

Vychází se ze skutečnosti, že v binárním formátu jsou čísla se znaménkem reprezentována dvojkovým doplňkem. Kladná čísla vyjádřená jako dvojkový doplněk mají mimo jiné nejvýznamnější bit (MSB) v logické nule, zatímco záporná čísla v logické jedničce. Pokud budou vstupní hodnoty rovnocenné, po odečtení první od druhé a druhé od první budou oba výsledky nulové (všechny bity budou v nule). Na výstup komparátoru se přivedou pouze nejvýznamnější bity obou rozdílů, které tedy mají také hodnotu nula, a proto ve výsledku bude b'00', přesně jak bylo stanoveno. Když bude hodnota na vstupu 1 větší než hodnota na vstupu 2, na výstupu první odčítačky, kde se počítá rozdíl $vstup1 - vstup2$, bude kladné číslo a nejvýznamnější bit zde bude v logické nule. Na výstupu druhé odčítačky, kde se počítá rozdíl $vstup2 - vstup1$, bude záporné číslo, jehož nejvýznamnější bit ponese hodnotu 1. Jestliže budou tyto nejvýznamnější bity připojeny na výstupní signál komparátoru v pořadí MSB1, MSB2, jak je ukázáno na obrázku 3.2, výsledek bude b'01', což tedy značí, že *první* vstup má větší hodnotu, přesně jak je požadováno. Naopak když bude na vstupu 2 větší hodnota než na vstupu 1, rozdíl první sčítačky bude záporný (MSB1 = 1), zatímco rozdíl druhé sčítačky bude kladný (MSB2 = 0). Po připojení těchto bitů na výsledný signál stejným způsobem, jako bylo popsáno předešle, bude na výstupu komparátoru požadovaná hodnota b'10'.

Na obrázku 3.3 je ukázka DSP bloku nastaveného jako odčítačka. Prvkem, ve kterém se provádí odčítání, je vrchní předčítačka (na obrázku 3.3 jako *Top Preadder*). Ve výchozím nastavení tento prvek pracuje jako sčítačka, což bylo změněno jednou generickou hodnotou, a dále se budou data přicházející ze vstupů *ay* a *az* ve vrchní předčítačce odečítat. Jejich rozdíl dál půjde do násobičky, kde se vynásobí

s daty ze vstupu *ax*, která musí být i jako v případě čítače nastavena na konstantní hodnotu 1. Rozdíl dále projde beze změny prvkem vytvářející dvojkový doplněk a ve sčítačce (Chainadder) se sečte s hodnotou 0. Poté se uloží do výstupního registru a v následujícím taktu bude na výstupu DSP bloku. I v tomto případě se zde jednalo o mód DSP bloku 27×27, aby mohly být porovnávány co nejširší signály.



Obr. 3.3: DSP blok v konfiguraci 27×27 nastaven jako odčítačka

3.2 Implementace

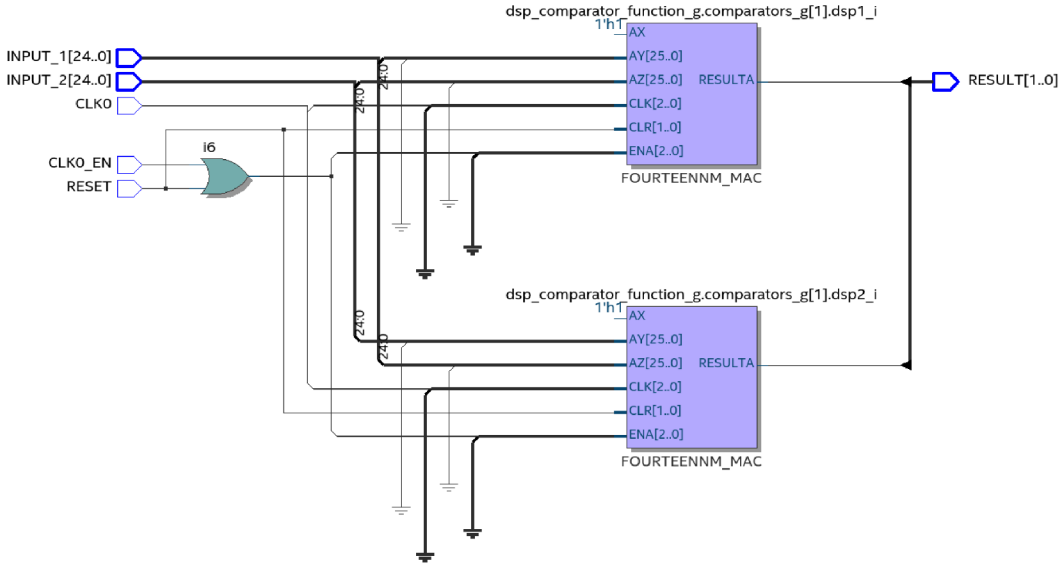
Nejprve byla vytvořena základní verze komparátoru. Ta porovnávala signály s limitovanou maximální šířkou 25 b. V druhé verzi byla vytvořena optimalizace pro operace větší nebo rovno a menší nebo rovno. Třetí a finální verze už umožňuje zřetězení jednotlivých komparátorů tak, aby mohly být porovnávány libovolné šířky vstupních dat (zde je limitem počet DSP bloků nacházející se na daném FPGA). Nad touto základní komponentou byly opět vytvořeny dvě obálky, první z nich umožňuje využití obecné logiky FPGA pro implementaci komparátoru a ta druhá vytváří společné rozhraní pro další komparátory, které využívají jiné typy DSP bloků.

3.2.1 Zapojení DSP bloků

Komponenta využívající DSP bloky se nazývá DSP_COMPARATOR_STRATIX_10_ATOM. Byla vyvíjena od nejjednodušší možné varianty a postupně nabalovala různá vylepšení, proto je i její implementace rozdělena do tří částí.

Základní verze komparátoru

V první verzi komparátoru byly zapojeny pouze dva DSP bloky v nastavení jako odčítačky, což do jisté míry ukazuje i obrázek 3.4. Z tohoto obrázku není zřejmé, že jsou dané DSP bloky nastaveny jako odčítačky, protože jsou zde zobrazeny pouze signály a nikoliv generické hodnoty, pomocí nichž byly DSP bloky nastaveny. V DSP bloku se jednalo zejména o nastavení vrchního vstupu do násobičky, který měl v tomto případě přijít z vrchní předčítačky, a o změnu funkce vrchní předčítačky ze sčítání na odčítání (viz obrázek 3.3). Ne tak zřejmým nastavením byl formát signálů *ay* a *az*. Výchozím nastavením



Obr. 3.4: První verze DSP komparátoru

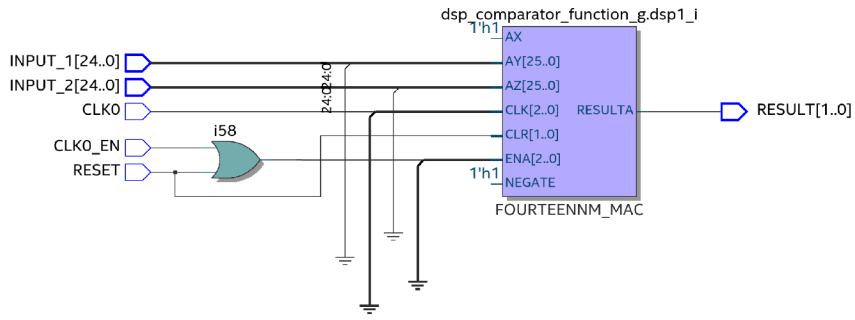
je bezznaménkový formát, ve kterém jsou všechna čísla považována za kladná. Nicméně i když bylo stanoveno, že porovnávat se budou pouze kladné hodnoty, výsledný signál odčítačky záporný být může, a protože formát tohoto výsledku je stejný jako formát vstupů *ay* a *az*, jejich formáty byly přenastaveny na znaménkové. Znaménkový formát na vstupech má za následek to, že šířka vstupů je nyní maximálně 26 b (požadavek DSP bloku). Navíc i když se mají porovnávat pouze kladné hodnoty, bylo potřeba vstupní kladné bezznaménkové signály uměle převést na kladné znaménkové. Toho bylo dosaženo další redukcí použitelné šířky komparátoru, protože na nejvýznamnější bit bylo potřeba napevno nastavit hodnotu 0, aby hodnota signálu byla považována za kladnou. Proto mají vstupy *INPUT_1* a *INPUT_2* na obrázku 3.4 šířku 25 b a po cestě do DSP bloku se k nim připojí uzemněný 1b signál. Lze si všimnout, že do druhého DSP bloku (přesněji na vstupy *AY* a *AZ*) vstupují *INPUT_1* a *INPUT_2* v opačném pořadí než do prvního – tak jak bylo navrženo. Výsledkem porovnání (*RESULT*) je pak

