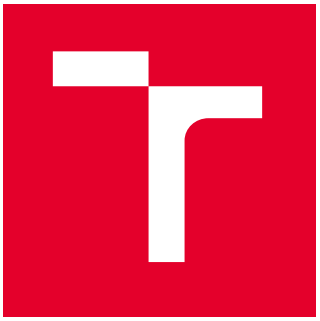


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

KOMUNIKACE UVNITŘ HARDWAROVĚ AKCELEROVANÉHO OBVODU

COMMUNICATION IN A HARDWARE ACCELERATED CIRCUIT

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Michal Rosa**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. David Smékal**

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Michal Rosa ***ID:*** 221012

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Komunikace uvnitř hardwarově akcelerovaného obvodu

POKYNY PRO VYPRACOVÁNÍ:

Téma práce je zaměřeno na komunikaci a přenosu dat uvnitř hardwarově akcelerovaného obvodu. Konkrétně komunikace mezi jednotlivými bloky vývojového testovacího FPGA kitu se SoC obvodem řady Zynq-7000.
Seznamte se s programovacím jazykem VHDL, FPGA obvody řady Zynq-7000 od společnosti Xilinx, architekturou ARM CORTEX a platformou Xilinx Vivado. Analyzujte možnostmi implementace uživatelského programu pro obsluhu blokových modulů FPGA kitu pomocí operačního systému Linux.
Úkolem bude zprovoznění jednotlivých periferií komunikující s FPGA obvodem Artix-7 (komunikace pomocí rozhraní USB 2.0, Micro SD, Ethernet, a další).
Cílem bakalářské práce je návrh a implementace uživatelského rozhraní obsluhující nabízené periferie na FPGA kitu pro přenos a zpracování dat pomocí zvolené technologie.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 26.5.2023

Vedoucí práce: Ing. David Smékal

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ROSA, Michal. *Komunikace uvnitř hardwarově akcelerovaného obvodu*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 64 s. Bakalářská práce. Vedoucí práce: Ing. David Smékal

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Michal Rosa
VUT ID autora:	221012
Typ práce:	Bakalářská práce
Akademický rok:	2022/23
Téma závěrečné práce:	Komunikace uvnitř hardwarově akcelerovaného obvodu

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné 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 závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné 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*

*Autor podepisuje pouze v tištěné verzi.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

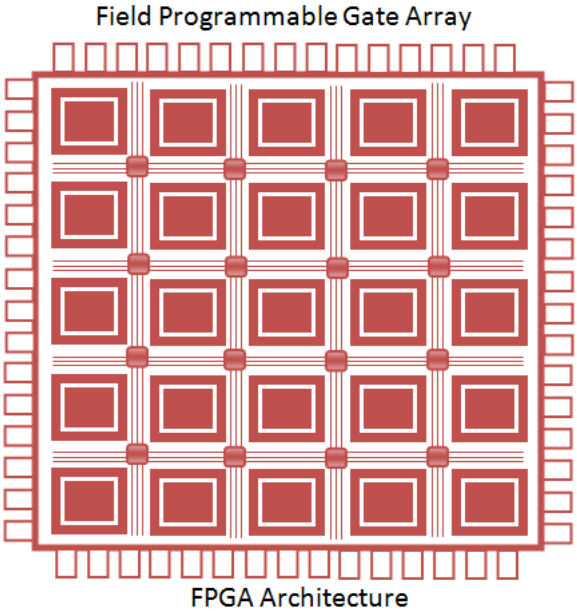
V dnešnej dobe rýchleho technologického pokroku je neustálym cieľom dosiahnuť výkonové a výpočtové kapacity, ktoré by zvládli záťaž moderných aplikácií a komplexných úloh. Jedným z prístupov, ktorý sa ukázal ako sľubný, je využitie hardvérovo zrýchlených zariadení. Tieto zariadenia kombinujú výhody paralelného spracovania a vysoko optimalizovaných hardvérových štruktúr na dosiahnutie výkonu, ktorý presahuje možnosti tradičných všeobecných procesorov.

Cieľom tejto práce je zoznámiť sa s problematikou hardvérovo zrýchlených zariadení, spôsobmi komunikáciami medzi ich jednotlivými blokmi. V praktickej časti sa zameriame na vytvorenie používateľského rozhrania na ovládanie komunikácie medzi nami zvolenými rozhraniami.

Práca sa skladá celkovo z troch častí. Prvá časť je zameraná na teóriu programovateľných hradlových polí, dosiek, ktoré tieto polia obsahujú a na **HDL**, čiže programovacie jazyky slúžiace na hardvérovú definíciu. V druhej časti sa zameriame na kryptografickú časť našej práce. Pozrieme sa na základné pojmy z oblasti kryptografie, rozdelenie kryptografických protokolov určených na šifrovanie a taktiež špeciálne na algoritmus **AES**, ktorého implementáciu použijeme aj v našej aplikácii. Poslednou časťou bude naša aplikácia, ktorá sprostredkuje komunikáciu medzi blokmi **FPGA** dosky.

1 FPGA

FPGA, celým názvom Field-Programmable Gate Arrays, v preklade programovateľné logické polia sú typom integrovaného obvodu. Na rozdiel od tradičných, aplikačne špecifických integrovaných obvodov (ASIC obvody), ktoré sú navrhované a produkované na presne určitú činnosť a nedajú sa upravovať, ponúkajú užívateľovi vysokú mieru flexibility a prispôsobenia. Z tohoto dôvodu sú veľmi obľúbené medzi vývojármi, keďže ich vlastnosti sú zárukou takmer neobmedzených možností, čo sa týka ich funkcionality a využiteľnosti na určité úlohy.[14] V tejto kapitole sa ďalej pozrieme na históriu, vlastnosti, využitie FPGA a taktiež na predpoklady do budúcna.



[14]

Obr. 1.1: FPGA architektúra

1.1 História FPGA

História programovateľných hradlových polí sa začala písať v 80. rokoch 20. storočia. Ich vznik je spojený so zavedením a používaním programovateľných logických zariadení (PLD) a programovateľnej pamäte určenej na čítanie (PROM), ktoré sa však museli napevno naprogramovať už počas výroby. FPGA umožňovali okrem programovania neskôr v procese aj programovanie samotným spotrebiteľom. Tento fakt umožnil konštruktérom pridávať funkcie do produktu, prípadne opravovať jeho chyby aj po nasadení výrobku na trh.

Prvé preprogramovateľné logické zariadenie bolo v roku 1984 vytvorené spoločnosťou *Altera*. Jeho názov bol **EP300** a obsahoval malé okienko, ktoré prepúšťalo ultrafialové svetlo na *EPROM*[1] bunky. Tieto bunky obsahovali konfiguráciu návrhu, čo znamenalo, že ju bolo možné podľa potreby zmeniť. Základy FPGA ďalej rozšírili *LuVerne R. Peterson* a *David W. Page*, ktorí v roku 1985 vytvorily patenty pre logické hradlá, bloky a programovateľné logické polia.

Koncom 80. rokov minulého storočia vznikla koncepcia FPGA vďaka experimentu Steve-a Casselman-a s finančnou podporou Navy Surface Warfare Center[2]. Išlo o návrh, ktorý spočíval vo vytvorení výpočtového zariadenia s viac ako 600 000 preprogramovateľnými hradlami. Tento experiment bol úspešný a v roku 1992 si dal tento svoj výtvor patentovať.

V polovici 90. rokov 20. storočia nastala, obrazne pomenované, explózia FPGA, ktorá spôsobila revolúciu v tomto odvetví, a tá sa tak rozšírila do celého sveta. Hoci v tom čase z tejto techonológie najviac profitovali telekomunikačné spoločnosti, FPGA sa rozšírili aj do automobilového priemyslu, ale aj do priemyselnej a spotrebiteľskej sféry, keďže svoje využitie našli pre všetky typy elektronických zariadení.[1]

V súčasnosti môžeme FPGA nájsť v akýchkoľvek zariadeniach. Okrem rôznych vedeckých sfér sa s nimi stretáme a používame ich v každodennom živote. Ide napríklad o tieto odvetvia:

- bezpečnostné aplikácie, vesmírne technológie, armáda
- letecký a automobilový priemysel
- spotrebná elektronika, signál prijímacie a vysielacie zariadenia
- dátové centrá, super počítače
- drôtová a bezdrôtová komunikácia

FPGA sa dotýkajú mnohých ďalších rôznych sfér a oblastí, ktorých zoznam sa stále rozširuje. Môže za to fakt, že elektronické zariadenia si čím ďalej, tým viac razia svoju cestu do viacerých oblastí obyčajných ľudí.[20]

Medzi najväčšie spoločnosti zaoberajúce sa tvorbou a distribúciou FPGA patria: **Xilinx Inc.**, **Altera Corporation**, **Lattice Semiconductor**, **Achronix**, **Efinix**, **Microsemi Corporation**, ... My budeme v našej práci pracovať s obvodom od prvej spomenutej firmy – Xilinx Inc.[1]

1.2 Vlastnosti FPGA

Jednou z kľúčových vlastností FPGA je ich vysoká rýchlosť. Keďže sú navrhované ako vysoko paralelné, umožňujú vykonávať mnoho operácií súčasne. To má za násle-

[1]Erasable Programmable Read-Only Memory – pamäť určená na čítanie, ktorá sa dá mazať ultrafialovým svetlom

[2]Námorné Centrum pre Povrchové Vojny USA

dok skrátenie celkového času. Tým pádom sú vhodné pre aplikácií, ktoré vyžadujú vysokú rýchlosť spracovania, ako napríklad spracúvanie digitálnych signálov, obrazov, videí a komunikácie. Vďaka možnosti optimalizácie pre špecifickú aplikáciu, pracujú oveľa rýchlejšie ako univerzálne procesory.

Ďalšou veľkou výhodou FPGA je ich už spomínaná flexibilita. Ich schopnosť takmer neobmedzenej konfigurácie na vykonávanie širokej škály funkcií, z nich robí vhodnú technológiu pre rôzne aplikácie. Táto vlasnosť tak umožňuje vývojárom vytvoriť vysoko špecializované riešenia na špecifické požiadavky. Okrem toho však flexibilita pomáha aj pri šetrení času, keďže vďaka možnosti optimalizácie pre špecifickú aplikáciu, pracujú oveľa rýchlejšie ako univerzálne procesory. V praktickej časti našej práce sa okrem iného zameriame na implementáciu vlastných šifrovacích algoritmov, ktorá je jednou z možných využití FPGA.

Rekonfigurovateľnosť je ďalším znakom FPGA. To znamená, že aj po naprogramovaní na určitú činnosť, umožňuje používateľovi preprogramovať ich alebo zmeniť ich dizajn, aby spĺňali špecifické požiadavky na plnenie iných funkcií bez nutnosti návrhu, výroby, alebo kúpy nového integrovaného obvodu.

Ďalšia vlastnosť, typická pre FPGA, sa týka spotreby elektrickej energie. Tá je v tomto prípade nízka. Existuje totiž možnosť FPGA navrhnúť tak, aby spotrebovávali menej energie, ako klasické digitálne obvody. To z nich robí obľúbenú voľbu pre prenosné zariadenia a zariadenia napájané elektrickou energiou z batérie alebo iného externého zdroja.

V neposlednom rade FPGA oplývajú vysokou komplexnosťou. To znamená, že ich môžeme navrhovať s vysokým počtom hradiel a vysokou úrovňou logickej zložitosti. Vďaka tejto skutočnosti je možné dizajnovať a implementovať veľmi zložité obvody.

Poslednou z kľúčových vlastností je nestála (nevolatilná) pamäť. Niektoré FPGA majú tento typ pamäti, ktorá umožňuje zachovať naprogramovanú funkciu aj po vypnutí napájania obvodu. Z tohoto dôvodu sú ideálne pre aplikácie, ktoré si vyžadujú cyklus vypnutia a zapnutia.

Vďaka týmto charakteristikám sú FPGA obľúbenou voľbou pre širokú škálu aplikácií vrátane už spomínaných sfér priemyslu, obrany, dátových centier, ... Okrem toho však v poslednej dobe nachádzajú uplatnenie aj v iných technológiach, ako napríklad umelá inteligencia, strojové učenie a IoT[3].[5] [2]

FPGA samozrejme oplývajú viacerými ďalšími vlastnosťami, ktoré ich od iných typov obvodov odlišujú. My sme sa zamerali len na tie hlavné z nich. Aj podľa nich vidíme, že FPGA sú v mnohých ohľadoch výhodné.

Okrem charakteristík pozitívnych má FPGA aj negatívne vlastnosti, ktoré ich používanie sťažujú a sú pre ich používateľov nevýhodné. Jednou z nich je ich cena.

[3]Internet of Things – internet vecí

Keďže sú veľmi prispôsobiteľné a ich použitie je rozšírené v celej škále aplikácií a odvetví, sú náklady na ich návrh a výrobu často drahšie ako tradičné procesory alebo iné typy integrovaných obvodov.[5] [2]

Ďalšou nevýhodou, ktorou oplývajú je ich zložitosť. Nakoľko môžu obsahovať nespočetný počet hradiel, ich návrh a programovanie môže byť zložitý a časovo náročný proces, ktorý vyžaduje špecializované nástroje a znalosti. Okrem toho môže nastať aj situácia, kedy sa FPGA ťažko ladia a testujú, čo môže mať za následok náročnejšiu identifikáciu a opravu chýb.[5]

1.3 Budúcnosť FPGA

Keďže sa technológie každým dňom posúvajú vpred, rovnaký trend naberajú aj programovateľné hradlové polia. Ich dopyt stále rastie a existujú predpoklady na to, že potreba ich využitia bude v budúcnosti väčšia a väčšia.

Tu je pár potenciálnych trendov , ktoré by v budúcnosti mohli ovplyvniť technológiu FPGA a smerov, ktorými by sa mohli uberať:

- Očakávaný nárast dopytu od širokej základne zákazníkov – S nárastom aplikácií, ktoré pre svoju činnosť vyžadujú špecializovaný hardvér sa FPGA môžu, vďaka svojej rekonfigurovateľnosti a flexibilite, stať jednou z najlepších možností. Táto hypotéza sa zdá byť veľmi pravdepodobná, nakoľko si stále viac a viac priemyselných odvetví osvojuje tento spôsob implementácie svojich aplikácií.[13]
- Rastúci význam kybernetickej bezpečnosti – FPGA sa v dnešných dňoch využívajú v bezpečnostných systémoch ako poskytovateľ ochrany na hardvérovej úrovni proti rôznym typom kybernetických útokov. Keďže nároky na ochranu a obavy o bezpečnosť každým dňom rastú, dá sa očakávať, že práve logické hradlá budú mať v budúcnosti ešte väčší význam. Tejto téme sa ešte budeme venovať v samostatnej podkapitole.[13]
- Pokrok v oblasti hardvéru a softvéru – Dá sa očakávať, že FPGA budú v budúcnosti profitovať z technologického pokroku. Spoločnosti zaoberajúce návrhom a produkciou takmer určite dokážu využiť vylepšení v oblasti hardvéru alebo aj softvéru a výhod s nimi súvisiacimi. Je tak celkom možné, že sa v budúcnosti dočkáme ešte rýchlejších, efektívnejších a výkonnejších logických hradiel, ako doteraz.[13]

V závere, na základe vyššie spomenutých hypotéz a predpovedí, môžme povedať, že sa celkovo budúcnosť javí ako svetlá.

