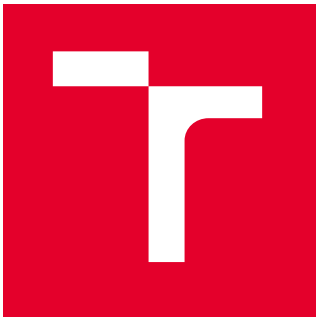


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

GENEROVÁNÍ NÁHODNÝCH ČÍSEL NA PLATFORMĚ FPGA

RANDOM NUMBER GENERATION ON FPGA PLATFORM

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Miroslav Písek**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Patrik Dobiáš**

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Miroslav Písek ***ID:*** 229096

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

NÁZEV TÉMATU:

Generování náhodných čísel na platformě FPGA

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s postupy hardwarové implementace na platformě FPGA. Analyzujte dostupné implementace a metody generování náhodných čísel na FPGA. Dále nastudujte a porovnejte metody testování náhodných čísel. Na základě analýzy vyberte vhodné metody generování náhodných čísel pro platformy FPGA.

Cílem bakalářské práce je hardwarová implementace bezpečného generátoru náhodných čísel na platformě FPGA a funkční testovací softwarová implementace, která umožní ověření náhodnosti čísel generovaných na FPGA. Dílčím cílem je rovněž testování kvality dostupných generátorů, a to pomocí používaných testů (např. Diehard a NIST STS).

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

Vedoucí práce: Ing. Patrik Dobiáš

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

ABSTRAKT

Bakalářská práce se zabývá implementací hardwarového generátoru náhodných čísel na FPGA obvodu. Snaží se přiblížit základní architekturu a možnou konfiguraci FPGA. V teoretické části jsou dále popsány principy generátorů náhodných čísel a poté statistické testy, skrze které tyto generátory hodnotíme. V následující části jsou shrnuty a analyzovány dosavadní generátory určené pro platformy FPGA. Dále probíhá bližší měření volně dostupných TRNG. V konečné části je popis implementace vlastního generátoru náhodných čísel. Ověřovaní náhodnosti probíhá skrze sadu testů NIST STS.

KLÍČOVÁ SLOVA

FPGA, VHDL, generátor pravých náhodných čísel, kruhové oscilátory, testování náhodnosti, NIST STS.

ABSTRACT

The bachelor's thesis deals with the implementation of a hardware random number generator on FPGA platform. It tries to explain the basic architecture and possible FPGA configuration. In the theoretical part, the principles of random number generators are further described and then the statistical tests through which we evaluate these generators. In the following section, the existing generators designed for FPGA platforms are summarized and analyzed. A closer measurement of freely available TRNG is also underway. In the final part, there is a description of the implementation of the own random number generator. Randomness verification takes place through the NIST statistical test suite.

KEYWORDS

FPGA, VHDL, true random number generator, ring oscillators, randomness testing, NIST STS.

PÍSEK, Miroslav. *Generování náhodných čísel na platformě FPGA*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 63 s. Bakalářská práce. Vedoucí práce: Ing. Patrik Dobiáš

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Miroslav Písek
VUT ID autora:	229096
Typ práce:	Bakalářská práce
Akademický rok:	2022/23
Téma závěrečné práce:	Generování náhodných čísel na platformě FPGA

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.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Patriku Dobiášovi za vedení, konzultace, trpělivost, návrhy a úpravy této práce. Neopomenou nelze rodinu, která mě při vytváření práce a studiu vždy podporovala.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Jádrem bakalářské práce je technologie FPGA (Field Programmable Gate Array) a generování náhodných čísel. Hradlová pole jsou s touto problematikou zvláště atraktivní, nabízí efektivnost (rychlost) a rekonfiguraci. V první kapitole je čtenář seznámen právě s problematikou programovatelných hradlových polí. Jejich architekturou, složitým procesem konfigurace, který zahrnuje volbu jazyka pro popisu hardwaru, různé typy simulací a závěrečnou implementaci do cílového FPGA čipu.

V dnešní době, kdy se rozšiřuje počet lidí využívající zařízení, která jsou schopna komunikace přes mobilní sítě nebo internet, je stále důležité klást větší důraz na soukromí. To zahrnuje zabezpečení jednotlivých dat. K ochraně se využívají kryptografické procesy, například pro šifrování je nutné vygenerovat náhodný klíč. Druhá teoretická kapitola je zaměřena na tuto problematiku, která souvisí s generátory náhodných čísel. Představeny jsou nároky na náhodné sekvence, druhy generátorů, jejich principy. Pojednáno je o náhodných číslech zejména z pohledu kryptografie.

Dílčím cílem je rovněž představit metody pro testování kvality generátorů, a to pomocí používaných testů. V práci je rozebrán soubor testů instituce NIST. Komplexní testy odhalí mezery v implementaci náhodného generátoru a řeknou, ve kterém testu byla náhodnost zamítnuta.

V následující, pořadí čtvrté, kapitole jsou vysvětleny možnosti implementací generátorů náhodných čísel. Názorně jsou popsány na konkrétních návrzích, které jsou následně analyzovány a krátce shrnuty.

V další kapitole jsou prakticky zkoumány volně dostupné implementace. Měření probíhalo na vývojové desce obsahující FPGA čip. TRNG jsou ověřeny skrze baterii testů NIST. Dále je hodnocena jejich náročnost na hardwarové zdroje.

Šestá kapitola je z hlediska této práce nejvýznamnější. V prvé řadě je charakterizováno cílové zařízení. Následuje vlastní implementace generátoru náhodných čísel. Návrh je založen na principu kruhových oscilátorů. Popsány jsou jednotlivé části návrhu a následné propojení s počítačem.

Na závěr bakalářské práce jsou popsané vlastnosti implementace. Dochází k jejímu ověření skrze statistické testy a je srovnávána s podobnými pracemi.

1 Postupy hardwarové implementace na platformě FPGA

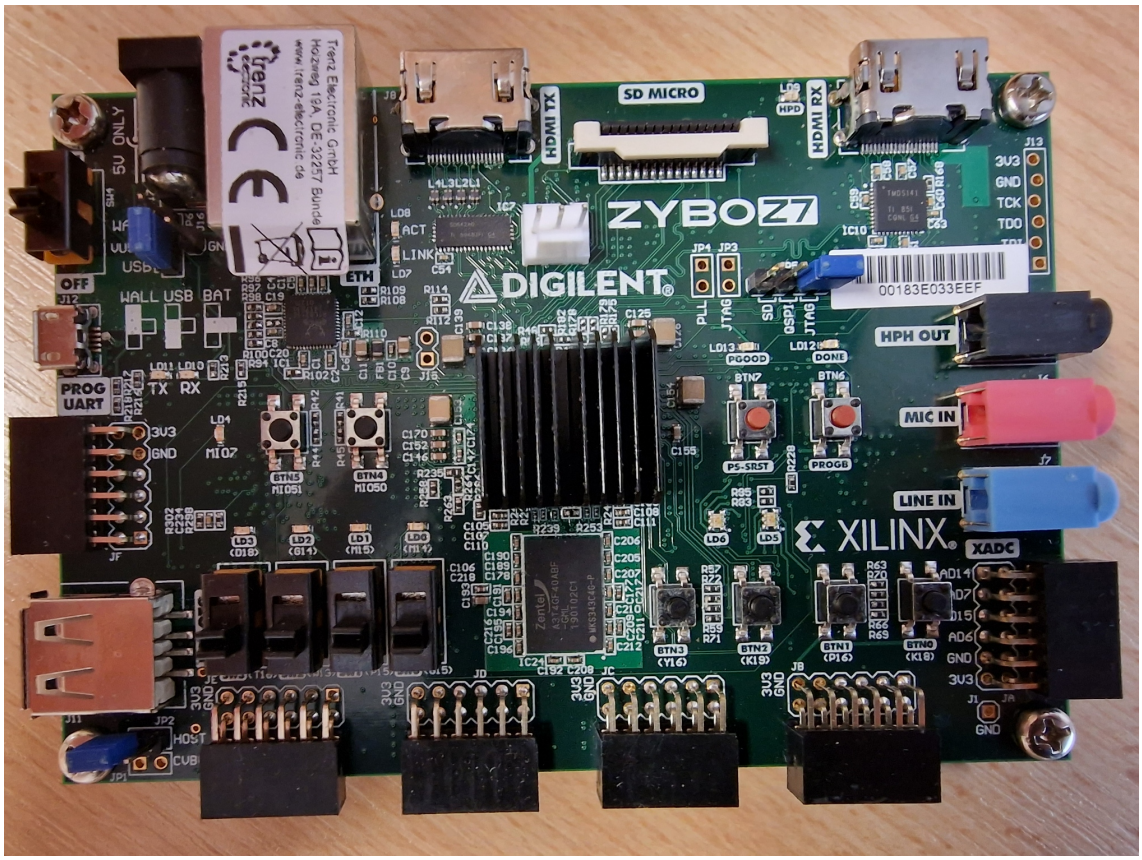
Programovatelná hradlová pole (FPGA – Field Programmable Gate Array) řadíme do skupiny programovatelných logických obvodů známých pod označením PLD (Programmable Logic Devices). Dnes patří mezi nejsložitější a nejpoužívanější ze všech programovatelných obvodů. FPGA obvody jsou programovatelné přímo u zákazníka nikoliv při výrobě, existuje tak možnost opětovné konfigurace. Zákazník si sám definuje funkci, kterou bude daný prvek plnit. Tímto se odlišují od mikroprocesorů, kde není možné měnit jejich funkcionalitu. Dnes navíc lze funkci u některých obvodů po částech měnit přímo za běhu. Mezi největší výrobce patří firmy Xilinx, Altera nebo Actel. Pro popis hardwaru jsou navrženy jazyky HDL (Hardware Description Language), mezi které řadíme VHDL (Very High Speed Integrated Circuit HDL) či Verilog [1].

Je patrné, že většina firem produkujících PLD obvody rozlišuje dvě architektury. Jedna z architektur vychází z konceptu programovatelné matice hradel AND, hradel OR a makrobuněk. Na spojeních je použita technologie výroby obvodu, tj. buňky EEPROM nebo flash. Další řešení je pak architektura FPGA obvodů. Jejím typickým znakem je použití generátorů logických funkcí s paměťmi (LUT tabulek), nejčastěji používaná technologie je SRAM, eventuálně Antifuse [1].

Jako měřítko velikosti obvodu lze použít počet ekvivalentních hradel, avšak nutno podotknout, že toto kritérium je hrubé a nemá za cíl říci, jestli zvolená architektura PLD je vhodná pro uskutečnění daného číslicového systému. Počet ekvivalentních hradel určuje množství dvouvstupových hradel NAND nebo NOR, které by bylo případně možné určitým PLD obvodem zaměnit. Běh programu v FPGA (oproti mikroprocesorům) může být efektivnější, jelikož program je tvořen propojením konkrétních hradel, je zde možnost využít ostatní hradla pro jiné procesy – paralelní operace [1].

Díky své programovatelnosti přímo v systému (ISP) hrají FPGA obvody velkou roli na mnoha různých trzích. Dnes jsou podstatnou složkou převážné většiny elektrotechnických zařízení – spotřební elektronika, systémy pro zpracovaní videa a obrazu, lékařská technika, průmyslová zařízení a další [2].

Na následujícím obrázku 1.1 je možné spatřit, jak může vypadat vývojová deska s FPGA čipem.

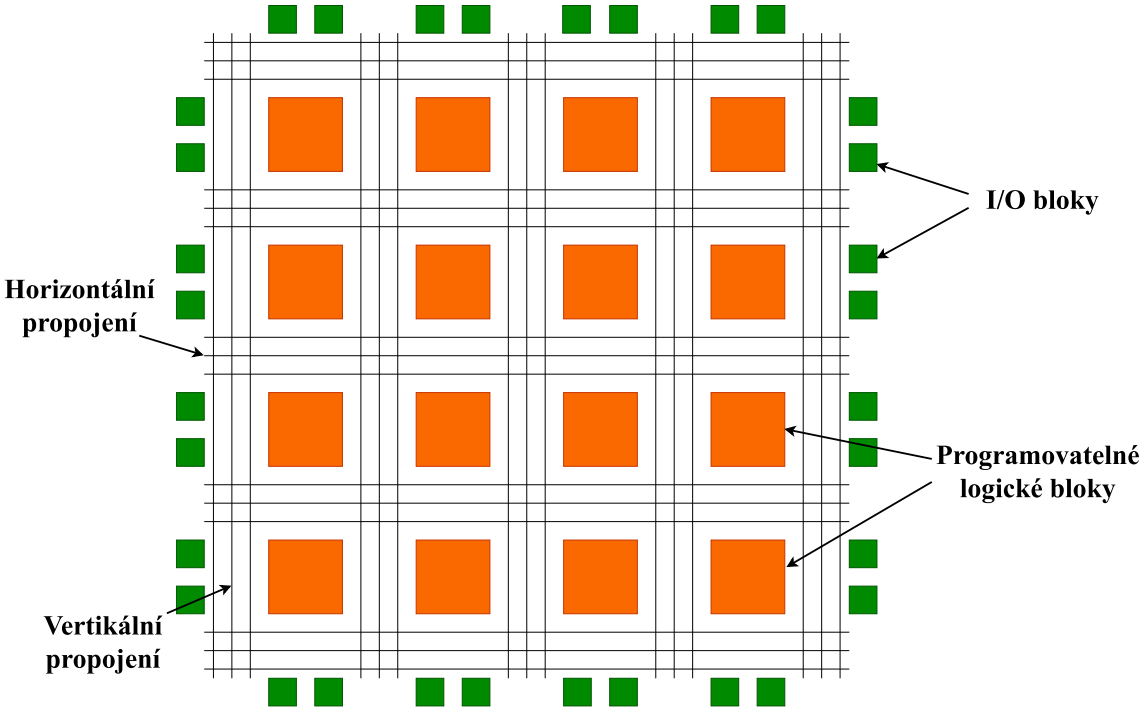


Obr. 1.1: Vývojová deska Digilent Z7-20 obsahující FPGA Xilinx Zynq řady 7.

1.1 Architektura FPGA obvodů

FPGA obvod je polovodičové zařízení, které se skládá z matice konfigurovatelných logických bloků spojených pomocí programovatelných propojení. Pro obstarání vstupu a výstupu jsou použity programovatelné I/O bloky, viz obrázek 1.2. Každý logický blok v sobě obsahuje generátor logické funkce (vyhledávací tabulku LUT), klopný obvod (flip-flop), lokální propojovací pole a multiplexor. Jako další specializované bloky lze označit např. násobičky, paměti, procesory a bloky pro úpravu hodinových signálů [1].

Dřívější organizace byla umístění bloků, propojení a I/O bloků v jedné vrstvě, v současnosti se jako lepší uspořádaní bere to, které má vrstvu logických bloků a I/O bloků a nad ní se nachází programovatelné propojení. To umožní značné zvýšení hustoty bloků [3].



Obr. 1.2: Zjednodušená architektura FPGA obvodu.

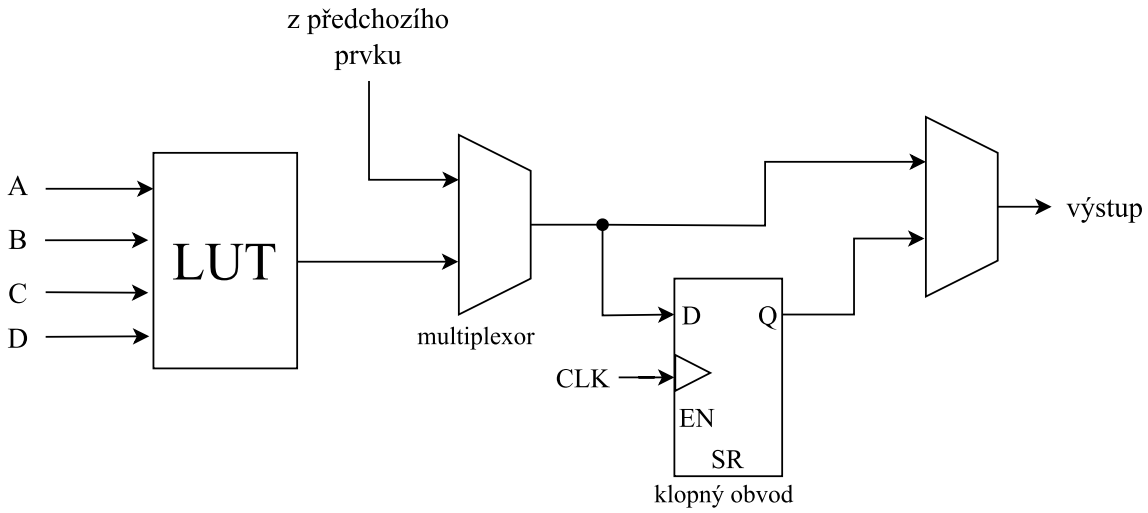
1.1.1 Programovatelný logický blok

Logický blok pod sebou sdružuje několik logických prvků (buněk) a lokální propojovací pole. Úlohou těchto bloků je realizace logických, paměťových a aritmetických funkcí v FPGA obvodu. Díky lokálním propojovacím polím lze tyto logické prvky propojit mezi sebou, případně i s dalšími v blízkých logických blocích [4].

Logický prvek se skládá z generátoru logické funkce pomocí paměti, také označován jako LUT tabulka, klopného obvodu a multiplexoru. Logický prvek je základním stavebním kamenem k uskutečnění kombinační a sekvenční logiky. Zjednodušené blokové schéma logického prvku můžeme vidět na obrázku 1.3 [4].

V novějších FPGA obvodech výrobce Xilinx se logický prvek neuvádí, ale hovoří se zde o tzv. řezu (Slice). Ten se skládá z určitého počtu LUT tabulek a klopných obvodů. Většinou dva řezy pak tvoří programovatelný logický blok, který je označován jako CLB (Configurable Logic Block) [4].

Počet logických prvků u nejmodernějších FPGA dnes dosahuje až deseti milionů (konkrétně až 10 200 000 u Intel Stratix 10 GX 10M FPGA) [5].



Obr. 1.3: Blokové schéma logického prvku.