spojení nejvýznamnějších bitů obou rozdílů. Zdroje hodinového i povolovacího signálu se zde nachází po jednom (v kontrastu s čítačem), protože bylo usouzeno, že není potřeba zastavovat zvlášť vstupní a zvlášť výstupní registry. Ale stejně jako v případě čítače zde vedou vstupní povolovací signály do logického členu OR stejně jako signál *RESET*, aby mohlo dojít k resetování komparátoru i v případě, když bude povolovací signál v logické nule.

Optimalizovaná verze komparátoru

Již bylo zmíněno, že druhou verzí byl komparátor optimalizován pro operace větší nebo rovno a menší nebo rovno. Bez této optimalizace by uživatel vyžadující jednu z těchto operací musel nějak upravit výstup z komparátoru. Pokud by znal jen formát výstupu (že rovnost = b'00', větší než = b'01' a menší než = b'10'), pravděpodobně by použil nějakou logickou funkci. V případě operace větší nebo rovno by ověřoval, jestli nejvýznamnější bit výsledku má hodnotu nula. Když by byl nula, věděl by, že výsledek komparátoru byl buď b'00' nebo b'01', tedy že vstupy jsou si buď rovny, nebo že první vstup je větší než ten druhý. Naopak v případě operace menší nebo rovno by takhle kontroloval nejméně významný bit výsledku komparátoru. Ač to není mnoho, zvýší to spotřebu logických zdrojů. Kdyby uživatel věděl, jak komparátor uvnitř funguje, mohl by se dívat vždy jen na jeden bit výsledku komparátoru. Věděl by, že výstup první odčítačky (resp. nejvýznamnější bit výsledku) má hodnotu 0, když je první vstup větší nebo roven druhému. Stačí tedy přidat invertor (což je nejjednodušší logický člen), aby v případě, že vstup 1 je větší nebo roven vstupu 2, byla výsledkem logická jednička. Naopak nejméně významný bit by využil stejným způsobem, jako byl v předešlém případě využit nejvýznamnější bit, když by vyžadoval operaci menší nebo rovno (opět s využitím invertoru). V každém z těchto případů jsou zde nějaké sankce, buď „zbytečné" plýtvání logickými prvky, nebo čas vývojáře (pokud funkci komparátoru již nezná).

Tato optimalizace vychází z druhého z předešle uvedených případů. Pro operace větší nebo rovno a menší nebo rovno stačí použít pouze jeden DSP blok nastavený jako odčítačka. Komparátor provádějící operaci „>=" je znázorněn na obrázku 3.5.



Obr. 3.5: Komparátoru v módu pro operaci větší nebo rovno

V tomto případě je první vstup odečítaný od druhého. Když by hodnota na vstupu 1 byla větší než hodnota na vstupu 2, nejvýznamnější bit by měl hodnotu 0, což v digitálních obvodech obvykle značí nepravdu. Aby značil pravdu, tedy že hodnota na vstupu 1 je opravdu větší než hodnota na vstupu 2, měl by nejvýznamnější bit mít hodnotu 1. Toho bylo dosaženo tak, že výsledek z odčítačky je invertován

pomocí prvku vytvářející dvojkový doplněk signálu. Tento prvek je řízený vstupem *NEGATE*, který při nastavené hodnotě 1 vytváří dvojkový doplněk z příchozích dat (viz čítání nahoru a dolů v kapitole 2).

Lze si všimnout, že výsledek komparátoru je široký 2 b, ačkoliv by bylo možné jej vyjádřit pouze 1 b. Je to z toho důvodu, že se stále jedná o tu samou komponentu, která využívá 2b výsledku v normálním nastavení (>, <, =). Pro jednoduchost byl výsledný bit zdvojen, takže v případě pravdy je výsledkem hodnota b'11' a v případě nepravdy b'00'.

Komparátoru v konfiguraci pro operaci menší nebo rovno by vypadal stejně jako na obrázku 3.5, jediným rozdílem by byly opačně zapojené vstupy. Výhodou této optimalizace je tedy nejen to, že uživatel nemusí znát vnitřní funkci komparátoru, nebo že nevyužije logiku navíc, ušetří se tím i celý jeden DSP blok. Typ operace komparátoru uživatel nastaví výběrem jednoho z módů „><=“, „>=“ a „<=“.

Finální verze komparátoru

V poslední verzi byl pomocí zřetězení komparátorů odstraněn limit 25 b pro vstupní signály, ale pouze pro výchozí mód (>, <, =). Při zadané šířce větší než 25 b je nejprve třeba určit celkový počet dílčích komparátorů, který bude potřeba pro dosažení požadované výsledné šířky. Pokud požadovaná šířka nebude přesně násobek 25, jeden z dílčích komparátorů nebude využívat plnou šířku vstupů. Počet komparátorů, které budou plně obsazeny (jejichž šířka bude 25 b) se vypočítá jako

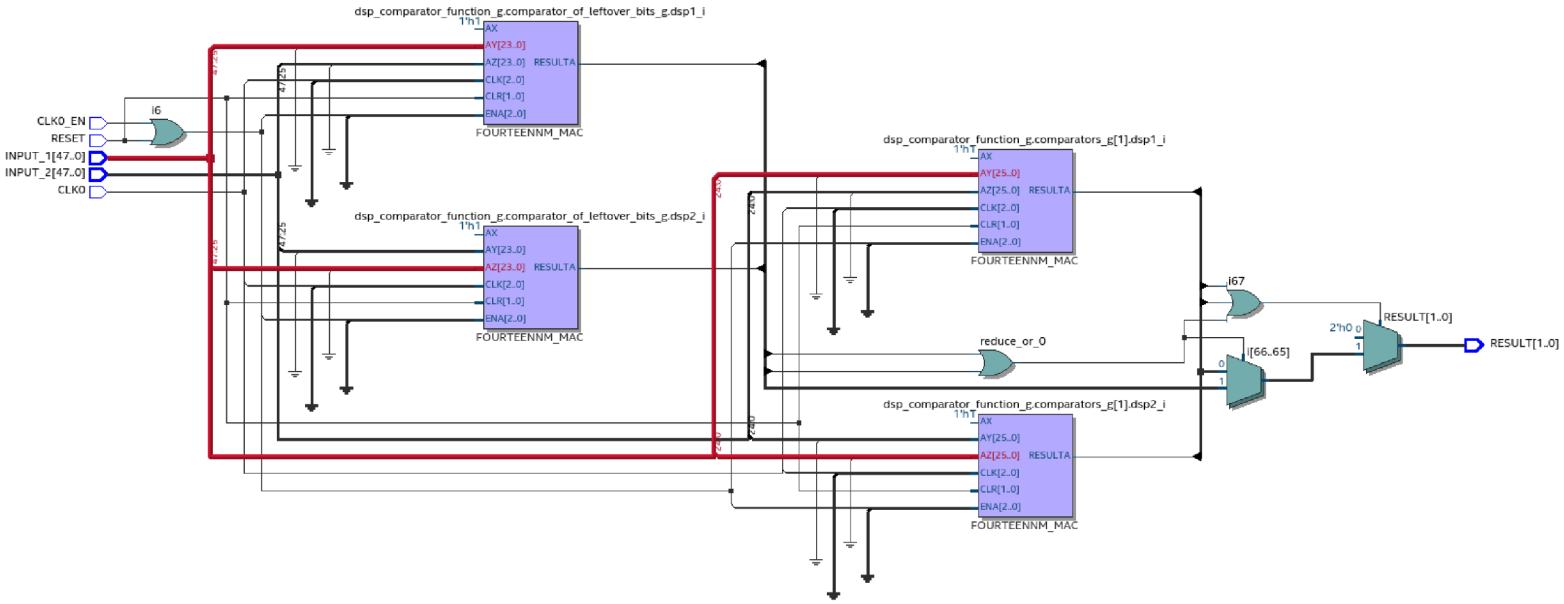
$$pocet\_plnych\_komparatoru = sirka\_vstupu/25, \tag{3.1}$$

