VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2020 Patrik Dobiáš

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

ASYMETRICKÁ KRYPTOGRAFIE NA FPGA
ASYMMETRIC CRYPTOGRAPHY ON FPGA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Patrik Dobiáš

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. Lukáš Malina, Ph.D.

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Patrik Dobiáš ***ID:*** 203169

Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Asymetrická kryptografie na FPGA

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s postupy hardwarové implementace na platformě FPGA. Analyzujte dostupné implementace asymetrických kryptografických schémat a jejich výsledky na FPGA platformě. Na základě analýzy vyberte vhodný asymetrický kryptosystém k hardwarové implementaci na platformě FPGA. Zaměřte se na digitální podpisy. Proveďte návrh a implementaci hardwarových komponentů schématu digitálního podpisu na platformě FPGA. Hlavním cílem bakalářské práce je funkční hardwarová implementace zvoleného asymetrického kryptografického schématu na platformě FPGA. Dalším cílem je testování funkčnosti, testování výkonnosti, zhodnocení efektivity, vyčíslení zabraných HW zdrojů a případná úprava implementace
schématu pro volání schématu jako extern jazyka P4.

DOPORUČENÁ LITERATURA:

[1] MENEZES, Alfred, Paul C VAN OORSCHOT a Scott A VANSTONE. Handbook of applied cryptography. Boca Raton: CRC Press, c1997. Discrete mathematics and its applications. ISBN 0-8493-8523-7.

[2] KRÁL, Jiří. Řešené příklady ve VHDL: hradlová pole FPGA pro začátečníky. Praha: BEN - technická literatura, 2010. ISBN 978-80-7300-257-2.

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

Vedoucí práce: doc. Ing. Lukáš Malina, Ph.D.

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

ABSTRAKT

Tato bakalářská práce se zabývá analýzou dosavadních hardwarových implementací asymetrických kryptografických schémat na FPGA platformě a následnou implementací kryptografického schématu Ed25519 na této platformě. Výsledná implementace je detailně popsána a porovnána s dosavadními implementacemi. V závěru této práce je popsáno nasazení této implementace na FPGA Virtex UltraScale+.

KLÍČOVÁ SLOVA

Asymetrická kryptografie, ECDSA, Ed25519, FPGA, VHDL

ABSTRACT

This bachelor thesis deals with the analysis of existing hardware implementations of asymmetric cryptographic schemes on the FPGA platform and the then implementation of the cryptographic scheme Ed25519 on this platform. The resulting implementation is described in detail and compared with existing implementations. At the end of this work, the deployment of this implementation on the Virtex UltraScale+ FPGA is described.

KEYWORDS

Asymmetric cryptography, ECDSA, Ed25519, FPGA, VHDL

DOBIÁŠ, Patrik. *Asymetrická kryptografie na FPGA*. Brno, Rok, 56 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Lukáš Malina, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Asymetrická kryptografie na FPGA“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Lukášovi Malinovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Dále bych rád poděkoval panu Ing. Davidovi Smékalovi za jeho ochotu a pomoc při nasazení implementace na FPGA.

Tato práce vznikla jako součást klíčové aktivity KA6 - Individuální výuka a zapojení studentů bakalářských a magisterských studijních programů do výzkumu v rámci projektu OP VVV Vytvoření double-degree doktorského studijního programu Elektronika a informační technologie a vytvoření doktorského studijního programu Informační bezpečnost, reg. č. CZ.02.2.69/0.0/0.0/16_018/0002575.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Projekt je spolufinancován Evropskou unií.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Tato práce se zabývá hardwarovými implementacemi asymetrických kryptografických schémat na platformě FPGA (Field Programmable Gate Array).

V první kapitole práce je uveden teoretický úvod k hardwarové implementaci na FPGA. Nejprve je popsáno FPGA a částí, ze kterých se skládá. Dále je popsán jazyk pro popis hardwaru VHDL (Very High Speed Integrated Circuit Hardware Description Language), který se využívá k programování FPGA, a nakonec postup hardwarové implementace od prvního návrhu až po nahrání výsledného programu do zařízení.

V druhé kapitole je provedena analýza dosavadních hardwarových implementací asymetrických kryptosystému. Přesněji jsou analyzovány hardwarové implementace RSA(Rivest–Shamir–Adleman) a algoritmů založených na využití eliptických křivek (ECDH - Elliptic-curve Diffie–Hellman, ECDSA - Elliptic Curve Digital Signature Algorithm).

Ve třetí kapitole je popsáno proměření dostupných hardwarových implementací, konkrétně jedna implementace modulárního mocnění a dvě implementace skalárního násobení bodu na EC (Elliptic Curve) a nakonec je provedeno jejich porovnání ve využití potřebných zdrojů.

Ve čtvrté kapitole je detailně popsán asymetrický kryptosystém Ed25519, který byl zvolen pro vlastní implementaci. Dále jsou zde popsány komponenty, které byly v rámci této práce vytvořeny.

V páté kapitole jsou uvedeny výsledky jednotlivých komponent, popsán postup ověření funkčnosti této implementace a nakonec porovnání s dosavadními implementacemi.

V poslední kapitole je popsáno jak probíhalo nasazení vlastní implementace na FPGA Virtex UltraScale+ za pomocí frameworku NDK (Netcope Development Kit).

1 Hardwarová implementace na FPGA

V této části je uveden teoretický úvod k hardwarové implementaci na FPGA. Nejprve je popsáno FPGA a částí, ze kterých se skládá. Dále je popsán jazyk VHDL, který se využívá k programování FPGA, a nakonec postup hardwarové implementace od prvního návrhu až po nahrání výsledného programu do zařízení.

1.1 Programovatelné hradlové pole

Programovatelné hradlové pole (FPGA) je polovodičové zařízení, které se skládá ze tří hlavních částí, a to konfigurovatelných logických bloků, programovatelných spojení a vstupně-výstupních bloků. Dále může obsahovat i další speciální bloky. Mezi výhody FPGA patří přeprogramovatelnost jejich funkcí po výrobě a možnost vykonávat velký počet paralelních operací [1].

Konfigurovatelné logické bloky – CLB

Každý blok se skládá z určitého počtu Slices. Tento počet se liší podle platformy, např. na platformě Spartan 6 se jeden blok skládá ze 2 Slices. Každý Slice se dále skládá z vyhledávací tabulky (LUT), pro reprezentaci logické funkce, klopných obvodů nebo registrů a může obsahovat i standardní logické bloky jako jsou multiplexory nebo sčítačky. Na obrázku 1.1 je zobrazen jeden CLB a jeho názorné rozložení částí, ze kterých se skládá. Počet komponentů v jednom CLB se liší podle druhů zařízení [1].

Vyhledávací tabulka – LUT

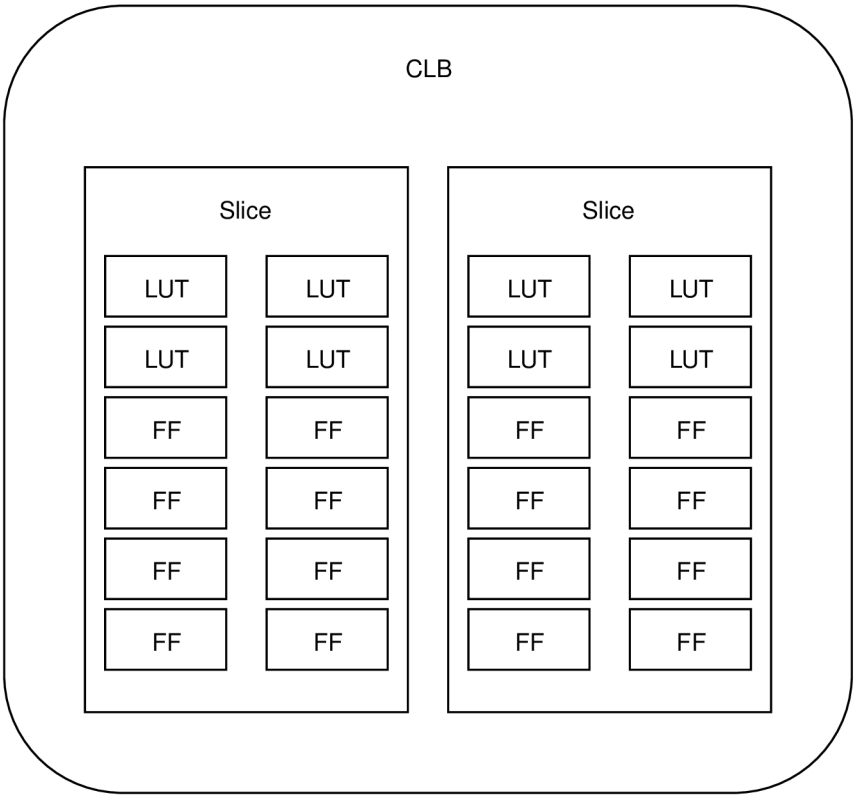
Vyhledávací tabulka obsahuje několik vstupů a jeden výstup. Pro každou kombinaci vstupů je možné naprogramovat jaký má být výstup. LUT je implementována jako blok RAM paměti, který je indexovaný pomocí vstupů. Na výstup je přivedena hodnota, která je uložená v paměti pro daný index [2].

Klopný obvod – Flip-Flop

Klopné obvody v FPGA platformě slouží jako paměti, čítače nebo děličky. Tyto klopné obvody jsou konfigurovatelné – může se specifikovat typ resetu (synchronní nebo asynchronní) nebo hodnota na kterou reset reaguje (logická 1 nebo logická 0). Nejčastěji se využívá klopný obvod typu D [2].

Slice

Slice je skupina LUT tabulek a klopných obvodů. Počet těchto prvků v každém Slice se liší podle platformy, např. na platformě Spartan 6 se jeden Slice skládá ze 4 LUT tabulek se 6 vstupy a 8 klopných obvodů viz obr. 1.1 [2].



Obr. 1.1: Ukázka jednoho CLB.

Speciální bloky

Moderní programovatelné hradlové pole často obsahuje i speciální bloky. Mezi ně patří:

- aritmeticko-logická jednotka (ALU),
- multiplexory,
- blokové paměti,
- digital signal processing (DSP) bloky.

Programovatelná spojení

Programovatelná spojení slouží k propojení logických a vstupně-výstupních bloků. Tyto propojení mohou být realizována například pomocí multiplexorů [1].

Vstupně-výstupní bloky – IOB

Vstupně-výstupní bloky se využívají pro komunikaci s externími zařízeními. Tyto bloky mohou být konfigurovány jako vstupní, výstupní nebo obousměrné. Vstupně-výstupní bloky zabírají největší část FPGA, proto je důležitý výběr standardů, které tyto bloky budou podporovat [1].

1.2 Jazyk VHDL

VHDL je jazyk, který se používá pro popisování hardwaru. Je navržen tak, aby splňoval potřeby při návrhů elektronických systémů. Umožňuje popis struktury obvodu, popis chování obvodu a simulaci obvodu, takže není kvůli každé změně potřeba vyrábět obvod nový [3].

Popis struktury

Při popisu struktury se popisují vstupy a výstupy obvodu. Závislost výstupů na vstupů se definuje až při popisu chování obvodu. Strukturu, kterou popisujeme pouze pomocí vstupů a výstupu, v jazyce VHDL nazýváme entitou a vstupy a výstupy nazýváme porty. Dále se popisuje, z jakých částí se obvod skládá. Každá část tohoto obvodu musí být nějaká entita. K propojení portů těchto entit dochází pomocí tzv. signálů [3].