1.1.2 LUT tabulka

Jedním z nejdůležitějších prvků v architektuře FPGA je LUT tabulka – je to jádro architektury FPGA. LUT tabulka většinou využívá čtyři vstupy a jeden výstup, a je tak schopna realizovat kombinační logické funkce čtyř proměnných. Avšak není nikterak neobvyklé, že novější FPGA obvody mají logické funkce až o šesti vstupech a dvou výstupech. Vyhledávací (LUT) tabulku lze propojit na další logické prvky prostřednictví multiplexoru [1].

Uvnitř vyhledávací tabulky nalezneme multiplexory a paměť typu RAM, které obsahují výstupy založené na vybraných linkách. Paměť implementovaná do LUT se nazývá distribuovaná RAM (více v kapitole 1.1.6). Stejně tak může být LUT nakonfigurována jako posuvný registr [1].

1.1.3 Klopný obvod

Výstup multiplexoru je veden do vstupu registru. Ten může být nakonfigurován tak, aby bylo řízení hranové (Flip-FLop) nebo hladinové (Latch). Klopné obvody v logických prvcích slouží k realizaci synchronní sekvenční části a jsou především typu D s hranovým řízením. Signály používané pro řízení jsou v jednom řezu shodné pro všechny klopné obvody. Patří mezi ně clock (CLK), enable (EN) a set/reset (SR) [4].

1.1.4 Propojovací síť

K propojení ostatních logických bloků a k propojení s I/O bloky slouží horizontální a vertikální propojení. Po ploše FPGA obvodu jsou propojení rozmístěna rovnoměrně a neobsahují žádné centrální propojovací pole [1].

1.1.5 I/O blok

I/O blok je vstupní/výstupní blok, který lze použít jako vstupní, výstupní nebo obousměrný. Vstupní a výstupní cesty obsahují hranou spouštěné D klopné obvody. Účelem I/O bloků je poskytnout uživatelské rozhraní z vnějšího světa k vnitřní architektuře FPGA, dále přizpůsobit signály různým napětím a řídit časování signálů [6].

1.1.6 Paměťové možnosti

Dnešní FPGA dovolují paměťové požadavky realizovat několika různými způsoby. Každý jednotlivý způsob má svoji aplikační oblast, kde představuje nejvhodnější řešení. Paměťové prvky v obvodech FPGA lze realizovat tímto výčtem:

- tabulkami LUT v logických buňkách, tzv. distribuovaná paměť RAM;
- klopnými obvody, které jsou částí logických buněk;
- speciální paměťovou strukturou, tzv. bloková paměť [7].

Prvním způsobem provedení může být distribuovaná paměť RAM. Paměť je vytvořena pomocí LUT. Jelikož tato tabulka je sama tvořena pamětí lze ji použít jako malou paměť typu RAM či ROM. Touto možností si zvýšíme paměť, ale připravíme se o využitelné prostředky pro realizaci jiné logiky. Například pro realizaci paměti o velikosti 1024 bitů použijeme 16 LUT tabulek se šesti vstupy ($2^6 \cdot 16$) [8].

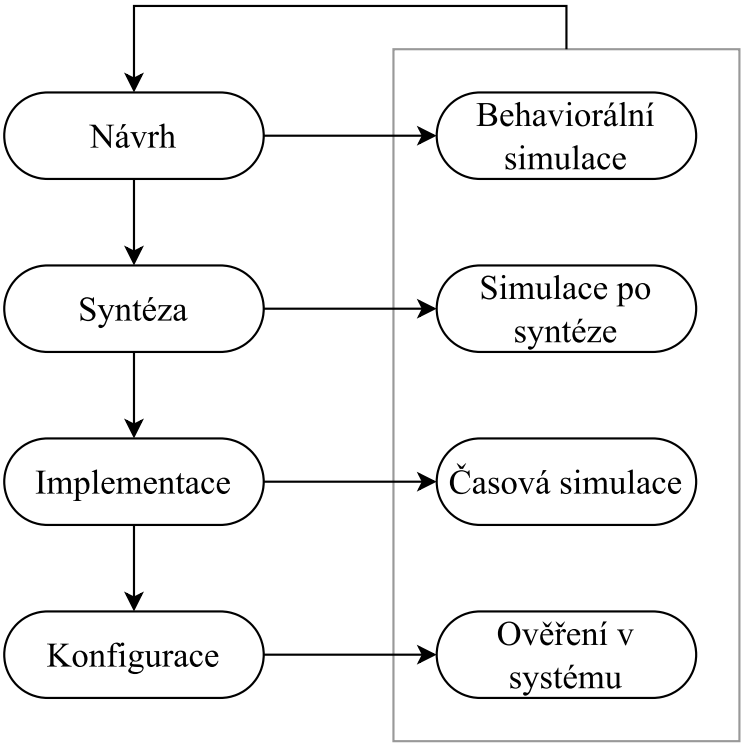
Dalším možným způsobem v pořadí je realizace paměti pomocí klopných obvodů. Nachází uplatnění především pro malé paměti, které jsou komponentem klasických synchronních subsystémů jako jsou kupříkladu čítače. Využijeme-li předchozí příklad, k paměti o velikosti 1024 bitů by bylo potřeba právě 1024 klopných obvodů [7].

Pro větší paměťové struktury se předchozí způsoby nehodí, rovněž použití standardní logiky je pomalé a neefektivní k realizaci paměti. Řešením je použití blokové paměti RAM označované jako vložená paměť (EBR). Jak už je z názvu patrné, bloková paměť je rozvrhnuta do bloků po FPGA čipu nebo do řad. Na rozdíl od výše zmíněných možností tato struktura zastává funkci pouze blokové paměti. Pokud není využita pro tuto funkci nelze ji použít pro jiné účely. V jednom obvodu jsou většinou bloky stejných velikostí, typicky 10 až 36 kb. Takových bloků může být na čipu i tisíce. Po sečtení je k dispozici kapacita i přes 100 Mb. Paměť se dá konfigurovat jako klasická jednoportová nebo jako dvouportová [4, 8].

1.2 Konfigurace obvodů FPGA

Návrh FPGA obvodů se neobejde bez specializovaných vývojových nástrojů, jelikož dnešní FPGA jsou velmi složitá zařízení s miliony ekvivalentních hradel, a celý proces implementace tak může být značně zdlouhavý. Pomocí těchto komplexních nástrojů lze provádět návrh, stejně tak slouží i pro verifikaci, implementaci, konfiguraci a testování [8].

V rámci vývoje číslicového systému je také přínosné vytváření testovacího prostředí (testbench), kterým vytvářený systém průběžně testujeme. Stejně tak existuje možnost ověření funkčnosti celého zdrojového kódu v závěru návrhu. Postup hardwarové implementace typicky zahrnuje kroky uvedené na obrázku 1.4 [8].



Obr. 1.4: Postup návrhu FPGA.

1.2.1 Simulace

Při vývoji číslicového obvodu je nutné ověřit, zda jsme se nedopustili chyb a či vývojové prostředí rozumí našemu programu tak, jak jsme si představovali. Je také dobré vědět, jaké bude mít daný obvod parametry (zpoždění, kmitočty apod.). Tohle vše nám zodpoví právě simulace [7].

Simulace probíhá tak, že je vytvořen model systému a na jeho vstupy se přivedou testovací signály a z výstupních signálů určíme, jestli se signály blíží očekávaným výstupům. Na základě druhu modelu, který je k simulaci využit, rozlišujeme:

- simulaci behaviorálního modelu – behaviorální simulace;
- simulaci modelů vytvořených na různých úrovních zpracování.

Behaviorální model je takový, který se skládá ze zdrojových souborů vytvořených návrháři číslicového systému [7].

Behaviorální (funkční nebo RTL) simulace není závislá na architektuře obvodu a nepřihlíží k časovým skutečnostem. Typicky se provádí pro verifikaci syntaxe kódu a ověření funkčnosti modelu. Simulaci je vhodné provádět v průběhu vývoje, dokud funkčně nevyhoví požadované specifikaci nebo před dalším zpracováním zdrojových souborů systémem. Simulace není zpravidla náročná na výkonnost stroje, kde probíhá, a je tak rychlá [7].

Funkční simulace po syntéze na úrovni hradel je především u FPGA obvodů značně nepřesná, jelikož zpoždění v propojovacích sítích je významně větší než ve vlastní logice. Tato simulace je zejména vhodná pro odhalení nevhodného popisu nebo nedokonalostí syntetizérů, kdy dochází k odchylkám, které by v praxi mohly vést k chybnému fungování zařízení. Pří špatných výsledcích testů je nutné poupravit zdrojový kód, eventuálně syntézní nástroj [4].

Časová simulace na úrovni hradel bere v úvahu model vytvářený při implementaci. Simulace již využívá informace o skutečném zpoždění na jednotlivých prvcích, vychází také z technologických knihoven výrobce číslicového systému a respektuje reálné funkční chování jednotlivých bloků. Oproti funkční simulaci je mnohem náročnější na výkonnost stroje a je tak nejvíce časově náročná. Pokud simulace dává správné výsledky, je velmi pravděpodobné, že stejných výsledků bude dosahovat i v reálném FPGA obvodu [4].

Ze statické časové analýzy, která požívá model generovaný systémem při syntéze, je rovněž možné získat časové údaje [7].

1.2.2 Návrh

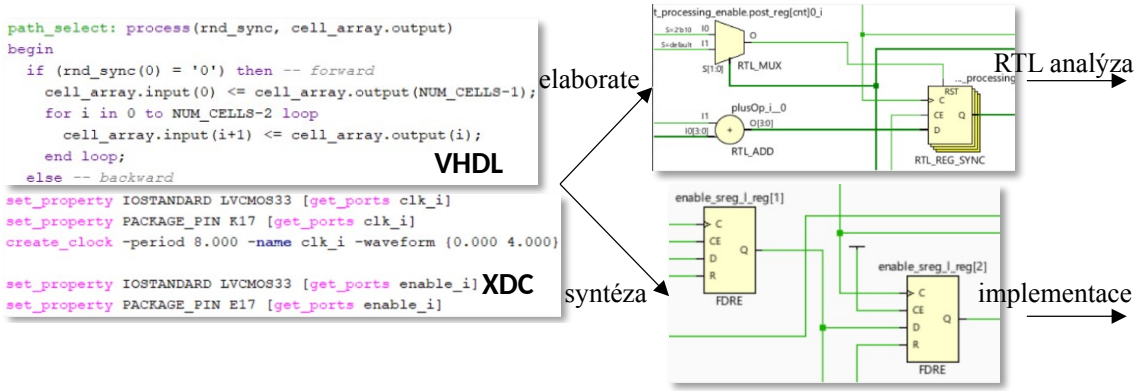
Prvním krokem je vytvoření popisu (modelu) vyvíjené aplikace na základě specifikace, označován jako návrh. Ve specifikaci by neměla být opomenuta kompletní funkce navrhovaného systému a také například do jakých pracovních podmínek bude výsledný obvod nasazen. Popis obvodu je prezentován v textové podobě pomocí jazyku pro popis hardware (VHDL nebo Verilog) [1].

Popis se následně ověří pomocí behaviorální (funkční) simulace, která ověřuje správné fungování vytvořeného modelu, probíhá také kontrola syntaxe popisu.

1.2.3 Syntéza

Při procesu syntézy program označovaný jako syntetizér konvertuje všechny zdrojové kódy do vzájemného zapojení elementárních logických prvků, kterým obvod disponuje. Výsledný soubor syntézy se nazývá RTL (Register Transfer Level) netlist, který popisuje propojení jednotlivých prvků cílového zařízení. Netlist se zapisuje nejčastěji ve formátu EDIF [8].

Na syntézu mají vliv kromě HDL kódu také nastavená omezení syntézy (XDC – Xilinx Design Constraints) a cílová technologie. Omezení formuluje návrhář obvodu. Jsou zde definovány klíčové parametry, např. časové parametry, očekávaná maximální hodinová frekvence atd. Zároveň probíhá optimalizace navrženého FPGA obvodu. Po zdařilé syntéze lze přikročit k funkční simulaci po syntéze [4].

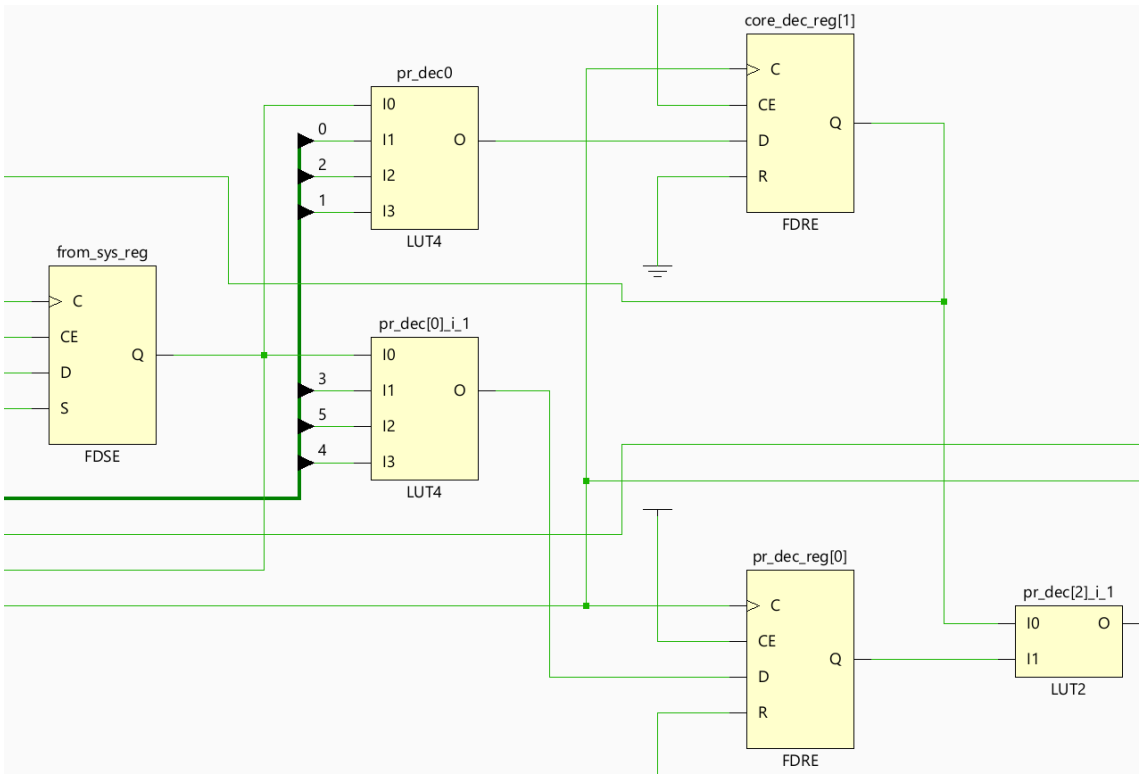


Obr. 1.5: Princip syntézy u FPGA obvodů.

1.2.4 Implementace

Implementace se skládá z několika postupných kroků. Ty následně vytvoří popis propojení obvodu, které je podkladem pro závěrečnou konfiguraci FPGA. Prvním krokem je proces zvaný mapování, jež přiřadí elementárním prvkům, použitých ve finále syntézy, konkrétní prvky využité v cílovém FPGA obvodu. Obrázek 1.6 ukazuje, že na rozdíl od RTL schématu, které obsahuje pouze jednotlivé prvky v podobě elementárních hradel, jsou po procesu mapování prvky v podobě LUT [8].

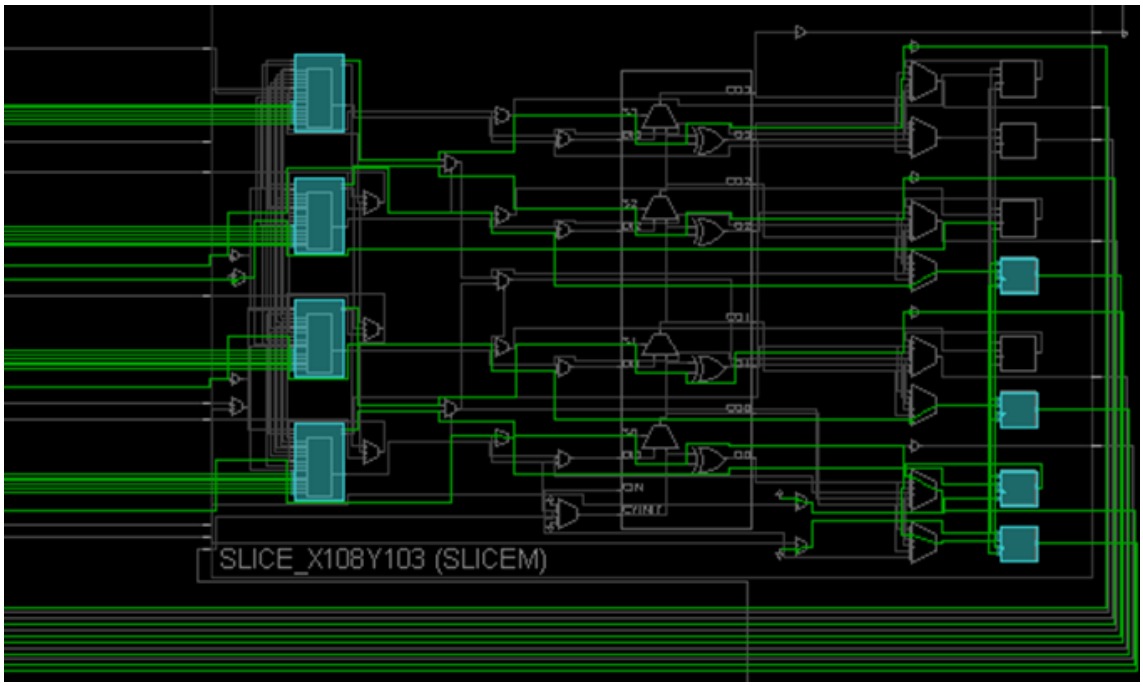
Nyní následuje rozmístění (Place), kdy dochází k rozmístění logických prvků do matice FPGA obvodu. Úkolem je rozmístit komponenty tak, aby jejich následné propojení bylo efektivní [8].



Obr. 1.6: Výsledek procesu mapování – výřez schématu.

