VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

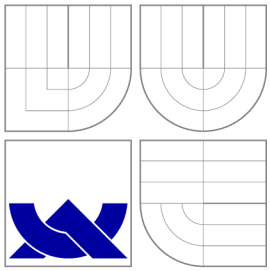
IMPLEMENTACE STATISTICKÝCH FUNKCÍ POMOCÍ HLS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

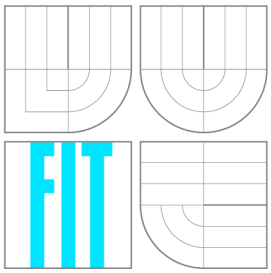
AUTOR PRÁCE
AUTHOR

PETER ŠINAĽ

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

IMPLEMENTACE STATISTICKÝCH FUNKCÍ POMOCÍ HLS

IMPLEMENTATION OF STATISTICAL FUNCTIONS USING HLS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETER ŠINAĽ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MILAN DVOŘÁK

BRNO 2014

Abstrakt

Cílem této bakalářské práce bylo navrhnout a implementovat vybrané statistické funkce používané v oblasti technické analýzy. Zaměřil jsem se na klouzavé průměry, Black- Schles model pro výpočet cen opcí a indikátor Delta. Tyto funkce jsou pomocí HLS transformované do popisu vhodného pro programovatelná hradlová pole FPGA. Během procesu transformace je kladen důraz na nízkou latenci a spotřebu zdrojů. Vytvořená řešení demonstrují potenciál HLS. Ukazují složitost technické analýzy a nároky na hardware. Získané výsledky vykazují v simulacích vysokou přesnost. Odchylka od referenčních hodnot je v průměru $6,615 \times 10^{-3}$. Výsledky též naznačují, že snížením latence nemusí nutně docházet ke zvýšení spotřeby zdrojů na čipu.

Abstract

The aim of this thesis was to design and implement selected statistical functions used in technical analysis. I focused on moving averages, Black – Schles model for calculating option prices and Indicator Delta. These functions are through HLS transformed into an appropriate description for programmable FPGA. During the transformation process, emphasis is on low latency and resource consumption. Created solutions demonstrate the potential of HLS. They show complexity of the technical analysis and hardware requirements. Achieved results show high accuracy in the simulations. Deviation from the reference value is approximately $6,615 \times 10^{-3}$. The results also indicate thet that reducing latency does not necessarily cause an increase in the consumption of resources on the chip.

Klíčová slova

HLS, technický indikátor, statistické funkce, klouzavý průměr, Black – Scholes model, Delta indikátor, vysoce obrátkové obchodování, logaritmus, exponenciální funkce, distribuční funkce normálního rozdělení, FPGA

Keywords

HLS, technical indicator, statistical functions, moving average, Black – Scholes model, Delta indicator, high frequency trading, logarithm, exponential function, standard normal distribution, FPGA

Citace

Peter Šinaľ: Implementace statistických funkcí pomocí HLS, bakalářská práce, Brno, FIT VUT v Brně, 2014

Implementace statistických funkcí pomocí HLS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Milana Dvořáka. Uvedl jsem všechy literární prameny a publikace, ze kterých jsem čerpal.

.....
Peter Šinaľ
20. mája 2014

Poděkování

Na tomto místě bych rád poděkoval vedoucímu práce Ing. Milanovi Dvořákovi za odborné vedení a veškerý čas věnovaný této práci. Také chci poděkovat své rodine za neustálou podporu.

Obsah

Kapitola 1

Úvod

Sučasný rozvoj v oblasti informačných technológií poskytuje široké možnosti uplatnenia jeho výsledkov vo všetkých sférach ľudskej činnosti. Výrazne sa podieľa na efektívnom spracovávaní informácií a riešení každodenných problémov. Jednou z oblastí využitia je aj obchodovanie na burzách používaním stratégie vysoko obratových investícií *high frequency trading* – HFT. Táto stratégia kladie vysoké nároky na výkon počítačov. Vyhodnotenie potencionálnej investície musí byť uskutočnené v čo najkratšom čase, aby sme maximalizovali zisky. Za týmto účelom sa používajú komplexné algoritmy založené na štatistických funkciách a metódach technickej analýzy. Tu sa nám ponúka možnosť presunúť výpočty týchto algoritmov alebo ich častí do aplikačne špecifických obvodov ASIC, alebo programovateľných hradlových polí FPGA. Každá z týchto technológií má svoje výhody a nevýhody. Majú však potenciál poskytnúť vyššiu výkonnosť než univerzálne procesory. Akceleráciou algoritmov v hardware skrátime čas potrebný na vyhodnotenie. Trvanie obchodov pri HFT stratégií sa počíta v mikrosekundách, rýchlosť spracovania tu hraje veľmi dôležitú úlohu.

Táto práca sa zaoberá implementáciou vybraných technických indikátorov používaných v technickej analýze. Tieto funkcie sú prostredníctvom HLS nástroja transformované do RTL popisu. Dokument je logicky členený do niekoľkých častí. V úvodnej kapitole 2 sa stručne oboznámime s problematikou transformácie obvodov popísaných jazykmi HLL. Predstavíme si konkrétny HLS nástroj a ako sa s touto problematikou vysporiadal. Kapitola 3 popisuje vybrané technické indikátory z matematického hľadiska a predstavuje nevyhnutný základ pre ich pochopenie. V kapitola 4 je popísaný spôsob implementácie a dosiahnuté výsledky. Spôsob overovania správnosti jednotlivých častí je popísaný v kapitole 5. V záverečnej kapitole 6 sa venujem dosiahnutým výsledkom a výhľade ich uplatnenia v najbližšom období. Definície použitých skratiek nájdeme v prílohe B.

Kapitola 2

Syntéza obvodov na vyššej úrovni – HLS

Pre popis integrovaných obvodov sa klasicky používajú HDL (*hardware description language*) jazyky. Tieto jazyky sú veľmi efektívne, no pri zvyšovaní komplexnosti dnešných obvodov je ťažké zachovať ich rovnakú produktivitu. Nízka produktivita znamená, že čas potrebný na uvedenie produktu na trh (*time to market*) sa môže značne predlžiť. Na riešenie tohto nedostatku sa nám ponuká možnosť zvýšenia abstrakciu popisu. Prax dokazuje, že týmto riešením sa zvyšuje aj samotná produktivita inžinierov. To znamená nevyhnutný prechod z jazykov HDL na HLL. Zo zvýšením úrovne abstrakcie potrebujeme aj novú technológiu na jej transformáciu. Táto technológia sa nazýva HLS (z anglického *high level synthesis*) a niekedy sa používa aj označenie ako behaviorálna syntéza alebo syntéza z algoritmu. Umožňuje obvod popísaný HLL transformovať až na úroveň RTL popisu vhodného pre ASIC/FPGA. Použitím technológie HLS zamedzíme množstvu chýb, ktoré by inak používatelia zaniesli do dizajnu obvodu. Čas potrebný na vývoj a verifikáciu obvodov sa pri použití HLS technológie výrazne skracuje.

Aj napriek počiatočným zlyhaniam v prvých verziách komerčných HLS systémov, vývoj pokračoval, pretože dopyt po kvalitných nástrojoch pretrvával a to hneď z niekoľkých dôvodov. Jedným z dôvodov bolo, že všetky vstavané procesory sú typu SoC a pri vytváraní moderných systémov vstupuje do tohto procesu stále viac nových elementov. Súčasné HLS nástroje s vysokou mierou automatizácie umožňujú popísať funkcionalitu v niektorom vyššom programovacom jazyku tak pre aplikačne špecifický hardware ako aj pre vstavané systémy. Za pomerne krátky čas môžeme experimentovať s rôznymi parametrami hardware/software a preskúmať veľké množstvo dizajnov obvodov.