kde čísla za desetinnou čárkou jsou ignorována. Toto je číslo, které se bude nejvýrazněji měnit. Pokud zbytek po tomto dělení není nulový, bude se muset zřetězit ještě jeden komparátor, který bude porovnávat jen tyto zbytkové bity. Na obrázku 3.6 se nachází komparátor v módu „>, <, =“ s nastavenou šířkou vstupů na 48 b. Takový komparátor je možné využít například pro porovnávání MAC adres.

Z rovnice 3.1 vyplývá, že bude použit jeden plný komparátor, a protože 48 není násobek 25, bude použit ještě jeden dílčí komparátor se zbytkovou šířkou, což je 23. Plný komparátor, resp. dva DSP bloky, které jej tvoří, jsou z pohledu obrázku ty napravo. To lze poznat tak, že mají 26b šířky vstupů ($AY[25..0]$ a $AZ[25..0]$). Druhý komparátor porovnávající zbylé bity se tedy nachází více vlevo, šířka vstupů je zde 24 b. Co z obrázku 3.6 nelze poznat je, že do komparátoru porovnávajícího zbylé bity jsou přivedeny vrchní bity vstupních signálů, přesně tedy *INPUT_1*[47..25] a *INPUT_2*[47..25], zatímco druhý (plný) komparátor porovnává spodních 25 b vstupních signálů, tedy *INPUT_1*[24..0] a *INPUT_2*[24..0]. Přívody vstupu *INPUT_1* do jednotlivých DSP bloků jsou na obrázku 3.6 zvýrazněny červeně.

Logika před výstupem komparátoru je zde proto, že je potřeba vybrat správný výsledek. V tomto případě je vybíráno ze dvou dílčích výsledků. Pokud je již z porovnání vrchních bitů jasné, že jedna hodnota vstupu je větší nebo menší než ta druhá, přivede se na výstup celého komparátoru tento výsledek a výsledek druhého dílčího komparátoru se ignoruje. Pokud ve vrchním komparátoru vyjde rovnost porovnávaných signálů, na výstup se přivede výsledek ze spodního komparátoru. Poslední multiplexor před výstupem komparátoru tomuto výstupu přiřadí hodnotu b'00' (na obrázku 3.6 jako 2'h0, což znamená to samé) v případě, že všechny dílčí výsledky jsou nulové, což zjistí operací OR, jinak přivede jeden z výsledků dílčích komparátorů.

Z toho vyplývá, proč není možné zřetězit více dílčích komparátorů pro módy „>=“ a „<=“. Z výsledku prvního dílčího komparátoru není jasné, jestli je jedna vstupní hodnota větší než druhá

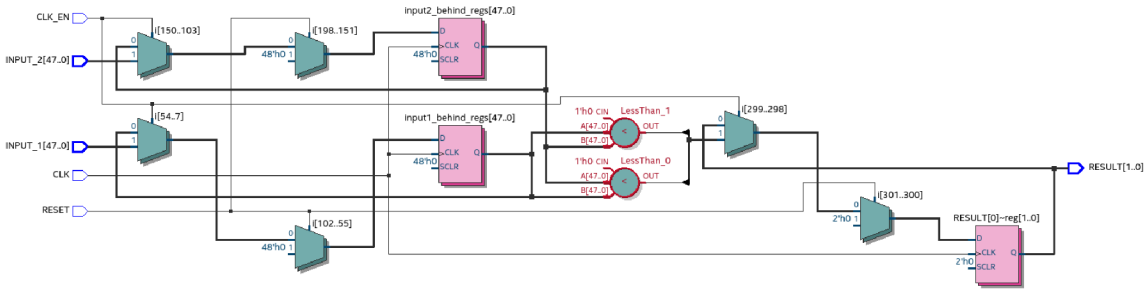


Obr. 3.6: Komparátoru se šířkou vstupů 48 b

a tento výsledek by se měl zapsat na výstup, nebo jestli si jsou vstupní hodnoty rovny a má se brát v potaz výsledek druhého dílčího komparátoru.

3.2.2 DSP komparátor pro Stratix 10

V první obálce komparátoru vytvořeného z DSP bloků se nachází jeho obecná varianta s názvem DSP_COMPARATOR_STRATIX_10, která pro implementaci využívá obecnou logiku FPGA. Na obrázku 3.7 je ukázka 48b komparátoru implementovaného v logice.



Obr. 3.7: Komparátoru vytvořený v logice

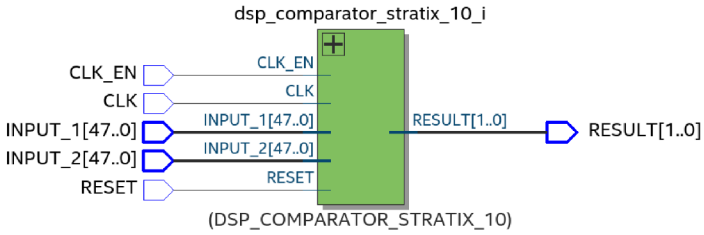
Nejde ani o to, jak obvod funguje, ale spíš o využité logické prvky, kterých je značně víc než ve variantě s DSP, což dále vyplývá z měření v kapitole 3.4. Jsou zde zobrazeny pouze typy použitých prvků, při zohlednění šířek signálů jich bude ve výsledku mnohem víc. To se hlavně týká prvků provádějící porovnání, které jsou na obrázku 3.7 vyznačeny červeně a které jsou implementovány pomocí vyhledávacích tabulek (LUT) zhruba takového množství, jaká je šířka porovnávaných signálů (zde kolem 48, záleží na optimalizaci syntézního nástroje). Implementace pomocí logiky však není limitována šířkou vstupů pro operace větší nebo rovno a menší nebo rovno.

Co se rozhraní týče, je úplně totožné se rozhraním komponenty DSP_COMPARATOR_STRATIX_10_ATOM až na jednu generickou hodnotu, kterou je nastavení implementace komparátoru (logika nebo DSP).

3.2.3 Obecný DSP komparátor

Komponenta obecného komparátoru s názvem DSP_COMPARATOR v tuto chvíli obsahuje pouze zapojenou již dříve zmíněnou obálku komparátoru a jednu funkci. Na obrázku 3.8 je proto vidět jen přeportovanou komponentu DSP_COMPARATOR_STRATIX_10. Do budoucna se počítá s tím, že do této obálky budou zapojeny další instance komparátorů, které budou využívat různé typy DSP bloků na různých modelech FPGA.

Na rozdíl od předešle zmíněné komponenty je zde místo generické hodnoty, pomocí které se nastavuje implementace komparátoru (logika nebo DSP), jiná generická hodnota, jíž se specifikuje model používaného FPGA. Pomocí funkce obsažené v této obálce je podle názvu FPGA nastavena generická hodnota zapojené komponenty tak, aby pokud možno implementovala komparátor pomocí DSP bloků. To znamená, že v tuto chvíli bude komponenta DSP_COMPARATOR_STRATIX_10 využívat DSP bloky vždy, když bude specifikováno FPGA Stratix 10.