Dalším krokem je již zmíněné propojení (Route). Zde je cílem najít vhodné propojovací struktury a ty pak uspořádat do jednotlivých spojů, aby zpoždění bylo co nejmenší. Propojením se zformuluje plán výsledné struktury s nakonfigurováním programovatelných propojek. Na obrázku 1.7 jsou vidět použité prvky a jejich propojení v konkrétním řezu daného FPGA obvodu [4].

Závěrem je pak provedení simulace k ověření časových a jiných parametrů výsledné implementace.



Obr. 1.7: Výsledek procesu Place and Route – ukázka jednoho řezu.

1.2.5 Konfigurace, používané technologie

Tato operace zahrnuje vygenerování konfiguračního souboru (bitstream), který se zavádí do různých prvků podle použité technologie. Existují tři hlavní technologie, které se běžně používají k implementaci konfiguračních prvků uvnitř FPGA: antifuse, flash a založené na SRAM.

U FPGA s non-volatilní konfigurací (energeticky nezávislé) se k ukládání bitstream využívají antifuses (antipojistky), nebo paměti flash/EEPROM. Jejich výhodou je hlavně okamžitá použitelnost po zapnutí zdroje. Další předností je nižší spotřeba energie výsledného obvodu a konfigurační soubor je přímo uložen v FPGA obvodu, a lze tak zabránit jeho vyčtení. Na rozdíl od zařízení na bázi antifuse jsou konfigurační buňky flash vícenásobně programovatelné, a proto se dají v případě potřeby přeprogramovat na novou konfiguraci. Antipojistky mají výhodu a vedou v odolnosti vůči okolnímu záření [9].

U FPGA s volatilní konfigurací (závislé na zdroji elektrické energie) se přenáší konfigurační informace do statických paměťových buněk SRAM, kde každý konfigurační bit má přidruženou buňku SRAM. Výhodou je snadná konfigurace a rekonfigurace i za běhu systému a také rychlost. Nevýhodou je, že jsou nestálé, což znamená, že jejich konfigurace se ztratí, pokud je napájení odpojeno. Z tohoto vyplývá důsledek, že FPGA založené na SRAM musí mít svou konfiguraci znovu načtenou při každém zapnutí systému. Jedním z režimů konfigurace je ten, kdy FPGA čte kon-

figurační data z externího zdroje (např. flash). Další způsob je konfigurace obvodu externím hlavním zařízením, jako je procesor [9].

FPGA na bázi SRAM s interní pamětí flash je kombinace, kterou výrobci na trhu nabízí. Tento typ FPGA je obecně stejný jako předchozí až na to, že tyto čipy obsahují interní bloky flash paměti, čímž se eliminuje potřeba mít externí energeticky nezávislou paměť [9].

Jak antifuse tak obvody na bázi flash mají výhodu v nízké spotřebě energie. Obě technologie však vyžadují následné kroky zpracování nad rámec základního procesu CMOS používaného k vytvoření křemíkových čipů. V tomto nalézáme nevýhodu, jsou obvykle o jednu až dvě generace pozadu za špičkovou výrobní technologií [9].

Testování přímo na FPGA může odhalit také chyby v návrhu a ty je pak nutné opravit ve zdrojových kódech. Korekce kódu ovšem znamená uskutečnit opět celý proces implementace včetně patřičných kroků verifikace [8].

1.2.6 Jazyky pro popis hardwaru

Pro popis číslicových systémů, tedy i FPGA, se používá některý jazyk pro popis hardwaru, tzv. HDL (Hardware Description Language), dnes nejčastěji VHDL, který převládá spíše v Evropě, nebo Verilog dominující v USA a Asii. Oba jazyky jsou hojně používány a také jsou akceptovány jako otevřené standardy IEEE [4].

Pro tyto jazyky je charakteristická velká míra abstrakce a ta hlavně roste se složitostí navrhovaného systému. Tato abstrakce v sobě může obsahovat i velké riziko pro syntézu, kde syntetizér mylně text zpracuje, nebo výsledná struktura bude pro nás neefektivní [7].

Mezi základní znaky VHDL jazyka lze označit přenositelnost kódu, zahájení návrhu i bez znalosti konkrétního obvodu, umožnění přímé simulace a implementace do cílového obvodu. Jazyk VHDL umožnuje popis řešení, které je syntetizovatelné, převeditelné do formy konfiguračního souboru. Po přenosu souboru do číslicového systému je obstarána fyzická funkce. Velkou výhodou, např. oproti jazyku C, je možnost souběžného chodu programu [10].

Doplňkem „nízkoúrovňových“ jazyků VHDL a Verilog mohou být vyšší programovací jazyky jako je System C a C/C++, které jsou stále více používány. Je dobré myslet na to, na jakou platformu cílíme, a zdrojový kód jí přizpůsobit i při použití pokročilých programovacích jazyků. Pro návrh FPGA obvodů je nezbytné pochopit a zvládat principy programování. S tím se pojí dodržování syntaktických pravidel daného jazyka při věcném správném popisu vytvářené konstrukce [8].

2 Generátory náhodných čísel

Využití pro náhodná čísla najdeme zejména v oblasti kryptografie, jelikož bezpečnost mnoha kryptografických systémů závisí na generování nepředvídatelných posloupností. Jedná se o zabezpečení elektronických dat a veškeré spojené komunikace. Užitečná jsou také jako estetický prvek, například v literatuře a hudbě, a samozřejmě jsou stále užívané pro hry a hazardní hry. Generování náhodných čísel představuje z kryptografického hlediska jeden z podstatných problémů. V následujících kapitolách bude hovořeno o RNG (Random Number Generator) z pohledu kryptografie [11].

V praxi namísto náhodných čísel pracujeme často s náhodnými bity, používají se generátory binárních posloupností. Pokud chceme náhodné číslo z konkrétního intervalu $\langle 0; 2^n - 1 \rangle$, pak nám stačí vygenerovat posloupnost o n bitech, kterou jednoduše bereme jako n-bitové číslo [12].

2.1 Nároky na generování

Pokud hovoříme o jednotlivých číslech, náhodné číslo je takové, které je vybráno z množiny hodnot, přičemž každá hodnota má stejnou pravděpodobnost výběru. Při zkoumání náhodných posloupností je primárním požadavkem, aby tyto posloupnosti měly rovnoměrné rozdělení [11].

Důležitost v posloupnosti náhodných čísel hraje fakt, že musí být každé vylosované číslo statisticky nezávislé na ostatních, není mezi nimi žádná korelace. Náhodné číslo je takové, které je výstupem generátoru náhodných čísel, který má nepředvídatelný výsledek a jeho průběh nelze přesně reprodukovat [13].

Dalším nárokem na generátory náhodných čísel je rychlost generování, což je veličina, která nám říká, kolik náhodných bitů je daný generátor schopný vyprodukovat za jednotku času [13].

2.2 Entropie

Velmi důležitou veličinou pro hodnocení generátoru je entropie. Entropie X je matematickým měřítkem množství informací poskytovaných pozorováním X. Ekvivalentně je to nejistota ohledně výsledku před pozorováním X. Entropie je také užitečná pro aproximaci průměrného počtu bitů potřebných k zakódování prvků X [13].

Entropie X je definována vztahem

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i \,,$$

kde X je generovaná hodnota, p_1 až p_n jsou pravděpodobnosti hodnot X_1 až X_n, které je generátor schopen vygenerovat [13].

Entropie je měření nejistoty nebo neuspořádanosti v systému. Dobrá entropie pochází z okolního prostředí, které je nepředvídatelné a chaotické. Můžeme si ji představit jako množství překvapení nalezeného ve výsledku randomizovaného procesu. Platí, čím vyšší je entropie, tím menší je jistota nalezená ve výsledku. Maximální entropie nastává, pokud se pro daný počet bitů generují všechny možné posloupnosti a každá se stejnou pravděpodobností. Pokud útočník následující generovanou hodnotu zná, entropie je nulová [11].

2.3 Typy generátorů

Existují dva hlavní přístupy ke generování čísel (binárních posloupností), a to generátory pseudonáhodných čísel (PRNG – Pseudo Random Number Generator) a generátory skutečných náhodných čísel (TRNG – True Random Number Generator). Přístupy mají zcela odlišné vlastnosti a každý má své pro a proti. Ještě je možné rozlišovat tzv. smíšené generátory, které kombinují uvedené typy generátorů [12].

2.3.1 Generátor pseudonáhodných čísel (PRNG)

PRNG jsou v podstatě algoritmy, které používají matematické vzorce nebo předem vypočítané tabulky ke generovaní posloupností čísel, která působí jako náhodná. Čísla jsou tedy generována softwarově, zdají se být náhodná (pokud útočník nezná parametry generátoru), ale ve skutečnosti jsou předem určená [11].

Pseudonáhodné generátory jsou velmi výkonné, tedy schopny rychle generovat velké množství čísel. Tyto generátory využívají deterministické postupy – pokud známe výchozí bod (tzv. semínko) v sekvenci, můžeme reprodukovat stejnou posloupnost čísel později. PRNG jsou obvykle také periodické, jinak řečeno, sekvence se může nakonec po nějaké době opakovat. Avšak dnešní moderní PRNG mají periodu tak dlouhou, že ji lze pro některé potřeby ignorovat. Realizace takových generátorů je poměrně snadná [11].

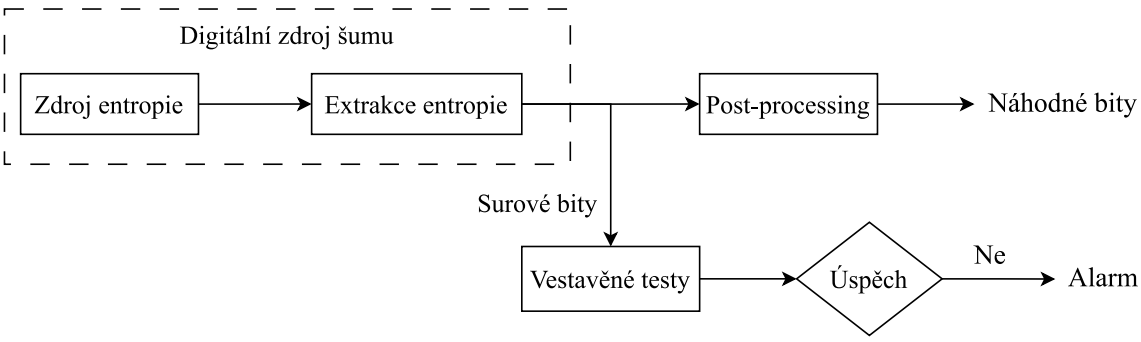
Díky těmto vlastnostem jsou pseudonáhodné generátory vhodné pro aplikace, kde je potřeba rychle získat velké množství náhodných čísel. Mezi příklady těchto aplikací patří simulační a modelovací aplikace, kde je potřeba generovat náhodná data pro opakovatelné experimenty a analýzy. Nicméně, pseudonáhodné generátory nejsou příliš vhodné pro aplikace, kde je klíčové, aby čísla byla skutečně nepředvídatelná. To platí například pro šifrování dat a hazardní hry [11].

Příkladem pseudonáhodného generátoru je například posuvný registr s lineární zpětnou vazbou (LFSR – Linear Feedback Shift Register), ale z hlediska kryptoanalýzy se řadí mezi málo odolné. Nelineární generátor, tedy kryptoanalyticky odolnější, nelze ho popsat soustavou lineárních rovnic jako v předchozím příkladě, je generátor šifry A5/1 [12].

Kryptograficky bezpečné PRNG vyžadují náhodný a také tajný vstup, který je označován jako seed. Právě od tohoto vstupu se odvíjí i kvalita generátoru. Entropie výstupu generátoru se rovná entropii, která vstupuje, tj. seed. Avšak zdroj entropie je třeba průběžně testovat, mohou se zhoršovat jeho vlastnosti [13].

2.3.2 Generátor pravých náhodných čísel (TRNG)

Generátory pravých náhodných čísel využívají ke generování náhodné fyzikální procesy, proto jsou označovány jako fyzikální. Tyto generátory lze popsat jako hardwarová zařízení, která přijímají nedeterministické vstupy ve formě fyzikálních měření teploty, fázového šumu, hodinových signálů atd. a jako výstup generují nepředvídatelná čísla. TRNG lze obecně rozdělit do čtyř částí, viz obrázek 2.1: zdroj entropie, extrakce entropie, následné zpracování (post-processing) a vestavěné testy [11].



Obr. 2.1: Obecná struktura TRNG.

Zdroj entropie je fyzikální proces, příliš komplikovaný nebo nestálý, a je skoro nemožné ho popsat matematickým modelem. Skutečně dobrým fyzikálním jevem je radioaktivní zdroj. Časové body, ve kterých se radioaktivní zdroj rozpadá, jsou nepředvídatelné, je možná jejich detekce a další využití [11].

Dalším vhodným fyzikálním jevem je atmosférický šum. Můžeme také použít teplotní šum rezistoru, kolísání kmitočtu oscilátoru, hluk na pozadí z kanceláře. U všech příkladů je nutné ověřit, jestli se neobjevují vzory [11].

Zdroj entropie se extrahuje pomocí *mechanismu sklizně*, který narušuje fyzikální proces a snaží se o „sběr“ co nejvíce entropie. Nakonec se provede vzorkovací operace pro akumulaci skutečné náhodnosti za účelem vytvoření náhodných bitů – funkce

spočívá v pravidelném vzorkování analogových signálů generovaných zdrojem entropie a jejich převodu na digitální. Spolu se zdrojem entropie vytváří digitální zdroj šumu [14].

Paradoxně náhodnost digitálního šumu sama o sobě přináší problém. Obecně platí, že TRNG produkuje nepředvídatelná náhodná čísla se špatnými statistickými vlastnostmi. Právě *post-processing* může být použit k maskování nedokonalostí (lepším statistickým výsledkům) nebo k zajištění stejných výsledků v případě změny prostředí. Může se například jednat o von Neumannův korektor, extraktor či jednosměrnou hašovací funkci [14].

Podle požadavků na pokyny pro kryptografické moduly musí být kvalita generovaného náhodného bitového toku ověřena některými ze standardních sad statistických testů, jak to vyžadují bezpečnostní standardy, např. FIPS 140-2. Dvě z nejrozšířenějších standardních testovacích sad pro hodnocení náhodnosti jsou SP 800-22 od NIST a Dieharder, kterým je věnována následující kapitola 3. *Vestavěné testy* nám sledují stav TRNG. Je důležité, aby ke kontrolám docházelo i průběžně, důvodem může být stárnutí součástek, vliv prostředí nebo úsilí útočníka [15].

Charakteristiky skutečně náhodných generátorů jsou zcela odlišné od pseudonáhodných. TRNG jsou obecně spíše neefektivní ve srovnání s PRNG, takže výroba čísel trvá podstatně déle. Další nevýhodou je problém technické realizace, která souvisí s dodatečnou hardwarovou částí. Výhodnou vlastnost nalézáme v tom, že jsou nedeterministické (sekvenci není možné reprodukovat). V pravých generátorech nenajdeme periodicitu [11].

Díky těmto vlastnostem najdeme uplatnění skutečných generátorů v oblasti kryptografie jako je šifrování dat, generování klíčů, dále v hrách, zejména hazardních hrách, a loteriích.

2.4 Využití náhodných čísel v kryptografii

Generování náhodných čísel je důležitým primitivem v mnoha kryptografických mechanismech. Například klíče pro šifrování je třeba generovat způsobem, který je pro protivníka zcela nepředvídatelný. Generování náhodného klíče obvykle zahrnuje výběr náhodných čísel nebo bitových sekvencí [13].

Bezpečnost mnoha kryptografických systémů závisí na generování nepředvídatelných posloupností. Příklady zahrnují jednorázové klíče (one-time pad) při Vernamově šifře, tajný klíč v šifrovacím algoritmu DES, prvočísla p, q v šifrování RSA a digitálním podpisu, soukromý klíč v DSA, výzvy používané v autentizačních systémech typu výzva-odpověď, generování čísel nonce, salt, výplní padding a také obrana vůči útokům postranními kanály. Ve všech těchto případech musí být generované posloupnosti dostatečné velikosti a musí být náhodné v tom smyslu, že

pravděpodobnost výběru jakékoli konkrétní hodnoty musí být dostatečně malá, aby zabránila protivníkovi získat výhodu prostřednictvím optimalizace vyhledávací strategie založené na takové pravděpodobnosti. Například klíče v algoritmu AES mohou mít velikost 256 bitů. Pokud by byl tajný klíč vybrán pomocí skutečně náhodného generátoru, musel by útočník v nejhorším případě zkusit 2^{256} možností, než uhodne správný klíč [13].

3 Testy náhodnosti

Síla zabezpečení mnoha systémů a aplikací závisí na vysoce kvalitním TRNG. Následující kapitola se zabývá testy používaných k vyhodnocení skutečného výstupu generátoru náhodných čísel. Testy bývají součástí obecně známých a používaných balíků (baterií). Testy náhodnosti zkoumají náhodně vygenerované bitové sekvence a snaží se nalézt periodu (vzor).

Můžeme najít mnoho testovacích programů. Za dobře známý a hojně využívaný v minulosti lze označit Diehard, ale dnes je již zastaralý. Proto organizace NIST (National Institute of Standards and Technology) přikročila k vydání souboru testů, které se snaží neustále vyvíjet a aktualizovat. Jako další příklad uveďme FIPS 140-1, FIPS 140-2 nebo Dieharder (re-implementace Diehard testů).

Statistické testy slouží k ověřování statistických hypotéz na základě stanovené úrovně významnosti. Za nulovou hypotézu označíme předpoklad, že zkoumaná posloupnost je náhodná. Testy rozhodují na základě vypočtené p-hodnoty. Pokud je tato p-hodnota překročí stanovenou mez, zamítáme nulovou hypotézu. V případě testů baterií poskytovaných organizací NIST je nutné, aby výsledná hodnota byla rovna nebo vyšší než kritická hodnota. Tento práh je možné upravit, ale standardně je nastavena na 0,01 [16].

3.1 Diehard

Baterie testů Diehard byly zveřejněny v roce 1995, jejich autotem je George Marsaglia. Původně obsahoval dvanáct různých testů. Tento soubor testů již momentálně není podporovaný ani aktualizovaný a nahradil ho NISTS STS (Statistical Test Suite), který Diehard testy upravuje a dále rozšiřuje. Navíc pokud chcete spustit všechny Diehard testy najednou, váš datový soubor musí mít velikost asi 10 megabajtů a obsahovat tak přibližně 10 milionu čísel [17].