1.4 FPGA v oblasti kybernetickej bezpečnosti

Ako sme už spomenuli, technológia programovateľných logických polí je vhodná na využitie v oblasti kybernetickej bezpečnosti. Táto problematika bude aj hlavnou témou našej praktickej časti.

Tu je niekoľko prípadov kedy FPGA nachádzajú svoje uplatnenie v oblasti kybernetickej bezpečnosti:

- Bezpečnosť komunikačných systémov: Jednou z hlavných výhod FPGA v oblasti kybernetickej bezpečnosti, je ich schopnosť urýchliť zložité algoritmy používané pri šifrovaní a dešifrovaní. Konkrétny príklad ponúka algoritmus AES[4], ktorý je široko používaný na zabezpečenie komunikačných systémov, no môže byť výpočtovo náročný. FPGA je teda možné naprogramovať tak, aby šifrovanie a dešifrovanie tohto algoritmu vykonávali rýchlejšie ako tradičné procesory. To z nich robí ideálnu voľbu pre implementáciu bezpečných komunikačných systémov.[6]
- Zabezpečenie siete: Okrem šifrovania a dešifrovania sa programovateľné logické polia dajú využiť aj vo sfére zabezpečení sietí, napríklad na funkciu detekcie a prevencie narušenia. Okrem toho môžu slúžiť aj ako firewally. FPGA taktiež možno naprogramovať na vykonávanie DPI[5] s cieľom identifikovať hrozby alebo anomálie. Keďže sa táto hĺbková kontrola pomocou FPGA vykonáva v „káblovej rýchlosti“, je možné zisťovať hrozby a reagovať na ne v reálnom čase, čím sa výrazne dokáže znížiť riziko bezpečnostného incidentu. Logické polia sú teda kritickou súčasťou systémov sieťovej bezpečnosti.[6]
- Odhaľovanie kybernetickej kriminalita: Ďalším prípadom kedy sa v kybernetickej bezpečnosti dajú FPGA uplatniť, je vyšetrovanie kybernetickej kriminality, napríklad pomocou kybernetickej forenznej analýzy. Tá zahŕňa zbieranie a analýzu digitálnych údajov na vyšetrovanie počítačovej kriminality. FPGA taktiež možno použiť na analýzu sieťovej prevádzky s cieľom odhaliť a izolovať kybernetické útoky. Analýza pomocou FPGA sa dá aplikovať aj pri pevných diskoch a dátach na nich uložených, s cieľom identifikovať dôkazy o kybernetickej kriminalite. Vďaka spracúvaniu údajov v reálnom čase tak logické polia môžu pomôcť vyšetrovateľom rýchlo identifikovať a reagovať na kybernetické hrozby.[6]
- Zabezpečenie cloudových systémov: Vlastnosti technológie FPGA možno uplatniť aj pri problematike zabezpečenia cloudových systémov a služieb. Pri tejto problematike sa dajú využiť už vyššie spomenuté znaky a vlastnosti. Ide napríklad o urýchlenie šifrovania a dešifrovania v cloudových aplikáciach. Vďaka

[4] Advanced Encryption Standard

[5] Deep packet inspection – hĺbková kontrola paketov

tomu je tak zaistený rýchlejší a bezpečnejší prenos dát medzi cloudovými službami. Táto skutočnosť je využívaná najmä vo sférach bankovníctva, e-shopov a iných aplikáciach, ktoré vyžadujú vysokú úroveň bezpečnosti. Taktiež je možné použiť ich na detekciu a prevenciu narušení v hardvérovej časti cloudových systémov. To zahŕňa monitorovanie sieťovej prevádzky s cieľom nájsť známky neoprávneného prístupu alebo škodlivej činnosti a taktiež blokovanie alebo zmierňovanie týchto hrozieb v reálnom čase. Okrem toho FPGA nachádzajú uplatnenie aj v iných súčastiach zabezpečenia cloudov, ako sú firewally, VPN[6] alebo ochrana pred DDoS[7] útokmi.[6]

- Detekcia malvéru: Logické programovateľné polia možno taktiež použiť na urýchlenie procesu detekcie malvéru[8], ktorá zahŕňa skenovanie súborov, sieťovej prevádzky a iných zdrojov údajov, a následné zabránenie ich šíreniu. To sa docieli vykonávaním analýzy týchto údajov v reálnom čase, s cieľom nájsť signatúry alebo vzory škodlivej činnosti. Niektoré systémy na detekciu malvéru však nemusia stíhať tempu moderných malvérových útokov. Z tohoto dôvodu sa na túto činnosť začali využívať FPGA. Prvým prípadom je poskytnutie hardvérovej akcelerácie detekcie malvéru. FPGA totiž možno naprogramovať tak, aby paralelne vykonávali porovnávanie vzorov a iné úlohy spracovania, čo umožňuje oveľa rýchlejšie odhaľovanie vzorových znakov malvéru. Ďalšou možnosťou využitia logických polí na detekciu škodlivého softvéru je implementácia systému založeného na signatúrach. Na FPGA sa naprogramuje databáza obsahujúca známe signatúry rôznych typov malvéru. Pri skenovaní údajov sú tieto údaje porovnávané so signatúrami v databáze. Ak sa medzi nimi nájde zhoda, systém môže zdroj údajov zablokovať alebo prípadne umiestniť do karantény. FPGA možno použiť aj v detekčných systémoch založených na strojovom učení. Tento princíp stojí na schopnosti logických polí paralelne spracúvať veľké množstvo údajov a následné efektívnejšie a rýchlejšie trénovanie modelov strojového učenia na detekciu malvéru.[6]
- Kryptografia: FPGA sa vďaka svojím vlastnostiam dokážu uplatniť aj v oblasti kryptografie. Vďaka svojej vysokej úrovni paralelnosti dokážu vykonávať viacero kryptografických operácií naraz, a teda oveľa rýchlejšie ako univerzálne CPU[9]. FPGA je taktiež možné ľahko prekonfigurovať na vykonávanie rôznych kryptografivkých algoritmov. Vďaka svojej flexibilite sa dokážu rýchlo prispôsobiť novým bezpečnostným štandardom alebo zmenám v bezpečnostnom prostredí. Ďalším prípadom kedy je možno uplatniť technológiu FPGA je ochrana

[6]Virtual Personal Network – virtuálne privátne siete
[7]Distributed Denial of Service – distribuovaný
[8]škodlivý softvér
[9]central processing unit – hlavný procesor počítača

pred útokmi postranným kanálom. Implementáciou určitých kryptografických algoritmov sa dá docieliť zníženie rizika tohoto typu útoku. FPGA sú vhodné na plnenie kryptografických úloh, aj vďaka tomu, že oproti klasickým, univerzálnym CPU spotrebúvajú menej elektrickej energie. Sú teda vhodnou voľbou pre zariadenia, ktoré vyžadujú nízku spotrebu.[6]

1.5 FPGA board

Ako sme už v predchádzajúcich kapitolách spomenuli, v prípade samotného FPGA ide o integrovaný obvod, ktorý podobne ako iné typy mikroprocesorov je možné naprogramovať na vykonávanie špecifických úloh.

Na druhej strane, FPGA board alebo FPGA doska je fyzická platforma, ktorá obsahuje čip a poskytuje ďalšie komponenty, a periférie ako:

- Pamäť – dosky FPGA často obsahujú rôzne typy pamäte, ako RAM, Flash alebo iné, ktoré možno použiť na ukladanie údajov a inštrukcií
- Vstupné a výstupné rozhrania – dosky FPGA poskytujú rôzne vstupné a výstupné zariadenia, ktoré možno pripojiť k iným zariadeniam, ako napríklad USB, Ethernet, HDMI alebo piny GPIO[10]
- Napájanie – dosky taktiež obsahujú napájacie zdroje, ktoré poskytujú potrebné napätie a prúd na prevádzku FPGA a iných komponentov dosky
- Hodiny – posledným komponentom, ktorý FPGA dosky obsahujú je jeden alebo viacero zdrojov hodín, ktoré sa môžu použiť na synchronizáciu operácií digitálnych obvodov.

[5] Stručne povedané, zatiaľ čo FPGA je programovateľný integrovaný obvod, doska je fyzická hardvérová platforma, obsahujúca čip a ďalšie komponenty potrebné na vývoj a testovanie digitálnych obvodov s použitím FPGA. Tiež býva zvyčajne navrhnutá tak, aby bola user-friendly [11] a ľahko použiteľná, so štandardizovanými rozhraniami a konektormi, ktoré umožňujú ich pripojenie k iným zariadeniam, ako napríklad počítače alebo iné FPGA dosky. Na našu prácu použijeme dosku **Nexys A7-100T** od spoločnosti **Digilent**.

1.6 Implementácia VHDL kódu na FPGA dosku

Ako sme už spomenuli, FPGA doska vznikne spojením viacerých hradiel a logických polí, s ktorými sme sa už stretli v predchádzajúcich kapitolách teoretickej časti. Okrem toho obsahujú rôzne iné vstupné a výstupné periférie, ako napríklad USB

[10] General-purpose input/output
[11] užívateľsky prívetivý

konektor, VGA konektor, RGB žiarovky, ... Na to, aby doska vykonávala funkciu, na ktorú ju chceme použiť, musíme na ňu implementovať kód nejakého jazyka HDL, v našom prípade to je VHDL. V tejto kapitole sa pozrieme na celý tento proces.

1. Vytvorenie nového projektu: Na implementáciu kódu jazyka VHDL na FPGA dosku sú vytvorené konkrétne vývojové prostredia, v ktorých sa celý tento proces odohrá. Medzi najznámejšie vývojové prostredia pre VHDL patria *ModelSim* a *Vivado Simulator*. My sa v našej práci stretneme s druhým menovať, ktorému sa budeme viac venovať v samostatnej kapitole. Pri vytváraní projektu je dôležitým krokom výber správneho typu obvodu, na ktorý budeme kód implementovať. V opačnom prípade by implementácia mohla zlyhať.
2. Tvorba VHDL kódu:Prvým krokom je napísanie samotného kódu v jazyku VHDL, ktorý, ako sme už spomenuli, slúži na popis hardvéru, a používa sa na popis digitálnych obvodov. Tento kód môžeme písať buď v klasickom textovom editori (Notepad) alebo priamo v, nami vybranom, IDE [12] programe. V našom prípade to je Vivado. Tento kód musí obsahovať všetky entity, moduly, architektúry a procesy, aby sme správne definovali funkcie, ktoré má potom obvod vykonať. Ide o najťažšiu časť celého postupu, keďže je najviac závislá od schopností vývojára. Ak v písaní kódu spravíme nejakú chybu a s touto chybou ho nahráme na obvod, môže to mať za následok jeho poruchu. Na kontrolu slúži simulácia kódu, ktorá je ďalšou časťou v procese implementácie.
3. Tvorba súboru s väzbami: Následujúcim krokom je vytvorenie súboru s väzbami – *constraint file*; ten obsahuje informácie o väzbách na nami programovaný logický obvod. Väzby môžu byť časové – špecifikujú požiadavky na časovanie návrhu, ako napríklad perióda hodín, rôzne vstupné a výstupné oneskorenia, ...; smerovacie – tieto väzby opisujú požiadavky na smerovanie návrhu, ako napríklad oneskorenie, počet smerovacích vrstiev alebo minimálny a maximálny fanout a väzby umiestnenia – ide asi o najdôležitejší typ väzby, keďže odkazuje priamo na fyzické umiestnenie prvkov obvodu, ktoré vo svojej aplikácii využívame.
4. Simulácia kódu: Ako ďalší krok pri implementácii prichádza simulácia. Tá zahŕňa vytvorenie testovacích zariadení – test bench, ktoré preveria rôzne časti návrh a overia či výstupy zodpovedajú očakávaným výsledkom. Ide teda o akýsi typ kontroly, ktorá má za úlohu zistiť správnosť kódu, objaviť v ňom chyby, upozorniť na ne a zamedziť tak možným ďalším nepríjemnostiam, ako už spomínané poškodenie celého obvodu. Simuláciu opäť vykoná Vivado.
5. Syntéza: Po overení správnosti VHDL kódu nasleduje ďalší krok a to syntéza. Ide o proces pri ktorom nastáva preklad kódu do netlistu na úrovni hradiel. Po

[12] Integrated Development Enviroment

preklade tento kód môžme implementovať priamo na FPGA dosku. Po syntéze vo Vivade sa vygeneruje správa, ktorá obsahuje rôzne užitočné informácie a metriky, ako napríklad veľkosť návrhu.

6. Implementácia: Po bez-chybnej syntéze už môžeme rovno prejsť k samotnej implementácii. Tá sa skladá z mapovania netlistu na úrovni hradiel, nami zvoleného konkrétneho FPGA obvodu a jeho konfigurácie na implementáciu. V tomto kroku prichádza k využitiu súboru s väzbami, ktorý obsahuje odkazy na konkrétne hradlá, na ktoré má byť návrh implementovaný. Výstupom tohto kroku je bitstream - binárny súbor, obsahujúci konfiguračné údaje pre FPGA.
7. Naprogramovanie FPGA: Po vygenerovaní bitstreamu nastáva posledný krok, a to naprogramovanie FPGA. Táto fáza zahŕňa načítanie vygenerovaného bitstreamu do obvodu a konfiguráciu logiky obvodu tak, aby sa správal podľa „inštrukcií“, ktoré sme napísali do nášho kódu.

Proces implementácie VHDL na FPGA dosku teda pozostáva z niekoľkých krokov, od návrhu logiky, až po naprogramovanie tejto logiky na konkrétny obvod. S veľkou časťou nám vedia pomôcť nástroje na to určené. Pri správnom naplánovaní a realizácii tak môžeme vytvoriť vysoko výkonný a spoľahlivú aplikáciu pre náš obvod.

2 Nexys A7-100T

Ako sme už v predchádzajúcej kapitole spomenuli, FPGA dosky sú výkonné nástroje na návrh digitálnych obvodov a ich prototypovanie. Na našu prácu sme si vybrali **Nexys A7-100T** od spoločnosti **Digilent**, ktorá je navrhnutá tak, aby poskytovala platformu pre širokú škálu aplikácií, od vzdelávacích projektov až po priemyselné prototypy. Táto doska je vybavená čipom **Xilinx Artix-7** . Ten poskytuje dostatok logických zdrojov a vysoko-rýchlostné pripojenie, vďaka čomu je vhodný pre širokú škálu aplikácií. V následujúcej kapitole sa zameriame práve na vlastnosti nami vybranej dosky.