Obr. 3.8: Obecný komparátor

3.3 Simulace a testování

Princip testování komparátoru byl velmi podobný jako v případě čítače. V souboru testbench byly generovány náhodné hodnoty signálů, které byly připojeny na vstupy komparátoru. Náhodné hodnot byly generovány od nuly až po maximální možnou hodnotu danou šířkou signálů. Povolovacím a mazacím signálům byly podobně měněny hodnoty z nuly na jedničku a obráceně v náhodné okamžiky. V souboru testbench byly také instance komparátorů, pro každý z módů jeden. Výsledky byly porovnávány s výsledky z testovaného komparátoru a jejich shody byly vypisovány do výpisového okna. Byla přidána možnost pro výpis jednotlivých iterací ve formátu: <hodnota vstupu 1> >/</=/>=/<= <hodnota vstupu 2>. To může zjednodušit práci při ladění, protože hned ve výpisu je vidět, o jakou chybu se jedná, a nemusí se to dohledávat v časovém diagramu. Rozšířený výpis, resp. jeho konečný úryvek, je ukázán na obrázku 3.9 b, v kontrastu s jednoduchým výpisem (obrázek 3.9 a). Oba výpisy pochází ze simulace trvající 20 tisíc taktů. Je zřejmé, že rozšířený výpis je výhodný spíše pro prvotní ladění, kdy se simulace spouští například na 10–20 taktů. Pro důkladnější testování, kdy se simulace spouští na desítky tisíc taktů, se nehodí, protože nejspíš nikdo nebude procházet tolik výpisů a navíc to dělá simulaci náročnější. Na konci simulace se vypíše vyhodnocení celého průběhu, „Success“ v případě bezchybného průběhu a „There were some incorrect results“ v případě jedné či více chyb.

Oproti testování čítače zde nejsou různé soubory testbench pro různé obálky komparátoru. Nejprve byl testbench vytvořen pro testování komponenty DSP_COMPARATOR_STRATIX_10_ATOM, přesněji jeho základní verzi. S každou novou verzí komparátoru byl upraven i testbench a proběhlo testování. Pro ověření funkčnosti první obálky komparátoru stačilo jen testbench mírně upravit. Druhá obálka testována nebyla, protože se jedná víceméně o stejnou komponentu. Chyba v druhé obálce mohla nastat takřka jen v dané funkci, ale protože nebyla nijak složitá, stačilo, že prošla přes tzv. „code review“, což je kontrola kódu někým jiným.

Na obrázku 3.10 je výstupní časový diagram ze simulace první obálky komparátoru, tedy komponenty DSP_COMPARATOR_STRATIX_10. Lze si všimnout konfigurace komparátoru ve vrchní části diagramu, kde je nastaven mód „><=“, jsou využity DSP bloky, šířka vstupních dat je 3 b (pro jednodušší ladění a také aby bylo možné narazit na nějaké rovnající se hodnoty) a jsou deaktivovány vstupní registry. Následně jsou vypsány konstanty, které byly na základě šířky vstupních dat vypočítány. *NUM_OF_FULL_COMPARATORS* udává počet plných dílčích komparátorů (tedy s šířkou 25 b), což je v tomto případě správně nula. *TOTAL_NUM_OF_COMPARATORS* říká, kolik dílčích komparátorů bude potřeba celkově, zde jen jeden na porovnání zbytkových bitů, jejichž počet je dán poslední konstantou *LEFTOVER_BITS* s hodnotou 3.

```
# 1000 results are correct.
# 2000 results are correct.
# 3000 results are correct.
# 4000 results are correct.
# 5000 results are correct.
# 6000 results are correct.
# 7000 results are correct.
# 8000 results are correct.
# 9000 results are correct.
# 10000 results are correct.
# 11000 results are correct.
# 12000 results are correct.
# 13000 results are correct.
# 14000 results are correct.
# 15000 results are correct.
# 16000 results are correct.
# 17000 results are correct.
# 18000 results are correct.
# 19000 results are correct.
# 20000 results are correct.
# Success.
```

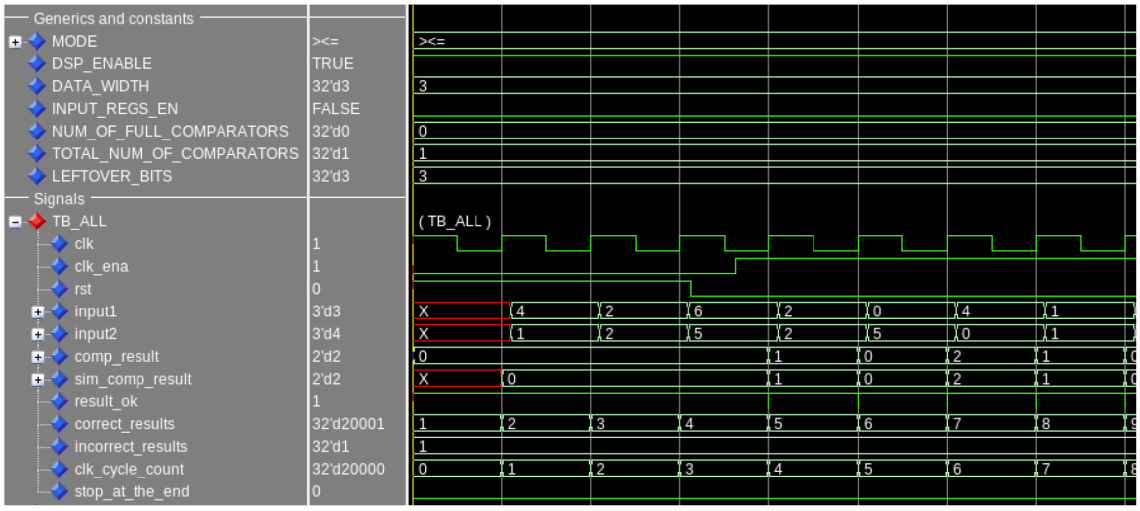
a)

```
# 6 > 3
# 7 > 5
# 2 < 4
# 2 > 0
# 7 > 2
# 2 < 5
# 4 > 2
# 4 > 0
# 4 > 2
# 0 < 4
# 1 < 5
# 2 > 1
# 2 < 4
# 2 < 7
# 7 > 2
# 6 < 7
# 5 > 1
# 2 < 3
# 3 < 4
# 20000 results are correct.
# Success.
```

b)

Obr. 3.9: Výpis z testování komparátoru: a) bez rozšířeného výpisu, b) s rozšířeným výpisem

Komparátor začne pracovat až čtvrtém taktu, kdy mazací signál nabude nulové hodnoty a povolovací signál naopak hodnoty 1. V tomto taktu se vezmou vstupní hodnoty 6 a 5 a ty se porovnají.



Obr. 3.10: Časový diagram ze simulace komparátoru pro Stratix10

V následujícím taktu je vidět, že výsledek komparátoru *comp_result* a výsledek simulovaného komparátoru *sim_comp_result* jsou 1, což tedy správně sděluje, že *první* vstupní hodnota byla větší. Protože se oba výsledky rovnají, signál *result_ok*, který je porovnává, hlásí jejich shodu, ostatně jako po celou dobu simulace (v prvním taktu je neshoda ignorována, protože *sim_comp_result* nemá jasnou hodnotu). Zvýší se i počet správných výsledků *correct_results* ze 4 na 5 a počet nesprávných výsledků

zůstane na výchozí hodnotě. Poslední dva signály slouží pouze k výpisu vyhodnocení průběhu celé simulace až na jejím úplném konci (viz poslední řádek výpisu na obrázku 3.9).