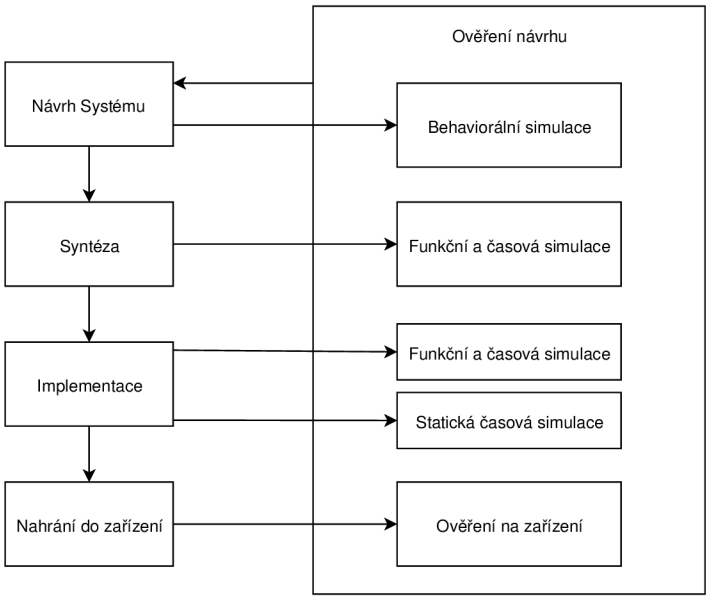
Popis chování

Po popisu struktury obvodu je nutné popsat, jak se tento obvod bude chovat. Využívá se v částech, kde není vhodné obvod popisovat strukturálně. Například při definování základních bloků obvodu jako jsou logická hradla. V tomto případě je nutné vytvořit funkci závislosti výstupů na vstupech [3].

1.3 Postup hardwarové implementace

Postup hardwarové implementace na FPGA se skládá ze tří fází. Nejprve se dělá návrh obvodů, poté následuje syntéza zdrojových souborů a na konec implementace, během které se vygeneruje výstupní soubor, který se nahraje do zařízení.

Na obrázku 1.2 je zobrazeno schéma postupu při návrhu hardwarové implementace. Je na něm znázorněn sled fází, které jsou dále blíže popsány, a možnosti ověření správnosti návrhu v každé fázi.



Obr. 1.2: Schéma postupu při návrhu hardwarové implementace.

Návrh systému

Návrh se provádí buď schematicky nebo pomocí jazyku pro popisování hardwaru (HDL), případně se může použít kombinace obou možností. Schématický návrh je vhodnější používat pouze u menších projektů, protože u větších projektů je složitější se v tomto návrhu orientovat.

V tomto kroku se může využít behaviorální simulace (také nazývána Register Transfer Level (RTL) simulace). Tato simulace je relativně rychlá, a proto se doporučuje ji využívat často k ověření funkčnosti návrhu a k nalezení chyb v logice obvodu [4].

Syntéza

V tomto kroku jsou zdrojové kódy přeloženy do reálných obvodů skládajících se z prvků jako jsou logické a klopné obvody. Výstupem syntézy je soubor netlist, který obsahuje list potřebných logických prvků a informace o jejich propojení. Během překladu se nejprve ověřuje správnost syntaxe a poté optimalizace pro dané zařízení. Při optimalizaci jsou odstraněny redundantní prvky.

Po syntéze se může provést funkční simulace, která se využívá k ověření správné funkce návrhu. Tato simulace by měla zahrnout všechny možné kombinace vstupů a stavů obvodů. Celá syntéza se provádí pomocí specializovaných nástrojů, např. Synopsys FPGA Compiler [4].

Implementace

Implementace se skládá ze tří kroků:

- překlad (Translate) – dochází ke spojení netlistů a omezení do jednoho souboru,
- mapování (Map) – obvod je rozdělen do dílčích bloků tak, aby se vešly do logických bloků FPGA,
- rozmístění a propojení (Place and Route) – dílčí bloky z předchozího kroku jsou vloženy do logických bloků podle omezení a propojeny mezi sebou.

Po těchto krocích je vygenerován výsledný soubor, který se nahrává do zařízení.

Dále je vhodné provést statickou časovou simulaci. Tato simulace poskytuje nejpřesnější obrázek o tom, jak se daný návrh bude chovat na reálném zařízení, protože již bere v potaz i zpoždění bloků. Nevýhodou této simulace je, že zabírá spoustu času [4].

2 Analýza dosavadních hardwarových implementací asymetrických kryptosystémů

Mezi nejčastěji využívané asymetrické kryptosystémy patří RSA a kryptografie na bázi eliptických křivek ECC. Při použití RSA se pracuje s velkými čísly, proto jsou hardwarové implementace tohoto algoritmu velice náročné. Z tohoto důvodů se častěji využívá ECC. V tabulce 2.1 je zobrazeno porovnání délek klíčů při použití RSA a ECC pro udržení stejné úrovně bezpečnosti. V této části jsou shrnuty dosavadní implementace asymetrických kryptografických schémat na FPGA.

Tab. 2.1: Porovnání délky klíče pro RSA a ECC při stejné úrovni bezpečnosti

RSA	1024	2048	3072	8192	15360
ECC	160	224	256	384	512

2.1 Hardwarová implementace RSA

Při šifrování a dešifrování pomocí RSA je potřebná pouze jedná operace, a to modulární mocnění. Proto je nejdůležitější se zaměřit na optimalizaci této operace.

Modulární mocnění

Modulární mocnění je pro velká čísla velice náročná operace, proto se nejčastěji zjednodušuje na sérii modulárního násobení a počítání druhých mocnin. Tento algoritmus se nazývá „square and multiply“. V tomto algoritmu se prochází exponent bit po bitu buď zleva doprava nebo zprava doleva a v závislosti na tom, jestli je daný bit logická jednička nebo logická nula provádíme operaci násobení nebo mocnění dvěma.

Dosavadní hardwarové implementace

Hardwarovou implementací RSA algoritmu se zabývali S. Kumar Sahu a M. Pradhan ve své práci [5] z roku 2011 a R. Verma, M. Dutta, R. Vig ve své práci [6] z roku 2013.

V práci [5] je prezentována implementace podporující šifrování a dešifrování pomocí klíče o velikosti 128 bitů, 256 bitů nebo 512 bitů. Pro modulární mocnění je použit Montgomeryho algoritmus, který využívá pouze bloky posunu a sčítání. Díky

tomu, že jsou tyto bloky opětovně použitelné, snižuje se množství potřebných zdrojů na FPGA. I přes toto snížení tato implementace jen pro 128bitový klíč potřebovala více než 100 % dostupných zdrojů na vybraném zařízení viz 2.2.

V práci [6] je modulární mocnění řešeno stejně jako v práci [5] pomocí Montgomeryho algoritmu. V této práci je však použita upravená verze tohoto algoritmu, díky čemu se můžou operace násobení a mocnění dvěma provádět paralelně. Pro šifrování a dešifrování je možné využít 512 nebo 1024bitový klíč. Architektura navržena v této práci [6] byla syntetizována a implementována na zařízení Virtex-5. Využití zdrojů pro tuto architekturu je uvedeno v tabulce 2.2.

Tab. 2.2: Výsledky syntézy prací [5] a [6] zaměřujících se na modulární mocnění

Práce	Zařízení	Velikost klíče	Slice	Flip Flop	LUT	Frekvence [MHz]
[5]	Spartan-3	128	2366	2943	4325	101,06
[6]	Virtex-5	512	5777	6773	6773	239,09
[6]	Virtex-5	1024	11898	13432	13437	229,49

2.2 Hardwarová implementace kryptografie na bázi eliptických křivek

Kryptografie na bázi eliptických křivek (ECC) je na zařízeních s omezenými zdroji častěji využívána, protože pro stejnou úroveň zabezpečení je potřeba menší velikosti klíčů, a tedy operace šifrování a dešifrování nejsou tak náročné. Při použití ECC je nejvíce využívanou operací násobení bodu, proto se na optimalizaci této operace zaměřuje velké množství publikací.

Při implementacích se nejčastěji používá křivka 25519, protože jsou výpočty na ni rychlejší, má menší délku klíče a je jednodušší pro implementaci než křivky standardizované Národním institutem standardů a technologie (NIST) [8].

Skalární násobení bodu na EC

Při násobení bodu na EC se využívá algoritmus „Montgomery ladder“. V tomto algoritmu se provádí sekvence sčítání dvou bodů (Point addition), dvojnásobení bodu (Point doubling) a prohazování bodů (Point swapping). Výhodou využití algoritmu „Montgomery ladder“ je to, že při správné implementaci všech operací trvá vždy konstantní čas, je tedy odolný proti útokům pomocí časové analýzy.

Při operaci sčítání dvou bodů je při použití afinních souřadnic potřeba dvou operací, a to násobení a dělení na konečném poli. Zatímco při použití projektivních souřadnic je potřeba pouze násobení. Proto se častěji pro implementaci využívají projektivní souřadnice, protože jejich použití snižuje množství potřebných zdrojů.

Dosavadní hardwarové implementace

Existuje velké množství prací zabývajících se hardwarovou implementací ECC. Tyto práce se zaměřují na implementace s využitím co nejmenšího množství zdrojů, implementací s nízkým zpožděním při generování sdíleného klíče, implementací s nízkou spotřebou či implementací podporující více křivek.

Hardwarové implementace s nízkým zpožděním

Implementací minimalizující časovou náročnost skalárního násobení bodu na ECC se zabývali P. Koppermann, F. De Santis, J. Heyszl, a G. Sigl ve své práci [7] z roku 2016 a R. Salarifard a S. Bayat-Sarmadi ve své práci [8] z roku 2019.

V práci [7] je pro jednu iteraci jejich implementace „Montgomery ladder“ algoritmu potřeba pouze 42 cyklů. Tato implementace podporuje na zařízení Xilinx Zynx-7030 frekvenci až 115 MHz. Díky tomu je možnost vygenerovat pomocí algoritmu ECDH sdílený klíč za méně než 120 μs viz tab. 2.3.

V práci [8] je pro jednu iteraci jejich implementace „Montgomery ladder“ algoritmu potřeba pouze 11 cyklů. Pro implementaci navrhli „Karatsuba-Ofman-based field multiplier“ architekturu. Maximální dosaženou frekvencí na zařízení Xilinx Zynx-7030 je 87 MHz. Pro vygenerování sdíleného klíče s využitím ECDH algoritmu je potřeba jen 44 μs viz tab. 2.3.

Tab. 2.3: Výsledky syntézy prací [7] a [8] zaměřujících se na skalární násobení bodu na EC s nízkým zpožděním

Práce	Slice	Flip flop	LUT	Frekvence [MHz]	Počet cyklů	Doba vykonávání [μs]	Spotřeba stat./dyn. [mW]
[7] X25519	8639	21107	26483	115	13639	118	150/789
[8] X25519	3362	2705	12989	87	3858	44	-

Hardwarové implementace podporující křivky Ed25519 a X25519

Na implementaci podporující křivky Ed25519 a X25519 se zaměřili F. Turan a I. Verbauwhede ve své práci [9] z roku 2018 a M. Ali Mehrabi a Ch. Doche ve své práci [10] z roku 2019.