2.1 Vlastnosti

Čo sa týka vlastností, môžeme si ich rozdeliť na dva typy, a to vlastnosti čipu, ktoré majú na starosti logické operácie, a vlastnosti dosky, ktoré potom obsahujú rôzne vstupné a výstupné zariadenia, slúžiace či už na jej ovládanie alebo na vykonanie konkrétnych aplikácií. Charakteristiky čipu sú následovné:

- Logické pláty: Pri popisovaní čipu začneme pri počte logických plátov. Ide o časti väčšieho obvodu, ktoré vykonávajú špecifickú funkciu. Ich úlohou je teda rozdeliť zložitý obvod na menšie časti. Náš čip ich obsahuje 15 850.
- Pamäť RAM: Čo sa týka pamäte RAM, XC7A100T obsahuje 4860kB pamäte, ktorá slúži na ukladanie údajov a implementáciu pamäťových štruktúr. Využiť ju môžeme napríklad ako vyrovnávaciu pamäť na dáta, spracovanie obrazu, ...
- DDR2 pamäť: Okrem RAM pamäte, tento čip obsahuje aj DDR2 pamäť o veľkosti 128MB. Táto pamäť pracuje dvojnásobnou rýchlosťou oproti pamäti DDR, čo znamená, že za určitý čas dokáže preniesť dvojnásobné množstvo dát. Je teda energeticky úspornejšia.
- Riadenie hodín: XC7A100T obsahuje 6 dlaždíc na riadenie hodín, ktoré možno použiť na generovanie a distribúciu hodín do rôznych častí dosky, ako aj do externých zariadení.
- DSP[1] pláty: XC7A100T obsahuje taktiež 240 DSP plátov. Tie slúžia na implementáciu vysoko výkonných algoritmov digitálneho spracovania signálu. Každý jeden plát obsahuje vyhradený násobič 18×18, 48- bitovú sčítačku a predsádku, ako aj kaskádový reťazec na implementáciu väčších násobičov. [2] [8]

Tento čip je navrhnutý najmä pre vysoko výkonné aplikácie s nízkou spotrebou energie a je vhodný pre širokú škálu priemyselných odvetví. Vďaka veľkému počtu

[1]Digital Signal proccesing

logických buniek, pamäti RAM a DSP plátom je vhodný na implementáciu zložitých digitálnych obvodov a algoritmov spracovania signálov.[2]

Čo sa týka samotnej dosky, vlastnosti ktorými oplýva sú:

- Funkcie FPGA: Interné hodiny, ktorých frekvencia presahuje 450MHz; XADC[2] – analógovo-digitálny prevodník na čipe; programovanie cez JTAG[3] a Flash pamäť
- Systémové funkcie: Programovacie obvody USB-Flash; 16MB QSPI[4] Flash pamät – typ nevolatilného pamäťového čipu, slúžiaceho na komunikáciu s inými zariadeniami; Napájanie z USB alebo akéhokoľvek zdroja s napätím 7V-15V konektor pre microSD kartu
- Systémové pripojenia: Konektor pre Ethernet s rýchlosťou 10/100MBps; most USB-UART[5] – zabezpečuje komunikáciu medzi počítačom a iným zariadením pomocou rozhrania UART; interakčné a senzorické zariadenia
- Interakčné a senzorické periférie: 3-osový akcelerometer – senzor, ktorý slúži k meraniu pohybu zariadenia, napríklad jeho náklon, otáčanie, ... Využiteľný je napríklad pri robotoch, autonómnych vozidlách alebo dronoch; PDM[6] mikrofón – mikrofón ktorý využíva digitálnu moduláciu signálu na prevod zvukových signálov na signály digitálne; PWM[7] zvukový výstup – ide o klasický audio výstup, ktorého princíp spočíva v periodickom striedaní logického stavu signálu v rôznych časových úsekoch; snímač teploty; 2 štvormiestne sedemsegmentové displeje – môžu byť nastavené na vypisovanie rôznych znakov a cifier; USB rozhranie pre používateľa na myš, klávesnicu a pamäťové karty; 16 switchov; 16 LED žiaroviek; 2 trojfarebné žiarovky v spektre RGB[8]; 12-bitový VGA[9] výstup pre obraz; rozširujúce konektory – konektory slúžiace k pripojeniu iných periférií a zariadení, a tak k rozšírení funkcionality FPGA dosky; 4 Pmod[10] konektory – ďalšie konektory, slúžiace k pripojeniu iných periférií a zariadení, no narozdiel od klasických rozširovacích, sú Pmod konektory štandardizované, Vďaka čomu sa minimalizuje rušenie a šum v signáloch; Pmod pre signály XADC;

[8]

[2] Xilinx Analog-to-Digital Converter
[3] Joint Test Action Group
[4] Quad Serial Peripheral Interface
[5] Universal Asynchronous Receiver/Transmitter
[6] Pulse Density Modulation
[7] Pulse width modulation
[8] Red Green Blue
[9] Video Graphic Array
[10] Peripheral Modules

2.2 Zaujimavé projekty

Ako sme už spomenuli, možnosti využitia programovateľných logických obvodov, sú vďaka ich vlastnostiam takmer neobmedzené. V tejto časti sa pozrieme na jedny z najzaujímavejších projektov a aplikácií, ktoré boli uskutočnené na práve nami používanej doske Nexys A7.

- **Implementácia detekcie srdcovej frenkvencie v reálnom čase** – V tomto projekte sa FPGA použil na implementáciu FIR filtra a algoritmu detektora maxima. Systém dokázal detekovať srdcovú frenkvenciu s 98-perecentnou prestnosťou.
- **Autonómna navigácia robotov pomocou FPGA a spracovania obrazu** – Tento projekt využíval dosku na vytvorenie autonómneho robota, ktorý sa dokázal sám navigovať pomocou techniky spracovania obrazu. Implementovaný bol Sobelov filter. Ide o typ operátora, ktorý slúži na detekciu hrán. Samotný robot bol riadený pomocou PID[11] regulátoru.
- **Rozpoznávanie tvárí v reálnom čase pomocou OpenCV** – Pri tvorbe tohto projektu bola využitá knižnica OpenCV a programovací jazyk Python. Systém dokázal rozpoznať tváre v toku videa a v reálnom čase ich porovnať s databázou známych tvárí a identifikovať tak určité osoby.

[11] proportional integral derviative regulator – proporcionálny, integračný a derivačný regulátor

3 HDL

Ako odpoveď na rastúcu zložitosť číslicových systémov sa do popredia začali dostávať nástroje na ich počítačový návrh a simuláciu. Ide o tzv. **EDA** (Electronic Design Aids) programy. HDL (Hardware Description Language) je skupina špeciálnych jazykov, využívaných v počítačovom spracovaní, ktoré slúžia k popisu číslicových systémov. Tieto jazyky požívame práve pri práci s číslicovými systémami v EDA. Dvomi najviac využívanými a najznámejšími z nich sú **VHDL** a **Verilog**. V našej práci sa budeme zaoberať prvým menovaným. V tejto kapitole budú prebrané vlastnosti HDL, ich história a prísľuby do budúcnosti.[18]

3.1 Vlastnosti HDL

Ako už bolo spomenuté, HDL je skupina jazykov používaných k popisu číslicových, no v niektorých prípadoch aj analógových systémov. V niektorých aspektoch sú podobné a taktiež sa podobne správajú ako klasické programovacie jazyky (C, C++, ...), no taktiež existujú aj rozdiely, na ktoré si pri práci s nimi musíme dávať pozor. Jazyk HDL obsahuje textový popis zložený z operátorov, výrazov, výrokov, vstupov a výstupov. Namiesto generovania, počítačom spustiteľného programu, kompilátor vytvára tzv. *gate map*. Ten je potom nahratý práve na programovateľný logický obvod, na ktorom je potom daný kód a jeho príkazy vykonaný. [18]

V našej práci, ako už vypovedá názov práce, sa budeme zaoberať programovaním digitálnych logických obvodov (viac kapitola ...).

3.2 Výhody HDL

Hlavnou výhodou jazyka je jeho rýchly design a lepšia verifikácia. Návrh kódu je postavený na hierarchickej logike, čiže „zhora nadol“. Táto skutočnosť znižuje časovú náročnosť, náklady a chyby návrhu. Ďalšou veľkou výhodou, vďaka komplexnosti designov, je možnosť ich jednoduchej kontroly, spravovania a overovania. HDL taktiež poskytuje informácie o časovaní a umožňuje opísať návrh na úrovni hradiel a registrov. Poslednou z veľkých výhod, je možnosť znovu-používania zdrojov z iných projektov.[18]

3.3 História HDL

Pôvod HDL jazykov sa datuje až do 60-ich rokov minulého storočia. Prvou dôležitou udalosťou v histórii HDL, bolo v roku 1971, zverejnenie knihy od C.Gordona Bella

a Allena Newella, „Computer Structures: Readings and Examples“. Tá obsahovala opis prvého, významnejšieho HDL jazyka. Dvojica vo svojom texte taktiež prišla s konceptom **RTL**[1], ktorý využila pri popise správania mikropočítača PDP-8. Jazyk sa ďalej rozšíril zavedením **PDP-16**[2] a knihy obsahujúcej jeho používanie. Po ňom nasledovali minimálne dve implementácie základného jazyka **ISP**[3], a to **ISPL** a **ISPS**. Druhý menovaný bol veľmi vhodný na opis vzťahov medzi vstupmi a výstupmi návrhu a boľ rýchlo prijatý obchodnými tímami spoločnosti DEC[4], ako aj mnohými americkými výskumnými sektormi, zaoberajúcimi sa práve touto problematikou, ktoré sa o svoje skúsenosti delili so spojencami z organizácie NATO. [17]

Ďalším významným krokom bolo vytvorenie jazyku **KARL**[5] na Kaiserslauternskej Univerzite v Nemecku. Išlo o samostatný projekt, nezávislý od ostatných predchádzajúcich pokusov. [17]

Koncom sedemdesiatych rokov 20. storočia sa populárnym stal dizajn využívajúci **PLD**[6], ktorý však bol primárne obmedzený na konečné automaty. Práve tento typ zariadení bol použitý pri vzniku prvého 32-bitového mini-počítaču Eclipse MV/8000. Vďaka tomu začal rýchlo rásť dopyt po jazykoch, ktoré by boli použiteľné pri práci s konečnými automatmi.[17]

V roku 1985 prišli s riešením spoločnosti *Gateway Design Automation* a *Intermetics*, keď predstavili jazyk **Verilog**, a najmä prvú dokončenú verziu **VHDL**, teda jazyka na opis hardvéru, ktorý bol vyvinutý na objednávku v programe VHSIC[7] Ministerstva obrany Spojených štátov amerických. Nás bude viac zaujímať práve ten druhý menovaný, ktoré mu bude venovaná celá samostatná kapitola.[17]

[1] systém založený na presunoch medzi registrami
[2] RTMs modul od spoločnosti DEC
[3] Instruction Set Processor
[4] Digital Equipment Corporation, založená v roku 1957
[5] Kaiserslautern Register Transfer Language
[6] PLD – programovateľné logické zariadenia
[7] program na výskum a vývoj vysokorýchlostných integrovaných obvodov pre ozbrojené sily USA

4 VHDL

Very-High-Speed Integrated Circuit Hardware Description Language, ako znie jeho celý názov, je programovací jazyk vysokej úrovne, ktorý sa používa pri automatizácii elektronického návrhu na opis, modelovanie a simuláciu digitálnych obvodov systémov. Tento jazyk vývojárom umožňuje opísať správanie systému, ako aj jeho štruktúru. Tento popis sa potom môže použiť na simuláciu správania systému, overenie jeho správnosti a syntézu do fyzickej implementácie. VHDL sa bežne používa pri návrhu digitálnych obvodov, ako sú mikroprocesory, programovateľné logické polia, pamäťové systémy, komunikačné systémy a riadiace systémy.[17] [9]

4.1 História VHDL

Jeho začiatky siahajú až do roku 1981, kedy chcelo Ministerstvo obrany Spojených štátov amerických zdokumentovať, ako v ich zariadeniach fungujú obvody ASIC[1], dodávané tretími stranami – VHDL teda spočiatku slúžil na sledovanie a dokumentáciu obvodov v mikroelektronických zariadeniach.[17] [9]

Prvá verzia bola vytvorená tak, aby zodpovedala štandardom IEEE[2], čo malo za následok začlenenie značného rozsahu dátových typov, vrátane logických, číselných, znakových, časových, textových polí a reťazcov, no až v ďalšej verzii bola zakomponovaná viac-hodnotová logika a taktiež *drive strength* spolu s neznámymi faktormi. To malo za následok vytvorenie normy IEEE 1164, ktorá definovala deväť-hodnotové logické typy spolu s ich vektorovými verziami. V roku 1993 bola zavedená norma IEEE 1076. Tá umožňovala použiť viac znakov z ISO-8859-1, z čoho pramenila konzistentnejšia syntax a flexibilita jazyka. Každým následujúcim prechodom na nový štandard sa rozšírila aj jeho funkčnosť, ako napríklad možnosť spracovania väčšieho množstva údajov, takže sa zariadenia mohli stať ešte sofistikovanejšími. V priebehu následujúceho desaťročia bolo vykonaných niekoľko menších zmien, ktoré pomohli zlepšiť funkčnosť jazyka VHDL a zariadení, na ktorých bol tento jazyk využívaný. [17] [9]

Následujúcou veľkou zmenou, bolo v roku 2006 schválenie verzie 3.0, Technickým Výborom pre VHDL, spoločnosti *Accellera*. Cieľom bolo zachovať jeho funkčnosť v súčasných a starších systémoch a zároveň ho rozšíriť do systémov nových. Nové rozšírenia uľahčili zápis a správu kódu VHDL, pričom kľúčovou zmenou bolo zakomponovanie podriadených štandardov do hlavného, ktorým bol 1076. Verzia 3.0

[1] integrovaný obvod navrhnutý a vyrábaný pre určitú špecifickú aplikáciu

[2] Inštitút pre elektrotechnické a elektronické inžinierstvo

rozšírila jazyk o nový súbor operátorov, bola používaná flexibilnejšia syntax a taktiež umožnila začlenenie *VHPI*[3]. Tieto zmeny dopomohli zvýšiť kvalitu syntetizácie kódu a flexibilitu testbenchov. [17] [9]

Poslednou schválenou verziou sa v roku 2008 stala 4.0, známa ako VHDL 2008. Tá dokázala vyriešiť viac ako 90 rôznych problémy objavených počas skúšobného obdobia predchádzajúcej verzie(WIKI VHDL 4) a obsahovala rozšírené generické typy. V súčastnosti vývoj VHDL stále pokračuje a v budúcnosti sa určite dočkáme ešte lepších verzií s či už novou alebo vylepšenou funkcionalitou. [17] [9]

4.2 Vlastnosti VHDL

Ako každý iný programovací jazyk, aj VHDL má svoje vlastnosti a funkcionality, ktoré ho charakterizujú a od ostatných rozlišujú. Hlavnou vlastnosťou jazyka VHDL, pokiaľ ide o využitie návrhu systému, je to, že umožňuje overiť a modelovať správanie základného systému ešte pred tým, ako nástroje na syntézu prevedú návrh na skutočný hardvér. To zamedzí jeho možnému poškodeniu, použitím chybného kódu. Ďalšou charakteristikou je, že projekty v ňom písané, sú prenosné, čo znamená, že existuje možnosť vygenerovať projekt pre jednu základňu prvkov a potom ho preniesť na inú, ako napríklad pri VLSI[4]. Taktiež dôležitou vlastnosťou je fakt, že VHDL je tzv. dataflow jazyk, teda že uvažuje súčasne o každom príkaze na vykonanie, narozdiel od procedurálne výpočtových jazykov, ako sú C, Assembler, BASIC, ktoré postupnosť príkazov vykonávajú sekvenčne po jednotlivých inštrukciách. Za ďalšiu pozitívnu vlastnosť jazyka VHDL a jeho projektov, sa dá určite považovať ich viac-účelovosť. To znamená, že už vytvorený projekt a jeho výpočtový blok je možné využiť v rôznych iných projektoch. Pri opätovnom použití taktiež existuje možnosť vykonania zmien v kóde, napríklad parametrov, veľkosti pamäte, zloženia blokov, štruktúry prepojenia a kapacity. [18]