Vysoká kapacita dnešných kremíkových čipov si vyžaduje vyšší stupeň abstrakcie najmä prípade, že chceme túto kapacitu naplno využiť. Na ilustráciu: v RTL kóde potrebujeme pre popis 1 milióna logických brán cca. 300 tisíc riadkov kódu a v jazyku HLL na popis rovnakého počtu brán potrebujeme už len 30–40 tisíc riadkov kódu [8].

Vysoký dopyt po tomto druhu syntézy podnietil a umožnil vznik viacerých nástrojov technológie HLS [17], všetky však museli čeliť rovnakým problémom pri vývoji.

2.1 Vývoj HLS nástrojov

Aby sme lepšie pochopili možnosti HLS syntézy, popíšeme si ako táto syntéza vznikala a s akými problémami ktorými vývojový inžinieri zaoberali. Možnosť ponechať všetky imple-

mentačné detaily vrátane časovania operácií, dátových prenosov a organizácie pamäte na sadu nástrojov zaujíma vedcov už od 70. rokov. Aby sa tak stalo vývojári museli vyriešiť niekoľko problémov bez ktorých by HLS nebolo možné:

- **Paralelizmus** a jeho extrahovanie z danej špecifikácie.
- **Časovanie operácií** nám poskytne predvídateľné simulácie.
- **Komponenty** nám poskytnú možnosť budovať veľké systémy.

Paralelizmus

Literatúra označuje tento problém za najviac predpojatý zo strany výskumníkov[9]. Dôvodom je, že jednou z prvých úloh HLS nástroja je extrahovať paralelizmus ukrytý v algoritme. Najčastejším spôsobom bolo extrahovanie *data – flow* grafu z popisu založenom na analýze závislostí jednotlivých operácií. Tieto data-flow grafy však majú tendenciu byť disjunktné, a teda aj príliš náročné pre algoritmus syntézy. Často sa kombinovali s uzlami a hranami na reprezentovanie riadenia toku. Označovali sa ako CDFG (*Control/Data Flow Graphs*). Operácie (násobenie, delenie) z popisu chovania sú reprezentované ako uzly a hodnoty (vstupy výrazov, premenne) sú vyjadrené hranami. Takýto model nezachytil pamäťové bloky, s ktorými sa zaobchádzalo ako z funkcionálny alebo štrukturovaným blokom. V takomto modeli však bola problémom hierarchia.

Inováciou v tejto oblasti bol pokus o presun nedeterminizmu (v operáciach, časovaní) do hierarchického grafového modelu v SIF (*Standford Intermediate Format*). V SIF grafe sú telá cyklov a podmienok reprezentované ako samostatné podgrafy. Tieto SIF grafy sa navyše stali acyklické a orientované, čo umožnilo použitie efektívnych algoritmov pre plánovanie a alokáciu zdrojov. Ich využitie vidíme v nástroji od spoločnosti Olympus Synthesis System [7].

V spoločnosti Olympus Synthesis System taktiež vyvinuli verziu jazyka C, nazývanú *HardwareC*, ktorá umožňovala špecifikovať paralelnosť operácií. Dve operácie by mohli byť naplánované paralelne, sekvenčne alebo dáta – paralelným štýlom [10]. Ďalšou lekciou pri modelovaní dizajnov bola otázka hardwardových konceptov v HLL. Jednou z možností bolo jednoducho pridať tieto koncepty do špecifikácie jazyka, ako je to možné vidieť v *HardwareC*. Ďalšou možnosťou je preťaženie existujúcich konštrukcií v HLL. Poslednou možnosťou je použiť knižnice, preťaženie operátorov a polymorfné typy. Tento prístup sa uplatnil v jazyku SystemC.

Časovanie

Na správanie obvodov z hľadiska časovania sa zameriava pozornosť v ranných 90. rokoch. Do popredia sa v tom čase dostávajú vstavané systémy a na HLS obvody sa pozerá ako na *system desing* problém. Popis vstupov vyzerá ako popis komponenty, ktorá je v neustálom kontakte s okolím. Preto bolo možné oddeliť funkčné chovanie a požiadavky na časovanie. Jednou z možností bolo generalizovať graf úloh tak, že úlohy reprezentovali uzly a komunikáciu hrany. Uzol mohol byť rozložený podľa typu úlohy. AND úloha reprezentuje akciu, ktorá mohla byť uskutočnená až po vykonaní predchodcov a OR úlohy mohli byť uskutočnené kedykoľvek po spustení predchodcu. Táto štruktúra umožnila generovať diskrétny model udalostí priamo z tohto grafu. Zameranie na časovú analýzu dalo možnosť vzniku *reactive behaviors* systémov. Scenic (neskôr SystemC) prichádza s jazykom na popis týchto systémov. Reaktivita je implementovaná cez *waiting* a *watching*. Kým *waiting* pozastavý

proces pokiaľ nenastane istá udalosť (synchronizáciu s hodinami), tak *watching* je asynchrónna podmienka (reset).[16]

Komponenty a transakcie

Komponenty predstavujú systémové bloky, ktoré môžu byť použité bez ďalších výrazných zmien v iných dizajnoch. Rozhranie týchto blokov zapuzdruje ich komplexnosť. Transakčné spracovanie sa snaží pozdvihnúť úroveň abstrakcie tak po funkčnej stránke komponenty, ako aj samotného rozhrania. Protokoly komunikácie sú tiež dôležité pre abstrakciu rozhrania. V počiatkoch HLS sa predpokladali implicitné protokoly a časovania uvedené priamo v popise jazyka. Táto oblasť je stále predmetom výskumu, pretože tvorba rozsiahlych systémov sa nezaobíde bez abstrakcie štruktúr a časovania.

Snaha o vytvorenie HLS nástroja prispela k vzniku dalších projektov presahujúcich originálnu myšlienku. Za zmienku stoja synchrónne jazyky Esterel a Luster[19], ktoré umožňujú formálne verifikovať syntézu hardwardových a softvérových systémov. SystemC sa začal používať ako štandard pre transakčne založené systémy a umožňuje polo-behaviorálnu syntézu. Daším príkladom je BlueSpec[1], nástroj na hardwardovú kompiláciu. Ten dokáže na základe jazyka popísaného atomickými pravidlami efektívne popísať cyklické chovanie a vytvoriť obvod kvalitou porovnateľný s obvodom vytvoreným človekom [9].