V Práci [9] je představena první hardwarová implementace ECDSA na křivce Ed25519. Je zde popsána architektura, která kombinuje ECDSA algoritmus založený na křivce Ed25519 a vytvoření sdíleného klíče pomocí křivky X25519. Účelem této práce bylo prozkoumat vhodnost implementace těchto dvou algoritmů na zařízeních s omezenými zdroji. V tabulce 2.4 jsou uvedeny výsledky jejich implementace. Práce [10] se primárně zaměřuje na minimalizaci spotřeby. Ke snížení náročnosti hardwarové implementace je v této práci navržen algoritmus prokládaného modulárního násobení (interleaved modular multiplication). Pomocí této architektury byla výrazně snížena dynamická spotřeba na 145mW oproti předchozím implementacím viz tab. 2.3 a tab. 2.4.

Tab. 2.4: Výsledky syntézy prací [9] a [10] zaměřujících se na skalární násobení bodu na křivkách Ed25519 a X25519

Práce	Flip flop	LUT	Frekvence [MHz]	Doba vykonávání [μs]	Spotřeba stat./dyn. [mW]
[9] Ed25519	2656	11148	82	1467	-
[9] X25519	962	2707	105	608	-
[10] Ed25519	3472	8680	137,5	628	107/172
[10] X25519	3411	7380	137,5	512	103/145

Hardwarové implementace ECDSA

Celkovou hardwarovou implementací algoritmu ECDSA se zabývali A. Abidi, B. Bouallegue, F. Kahri ve své práci [11] z roku 2014 a B. Glas, O.Sander, V. Stuckert, K. D. Müller-Glaser a J. Becker ve své práce [12] z roku 2011.

V práci [11] je pro skalární násobení bodu na EC využit algoritmus „Montgomery ladder“, který podporuje 163bitovou EC. Dále je zde implementována hashovací funkce SHA-256 pro generování podpisu a generátor náhodných čísel, který se využívá při generování soukromého klíče a při generování podpisu. Tato implementace pracuje pouze v režimu podepisování a maximální možná frekvence je 94MHz viz tab. 2.5.

V práci [12] jsou implementovány všechny bloky potřebné pro ECDSA. Funguje jak pro generování, tak i pro ověřování podpisu. Jako hashovací funkci využívá SHA-256. Tato implementace je navržena pro dvě různé eliptické křivky, a to secp224r1 a secp256r1, podporuje tedy 224 a 256bitové operace. Maximální možná frekvence této implementace je 50MHz viz tab. 2.5. Tato implementace umožňuje také využití pouze modulu pro generování podpisu, při čemž může dosahovat rychlosti generování až 184 podpisů za sekundu ověřování podpisu, při čemž může dosahovat rychlosti ověřování až 140 ověření za sekundu při využití křivky secp256r1 viz tab. 2.6.

Tab. 2.5: Výsledky syntézy prací [11] a [12] zaměřujících se na ECDSA

Práce	Flip flop	LUT	BUFG	Frekvence [MHz]
[11] ECDSA 163bitová	15939	20647	3	94
[12] ECDSA 256bitová	-	69120	0	50

Tab. 2.6: Rychlost generování a ověřování podpisu v práci [12]

Činnost	Doba vykonávání [ms]	Počet za sekundu
generování podpisu secp224r1	5,45	184
ověřování podpisu secp224r1	7,17	140
generování podpisu secp256r1	7,15	140
ověřování podpisu secp256r1	9,09	110

2.3 Ochrana proti útokům na hardwarové implementace

Hardwarové implementace jakýchkoliv kryptografických schémat mohou poskytovat informace o soukromých klíčích při využití útoků postranními kanály. Příkladem útoků pomocí postranních kanálů může být měření spotřebovávané energie (např. proudová analýza) nebo měření času potřebného pro vykonání algoritmu.

Například u implementace modulárního mocnění s využitím algoritmu *square and multiply* se prochází exponent (soukromý klíč) bit po bitu a v závislosti na tom, jestli je hodnota aktuálního bitu log 0 nebo log 1 vykonává jiný počet operací. Při hodnotě bitu log 0 i log 1 se provádí operace mocnění dvěma, ale jen při hodnotě bitu log 1 se vykonává i operace násobení. Proto bude na základě hodnoty soukromého klíče v každém kroku algoritmu prováděn jiný počet operací a může být možné pomocí časové analýzy zjistit hodnotu soukromého klíče.

Existuje více možností ochrany. Využívají se jak softwarové, tak i hardwarové. Mezi softwarové patří například úprava algoritmu tak, aby se hodnota klíče maskovala nějakou náhodnou hodnotou. Hardwarovou ochranou může být třeba vyhlazování využití energie nebo změnou úrovní tranzistorové logiky. Nicméně je velice náročné tyto opatření aplikovat bez podpory výrobců, viz [13].

2.4 Shrnutí

V této kapitole byl analyzován stav dosavadních návrhů pro hardwarovou implementaci asymetrických kryptosystémů. V první části byly analyzovány práce zaměřující se na implementaci modulárního mocnění potřebného pro RSA. Z tabulky 2.2 je vidět, jak s rostoucí velikosti klíče se přímo úměrně zvyšuje množství potřebných zdrojů, proto tyto implementace nejsou vhodné pro větší klíče.

V druhé části byly analyzovány práce popisující implementace operací potřebných pro ECC. Analyzované práce se zabývají především optimalizací implementace skalárního násobení bodu na EC. Nejlepších výsledků pro implementaci s nízkým zpožděním bylo dosaženo v práci [8]. Jejich návrh potřebuje pro vygenerování sdílené klíče s využitím ECDH pouze 3858 cyklů, což na testovaném zařízení Xilinx Zynx-7030 odpovídá 44 μs.

Analyzované implementace je těžké mezi sebou přímo porovnávat, protože jsou implementovány různé kryptografické schémata podporující různé velikosti klíčů, které zaručují jiné úrovně bezpečnosti. Dále se tyto implementace zaměřují na různé optimalizace - některé na optimalizované využití zdrojů, některé na minimální zpoždění nebo maximální propustnost.

3 Měření dostupných hardwarových implementací asymetrických kryptosystémů

V této kapitole bude proveden rozbor veřejně dostupných hardwarových implementací asymetrických kryptosystémů. Konkrétně implementace modulárního mocnění potřebného pro RSA[1], která je volně dostupná pod licencí LGPL, implementace operací potřebných pro ECC[2] a implementaci algoritmu ECDSA[3], obě volně dostupné pod licencí GNU-GPL3.0 . Dále ověření správné funkčnosti pomocí simulací a zjištění využití potřebných zdrojů pomocí syntézy.

Pro simulaci i syntézu dostupných implementací byl využit program Vivado 2019.1 od firmy Xilinx. Simulace byla prováděna pro zařízení rodiny Artix-7. U každé implementace je uvedeno využití hardwarových zdrojů (Flip Flop, LUT), maximální možná frekvence a statická a dynamická spotřeba.

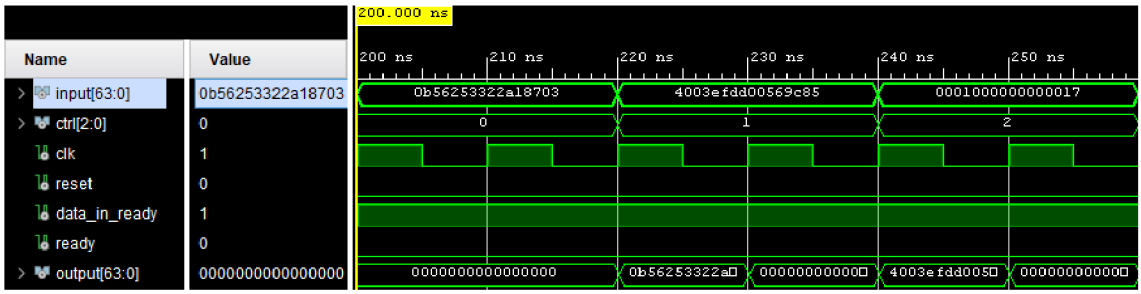
3.1 Hardwarová implementace modulárního mocnění

Tato implementace je napsána v jazyku VHDL a podporuje 32, 64 a 512bitové modulární mocnění. Aby bylo možné v další části porovnat výsledky mezi touto implementací a implementacemi využívajícími eliptické křivky byla tato implementace upravena tak, aby podporovala i 1024bitové modulární mocnění. Základem pro provedení mocnění je Montgomeryho algoritmus stejně jako v práci [5]. Pro zprovoznění implementace je potřeba přidat existující IP (Intellectual Property) blok „Block Memory Generator“, který se v implementaci používá pro uchování vstupů modulárního násobení.

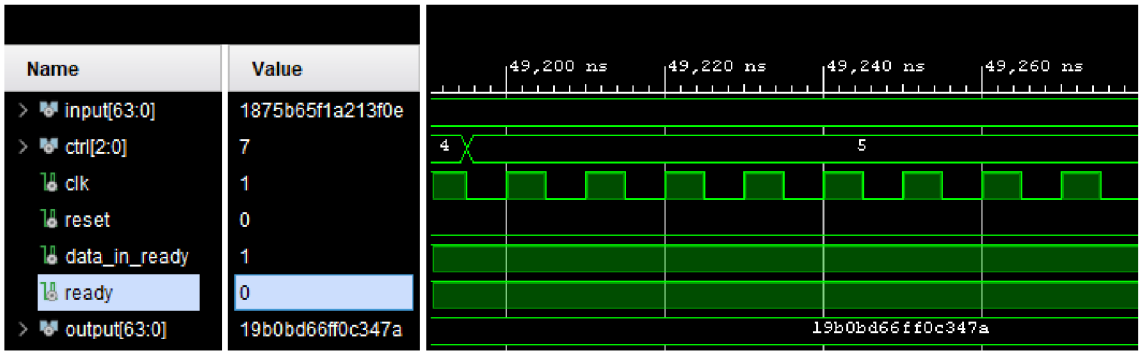
Pomocí simulace byla ověřena správnost této implementace pro všechny tři bitové velikosti vstupních čísel. Na obrázku 3.1 je zobrazena ukázka nastavení vstupů. Nejprve se nahraje na vstup (signál „input“) základ mocniny, poté modulo a nakonec exponent. Potom probíhá výpočet výstupu, který je nakonec uložen do signálu "output". Hodnotu výstupu je možné vidět na obrázku 3.2. Z obrázků 3.1 a 3.2 je vidět, že celý výpočet trval přibližně 49 μs, při použití hodinového signálu s frekvencí 100MHz.

V tabulce 3.1 jsou uvedeny výsledky syntézy této implementace. Z využití zdrojů lze vidět, že se zvětšující se bitovou velikostí vstupních hodnot se přímo úměrně zvyšuje počet potřebných zdrojů.

[1]Dostupné z: `https://opencores.org/projects/mod_mult_exp`
[2]Dostupné z: `https://github.com/mmoraless/ecc_vhdl`
[3]Dostupné z: `https://github.com/lennartbublies/ecdsa/`



Obr. 3.1: Ukázka simulace načtení vstupů pro modulární mocnění.



Obr. 3.2: Ukázka simulace výstupu modulárního mocnění.