3.2 NIST STS

Americký Národní institut standardů a technologií publikoval sadu statistických testů, které jsou popsané v SP 800-22, pro testování náhodných čísel vytvořených náhodnými a pseudonáhodnými generátory [16]. S tímto se pojí i publikace řady NIST SP 800-90, které specifikují generování vysoce kvalitních náhodných bitů pro kryptografické i nekryptografické použití, konkrétně poté dokument SP 800-90C pro konstrukci generátoru náhodných bitů [18].

NIST STS zahrnuje patnáct testů, které hodnotí náhodnost libovolně dlouhých binárních souborů. Níže jsou vyjmenovány a krátce je popsán jejich princip fungování. Některé testy v sobě obsahují více menších testů [16].

1. **Frekvenční test** (Frequency (Monobit) Test)
   Tento test ověřuje, zda jsou jednotlivé bity v testované sekvenci rovnoměrně rozloženy mezi 0 a 1. Porovnává se počet jedniček a nul s očekávaným počtem za předpokladu rovnoměrného rozložení [16].
2. **Frekvenční test v rámci bloku** (Frequency Test within a Block)
   Jak je z názvu patrné, test se bude podobat frekvenčnímu testu. Rozdílem je, že vygenerovaná posloupnost je rozdělena do bloků o délce M. Zjišťuje se, jestli blok obsahuje očekávaný počet jedniček a nul [16].
3. **Test série bitů** (Runs Test)
   Tento test identifikuje počet sérií v dané sekvenci, přičemž série představuje nepřerušenou posloupnost identických bitů. Série délky k se skládá z přesně k po sobě jdoucích identických bitů a je ohraničena bity opačné hodnoty. Cílem testu je posoudit, zda nedochází příliš ke zpomalení nebo zrychlení výskytu změn mezi nulami a jedničkami [16].
4. **Test nejdelší sekvence jedniček uvnitř bloku** (Tests for the Longest-Run-of-Ones in a Block)
   Testování probíhá tím, že se vypočítá počet nejdelších sekvencí jedniček v každém bloku o délce M bitů. Výsledek se následně porovná s předpokládanými hodnotami. [16].
5. **Test sérií binárních matic** (Binary Matrix Rank Test)
   Úkolem testu je ověřit, zda jsou podřetězce v testované sekvenci nezávislé a náhodně distribuované. Test se provádí konstrukcí binární matice z testované sekvence [16].
6. **Test diskrétní Fourierovy transformace** (Discrete Fourier Transform Test)
   Test umožňuje detekovat periodické rysy v testované sekvenci, které by naznačovaly odchylku od předpokladu náhodnosti. Při tomto testu je využita Fourierova transformace, která převádí sekvenci symbolů na frekvenční spektrum. Zjišťujeme, zda počet vrcholů se stejnou výškou nepřekračuje práh [16].
7. **Test nepřekrývajících se vzorů** (Non-overlapping Template Matching Test)
   Test hledá a počítá vzory v sekvenci, odhaluje tak generátory, které produkují příliš mnoho výskytů neperiodických vzorů. Pro test se používá M-bitové okno k vyhledání specifického M-bitového vzoru. Okno se postupně posunuje podle toho, jestli byl vzor nalezen [16].

8. **Test překrývajících se vzorů** (Overlapping Template Matching Test)
   Test se zaměřuje na počet výskytů předem zadaných cílových řetězců. Tento test také používá M-bitové okno k hledání specifického vzoru. Rozdíl nalézáme v tom, že vzory se mohou vzájemně překrývat [16].
9. **Mauerův univerzální statistický test** (Maurer's Universal Statistical Test)
   Tento test cílí na analýzu počtu bitů mezi odpovídajícími si vzory. Jeho účelem je posoudit, zda je možné tuto sekvenci výrazně zredukovat bez ztráty informace. Pokud je komprimace minimální, naznačuje to, že sekvence lze považovat za náhodnou [16].
10. **Test lineární složitosti** (Linear Complexity Test)
    V tomto testu hraje roli délka lineárního zpětnovazebního posuvného registru (LFSR). Test si klade za cíl určit, zda je posloupnost natolik komplexní, aby byla považována za náhodnou [16].
11. **Test sérií** (Serial Test)
    Těžištěm tohoto testu je zjistit počet všech M-bitových vzorů v testované sekvenci. Náhodné sekvence mají jednotnost, to znamená, že každý M-bitový vzor má stejnou šanci, že se objeví jako každý jiný M-bitový vzor [16].
12. **Přibližný test entropie** (Approximate Entropy Test)
    Tento test ověřuje, zda je testovaná sekvence chaotická, nebo periodická. Test se provádí vypočítáním aproximace entropie a porovnáním této aproximace s očekávanou hodnotou za předpokladu náhodné sekvence. Hledány jsou opět M-bitové vzory [16].
13. **Test kumulativní sumy** (Cumulative Sums (Cusums) Test)
    Středobodem testu je kumulativní součet upravených $(-1, +1)$ číslic v sekvenci a jeho odchylka od nuly. Bit 1 značí $+1$ a bit 0 představuje -1. Pro určité typy nenáhodných sekvencí budou součty od nuly velké [16].
14. **Test náhodných návštěv** (Random Excursions Test)
    Test hledá, zda je v testované sekvenci dostatečné množství náhodných „návštěv“ v různých směrech, tedy tam i zpět. Stejně jako v předchozím testu i tento pracuje s kumulativními součty, přesněji s částečnými. Je vypočten počet návštěv dané délky v jednom směru a porovnává se s očekávanou hodnotou za předpokladu náhodné sekvence. Test nám odhalí, zda není v sekvenci příliš mnoho opakujících se vzorců v určitém směru. Tento test obsahuje celkem osm testů [16].
15. **Test náhodných variant návštěv** (Random Excursions Variant Test)
    Test je obdobný testu náhodných návštěv. Rozdíl je patrný v tom, že se bere v úvahu variabilita délky návštěv. Tento test je série osmnácti testů [16].

4 Analýza dosavadních hardwarových implementací TRNG

Jak je uvedeno v části 2.4, bezpečnost kryptografických systémů je spojena především s generováním náhodných bitových sekvencí – klíčů. Aby se zabránilo útokům v nepřátelském prostředí, jsou klíče generovány v nepředvídatelných generátorech (TRNG) uvnitř zabezpečeného čipu. Tato kapitola představuje různé možnosti implementace TRNG.

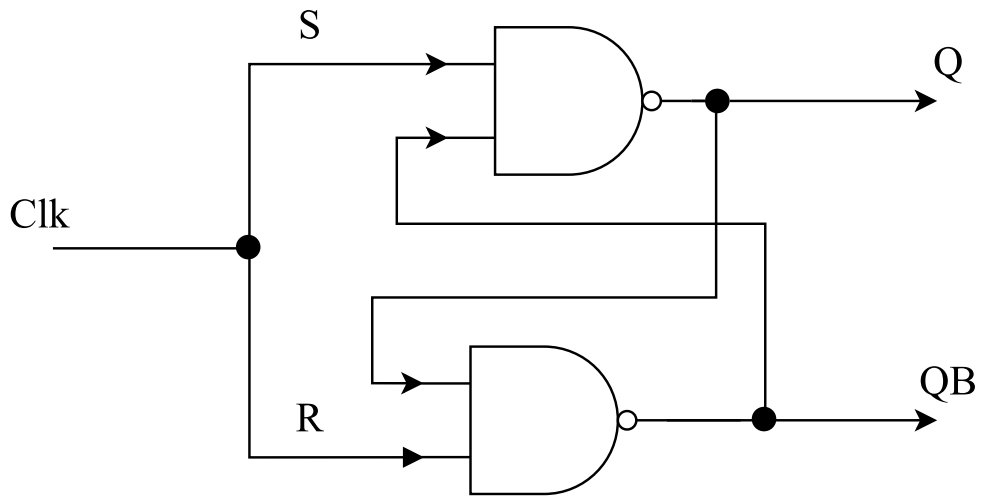
Použití FPGA při konstrukci hardwarových bezpečnostních systémů je výhodné, protože umožňují snižovat bezpečnostní rizika snadnou rekonfigurací zastaralých nebo nedostatečně bezpečných kryptografických protokolů. Nejčastěji RNG využívají jeden zdroj entropie, případně operace následného zpracování. Mezi běžně používané zdroje entropie patří tepelný šum, metastabilita, jitter hodin v obvodech a chaos. V současnosti je většina generátorů implementovaných v FPGA založena na časovém šumu hodinových signálů generovaných v PLLs (Phase-Locked Loop – smyčka fázového závěsu) a především v prstencových oscilátorech (RO – ring oscillator) [19].

4.1 Implementace založena na metastabilitě

Metastabilita využívá skutečnosti, že logické obvody, jako je klopný obvod typu D, vyžadují určitý čas, než se usadí na jedné z logických úrovní. Pokud je toto časové okno narušeno, obvod může vstoupit do metastabilního stavu – výstup osciluje mezi „0“ a „1“ [20]. Metastabilita je vhodná jako zdroj náhodnosti, jelikož výstupní hodnota není deterministická. Tato metoda však není široce používána, i přesto lze některé návrhy najít.

Jako příklad je možné uvést práci [20] z roku 2019, kde architektura TRNG byla ovlivněna metastabilitou na Altera FPGA, a to prostřednictvím staticky řízených klopných obvodů (latch) a LSFR. Návrh používá řadu SR latch (256), kdy každý je vytvořen dvěma hradly NAND a výstupy jsou křížově propojeny, viz obrázek 4.1. Výstupy z latchů jsou poté připojené k funkci XOR a následně zpracovány.

Pokud je vstup pro S a R hodinový signál, je snadné dosáhnout stavu metastability. Měnící se hodinový signál má za následek větší spínací aktivitu v SR latch s větší nejistotou.



Obr. 4.1: Znázornění SR latch.

Pokud uvažujeme stav, kdy hodinový signál *Clk* je na záporné hraně, vstupy S a R změní vstupy $\{Q; QB\} = \{1; 1\}$. Následně se napěťová hrana posune z negativní na pozitivní, latch se pokusí ustálit na $\{0; 1\}$ nebo $\{1; 0\}$, což zavádí metastabilitu. Metastabilní stavy lze získat na vzestupné hraně *Clk*.

TRNG na bázi SR latch bez následného zpracování spotřebovává 865 konfigurovatelných logických bloků. Tato navrhovaná práce dosáhla propustnosti 26,6 Mbps. Pro ověření statistických vlastností TRNG byly provedeny baterie testů NIST SP 800-22.

4.2 TRNG založené na chaosu

TRNG založený na chaosu využívá chaotický systém k vytváření náhodných čísel. Tyto systémy jsou založeny na deterministických rovnicích, které jsou citlivé na počáteční podmínky. Právě Lorenz ve studii [21] z roku 1963 odhalil, že i velmi malé změny počátečních podmínek mohou mít velký dopad na výstupní data, takže tato data jsou prakticky nepředvídatelná.

Pro TRNG založený na chaosu se běžně používá například Lorenzův systém nebo systém Rösslera [22]. Tyto systémy vytvářejí neustále se měnící oscilace, které se zdají být náhodné, ale jsou vlastně založeny na deterministických rovnicích.

Například Lorenzův systém má tři diferenciální rovnice, které zjednodušeně popisují proudění vzduchu v atmosféře. Tyto rovnice mají chaotické chování a malé změny v počátečních podmínkách mohou vést k zcela odlišnému chování systému. Využitím Lorenzova systému lze vytvořit signál, který se zdá být náhodný.

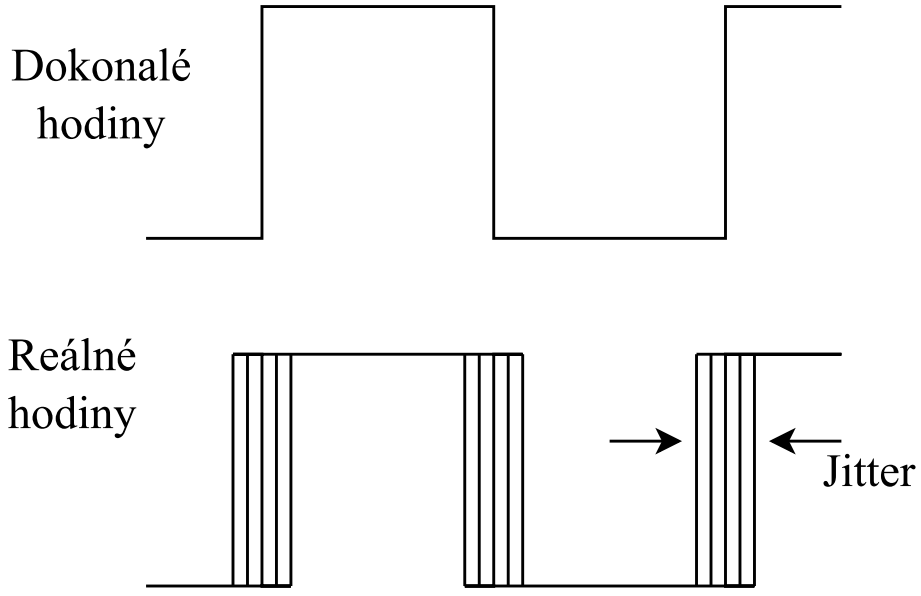
Jako relativně novou práci [23] je možné uvést implementaci Sundarapandian-Pehlivana chaotického systému z roku 2017. Jedná se o systém, který je modelován numericky pomocí čtvrtého řádu metody Runge–Kutta. TRNG obsahuje tři funkční jednotky: chaotický oscilátor, kvantování a usměrňovač. Maximální pracovní frekvence byla 293 MHz s rychlostí 58,76 Mbps. Samozřejmostí bylo úspěšné ověření pomocí statistických standardů, FIPS 140-1 a NIST 800-22.

4.3 Jitter jako zdroj entropie

Návrhy TRNG také využívají jako původce entropie jitter (náhodné vibrace) v hodinových signálech přítomných ve všech digitálních taktovaných obvodech. Protože takový signál bude dostupný ve většině aplikací, je tento přístup obzvláště zajímavý.

Jitter je termín popisující odchylku v periodickém signálu. Pokud uvažujeme digitální hodinový signál, hrana se zde nevyskytuje v přesných intervalech. Náhodný jitter následuje Gaussovo rozdělení. Toto chování je obvykle nežádoucí, ale může být použito jako zdroj náhodnosti. Implementace TRNG, které využívají zmíněný jev, toho obvykle dosahují prostřednictvím kruhových oscilátorů. Takto získaný jitter je v čase akumulován [24].

Na obrázku 4.2 je znázorněn výstup dokonalého a reálného oscilátoru, kde jitter je reprezentován čarami při každé změně hladiny.



Obr. 4.2: Znázornění jitteru.

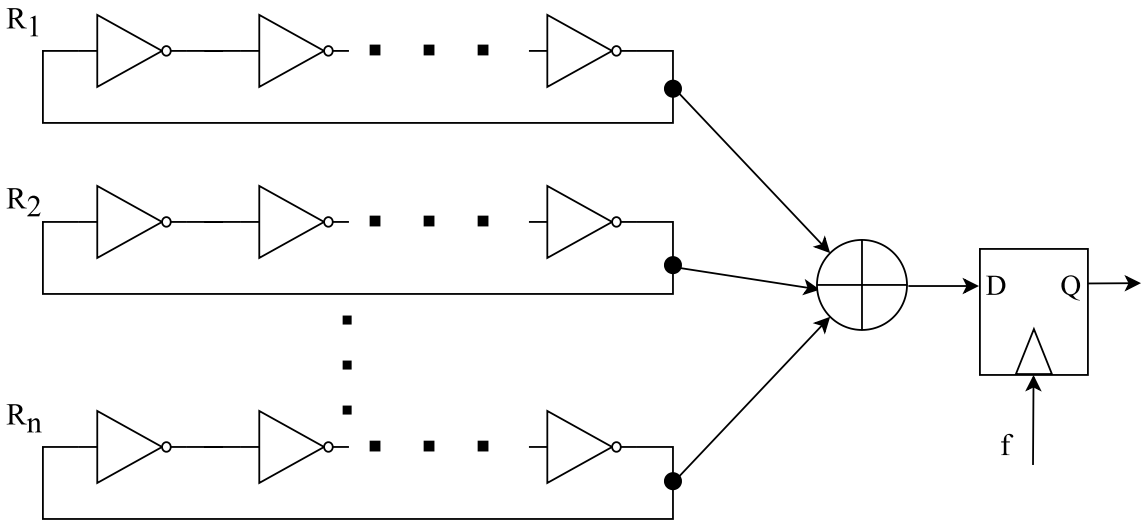
4.3.1 TRNG založené na RO

TRNG založené na prstencových oscilátorech používají jitter, který se akumuluje ve výstupních hodinách RO mezi dvěma instancemi vzorkování. Generovaný náhodný bitový tok závisí hlavně na způsobu akumulace jitteru.

Prstencový oscilátor je volně běžící oscilátor skládající se z lichého počtu logických hradel NOT (invertorů) v kruhové konfiguraci. Obecnější verze prstencového oscilátoru obsahuje jeden invertor a libovolný počet zpožďovacích hradel. To umožňuje zastavení generování hodinového signálu [19].

V práci [14] z roku 2006 byl navržen obecný model, který je schopen generovat prokazatelně náhodné bity. Tato práce se stala velmi citovanou a na základě ní mohly vznikat další vylepšené návrhy.

Středobodem práce je implementace R_1 až R_n prstencových oscilátorů, analogové výstupy z jednotlivých RO jsou vedeny do funkce XOR (binární strom) a následně je signál převeden na digitální, viz obrázek 4.3.



Obr. 4.3: Konstrukce pro kombinování a vzorkování oscilátorů.

Autoři další práce [25] zkoumali, jestli budou dva prstence vykazovat velké překrývání ve svých přechodových zónách, pokud délky prstenců budou identické. Vyšlo najevo, že nejlepších výsledků návrh dosahuje, když je délka u všech RO konstantní. Samozřejmostí je, že zvyšováním RO dochází k vyšší akumulaci jitteru. Pokud se naopak použije menší počet invertorů v RO, rozptyl frekvencí bude větší.