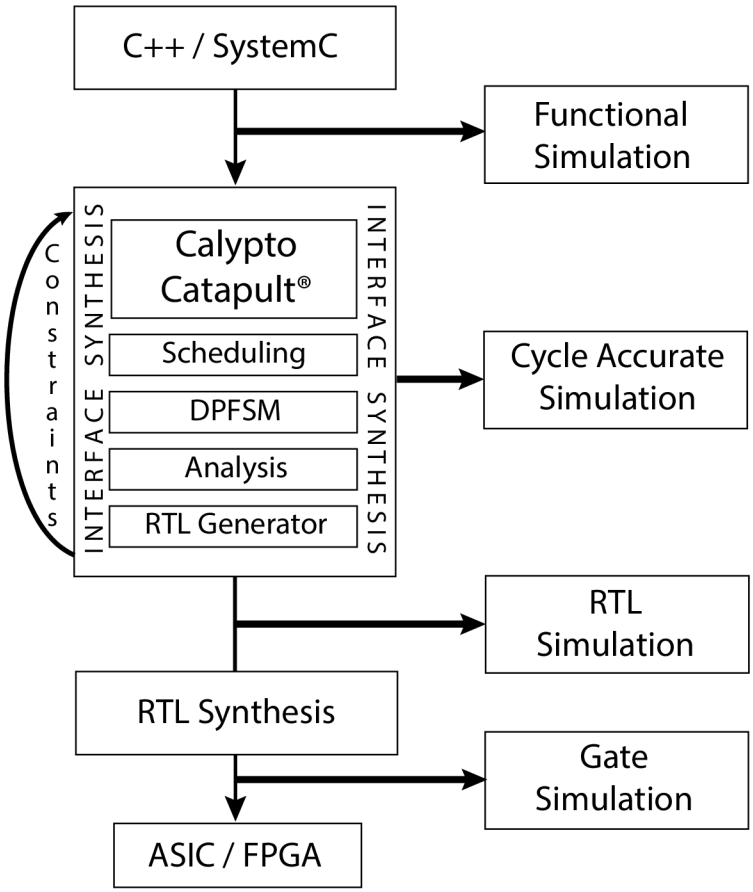
2.2 Vývojové prostredie Catapult C®

Tento nástroj definuje nový prístup k HLS a to tým, že na popis chovania obvodu sa používa ANSI C++. Nevýhody pri štrukturovaných jazykoch ako VHDL, Verilog dokonca aj jazyka SystemC, ktorý sa používa v polo – behaviorálnej syntéze je niekoľko. Sú cudzie pre množstvo vývojárov, ich abstrakcia je už nepostačujúca a ukázali sa ako náročne na pochopenie. Množstvo logických členov, ktoré dokážu svojim popisom obsiahnuť naráža na limity. ANSI C++ na druhú stranu predstavuje najrozšírenejší jazyk na svete. V porovnaní s prvou generáciou nástroja prichádza Catapult® s riešením, kde časovanie I/O, protokoly a paralelizmus sú oddelené od funkcionality zdrojových kódov. S takou funkcionalitou môže byť časovanie dizajnu vyvinuté a verifikované aj oddelene.

2.2.1 Pracovný cyklus – *workflow*

Typický pracovný cyklus HLS nástroja je znázornený na obrázku 2.1. Transformácia C++ kódu je automatizovaná, pričom vývojár usmerňuje syntézu, aby splnila požiadavky na výkon/spotrebu energie/spotrebu zdrojov. Výhodou je, že v pomerne krátkom čase je možné preskúmať viacero architektúr a nájsť najvhodnejšie riešenie. Celý proces je dekomponovaný do niekoľkých krokov.

[1] Bluespec Technologies and Solutions. http://www.bluespec.com



Obr. 2.1: Catapult workflow [2]

Písanie kódu

Na popis želaného chovania slúži ANSI C++. Popis je čiste algoritmický bez informácií o cieľovej technológii alebo časovania. Syntax a sémantika C++ je zachovaná. Podporovaná je široká podmnožina jazyka C++. Všetky známe konštrukcie (podmienky, cykly, volanie funkcií, statické premenné) sú však vývojárom k dispozícii. Jediným obmedzením je, že kód musí byť staticky determinovaný, jeho charakteristiky je nutné vedieť v dobe prekladu. Z čoho vyplýva, že dynamická alokácia pamäte (malloc, free, new, delete) nie sú podporované [6].

Obmedzenia syntézy

V tomto kroku je nutné špecifikovať cieľovú technológiu a frekvenciu hodín. To sú základné informácie nutné k úspešnému vybudovaniu dizajnu. Ďalej je možné nastaviť globálne ciele syntézy, napríklad *resety*, *clock enable* chovanie, *handshake*. Ďalšie obmedzenia sú nastavované individuálne na konkrétne I/O, cyklus, pamäť, zdroje dizajnu. Všetky nastavenia je možné uskutočniť cez grafické rozhranie alebo prostredníctvom TCL skriptu.

Všetky cykly v programe môžeme kontrolovať rozbalením (*loop unrolling*), spájaním (*loop merging*) a zreťazením (*loop pipeline*). Rozbaľovanie cyklov umožňuje telo cyklu spra-

[2]Catapult: Product Family Overview http://calypto.com/en/products/catapult/overview/

covávať paralelne. Cyklus je možné rozbaliť čiastočne a pokiaľ je známy počet opakovaní, tak aj celý. Rozbalením cyklov znížime latenciu dizajnu avšak za cenu vyššej spotreby zdrojov. Spájanie aplikujeme na sekvenčné cykly a vytvoríme z nich jeden cyklus so zachovaním pôvodnej funkcionality. Tento prístup zníži réžiu potrebnú pre riadenie cyklov. Zreťazenie ponúka možnosť ako zvýšiť priepustnosť cyklov a zlepšiť ich celkovú latenciu. Je to dosiahnuté tým, že ďalšia iterácia cyklu je zahájená ešte pred dokončením tej predchádzajúcej. Môžeme teda pomerne jednoducho kontrolovať paralelizmus a nájsť rozumný kompromis medzi latenciou, priepustnosťou a spotrebou zdrojov.

Plánovanie, analýza

Po nastavení všetkých obmedzení nasleduje plánovanie. Plánovanie je proces budovania a optimalizácie dizajnu podľa zadaných obmedzení, cieľovej technológie a frekvencie hodín. Preto pre dva rôzne technológie bude výsledok pravdepodobne iný, tak aby lepšie reflektoval možnosti cieľovej technológie. Výsledkom je Ganttov graf s prehľadom operácií. Ganttov graf poskytuje pohľad do vnútra cyklov, rôznych algoritmických závislostí. Vývojári môžu plne pochopiť, ako rôzne nastavenie obmedzení, ovplyvní výsledný dizajn. Ľahšie môžeme nájsť úzke miesto dizajnu, ktoré je možné podľa potreby optimalizovať.

Generovanie RTL a verifikácia

Po analýze a nastavení správnych obmedzení vygeneruje Catapult RTL popis vhodný pre FPGA alebo ASIC. Catapult vygeneruje aj VHDL, Verilog a SystemC netlisty. Ďalej je vygenerovaných niekoľko záverečných správ *reportov* poskytujúcich detailné informácie o charakteristike dizajnu. Vývojárovi je napokon poskytnuté aj verifikačné prostredie, ktoré automatizuje proces porovnania medzi HDL netlist a originálnym C/C++ kódom. [11]

2.2.2 Práca s datovými typmi Algorithmic C

Podpora čísel s plávajúcou rádovou čiarkou sa líši podľa použitého nástroja, niektoré HLS nástroje podporujú tieto typy čísel tak ako to definuje štandard IEEE – 754. Vo všeobecnosti je takáto reprezentácia je náročnejšia na zdroje, zvyšuje latenciu. Logika na implementáciu aritmetických operácií je komplexnejšia [14]. HLS nástroj používaný v tejto práci tieto dátové typy (double, float) podporuje ale tak, že konvertuje do formátu *Algorithmic C* s pevnou rádovou čiarkou podľa *pragmy* na začiatku kódu. Napriek ich podpore ich používanie Catapult nedoporučuje. Prináša vlastné riešenie v podobe „Algorithmic C“ dátových typov, ktoré sú bitovo presné, poskytujú jednoduché použitie a nízku réžiu. Užívateľ ma k dispozícií celočíselné a desatine čísla s pevnou rádovou čiarkou, so znamienkom alebo bez.