Tab. 3.1: Výsledky syntézy hardwarové implementace modulárního mocnění

Implementace	Flip flop	LUT	Frekvence [MHz]	Spotřeba stat./dyn. [mW]
32bitová	149	280	128,2	131/44
64bitová	280	428	103,1	131/51
512bitová	2082	3281	27,0	132/119
1024bitová	4140	6478	14, 95	133/135

3.2 Hardwarová implementace operací na EC

Autorem této implementace je Miguel Morales-Sandoval, který na ní pracoval od roku 2004 do roku 2008. Celá implementace je napsaná v jazyku VHDL a obsahuje moduly pro aritmetické operace s konečnými poli, aritmetické operace na EC a skalární násobení bodu na EC. Operace podporují tři velikosti a to 113, 131 a 163bitové. Tato implementace využívá na rozdíl od většiny ostatních afinní souřadnice bodů. K této

implementaci nebyl žádný testovací soubor a neexistuje ani žádná dokumentace, a proto nebylo možné ověřit správnost pomocí simulace.

V tabulce 3.2 jsou uvedeny výsledky syntézy implementací skalárního násobení na EC. Byly proměřeny všechny tři podporované velikosti. Z tabulky je vidět, že pro 163bitovou implementaci bylo využito přibližně stejné množství zdrojů, jako u 512bitové implementace modulárního mocnění, ale při porovnání s 1024bitovou implementací zabírá pouze polovinu potřebných zdrojů viz tab. 3.1. Jak bylo uvedeno již v druhé kapitole, 163bitová implementace ECC odpovídá úrovni bezpečnosti při použití 1024bitové implementace RSA viz tab. 2.1. Tedy pro stejnou úroveň bezpečnosti je potřeba pouze polovina zdrojů než u implmenetace modulárního mocnění.

Tab. 3.2: Výsledky syntézy hardwarové implementace skalárního násobení na EC

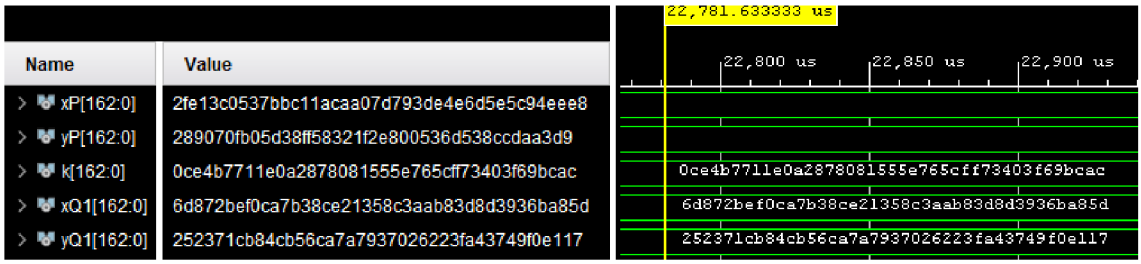
Implementace	Flip flop	LUT	Frekvence [MHz]	Spotřeba stat./dyn. [mW]
113bitová	1755	1840	157,3	109/48
131bitová	2038	1993	184,9	131/72
163bitová	2530	2615	200,0	131/104

3.3 Hardwarová implementace ECDSA

Autoři této implementace jsou Leander Schulz a Lennart Bublies. Na této implementaci ECDSA, napsané v jazyce VHDL, pracovali od roku 2017 do roku 2018. Implementace podporuje různé eliptické křivky nad konečnými binárními poli, je potřeba pouze definovat parametry vybrané křivky. Jejich zdroje obsahují implementace základních bloků potřebných aritmetických operací na konečném poli, operací na EC, ale také bloků z nich složených pro implementaci algoritmů využívaných v ECDSA (generování klíče, generování podpisu, ověření podpisu atd.).

Ověření správnosti bylo provedeno na křivce sect163k1 definované organizací NIST. Byla ověřena správnost modulu pro generování klíče, modulu hashovací funkce SHA-256, modulu pro výpočet skalárního násobení bodu na EC. Na obrázku 3.3 je možné vidět ukázku ze simulace pro ověření skalárního násobení bodu na EC. Signály „xP“ a „yP“ jsou souřadnice bodu, který se má násobit, signál „k“ udává kolikrát se tento bod bude násobit a signály „xQ1“ a „yQ1“ jsou souřadnice výsledného bodu.

V tabulce 3.3 jsou uvedeny výsledky syntézy bloku pro generování podpisu a ověření podpisu, bloku pro generování klíče, bloku hashovací funcke SHA-256



Obr. 3.3: Ukázka simulace modulu pro skalární násobení bodu na EC.

a bloku pro skalární násobení bodu. Syntéza byla provedena při nastavení parametrů pro křivku sect163k1. Z výsledků syntézy je vidět, že implementace skalárního násobení bodu v tomto projektu je srovnatelná s výsledky skalárního násobení 163bitové implementace práce od Miguela Morales-Sandovala a že implementace bloku pro generování podpisu je méně náročná na potřebné zdroje než implmentace bloku pro ověření podpisu.

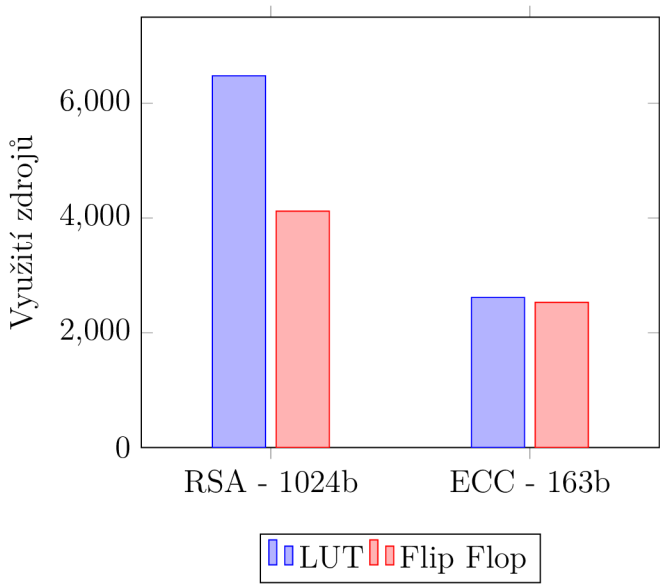
Tab. 3.3: Výsledky syntézy hardwarové implementace operací na EC

Implementace	Flip flop	LUT	Frekvence [MHz]	Spotřeba stat./dyn. [mW]
generování podpisu	3682	3335	162,08	71/172
ověření podpisu	7545	7884	168,15	71/248
generování klíče	2495	2854	168,15	71/104
SHA-256	2838	2612	90,66	71/84
násobení bodu na EC	2492	2664	168,15	71/94

3.4 Shrnutí

V této kapitole byly syntézovány tři volně dostupné implementace a výsledky byly prezentovány v tabulkách. Z těchto výsledků je možné vidět, že hardwarová implementace modulárního mocnění podporující 1024bitové čísla využívá přibližně dvakrát větší množství zdrojů než obě hardwarové implementace skalárního násobení bodu na EC podporující 163bitovou EC s odpovídající úrovní bezpečnosti. Dále byla pomocí simulací ověřena správnost hardwarové implementace modulárního mocnění a modulů implementace ECDSA.

V grafu na obrázku 3.4 je uvedeno porovnání využitých zdrojů hardwarové implementace modulárního mocnění potřebného pro RSA a skalárního násobení bodu na EC potřebného pro ECC při stejné úrovni bezpečnosti. Z tohoto porovnání implementací modulárního mocnění a operací na EC se dá říct, že z hlediska využitých zdrojů je vhodnější implementovat asymetrický kryptosystém založený na EC.



Obr. 3.4: Porovnání využití zdrojů RSA a ECC při stejné úrovni bezpečnosti.

4 Vlastní hardwarová implementace asymetrického kryptosystému

Na základě analýzy dosavadních návrhů hardwarových implementací asymetrických kryptosystémů uvedené v kapitole 2 a měření dostupných hardwarových implementací uvedených v kapitole 3 byl jako vhodný pro vlastní hardwarovou implementaci vybrán asymetrický kryptosystém ECDSA, konkrétněji Ed25519, což je ECDSA, které využívá jako hashovací funkci SHA-512 a Edwardsovu křivku 25519.

4.1 ECDSA

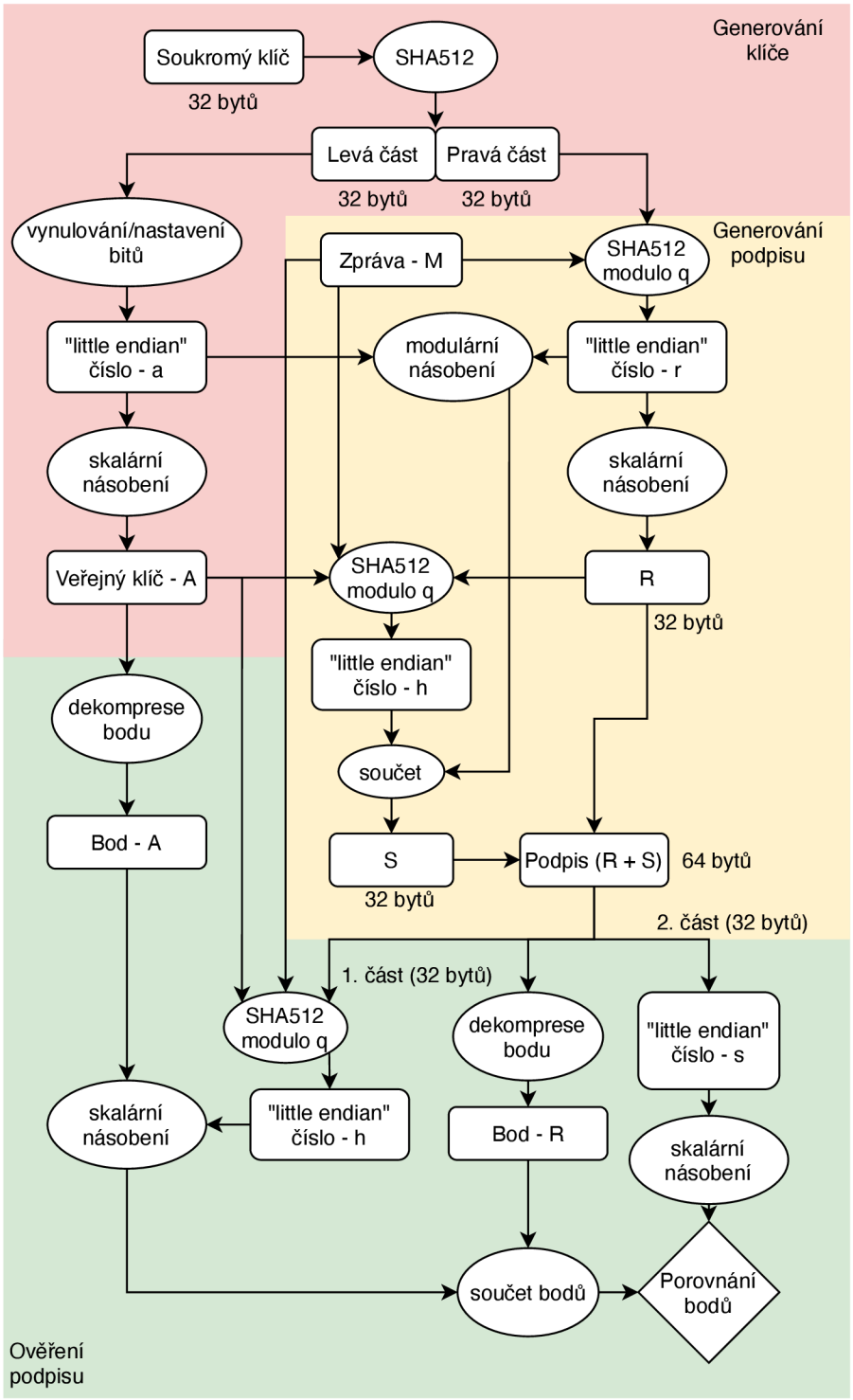
ECDSA je variantou asymetrického kryptografického schématu DSA využívající eliptické křivky. Pro implementaci ECDSA je potřeba implementovat komponenty pro operace na eliptických křivkách (skalární násobení bodu, sčítání dvou bodů a dvojnásobení bodu), pro které je také potřeba aritmetika nad konečnými poli, dále hashovací funkci, která je potřeba při generování podpisu a nakonec komponentu pro generování náhodných čísel, která je potřebná pro vygenerování soukromého klíče.