I tato simulace byla přidána do sbírky automatických testů, které byly popsány v podkapitole 2.3. Výpis po ručním spuštění testovacího skriptu je na obrázku 3.11. Zde byl ve výchozí konfiguraci nastaven mód „><=“, byly aktivovány vstupní registry a použity DSP bloky a šířka vstupních dat byla 25 b. Položka „input_regs_dis“ deaktivovala vstupní registry, „high_data_width“ nastavila šířku vstupních dat na 128 b (byla použita pouze pro tento mód), „dsp_dis“ vytvořilo komparátor v logice a „mode1“ a „mode2“ nastavovaly mód na „>=“ a „<=“.

```
Running combination:
# Success.
Running combination: input_regs_dis
# Success.
Running combination: high_data_width
# Success.
Running combination: input_regs_dis,high_data_width
# Success.
Running combination: high_data_width,dsp_dis
# Success.
Running combination: input_regs_dis,high_data_width,dsp_dis
# Success.
Running combination: mode1
# Success.
Running combination: input_regs_dis,mode1
# Success.
Running combination: dsp_dis,mode1
# Success.
Running combination: input_regs_dis,dsp_dis,mode1
# Success.
Running combination: mode2
# Success.
Running combination: input_regs_dis,mode2
# Success.
Running combination: dsp_dis,mode2
# Success.
Running combination: input_regs_dis,dsp_dis,mode2
# Success.
```

Obr. 3.11: Výpis ručně spuštěného verifikačního skriptu pro komparátor

3.4 Integrace a výsledky měření

Také v případě komparátoru, přesněji komponenty DSP_COMPARATOR, byly měřeny spotřeby zdrojů a maximální frekvence u variant s využitím DSP bloků a bez. Výsledky těchto měření jsou ukázány v tabulce 3.1. Při měření byla nastavena šířka vstupů na 25 b a mód na „><=“, vstupní registry byly aktivovány.

Základní logické prvky (ALUT a registry) nebyly v DSP variantě spotřebovány téměř žádné, což ukazují i procentuální úbytky $-98,2\,\%$ v případě ALUT a $-100\,\%$ v případě registrů. Jediná spotřebovaná ALUT slouží pro výběr výsledku z dílčích komparátorů a na operaci OR mazacího a povolovacího signálu v zapojené komponentě DSP_COMPARATOR_STRATIX_10_ATOM, jak bylo popsáno v podkapitole 3.2. Maximální frekvence značně klesla, nicméně to ani zde není fatální pro finální využití, protože standardní frekvence hodinového signálu pro obvody, ve který bude zapojen tento komparátor, bude maximálně 400 MHz.

Výsledky měření komparátoru optimalizovaného pro operaci „>=“ dopadly podobně, viz tabulka 3.2. Ve variantě s DSP je vidět, že byl opravdu použit jen jeden DSP blok místo dvou. Využitá vyhledávací tabulka ALUT realizuje funkci OR mazacího a povolovacího signálu. Při implementaci

Tab. 3.1: Tabulka porovnání zdrojů a maximální frekvence v DSP_COMPARATOR s módem „><="

Název měřeného parametru	Parametry bez použití DSP	Parametry s použitím DSP	Rozdíl [%]
ALUT [-]	55	1	$-98,2$
Registry [-]	52	0	$-100,0$
ALM [-]	34	1	$-97,1$
LAB [-]	4	1	$-75,0$
DSP [-]	0	2	–
Maximální frekvence [MHz]	665,78	471,25	$-29,2$

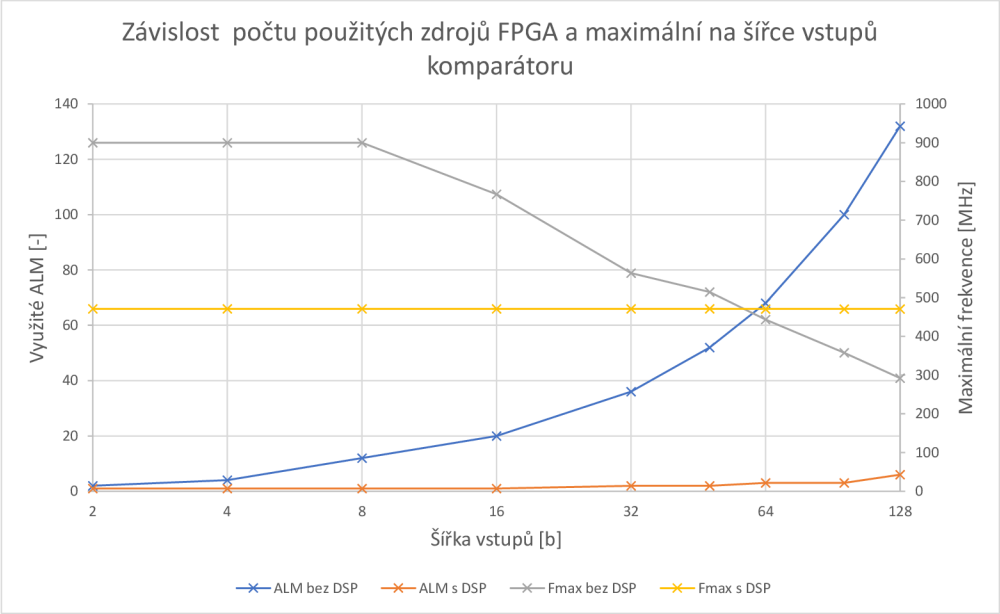
pomocí logiky bylo v módu „>=“ využito asi o 50 % méně ALUT než ve standardním módu „><=“, s čímž poklesl i počet využitých ALM. Dále si lze všimnout mírného nárůstu maximální frekvence oproti komparátoru využívajícího DSP ve standardním módu. Maximální frekvence se zvýšila přibližně o 130 MHz, což způsobilo, že oproti implementaci s DSP bloky je nyní o 40,4 % vyšší.

Tab. 3.2: Tabulka porovnání zdrojů a maximální frekvence v DSP_COMPARATOR s módem „>=“

Název měřeného parametru	Parametry bez použití DSP	Parametry s DSP komparátory	Rozdíl [%]
ALUT [-]	27	1	$-96,3$
Registry [-]	51	0	$-100,0$
ALM [-]	27	1	$-96,3$
LAB [-]	3	1	$-66,7$
DSP [-]	0	1	–
Maximální frekvence [MHz]	794,28	473,48	$-40,4$

V grafu na obrázku 3.12 je vidět různé spotřeby zdrojů (konkrétně prvků ALM) pro šířky vstupů od 2 do 128 b. Ve variantě komparátoru bez DSP je nárůst spotřebovaných zdrojů se zvyšující se šířkou vstupů značný, zatímco ve variantě s DSP je minimální. V druhé ze zmíněných variant se kromě ALM, v nichž roste počet vyhledávacích tabulek ALUT pro výběr výsledku z dílčích komparátorů, navíc zvyšuje počet využitých DSP bloků, které jsou řetězeny pro větší šířky porovnávaných signálů. Pro šířku 128 b bylo využito 7 ALUT, které Quartus rozmístil do 6 ALM, a 12 DSP bloků. Při vhodném rozmístění logických prvků je možné dosáhnout lepších propojů, a tím pádem i vyšší maximální frekvence, a protože zde byl syntetizován pouze tento komparátor, Quartus místem na FPGA nemusel šetřit. Jinak by mohl pro všech 7 ALUT využít 4 ALM, protože se jedná o funkce o méně než 4 proměnných, a ALUT v jednom ALM mohl tedy rozdělit na dvě samostatné. Dále je vidět, že ve variantě bez DSP po určité šířce vstupů maximální frekvence lineárně klesá a při šířce nad přibližně 60 b je nižší, než maximální frekvence varianty s DSP, která je konstantní.