Celočíselný typ sa zadefinuje ako `ac_int<W,S>` kde `W` označuje bitovú šírku a `S` znamienko. Desatinné číslo vyjadríme ako `ac_fixed<W,I,S,Q,O>` kde `W` označuje celkovú šírku, `I` označuje šírku celočíselnej časti, `Q` označuje režim kvantovania [3] a `O` mód pretečenia [4]. Okrem Algorithmic C sú k dispozícií aj typy známe z C++ (int, short, long, double) a SystemC (`sc_int`, `sc_fixed`, `sc_bigint`)[4].

[3] jedno z nasledujúcich : AC_TRN, AC_TRN_ZERO, AC_RND, AC_RND_ZERO, AC_RND_INF, AC_RND_MIN_INF, AC_RND_CONV

[4] jedno z nasledujúcich: AC_WARP, AC_SAT, AC_SAT_ZERO, AC_SAT_SYM

Kapitola 3

Technická analýza a štatistické funkcie

Technická analýza je jedným z nástrojov na určovanie budúceho vývoja akciových kurzov. V tejto práci budeme hovoriť len o technickej analýze založenej na technických indikátoroch. Cieľom technickej analýzy je na základe dát z minulosti odhadnúť vývoj cenového trendu resp. jeho ukončenie.

Technická analýza vychádza z nasledujúcich predpokladov:

- Kurzy akcií sa nepohybujú náhodne, ale v určitých trendoch. Pri rozpoznaní správneho trendu možno generovať zisk.
- Všetky udalosti vplývajúce na trh sú už premietnuté v cene akcií, ktorá je predmetom technickej analýzy.
- Technická analýza monitoruje pohyb na trhu, a nezaoberá sa faktormi ktoré tento pohyb spôsobujú. Pri prevahe dopytu je očakavaný rast cien, naopak pri prevahe ponuky sa očakáva pokles cien.
- História pohybov ma tendenciu opakovať sa.

Na závery technickej analýzy nie je možné hľadieť s absolútnou istotou, určujú iba to čo je pravdepodobnejšie[20].

3.1 Kĺzavé priemery

Kĺzavé priemery patria medzi významné a v investičnej praxi medzi najčastejšie používané technické indikátory. Patria do skupiny oneskorených ukazovateľov technickej analýzy. Používajú sa na odhadovanie trendu kurzov časovej rady, a to buď samostatne, alebo ako súčasť iných technických indikátorov (napr. MACD). Kĺzavé priemery sledujú vždy istý časový úsek, napríklad históriu cien za posledných 50 dní. Sledujú trendy a podieľajú sa na generovaní obchodných signálov. Bolo preukázané, že aj napriek ich jednoduchosti sú pravidlá a obchodné stratégie na nich založené profitové [18].

3.1.1 Jednoduchý kĺzavý priemer – SMA

Jednoduchý kĺzavý priemer (*Simple Moving Average*) sa počíta ako nevážený priemer N predchádzajúcich časových úsekov.

$$SMA = \frac{p_m + p_{m-1} + \cdots + p_{m-(N-1)}}{N} \tag{3.1}$$

Pre počítanie len nasledujúcej hodnoty priemeru je možné použiť vzorec.

$$SMA_{today} = SMA_{yesterday} - \frac{p_{m-N}}{N} + \frac{p_m}{N} \tag{3.2}$$

3.1.2 Exponenciálny kĺzavý priemer – EMA

Exponenciálny kĺzavý priemer (*exponential moving average*) môžeme definovať ako filter s nekonečnou impulznou reakciou. Váhy aplikované na dáta nemajú lineárny, ale exponenciálny charakter. Najväčšiu váhu majú teda súčastné dáta a smerom do histórie váha klesá. Ako sledované udalosti (kurzy akcií) postupujú v čase, ich váha sa zmenšuje a ich vplyv na výsledok je zanedbateľnejší. EMA je pre sériu udalostí Y definovaný takto:

$$EMA_t = \lambda Y_t + (1 - \lambda) \times EMA_{t-1} \text{ pre } t = 1, 2, \ldots n \tag{3.3}$$

kde

- EMA_0 je priemer minulých dát,
- Y_t je sledovaná udalosť v čase t,
- n počet sledovaných udalostí zahrňujúcich EMA_0,
- $0 < \lambda \leq 1$ je konštanta určujúca váhu sledovanej udalosti.

Parameter λ vyjadruje do akej miery ovplyvňujú staršie dáta konečný výsledok. Ak zvolíme λ blízku 1, iba najnovšie udalosti ovplyvnia konečný výsledok, znížením λ konštanty zvyšujeme váhu starším udalostiam. V technickej analýze za udalosť považuje vývoj kurzu akcie v určitom časovom horizonte.

V praxi sa používa mierne upravená verzia rovnice 3.3 vyjadrená nasledovne:

$$\begin{aligned} EMA_{today} &= EMA_{yesterday} + \lambda \left(price_{today} - EMA_{yesterday}\right) \\ \lambda &= \frac{2}{N-1} \end{aligned} \tag{3.4}$$

Pre výpočet parametra λ je nutné stanoviť veľkosť obdobia N s najväčším vplyvom. Táto perióda predstavuje obdobie s najväčším vplyvom na ďalšiu hodnotu EMA a neznamená hranicu vo výpočte, tak ako je to pri SMA. Týmto (rovnica 3.4) spôsobom vyrátaná lambda predstavuje určité vlastnosti. Na ilustráciu: sledované obdobie má 86% podiel pri vytváraní nasledujúcej hodnoty EMA. V prípade EMA $N = 50$ dní, akciové kurzy za prvých 50 dní majú 86 % podiel na výslednej hodnote a zvyšných 14 % je vplyv z obdobia za hranicou 50 dní.

3.2 Black – Scholes model a indikátry *Greeks*

Tento matematický model a s ním spojené indikátory predstavuje jednu z možností na oceňovanie opčných kontraktov. S opciami sa obchoduje buď zmluvne alebo sa kótujú na špecializovaných opčných burzách.

3.2.1 Model Black – Scholes

Black – Scholes model je matematicky model finančného trhu obsahujúci určité finančné inštrumenty [3]. Tento model umožňuje určiť teoretickú cenu európskych opcií. Pomocou tohto modelu a vhodnej stratégie je možné dosiahnuť napríklad bezrizikovú pozíciu na trhu (*delta hedging*)[13]. Túto myšlienku je možné vyjadriť parciálnou diferenciálnou rovnicou. Jej vyriešením získame Black – Scholes vzorec pre určenie hodnoty opcie:

$$C\left(S,T\right) = S \times \Phi\left(d_1\right) - Ke^{-rT} \times \Phi\left(d_2\right) \tag{3.5}$$

$$P\left(S,T\right) = Ke^{-rT} - S + C\left(S,T\right) = \Phi\left(-d_2\right) Ke^{-rT} - \Phi\left(-d_1\right) S \tag{3.6}$$

$$d_1 = \frac{1}{\sigma\sqrt{T}} \left[ln\left(\frac{S}{K}\right) + \left(r + \frac{\sigma^2}{2}\right) \times T \right] \tag{3.7}$$

$$d_2 = d_1 - \sigma\sqrt{T} = \frac{1}{\sigma\sqrt{T}} \left[ln\left(\frac{S}{K}\right) + \left(r - \frac{\sigma^2}{2}\right) \times T \right] \tag{3.8}$$

kde:

- $C\left(S,T\right)$ – teoretická hodnota pre európsku kúpnu opciu,
- $P\left(S,T\right)$ – teoretická hodnota pre európsku predajnú opciu,
- S – aktuálna cena podkladového aktíva na trhu (*spot price*),
- T – splatnosť opcie (*time to maturity*),
- K – realizačná cena opčného kontraktu (*strike price*),
- r – bezriziková úroková miera,
- σ – očakávaná/implicitná volatilitu,
- $\Phi\left(d_1\right)$; $\Phi\left(d_2\right)$ – distribučná funkcia štandardného normálneho rozdelenia pre koeficienty d_1 a d_2.