Ďalšie významné vlastnosti VHDL:

- Podporuje rôzne metódy návrhu; okrem prístupu zhora nadol aj zdola nahor
- Poskytuje úzke prepojenie s nižšími úrovňami návrhu
- Umožňuje lepšiu správu návrhu a detailnú implementáciu
- Podporuje viacúrovňovú abstrakciu
- Umožňuje využitie všetkých nástrojov CAD

[18]

[3] procedurálne rozhranie pre VHDL do jazykov C/C++

[4] Very large-scale integration – proces tvorby integrovaného obvodu

4.3 VHDL verzus Verilog

Ako už bolo spomenuté, VHDL je len jedným z jazykov na popis hardvéru. Ďalšími najznámejšími sú Verilog a SystemVerilog (vylepšená verzia Verilogu), z ktorých každý ma svoj osobitný štýl, ktorým sa navzájom líšia. V tejto kapitoly sa zameriame práve na rozdiely medzi nimi, ale aj na ich spoločné znaky. [18]

Tab. 4.1: Základné rozdiely medzi VHDL a Verilog

VHDL	**Verilog**
Vychádza z jazyka Ada	Vychádza z jazyka C
Vhodný pre FPGA	Vhodný pre ASIC
Silno typovaný	Slabo typovaný
Komplexnejší	Menej komplexný
Nie je case sensitive	Case sensitive
Možnosť definovať dátové typy	Bez možnosti definovať dátové typy
Podporuje viacrozmerné polia	Viacrozmerné polia neexistujú
Umožňuje súčasné volanie procedúr	Procedúry volá jednu po druhej

[18]

Ako je z tabuľky 4.1 jasné, VHDL je založený na programovacom jazyku Ada, zatiaľ čo Verilog vychádza z jazyka C. Čo sa týka využitia, prvý menovaný sa používa pri programovateľných hradlových poliach (FPGA). Verilog je zasa vhodnejší na popis zákazníckych obvodov (ASIC). Čo sa týka typovania, VHDL je silno typovaný, čiže vyžaduje iba predom vymedzené dátové typy, no jeho konkurent je naopak slabo tipovaný – snaží sa operáciu vykonať za každú cenu a dochádza k pretypovaniu. VHDL je taktiež voči Verilogu komplexnejší a narozdiel od svojho konkurenta nerobí rozdiely medzi veľkými a malými písmenami. Čo sa týka volania a vykonávania procedúr, Verilog tieto činnosti vykonáva jednu po druhej. V prípade VHDL ide o tzv. simultánne volanie. Okrem iného sa rozlišujú aj v možnosti využitia viacrozmerných polí. Zatiaľ čo VHDL tento dátový typ podporuje, Verilog ho neobsahuje. Samozrejme, rozdielov existuje oveľa viac, no v našej práci sme sa zamerali len na tie hlavné.

Výpis 4.1: Príklad jazyku VHDL

```
reg1: process (rst, clk)
begin
    if
        rst = '1' then q_reg <= (others => '0');
        q_i <= (others => '0');
        elsif rising_edge(clk)
            then if s_l = '1' then q_i(0) <= q_i(7);

        loop1: for i in 6 downto 0
        loop q_i(i + 1) <= q_i(i);
        end loop loop1;
        q_reg <= y;
        else q_i <= q_reg;
        q_reg <= y;
    end if;
    end if;
end process reg1;
```

Výpis 4.2: Príklad jazyku Verilog

```
always @(posedge CLK or posedge RST)
begin
   if (RST) begin
      q_reg = 0;
  Q = 0;
  end else if (S_L) begin
      Q[7:0] = {Q[6:0],Q[7]};
 q_reg = Y;
  end else begin
      Q = q_reg;
  q_reg = Y;
  end
end
```

4.4 Syntax VHDL

VHDL tak ako každý iný jazyk, má svoju špecifickú syntax. V tejto kapitole sa zameriame práve na ňu. Keďže VHDL nie je tak známy, ako napríklad C, Java alebo iné

jazyky, v tejto kapitole sa bližšie pozrieme na jeho syntax. Syntax VHDL obsahuje niekoľko základných prvkov. Medzi ne patria napríklad kľúčové slová. Ide o slová, ktoré opisujú určitú časť kódu podľa jeho významu – slovo „library“ je kľúčovým slovom pre knižnicu, „entity“ pre entitu, „architecture“ pre architektúru, „signal“ pre signál, ... Ďalším základným prvkom sú identifikátory. Pomocou nich označujeme názvy entít a signálov. Niemenej dôležitým elementom sú doslovné typy. Ide o konštantné hodnoty v podobe čísel alebo textových reťazcov. Pri práci s VHDL sa taktiež stretávame s operátormi. Ide o znaky a slová, pomocou ktorých sú vykonávané rôzne matematické a logické operácie. Sú to napríklad „+“, „-“, „*“, „/“ alebo slová „and“ a „or“. Ďalším dôležitým prvkom sú oddeľovače. Ide o znaky „;“ a „:“. ; sa používa ako ukončovač každého príkazu podobne ako v jazyku C, : sa zasa používa pri popisoch entity, procesu, ... Posledným dôležitým prvkom sú atribúty. V tomto prípade ide o určité hodnoty, atribúty priradené k určitým objektom, ako je napríklad oneskorenie signálu.[18]

4.4.1 Entita

Entita je jedna z hlavných návrhových jednotiek jazyka VHDL. Ide o objekt, ktorý slúži ako predpis na vytvorenie inštancie modulu, pričom každý návrh v jazyku VHDL musí obsahovať aspoň jednu entitu. Môže reprezentovať jednoduché logické hradlo, obvod alebo rovno celý veľký systém. Obsahuje popis vstupov a výstupov, čiže rozhrania objektu a jeho správanie. Môže taktiež obsahovať identifikátory, pomocou ktorých je možné nastaviť rôzne parametre danej entity.[18]

Výpis 4.3: Príklad entity

```
1 entity my_entity is
2 port(
3  in_port1 : in std_logic;
4  in_port2 : in std_logic;
5
6  out_port1 : out std_logic;
7  out_port2 : out std_logic);
8 end my_entity;}
```

Bližšie si entitu popíšeme na jednoduchom príklade 4.3. Z kódu je jasné, že entita sa vytvára za použitia kľúčového slova „entity“, po ktorom nasleduje jej názov a ďalšie kľúčové slovo „is“. V ďalšej časti už vidíme určitý popis rozhraní objektu, ktorý začína slovíčkom „port“. Za ním už nasledujú rozhrania „in_port1“, „in_port2“, ..., ktorých režim prenosu dát je popísaný po dvojbodke. V tomto prípade ide o prenos dovnútra – **in**; a prenos dát von – **out**. Okrem týchto dvoch

smerov existujú ďalšie 3 smery prenosu. Ide o **buffer** – dáta pomocou portu z entity len vystupujú, no narozdiel od módu out, entita dokáže výstupné dáta čítať. Tento typ sa používa napríklad pri výstupoch čítačov. Ďalšou možnosťou smeru prenosu dát je **inout** režim. Dáta buď môžu do tohto portu vstupovať alebo z neho môžu vystupovať. Ide teda o obojsmerný dátový tok. Posledným typom je **linkage**, ktorý sa však v bežnej práco takmer nepoužíva. Pri deklarácii portov entity si môžme všimnúť typ „std_logic“. Ide o výpočtový typ, ktorý môže dosahovať až 9 hodnôt. Okrem klasických 0 a 1 sú to hodnoty „U“, „X“, „Z“, „W“, „L“, „H“, „-“.

Ruku v ruke s entitou prichádza ďalšia dôležitá súčasť každého VHDL kódu, a to **architektúra**.

4.4.2 Architektúra

Architektúra je sekundárna návrhová jednotka. Je závislá od entity, ktorej je priradená a definuje jej vnútrajšok, čiže jej správanie a funkcie. Ako už sme spomenuli, každá entita musí mať aspoň jednu architektúru. Ak však nastane prípad, kedy má jedna entita väčší počet architektúr, každá z nich musí mať rozdielny názov. [18]

Bližšie si architektúru popíšeme na konkrétnom príklade nižšie

Výpis 4.4: Príklad architektúry

```
1  architecture Behavioral of blinky is
2      type state is (LED_On, LED_Off);
3      signal s: state;
4      signal counter : unsigned(24 downto 0);
5      signal clk_counter: natural range 0 to 50000000 := 0;
6      signal counter2: natural range 0 to 9 :=0;
7  begin
8       blinker0: process (clk_i)
9      begin
10
11         if rising_edge(clk_i) then
12             case (s) is
13                 when LED_On =>
14                     led_0 <= '1';
15                 when LED_Off =>
16                     led_0<= '0';
17             end case;
```

V tomto prípade ide o architektúru entity blinky, ktorá popisuje jej chovanie – kľúčové slovo „Behavioral“. Okrem tohto ešte existujú architektúry popisujúce tok

dát, štruktúru entity a jej RTL popis.

Ďalšie kľúčové slovo, s ktorým sa stretáme v tomto príklade, je „type“ nasledované slovom „state“. Prvé sa vo VHDL používa ako dátový typ, ktorý definuje súbor hodnôt, ktoré môžu signály alebo hodnoty nadobúdať. Druhé menované sa zasa vzťahuje na aktuálnu hodnotu alebo stav systému. Dokopy sa tieto dve slová môžu vzťahovať na používateľom definovaný typ, predstavujúci rôzne stavy systému. V tomto prípade ide o stav LED diódy.

Ďalej vidíme definované 4 signály. Teórii signálom sa budeme viac venovať v samostatnej podkapitole.

Na následujúcom riadku máme možnosť vidieť kľúčové slovo „begin“. To má na starosti, ako nám jeho preklad napovedá, za úlohu spustiť nejaký proces, ktorý ma daná architektúra vykonať. V ďalších riadkoch už vidíme konkrétne inštrukcie. Tými sa budeme viac zaoberať na konkrétnych príkladoch v praktickej časti našej práce.

4.4.3 Konfigurácia

Poslednou návrhovou jednotkou v jazyku VHDL je konfigurácia. Definuje sa pomocou kľúčového slova „configuration“ a je nepovinná. Jednou z funkcií ktorou konfigurácia, alebo aj konfiguračná deklarácia je určiť, ktorá architektúra sa má použiť, v prípade, že ich má entita viac. Ako už bolo spomenuté, kód konfiguráciu nemusí obsahovať a v tom prípade sa pre entitu použije posledná deklarovaná architektúra. [18]

Ďalšie prípady, v ktorých sa dá konfigurácia použiť:

- Riadené testovanie – ide o ďalšie možné využitie konfigurácií. Jeho význam spočíva vo výbere rôznych architektúr na testovanie. Pri simulácií je možné tie isté porty použiť pri testoch viacerých architektúr. Pomocou konfigurácie potom môžme určiť, ktoré architektúry sa majú testovať.
- Používanie funkčných modelov zberníc (BFM) – Pri rozbiehaní väčších projektov sa občas používa technika výmeny RTL kódu, za BFM. Pri písaní skutočného kódu tak môže nastať situácia, kedy práve pomocou konfigurácií nastane spomínaná výmena
- Testovanie neobyklého správania – Táto technika sa používa pri hľadaní neobvyklých krajných prípadov v návrhoch. Docielime jej tak, že k entite priradíme architektúru, ktorá je schválne navrhnutá tak, aby sa správala chybne.
- Zvýšenie rýchlosti simulácie – Ide o veľmi praktickú techniku, vhodnú pri veľkom počte signálov. Čím väčší počet signálov sa ma simulovať, tým bude, logicky, celková simulácia trvať dlhšie. Skrátenie času simulácie sa dá dosiahnuť výmenou veľkých architektúr za menšie.

[18]

Príklad konfigurácie:

Výpis 4.5: Príklad konfigurácie

```
1 configuration Config of my_entity is
2   for Behavioral
3     for Inst : Component
4       use entity my_entity_test(Behavioral_test);
5     end for;
6   end for;
7 end Config;
```

V príklade — vidíme konfiguráciu **Config** základnej entity *my_entity*. V ďalších riadkoch kódu konfigurácie už vidíme určenie konkrétnej testovacej architektúry *Behavioral_test* pre testovanú entitu *my_entity_test*.

5 Komunikácia vrámci obvodu

V tejto kapitole sa zameriame na možnosti komunikácie testovacieho FPGA kitu na obvode Artix7. Taktiež sa pozrieme a zanalyzujeme si možnosti implementácie užívateľského programu slúžiaceho na komunikáciu medzi týmito rozhraniami.

Komunikácia medzi určitými blokmi vývojového testovacieho kitu s obvodom vždy závisí na konkrétnom spôsobe implementácie. Vo všeobecnosti však platí, že všetky bloky komunikujú prostredníctvom interných komunikačných kanálov, definovaných vrámci FPGA obvodu. Komunikácia teda môže byť realizovaná pomocou rôznych rozhraní a protokolov. V nasledujúcich podkapitolách sa zameriame práve na ne.[5]

5.1 AXI

Prvým spôsobom, akým je možné zrealizovať komunikáciu, je rozhranie AXI[1]. Ide o štandart pre prenos dát v digitálnych obvodoch, ktorý poskytuje jednotný spôsob komunikácie medzi perifériami a procesorom. Toto rozhranie taktiež obsahuje funkcie zabezpečenia synchronizácie a bezpečnosti prenosu dát. Navyše definuje rôzne režimy prenosu údajov, a to *block* na prenos údajov v bloku, režim *streaming* na prenos údajov jednotlivo za sebou a režim *burst* na prenos údajov po menších blokoch. [5]

5.2 DDR

Ďalšou možnosťou komunikácie je použitie rozhrania DDR, teda štandard pre synchrónnu dynamickú pamäť, ktorý umožňuje prenos veľkého veľkého množstva dát mezi blokmi FPGA a SoC obvodom. Tento spôsob prenosu dát je pomalší než využitie rozhrania AXI, no okrem prenosu, umožňuje aj ukladanie väčšieho množstva dát a je tak vhodnejší pre niektorý typ aplikácií.[5]

5.3 Ethernet

Toto rozhranie slúži na prenos dát medzi FPGA a sieťovým prostredím v reálnom čase. Táto komunikácia na FPGA karte môže byť realizovaná pomocou jedného z vybraných protokolov, ako sú *TCP/IP*[2], *UDP*[3], *ICMP*. Každý z nich má pre prenos dát svoje vlastné postupy a pravidlá. [5]

[1] Advanced eXtensible Interface
[2] Transmission Control Protocol
[3] User Datagram Protocol

Ak sa bavíme o komunikácii s inými zariadeniami v sieti, na jej realizáciu môžeme využiť *FTP*, *HTTP* protokoly alebo *Telnet*. V tomto prípade nesmieme zabúdať ani na na bezpečnostné opatrenia, keďže v sieťovom prostredí môžeme byť vystavení rôznym hrozbám. Preto je dôležité využívať bezpečnostné protokoly *SSL* alebo *SSH*. Tie zabezpečia integritu a dôvernosť dát.[5]