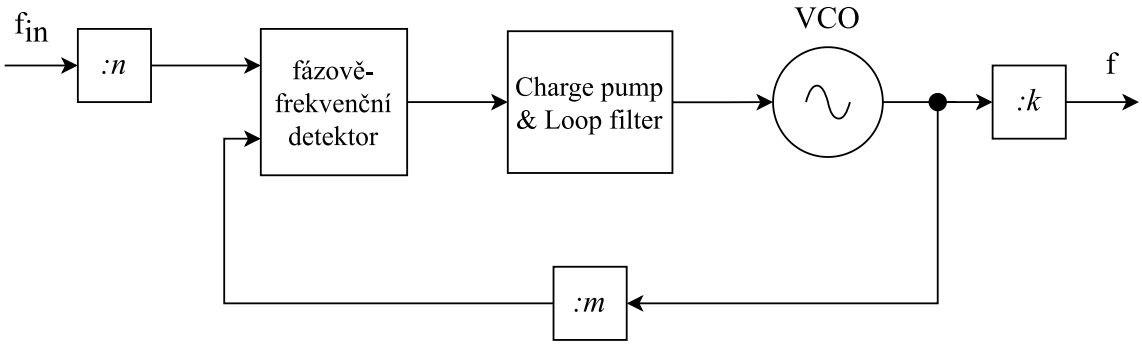
V této práci byl zkonstruován nový návrh, který vylepšuje předchozí práci [14]. Hlavní změnou je přidání klopných obvodů za každý RO. Tím dochází ke zlepšení celkové náhodnosti. Bylo použito 25 prstenců a zvolený počet inventorů v každém

prstenci byl tři. Vzorkovací frekvence byla nastavena na 100 MHz, a tím implementace dosahovala propustnosti 100 Mbps, jelikož nevyužívala žádné následné zpracování.

4.3.2 TRNG založené na PLL

Další pojetí TRNG využívá smyček fázového závěsu. Rozdíl nalézáme v tom, že PLL je součástí vestavěného obvodu a není syntetizován uvnitř FPGA. Architektura PLL obsahuje analogový napěťově řízený oscilátor (VCO), fázově-frekvenční detektor ve zpětnovazební smyčce, viz obrazek 4.4. Různý šum způsobuje kolísání frekvence interního oscilátoru (VCO). Fázově-frekvenční detektor porovnává fázi a frekvenci jak vstupních, tak zpětnovazebních hodin a generuje vzestupný nebo sestupný signál na základě rozdílu mezi nimi, aby určil, zda má VCO pracovat při vyšší nebo nižší frekvenci. Tato změna je právě vnímána jako jitter [26].

Výstupní frekvence je poté dána jako $f = f_{in} * K_M/K_D$, kde f_{in} je vstup z referenčních hodin, K_M je hodnota dělicího čítače vloženého do zpětnovazební smyčky PLL a $K_D = n * k$, n značí čítač umístěný za vstupem hodin a k je hodnota čítače těsně před výstupem (pro každý výstup hodin je jiný). Tímto způsobem bude výstup klopného obvodu nedeterministický alespoň v jednom vzorku během jedné periody hodin klopného obvodu.



Obr. 4.4: Zjednodušený diagram PLL.

V konkrétní práci [27] z roku 2015 návrháři využívají čip Kintex 7 pro lepší zdroj entropie – blok PLL má 6 výstupů a vzorkování je prováděno na každém z nich. Pro zmenšení post-processingu byly využity dvě PLL. Následné zpracování je založeno na funkcích XOR a po šestnáctihodinových cyklech je vytvořen náhodný bit. Vzorkovací frekvence byla zvolena 100 MHz a výsledná propustnost TRNG je 6,25 Mbps. Vygenerované výstupní bity prošly nejznámějšími bateriemi testů náhodnosti s velmi dobrými výsledky.

4.4 Shrnutí

V této kapitole byly popsány a analyzovaný různé druhy dosavadních návrhů, které se zabývají generátory náhodných čísel. Existuje celá škála RNG, které používají různé druhy entropie a které se různými způsoby vypořádávají s následným zpracováním bitů.

Ukázány byly nejvíce používané zdroje entropie pro RNG, konkrétně metastabilita SR latch, chaotické systémy, jitter z kruhových oscilátorů a jitter ze smyček fázového závěsu. Problematika těchto návrhů byla stručně vysvětlena.

Všechny práce úspěšně prošly různými druhy statistických sad. Srovnání předvedených generátorů je ukázáno v tabulce 4.1, kde každá práce využívá jiný zdroj entropie. Nejvyšší propustnosti a zároveň frekvence dosahuje implementace založená na chaosu, má však nejvyšší využití zdrojů.

Tab. 4.1: Srovnání zmíněných návrhů RNG

Práce	Zdroj entropie	Propustnost [Mbps]	Frekvence [MHz]	Využití zdrojů
Sridevi et. al [20]	metastabilita	26,6	27	1273 LUT
Koyuncu et. al [23]	chaotický systém	58,7	293,8	43391 LUT
Sunar et. al [14]	jitter RO	2,5	40	–
Wold et. al [25]	jitter RO	100	100	83 LUT
Deák et. al [27]	jitter PLL	6,25	100	19 slice, 2 PLL

Nejčastější možností, na které autoři hardwarových RNG staví svůj návrh, je technika kruhových oscilátorů. RO také poskytují dobré výsledky v poměru propustnosti a spotřeby hardwarových zdrojů. Nutno podotknou, že tyto návrhy je těžké mezi sebou přímo porovnat. Fungují totiž na jiných principech, mají různé bezpečnostní vlastnosti a také jsou testovány na jiných zařízeních.

Například z výzkumu [28] vyplývá, že dochází k rozdílu mezi výrobci, kteří používají jiné technologie – paměti SRAM a flash. Nejvyšší rozptyl frekvence u RO je dosažen použitím tří invertorů a to u FPGA založených na SRAM. Čip Actel (flash) dosahuje nejvyšší disperze až pří použití devíti invertorů. Bylo také pozorováno, že úroveň rozptylu je odlišná mezi zařízeními Xilinx a Altera. Zejména při nízkém počtu invertorů, kde je disperze vyšší pro Xilinx FPGA.

5 Měření dostupných hardwarových implementací TRNG

V následující kapitole jsou uvedeny výsledky již navržených implementací. Všechny představené implementace jsou dostupné pod veřejnými licencemi. Ověřena je správná funkčnost TRNG, zjištění využití potřebných zdrojů a ověření náhodnosti skrze NIST STS, viz kapitola 3.2.

Pro toto měření byl využit volně dostupný software Vivado 2022.2 od firmy Xilinx. Implementace byla prováděna pro cílové zařízení Zybo Z7-20 od Digilent s FPGA čipem Xilinx Zynq řady 7, kde následně probíhalo generování náhodných sekvencí.

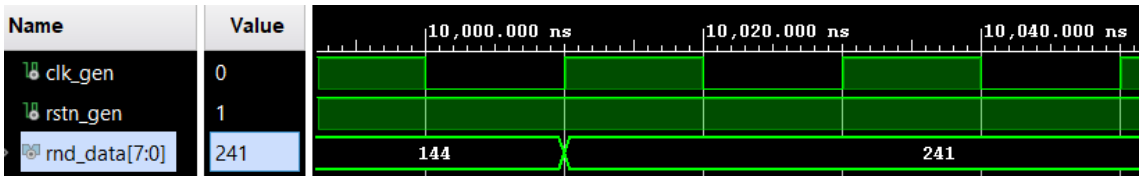
5.1 TRNG využívající krátké a dlouhé RO

Projekt [29] využívá fázový šum volně běžících a křížově vázaných prstencových oscilátorů jako zdroj entropie. NeoTRNG je vyhodnocován jako součást procesoru NEORV32, kde je neoTRNG k dispozici jako standardní SoC modul. Procesor byl autorem syntetizován pro Intel Cyclone IV běžící na 100 MHz.

TRNG se skládá ze zvoleného počtu entropických buněk. Každá buňka obsahuje přepínatelný prstencový oscilátor, který poskytuje „režim rychlé oscilace“ (používá krátký oscilační řetězec) a „pomalý oscilační režim“ (používá dlouhý kruhový oscilační řetězec). Výstup předchozí/následující buňky určuje, jaký režim bude použit. Výsledek je vzorkován posuvným registrem a v tomto okamžiku se entropie shromažďuje vzorkováním fázového šumu výstupu úplně poslední buňky. Pro zlepšení náhodnosti lze aktivovat volitelnou logiku následného zpracování.

Uživatel má možnost si navolit celkový počet RO. Další možností je nastavit počet hradel NOT v prvním RO, který by zpravidla měl být lichý. V konfiguraci nemusí každý RO obsahovat stejný počet invertorů, ale jejich počet může lineárně narůstat o zvolený koeficient. Jelikož tento typ generátoru zavádí výše zmíněné dva řetězce, poslední možností je tak nastavení dalšího počtu invertorů, který se použije pro dlouhý řetězec. Například pokud navolíme počet inventorů v první buňce pět, dva v každé další, v dlouhém řetězci jich bude o tři více. Chceme-li zjistit počet těchto inventorů v šesté buňce, výsledkem je patnáct hradel pro krátký a osmnáct pro dlouhý řetězec.

Následující obrázek 5.1 znázorňuje simulaci generování náhodných čísel, která je poskytnuta návrháři. Avšak jedná se pouze o ukázku PRNG, jelikož asynchronní kruhové oscilátory nelze simulovat na úrovni RTL.



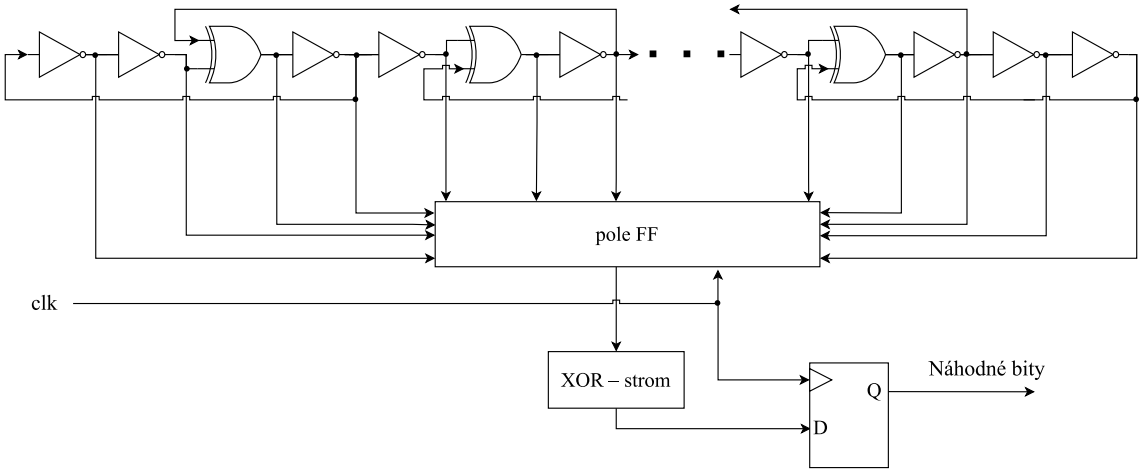
Obr. 5.1: Ukázka simulace generovaných čísel.

5.2 TRNG využívající zřetězené kruhové oscilátory

Na principu kruhových oscilátorů staví jiná volně dostupná implementace [30], která je v návrhu poměrně jednoduchá. Použití kruhových oscilátorů je odlišné, než jak je uvedeno v kapitole 4.3.1. Využívají se zde tzv. zřetězené kruhové oscilátory – hradla typu XOR umožňují kombinaci signálu ze sousedního kruhového oscilátoru, zvýší se tak rozsah jitteru. RO se v implementaci skládá vždy ze tří invertorů, lze pouze měnit celkový počet RO. Z každého hradla je signál veden do odpovídajícího klopného obvodu, kterých je stejný počet jako hradel. Výstupy ze všech těchto FF jsou postupně xorovány, dokud nezbude jediný signál. Výsledný signál je převeden vzorkováním na náhodné bity.

V tomto návrhu je umožněno pouze omezeně nastavit počet RO, a to z toho důvodu, že se počet musí rovnat $3 * n + 1$, kde $n \in \mathbb{Z}$. Pokud by se tak nestalo, zřetězený RO by přestal oscilovat a signály by se nastavily (uzamkly) pouze na jednu určitou hodnotu. Návrh implementuje 16 zřetězených RO.

Celková architektura TRNG, viz obrázek 5.2, kombinuje dva náhodné zdroje entropie – jitter z kruhových oscilátorů a metastabilní stav FF.

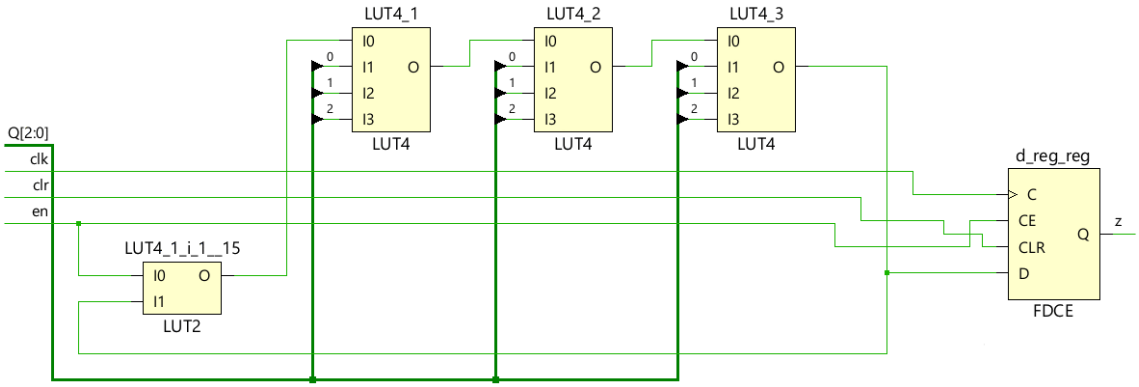


Obr. 5.2: Zjednodušená struktura navrženého TRNG.

5.3 TRNG využívající programovatelné zpožďovací linky

V pořadí třetí volně dostupná implementace [31] opět nepoužívá pouhé RO, ale navíc k nim uplatňuje programovatelné zpožďovací linky (PDL – Programmable Delay Line). Kdybychom uvážili stejně dlouhé kruhové oscilátory, pak by pravděpodobně mohly korelovat kvůli identickým zpožděním. PDL tento problém řeší tak, že výstup pro každý vzorkovací takt je jiný.

Každý RO se celkem skládá ze čtyř LUT, která představují jednotlivá hradla. První vyhledávací tabulka ztvárňuje hradlo typu AND, ty následující jsou typu NOR. Aby mohlo být využito PLD, jsou právě tyto inventory konfigurovány v podobě čtyřvstupové LUT – využití různě dlouhých cest pro zpoždění. Vzorkovací klopný obvod se nachází hned za RO z důvodu, aby vysoký počet spínacích aktivit neležel pouze na následném XOR stromu a konečného FF pro vzorkování. Na obrázku 5.3 je možné spatřit výřez schématu zobrazující kruhový oscilátor, který je tvořen zmíněnými LUT, a D klopný obvod po syntéze.



Obr. 5.3: Schéma kruhového oscilátoru a klopného obvodu.

Práce implementuje právě 32 RO a architektura v základu neobsahuje žádné další funkce následného zpracovaní pro ještě lepší náhodnost.

5.4 Testování a srovnání

V kapitole byl proveden rozbor tří veřejně dostupných hardwarových implementací TRNG. Všechny výše zmíněné návrhy používají kruhové oscilátory, ale zásadně se od sebe odlišují v tom, jakým způsobem jsou tyto RO implementovány – využití různě dlouhých RO, zřetězených RO a zpožďovacích linek. V následující části jsou tyto implementace srovnány, a to z pohledu kvality náhodnosti a využití zdrojů.

Pro testování bylo použito linuxové jádro v distribuci Debian, který byl spuštěn skrze virtuální prostředí Oracle VM VirtualBox. Pro správné spuštění testovací sady je nutné doinstalovat kompilátor GCC. Následující obrázek 5.4 zobrazuje výřez průběhu NIST STS – aplikace všech testů, ponechání výchozích parametrů, zvolení počtu sekvencí a následné vybrání formátu vstupního souboru.

```
            S T A T I S T I C A L   T E S T S
            _________________________________

 [01] Frequency                       [02] Block Frequency
 [03] Cumulative Sums                 [04] Runs
 [05] Longest Run of Ones             [06] Rank
 [07] Discrete Fourier Transform      [08] Nonperiodic Template Matchings
 [09] Overlapping Template Matchings  [10] Universal Statistical
 [11] Approximate Entropy             [12] Random Excursions
 [13] Random Excursions Variant       [14] Serial
 [15] Linear Complexity

      INSTRUCTIONS
         Enter 0 if you DO NOT want to apply all of the
         statistical tests to each sequence and 1 if you DO.

Enter Choice: 1

     P a r a m e t e r   A d j u s t m e n t s
     ----------------------------------------
 [1] Block Frequency Test - block length(M):         128
 [2] NonOverlapping Template Test - block length(m): 9
 [3] Overlapping Template Test - block length(m):    9
 [4] Approximate Entropy Test - block length(m):     10
 [5] Serial Test - block length(m):                  16
 [6] Linear Complexity Test - block length(M):       500

Select Test (0 to continue): 0

How many bitstreams? 100

Input File Format:
 [0] ASCII - A sequence of ASCII 0's and 1's
 [1] Binary - Each byte in data file contains 8 bits of data

Select input mode:  0

  Statistical Testing In Progress.........

  Statistical Testing Complete!!!!!!!!!!!!
```

Obr. 5.4: Ukázka spuštěné terminálové aplikace NIST STS.

Tabulka 5.1 reflektuje patnáct testů z baterie NIST STS. Dokumentace NIST uvádí, že na hladině významnosti α má být otestováno minimálně α^{-1} sekvencí. U některých testů je nutné splnit podmínku, že testovaná sekvence musí být delší než 10^6. V našem případě se jedná o testování 100 sekvencí ($\alpha = 0,01$), kdy každá

má délku 10^6 bitů. Celkově je generováno a testováno číslo o délce 100 Mb.