Správna funkčnosť Black – Scholes modelu je podmienená rešpektovaním týchto predpokladov. Je vylučujeme možnosť arbitráže. Všetky aktíva sú dokonale deliteľné. Volatilita podkladového aktíva je konštantná. Investor si môže neobmedzene požičiavať za bezrizikovú sadzbu. S obchodovaním nevznikajú žiadne dodatočné náklady. Z akcií nevyplácajú dividendy. Cena podkladového aktíva predstavuje stochastický proces[1]. Aj keď ide o pomerne zjednodušené predpoklady, ich počet je značný a preto je v praxi nedosiahnuteľne, aby cena vypočítaná podla tohto modelu presne reflektovala skutočnosť [20]. Tieto problémy môžu do istej miery riešiť použitím indikátorov rizika *Greeks*.

[1] Konkrétne Brownov pohyb.

3.2.2 *Greeks* indikátory rizika

Ide o skupinu indikátorov, ktoré analyzujú citlivosť ceny finančného derivátu, predovšetkým opcií k rôznym ukazovateľom. Môže to byť cena podkladového aktíva (komodita, akcia, dlhopis), čas, ako aj samotný *Greeks* indikátor. Jednotlivé indikátory sú označované písmenami gréckej abecedy, odkiaľ vzniklo aj ich pomenovanie „*Greeks*“ [15].

Najznámejším indikátorom je delta – Δ, ktorý zachytáva citlivosť ceny opcie vzhľadom k cene podkladového aktíva. Nadobúda hodnoty z intervalu $< -1, 1 >$. V praxi sa však znamienko vynecháva, pretože je implicitne vyjadrené typom opcie. Delta je definovaný ako parciálna derivácia ceny opcie a ceny podkladového aktíva.

$$\Delta = \frac{\partial C}{\partial S} \tag{3.9}$$

Pre kupné a predajné opcie je možné delta vyjadriť aj takto:

$$\Delta_{call} = \Phi\left(d_1\right) \tag{3.10}$$

$$\Delta_{put} = -\Phi\left(-d_1\right) = \Phi\left(d_1\right) - 1 \tag{3.11}$$

Ak máme napríklad opcie s hodnotou $\Delta = 0,15$, tak na rast podkladového aktíva o \$1 bude opcia reagovať rastom o 15 centov.

Delta indikátor sa využíva v rôznych stratégiách investovania a ide o najdôležitejší ukazovateľ správania opcií. Príkladom takejto stratégie môže byť delta neutrál, kde sa investor snaží o pozíciu s $\Delta \approx 0$, čím sa odstráni riziko plynúce z pohybu ceny podkladového aktíva. Delta participuje aj v ďalších stratégiách a je celkovo investormi hojne používaný.

Kapitola 4

Tvorba a návrh technických indikátorov

Táto kapitola je venovaná technickým indikátorom a problémoch spojených s ich implementáciou. Jedným z najvážnejších je absencia štandardnej matematickej knižnice C++.

S čiastočným riešením prichádza Catapult vlastnou implementáciu základných matematických funkcií. Táto implementácia však neobsahuje všetky funkcie používané v tejto práci. Jedinou možnosťou bolo preto prísť s vlastným riešením, čo znamenalo vyhľadať vhodné aproximačné algoritmy a upraviť ich tak aby boli syntetizovateľné. Detaily popisuje sekcia 4.2.

Implementácia bola rozdelená do dvoch častí. Prvú čať tvoria matematické funkcie a druhou je skupina technických indikátorov. V tejto časti sú spravidla indikátory implementované viacerými spôsobmi od ktorých sa ďalej odvíja latencia a množstvo spotrebovaných zdrojov.

Samotné algoritmy sú platformne nezávislé, no pre účely testovania bolo nutné už na začiatku zvoliť cieľovú technológiu. Tú predstavujú FPGA obvody Virtex-6 od firmy Xilinx (VIRTEX – 6 rýchlosť: -2 označenie: 6VLX75TFF484). Pokiaľ nie je uvedené inak, tak pracovná frekvencia hodín je 100 Mhz. V rámci nastavení dizajnu je povolená pamäť RAM (jednoportova a dvojportova) pre uchovávanie rozsiahlejších polí.

4.1 Reprezentácia dát

HLS nástroje, konkrétne Catapult používa k popisu správania C++, takže mojou snahou bolo zachovať aj pôvodné dátové typy. Na reprezentáciu cien a výstupov technických indikátorov by slúžil dátový typ `double` alebo `float`. Tento prístup je v prípade že chceme C++ syntetizovať nevhodný a ako už bolo spomenuté v 2.2.2 Catapult si tieto typy konvertuje a z tohto dôvodu sa ich použitie nedoporučuje. Jednotný prístup k dátovým typom ponúka *Algoritmic C*. Programátor má možnosť presne definovať šírku, definovať sémantiku a správanie v neštandardných situáciach. Výhody spojené s takýmto prístupom prevažujú nad možnosťami bežných C++ typov alebo typov z SystemC. V tabuľke 4.1 je prehľad dátových typov používaných pri všetkých výpočtoch uskutočňovaných v tejto práci. Najčastejšie používaný je práve typ `q16_16`, ktorý poskytuje dostatočný rozsah aj presnosť pre výpočty.

K uchovávaniu cien slúži štruktúra `tHistoryPrice`. Obsahuje index na najaktuálnejšiu hodnotu a pole s cenami. Pri aktualizácií sa vždy odstráni najstarší prvok a index sa

Názov	Algorithmic C	Rozsah	Kvantovanie
qshort	`ac_int<16,true>`	-32 768 až 32 767	1
qint	`ac_int<32,true>`	-2 147 483 648 až 2 147 483 647	1
qint_u	`ac_int<32,false>`	0 až 4 294 967 295	1
qlong	`ac_int<64,true>`	-9 223 372 036 e09 až 9 223 372 036 e09	1
q16_16	`ac_fixed<32,16,true>`	-32 768 až 32 767.99998	1,52588e-05
q32_32	`ac_fixed<64,32,true>`	-2 147 483 648 až 2 147 483 647,99999	2,32831e-10
q16_16_u	`ac_fixed<32,16,false>`	0 až 65535,99998	1,52588e-05

Tabuľka 4.1: Prehľad použitých dátových typov

aktualizuje tak, aby uchoval pozíciu aktuálnej hodnoty. Tato štruktúra môže podľa potrieb uchovať 16, 32, 64 alebo až 128 položiek. Veľkosť nie je zvolená náhodne, ale tak aby bolo zabezpečene optimálne využitie zdrojov a jednoduchú zmenu indexu.

4.2 Použité matematické funkcie

Pre uskutočnenie výpočtov technických indikátorov je nutné implementovať niekoľko matematických funkcií. Funkcie implementované samostatne, kvôli jednoduchej správe. V prípade že ich chceme použiť, tak ich jednoducho importujeme do zdrojových kódov. Pri použití algoritmov boli vo funkciách ešte aplikované techniky na zlepšenie parametrov, predovšetkým latencie. Prehľad vlastnosti jednotlivých funkcií je uvedený v tabuľkách 4.2, 4.3, 4.4 a 4.5.