Ak sa však bavíme o komunikácii pomocou ethernetového rozhrania vrámci dosky, je dôležité implementovať Ethernet protokoly na softvérovú časť FPGA čipu, ktoré túto komunikáciu umožnia. Implementované sú pomocou HDL jazyka, v našom prípade VHDL. Zahŕňajú informácie o MAC[4] vrstve, IP[5] vrstve, a taktiež TCP alebo UDP protokole.[5]

5.4 USB 2.0

Rozhranie USB[6] 2.0 je sériové zbernicové rozhranie určené na prenos dát. Je nástupcom staršej verzie 1.1 od ktorej má vyššiu prenosovú rýchlosť a väčšiu šírku pásma. K jednému USB koreňovému zariadeniu je možné pripojiť až 127 zariadení, ktoré sú klasifikované do piatich rôznych tried podľa spôsobu interakcie medzi zariadením a hostiteľom. USB 2.0 pre komunikáciu medzi zariadeniami využíva diferenciálny signál a narozdiel od AXI rozhrania, prenos prebieha asynchrónne. [5]

Ak sa zameriame na USB 2.0 v spojení s využitím vo vývojovom testovacom kite, môže byť tento kit definovaný buď ako *USB Host* – zariadenie v tomto móde musí ovládať pripojené zariadenia a taktiež vykonávať funkcie, ako sú zápis a čítanie dát – alebo *USB Device*, teda zariadenie ktoré obsahuje nejaké dáta, ktoré môže byť riadené pomocou iného USB Hosta. Na implementáciu USB 2.0 v FPGA obvode môžeme použiť rôznych IP jadier, ako sú *USB 2.0 Host Controller*, *USB 2.0 Device Controller* alebo *USB 2.0 Transciever*, ktoré môžu byť podľa potreby nakonfigurované a integrované do návrhu FPGA obvodu.[5]

5.5 Micro SD

Micro SD[7], je typom pamäťovej karty, ktorá sa používa na ukladanie dát v rôznych elektronických zariadeniach. Čo sa týka vývojových kitov s FPGA sú tieto karty jednou z možností pre ukladanie a nahrávanie konfiguračných súborov s inštrukciami pre FPGA obvod. Karta sa do kitu pripája pomocou rozhrania priamo na to určeného. [5]

[4]Media Acces Control
[5]Internet Protocol
[6]Universal Serial Bus
[7]Secure Digital

Na komunikáciu sa využívajú rôzne protokoly. Najčastejšie ide o *SPI*[8]protokol. V tomto prípade je komunikácia realizovaná pomocou spoločnej zbernice a štyroch typov signálu. Prvé dva, MOSI – Master Out, Slave in; a MISO – Master In, Slave Out; slúžia na prenos dát medzi kartou a FPGA. Ďalším, tretím typom signálu v poradí, je SCLK. Ten určuje rýchlosť prenosu dát. Posledným typom je CS – Chip Select – ktorý umožňuje vybrať konkrétnu kartu pre komunikáciu.[5]

Ďalším typom protokolu, využívajúceho sa na komunikáciu medzi FPGA obvodom a kartou, je *SDIO*[9] protokol, ktorý pri komunikácii používa až 8 typov signálu. Vďaka tomu umožňuje vysoko-rýchlostnú komunikáciu s väčším objemom dát. Navyše, protokol SDIO taktiež zabezpečuje dáta aj riadiace signály.[5]

[8]Serial Peripheral Interface
[9]Secure Digital Input Output

6 Kryptografia

V tejto kapitole sa zameriame na kryptografiu a jej základné pojmy. Ďalej sa taktiež zoznámime s rôznymi typmi šifrovacích algoritmov.

6.1 Základné pojmy

Kryptografia je veda zameraná na konštrukciu matematických metód pre zaistenie dôvernosť a taktiež autentičnosti prenášaných dát. Medzi jej základné pojmy patria:

- **Kryptoanalýza** – veda zameraná na prekonávanie matematických metód pre zaistenie autentičnosti a dôveryhodnosti prenášaných dát
- **Kryptológia** – veda zameraná na konštrukciu a prekonávanie matematických metód pre zaistenie autentičnosti a dôveryhodnosti prenášaných dát – skláda sa z kryptoanalýzy a kryptografie
- **Kryptografické systémy** – systémy algoritmov, ktoré zabezpečujú funkcie kryptografie
- **Správa** – „plaintext“, sú dáta, ktoré je treba zašifrovať pomocou kryptografických protokolov
- **Šifrovanie** – je proces transformácie vstupných dát – správa – do dát výstupných – zašifrovaná správa
- **Dešifrovanie** – je opačný proces k šifrovaniu, teda proces transformácie zašifrovanej správy do plaintext-u
- **Kľúč** – „key“, je parameter použitý na šifrovanie a dešifrovanie správy
- **Šifrovaná správa** – „cipher text“, sú nečítateľné dáta, ktoré vzniknú po zašifrovaní správy
- **Šifrátor** – zariadenie, ktorého úlohou je pomocou šifrovania premeniť správu na zašifrovaný text
- **Šifra** – „cipher“ je algoritmus pomocou ktorého sa správa šifruje alebo dešifruje

[4]

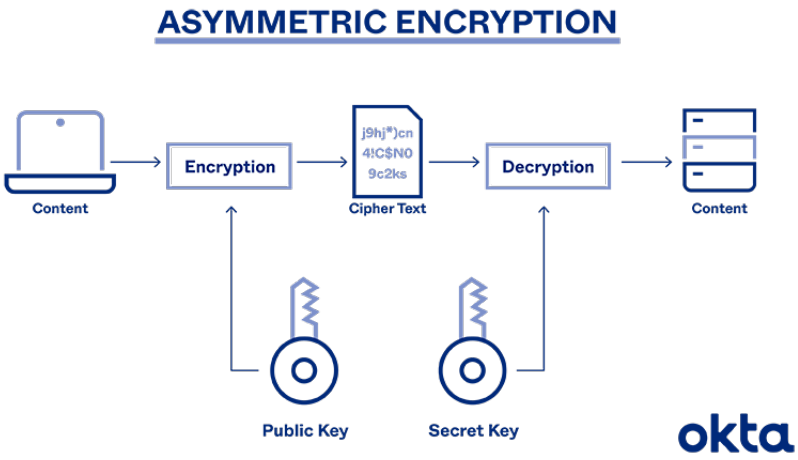
Nakoľko je kryptografia dôležitou súčasťou našich životov, existuje niekoľko princípov, ktoré musí každý kryptografický systém spĺňať, aby sa zabezpečil bezpečný prenos dát.

Prvým princípom je **utajenie**. Jeho úlohov je zabezpečenie dát, ktoré sú odosielané prostredníctvom kryptografických systémov. To dosiahneme pomôcou procesu šifrovania, ktoré sme si vysvetlili vo výpočte vyššie. Druhým princípom je **integrita dát**. To znamená, že posielané dáta a iné informácie musia byť chránené pred poškodením alebo neoprávnenými zmenami. Na splnenie integrity sa používajú rôzne

algoritmy, ako napríklad kontrolný súčet alebo rôzne hashovacie funkcie. Ďalšou nezbytnou častou je zabezpečenie overenia totožnosti účastníkov komunikácie, teda **autentifikácia**. Tú docielime pomocou hesiel, digitálnych certifikátov, ale aj biometrických údajov. Týmto procesom sa zabezpečuje, že všetky strany komunikujú oprávnene. Posledným princípom je **nepopieratelnosť**, teda zamedzenie možnosti popretia činov alebo správ komunikujúcimi stranami.[4]

6.2 Asymetrická kryptografia

Asymetrická kryptografia je typ kryptografie ktorá používa dva rôzne kľúče 6.1. Jedným z nich je verejný kľúč – „public key“ – slúžiaci na zašifrovanie správy a druhým je kľúč súkromný – „private key“ – ten sa zasa používa na opačnú operáciu, teda dešifrovanie. Tieto dva kľúče sú prepojené zložitým matematickým výpočtom a zároveň verejný kľúč neobsahuje informácie o súkromnom kľúči, takže je pri dnešných technológiách nemožné z verejného kľúču vypočítať hodnotu súkromného kľúču. Hlavnou výhodou je teda fakt, že aj v prípade odoslania dát nesprávnemu príjemcovi, ju tento príjemca nebude môcť bez znalosti súkromného kľúču dešifrovať. Postup šifrovania a dešifrovania dát pomocou Asymetrickej kryptografie je následovný: Príjemca vydá verejný kľúč, ten je spolu s asymetrickou šifrou použitý odosielateľom na zašifrovanie správy. Táto správa je poslaná príjemcovi, ktorý ju potom pomocou súkromného kľúča dešifruje. Typickým príkladom tohto typu šifry je **RSA**[1].[21]



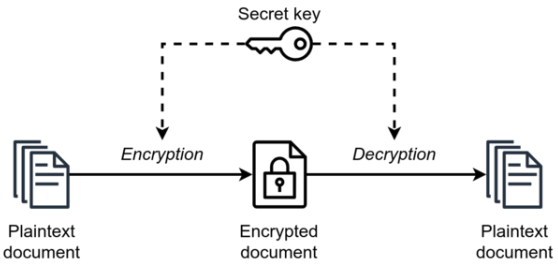
Obr. 6.1: Jednoduchá schéma asymetrického šifrovania

[21]

[1]Rivest-Shamir-Adleman

6.3 Symetrická kryptografia

V šifrách tohto typu sa v drvivej väčšine používa len jeden kľúč, a to súkromný 6.2. Ten slúži ako na šifrovanie, tak aj na dešifrovanie. Pri komunikácií si ho musia obidve strany vymeniť. Počas ustanovenia alebo už samotnej výmene kľúčov, môže nastať problém. Ak tento prípad nastane, je velice jednoduché kľúč odchytiť, pokiaľ komunikácia nie je zabezpečená. Jedinou bezpečnou možnosťou, ako bezpečne predať súkromný kľúč je fyzická výmena, no to je pri dnešnej globálnej spoločnosti neprevediteľné. Ďalším spôsobom ustanovenia a výmeny kľúču cez nezabezpečenú sieť, je využitie asymetrickej kryptografie. V tomto prípade sa aj v symetrickej kryptografii používajú dva kľúče. Jeden na šifrovanie a druhý na dešifrovanie. Nakoľko je proces ustanovenia známy, je dôležité zvoliť veľký rozsah kľúčov. V opačnom prípade by mohli byť prelomené hrubou silou. [21]



[21]

Obr. 6.2: Jednoduchá schéma symetrického šifrovania

Tento typ sa delí na blokové a prúdové šifry, z pohľadu rozdelenia dát určených na šifrovanie.[21]

6.3.1 Blokové šifry

Prvým typom šifier symetrickej kryptografie sú blokové šifry. Ich hlavným znakom je, že sa správa rozdelí do určitého počtu blokov rovnakej veľkosti a šifrovanie prebieha na každom bloku zvlášť. V dnešnej dobe blokové šifry využívajú princíp kaskádových šifier. Ide o zreťazenie viacerých šifier za sebou spôsobom, že vstupným blokom šifry, je výstupný blok šifry predchádzajúcej, pričom výstup poslednej šifry je výstupom šifrovanou správou celého procesu šifrovania.[10] [11]

Ďalším typom blokových šifier sú šifry iterované. Sú založené na princípe kaskádových šifier, konkrétne zreťazenia viacerých, najčastejšie jednoduchých šifier za sebou. Tá môže byť reprezentovaná Feistelovou sieťou alebo substitučno-permutačnou sieťou. Pri iterovaných blokových šifrách sa v expanznom bloku odvodí niekoľko

iteračných kľúčov zo súkromného kľúča, ktoré sú následne použité pre každú iteráciu. Pri šifrovaní v rôznych iteráciach sa používajú aj matematické operácie, ako substitúcie, permutácie a aritmetické operácie.[10] [11]

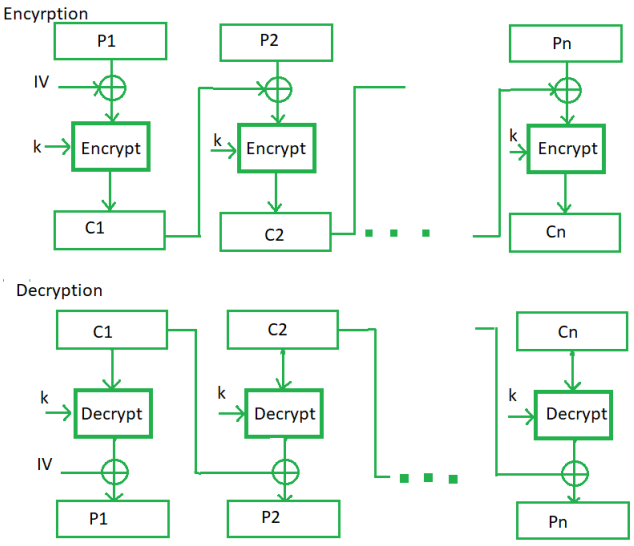
Feistelova sieť je typ iterovanej šifry, ktorý funguje na princípe rozdelenia vstupného bloku na dve polovičky – ľavú a pravú. Pravá polka vstupuje do konverznej funkcie s iteračným kľúčom. Výstup tejto funkcie ma polovičnú veľkosť vstupného bloku dát. Po iterácii nám vznikne blok o veľkosti vstupného bloku, kde ľavou stranou je predošlá pravá strana a naopak.[10] [11]

Okrem Feistelovej siete, môže byť iterovaná šifra reprezentovaná substitučno-permutačnou sieťou. Proces šifrovania je postavený na substitúcii a následjnej permutácii vstupných bitov. Substitúcie využívajú S-boxy, ktoré pracujú časťami dát veľkosti 4 a 8 bitov, a permutácie P-boxy, ktoré sa snažia dosiahnúť čo najviac difúzií. Tým sa ovplyvní počet S-boxov. Substitučno-permutačné siete taktiež využivajú rôzne operácie. Ide napríklad o *ShiftRow* – náhodné, posunutie riadku matice do strán, pričom sa žiadne dva riadky nikdy neposunú o rovnaký počet miest – *MixColumns* – výmena stĺpcov, nasledovaná vynásobením každého stĺpca rovnakým polynomom – *AddRoundKey* – kombinácia každého bitu dát s iteračným kľúčom.[10] [11]

6.3.2 Prevádzkový režim u blokových šifier

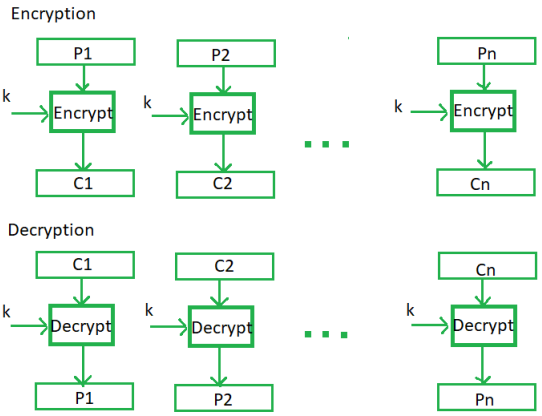
Prevádzkový režim je akýsi mód, ktorý určuje akým spôsobom sú šifrované a dešifrované dáta s väčšou dĺžkou bloku než šifrovací algoritmus podporuje. Je ich 5 a sú to:

1. **CBC** – „cipher block chaining“ (6.3) alebo reťazenie šifrových blokov, je jedným z najpoužívanejších prevádzkových režimov. Prvým krokom je doplnenie paddingu správy. Ďalej nasleduje správa rozdelenie správy na bloky o n-veľkosti. V nasledujúcom kroku sa medzi určitým blokom a jeho predchádzajúcim použije operácia XOR, na ktorej výsledok je následne aplikovaná bloková šifra. Keďže prvý blok, predchádzajúci blok nemá, je na jeho miesto umiestnení náhodná vstupná hodnota, slúžiaca k dosiahnutiu rozdielnosti šifrovaných dát pri rovnakej správe.[10]
2. **ECB** – „electronic codebook“(6.4 je najjednoduchší operačný mód. Prvý krok je rovnaký ako u ECB, čiže doplnenie paddingu. Ďalej je správa rozdelená do blokov o veľkosti n-bitov a tie sú následne blokovou šifrou šifrované samostatne. Proces dešifrovania je rovnaký ako u šifrovania.[10]
3. **CFB** – „cipher feedback“(6.5) má schopnosť zmeniť blokovú šifru, do asynchronnej prúdovej šifry. Počas celého procesu nepotrebuje používať doplnenie paddingu. Postup tohoto operačného módu je následovný Do vstupu je vložený



[10]

Obr. 6.3: CBC



[10]

Obr. 6.4: ECB

výsledok XOR operácie správy

$$m_i$$

a šifrovaného vstupného bloku. Pre prvý vstupný blok je opäť využitý inicializačný vektor, no tentokrát je šifrovaný individuálne. Často nastáva situácia kedy operačný mód používa XOR operáciu menšej časti správy než je veľkosť bloku. To ma za následok zvýšenie počtu potrebných previazaní a je k nemu potrebné

$$s$$

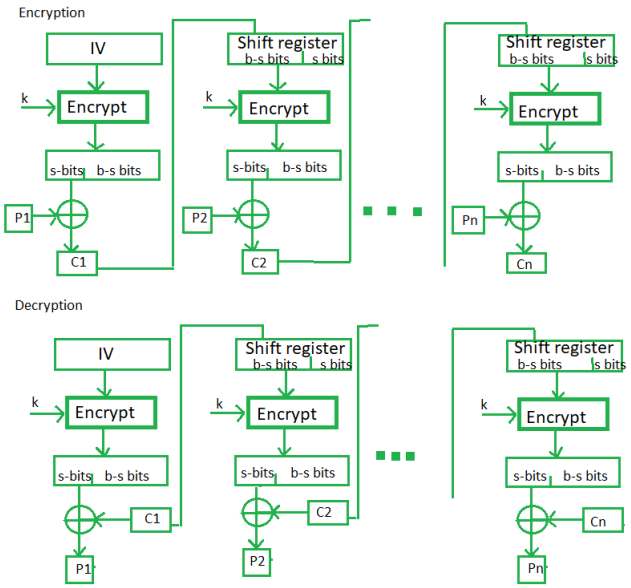
, kde

$$s \in N^+, 1 \leq s \leq b$$

, kde b je veľkosť bloku šifry. Zo zašifrovaného vstupného bloku sa použije s najviac reprezentačných bitov, na ktoré sa znova použije XOR operácia, tentokrát ale s rovnakým počtom bitov správy. Výsledok tejto operácie je prvá časť zašifrovanej správy. Vstupom následujúceho bloku je spojenie s bitov šifrovanej správy a

$$b - s$$

najmenej reprezentačných bitov inicializačného vektora. Toto spojenie je použité ako ďalší vstupný blok.[10]



[10]

Obr. 6.5: CFB

4. **OFB** – „output feedback“(6.6) alebo výstupná spätná väzba mení blokovú šifru do synchrónnej prúdovej šifry. Ide o veľmi podobný prevádzkový režim ako CFB. Hlavným rozdielom je spôsob výpočtu vstupného bloku. V tomto prípade sa ako vstupný blok používa výstupný blok bez XOR operácie so správou. Pre prvý vstupný blok je použitý ako vstup znova inicializačný vektor. Šifrovaná správa sa získa vykonaním XOR operácie medzi správou a výstupným blokom. Ak nastane situácia, kedy má posledný blok veľkosť menšiu

$$u$$

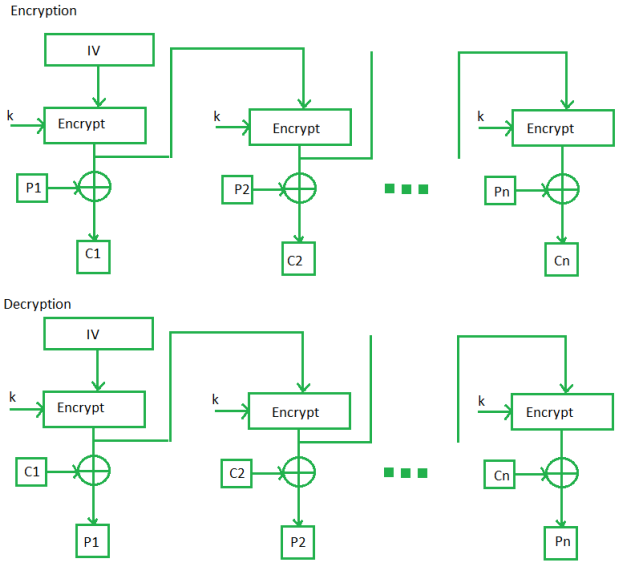
, kde

$$u \in N^{+}, 1 \leq u \leq b$$

je velkosť bloku blokovej šifry, je

$$u - b$$

najmennej reprezentačných bitov z výstupnej správy odstránených. Pre výpočet poslednej časti správy je využitá XOR operácia medzi ostatnými bitmi výstupnej správy a poslednou časťou správy.[10]



[10]

Obr. 6.6: OFB

5. **CTR** – „counter“(6.7) alebo aj počítací režim rovnako ako OFB mení blokovú šifru na synchrónnu prúdovú šifru. Ako vstupný blok vstupuje do systému hodnota „nonce“, ktorú najčastejšie reprezentuje inicializačný vektor. K tejto hodnote je pripočítaná hodnota

$$i$$

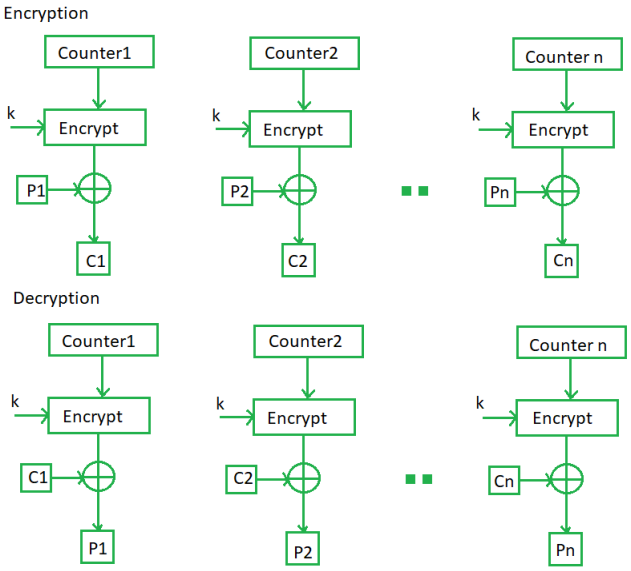
, ktorá reprezentuje hodnotu poradia bloku. Výsledok tohto súčtu je následne modulovaný

$$2^n$$

, kde

$$n$$

je veľkosť vstupného bloku v bitoch. To zabezpečí, že vstup je vždy o rovnakej dĺžke. Vstupný blok je následne šifrovaný blokovou šifrou. Na blok správy je spoločne so šifrovaným výstupným blokom použitá XOR operácia. Výsledkom je šifrovaný text. Ak je posledný blok správy kratší, postupuje sa presne ako v prípade OFB.[10]



[10]

Obr. 6.7: CTR

6.3.3 Prúdové šifry

Tento typ spracúva dáta ako prúd bitov, pričom sa šifruje každý bit samostatne. Ich výhodami proti blokovým šifrám je veľmi malá chybovosť, väčšia rýchlosť a taktiež menšia hardvérová náročnosť.[11]

Na základe spôsobu generácie „keystreamu“, teda prúdu znakov kľúču, poznáme dva typy prúdových šifier, a to synchrónne a asynchrónne. Pri prvom type nezáleží na šifrovanom texte a „plaintext“ pri generácii keystreamu. Tá závisí len na kľúči a algoritme, ku ktorému sa v moderných šifrách pridávajú inicializačné vektory. Tento typ šifier funguje na princípe synchronizácie odosielateľa a príjemcu, čo znamená, že pre obidvoch musí byť rovnaký zdieľaný kľuč a stav algoritmu. Pri strate synchronizácie príjemca správu nedokáže dešifrovať. Pozitívom synchrónnych blokových šifier, je možnosť doplnenia správy, bez jej ovplyvnenia, v prípade, že by obsahovala menej znakov, než by mala mať.[11]

Druhým typom prúdových šifier sú asynchrónne prúdové šifry. V tomto prípade je keystream generovaný pomocou šifrovacieho algoritmu pomocou predchádzajúcich znakov šifrovanej správy a kľúča. Aj tu existuje možnosť synchronizácie, no narozdiel od synchrónnych sa šifra v prípade straty synchronizácie, pri tomto type dokáže po niekoľkých šifrovaných znakoch synchronizovať. Ďalšou vlastnosťou je chybné dešifrovanie pár znakov po výskyte chyby, no zvyšok správy bude dešifrovaný správne.[11]

7 AES

AES (Advanced Encryption System) je symetrický šifrovací algoritmus, ktorý sa používa na zabezpečenie utajenia informácií. Bol vytvorený ako nástupca štandardu DES(Data Encryption Standard) a stal sa jedným z najpoužívanejších šifrovacích algoritmov na svete. Ide o blokový šifrovací algoritmus. Podporuje rôzne dĺžky kľúčov vrátane veľkosti 128, 192 a 256 bitov. Štandardná dĺžka bloku je 128 bitov.[7]

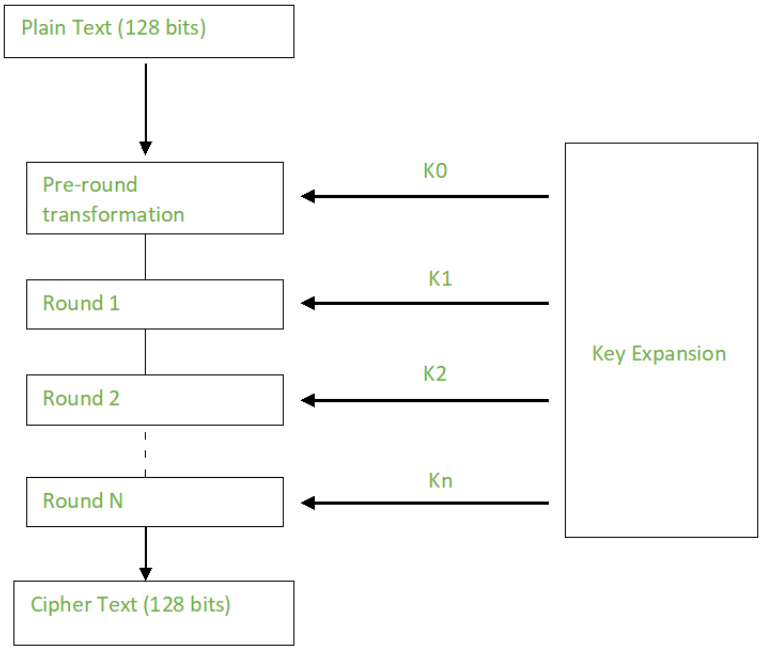
Na symetrické blokové šifry sme sa zamerali v predchádzajúcej kapitole, čiže v tejto si spomenieme len stručný popis.

7.1 Operácie pre AES

AES ako šifrovací algoritmus vykonáva viacero operácií, ktoré sú „vnorené“ do procesu zašifrovania a dešifrovania dát. Ide o tieto operácie:

1. **Tvorba podkľúčov** – Pri inicializácii AES algoritmu sa súkromný kľúč rozdelí na viacero menších podkľúčov, ktoré sa neskôr využijú v jednotlivých krokoch šifrovania. Ich počet závisí od veľkosti súkromného kľúča. Pre porovnanie, AES-128 generuje 11 podkľúčov, pre AES-192 je to 13 podkľúčov, a pre AES-256 je to 15 podkľúčov. Každý podkľúč je odvodený od predchádzajúceho pomocou rôznych operácií, ako sú rotácie, XOR operácie, nahrádzanie bitov,...
2. **Mixovanie dát** – tieto operácie sú vykonávané pomocou funkcií MixColumns, AddRoundKey a ShiftRows, ktoré sme si bližšie popísali v predchádzajúcej kapitole.
3. **Substitučné a permutačné operácie** – operácie s využívaním S-boxov a P-boxov, ktoré sme si taktiež popísali v predchádzajúcej kapitole

[7]



Obr. 7.1: Substitučno-permutačná sieť

8 Vivado

Ako sme už spomenuli, na implementáciu našej aplikácie na programovateľný logický obvod, musíme použiť nejakú zo softvérových aplikácií, na to určených. V našom prípade to je **Vivado**. Ide o nástroj na automatizáciu elektronického návrhu, vyvinutého spoločnosťou **Xilinx**, teda popredným dodávateľom programovateľných logických zariadení. Jeho prvá verzia *1.0* bola vydaná v roku 2012, ako náhrada za predchádzajúci nástroj *ISE*. Od svojho prvého vydania sa Vivado stalo najrozšírenejším nástrojom na automatizáciu elektronických návrhov, ktorý zákazníci spoločnosti Xilinx používajú na návrh FPGA.[3]

Tu je výpis niektorých verzií Vivado, ktoré do návrhu programovateľných logických polí priniesli nové, zaujímavé možnosti a rozšírili funkcionalitu tohto nástroju.

- *2013.1* – pridaná podpora pre zariadenia *Zynq-7000 SoC* a *UltraScale*
- *2014.1* – táto verzia prišla s podporou pre syntézu vysokej úrovne a metodiku návrhu *UltraFast*
- *2015.1* – podpora pre rodinu *UltraScale+*
- *2016.1* – pridaná podpora pre vývojové prostredie *SDK* pre softvérom definované systémy
- *2018.1* – táto verzia pridala podporu pre karty *Alveo Data Center Accelerator Cards*, nové možnosti časovej uzávery a optimalizácie napájania a taktiež podporu pre *Zynq RFSoC*
- *2020.1* – podpora pre strojové učenie (*Vivado Machine Learning*) a vývojové prostredie umelej inteligencie *Vitis*
- *2021.1* – verzia s podporou pre *Vivado System Edition* s novými možnosťami návrhu a ladenia na úrovni systému

[3]

V roku 2022 bola predstavená najnovšia verzia s názvom *Vivado 2022.1*. V súčasnosti sa však naďalej vyvíja a rozširuje pole svojich možností a ďalších funkcií.[3]

V tejto kapitole sa zameriame na jeho funkcionalitu, vlastnosti, ktoré Vivado ponúka a možné problémy, s ktorými sa pri jeho používaní, môžeme stretnúť.