Tab. 5.1: Výsledky baterie NIST STS pro vygenerované bity

Implement. / **Název testu**	[29]		[30]		[31]	
	p	%	p	%	p	%
Frekvenční test (96)	0,0046	97	0,4750	100	0,5955	100
Frekvenční test (96) uvnitř bloku	0,1296	100	0,2492	100	0,3190	98
Test série bitů (96)	0,1626	97	0,7399	99	0,5141	99
Test nejdelší sekvence (96) jedniček v bloku	0,7981	97	0,0757	99	0,4373	99
Test sérií binárních matic (96)	0,4011	100	0,0109	100	0,1537	100
Test diskrétní Fourierovy (96) transformace	0,8677	99	0,3345	98	0,8832	97
Test nepřekrývajících(96) se vzorů	0,5892	97-100	0,5713	97-100	0,7581	97-100
Test překrývajících (96) se vzorů	0,6579	99	0,9780	98	0,5749	100
Mauerův univerzální (96) statistický test	0,0401	97	0,7792	98	0,3345	100
Test lineární složitosti (96)	0,3838	99	0,7197	100	0,6579	100
Test sérií (96)	0,8165	98	0,6243	99-100	0,5341	100
Přibližný test entropie (96)	0,3215	96	0,1626	100	0,8831	99
Test kumulativní sumy (96)	0,0032	97	0,0329	100	0,4190	100
Test náhodných návštěv (94/93/94)	0,5542	98-100	0,8824	95-100	0,6475	96-100
Test náhodných variant návštěv (94/93/94)	0,7584	98-100	0,2070	98-100	0,7921	98-100

Pro úspěšný test je nutné, aby zdárně prošlo minimálně 96 sekvencí. Pro test náhodných návštěv (variant) je minimální hranice pro každý vzorek dat jiná – 94/100, 93/100 a 94/100 (minimální/maximální počet úspěšných sekvencí pro TRNG uvedené v tabulce zleva). Kolik sekvencí prošlo, je možné vidět ve druhém sloupci každého TRNG. Tím je možné konstatovat, že všechny RNG zdárně prošly. Z tabulky je dále patrné, že o trochu horších výsledků dosahuje implementace [29].

Vypočtena p-hodnota pro každou sekvenci nás informuje o tom, jestli sekvence projde. Tyto hodnoty by v rámci každého testu měly pocházet z rovnoměrného rozložení. O této skutečnosti nás informuje první sloupec jednotlivých TRNG.

Následující obrázek 5.5 ukazuje příklad výstupu aplikace NIST STS. Zmíněné p-hodnoty pro každou sekvenci jsou zanesené do konkretního intervalu $[0,0;0,1), [0,1;0,2), \ldots, [0,9;1,0)$, které představují sloupce C1, C2, ..., C10.

```
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <data/TRNG3.txt>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
  8   8  12  12  12   7   8  14  13   6  0.595549    100/100     Frequency
 15  14  13   6   8  11   6  12   7   8  0.319084     98/100     BlockFrequency
  7   9  11  11   8  12   8  17   8   9  0.554420    100/100     CumulativeSums
  7  10   9  10  14  14   8   4  13  11  0.419021    100/100     CumulativeSums
  9  13  11   9   7   9   8  12  16   6  0.514124     99/100     Runs
 10   5  14   9   7  13   9  14  12   7  0.437274     99/100     LongestRun
 10   6  15  11  10  16   5  13   8   6  0.153763    100/100     Rank
 10  10  15  11   8   7  10   8  10  11  0.883171     97/100     FFT
 12   9   9  11  12  10   9  13   9   6  0.924076    100/100     NonOverlappingTemplate
 13  12   9  13  10   8   5   9  11  10  0.798139     99/100     NonOverlappingTemplate
  6  13  13   7  10   7  14  13   9   8  0.514124     99/100     NonOverlappingTemplate
 12  13   9  14  10   9  10   9   6   8  0.816537     98/100     NonOverlappingTemplate
  6   9  16  10  21   8   6   7  11   6  0.008879    100/100     NonOverlappingTemplate
```

Obr. 5.5: Ukázka vyhodnocení náhodných sekvencí NIST STS.

Výsledky syntézy jednotlivých implementací jsou srovnány v tabulce 5.2. Z pohledu využití hardwaru je náročnější TRNG využívající PLL [31], jelikož je implementace navržena tak, aby implementovala 32 RO. Z hlediska maximální frekvence je horší práce [29] – má poměrně složitý návrh a post-processing.

Tab. 5.2: Srovnání RNG

Implementace	LUT	FF	Spotřeba stat./dyn. [W]	Max. frekvence [MHz]
[29]	61	106	0,138/0,010	147,4
[30]	58	49	0,138/0,018	174,8
[31]	151	57	0,138/0,016	172,5

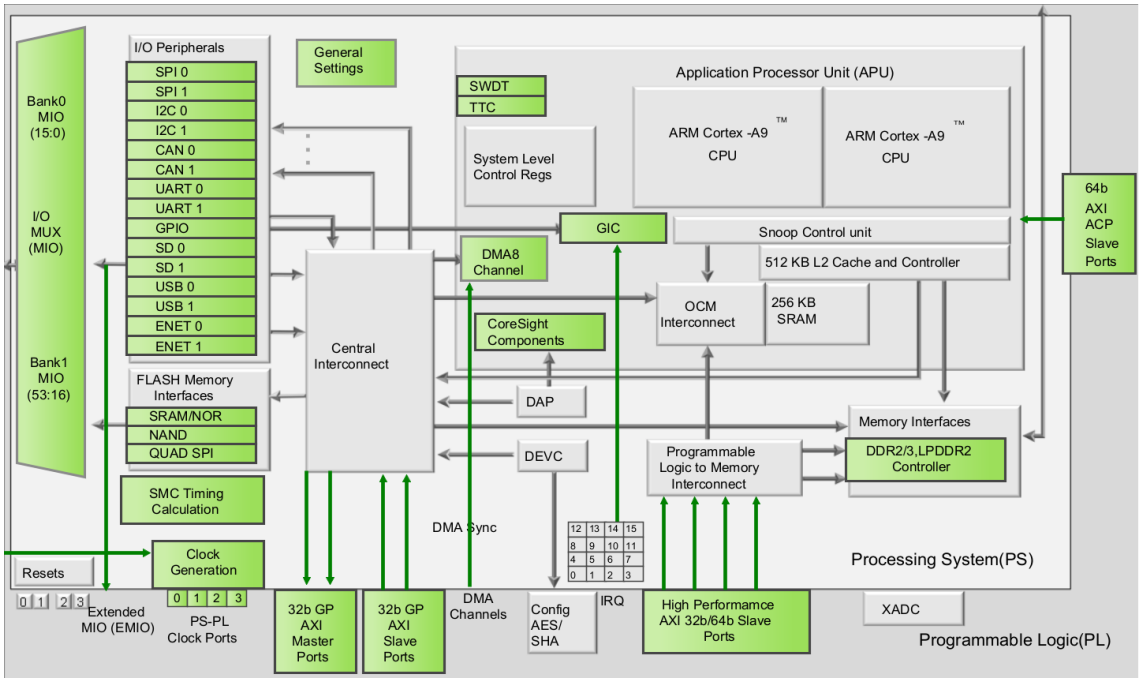
6 Vlastní hardwarová implementace TRNG

Kapitola obsahuje popis řešení stěžejního úkolu této práce. Nejprve následují základní vlastnosti použité vývojové desky s jejím FPGA čipem. Dále jsou krátce objasněna uplatněná vývojová prostředí. V hlavní části je řešena implementace hardwarového generátoru náhodných čísel, který je založen na kruhových oscilátorech, viz kapitola 4.3.1.

6.1 Cílová platforma

Cílovou platformou je FPGA čip od firmy Xilinx a to Zynq 7000 s označením xc7z020-1clg400c. Tento čip je umístěn na vývojovou desku Zybo Z7 od Digilent. Čip Zynq integruje dva subsystémy. První je označován jako programovatelná logika (PL – Programmable Logic) a obsahuje jednotlivé programovatelné prvky, tudíž v základu čistě FPGA. Jako další je Processing System (PS), který zahrnuje dvoujádrový procesor ARM Cortex-A9 a dále obsahuje řadu periferních rozhraní [32].

Na desce je možné nalézt rozhraní pro USB, microSD kartu, pro síťový provoz využívá rozhraní RJ45. Komunikace PC s deskou probíhá skrze micro USB konektor přes UART/JTAG. Pro zpracovaní obrazu je deska osazena HDMI rozhraním a DDR3L pamětí o velikosti až 1 GB. Konfigurovat je dále možné šest tlačítek, čtyři přepínače a sedm LED. Zmíněnou architekturu čipu znázorňuje obrázek 6.1 a hardwarové vlastnosti desky zobrazuje tabulka 6.1.



Obr. 6.1: Architektura Zynq 7000.

Tab. 6.1: Dostupné zdroje FPGA karty

Logické buňky	Řezy	LUT	FF	BRAM [kB]	Speed grade
85000	13300	53200	106400	630	−1

6.2 Vývojová prostředí

Pro tuto práci byla zvolena dvě volně dostupná vývojová prostředí od společnosti Xilinx. Vivado 2022.2 je první z nich a jedná se o komplexní nástroj pro návrh a následnou konfiguraci FPGA obvodu. Zahrnuje tak všechny kroky popsané v kapitole 1.2. Dochází tedy k využití programovatelné logiky (PL) skrze jazyk pro popis hardwaru a to konkrétně VHDL.

Druhým prostředím je Vitis Platform 2022.2, který umožňuje softwarový přístup a komunikace probíhá s Processing system (PS). Pokud chceme komunikovat s programovatelnou logikou vytvořenou ve Vivado, je nutné nejprve exportovat XSA (Xilinx Support Archive) soubor. Ze souboru se následně ve Vitis vytvoří projekt. Programovat je možné v jazyku C/C++. Konečná konfigurace zařízení probíhá skrze Vitis.

6.3 Implementace TRNG

Implementace navrženého generátoru je založena na kruhových oscilátorech a operacích XOR. Návrh se opírá o vědecký článek [33] vydaný ke konci roku 2022. Celkový návrh se skládá ze čtyř hlavních komponent – dvě komponenty pro kruhové oscilátory, další pro samostatný TRNG a poslední, která vytváří a spojuje paralelní instance TRNG. Následuje bližší popis implementovaných komponent.

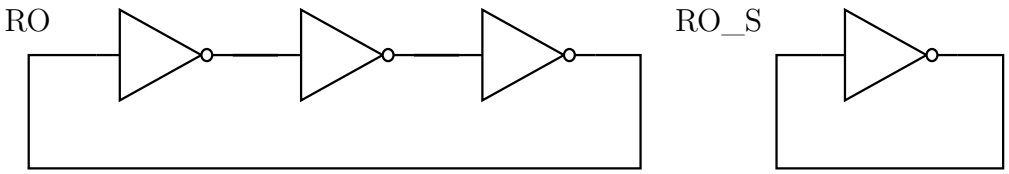
6.3.1 Komponenta kruhových oscilátorů

Jedná se o jádro navrženého generátoru, tyto volně běžící oscilátory představují zdroj entropie. Na výstupu každého RO dochází k akumulaci jitteru, jak již bylo vysvětleno v kapitole 4.3. Jitter je odchylka v přechodové době napěťové úrovně logických hradel, která je ovlivněna různými faktory (šum, teplota, ...).

TRNG využívá dva druhy RO – dlouhý a krátký. V každém z nich se musí nastavit lichý počet logických hradel NOT, v případě sudého počtu by kruh udržoval stále stejné hodnoty, a tím by přestal oscilovat. Jednotlivé výstupy z komponent (RO) jsou vedeny do funkce XOR. Krátký prstenec je zde zejména použit pro generování

krátkých impulzů využitých ve funkcích XOR, kde zvyšují celkovou náhodnost, viz dále.

V implementaci je možné volně konfigurovat počet krátkých a dlouhých prstenců, tak jako množství hradel NOT. Jediné omezení se nachází v počtu RO, který musí být sudý (dělitelný dvěma). Pro optimální výsledky je počet hradel (NUM_INV) tři, respektive jedno pro krátký prstenec (NUM_INV_S). Obrázek 6.2 znázorňuje vnitřní uspořádání RO na úrovni RTL, po syntetizovaní návrhu jsou jednotlivé brány NOT převedena do podoby LUT o jednom vstupu.



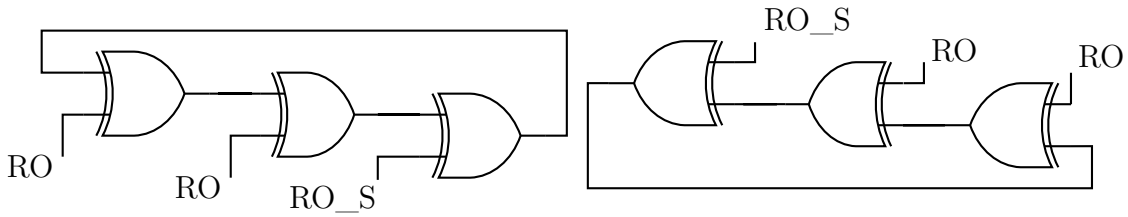
Obr. 6.2: Znázornění navržených RO.

6.3.2 XOR prstenec

Mimo prstenců sestavených z invertorů, návrh využívá jiný druh prstenců, které jsou tvořený pouze hradly XOR. Hojně se využívají při návrzích TRNG pro kombinaci bitů v závěrečné fázi – zlepšují celkovou náhodnost a poměr nul a jedniček.

Důležité je, aby operace XOR nebyly přímo míchány k hradlům NOT, ale využívaly svůj vlastní prstenec. Předchází se tak situaci, že krátké impulsy by mohly být eliminovány uvnitř takového oscilátoru, což se může negativně projevit na rychlosti akumulaci jitteru (jak plyne z [33]). Tyto impulsy vznikají při rychle měnícím se vstupním signálu, kdy hradla nedosáhnou plné napěťové úrovně.

Krátké impulsy z XOR změní fázi a amplitudu dalšího výstupu z následujícího hradla XOR, a tím ovlivní oscilaci obvodu. První vstup XOR je tedy výstup předchozího hradla, na druhý vstup je přiveden signál z oscilujících invertorů. Hradla XOR tvoří dva na sobě nezávislé prstence, jejich celkový počet odpovídá počtu RO skládajících se z hradel NOT. Následující obrázek 6.3 zobrazuje propojení hradel XOR a čtyř RO, respektive dva krátké RO.



Obr. 6.3: Znázornění navržených XOR.

6.3.3 Komponenta generátoru

Tato komponenta generuje náhodný proud dat o šířce jednoho bitu. Modul se stará o vytvoření požadovaného počtu RO a propojení. Signály z RO propojuje s oscilačním kruhem XOR. Hradla XOR tvoří vždy dva oddělené prstence.

Každý výstup z operací XOR je ještě veden ke klopným obvodům typu D, které zajišťují „sběr“ náhodnosti – vzorkování výstupu XOR zvolenou frekvencí. Pokud vstupní signál obsahuje velký počet přechodů, zvyšuje se pravděpodobnost, že během procesu vzorkování bude mít FF metastabilní stav (viz kapitola 4.1), což zlepšuje celkovou náhodnost.

Následujícím procesem je XOR operace, která opět zlepšuje náhodnost a zejména udržuje podobný počet nul a jedniček. Dále bylo ještě přikročeno k implementaci závěrečných FF, jelikož se objevovaly zdvojené znaky oddělující jednotlivá čísla a také docházelo k horším statistickým výsledkům (Test sérií).

6.3.4 Komponenta TRNG_top

Přídavek „top“ mají komponenty, které jsou v hierarchii nejvýše a případně obsahují celkové propojení a nastavení. Modul obsahuje následující generické parametry, které je možné upravovat. Úprava *NUM_INV*(_*S*) má za následek změnu počtu bran NOT v dlouhém (krátkém) oscilačním kruhu. Dalším nastavením je počet dlouhých, respektive krátkých RO, který je v parametrech označován jako *NUM_RO*, respektive *NUM_RO_S*.

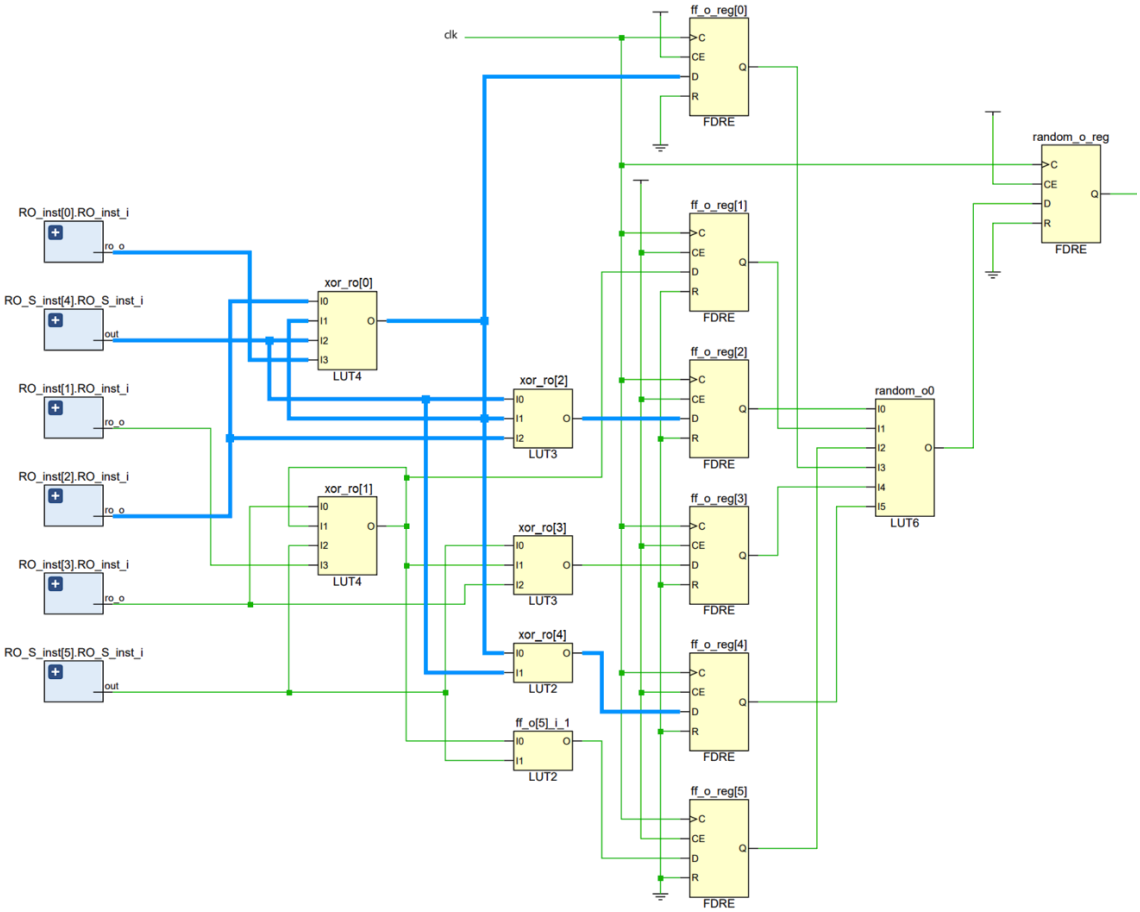
Cílem návrhu bylo také generovat vícebitová čísla, jakožto jednu z předností FPGA bylo využito více komponent TRNG pro paralelní provoz. V konkrétní implementaci jsou výsledná čísla o velikosti 32 bitů, která jsou vytvořena spojením výstupů z 32 komponent TRNG. V nastavení je to opět možné změnit parametrem *NUM_TRNG*, který se tedy rovná velikosti výsledného čísla v bitech.