Ed25519

Jak bylo výše zmíněno, tato práce se bude dále zabývat konkrétně implementací Ed25519. Výhodou Ed25519 je to, že generování podpisu je deterministické, nemůže se tak stát jako u běžného ECDSA, že během generování podpisu se příjde na to, že pro náhodně zvolené číslo nelze podpis vygenerovat a musí se zvolit nějaké jiné. Další výhodou je využití Edwardsovy křivky, pro které platí, že operace nad křivkou jsou definovány pro všechny body, nemusí se tedy při implementaci nijak řešit bod v nekonečnu, jako u eliptických křivek.

Z blokového schématu na obrázku 4.1 je vidět postup generování a následného ověření podpisu. Nejprve je ze soukromého klíče vygenerován veřejný klíč a poté jsou oba tyto klíče využity během generování podpisu. Pro generování je dále nutná zpráva, která má být podepsána. S využitím hashovací funkce SHA-512 a operací na eliptické křivce je vygenerován výsledný podpis. Tento podpis je pak společně se zprávou odeslán a příjemce pak s pomocí veřejného klíče může ověřit, jestli je zpráva skutečně od daného odesílatele a jestli nebyla nějak pozměněna.

Během generování i ověřování podpisů se využívá komprese a dekomprese bodů. Při kompresi je bod uložen jako 32bytové číslo, které reprezentuje y souřadnici bodu a nejvýznamější bit je nastaven na *1* nebo *0* podle znaménka bodu x. Při dekompresi jsou souřadnice bodu z těchto 32bytů obnoveny.



Obr. 4.1: Blokové schéma Ed25519.

Algoritmus generování veřejného klíče

1. Ze soukromého klíče se pomocí hashovací funkce SHA-512 vygeneruje hash a zjeho prvních 32bytů se vygeneruje číslo a.
2. Vypočte se bod A jako skálární součin $A = a \cdot G$.
3. Tento bod A se kompresuje na 32bytů a výsledek je vygenerovaný veřejný klíč.

Algoritmus generování podpisu

1. Ze soukromého klíče se pomocí hashovací funkce SHA-512 vygeneruje hash a z jeho prvních 32bytů se vygeneruje číslo a a zbylých 32bytů je využito později jako *prefix*.
2. Vypočte se bod A jako skálární součin $A = a \cdot G$. Tento bod je využit jako veřejný klíč.
3. Vypočte se bod R jako skalární součin $R = r \cdot G$, kde se r vygeneruje pomocí hashovací funkce z *prefixu* a zprávy.
4. Vypočte se hash h z bodů A, R a zprávy.
5. Vypočte se 32bytové číslo s jako součet čísla a a výsledku modulárního násobení čísel r a h.
6. Výsledný podpis vznikne spojením R a s.

Algoritmus ověření podpisu

1. Z veřejného klíče se získá pomocí dekomprese bod A, pokud se bod nepodaří získat, je podpis neplatný.
2. Z prvních 32 bytů podpisu se získá pomocí dekomprese bod R, pokud se bod nepodaří získat, je podpis neplatný.
3. Ze zbylých 32 bytů podpisu se se získá číslo s a je vypočten hash h z prvních 32 bytů podpisu, veřejného klíče a zprávy.
4. Vypočtou se body sB, hA, pomocí skalárního násobení $sB = s \cdot G$, $hA = h \cdot A$
5. Porovná se bod sB a součet bodů R a hA, pokud se tyto body rovnají, je podpis platný, jinak neplatný

Edwardsova křivka 25519

Tato křivka je definována nad konečným prvočíselným polem $F_{2^{255}-19}$. Rovnice Edwardsových křivek je $ax^2 + y^2 = 1 + dx^2y^2$. Konkrétní parametry Edwardsovy křivky 25519 jsou uvedeny v tabulce 4.1, viz [16].

Tab. 4.1: Parametry Edwardsovy křivky 25519

```
a = -1
p = 0x 7FFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF
 FFFFFFFF FFFFFFED
d = 0x 52036CEE 2B6FFE73 8CC74079 7779E898 00700A4D 4141D8AB
 75EB4DCA 135978A3
x = 0x 216936D3 CD6E53FE C0A4E231 FDD6DC5C 692CC760 9525A7B2
 C9562D60 8F25D51A
y = 0x 66666666 66666666 66666666 66666666 66666666 66666666
 66666666 66666658
```

4.2 Hardwarové implementace komponent

V této části budou podrobněji popsány jednotlivé komponenty, které byly implementovány. Tyto komponenty byly implementovány v jazyku VHDL ve vývojovém prostředí Vivado 2019.1 od firmy Xilinx. Implementace komponent byly optimalizované pro parametry využívané křivky. K ověření správnosti implementací jednotlivých komponent byly vytvořeny testovací soubory a pro ověření celkové implementace byly využity testovací vektory uvedené v kapitole 5.1.

Modulární násobení

Pro modulární násobení $z = x \cdot y \ mod \ p$ byl využit algoritmus Montgomeryho násobení, při kterém se vypočte $Z = X \cdot Y \cdot R^{-1} \ mod \ p$. Čísla Z, X, Y jsou čísla v Montgomeryho doméně, proto je nejprve potřeba převést vstupní čísla x, y do této domény a nakonec výstupní číslo Z zpět do celočíselné domény. Pro tyto převody může být opět využit algoritmus Montgomeryho násobení. Číslo R je konstanta, která se nazývá "Montgomery radix", a musí být volena tak, aby největší společný dělitel *gcd(R, M)* byl roven 1. Je doporučeno volit $R = 2^n$, kde n je počet bitů čísla M.

Tento algoritmus je vhodný pro zařízení s omezenými zdroji, protože všechny mezivýsledky v průběhu tohoto algoritmu potřebují maximálně o 1 bit paměti více než je velikost vstupů. Další výhodou tohoto algoritmu je, že při vhodném zvolení parametrů využívá pouze operace součtu a posunu.

Modulární dělení

Další potřebnou operací pro implementaci je modulární dělení $z = \frac{x}{y} \, mod \, p$. Vstupem této komponenty jsou tedy tři čísla x, y a p a výstupem číslo z. Modulární dělení bylo implementováno pomocí hardwarového algoritmu uvedeného v práci [15], který je založený na rozšířeném algoritmu pro hledání největšího společného dělitele.

Sčítání dvou bodů a dvojnásobení bodu

Pro sčítání dvou bodů byla využita metoda podle doporučení uvedeném v [16]. Tedy body nejsou reprezentovány pomocí tradičních afinních souřadnic $(x, \, y)$, ale pomocí rozšířených homogenních souřadnic $(X, \, Y, \, Z, \, T)$, kde $x = \frac{X}{Z}$, $y = \frac{Y}{Z}$, $x \cdot y = \frac{T}{Z}$. Pro převod z afinních souřadnic se zvolí $Z = 1$, potom se $X = x$ a $Y = y$ a jediné co je potřeba dopočítat je $T = x \cdot y$.

Výhodou tohoto řešení je, že je pro výpočet součtu dvou bodů potřeba méně operací než při výpočtech v afinních souřadnicích.

Pro výpočet dvojnásobení bodu byla využita komponenta na sčítání dvou bodů, na jejiž vstupy je přiveden dvakrát stejný bod.

Skalární násobení bodu

Jak již bylo zmíněno v předchozích kapitolách, skalární násobení bodu na EC je u kryptosystému využívajících EC nejčastěji využívané a proto je potřeba, aby bylo optimalizované. Proto v této práci bylo skalární násobení bodu implementováno pomocí algoritmu "Montgomery Ladder".

Implementovaná komponenta má jako vstup souřadnice bodu P a číslo k, výstupní bod $Q = k \cdot P$. Na začátku algoritmu je bod Q nastaven na bod generátoru EC a bod R je uložen do pomocného signálu R. Poté se prochází číslo k bit po bitu v každém kroku se provádí operace sčítání bodu Q a R a v závislosti na tom jestli je hodnota aktuálního bitu čísla k log 0 nebo log 1 se dvojnásobí bod Q respektive R. Díky tomu, že se v každém kroku, nezávisle na hodnotě bitu čísla k, provádí vždy stejný počet operací, je při správné implementaci těchto operací, tento algoritmus odolný proti útokům postranními kanály s využitím časové analýzy.

Ve výpisu 4.1 je ukázka využití komponent pro sčítání dvou bodů a dvojnásobení bodu. Je možné vidět, že na vstup komponenty *point_addition* jsou vždy přivedeny body Q a R a výstup je uložen do pomocného signálu. Který je v závislosti na hodnotě aktuálního bitu k uložen do Q nebo R. U komponenty *point_doubling* vždy záleží na aktuální hodnotě bitu k, proto jsou na vstup i výstup přivedeny pomocné signály.

SHA-512

Pro hashovací funkci SHA-512 byla využita volně dostupná komponenta[1] pod licencí MIT. Tato komponenta byla dále optimalizována a tak se snížil počet využívaných zdrojů. Porovnání původní a upravené verze je uvedeno v tabulce 4.2.

Tab. 4.2: Porovnání využití zdrojů SHA-512 komponent

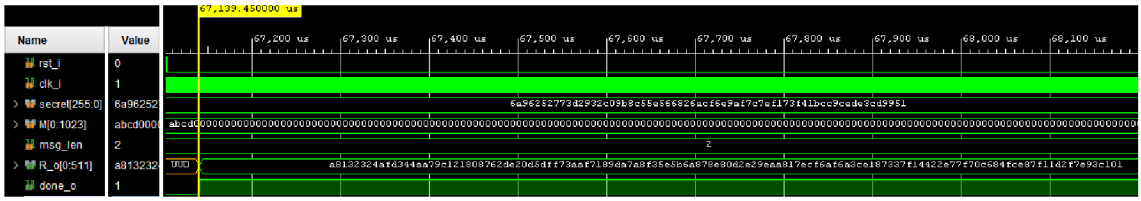
Implementace	LUT	Flip flop	BRAM
SHA-512 - původní	14356	5330	0
SHA-512 - upravená	3028	1094	1

Generování veřejného klíče

Komponenta pro generování veřejného klíče byla vytvořena podle algoritmu uvedeného v podkapitole 4.1. Tato komponenta potřebuje jako vstup pouze soukromý klíč, z kterého vygeneruje 32bytový veřejný klíč.