Hotový komparátor byl zapojen do komponenty RX_DMA_SW_MANAGER, který je součástí nového DMA (Direct Memory Access – přímý přístup do paměti) modulu a kde ve dvou případech porovnává dva 16b signály. Jedná se o poměrně velkou komponentu, a proto je úspora zdrojů uvedená v tabulce 3.3 nízká. Také šířka 16 b není dostatečná na to, aby rozdíl ve spotřebovaných zdrojích byl



Obr. 3.12: Graf znázorňující závislost užitých ALM na šířce vstupů komparátoru při a bez využití DSP

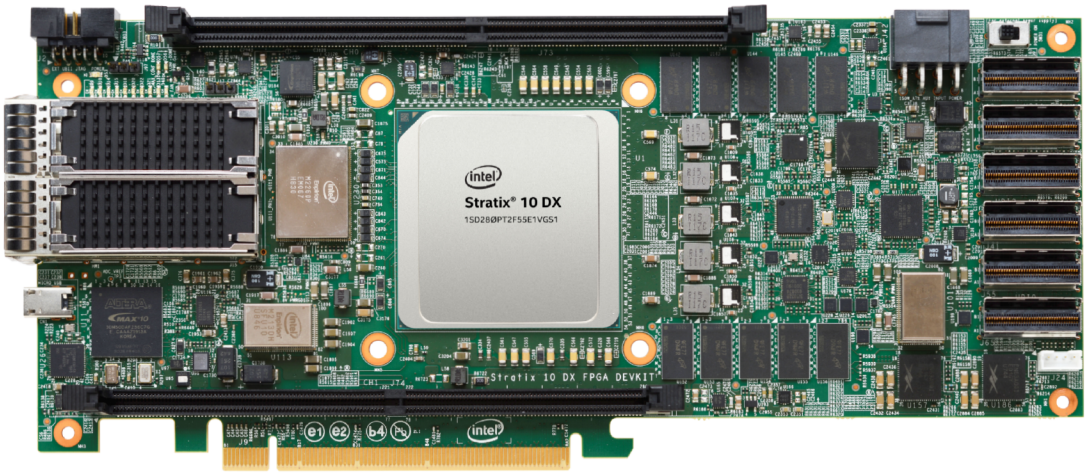
markantní, což vyplývá i z grafu na obrázku 3.12. Nicméně po zapojení dříve vytvořených DSP čítačů do zmíněné komponenty RX_DMA_SW_MANAGER, přesněji 4 čítačů pro každý z 8 DMA kanálů, tedy celkem 32 čítačů, úspora zdrojů pak byla mnohem zřetelnější. S DSP čítači se ušetřilo přes 50 % ALUT a registrů. Je vidět, že z pohledu maximální frekvence v tomto případě využití DSP bloků není zcela výhodné, protože zapojení s DSP komparátorem způsobilo pokles maximální frekvence o 11,4 %. Celá komponenta však má běžet na hodinovém signálu s frekvencí 200 MHz, snížení maximální frekvence zde tedy není podstatné. Maximální frekvenci při použití DSP komparátorů vylepšilo přidání DSP čítačů, její pokles oproti původní verzi byl tak zvýšen z $-11,4\,\%$ na $-6,9\,\%$.

Tab. 3.3: Tabulka porovnání zdrojů a maximální frekvence v SW manageru s DSP komponentami

Název měřeného parametru	Parametry bez použití DSP	Parametry s DSP komparátory	Rozdíl [%]	Parametry s DSP komparátory i čítači	Celkový rozdíl [%]
ALUT [-]	3450	3400	$-1,4$	1611	$-53,3$
Registry [-]	3904	3747	$-0,4$	1774	$-54,6$
ALM [-]	3189	3162	$-0,1$	2291	$-28,2$
LAB [-]	365	362	$-0,1$	259	$-29,0$
DSP [-]	0	4	–	36	–
Maximální frekvence [MHz]	270,71	239,87	$-11,4$	251,95	$-6,9$

Verifikace samotné komponenty RX_DMA_SW_MANAGER neexistuje, jeho funkčnost se ověřuje při verifikaci celé RX (přijímací) části DMA modulu. Tímto způsobem bylo tedy také ověřeno správné

zapojení komparátorů a čítačů do této komponenty. Protože karta COMBO-400g1 ještě není dostupná, DMA modul byl testován na vývojovém kitu Intel Stratix 10 DX FPGA Development Kit [45], který FPGA Stratix 10 DX 2800 obsahuje. Tento vývojový kit je ukázán na obrázku 3.13.



Obr. 3.13: Vývojový kit Intel Stratix 10 DX FPGA Development Kit [45]

4 Závěr

V první části této práce byly vysvětleny souvislosti síťových zařízení a technologie FPGA, jejichž architektura byla dále trochu detailněji rozebrána, stejně jako obecný postup při návrhu digitálního obvodu na FPGA. Ještě podrobněji byly dále popsány DSP bloky na FPGA Virtex UltraScale+ XCVU7P a Stratix 10 DX 2800. Bylo zjištěno, že DSP bloky na Stratix 10 jsou jednodušší ve srovnání s těmi na UltraScale+, že jsou to spíše prvky s násobičkou a dalšími několika funkcemi postavenými okolo ní. DSP bloky na UltraScale+ kromě násobičky navíc obsahují prvek pro rozpoznávání vzorců a konfigurovatelnou logickou jednotku, která díky čtyřem vstupním multiplexorům může pracovat s poměrně velkým počtem různých vstupů.

V praktické části byly popsány realizace čítače a komparátoru, které byly implementovány pomocí DSP v FPGA Stratix 10 DX 2800. Obě tyto komponenty byly navrženy, implementovány a otestovány. Poté byla změřena úspora zdrojů a změna maximální frekvence při implementaci s DSP bloky oproti té bez. Úspora zdrojů byla výrazná, u obou vytvořených komponent se jednalo o úsporu více než 95 % ALM prvků. Maximální frekvence v případě DSP čítače vzrostla o 5,3 %, zatímco u komparátoru naopak o 47,6 % klesla. Nad rámec zadání byly tyto komponenty integrovány do dalších obvodů, které jsou součástí firmwaru pro kartu COMBO-400g1 vyvíjenou sdružením CESNET. Jedním z těchto obvodů je DMA modul, ve kterém byly zapojeny jak čítače, tak komparátory. Tento DMA modul byl testován na vývojovém kitu Intel Stratix 10 DX FPGA Development Kit, čímž tak byly otestovány i vytvořené komponenty. U komponent s integrovanými čítači a komparátory byla také měřena úspora zdrojů a změna maximální frekvence. Protože čítač našel více využití než komparátor, měl výraznější vliv na úsporu spotřebovaných zdrojů.

Tuto práci lze rozvinout například rozšířením implementace čítače o podporu zřetězení několika DSP bloků, a tím dosáhnout širšího výstupu. Čítač je také možné upravit tak, aby se dala nastavit libovolná maximální hodnota (MAX_VAL). V případě odčítání by maximální hodnota udávala číslo, od kterého se začne čítat dolů. DSP bloky na Stratix 10 DX 2800 lze využít také jen jako odčítačky, což vyplývá z vytvořeného komparátoru, který již odčítačky využívá. Lehkou obměnou je pak možné vytvořit sčítačku. S využitím DSP bloků je možné implementovat operace OR nebo AND nad jedním signálem, což se používá pro zjištění, zda-li se v daném signálu nachází alespoň jedna jednička (OR) nebo alespoň jedna nula (AND).

Tato práce představuje alternativní způsob využití DSP bloků na FPGA Stratix 10 DX 2800 a analyzuje výhody a nevýhody uvedených řešení. Mezi zásadní výhody patří úspora obecných zdrojů FPGA a také upotřebení jinak nevyužitých DSP bloků. Jistým omezením těchto řešení může být maximální frekvence, která v některých případech při využití DSP bloků byla nižší, než při využití obecné logiky. Přesto hodnota maximální frekvence nijak neomezuje použití implementovaných řešení v zamýšlených aplikacích.

Literatura

[1] ROUSE, Margaret. SearchNetworking. *Ethernet* [online]. [cit. 2019-10-05]. Dostupné z URL: <https://searchnetworking.techtarget.com/definition/Ethernet>.

[2] TANG, Steve. TC Communications. *Advantages of Ethernet vs. SONET/SDH* [online]. [cit. 2019-10-10]. Dostupné z URL: <https://www.tccomm.com/Literature/Literature/Ethernet-Network-White-Papers/SONET-to-Ethernet-comparison>.