8.1 Funkcionalita

Vivado bolo navrhnuté s cieľom riešiť rastúcu zložitosť návrhov FPGA, ktorých zvládnutie bolo čoraz náročnejšie a vtedajšie tradičné EDA nástroje na to nestačili. Vivado tak obsahuje pokročilé funkcie, ako je syntéza na vysokej úrovni, grafické používateľské rozhranie na integráciu IP a nový engine na umiestnenie a smerovanie, ktorý podporuje ako hierarchické, tak aj inkrementálne návrhové toky.

Jedným z najzaujímavejších aspektov systému Vivado je využitie technológie už spomenutej syntéze na vysokej úrovni, ktorá dizajnérom umožňuje vytvárať komplexné návrhy FPGA pomocou programovacích jazykov vysokej úrovne, ako sú C, C++ a SystemC. To uľahčuje softvérovým inžinierom vstup do sféry návrhu FPGA, čím je umožnená rýchlejšia a efektívnejšia iterácia návrhu.

Ďalšou zaujímavou funkciou programu Vivado je podpora čiastočnej rekonfigurácie, ktorá návrhárom umožňuje opätovne upraviť časti návrhu FPGA, zatiaľ čo zvyšok systému funguje naďalej normálne. Užitočnosť tejto vlastnosti sa môže prejaviť v aplikáciách, kde je dôležité minimalizovať prestoje alebo zachovať schopnosť prevádzky systému.[3]

9 IDE

IDE, anglicky „Integrated Development Enviroment“ predstavuje softvérovú aplikáciu, ktorá slúži na vývoj, testovanie ladenie softvérových projektov. Väčšinou programátorom poskytuje všetky potrebné nástroje a funkcie pre efektívne vytváranie softvéru v jednom centralizovanom prostredí, od napísania kódu, cez ladenie až po spustenie a testovanie.

Takisto aj pre FPGA problematiku existujú integrované vývojové prostredia na tvorbu softvérových aplikácií. V minulosti išlo o **Xilinx-SDK**, ktorý bol v súčasnosti nahradený programom **Xilinx-Vitis**. Hlavné rozdieli týchto prostredí sa týkajú najmä úrovne podpory pre série platforiem. Ďalším rozdielom je formát používaných hardvérových súborov. Vitis taktiež ponúka viaceré nástroje pre analýzu výkonu a profilovanie, ktorý umožňuje programátorovi optimalizovať ladenie softvéru.[19]

10 Block design

Okrem návrhu hardvéru v jazykoch HDL existuje aj možnosť vytvorenia blokového dizajnu, anglicky „block design“, pomocou rôznych IP(Intellectual Property) jadier. Tento typ návrhu je vo veľa ohľadoch jednoduchší, keďže sa človek nemusí zaoberať písaním kódu, len jednoducho vloží potrebné IP jadrá a prepojí ich. Blokový dizajn ďalej slúži na komunikáciu medzi procesným systémom a programovacou logikou. V tejto kapitole sa pozrieme na proces tvorby blokového dizajnu, ako aj náš dizajn použitý v aplikácii.

10.1 Tvorba blokového dizajnu

Vytvoríme si projekt v prostredí Vivado, vyberieme si dosku, v našom prípade Nexys A7-100T. Existuje možnosť, že by sa zvolená doska nenachádzala v katalógu. V tom prípade sa dajú nájsť súbory na oficiálnych stránkach spoločnosti Digilent. V paneli *PROJECT MANAGER* klikneme na *Create Block Design*, pridáme jadrá, nastavíme parametre IP jadier, prepojíme ich, vygenerujeme Bitstream, ten potom exportujeme do nami zvoleného IDE prostredia, a toto prostredie spustíme.

10.2 Popis blokového dizajnu

Na doske sa nachádza nespočetné množstvo periférií, cez ktoré by sa dala sprostredkovať komunikácia. Či už ide o **ETHERNET**, **VGA** a mnohé ďalšie. My sme si však, zo zložitejších periférií, pre náš projekt vybrali periférium **USB** v **DMA** režime. V tejto časti sa bližšie zoznámime s blokovým návrhom pre našu finálnu aplikáciu.

V našom blokovom 10.1 návrhu sa nachádza 14 IP blokov – **Microblaze proccesing system, Clocking Wizard, Processor system reset, MDM, AXI Interconnect, Local memory, AXI USB2 Device, AXI UARTLITE, Block Memory Generator, AXI BRAM CONTROLLER** a tri **AXI GPIO**. Okrem nich sa v blokovom návrhu nachádza aj 6 I/O prvkov pre interakciu s vývojovou doskou – **rgb led, dip switches 16bits 0, push buttons 5bits** a taktiež I/O ovládajúce systémové hodiny a tlačidlo reset.

IP jadro **Microblaze** je veľmi dôležitou časťou celého návrhu. Jeho hlavnou funkciou je komunikácia s I/O prvkami. Komunikácia s týmito vstupno-výstupnými zariadeniami je možné pomocou niektorej z linuxových distribúcií, napríklad Petalinux. Ďalším spôsobom je „bare metal“.

Clocking Wizard slúži na generovanie a konfiguráciu hodín.

Processor system reset obsluhuje požiadavky prerušenia. Má za úlohu zaistiť, že požiadavky prerušenia budú jednotné pre celý návrh.

MDM umožňuje ladenie a správu programov pre mikroprocesor Microblaze.

AXI Interconnect má za úlohu prepájať primárne a sekundárne AXI rozhrania, ktoré využívajú mapovanie pamäti.

AXI USB2 Device umožňuje implementáciu a správu, a taktiež softvérové rozhranie, umožňujúce komunikáciu s USB rozhraniami.

AXI UARTLITE poskytuje jednoduché sériové rozhranie UART pre komunikáciu medzi FPGA a externými zariadeniami.

GPIO IP bloky slúžia k prepojeniu I/O rozhraní s rozhraním AXI. Každý GPIO IP blok sme pripojili ku nejakému I/O prvku. Každé GPIO IP jadro sme pripojili ku príslušnému rozhraniu na komunikáciu s doskou. Každému GPIO rozhraniu môžeme nastaviť rôzne parametre, ako napríklad šírku prenosu dát, anglicky „data width“ alebo smer prenosu dát. Či ide o *input* alebo *output*. Zaujímavosťou je, že na jedno GPIO jadro, môžu byť pripojené 2 I/O prvky s rôznou šírkou a smerom prenosu dát. Prvé GPIO je nastavené ako output a je pripojené na 2 RGB LED. Ich farba nám bude znázorňovať proces vykonanej funkcie. Ostatné jadrá sú nastavené ako input. I/O prvky tlačidiel budú slúžiť na orientáciu v aplikácii, ako výber, spustenie či skončenie funkcie, ale aj celej aplikácie. Prepínače zasa poslúžia na jednu z funkcií, ktorá naša doska umožní.

10.3 Hardvérové nároky blokového dizajnu

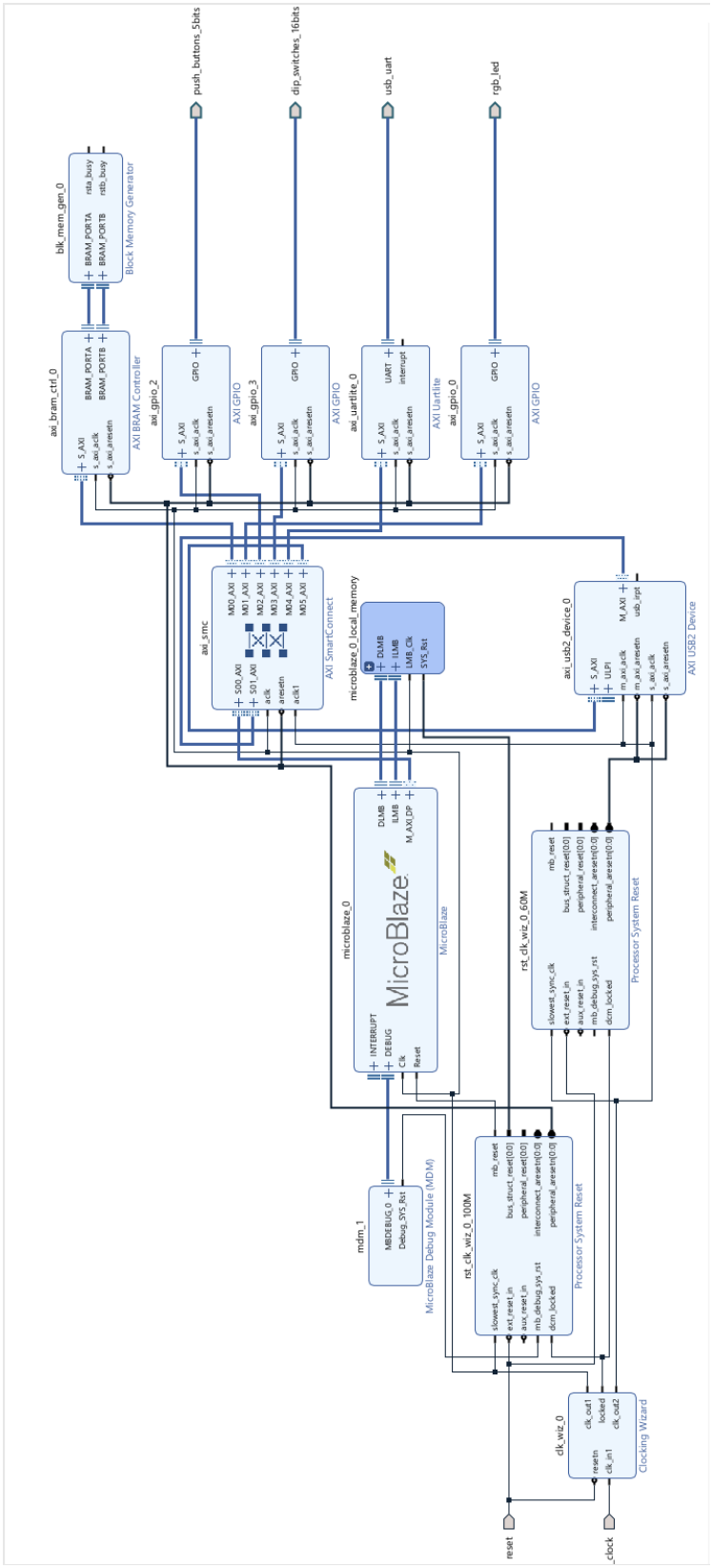
Návrh nášho blokového dizajnu je náročnejší, ako u iných prác, nakoľko náš návrh obsahuje veľký počet jadier, navyše bolo treba alokovať väčší objem pamäte, keďže počítame s tým, že naša aplikácia bude obsahovať veľa inštrukcií:

ZDROJ	VYUŽITÉ ZDROJE	MAXIMÁLNE ZDROJE	PERCENTUÁLNE VYUŽITIE
LUT	7283	63400	11,49%
LUTRAM	784	19000	4,13%
FF	7883	126800	6,22%
BRAM	115	135	85,19%
IO	31	210	14,76%
BUFG	4	32	12,50%
MMCM	1	6	16,67%

Tab. 10.1: Tabuľka využitých zdrojov

Z tabuľky (10.1) môžeme vyčítať, že nás dizajn naozaj využíva veľa zdrojov. Najmä BRAM zdroje sú skoro maximálne vyčerpané. Na druhú stranu LUTRAM a FF sú v porovnaní s ostatnými využité oveľa menej.

Takéto veľké využitie zdrojov, môže zapríčiniť problémy pri implementácii softvérovej aplikácie na dosku. Hlavne čo sa týka času potrebného na naprogramovanie dosky, bude určite väčší ako iných návrhov, využívajúcich menšie zdroje.



Obr. 10.1: Block design

11 Softvérová aplikácia

V tejto kapitole sa zameriame na prepojenie aplikácie, s prvkami blokového návrhu. Úlohou aplikácie bola inicializácia jednotlivých periférií komunikujúcich s FPGA a taktiež vytvorenie užívateľského rozhrania na ich ovládanie.

11.1 Prepojenie procesného systému s blokovým návrhom

Na procesnom systéme nám bude fungovať naša aplikácia ako „embedded“ aplikácia, skompilovaná pomocou „bare metal“, takže by mohla byť nahratá na akúkoľvek vývojovú dosku Nexys A7, bez potreby inštalovania operačného systému.

Na zaistenie komunikácie pomocou GPIO používame základné knižnice *xgpio.h* od spoločnosti Xilinx. Aby sme s GPIO vedeli komunikovať, musíme ich najprv inicializovať. To dosiahneme vytvorením konfiguračného súbor **XGpio_Config** pre každé GPIO, do neho načítame hodnotu funkcie **XGpio_LookupConfig(u16 DeviceId)**, kde jediným parametrom je premenná lokality GPIO inštancie. Druhým krokom inicializácie je zavolanie funkcie **XGpio_CfgInitialize(XGpio *InstancePtr, XGpio_Config *Config, UINTPTR EffectiveAddr)** , kde prvý parameter je inštancia XGpio, druhý je konfiguračný súbor, ktorý bol popísaný vyššie, a ako tretí parameter vložíme parameter **BaseAddr** konfiguračného súboru. Okrem tohoto musíme pre jadrá slúžiace na input a output, ďalej nastaviť smer dátového toku pomocou funkcie XGpio_SetDataDirection(. . .).

11.2 Komunikácia s RGB LED a tlačidlami

Okrem inicializácie a smeru AXI GPIO jadra pre RGB LED, musíme zavolať funkciu na **XGpio_DiscreteWrite(XGpio *InstancePtr, unsigned Channel, u32 Data)** . S prvým parametrom sme sa už stretli, druhý znázorňuje číslo komunikačného kanálu, na ktorom bude prebiehať komunikácia a posledným je hodnota v hexadecimálnom tvare predaná GPIO bloku, ktorá značí farbu, ktorou má daná RGB_LED svietiť.

Čo sa týka komunikácie pomocou tlačidiel, existuje výstupná funkcia **XGpio_DiscreteRead(XGpio *InstancePtr, unsigned Channel)** . Prvý parameter je adresa inicializovanej premennej a druhý parameter značí číslo kanálu, na ktorom daná operácia bude prebiehať. V nasledovnej (11.1) tabuľke môžete vidieť bitové hodnoty tlačidiel.

Názov tlačidla	BTNR	BTNC	BTNU	BTNL
Bitová hodnota	0b1000	0b0001	0b0010	0b0100

Tab. 11.1: Tabuľka bitových hodnôt tlačidiel

Podobným spôsobom dokážeme inicializovať všetky ostatné jadrá, ktoré pre našu aplikáciu budeme potrebovať. Každé jadro navyše obsahuje knižnicu s príkazmi na prácu s nimi. Tieto knižnice sú automaticky pridané do nami zvoleného IDE programu.