Generování podpisu

Pro generování podpisu je potřeba zadat soukromý klíč, který je náhodné 32bytové číslo, a zpráva, která má být podepsána. Dále je uveden algoritmus generování podpisu a na obrázku 4.2 je zobrazena ukázka simulace této komponenty. Generování podpisu probíhá podle algoritmu uvedeného v podkapitole 4.1.



Obr. 4.2: Ukázka generování podpisu.

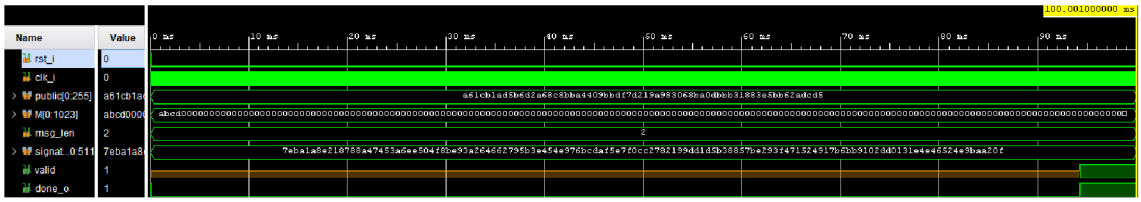
[1]Dostupné z: `https://github.com/dsaves/SHA-512`

Výpis 4.1: Ukázka využití komponent u skalárního násobení

```
addition: entity work.point_addition port map (
    rst_i => rst_add,
    clk_i => clk_i,
    Xp_i => Qx,
    Yp_i => Qy,
    Zp_i => Qz,
    Tp_i => Qt,
    Xq_i => Rx,
    Yq_i => Ry,
    Zq_i => Rz,
    Tq_i => Rt,

    Xr_o => X_add,
    Yr_o => Y_add,
    Zr_o => Z_add,
    Tr_o => T_add,
    done_o => done_add
);

doubling: entity work.point_addition port map (
    rst_i => rst_double,
    clk_i => clk_i,
    Xp_i => Xd,
    Yp_i => Yd,
    Zp_i => Zd,
    Tp_i => Td,
    Xq_i => Xd,
    Yq_i => Yd,
    Zq_i => Zd,
    Tq_i => Td,

    Xr_o => X_double,
    Yr_o => Y_double,
    Zr_o => Z_double,
    Tr_o => T_double,
    done_o => done_double
);
```

Ověření podpisu

Pro ověření podpisu je potřeba zadat veřejný klíč, který je vygenerovaný ze soukromého klíče a má také 32 bytů, zprávu a její podpis, který má být ověřený. Dále je uveden algoritmus generování podpisu a na obrázku 4.3 je zobrazena ukázka simulace této komponenty. Ověření podpisu probíhá podle algoritmu uvedeného v podkapitole 4.1.



Obr. 4.3: Ukázka ověření podpisu.

Pomocné komponenty

Dále byly vytvořeny pomocné komponenty a to pro kompresi a dekompresi bodu, komponenta pro přípravu zprávy pro komponentu SHA-512 a komponenta pro výpočet modula.

4.3 Shrnutí

V této části byly popsány komponenty, které byly implementovány v rámci této práce. Tyto komponenty jsou dostupné v příloze této práce. V další kapitole jsou uvedeny informace o způsobu ověření správnosti těchto komponent a výsledky proměření těchto komponent.

5 Ověření a výsledky vlastní implementace

Ověření a měření využití zdrojů jednotlivých komponent implementace probíhalo pomocí vývojového prostředí Vivado 2017.4.1 od společnosti Xilinx.

5.1 Ověření hardwarové implementace

Pro ověření komponent byly napsány testovací soubory. Při každé změně byla využívána behaviorání simulace, protože jak bylo uvedeno v první kapitole 1, tato simulace je velice rychlá a dá se tak jednoduše ověřit správnost změn.

5.1.1 Testovací vektory

Pro ověření byly využity testovací vektory uvedeny v práci [16] a dále na vlastních testovacích vektorech, které byly ověřeny pomocí knihovny v jazyce Python. Ukázka těchto testovacích vektorů je uvedena v tabulce 5.1. Testovací vektory jsou uvedeny v hexadecimálním formátu a je vždy zobrazen soukromý klíč, který byl využit, podepisovaná zpráva a její výsledný podpis.

Tab. 5.1: Testovací vektory pro ověření vlastní implementace

Soukromý klíč:	Soukromý klíč:
9d61b19deffd5a60ba844af492ec2cc4	4ccd089b28ff96da9db6c346ec114e0f
4449c5697b326919703bac031cae7f60	5b8a319f35aba624da8cf6ed4fb8a6fb
Zpráva (délka 0 bytů):	Zpráva (délka 128 bytů):
	1784ac38a00e12506717257c44424d30
Podpis:	e14de768e5961139fe2e426db1b60f39
e5564300c360ac729086e2cc806e828a	e7ceb15f6630ec3aedc8a4270aa12d74
84877f1eb8e5d974d873e06522490155	809a351e56ceea3dbd20e951a42425da
5fb8821590a33bacc61e39701cf9b46b	1784ac38a00e12506717257c44424d30
d25bf5f0595bbe24655141438e7a100b	e14de768e5961139fe2e426db1b60f39
	e7ceb15f6630ec3aedc8a4270aa12d74
Soukromý klíč:	809a351e56ceea3dbd20e951a42425da
1f6c069a6b5bc2f793aae139694648ba	
aa7ece8a856a735df0aa071ee9ea67c1	Podpis:
	36c3af2f5c873295c5eb42d4dd8eabd3
Zpráva (délka 0 bytů):	bdcf5b5315ac529858c10111904f8869
	98d358583d58ff345fddc9e47645fa7d

Podpis:
c7847f6b06584a44c5daa868dff50da5
4c241b142549aac226eba7d0eed20ac4
d011494b6216f9ba74fd0997affb9786
4ef92a7f1d03e667fe52fefd37a7e105

Soukromý klíč:
4ccd089b28ff96da9db6c346ec114e0f
5b8a319f35aba624da8cf6ed4fb8a6fb

Zpráva (délka 1 byte):
72

Podpis:
92a009a9f0d4cab8720e820b5f642540
a2b27b5416503f8fb3762223ebdb69da
085ac1e43e15996e458f3613d0f11d8c
387b2eaeb4302aeeb00d291612bb0c00

Soukromý klíč:
c5aa8df43f9f837bedb7442f31dcb7b1
66d38535076f094b85ce3a2e0b4458f7

Zpráva (délka 2 byty):
af82

Podpis:
6291d657deec24024827e69c3abe01a3
0ce548a284743a445e3680d7db5ac3ac
18ff9b538d16f290ae67f760984dc659
4a7c15e9716ed28dc027beceea1ec40a
2f1a74160906ce8514ed7dee176c2b00

Soukromý klíč:
833fe62409237b9d62ec77587520911e
9a759cec1d19755b7da901b96dca3d42

Zpráva (délka 64 bytů):
ddaf35a193617abacc417349ae204131
12e6fa4e89a97ea20a9eeee64b55d39a
2192992a274fc1a836ba3c23a3feebbd
454d4423643ce80e2a9ac94fa54ca49f

Podpis:
dc2a4459e7369633a52b1bf277839a00
201009a3efbf3ecb69bea2186c26b589
09351fc9ac90b3ecfdfbc7c66431e030
3dca179c138ac17ad9bef1177331a704

Soukromý klíč:
6a96252773d2932c09b8c55e566826ac
f6e9af7c7ef173f41bcc9cede3cd9951

Zpráva (délka 2 byty):
abcd

Podpis:
a8132324afd344aa79c121808762de20
d5dff73aaf7189da7a8f35e5b6a878e8
0d2e29eaa817ecf6af6a3ce187337f14
422e77f70c684fce87f11d2f7e93c101

5.2 Výsledky syntézy hardwarové implementace

V tabulce 5.2 jsou uvedeny výsledky syntézy jednotlivých komponent. Tyto výsledky byly získány pomocí syntézy pro zařízení Virtex UltraScale+. Pro každou komponentu je uvedeno její využití zdrojů - počet využitých vyhledávacích tabulek a klopných obvodů, dále maximální frekvence, při které může daná komponenta pracovat a statická a dynamická spotřeba této komponenty.

Tab. 5.2: Výsledky syntézy vlastní hardwarové implementace

Implementace	LUT	Flip flop	Frekvence [MHz]	Spotřeba stat./dyn. [mW]
montogomeryho násobení	1168	783	468,8	1738/116
modulární dělení	4781	1833	351,1	1721/345
sčítání bodů	7608	3105	330,5	1748/290
násobení bodu	17427	8546	301,7	1751/755
SHA-512	3028	1094	308,5	1738/226
generování veřejného klíče	25830	12317	307.3	1731/1153
generování podpisu	31024	16706	307.3	1750/1420
ověření podpisu	45420	24762	207,2	1729/1030

Pro vygenerování podpisu je potřeba 665932 cyklů, pro ověření podpisu je potřeba 933014 cyklů. Z těchto čísel vyplývá, že je možné generovat 462 podpisu za sekundu a ověřovat 222 podpisů za sekundu viz tabulka 5.3.

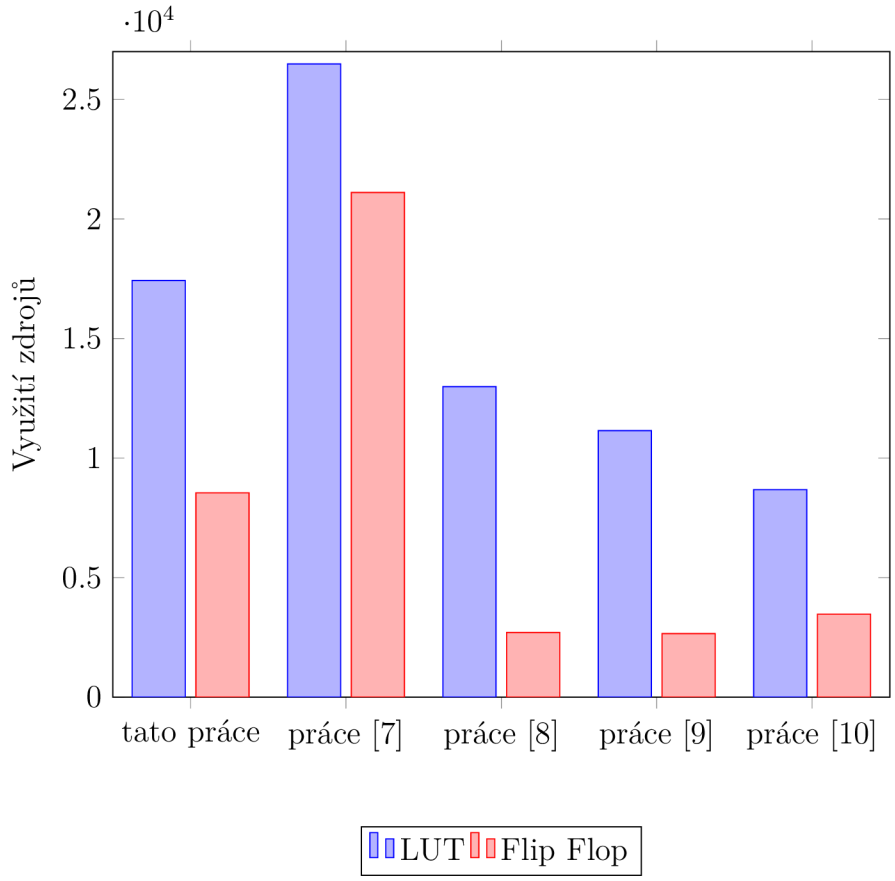
Tab. 5.3: Rychlost generování a ověřování podpisů vlastní implementace

Operace	Počet cyklů	Počet za sekundu
generování podpisu	665932	462
ověření podpisu	933014	222

5.3 Porovnání s dosavadními hardwarovými implementacemi

Jak již bylo uvedeno v kapitole 2 porovnání hardwarových implementací kryptografických schémat je náročné. Nejlépe jde porovnat výsledky této práce s prácemi

[7], [8], [9] a [10], které se zabývaly implementací skalárního násobení bodu na EC pro stejnou křivku jako v této práci, tedy 25519. V grafu na obrázku 5.1 je zobrazeno porovnání využitých zdrojů mezi těmito prácemi. Při porovnání zdrojů je vidět, že práce [7] má využití největší s 26483 vyhledávacími tabulkami a 21107 klopnými obvody, zatímco práce [10] má využití nejmenší s 3472 vyhledávacími tabulkami a 8680 klopnými obvody. Implementace v rámci této práce je někde mezi těmito implementacemi s 17427 vyhledávacími tabulkami a 8546 klopnými obvody. Co se týče maximálni frekvence a spotřeby tyto implementace nelze přímo porovnávat, protože výsledky se budou výrazně lišit kvůli měření pro různá zařízení.