Avšak nutno poznamenat, že jakékoliv změny návrhu znamenají detailnější pohled na výstupní data, především jde o výsledky statistických testů. Do určité míry

pomáhá k lepším výsledkům změna vzorkovací frekvence. Typicky je optimální vzorkovací frekvence určena metodou pokus-omyl.

Pokud tedy chceme, aby tento TRNG běžel několikrát paralelně je nutné udělat změny v počtu RO v jednotlivých instancích TRNG. Jinak se setkáváme s neúspěšnými výsledky statistických testů. Především jde o test serií, entropie, nepřekrývajících se vzorů a samozřejmě frekvenční test. V implementaci se s tím počítá a cesta k úspěšným testů vede přes rozdílný počet RO v jednotlivých TRNG. Každý druhý je tak upraven, aby obsahoval o dva RO méně – parametr *N*.

Na následujícím obrázeku 6.4 je pohled na schéma TRNG o pěti RO po závěrečném procesu implementace. Modrou barvou je poté znázorněn oscilační XOR prstenec, jeho vstupy a výstupy. Je zřejmé, že tento prstenec je namapován do podoby LUT, která jsou o dvou a třech vstupech.



Obr. 6.4: Komponenta TRNG po procesu implementace.

Implementace pracuje se zpětnými vazbami, je tak nutné v souboru omezení tyto časové smyčky povolit (*set_property ALLOW_COMBINATORIAL_LOOPS*

true [get_nets <myHier/myNet>]). V opačném případě nebude možné přikročit ke konfiguraci zařízení.

6.4 Blokový návrh

Cílem kapitoly je popis implementace UART rozhraní, tedy realizace sériového přenosu náhodných dat z vývojové desky do PC. Toho je docíleno propojením PS a PL skrze blokový návrh. Obrázek 6.5 ukazuje konečnou blokovou architekturu TRNG.

Jádrem blokového schématu je *ZYNQ7 Processing System*, dále implementovaná komponenta *TRNG_top*, ostatními neméně důležitými IP (Intellectual Property) bloky je *Processor System Reset*, *AXI Interconnect* a *AXI GPIO*.

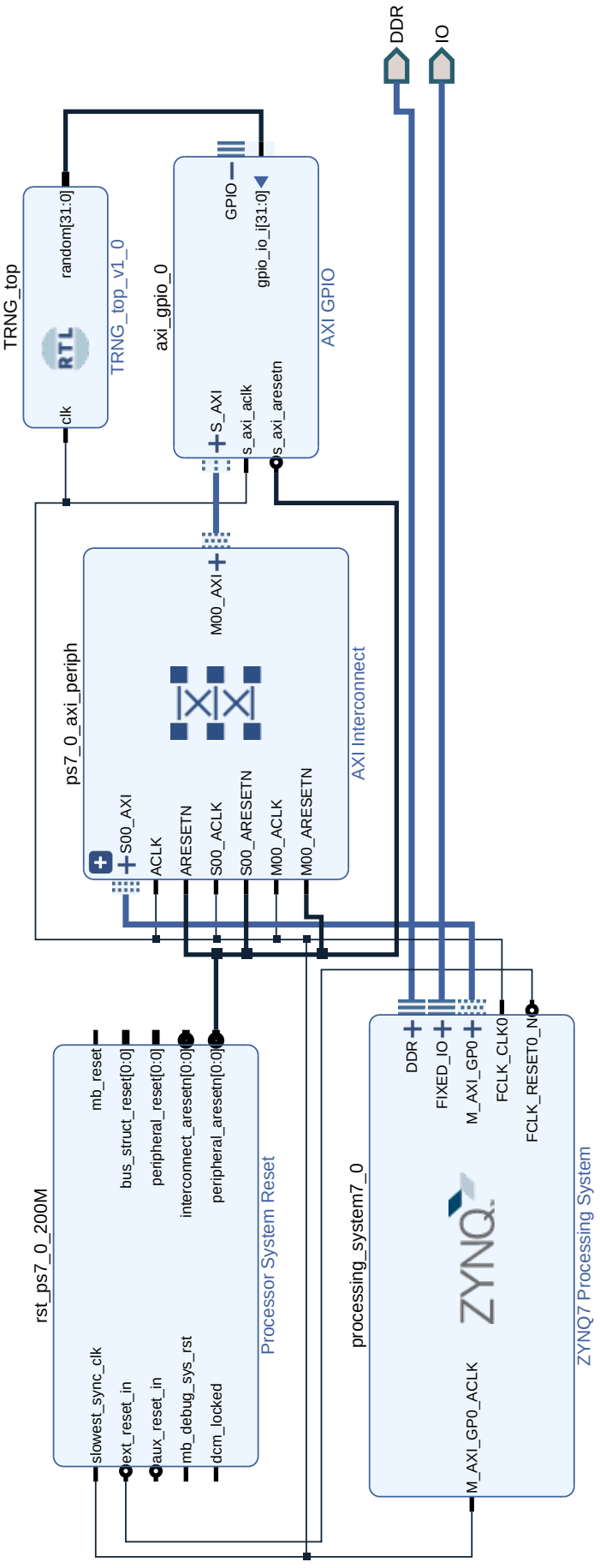
ZYNQ7 Processing System funguje jako most mezi PS a PL a současně pomáhá integrovat vlastní IP bloky. Procesor se stará hlavně o komunikaci s I/O prvky a periferiemi na FPGA kartě. Vše, co je možné obsluhovat, bylo vyjmenováno v kapitole 6.1 a přiloženém obrázku. K úspěšnému přenosu dat z implementovaného generátoru do PC je zapotřebí aktivovat rozhraní UART a zvolit požadovanou přenosovou rychlost. Navíc slouží ke generování čtyř referenčních systémových hodin, z nichž jsou jedny využité pro samotný TRNG.

Processor System Reset poskytuje přizpůsobené resety (přerušení) pro blokový návrh, které jsou odvozené z externího resetu, tedy ze *ZYNQ7 PS*.

Úkolem *AXI Interconnect* bloku je propojit AXI (Advanced eXtensible Interface) komunikační kanály tak, aby se mezi IP bloky mohla přenášet data, řídící signály a další informace. Blok je zejména důležitý při použití více AXI bloků, směruje datové toky. Hlavní AXI rozhraní je přímo umístěné na bloku *ZYNQ7 PS*, následně jako sekundární rozhraní v návrhu figuruje *AXI GPIO*.

AXI GPIO je univerzální vstupně/výstupní (General Purpose Input/Output) blok, který se stará o propojení signálů mezi procesorem a periferními zařízeními, jako jsou LED diody, tlačítka a další. Blok je v tomto konkrétním návrhu nastaven tak, aby přijímal pouze datové vstupy o specifické šířce kanálu (max. 32 bitů), které jsou následně určené pro *ZYNQ7 PS*.

Navržená komponenta *TRNG_top* byla blíže specifikovaná v předešlé kapitole 6.3 a zde je pouze převedena do blokového návrhu. Výstup bloku, tedy náhodné bity, jsou právě připojeny k *AXI GPIO*. Dochází tak k celkovému zprovoznění UART rozhraní na úrovni hardwaru.



Obr. 6.5: Bloková struktura TRNG.

6.5 Propojení s procesním systémem

V této části je vysvětlen postup závěrečného spojení mezi PL a PS. To je umožněno skrze druhé vývojové prostředí – Vitis, přes které probíhá i konfigurace FPGA karty. Připojená vývojová deska se poté objeví jako sériové rozhraní ve zvoleném komunikačním programu (např. Tera Term, PuTTY), kde je možné pozorovat přijímaná data.

Pro potřeby Vitis je nutný soubor XSA, který je možné vytvořit ve Vivado možností export hardwaru včetně bitstreamu. Ze souboru se ve Vitis vytvoří softwarová platforma, která zahrnuje hardwarovou specifikaci a nastavení softwarového prostředí. Následuje vytvoření systémového projektu, kde se specifikuje námi vytvořená platforma a dojde k propojení. V tomto systémovém projektu již můžeme vytvořit požadovanou aplikaci (softwarový projekt), například v jazyku C.

Vytvořená jednoduchá aplikace inicializuje všechny potřebné instance a zaručuje čtení náhodných dat, která jsou posílaná z TRNG komponenty. Následuje závěrečná konfigurace FPGA karty. Náhodná čísla se poté objeví v příslušném terminálovém okně, kde je možné je dále ukládat a provádět bližší analýzu.

7 Validace a výsledky vlastní implementace

Závěrem bakalářské práce je seznámit čtenáře s výsledky implementovaného návrhu, to zahrnuje potřebné zdroje návrhu, ověření kvality generátoru a srovnání s jinými TRNG.

7.1 Hardwarové nároky a další vlastnosti

Potřebné zdroje po procesu implementace by v konkrétní konfiguraci měly dosahovat 950 LUT a 1009 FF. V tabulce 7.1 jsou detailněji rozepsány tyto zdroje. Tabulka také obsahuje maximální počet jednotlivých prvků na FPGA čipu a procentuální využití zdrojů. Do těchto výsledků je započítán celkový blokový návrh. V souvislosti s maximálními zdroji FPGA karty, se jedná o poměrně „malou“ implementaci.

Tab. 7.1: Potřebné zdroje návrhu

Název	Spotřebované zdroje	Dostupné zdroje	Poměr v %
LUT	950	53200	1,79
LUT jako RAM	60	17400	0,34
FF	1009	106400	0,95
BUFG	1	32	3,13
Řez	386	13300	2,90

Následující 7.2 tabulka zobrazuje hardwarové nároky jednotlivých částí návrhu. Můžeme si všimnout, že blok *ZYNQ7 Processing System* nevyužívá žádné zdroje. Je to z toho důvodu, že sám procesní systém neobsahuje žádnou programovací logiku (je typu ARM). Pouze s ní komunikuje prostřednictvím vstupů a výstupů. Naopak má největší odběr výkonu, využívá 97 % celkové dynamické spotřeby. Statická spotřeba implementace činí 0,138 W a celkově s dynamickou složkou návrh spotřebovává 1,57 W.

Pokud se podíváme blíže na navrženou komponentu generátoru (*TRNG_top*), tak pro vygenerování 32 bitového čísla spotřebuje 455 LUT a 192 FF. Jednotlivé jednobitové generátory spotřebují 19 LUT a 7 FF, při rozvržení čtyř dlouhých a dvou krátkých RO. Větší spotřeba LUT je zejména proto, že jednotlivé jednovstupové LUT (inventory) jsou oddělené a nejsou spojené do větších vícevstupových LUT.

Tab. 7.2: Potřebné zdroje jednotlivých bloků

Název	LUT	FF	Dyn. spotřeba [W]
AXI GPIO	94	318	0,007
ZYNQ7 Processing System	0	0	1,403
AXI Interconnect	384	466	0,010
Processor System Reset	17	33	0,001
TRNG_top	455	192	0,012

Optimální frekvence je stanovena 214,3 MHz. Maximální frekvence může být i vyšší, ale poté se budeme setkávat s neúspěšnými statistickými testy. Jelikož TRNG nevyužívá dodatečné zpracování dat, propustnost jednobitového výstupu se přibližně rovná frekvenci návrhu, tj. $\approx$ 214 Mb/s. Pokud zvolíme vstup o šířce 32 b, propustnost dosahuje $\approx$ 6848 Mb/s. V případě odesílání dat přes UART rozhraní je propustnost omezena jeho rychlostí, maximální reálná rychlost činí 92160 B/s. Průměrná teplota zařízení při generování čísel dosahovala 44 ℃.

7.2 Testování náhodnosti

Testování kvality generátoru náhodných čísel probíhá skrze NIST STS, popsané v kapitole 3.2. Jedná se o jednu z nejběžnějších a nejvíce komplexních statistických testovacích sad. Testovací sada byla provedena na několika vstupních souborech o velikosti 100 MB vytvořených pomocí RNG, přičemž bylo testováno 100 sekvencí, každá o velikosti 1 Mb. Pro potřeby STS byla náhodná čísla převedena do binární podoby. Celkem tři soubory, které obsahující náhodné sekvence, byly podrobeny testům.

Hladina významnosti α byla ponechána na výchozí hodnotě 0,01, protože znamená 99 % interval spolehlivosti. Všechny ostatní parametry testovací aplikace byly také ponechány na výchozích hodnotách. Měření probíhalo stejným způsobem jako při testování volně dostupných TRNG, viz kapitola 5.4.

Závěr testovací sady je shrnut v následující tabulce 7.3. V každém měření je uvedena procentuální úspěšnost jednotlivých testů. V případě výsledku 98 %, uspělo 98 sekvencí a dvě selhaly (při celkovém počtu 100 sekvencí). Počet úspěšných testů musí splňovat minimální hranicí, která je za každým testem napsaná v závorce. Pokud se test skládá z více testů, je uveden rozsah úspěšnosti a p-hodnoty jsou průměrovány. Týká se to zejména testu nepřekrývajících se vzorů a testu náhodných návštěv (variant). Závěry obsahují výsledné p-hodnoty, které informují o tom, jestli

p-hodnoty v rámci testu pochází z rovnoměrného rozložení.

Tab. 7.3: Výsledky jednotlivých testů NIST STS pro tři měření

Měření č. / **Název testu**	**1**		**2**		**3**	
	p	%	p	%	p	%
Frekvenční test (96)	0,8978	100	0,0428	97	0,3345	98
Frekvenční test uvnitř bloku (96)	0,3838	99	0,3191	98	0,7598	99
Test série bitů (96)	0,8343	98	0,7792	100	0,5955	99
Test nejdelší sekvence jedniček v bloku (96)	0,6579	98	0,9463	99	0,2368	99
Test sérií binárních matic (96)	0,7399	97	0,3345	99	0,6787	100
Test diskrétní Fourierovy transformace (96)	0,8677	97	0,7598	99	0,4750	99
Test nepřekrývajících se vzorů (96)	0,4729	96-100	0,4763	96-100	0,4991	95-100
Test překrývajících se vzorů (96)	0,1300	99	0,9943	100	0,9915	99
Mauerův univerzální statistický test (96)	0,6993	99	0,8165	99	0,0220	99
Test lineární složitosti (96)	0,5141	98	0,2622	100	0,1917	100
Test sérií (96)	0,7253	99	0,6546	100	0,8156	100
Přibližný test entropie (96)	0,2370	100	0,7981	98	0,1917	100
Test kumulativní sumy (96)	0,4282	98-99	0,2289	96-97	0,0946	98-99
Test náhodných návštěv (94)	0,4068	56-100	0,3922	94-100	0,3230	98-100
Test náhodných variant návštěv (94)	0,4928	98-100	0,3945	94-100	0,2718	98-100

Pokud se blíže podíváme na výsledky testu nepřekrývajících se vzorů, tak se skládá celkem z 148 podtestů. Každý odpovídá odlišné sekvenci o devíti bitech. V podrobnější zprávě tohoto testu najdeme přesně kolik stejných bloků bylo v sekvenci nalezeno a jestli to splňuje předpoklad náhodné sekvence. V posledním měření jeden z těchto podtestů neprošel o jednu sekvenci, tj. výsledek 95 ze 100. Vícekrát se objevila sekvence, která má sled „000001011“. Pokud se podíváme na p-hodnotu, která dosahuje průměru, můžeme konstatovat, že p-hodnoty nesou rovnoměrné rozložení. Čili můžeme předpokládat, že většina podtestů se nenacházela blízko kritické meze.

Ze tří nezávisle provedených pozorování neprošel u posledního měření zmíněný

test nepřekrývajících se vzorů. Všechny ostatní testy dále prošly. Je důležité poznamenat, že TRNG byl testován a upravován pro konkrétní zařízení. Pro jiná zařízení je vždy nutné ověřit náhodnost a případně upravit parametry, zejména počet RO nebo frekvenci.