4.2.1 Logaritmus zo základom 2

Pri snahe minimalizovať odozvu je možné na niektorých miestach nahradiť delenie bitovým posunom, čo výrazne zlepší priepustnosť a zníži latenciu. Pre určenie veľkosti tohto posunu sa využíva logaritmus so základom 2 z daného čísla. Táto implementácia je obmedzená len na celé kladné čísla, čo pre zistenie veľkosti posunu postačuje. Vstupom je číslo typu `qint_u`. Výstupom je kladné celé číslo typu `qint_u` zaokrúhlené nadol. Samotný logaritmus využíva jednoduchú tabuľku a sériu porovnávaní [2].

Pri takejto jednoduchej funkcií sa vygenerovaná architektúra vyznačuje nízkou latenciou a prijateľnou spotrebou zdrojov. Pri experimentovaní so zmenou parametrov syntézy (zreťazenie hlavného cyklu) nedošlo k výraznemu zlepšeniu. Latencia vygenerovanej architektúry je 10 ns, čo predstavuje jeden hodinový cyklus. Veľkosť plochy na čipe je ohodnotená na $970,68$. Pri nahliadnutí do reportu je vidieť že celý dizajn je mapovaný v LUT tabuľkách. Najväčšiu časť tvoria multiplexory, celkovo je ich v dizajne 4 a každý z nich zaberá $230,037$.

4.2.2 Prirodzený logaritmus

Logaritmus so základom e je využívaný pri výpočte parametrov d_1 a d_2 z Black – Scholes modelu popísaného v sekcii 3.2.1. Vstupom je kladné číslo typu `q16_16`. Špeciálne prípady vstupu, ako sú 0 ($log_e(0.0) = -\infty$) a 1 ($log_e(1.0) = 0$) sú ošetrené na začiatku funkcie.

Pôvodný algoritmus bol implementovaný v dvoch rôznych modifikáciách (`qlog, qlog_mod`) [12]. Prvá verzia funkcie – `qlog` mení pôvodný algoritmus len minimálne. Druhá verzia funkcie označená ako `qlog_mod` nahrádza početné delenie násobením so snahou o dosiahnutie

cyklus			vlastnosti		
log_modify	log_computation	div#1:for	latencia [ns]	priepustnosť [ns]	zdroje
–	–	–	4 120	4 130	4 079,41
–	U=8 (max)	U=48 (max)	140	150	10 248,24
II=1	U=4, II=2	U=48, II=2	240	250	15 424,6
–	II=1	U=48 (max)	160	170	4 583,11
II=1	II=1	U=48 (max)	130	90	6 127,31

Tabuľka 4.2: Prehľad vplyvu rôznych parametrov syntézy na latenciu a spotrebu zdrojov vo funkcii `qlog`.

nižšej odozvy a celkovo lepších vlastností obvodu. Zároveň vstupnému číslu odstránime desatinu čiarku čím sa z neho stane číslo celé a celý algoritmus tak využíva len celočíselnú aritmetiku. Po dokončení výpočtov je výsledok opäť konvertovaný do požadovaného formátu desatinného čísla. Obe funkcie sa líšia iba samotnou implementáciou algoritmu, ich rozhranie je zhodné.

Algoritmus je zložený z dvoch cyklov. Prvý, s návestím `log_modify:` násobí vstup konštantou $1/e$. Jeho úlohou je znížiť vstup pod číslo 2,0. Je to cyklus s vopred neurčeným počtom opakovaní, preto ho nie je možné úplne rozbaliť. Další cyklus označený návestím `log_computation:` násobí a delí upravený vstup. Iterácie tohto cyklu tvoria a upresňujú výsledok. Tento cyklus je možné plne rozbaliť, keďže je známy počet jeho opakovaní.

Vplyv rôznych parametrov na celkovú latenciu a priepustnosť je zachytený v tabuľkách 4.2 a 4.3. Z tabuliek možno usudzovať, že bez paralelizmu a zreťazenia majú obvody vysokú latenciu z tohto dôvodu používame zreťazenie a paralelizmus s rôznymi nastaveniami. Stupeň rozbalenia je v označený ako U a zreťazenie ako II, toto označenie používa aj program Catapult. Úplným rozbalením cyklov výrazne zlepšime celkovú latenciu, avšak na úkor zdrojov. V prípade `qlog` funkcie sa ukázalo ako najvhodnejšie ponechať cyklus delenia plne rozbalený (U=48), a zvyšné cykly zreťaziť s II=1. V detailnom reporte sa dozvieme, že táto verzia používa dve násobičky. Jedna obsadí 2 DSP bloky a druhá 4 DSP bloky. Celkovú spotrebu tvoria 1291 LUT, 6 DSP48E a 756 MUX_CARRY.

Druhá verzia funkcie (`qlog_mod`) má bez akýchkoľvek úprav podstatne lepšie vlastnosti. Plné rozbalenie cyklu nemalo na latenciu výraznejší vplyv a najlepšou voľbou sa ukázala aplikácia zreťazenia s II=2. Pri nastavení II=1, proces plánovania zlyhal. Ukázalo sa, že program Catapult nedokázal naplánovať požadované operácie (chyba SCHD-3).

Pri pohľade na obe tabuľky (4.2 a 4.3) zisťujeme, že po nastavení vhodných parametrov je latencia funkcií `qlog` a `qlog_mod` podobná a uskutočnené modifikácie vo funkcií `qlog_mod` nemajú na latenciu až taký výrazný vplyv.

4.2.3 Exponenciálna funkcia

Exponenciálna funkcia sa používa pri výpočte teoretických cien vo vzorcoch 3.5, 3.6 a tak ako logaritmus ani exponenciálna funkcia nie je súčasťou matematickej knižnice Catapultu. Z tohto dôvodu ju bolo potrebné implementovať. Pôvodný algoritmus je implementovaný bez väčších úprav [12] a vstupom je desatinné číslo typu `q16_16`.

V tabuľke 4.4 je možné vidieť prehľad o latencii a spotrebovaných zdrojoch pri rôznych nastaveniach syntézy algoritmu. Latencia v tabuľkách nie je konečná, pretože nepoznáme počet iterácií cyklu s návestím `exp_computation`. S istotou vieme, že sa nebude meniť

cyklus		vlastnosti		
log_modify	log_computation	latencia [ns]	priepustnosť [ns]	zdroje
–	–	360	370	5 264,66
–	U=8 (max)	210	220	7 179,17
–	U=4	250	260	5 537,51
II=2	II=2	190	180	5 225,19

Tabuľka 4.3: Prehľad vplyvu rôznych parametrov syntézy na latenciu a spotrebu zdrojov vo funkcii `qlog_mod`.

cyklus		vlastnosti		
exp_computation	div#1:for	latencia [ns]	priepustnosť [ns]	zdroje
–	–	530	540	3 792,01
–	U =48 (max)	80	90	4 926,00
U=4	U=48 (max)	110	120	8 017,30
II=2	II=2	980	990	4 202,13
II=2	U=4 II=2	260	270	4 265,03

Tabuľka 4.4: Prehľad vplyvu parametrov syntézy na latenciu a spotrebu zdrojov vo funkcii `qexp`

je ohodnotenie zdrojov. To je dostupné po ukončení procesu plánovania a definitívne sa upresní po vygenerovaní RTL schém. Každá komponenta je ohodnotená podľa spotreby LUT alebo DSP blokov. Sčítaním všetkých ohodnotení získame celkovú hodnotu spotrebovaných zdrojov. Môžeme tak jednoducho porovnať vplyv rôznych parametrov na spotrebu. Kto má záujem o detaily jednoducho nahliadne do RTL reportu kde nájde podrobné detaily o každej použitej komponente.