Obr. 5.1: Porovnání využití zdrojů pro skalární násobení bodu na EC.

6 Nasazení a ověření hardwarové implementace na FPGA Virtex UltraScale+

Nasazení hardwarové implementace na FPGA probíhalo pomocí nástroje NDK firmy Netcope Technologies, a.s. Na FPGA byly nasazeny a ověřeny komponenty pro generování podpisu a pro ověřování podpisu.

6.1 Nasazení hardwarové implementace na FPGA

Nasazení probíhalo na FPGA Virtex UltraScale+. Pro nasazení bylo potřeba komponenty zabalit do aplikačního jádra. Z tohoto důvodu byly vytvořeny další dvě komponenty *application_sign.vhd* a *application_verify.vhd*, které toto zabalení zajišťují. Z těchto komponent byly pomocí vivada vygenerované výstupní soubory, které byly poté nahrány na FPGA.

6.1.1 Zařízení Virtex UltraScale+

Jak bylo výše uvedeno, nasazení probíhalo na FPGA Virtex UltraScale+, konkrétní typ xcvu7p-flvb2104-2-i. Níže je uvedena tabulka 6.1 zobrazuje specifikace využitého FPGA čipu. Tyto specifikace byly získány z Vivada a ze stránek firmy xilinx. [1]

Tab. 6.1: Specifikace FPGA Virtex UltraScale+

Název	Počet
Speed grade	-2
LUTs	788160
Flip-Flops	1576320
BRAMs	1440
I/O Pin Count	2104
Available IOBs	702

6.1.2 NDK - Netcope Development Kit

NDK je konfigurovatelný framework, který se využívá při vývoji síťových aplikací na FPGA. Tento framework zajišťuje pomocí komunikačního rozhraní komunikaci

[1] Dostupné z: `https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf`

mezi hardwarovou části běžící na FPGA a softwarovou částí běžící na hostitelském počítači, což umožňuje výměnu dat mezi těmito částmi a ovládání hardwarové aplikace z hostitelského počítače [17].

Níže je zobrazen výpis 6.1 výstupu příkazu *ndktool info*, který vypíše info o kartě po tom co na kartu byla nahrána komponenta pro generování podpisu.

Výpis 6.1: Výstup příkazu ndktool info

```
SOFTWARE:
---------
Version:         3.0.2

FIRMWARE:
---------
Version:         3.0.1
Name:            NDK_NIC_200G2QL_100G2
Built at:        2020/06/06 13:14:46
Interfaces:      2
RX queues:       32 (out of 64)
TX queues:       8 (out of 16)

HARDWARE:
---------
Board:           NFB-200G2QL
Serial number:   0028
NUMA master:     0
NUMA slave:      n/a
```

Sběrnice MI32

Sběrnice MI32 je komunikační sběrnice pro přenášení dat menší rychlostí. Jejím hlavním učelem je konfigurace hardwarových komponent ze softwaru. Podporuje čtení i zápis podle zadané adresy [17].

6.1.3 Nahrání implementace na FPGA

Nahrání vygenerovaného souboru probíhalo pomocí příkazu *ndktool reload --file=<název souboru>*. Poté již byla komponenta na FPGA připravena a mohlo se pomocí příkazů *ndktool core --read* a *ndktool core --write* číst a zapisovat hodnoty z, respektive do jeho registrů, pomocí výše zmíněné sběrnice MI32.

6.2 Ověření hardwarové implementace na FPGA

Pro ověření byly vytvořeny dva skripty, které využívají příkazy *ndktool core --read* a *ndktool core --write* pro čtení a zápis hodnot do FPGA. Pomocí nich byla ověřena správná funkčnost komponenty pro generování podpisu a komponenty pro ověření podpisu. Komponenta pro generování podpisu pracovala s frekvencí 200MHz a komponenta pro ověření podpisu s frekvencí 150MHz.

Ukázka výstupu testovacích skriptů je zobrazena níže ve výpisech 6.2 a 6.3. Celý výstup těchto testovacích skriptů je uveden v příloze A a B. Ve výpisech 6.3 a B je "Help signal", což je pomocný 32bytový signál, kde 1. bit je nastaven podle toho jestli byl podpis ověřen do log 1 jinak do log 0 a 2. bit je nastaven ve chvíli, kdy je ověření hotovo. Tedy hodnota "00000003", která je ve výpisech zobrazena znamená, že ověření bylo dokončeno a podpis byl ověřen.

Výpis 6.2: Ukázka výstupu testu komponenty pro generování podpisu

```
[root@dellemc dobias]# ./test_sign.sh
Secret: 4ce85a5b48937e402bf4921a51051b45
69e03444faa68c67f67fb6ec6a2ce87c
Message: cfdb7d4ba74dd85f8c046449afc8d3fb
00000000000000000000000000000000
Generated signature: 6aa3e99ee568bfc203b07310413668b9
879782921a2ee50a1dbf47f411b8364b
b7316a46e35a4e4c024e3374f9d17e6e
cea2126414c4095626b774221c181f01
Test successful
```

Výpis 6.3: Ukázka výstupu testu komponenty pro ověření podpisu

```
[root@dellemc dobias]# ./test_verify.sh
Message: cfdb7d4ba74dd85f8c046449afc8d3fb
00000000000000000000000000000000
Public: 97be514522ac230572c7dae7baf091d8
ce699ac4cf974d134e7206ea844771cd
Signature: 6aa3e99ee568bfc203b07310413668b9
879782921a2ee50a1dbf47f411b8364b
b7316a46e35a4e4c024e3374f9d17e6e
cea2126414c4095626b774221c181f01
Help signal: 00000003
Test successful
```

Závěr

Cílem bakalářské práce bylo seznámení se s postupy hardwarové implementace na platformě FPGA, dále analýza dostupných implementací asymetrických kryptografických schémat a vybrat a implementovat vhodné asymetrické kryptografické schéma na platformě FPGA.

Nejprve tedy bylo v teoretické části postupně popsáno zařízení FPGA, programovací jazyk VHDL a postup hardwarové implementace od prvního návrhu systému až po nahrání výsledného programu do zařízení.

Poté byl v kapitole 2 analyzován stav dosavadních návrhů pro hardwarovou implementaci asymetrických kryptosystémů. A protože u žádné z analyzovaných prací nebyly dostupné žádné zveřejněné zdrojové soubory, byly poté v kapitole 3 blíže rozebrány a proměřeny tři volně dostupné implementace asymetrických kryptosystémů. Byl zde i pokus jednotlivé implementace porovnat, ale jak již bylo napsané ve shrnutí kapitoly 2 toto porovnání je velice náročné.

Na základě analýz a teoretických znalostí z kryptografie bylo jako vhodné asymetrické kryptografické schéma pro vlastní hardwarovou implementaci vybráno Ed25519. Toto schéma bylo v rámci této práce úspěšně implementováno a komponenty, které byly vytvořeny, byly blíže popsány v kapitole 4.

V další kapitole 5 pak byly uvedeny výsledky syntézy komponent vytvořených v rámci této práce a to hlavně pro komponentu generování veřejného klíče, kde byl počet využitých vyhledávacích tabulek 25830 a klopných obvodů 12317, pro komponentu generování podpisu, kde byl počet využitých vyhledávacích tabulek 31024 a klopných obvodů 16706, a pro komponentu ověření podpisu, kde byl počet využitých vyhledávacích tabulek 45420 a klopných obvodů 24762. Dále bylo uvedeno porovnání komponenty pro výpočet skalárního násobení bodu na EC s prácemi analyzovanými v kapitole 2.

V rámci poslední kapitoly byl popsán postup nasazení vytvořené hardwarové implementace na FPGA Virtex UltraScale+ s využitím NDK firmwaru. Nakonec se úspěšně podařilo na FPGA nahrát komponenty pro generování i ověření podpisu a také ověřit jejich správnou funkci při pracovní frekvenci 200MHz u generování podpisu a 150MHz u ověřování podpisu.

Literatura

[1] *Basic FPGA Architecture and its Applications* [online]. [cit. 18. 10. 2019]. Dostupné z URL: <https://www.watelectronics.com/fpga-architecture-applications>.

[2] *How does an FPGA work?* [online]. 17.1.2018 [cit. 4. 11. 2019]. Dostupné z URL: <https://alchitry.com/blogs/tutorials/how-does-an-fpga-work>.

[3] Peter J. Ashenden. *The VHDL Cookbook* 1990, Dept. Computer Science University of Adelaide, South Australia. Dostupné z URL: <https://www.ics.uci.edu/~alexv/154/VHDL-Cookbook.pdf>.

[4] *The Ultimate Guide to FPGA Design Flow* [online]. 17.2.2019 [cit. 18. 10. 2019]. Dostupné z URL: <http://hardwarebee.com/ultimate-guide-fpga-design-flow/>.

[5] Sahu Sushanta, Pradhan Manoranjan. *FPGA Implementation of RSA Encryption System* 2011, International Journal of Computer Applications. 19. 10-12. 10.5120/2391-3173.

[6] Rupali Verma, Maitreyee Dutta, Renu Vig. *FPGA Implementation of Modified Montgomery for RSA Cryptosystem* January 2013, International Journal of Computer Science and Telecommunications, vol. 4, no. 1.

[7] Philipp Koppermann, Fabrizio De Santis, Johann Heyszl, and Georg Sigl. *X25519 hardware implementation for low-latency applications* in 2016 Euromicro Conference on Digital System Design (DSD), 2016 Euromicro Conference on. IEEE, 99–106.

[8] Salarifard, Raziyeh, and Siavash Bayat-Sarmadi. *An Efficient Low-Latency Point-Multiplication Over Curve25519* 2019, IEEE Transactions on Circuits and Systems I: Regular Papers.

[9] Turan, Furkan, and Ingrid Verbauwhede. *Compact and Flexible FPGA Implementation of ED25519 and X25519* 2019, ACM Transactions on Embedded Computing Systems, 18(3), 1-21.