[3] ISO/IEC/IEEE International Standard. *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Standard for Ethernet AMENDMENT 10: Media access control parameters, physical layers, and management parameters for 200 Gb/s and 400 Gb/s operation,"in ISO/IEC/IEEE 8802-3:2017/Amd.10:2019(E) , vol., no., pp.1-376, 28 Feb. 2019.* DOI: 10.1109/IEEESTD.2019.8664714. Dostupné z URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8664714&isnumber=8664713>.

[4] Electronics Notes. *Ethernet IEEE 802.3 Frame Format / Structure: a summary of the Ethernet, IEEE 802.3, data frame format or structure and how Ethernet data frames are sent* [online]. [cit. 2019-10-11]. Dostupné z URL: <https://www.electronics-notes.com/articles/connectivity/ethernet-ieee-802-3/data-frames-structure-format.php>.

[5] LiveAction. *Interframe Gap* [online]. [cit. 2019-12-13]. Dostupné z URL: <https://www.liveaction.com/docs/glossary/ethernet-ieee-802-3/interframe-gap/>.

[6] ISO/IEC/IEEE International Standard *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Standard for Ethernet AMENDMENT 10: Media access control parameters, physical layers, and management parameters for 200 Gb/s and 400 Gb/s operation,"in ISO/IEC/IEEE 8802-3:2017/Amd.10:2019(E) , vol., no., pp.1-376, 28 Feb. 2019.* DOI: 10.1109/IEEESTD.2019.8664714. Dostupné z URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8664714&isnumber=8664713>.

[7] Sensagent dictionary. *Interframe gap* [online]. [cit. 2019-12-13]. Dostupné z URL: <http://dictionary.sensagent.com/interframe%20gap/en-en/>.

[8] MACKENZIE, Lewis. University of Glasgow - Schools - School of Computing Science. *802.3 AT 10Mbps* [online]. [cit. 2019-10-11]. Dostupné z URL: <http://www.dcs.gla.ac.uk/~lewis/networkpages/m04s03EthernetFrame.htm>.

[9] Howling Pixel. *Kontrolní součet* [online]. [cit. 2019-10-11]. Dostupné z URL: <https://howlingpixel.com/i-cs/Kontroln%C3%AD_sou%C4%8Det>.

[10] Arrow Electronics Components Online. *FPGA vs CPU vs GPU vs Microcontroller* [online]. [cit. 2019-11-26]. Dostupné z URL: <https://www.arrow.com/en/research-and-events/articles/fpga-vs-cpu-vs-gpu-vs-microcontroller>.

[11] MARTÍNEK, Tomáš. *Návrh číslicových systémů (INC): Technologie FPGA*. Vysoké učení technické v Brně, Fakulta informačních technologií.

[12] E. Nurvitadhi a ostatní. *Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC*. 2016 International Conference on Field-Programmable Technology (FPT), Xi'an, 2016, pp. 77-84. doi: 10.1109/FPT.2016.7929192 Dostupné z URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7929192&isnumber=7929174>.

[13] MOORE, Andrew. *FPGAs For Dummies®* [online]. 2nd Intel® Special Edition. Hoboken (New Jersey): Wiley, c2017 [cit. 2019-10-28]. ISBN 978-1-119-39047-3. Dostupné z URL: <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/misc/FPGAs_For_Dummies_eBook.pdf>.

[14] Liberouter / Cesnet TMC group. *Cards* [online]. [cit. 2019-10-18]. Dostupné z URL: <https://www.liberouter.org/technologies/cards/>.

[15] DANĚK, Martin. *Programovatelná hradlová pole - FPGA*. Automa. 2006, (2). Dostupné z URL: <https://automa.cz/cz/casopis-clanky/programovatelna-hradlova-pole-fpga-2006_02_30930_672/>.

[16] Intel Corporation. *FPGA Architecture: Altera white paper* [online]. červenec 2006 [cit. 2020-04-15]. Dostupné z URL: <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01003.pdf>.

[17] Intel FPGA. In: *Basics of Programmable Logic: FPGA Architecture* [online]. [cit. 2020-03-29]. Dostupné z URL: <https://youtu.be/jbOjWp4C3V4>.

[18] EE Times. *All about FPGAs* [online]. [cit. 2020-04-16]. Dostupné z URL: <https://www.eetimes.com/all-about-fpgas/#>.

[19] Xilinx, Inc. *Field Programmable Gate Array (FPGA)* [online]. [cit. 2019-10-18]. Dostupné z URL: <https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html>.

[20] Intel Corporation. *Intel® FPGAs and Programmable Devices: Industry Applications* [online]. [cit. 2019-10-17]. Dostupné z URL: <https://www.intel.com/content/www/us/en/products/programmable.html>.

[21] Xilinx, Inc. *UltraScale Architecture: Configurable Logic Block: User Guide* [online]. [cit. 2019-11-01]. Dostupné z URL: <https://www.xilinx.com/support/documentation/user_guides/ug574-ultrascale-clb.pdf>.

[22] Xilinx, Inc.: Community Forums. In: *Logic cell vs CLB in Virtex® Family FPGAs* [online]. [cit. 2020-04-17]. Dostupné z URL: <https://forums.xilinx.com/t5/Virtex-Family-FPGAs-Archived/logic-cell-vs-CLB/td-p/743699>.

[23] Intel Corporation. *Intel Cyclone 10 LP Device: Overview* [online]. [cit. 2020-04-25]. Dostupné z URL: <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-10/c10lp-51001.pdf>.

[24] Intel Corporation. *Intel Stratix 10 Logic Array Blocks and Adaptive Logic Modules: User Guide* [online]. [cit. 2019-11-19]. Dostupné z URL: <https://www.intel.com/content/www/us/en/programmable/documentation/wtw1441782332101.html>.

[25] Xilinx, Inc. *UltraScale+ FPGAs: Product Tables and Product Selection Guide* [online]. [cit. 2019-10-29]. Dostupné z URL: <https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf#VUSP>.

[26] Intel Corporation. *Intel® Stratix® 10 DX FPGAs: product table* [online]. [cit. 2019-10-29]. Dostupné z URL: <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/pt/stratix-10-dx-product-table.pdf>.

[27] Xilinx, Inc. *Virtex UltraScale Architecture and Product Data Sheet: Overview* [online]. [cit. 2019-11-01]. Dostupné z URL: <https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf>.

[28] Intel Corporation. *Intel Stratix 10 DX Device: Overview* [online]. [cit. 2019-11-01]. Dostupné z URL: <https://www.intel.com/content/www/us/en/programmable/documentation/sfx1556814887907.html>.

[29] Intel Corporation. *E-Tile Transceiver PHY User Guide* [online]. [cit. 2019-11-01]. Dostupné z URL: <https://www.intel.com/content/www/us/en/programmable/documentation/kqh1479167866037.html>.

[30] WatElectronics. *Basic FPGA Architecture and its Applications* [online]. [cit. 2020-04-27]. Dostupné z URL: <https://www.watelectronics.com/fpga-architecture-applications/>.

[31] CHEUNG, Peter. *Mastering Digital Design in Verilog with FPGAs* [online]. South Kensington Campus, London SW7 2AZ, UK, 2017. Dostupné z URL: <http://www.ee.ic.ac.uk/pcheung/teaching/MSc_Experiment/Lecture%201%20-

%20Introduction%20to%20Mastering%20Digital%20Design%20(x2).pdf>.
Prezentace. Imperial College London.

[32] Xilinx, Inc. *FPGA Design Flow Overview* [online]. [cit. 2020-04-27]. Dostupné z URL: <https://www.xilinx.com/support/documentation/sw_manuals/xilinx10/isehelp/ise_c_fpga_design_flow_overview.htm>.

[33] Electronic Design. *What's the Difference Between VHDL, Verilog, and SystemVerilog?* [online]. [cit. 2020-04-27]. Dostupné z URL: <https://www.electronicdesign.com/resources/whats-the-difference-between/article/21800239/whats-the-difference-between-vhdl-verilog-and-systemverilog>.