4.2.4 Distribučná funkcia normálneho rozdelenia pravdepodobnosti

Distribučná funkcia udáva pravdepodobnosť, že hodnota náhodnej premennej je menšia ako hodnota zadanej hodnoty. Distribučná funkcia normálneho rozdelenia sa niekedy označuje gréckym písmenom Φ, čoho sa držíme aj v tejto práci.

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-t^2}{2}} \mathrm{d}t \tag{4.1}$$

Často sa používa pri oceňovaní kontraktov a je prítomná aj v modeli Black – Scholes, takže jej implementácií sa nevyhneme. Pri vyčísľovaní jej hodnoty používame funkciu zo štatistiky a to Gaussovu chybovú funkciu – *erf* zpravidla aproximovanú polynomom piateho stupňa a exponenciálnou funkciou podľa Abrahamowitz a Stegun, vzorec 7.1.26 [1].

$$\Phi(x) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right] \tag{4.2}$$

cyklus			vlastnosti		
div#1:for	exp_computation	div#3:for	latencia [ns]	priepustnosť [ns]	zdroje
–	–	–	32 740	32 750	26 291,84
U=64	U=10 (max)	U=48	250	260	39 511,52
U=64	II=1	U=48	410	420	30 544,47
U=64	II=1	U=48	340	100	49 692,96

Tabuľka 4.5: Prehľad vplyvu parametrov syntézy na latenciu a spotrebu zdrojov vo funkcii `qphi`

$$\mathrm{erf}(x) = 1 - \left(a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5\right) e^{-x^2} + \epsilon(x), \qquad (0 \leq x < \infty) \tag{4.3}$$

$$t = \frac{1}{1 + px}$$

$$|\epsilon(x)| \leq 1.5 \times 10^{-7}$$

$$p = .32759\ 11 \qquad a_1 = .25482\ 9592 \qquad a_2 = -.28449\ 6736$$

$$a_3 = 1.42141\ 3741 \qquad a_4 = -1.45315\ 2027 \qquad a5 = 1.06140\ 5429$$

Uvedený výpočet Φ je súčasťou tejto práce v podobe funkcie `qphi`. Táto aproximácia je pomerne náročná na zdroje a bez paralelizmu má aj vysokú latenciu. Prehľad o spotrebe zdrojov je uvedený v tabuľke 4.5. Jednotlivé označenia zodpovedajú vysvetlivkám v predchádzajúcom texte. Cyklus s návestím `exp_computation` a delenie `div#3:for` je súčasťou volanej exponenciálnej funkcie `qexp`. Posledné dve verzie/riadky funkcie/tabuľky sa môžu zdať parametrami rovnaké, no poslednom riadku je zreťazený aj hlavný cyklus programu a generované distribuované zreťazenie.

4.3 Jednoduchý kĺzavý priemer – SMA

Tento kĺzavý priemer patrí medzi technické indikátory s jednoduchým výpočtom popísaným v 3.1.1. Samotné sčítanie nepredstavuje náročnú operáciu. To ale neplatí o delení, ktoré je náročnejšie na zdroje a má výrazný vplyv na celkovú latenciu. Možnosťou ako zlepšiť latenciu a zároveň aj zmenšiť spotrebu zdrojov je uskutočniť delenie posunom. Takéto delenie je možné v prípade že deliteľ je mocnina čísla 2, čo znamená, že napríklad 30–dňovú periódu SMA zmeniť je potrebné zmeniť na 32 dní a podobne. Pokiaľ sa nechceme obmedzovať v perióde na mocniny čísla 2, deleniu sa nevyhneme. Pre zaistenie lepších vlastností obvodu používame¡ techniky rozbaľovania a zreťazenie cyklov.

4.3.1 Rozhranie

Pri popisovaní obvodov jazykom C++ predstavuje rozhranie obvodu hlavička funkcie a HLS nástroj rozpozná o aký druh rozhrania ide (vstupné/výstupne/vstupno–výstupné). Pre výpočet SMA podľa 3.1 bolo rozhranie zvolené takto:

```
q16_16 sma(tPriceHistory128 *history, qint_u period, qint_u index=0);
```

- `tPriceHistory128 *history` – ukazateľ na štruktúru, kde sú uchované ceny akcií sledovanej spoločnosti. Táto štruktúra môže podľa potreby uchovať 16/32/64/128 hodnôt. V tomto prípade ide o históriu cien za posledných 128 dní čo prezrádza číslo na konci označenia štruktúry.
- `qint_u period` – veľkosť časového úseku, v našom prípade počet dní, z ktorých je počítaný priemer.
- `index` – označuje miesto v štruktúre, kde sa začína počítať.

V prípade, že chceme vypočítať 50-dňové SMA so začiatkom na 10 dni v sledovanej histórii budú parametre nastavené takto: `period=50 index=10`.

Pre výpočet podľa vzorca 3.2, kde počítame len nasledujúcu hodnotu je rozhranie mierne rozšírené:

```
q16_16 sma_next(tPriceHistory128 *history, q16_16 prev,
                q16_16 close, qint_u period, qint_u index=0);
```

a pridané sú:

- `q16_16 prev` – predchádzajúca hodnota SMA
- `q16_16 close` – nasledujúca cena sledovanej spoločnosti.

Ak `index` nadobúda hodnoty rôznej od 0, je ďalšia cenu zistená zo štruktúry histórie cien, v opačnom prípade (`index`=0) je použitá hodnota `close`.

Keďže obvody popisujeme abstraktne, ďalšie signály typické pre obvody si HLS nástroj definuje sám. Napríklad hodinový signál, reset signál a v prípade RAM aj signály riadiace prístup do pamäte sú definované automaticky. Program mám zároveň ponecháva možnosť upraviť tieto generované signály poľa našich potrieb. Máme možnosť nastaviť oneskorenie a synchronizáciu so zostupnou alebo vzostupnou hranou hodinového signálu.

4.3.2 Implementácia

Z 3.1.1 vieme že hodnota SMA sa dá vyjadriť dvom spôsobmi. Pri obidvoch spôsoboch môžeme nahradiť delenie bitovým posunom. To nám dáva celkovo 4 možnosti ako vypočítať tento kĺzavý priemer. Každá z možností je implementovaná ako samostatná funkcia. Umožňuje nám to jednoducho porovnať ich latenciu a spotrebu zdrojov. Varianty bez delenia sú označené príponou `_mod` (`sma_mod, sma_next_mod`). Vo všetkých funkciách sa nachádza kontrola či perióda nepresahuje povolený rozsah. Perióda môže byť nanajvýš taká veľká ako veľkosť dostupnej histórie cien.

sma

Implementácia pozostáva z jedného cyklu a delenia. Pri ďalšom skúmaní zistíme, že delenie je implementované cyklom. Cyklus delenia ma podľa bitovej šírky vstupných operátorov rôzny počet iterácií, pričom v tomto prípade ide o cyklus s 80 – imi iteráciami. Pred delením ešte prebehne cyklus s návestím `PRICE_ACC`. V ňom sa vypočíta index do histórie cien, z ktorej načítana hodnota sa pripočíta k výsledku sčítania z predchádzajúcej iterácie.

sma_mod

Cyklus s návestím `PRICE_ACC` je zhodný, no delenie je nahradené bitovým posunom. To si však vyžaduje aby perióda bola mocninou čísla 2. Správnosť periódy nie je kontrolovaná a v prípade iného vstupu bude výsledok nekorektný. Delenie bolo odstránené, čo pozitívne