[10] Mehrabi, Mohamad Ali, and Christophe Doche. *Low-Cost, Low-Power FPGA Implementation of ED25519 and CURVE25519 Point Multiplication* 2019, Information 10.9 (2019): 285.

[11] Abidi, Abdessalem, Belgacem Bouallegue, and Fatma Kahri. *Implementation of elliptic curve digital signature algorithm (ECDSA)* 2014, Global Summit on Computer & Information Technology (GSCIT). IEEE, 2014.

[12] Glas, Benjamin, et al. *Prime field ECDSA signature processing for reconfigurable embedded systems* 2011, International Journal of Reconfigurable Computing 2011 (2011): 5.

[13] Wollinger, Thomas, Jorge Guajardo, and Christof Paar. *Security on FPGAs: State-of-the-art implementations and attacks* 2003, ACM Transactions on Embedded Computing Systems (TECS) 3.3 (2004): 534-574.

[14] Michael Mühlberghuber. *Comparing ECDSA Hardware Implementations based on Binary and Prime Fields* 2011, Institute for Applied Information Processing and Communications Graz University of Technology Inffeldgasse 16a 8010 Graz, Austria

[15] Kaihara, M.E. and Takagi. *A hardware algorithm for modular multiplication/-division* 2005, IEEE Transactions on Computers, 54(1), pp.12-21.

[16] Josefsson, Simon, and Ilari Liusvaara. *Edwards-curve digital signature algorithm (EdDSA)* 2017, Internet Research Task Force, Crypto Forum Research Group, RFC. Vol. 8032.

[17] Netcope Technologies, a.s. *NETCOPE DEVELOPMENT KIT - Firmware Developer's Manual* Brno, 2018

Seznam symbolů, veličin a zkratek

CLB	Konfigurovatelný logický blok - Configurable Logic Block
EC	Eliptická křivka - Elliptic Curve
ECC	Kryptografie na bázi eliptických křivek - Elliptic Curve Cryptography
ECDH	Elliptic-curve Diffie–Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
FF	Klopný obvod - Flip Flop
FPGA	Programovatelné hradlové pole - Field Programmable Gate Array
IOB	Vstupně-výstupní blok - Input-Output Block
LUT	Vyhledávací tabulka - LookUp Table
NDK	Netcope Development Kit
NIST	Národní institut standardů a technologie - National Institute of Standards and Technology
RSA	Rivest–Shamir–Adleman
VHDL	Very High Speed Integrated Circuit Hardware Description Language

Seznam příloh

A Výstup testovacího skriptu pro generování podpisu

```
[root@dellemc dobias]# ./test_sign.sh
Secret: 4ce85a5b48937e402bf4921a51051b45
69e03444faa68c67f67fb6ec6a2ce87c
Message: cfdb7d4ba74dd85f8c046449afc8d3fb
00000000000000000000000000000000
Generated signature: 6aa3e99ee568bfc203b07310413668b9
879782921a2ee50a1dbf47f411b8364b
b7316a46e35a4e4c024e3374f9d17e6e
cea2126414c4095626b774221c181f01
Test successful

Secret: 023206c481154eeebd6ba5f609b05a83
e19cddd880a70488725c2ee4eedf6066
Message: 05bb3f9cfc32c1659d3c0762714793e1
00000000000000000000000000000000
Generated signature: 123c2210157ad1ed696f24c867f7a21b
21301347386a01a682d87ffcbd835835
8c19111473c5981cf19d05ff1132ea85
fb96a77f1cf9ff156aa9bf3d798acf0e
Test successful

Secret: d9d170fc80fd7dbb09e25d622373da72
d3983b251712f5c73c55d2d3cbae5bff
Message: 189493711b822b73a73a487e8ff64812
00000000000000000000000000000000
Generated signature: 1866b325e8f756cc2918a3872b855a4d
f22289c66d1d49570f02be64024c6123
d212fc4928d2c438158b84bdbbbd6551
91e6c66c80043de61c5b1c212712b10e
Test successful

Secret: b2a15e75da57b245fbc906fb408021ba
036d3932491215b577dc981a95c5bcf5
Message: 50e589b9a1762944bd84d66f0c8c282c
00000000000000000000000000000000
```

```
Generated signature: 3d34e1df2c74a959397064de30f32e9e
8fce0ca406d84e11ad24cf9b2a04385a
557c930c3096b91bf16cb4e146fd9099
c5c2397e5c3edb8b9dc9b6d725de030e
Test successful

Secret: bab72c0891447e5f6aa8fa7630b3f645
d28633fdbf5836e8f13f4d8b0cb00fb6
Message: 5dd4d76fafa70c90af1386cc0eb98609
00000000000000000000000000000000
Generated signature: 0ea761c98a9cf20ccad0972ce4c4f973
36a13af3b0c3d6c36e4449fdaa4541c1
60050c5c43f7866b2fcf852ab30dc731
8b6b01b094b64f397b85f23edfa4b80d
Test successful

Secret: 619cf85b455ba407e2ad83f2fca8c6cb
a17504c0159b098f74c30386969301ce
Message: 82d9fa7d9df95090cb936e09e54f9da7
00000000000000000000000000000000
Generated signature: ab7972665931e1591259ab3703fe0edd
c6e5302665e4cfa0d689f86c38eac811
9f38357a129361d33584696c932e442b
f966712c177d82e23a716335025dce08
Test successful

Secret: 398070663e486775e03a9d75235aa907
39a0f2f7953a8bc0b91897767fb7085d
Message: 6ea3c4189ab63e46d6ed2455143b0607
00000000000000000000000000000000
Generated signature: 68a000b2b9570111d143e2d6c0d955ee
b8cb2a88e8da1aacd1a40ad64955265b
8e3318235c7fcbce6bacd7f249ea5b6b
2d7d969e1f554f6bb92d061ff6f75d04
Test successful

Secret: 4061cab6d1a62a782235f26ab012448b
72dc6a247f559c713203553cd0f925aa
Message: 094430d389e4a7a87fbcdbe429eb8665
```

```
00000000000000000000000000000000
Generated signature: b38bec119733ecef404a7e361884a757
60238b1da956356e92ce52481bd586ce
4c2f8e0e9b6f0d1af410b910a5d7c9a0
dfab4d65fd4ab48855bf4947ca560805
Test successful
```

B Výstup testovacího skriptu pro ověření podpisu

```
[root@dellemc dobias]# ./test_verify.sh
Message: cfdb7d4ba74dd85f8c046449afc8d3fb
00000000000000000000000000000000
Public: 97be514522ac230572c7dae7baf091d8
ce699ac4cf974d134e7206ea844771cd
Signature: 6aa3e99ee568bfc203b07310413668b9
879782921a2ee50a1dbf47f411b8364b
b7316a46e35a4e4c024e3374f9d17e6e
cea2126414c4095626b774221c181f01
Help signal: 00000003
Test successful

Message: 05bb3f9cfc32c1659d3c0762714793e1
00000000000000000000000000000000
Public: 792c255d41976434cad0f047df8e230c
bc0b45f7295d6841c8655e662fd355a7
Signature: 123c2210157ad1ed696f24c867f7a21b
21301347386a01a682d87ffcbd835835
8c19111473c5981cf19d05ff1132ea85
fb96a77f1cf9ff156aa9bf3d798acf0e
Help signal: 00000003
Test successful

Message: 189493711b822b73a73a487e8ff64812
00000000000000000000000000000000
Public: f5b39d667e68f064ba880d789065cf98
55b9a54a09c195cf1a3afd90d5fb1704
Signature: 1866b325e8f756cc2918a3872b855a4d
f22289c66d1d49570f02be64024c6123
d212fc4928d2c438158b84bdbbbd6551
91e6c66c80043de61c5b1c212712b10e
Help signal: 00000003
Test successful

Message: 50e589b9a1762944bd84d66f0c8c282c
```

```
00000000000000000000000000000000
Public: f63aaf786fb66f806e0b583a084c4430
2d8a252d4b4254bb01fe45020e761171
Signature: 3d34e1df2c74a959397064de30f32e9e
8fce0ca406d84e11ad24cf9b2a04385a
557c930c3096b91bf16cb4e146fd9099
c5c2397e5c3edb8b9dc9b6d725de030e
Help signal: 00000003
Test successful

Message: 5dd4d76fafa70c90af1386cc0eb98609
00000000000000000000000000000000
Public: 934b9550587e17a63af17c59c139c4a2
aacb454156819820ecf4080ffd2f4965
Signature: 0ea761c98a9cf20ccad0972ce4c4f973
36a13af3b0c3d6c36e4449fdaa4541c1
60050c5c43f7866b2fcf852ab30dc731
8b6b01b094b64f397b85f23edfa4b80d
Help signal: 00000003
Test successful

Message: 82d9fa7d9df95090cb936e09e54f9da7
00000000000000000000000000000000
Public: 114d5fbab60fbeed6926b397122d7fc1
8c9bcf38db900d53d3855723703d0f15
Signature: ab7972665931e1591259ab3703fe0edd
c6e5302665e4cfa0d689f86c38eac811
9f38357a129361d33584696c932e442b
f966712c177d82e23a716335025dce08
Help signal: 00000003
Test successful

Message: 6ea3c4189ab63e46d6ed2455143b0607
00000000000000000000000000000000
Public: 411246e878c0f6f7c1d0beaa11245ce1
3a4024e523992ba370b21079f7fe4347
Signature: 68a000b2b9570111d143e2d6c0d955ee
b8cb2a88e8da1aacd1a40ad64955265b
8e3318235c7fcbce6bacd7f249ea5b6b
```

```
2d7d969e1f554f6bb92d061ff6f75d04
Help signal: 00000003
Test successful

Message: 094430d389e4a7a87fbcdbe429eb8665
00000000000000000000000000000000
Public: a6a4e05ff9ff0debbeb8f0c6e4dc4b21
04ff59b0b8890f5f697e9f443b5fcde9
Signature: b38bec119733ecef404a7e361884a757
60238b1da956356e92ce52481bd586ce
4c2f8e0e9b6f0d1af410b910a5d7c9a0
dfab4d65fd4ab48855bf4947ca560805
Help signal: 00000003
Test successful
```

C Obsah přiloženého CD

- / ..... kořenový adresář přiloženého CD
  - Asymetrická kryptografie na FPGA.pdf ..... text závěrečné práce práce
  - src ..... adresář obsahující zdrojové soubory hlavních modulů
    - application_sign.vhd
    - application_verify.vhd
    - public_key_generation.vhd
    - sign.vhd
    - verify.vhd
    - src ..... adresář obsahující zdrojové soubory
      - ecdsa_package.vhd
      - division.vhd
      - modulo.vhd
      - montgomery_multiplication.vhd
      - point_addition.vhd
      - point_compress.vhd
      - point_multiplication.vhd
      - recover_x.vhd
      - sha_512_core.vhd
      - sha_512_pkg.vhd
      - sha_input_prepare.vhd
      - square_and_multiply.vhd
      - test ..... adresář obsahující testovací soubory
        - division_tb.vhd
        - montgomery_multiplication_tb.vhd
        - point_addition_tb.vhd
        - point_multiplication_tb.vhd
    - test ..... adresář obsahující testovací soubory hlavních modulů
      - public_key_generation_tb.vhd
      - sign_tb.vhd
      - verify_tb.vhd
      - testrom.vhd
      - test_sign.sh
      - test_verify.sh