7.2.1 Vlastní aplikace

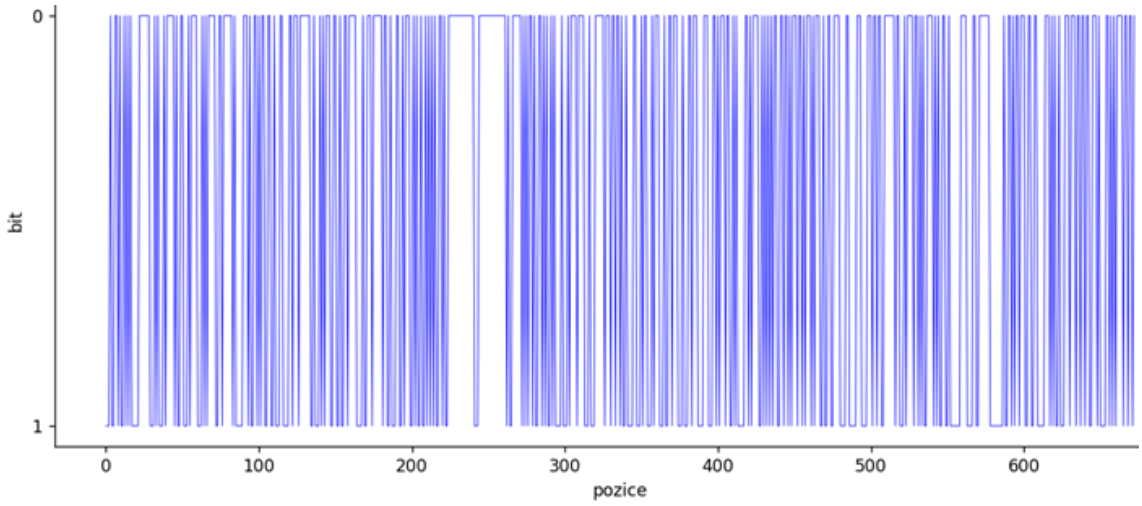
Součástí této práce je návrh a realizace jednoduché testovací implementace pro metody generování náhodných čísel na platformě FPGA. Aplikace na základě statistických výpočtů zhodnotí sekvence bitů, které jsou výstupem generátorů čísel, a označí, jestli se daná sekvence dá považovat za náhodnou, nebo nikoliv.

Pro tvorbu aplikace byl zvolen programovací jazyk Python ve verzi 3.9, k vytvoření aplikace bylo využito populární integrované vývojové prostředí PyCharm. Testovací aplikace byla vytvářena v operačním systému Microsoft Windows 11 Pro.

Jelikož NIST STS je standardizovaný nástroj, je náročné přijít na nové testy, které by odhalovaly a testovaly v sekvenci nové skutečnosti. Proto aplikace byla pojata jako lehčí verze STS. Jednoduchá testovací sada zejména ulehčila testování TRNG při jeho sestavování. Obsahuje základní testy a je možné rychle odhalit špatnou implementaci ihned ze začátků. Sady testů NIST jsou sice spolehlivé, ale poměrně časově náročné, navíc implementovány pro OS Linux. K testům NIST bylo přikročeno až po dobrých výsledcích navržené aplikace.

Program z odpovědi od uživatele otevře soubor se sekvencemi a načte do paměti k posouzení náhodnosti. Následují jednotlivé testy. Závěry jsou učiněny na základě statistických hypotéz, které jsou na určité hladině významnosti. Konkrétně byla zvolena hladina 0,01, stejně jako v případě souboru testů od NIST.

Zároveň aplikace přidává grafy (kde je to vhodné) vytvořené z analýzy dat, viz obrázek 7.1. Kde můžeme vidět, že sekvence obsahuje větší množství stejných bitů ihned za sebou. Dále implementuje výpis výsledků do textového souboru.



Obr. 7.1: Ukázka grafu – Oscilace nul a jedniček.

7.3 Srovnání

Následující tabulka 7.4 srovnává výsledky této práce s ostatními implementacemi. Byly vybrány práce, které pracují na podobném principu. Některé návrhy již byly představeny v analýze generátorů. Je patrné, že mezi pracemi jsou znatelné rozdíly, a to jak z hlediska propustnosti, tak využití zdrojů.

Tab. 7.4: Srovnání návrhů RNG

Práce	Zdroj entropie	Propustnost [Mbps]	Frekvence [MHz]	Využití zdrojů
Tato práce	jitter RO	6848	214,3	455 LUT, 192 FF
Anandak et. al [34]	jitter RO	6	24	528 LUT, 117 FF
Sunar et. al [14]	jitter RO	2,5	40	–
Wold et. al [25]	jitter RO	100	100	83 LUT
Deák et. al [27]	jitter PLL	6,25	100	19 slice, 2 PLL

Implementace [34] využívá u kruhových oscilátorů PDL. Pro lepší náhodnost využívá post-processing, který je na bázi Von Neumannova korektoru. To má za následek nižší propustnost, průměrně ze čtyř bitů vznikne jeden náhodný bit. Co do využití zdrojů, dosahuje podobných hodnot jako navržená implementace.

Propustností nejvíce vyniká tato navržená práce. Je to hlavně z toho, že využívá 32 paralelních drah (výstupů).

Závěr

Cílem bakalářské práce bylo seznámit se s postupy hardwarové implementace na platformě FPGA. Nastudovat problematiku generátorů náhodných čísel a s tím spojené testování náhodnosti. Dále analyzovat možnosti hardwarových generátorů. Na základě ní implementovat vybraný TRNG v jazyce VHDL. Vedlejším cílem rovněž bylo navrhnout a vytvořit testovací aplikaci pro ověřování náhodnosti čísel, která jsou generována na kartě s FPGA obvodem.

Pro každého, který cílí na konfiguraci programovatelných hradlových polí, je důležité nastudovat a chápat tyto obvody. Práce proto obsahuje v první kapitole rozsáhlejší popis architektury a také postupy konfigurace FPGA obvodů. Důležitým aspektem je rovněž pochopit základy jazyků pro popis hardwaru.

Nedílnou součásti práce je také teoretické pojednání o generátorech náhodných čísel, které jsou základem dnešní kryptografie. Díky náhodným a dostatečně dlouhým sekvencím je možné šifrovat, chránit naše data. Aby to bylo uskutečnitelné, je důležité mít skutečně náhodné sekvence. S ověřováním kvality generátorů si lze pomoci různými testy. Ve třetí kapitole je tak popsána baterie testů Národního institutu standardů a technologií.

V poslední teoretické kapitole byly představeny a srovnány postupy generování náhodných čísel na FPGA. Zejména se jednalo o návrhy, které implementovaly metastabilitu, chaos, kruhové oscilátory a smyčky fázového závěsu.

Pátá kapitola, praktická část, zahrnovala seznámení se se softwarovým balíkem Vivado a Vitis od Xilinx, který slouží pro analýzu, syntézu a následnou implementaci jazyka pro popis hardwaru. V případě Vitis se jedná o jazyk C. Pro hodnocení byly vybraný dostupné implementace hardwarových generátorů. Experimentům posloužila vývojová deska Digilent Zybo Z7-20 obsahující FPGA Xilinx řady 7. Bylo provedeno jejich srovnání a testování náhodnosti.

Hlavním bodem bakalářské práce se stala šestá kapitola, ve které probíhal popis vlastní implementace TRNG. Návrh je založen na odlišných prstencích, které tvoří hradla NOT, nebo XOR. Jednotlivé komponenty jsou popsány a schématicky zobrazeny. Dále je popsán blokový návrh a propojení s procesním systémem. Výsledkem je odesílání náhodných dat z TRNG přes UART rozhraní do počítače.

V poslední kapitole dochází ke konfiguraci FPGA obvodu a ověření navržené implementace. Prezentovány jsou výsledky konfigurace, která slouží ke generování náhodného čísla o délce 32 bitů. Návrh dosahuje nízkých hardwarových nároků vzhledem k celkovému počtu dostupných zdrojů. Komponenta generátoru konkrétně využívá 455 LUT a 192 FF. Maximální teoretická propustnost dosahuje 6848 Mb/s. TRNG poté prochází statistickými testy NIST STS. Dále je popsána vlastní testovací aplikace. Závěrem kapitoly je krátké srovnání jednotlivých prací.

Literatura

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

[2] *What is an FPGA? Field Programmable Gate Array.* [online]. Advanced Micro Devices, 2022 [cit. 2022-11-06]. Dostupné z: <https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html>.

[3] ZDRÁLEK, Jaroslav. *Programovatelné logické prvky, studijní opora* [online]. Ostrava: VŠB – Technická univerzita Ostrava, 2007 [cit. 2022-10-23]. ISBN 978-80-248-1502-2. Dostupné z: <http://www.elearn.vsb.cz/archivcd/FEI/PLP/zdralek_PLP.pdf>

[4] KOLÁŘ, Milan. *Architektura a návrh FPGA obvodů* [online]. Liberec, 2010 [cit. 2022-10-23]. Habilitační práce. Technická univerzita v Liberci. Dostupné z: <https://dspace.tul.cz/handle/15240/39016>

[5] *Intel Stratix 10 GX 10M FPGA* [online]. Santa Clara: Intel Corporation, 2022 [cit. 2022-11-18]. Dostupné z: <https://www.intel.com/content/www/us/en/products/sku/210290/intel-stratix-10-gx-10m-fpga/specifications.html>

[6] *The Ultimate Guide to FPGA Architecture* [online]. HardwareBee, 2022 [cit. 2022-11-06]. Dostupné z: <https://hardwarebee.com/the-ultimate-guide-to-fpga-architecture>

[7] KOLOUCH, Jaromír. *Programovatelné logické obvody* [online]. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, 2006 [cit. 2022-11-06]. Dostupné z: <https://www.vut.cz/www_base/priloha_fs.php?dpid=25921&skupina=dokument_priloha>

[8] KUBÍČEK, M. *Úvod do problematiky obvodů FPGA pro integrovanou výuku VUT a VŠB-TUO* [online]. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2014 [cit. 2022-11-06]. ISBN 978-80-214-5069-1. Dostupné z: <https://www.vut.cz/www_base/priloha_fs.php?dpid=87427&skupina=dokument_priloha>

[9] *The MCU guy's introduction to FPGAs: Configuration Techniques & Technologies* [online]. Cambridge: Embedded by AspenCore, 2015 [cit. 2022-11-17]. Dostupné z: <https://www.embedded.com/the-mcu-guys-introduction-to-fpgas-configuration-techniques-technologies>

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

[11] HAAHR, Mads. *Introduction to Randomness and Random Numbers* [online]. Dublin: Random.org, 2022 [cit. 2022-11-23]. Dostupné z: <https://www.random.org/randomness>

[12] BURDA, Karel. *Aplikovaná kryptografie.* Brno: VUTIUM, 2013. ISBN 978-80-214-4612-0.

[13] MENEZES, Alfred, Paul C VAN OORSCHOT a Scott A VANSTONE. *Handbook of applied cryptography.* Boca Raton: CRC Press, 1997. Discrete mathematics and its applications. ISBN 0-8493-8523-7.

[14] B. Sunar, W. J. Martin and D. R. Stinson. *A Provably Secure True Random Number Generator with Built-In Tolerance to Active Attacks.* IEEE Transactions on Computers, 2007, s. 109–119.

[15] Govindan, V., Chakraborty, R., Santikellur, P., Chaudhary, A. *A Hardware Trojan Attack on FPGA-Based Cryptographic Key Generation: Impact and Detection.* Journal of Hardware and Systems Security, 2018.

[16] RUKHIN, Andrew, Juan SOTO a James NECHVATAL et al. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications* [online]. Gaithersburg: National Institute of Standards and Technology, 2010 [cit. 2022-11-18]. Dostupné z: <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf>

[17] BROWN, Robert G. *Robert G. Brown's General Tools Page* [online]. Durham: Duke University Physics Department, 2022 [cit. 2022-11-18]. Dostupné z: <https://webhome.phy.duke.edu/~rgb/General/dieharder.php>

[18] Barker EB, Kelsey JM, McKay KA, Roginsky AL, Sönmez Turan M. *Recommendation for Random Bit Generator (RBG) Constructions* [online]. Gaithersburg: National Institute of Standards and Technology, NIST Special Publication (SP) 800-90C 3pd, 2022 [cit. 2023-03-25]. Dostupné z: <https://doi.org/10.6028/NIST.SP.800-90c.3pd>

[19] V. Fischer, F. Bernard, N. Bochard and M. Varchola. *Enhancing security of ring oscillator-based trng implemented in FPGA.* Heidelberg: International Conference on Field Programmable Logic and Applications, 2008, s. 245–250.

[20] Sivaraman, R., Rajagopalan, S., Sridevi, A. et al. *Metastability-Induced TRNG Architecture na FPGA.* Iran J Sci Technol Trans Electr Eng 44, 2020, s. 47–57.

[21] Lorenz, E. N. *Deterministic Nonperiodic Flow*. Journal of the Atmospheric Sciences, 1963, s. 130-–141

[22] Rössler, O.E. *Continuous choas-four prototype equations*. Annals of the New York Academy of Sciences, 1979, s. 376–392.

[23] İ. Koyuncu, A.T. Özcerit. *The design and realization of a new high speed FPGA-based chaotic true random number generator*. Computers and Electrical Engineering 58, 2017, s. 203-–214.

[24] B. Valtchanov, V. Fischer, A. Aubert and F. Bernard. *Characterization of randomness sources in ring oscillator-based true random number generators in FPGAs*. Vienna: 13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems. 2010, s. 48–53.

[25] K. Wold and C. H. Tan. *Analysis and Enhancement of Random Number Generator in FPGA Based on Oscillator Rings*. Cancun: International Conference on Reconfigurable Computing and FPGAs, 2008, s. 385–390.

[26] M. Drutarovsky, M. Simka, and V. Fischer. *A Simple PLL-based True Random Number Generator For Embedded Digital Systems*. Computing and Informatics, 2004, s. 501–516.

[27] N. Deák, T. Györfi, K. Márton, L. Vacariu and O. Cret. *Highly Efficient True Random Number Generator in FPGA Devices Using Phase-Locked Loops*. Bucharest: 20th International Conference on Control Systems and Computer Science, 2015, s. 453–458.

[28] K. Wold and S. Petrović *Security properties of oscillator rings in true random number generators*. Tallinn: IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits and Systems, 2012, s. 145–150.

[29] Nolting, S. *The neoTRNG V2 - A Tiny and Platform-Independent True Random Number Generator for any FPGA 2.0.0* [online]. 2022 [cit. 2023-03-26]. Dostupné z: <`https://github.com/stnolting/neoTRNG`>

[30] Blase, B. *A True Random Number Generator in VHDL* [online]. 2020 [cit. 2023-03-26]. Dostupné z: <`https://github.com/benbr8/trng_fpga`>

[31] Sharma, A. *FPGA-based-True-Random-Number-Generator* [online]. 2021 [cit. 2023-03-26]. Dostupné z: <`https://github.com/Adi-SRAM25/FPGA-based-True-Random-Number-Generator`>

[32] *Zybo Z7 Reference Manual* [online]. Pullman: Digilent, 2023 [cit. 2023-04-08]. Dostupné z: <`https://digilent.com/reference/programmable-logic/zybo-z7/reference-manual`>

[33] T. Ni et al. *MRCO: A Multi-ring Convergence Oscillator-based High-Efficiency True Random Number Generator*. Singapore: Asian Hardware Oriented Security and Trust Symposium (AsianHOST), 2022.

[34] N. Nalla Anandakumar, S. K. Sanadhya and M. S. Hashmi. *FPGA-Based True Random Number Generation Using Programmable Delays in Oscillator-Rings*. IEEE Transactions on Circuits and Systems II: Express Briefs, 2020, s. 570–574.

Seznam symbolů a zkratek

AES	Advanced Encryption Standard – standardizovaný algoritmus k šifrování
AXI	Advanced eXtensible Interface – pokročilé rozšiřitelné rozhraní
CLB	Configurable Logic Block – konfigurovatelný logický blok
CMOS	Complementary Metal Oxide Semiconductor – druh výrobní technologie
CPU	Central Processor Unit – centrální procesorová jednotka
DES	Data Encryption Standart – šifrovací standart
DSA	Digital Signature Algorithm – algoritmus digitálního podpisu
EBR	Embedded Block RAM – vložená bloková paměť RAM
EEPROM	Electrically Erasable PROM – elektricky mazatelná paměť PROM
FPGA	Field Programmable Gate Array – programovatelné hradlové pole
GPIO	General Purpose Input/Output – Univerzální vstup/výstup
GUI	Graphic User Interface – grafické uživatelské rozhraní
HDMI	High-Definition Multimedia Interface – multimediální rozhraní s vysokým rozlišením
HDL	Hardware Description Language – Hardwarový deskriptivní jazyk
IEEE	Institute of Electrical and Electronics Engineers – Institut pro elektrotechnické a elektronické inženýrství
I/O	Input/Output – vstup/výstup
ISP	In System Programming – programovatelný v systému
IP block	Intellectual Property – Intellectual property blok
LSFR	Linear-Feedback Shift Register – posuvný registr s lineární zpětnou vazbou
LUT	Look Up Table – vyhledávácí tabulka

NIST	National Institute of Standards and Technology – Národní institut standardů a technologie
NIST STS	The NIST Statistical Test Suite – soubor statistických testů vytvořených NIST
PDL	Programmable Delay Line – programovatelná zpožďovací linka
PL	Programmable Logic – programovatelná logická
PLD	Programmable Logic Device – programovatelná logická součástka
PLL	Phase-Locked Loop – smyčka fázového závěsu
PROM	Programmable Read Only Memory – jednorázově programovatelná paměť
PS	Processing System – procesní systém
RAM	Random-Access Mmemory – paměť s libovolným přístupem
RNG	Random Number Generator – generátor náhodných čísel
RO	Ring Oscillator – prstencový oscilátor
RSA	Rivest–Shamir–Adleman – kryptografický systém
RTL	Register Transfer Level – úroveň meziregistrových přenosů
SRAM	Static RAM – statická paměť RAM
UART	Universal Asynchronous Receiver-Transmitter – Univerzální asynchronní přijímač–vysílač
USB	Universal Serial Bus – Univerzální sériová sběrnice
VCO	Voltage-Controlled Oscillator – napěťově řízený oscilátor
VHDL	Very High Speed Integrated Circuit HDL – HDL pro velmi rychlé integrované obvody

A Obsah elektronické přílohy

- `/` ..... kořenový adresář
  - `BC_TRNG.srcs` ..... zdrojové soubory Vivado (PL)
    - `sources` ..... soubory .vhd
      - `ro.vhd`
      - `ro_s.vhd`
      - `trng.vhd`
      - `trng_top.vhd`
    - `constrs` ..... soubory omezení
    - `synth` ..... soubory syntézy
    - `design` ..... soubory pro blokový návrh
  - `PS_src` ..... zdrojové soubory Vitis (PS)
    - `TRNG_design.xsa` ..... exportovaný blokový návrh
    - `TRNG_design.bit` ..... soubor pro konfiguraci zařízení
  - `BC_TRNG.tcl` ..... skript pro vytvoření návrhu
  - `test` ..... zdrojové soubory testovací aplikace
  - `Generování náhodných čísel na platformě FPGA.pdf` ..... text práce