[34] Xilinx, Inc. *Design Tools: Vivado Design Suite* [online]. [cit. 2020-06-02]. Dostupné z URL: <https://www.xilinx.com/products/design-tools/vivado.html#documentation>.

[35] Intel Corporation. *Intel Quartus Prime Pro Edition User Guide: Design Compilation* [online]. [cit. 2020-04-30]. Dostupné z URL: <https://www.intel.com/content/www/us/en/programmable/documentation/zpr1513988353912.html>.

[36] CABAL, Jakub. *Školení pro nováčky VHDL+*. Brno, 2019.

[37] Intel Corporation. *AN 307: Intel® FPGA Design Flow for Xilinx* Users* [online]. [cit. 2020-04-30]. Dostupné z URL: <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/an/an307.pdf>.

[38] Intel Corporation. *Intel Quartus Prime Pro Edition User Guide: Programmer* [online]. [cit. 2020-04-30]. Dostupné z URL: <https://www.intel.com/content/www/us/en/programmable/documentation/ftt1513991830769.html>.

[39] The Massachusetts Institute of Technology. *Simulating with ModelSim* [online]. [cit. 2020-04-30]. Dostupné z URL: <http://web.mit.edu/6.111/www/labkit/simulation.shtml>.

[40] Xilinx, Inc. *Virtex UltraScale+: UltraScale Architecture DSP Slice: User Guide* [online]. [cit. 2019-11-11]. Dostupné z URL: <https://www.xilinx.com/support/documentation/user_guides/ug579-ultrascale-dsp.pdf>.

[41] BAJAJ, Ronak. *Exploiting DSP Block Capabilities in FPGA High Level Design Flows* [online]. 2016, 215 [cit. 2019-11-06]. Dostupné z URL: <https://warwick.ac.uk/fac/sci/eng/staff/saf/publications/ronak-phdthesis2016.pdf>.

[42] Intel Corporation. *Intel® Stratix® 10 Variable Precision DSP Blocks User Guide: UG-S10-DSP* [online]. 18.1. Dostupné z URL:

<https://www.intel.com/content/www/us/en/programmable/documentation/kly1436148709581.html>.

[43] Liberouter / Cesnet TMC group: Publications. *CRC based hashing in FPGA using DSP blocks* [online]. [cit. 2019-11-26]. Dostupné z URL: <https://www.liberouter.org/wp-content/uploads/2013/02/crc.pdf>.

[44] Stason.org. *What is jabber? Ethernet Errors and Troubleshooting* [online]. [cit. 2019-12-09]. Dostupné z URL: <https://stason.org/TULARC/networking/lans-ethernet/5-6-What-is-jabber-Ethernet-Errors-and-Troubleshooting.html>.

[45] Intel Corporation. *Stratix 10 DX FPGA Development Kit* [online]. [cit. 2020-06-05]. Dostupné z URL: <https://www.intel.com/content/www/us/en/programmable/products/boards_and_kits/dev-kits/altera/kit-s10-dx.html>.

Seznam symbolů, veličin a zkratek

ALM	přizpůsobivý logický modul – Adaptive Logic Module
ALUT	přizpůsobivá vyhledávací tabulka – Adaptive Look-Up Table
ASIC	integrovaný obvod pro konkrétní aplikaci – Application-Specific Integrated Circuit
BNN	binární neuronové sítě – Binarized Neural Networks
BRAM	bloková paměť s přímým přístupem – Block Random Access Memory
CLB	konfigurovatelné logické bloky – Configurable Logic Blocks
CRC	kontrolní součet – Cyclic Redundancy Check
DMA	přímý přístup do paměti – Direct Memory Access
DSP	digitální zpracování signálů – Digital Signal Processing
FCS	kontrolní součet rámce – Frame Check Sequence
FEC	opravné protichybové kódy – Forward Error Correction
FFT	rychlá Fourierova transformace – Fast Fourier Transform
FIR	konečná impulzní odezva – Finite Impulse Response
FPGA	programovatelná hradlová pole – Field Programmable Gate Array
IEEE	Institut pro elektrotechnické a elektronické inženýrství – Institute of Electrical and Electronics Engineers
IOB	vstupní/výstupní piny – Input/Output Blocks
IP	duševní vlastnictví – Intellectual Property
LAB	blok logických polí – Logic Array Block
LE	logický element – Logic Element
LUT	vyhledávací tabulka – Look-Up Table
MAC	řízení přístupu k médiu – Media Access Control
MACC	násobení s akumulací – Multiply And Accumulate
MLAB	paměťový blok logických polí – Memory Logic Array Block
MSB	nejvýznamnější bit – Most Significant Bit
NRZ	bez návratu k nule – Non Return to Zero
PAM	pulzně amplitudová modulace – Pulse Amplitude Modulation
PCIe	expresní propojení periferních zařízení – Peripheral Component Interconnect Express
qNaN	není číslo – Quiet Not A Number
RAM	paměť s přímým přístupem – Random Access Memory
SFD	oddělovač začátku rámce – Start of Frame Delimiter
SIMD	jednoduchá operace s vícero daty současně – Simple Instruction Multiple Data
VHDL	jazyk popisující hardware s velmi rychlými integrovanými obvody – Very-High Speed Integrated Circuit Hardware Description Language

A Obsah příloh

V této kapitole je popsán obsah přílohy. Jeho obsah se z větší části skládá ze zdrojových kódů navržených obvodů a kódů pro jejich simulace. Zdrojové kódy jsou napsané v jazyce VHDL 2008 a jsou umístěny ve dvou složkách, created a external. Ve složce created jsou vlastní vytvořené soubory pro tuto práci a ve složce external se nachází převzaté soubory v této práci využité. Vedle zdrojových kódů se v kořenovém adresáři nachází složka se soubory pro Latex, elektronická verze bakalářské práce a soubor readme.txt, ve kterém se nachází popis obsahu příloh.

- `/` ..... kořenový adresář
- `hdl` ..... zdrojové soubory
  - `created` ..... vytvořené zdrojové soubory
    - `dsp_comparator` ..... zdrojové soubory obecného komparátoru
      - `synth` ..... složka s nastavením pro překlad
      - `dsp_comparator.vhd`
      - `Modules.tcl` ..... pomocný soubor pro překladový systém
    - `dsp_comparator_intel` ..... zdrojové soubory komparátoru pro Stratix 10
      - `sim` ..... zdrojové kódy pro simulaci
      - `synth` ..... nastavení pro překlad
      - `dsp_comparator_stratix_10.vhd`
      - `dsp_comparator_stratix_10_atom.vhd`
      - `Modules.tcl` ..... pomocný soubor pro překladový systém
    - `dsp_counter` ..... zdrojové soubory obecného čítače
      - `sim` ..... zdrojové kódy pro simulaci
      - `synth` ..... složka s nastavením pro překlad
      - `dsp_counter.vhd`
      - `Modules.tcl` ..... pomocný soubor pro překladový systém
    - `dsp_counter_intel` ..... zdrojové soubory čítače pro Stratix 10
      - `sim` ..... zdrojové kódy pro simulaci
      - `synth` ..... nastavení pro překlad
      - `dsp_counter_stratix_10.vhd`
      - `dsp_counter_stratix_10_atom.vhd`
      - `Modules.tcl` ..... pomocný soubor pro překladový systém
  - `external` ..... využité zdrojové soubory
    - `build` ..... skripty pro překlad a automatické verifikace
    - `count` ..... zdrojové soubory čítače na UltraScale+
      - `sim` ..... zdrojové kódy pro simulaci
      - `synth` ..... nastavení pro překlad
      - `count_dsp.vhd`
      - `count48.vhd`
      - `Modules.tcl` ..... pomocný soubor pro překladový systém
- `latex` ..... Latex soubory
- `bakalarska_prace_xkondy00.pdf` ..... elektronická verze bakalářské práce
- `readme.txt` ..... soubor s popisem obsahu příloh