11.3 AES knižnica

Pre potreby našej aplikácie sme si vytvorili knižnicu v jazyku c s názvom **aes.c**, a taktiež header file **aes.h**. Knižnica slúži na šifrovanie dát pomocou AES. Ide o AES 128, 192, 256, pre ktoré podporuje módy: ECB, CTC aj CTR. Header file tejto knižnice funguje na spôsobe konfiguračného súbora, kde sa dá zvoliť mód šifrovania a veľkosť kľúča. Podrobnejšie je táto knižnica popísana v zdrojových súboroch, v prílohe.

Okrem AES knižnice, aplikácia pracuje ešte s jednou externou knižnicou, a to FATfs. Ide o knižnicu ktorá slúži na súborové operácie.

11.4 Aplikácia

Aplikáciu na FPGA spustíme následovne:

1. V súbore aes.h nastavíme hodnotu AES na 128, 192 alebo 256 – tento krok nie je nutný
2. Overíme, že je naša doska pripojená cez UART k hostovi a že je hostom rozpoznaná
3. Otvoríme si IDE v pracovnom adresári so zdrojovými kódmi
4. Naprogramujem FPGA s priloženým bitstreamom
5. Pripojíme sa cez sériový port na konzoly, na čítanie výstupu a vkladanie textového vstupu do nasej dosky – (baudrate = 9600)
6. Spustíme aplikáciu

Naša aplikácia je postavená na jednoduchom GUI, ktoré sa po inicializácie aplikácie, inicializuje GPIO INPUT a OUTPUT. Ďalej sa nám vypíše jednoduché menu, v ktorom sa orientujeme pomocou tlačidiel na doske. Menu sa skladá zo 4 možností:

1. BTNU – suma zapnutých switchov s exponenciálnom hodnotou
2. BTNR – šifrovanie UART vstupu
3. BTNC – zašifrovanie súboru z USB kľúču

4. BTNL – koniec aplikácie

Teraz sa pozrieme na základné funkcie jednotlivých API našej aplikácie. Zameriame sa na postupy ich správneho vykonania, ich potrebné vstupy a výstupy. Začneme prvou, najjednoduchšou, a to sumou zapnutých switchov

11.4.1 Suma zapnutých switchov

Prvá funkcia po výbere začne automaticky vypisovať sumu zapnutých switchov, ktoré si v tomto príklade predstavíme ako základ mocniny. Na doske máme 16 switchov a každý z nich je identifikovaný číslom 0-15. Toto číslo teda budeme brať ako exponent. Ak teda bude bude zapnutý switch číslo 4 a taktiež číslo 8, výsledný výstup do konzoly bude 272, keďže

$$2^4 + 2^8 = 272$$

. Aplikácia funguje na neustálom posielaní dát cez UART do konzole, až kým užívatel nestlačí tlačídlo BTNL pre návrat do hlavného menu.

Exponent	Hodnota mocniny čísla 2
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768

Tab. 11.2: Hodnoty mocnín čísla 2

Hodnoty mocnín čísla 2 (11.2) sa potom sčítavajú. Vypísaná bude hodnota súčtu zapnutých switchov.

11.4.2 Šifrovanie UART vstupu

Na začiatku sa inicializuje UARTLite IP jadro, čaká sa na užívateľa kým do konzole zadá správu, užívateľ opäť stlačí BTNR a začne proces šifrovania. Vybraný mód šifry bude podľa výberu pred spustením aplikácie. Do konzole je vypísaný šifrovací kľúč a následne aj zašifrovaná správa. Užívateľ je po uplynutí 2 sekúnd vrátený do menu. Šifrovanie pomocou AES je možné vykonať v režimoch ECB, CBC a CTR. Výber parametrov AES šifrovania je závislý od nastavenia daných parametrov v súbore *aes.c*. Pre dokázanie komplexnosti našej knižnice sme ako pôvodné nastavenie zvolili CTR prevádzkový režim. Postup šifrovania je následovný:

1. Z menu je volaná funkcia na šifrovanie v danom móde – v našom prípade ctr
2. Je načítaný kľúč o danej veľkosti a inicializačný vektor
3. Užívateľ vloží svoju správu
4. Je volaná funkcia na inicializáciu inicializačného vektora
5. V ďalšom kroku je volaná funkcia na zašifrovanie správy
6. V poslednom kroku nastáva výpis zašifrovanej správy

API používa preddefinovaný kľúč priamo v aplikácii. To iste platí o inicializačnom vektore.

11.4.3 Zašifrovanie súboru z USB kľúču

API na zašifrovanie súboru z USB kľúču je zo všetkých prítomných najzložitejšia, keďže USB kľúč nie je priamo perifériou dosky, ako to bolo u ostatných dvoch, ale je pripojený do periférie **USB HOST**. Presný postup vykonania API je následovný:

1. Vyberieme príslušnu API z menu.
2. Uistíme sa, že máme pripojený USB kľúč do našej periférie
3. Inicializujeme *Xusb* entitu
4. Aplikácia skúma či sa na USB kľúči nachádzajú nejaké súbory
5. Po nájdení prichádza k zašifrovaniu dát pomocou AES
6. Tieto zašifrované dáta sú uložené do nového súboru na USB kľúči

Pri každej z vyššie uvedených aplikácií vykonávajú svoju funkcionalitu taktiež RGB_LED žiarovky. Znázorňujú napríklad stav šifrovania.

11.5 Ďalšie použiteľné periférie

Nakoľko doska Nexys A7-100t obsahuje len 4 ovládateľné tlačidlá, implementácia API ďalších periférií by bola zložitejšia. Na druhú môžu byť operácie vykonávané aj bez nutnosti ich inicializácie v IDE prostrediach. Postačí nám na to vývojové prostredie Vivado. V prílohe nájdeme príklady primitívnych aplikácií na ovládanie

výpisu Seven-Segmen displeju pomocou switchov a taktiež aplikáciu na sekvenčné blikanie LED žiaroviek v jazyku VHDL.

Závěr

Cieľom práce bolo zamerané na komunikáciu vnútri hardvérovo akcelerovaného obvodu, konkrétnejšie na komunikáciu medzi blokmi testovacieho FPGA kitu. Ďalším cieľom bolo vytvorenie užívateľského rozhrania na ovládanie tejto komunikácie medzi blokmi, konkrétne na USB periférii. Na začiatku práce sme sa zamerali na teoretickú časť práce.

Úvodnou témou bola problematika kryptografie, kde sme sa zamerali na kľúčové pojmy, algoritmy sme si rozdelili podľa rôznych kritérií a neskôr sa zaoberali konkrétným šifrovacím algoritmom AES. Ten bol neskôr použitý, pomocou nami inicializovanej knižnice v jazyku c, aj v praktickej časti našej práce.

Ďalšou témou bola teória hradlových polí, ich história, funkcionalita a pozreli sme sa taktiež na určité predpovede do budúcnosti, čo sa týka vývoja tohto typu hardvéru. Ďalším bodom bolo logické programovateľné hradlové pole Artix-7. Okrem toho, sme sa bližšie pozreli aj na FPGA kit Nexys A7-100T, ktorý sme použili ako platformu v praktickej časti. Pozreli sme sa na jeho technické špecifikácie a taktiež na možnosti komunikácie uživateľa s FPGA čipom pomocou dostupných periférií.

V ďalšej časti sme sa zaoberali HDL jazykmi na implementáciu návrhov hardvéru, konkrétne na VHDL. Preštudovali sme si jeho základné schémy, syntax a porovnali s ostatnými HDL jazykmi, ako napríklad Verilog.

Potom sme sa vrhli na vývojové prostredia, a to Vivado a Xilinx-SDK, respektíve Vitis. Prvý nám poslúžil pri vytváraní opisu hardvéru, ostatné dva zase už na implementáciu softvérovej aplikácie na FPGA kit.

V predposlednej časti sme sa zamerali na nami navrhnutý hardvérový opis pomocou blokového dizajnu. Rozobrali sme si postup, taktiež sme si detailnejšie opísali časti nášho návrhu a pozreli sme sa aj na jeho hardvérové nároky.

V poslednej časti sme sa zamerali na tvorbu samotnej aplikácie, prepojenie softvéru a hardvéru, opísali nami vytvorenú AES knižnicu. V poslednom kroku sme si zhrnuli funkcionalitu API našej aplikácie.

Celá aplikácia bola ovládaná pomocou UART rozhrania, a to mohlo v určitých prípadoch vytvárať problémy, počas behu programu, nakoľko, keď posielal dát, tak zároveň blokoval prenos dát do GPIO INPUT. Našťastie sme si túto chybu všimli a opravili

Z našej práce vychádza, že sme mohli určite skúsiť blokový dizajn s menším počtom jadier, aby sme tak ovplyvnili rýchlosť vykonávania aplikácie a programovania samotnej dosky. Taktiež sme videli, že AES je vhodný pre šifrovanie dát, keďže ako symetrická šifra ponúka rôzne moduly šifrovania a ich kombinovanie.

Čo sa týka možností vylepšenia, tak jednou z nich je, že by mohlo byť do aplikácie implementované viacero periférií, no na druhú stranu, ovládanie užívateľského

rozhrania by sa kvôli tomu mohlo stať zložitejším, nakoľko naša doska obsahuje len 5 tlačidiel, z ktorých na ovládanie komunikácie slúžia len 4. Ďalším vylepšením by mohlo byť možnosť zvoliť režim AES šifrovania priamo v programe, pomocou rozhodovacieho stromu, ako je to v prípade menu.

Literatura

[1] What is FPGA and How Does it Work? - History-Computer. History-Computer [online]. Copyright © Maslakhatul Khasanah [cit. 24.05.2023]. Dostupné z: https://history-computer.com/fpga-field-programmable-gate-array/

[2] Artix 7 FPGA Family. Xilinx - Adaptable. Intelligent | together we advance [online]. Copyright © 2023 Advanced Micro Devices, Inc [cit. 24.05.2023]. Dostupné z: https://www.xilinx.com/products/silicon-devices/fpga/artix-7.html

[3] Xilinx. CompaniesHistory.com - The largest companies and brands in the world! [online]. Copyright © [cit. 25.05.2023]. Dostupné z: https://www.companieshistory.com/xilinx/

[4] What is Cryptography?. Kaspersky Cyber Security Solutions for Home and Business | Kaspersky [online]. Copyright © [cit. 24.05.2023]. Dostupné z: https://www.kaspersky.com/resource-center/definitions/what-is-cryptography

[5] 8 Uses of FPGA (Field-Programmable Gate Array). Business and Industry News, Analysis and Expert Insights - Spiceworks [online]. Dostupné z: https://www.spiceworks.com/tech/networking/articles/what-is-fpga/

[6] Edwar Gomez, Cesar Hernandez, Fredy Martinez, Performance evaluation of the present cryptographic algorithm over FPGA. Hikari | International Publishers | Science, Technology, Medicine and Economics [online]. Copyright © All Rights Reserved [cit. 24.05.2023]. Dostupné z: http://www.m-hikari.com/ces/ces2017/ces9-12-2017/7653.html

[7] What is the Advanced Encryption Standard (AES)? Definition from SearchSecurity. Purchase Intent Data for Enterprise Tech Sales and Marketing - TechTarget [online]. Dostupné z: https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard

[8] Nexys A7 Artix-7 FPGA Trainer Board - Digilent. Digilent – Start Smart, Build Brilliant. [online]. Copyright © 2023 Digilent [cit. 24.05.2023]. Dostupné z: <https://digilent.com/shop/nexys-a7-fpga-trainer-board-recommended-for-ece-curriculum/>.

[9] Doulos. Doulos - Global Independent Leaders in Design and Verification KnowHow [online]. Copyright © Copyright 2005 [cit. 25.05.2023]. Dostupné z: https://www.doulos.com/knowhow/vhdl/a-brief-history-of-vhdl/

[10] Block Cipher modes of Operation - GeeksforGeeks. GeeksforGeeks | A computer science portal for geeks [online]. Dostupné z: https://www.geeksforgeeks.org/block-cipher-modes-of-operation/

[11] What is a Cipher? Definition, Types, Examples and Methods. [online]. Copyright © Copyright 2011 [cit. 25.05.2023]. Dostupné z: https://intellipaat.com/blog/what-is-a-cipher/?US

[12] Symmetric and Asymmetric Encryption - Huawei Enterprise Support Community. [online]. Copyright © 2023 Huawei Technologies Co., Ltd. All rights reserved. [cit. 24.05.2023]. Dostupné z: <https://forum.huawei.com/enterprise/en/symmetric-and-asymmetric-encryption/thread/1063612-867>.

[13] Shifting to an FPGA Data Center Future: How are FPGAs a Potential Solution? [online]. Dostupné z: https://www.allaboutcircuits.com/news/shifting-to-a-field-programable-gate-array-data-center-future/

[14] FPGA Architecture. Invent Logics - Shop Now for Xilinx FPGA development boards [online]. Copyright © 2023 [cit. 24.05.2023]. Dostupné z: https://allaboutfpga.com/fpga-architecture.

[15] ECB vs CBC | Learn the Key Differences of ECB and CBC. EDUCBA | Best Online Training & Video Courses Certification [online]. Copyright © 2023 [cit. 24.05.2023]. Dostupné z: https://www.educba.com/ecb-vs-cbc/

[16] What is CBC?. Educative: Interactive Courses for Software Developers [online]. Copyright ©2023 Educative, Inc. All rights reserved [cit. 24.05.2023]. Dostupné z: /https://www.educative.io/answers/what-is-cbc/

[17] Roheim PS. The origin and fate of HDL. Adv Exp Med Biol. 1977;82:421-6. doi: 10.1007/978-1-4613-4220-5_99. PMID: 200102.

[18] What is the Difference Between Verilog and VHDL - Pediaa.Com. Pediaa.Com - Know about Anything [online]. Copyright © 2017 [cit. 24.05.2023]. Dostupné z: https://pediaa.com/what-is-the-difference-between-verilog-and-vhdl/

[19] Vitis Software Platform. Xilinx - Adaptable. Intelligent | together we advance [online]. Copyright © 2023 Advanced Micro Devices, Inc [cit. 24.05.2023]. Dostupné z: https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html

[20] What is an FPGA? Field Programmable Gate Array. Xilinx - Adaptable. Intelligent | together we advance [online]. Copyright © 2023 Advanced Micro Devices, Inc [cit. 25.05.2023]. Dostupné z: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html

[21] Symmetric and Asymmetric Encryption - Huawei Enterprise Support Community. [online]. Copyright © 2023 Huawei Technologies Co., Ltd. All rights reserved. [cit. 25.05.2023]. Dostupné z: https://forum.huawei.com/enterprise/en/symmetric-and-asymmetric-encryption/thread/1063612-867

Seznam symbolů a zkratek

EPROM Erasable Programmable Read-Only Memory – pamäť určená na čítanie, ktorá sa dá mazať ultrafialovým svetlom

IoT Internet of Things – internet vecí

AES Advanced Encryption Standard

VPN Virtual Personal Network

CPU Central Processing Unit – hlavný procesor počítača

GPIO General Purpose Input Output

QSPI Quad Serial Peripheral Interface

USB Universal Serial Bus

RGB Red Green Blue

PID Proportional Integral Derivative Regulator

ASIC Integrovaný obvod navrhnutý a vyrábaný pre určitú špecifikáciu

VLSI Very Large-Scale Integration

AXI Advanced eXtensible Interface

Seznam příloh