Funkcia	Cyklus		Vlastnosti		
	price_acc	div#1:for	latencia [ns]	priepustnosť [ns]	zdroje
sma	–	–	850	860	1 001,44
sma	U=4	U=80 (max)	130	140	3 370,63
sma	II=2	U=80 (max)	90	20	3 241,61
sma	–	U=10	180	180	1 478,08
sma_mod	–	————	40	50	1 873,26
sma_mod	U=4 II=2	————	40	20	2 126,74
sma_mod	U=2 II=1	————	30	10	2 054,40

Tabuľka 4.6: Prehľad vplyvu parametrov syntézy na latenciu a spotrebu zdrojov vo funkciách `sma` a `sma_mod`

vplýva na latenciu.

sma_next
Celý algoritmus pozostáva z dvoch delení, sčítania a odčítania. Každé delenie má 48 iterácií, čo má negatívny dopad na latenciu. Pred samotným delením je ešte nutné určiť index ktorému zodpovedá posledná a prípadne aj ďalšia hodnota v štruktúre histórie cien.

sma_next_mod
Algoritmus funkcie je rovnaký s jedinou zmenou. Delenie je tu uskutočňované bitovým posunom, čo pozitívne vplýva na latenciu. Vyššia spotreba zdrojov je spojená s volaním funkcie `fastlog2`. Táto slúži na správne určenie veľkosti posunu. Potenciálnou prekážkou v použití sa javý nevyhnutnosť poznať predchádzajúcu hodnotu priemeru a dodržiavať periódu sledovaného obdobia.

4.3.3 Parametre syntézy

Pre získanie optimálnych výsledkov syntézy je nevyhnutné definovať určitý počet parametrov, ktoré v konečnom dôsledku vplývajú na celkový výsledok. Možnosť voľby máme už pri určovaní typu rozhrania. Štruktúra slúžiaca na uchovávanie cien obsahuje pole a tu máme možnosť vybrať spôsob jeho organizácie v pamäti. Pole môže byť mapované priamo, alebo ako pamäť typu RAM. Pri väčších poliach priame mapovanie nepriaznivo zvyšuje spotrebu zdrojov. Pri implementácií kĺzavých priemerov bola pre uchovanie štruktúry `tHistryPrice` zvolená dvojportova pamäť RAM.

Ďalšie možnosti pre úpravu nám ponúkajú cykly, kde uplatníme rozbaľovanie a zreťazenie v rôznych kombináciách. Snažíme sa nájsť architektúru s čo najnižšou latenciu a minimálnou spotrebou zdrojov. Dosiahnuté výsledky sú prezentované v tabuľkách 4.6 a 4.7. Každý riadok tabuľky predstavuje jednu verziu funkcie, jej parametre a vplyv týchto parametrov na vlastnosti vygenerovaného obvodu. Pre jednoduchú identifikáciu použitých parametrov sú cykly označené návestím. Rozbalenie je označené písmenom U a číslom ktoré vyjadruje mieru paralelizácie. Zreťazenie ma skratku II, ktorá je doplnená o jeho stupeň. Vidíme, že latencia modifikovaných verzií je aj bez akýchkoľvek úprav nízka. To je dané absenciou delenia, avšak paralelizmom a zreťazením je možné dosiahnuť podobných vlastností aj u nemodifikovaných funkciách. Verzia funkcie `sma` s čiastočným rozbalením predstavuje kompromis medzi latenciou a spotrebovanými zdrojmi.

Funkcia	Cyklus		Vlastnosti		
	div#1:for	div#3:for	latencia [ns]	priepustnosť [ns]	zdroje
sma_next	–	–	1 020	1 030	1 123,14
sma_next	U=48 (max)	U=48 (max)	70	20	2 752,03
sma_next	U=8 II=2	U=8 II=2	240	240	1 982,57
sma_next_mod	————	————	40	50	2 219,83
sma_next_mod	————	————	30	10	2 171,39

Tabuľka 4.7: Prehľad vplyvu parametrov syntézy na latenciu a spotrebu zdrojov vo funkciách `sma_next` a `sma_next_mod`.

4.3.4 Zhodnotenie

Výsledky ukázali, že implementácia týchto funkcií môže byť veľmi efektívna. Pokiaľ by sme sa obmedzili len na určité vstupy latencia obvodu predstavuje len 3 hodinové takty. V procese plánovania bola časť výpočtov presunutá do DSP48E/DSP48El blokov. Získali sme tak základný technicky indikátor, ktorý je možné použiť samostatne alebo zakomponovať do vlastných obchodných pravidiel. Úspory zdrojov sme sa v prípade modifikovaných verzií nedočkali. To je spôsobené volaním funkcie `fastlog2`. Tá môže tvoriť až 50% z celkovej spotreby zdrojov. Modifikované verzie nám pri porovnateľnej spotrebe zdrojov ponúkajú výrazne nižšiu latenciu.

4.4 Exponenciálny kĺzavý priemer – EMA

Stanovenie exponenciálneho kĺzavého priemeru je možné rozložiť do dvoch častí. Najprv vyrátame koeficient lambda, ktorý sa následne použije k výpočtu novej hodnoty priemeru. Postupujeme podľa rovnice 3.4. Pri vhodne zvolenej perióde sa opäť môžeme vyhnúť deleniu vo výpočte lambda. Toto obmedzenie nemusí byť vždy akceptovateľné, príkladom je indikátor MACD.

4.4.1 Rozhranie

Rozhranie definuje hlavička C++ funkcie. Program Catapult rozpozná druh parametrov na základe spôsobu používania vo funkcii. Toto nastavenie nie je smerodajné a môžeme ho podľa vlastného uváženia korigovať. Pri vytváraní tohto indikátora som sa rozhodol, že celý výpočet rozdelím do dvoch samostatných funkcií. V prípade, že lambdu poznáme zavoláme iba funkciu `ema_computation` a ušetríme zdroje. Častejším prípadom však je, že lambda nepoznáme. Pre výpočet podľa 3.4 je rozhranie zvolené nasledovne:

```
q16_16 ema(q16_16 ema_prev, q16_16 close_price, qint_u period);
q16_16 ema_mod(q16_16 ema_prev, q16_16 close_price, qint_u period);
```

- `ema_prev` – predchádzajúca hodnota kĺzavého priemer,
- `close_price` – cena pre obdobie pre ktoré počítame EMA,
- `period` – veľkosť sledovaného obdobia.

```
q16_16 ema_computation(q16_16 ema_prev, q16_16 lambda, q16_16 close_price);
```

- `lambda` – parameter lambda

Parametre funkcií sa líšia jedine v tom, že namiesto periódy dosadíme už hodnotu lambdy.

4.4.2 Implementácia

Pri implementácií máme teda 2 možnosti stanovenia novú EMA a to s delením a posunom. Obe sú implementované samostatne, čo nám neskôr umožní ich jednoduché porovnanie. Funkcia používajúca delenie je označená `ema` a funkcia s bitovým posunom má označenie`ema_mod`.

ema
Implementácia obsahuje delenie na výpočet lambda a volanie funkcie `ema_computation`. Ako je známe program Catapult implementuje delenie cyklom. Vzhľadom na bitovú šírku vstupov (typ `q16_16` – 32 bitov a znamienko) má tento cyklus 48 iterácií.

ema_mod
Výpočet lambda nie je implementovaný klasickým delením ale bitovým posunom. Táto modifikácia si vyžaduje aby bol deliteľ mocninou čísla 2. To pri pohľade na vzorec 3.4 znamená, že pre periódu musí platiť nasledovný vzťah: perióda $= 2^n - 1$. V opačnom prípade by výsledok tohto delenie bol nekorektný. A bez validného výsledku je chybný aj další postup ako aj celkový výsledok. Pre určenie veľkosti posunu použijeme funkciu `fastlog2`. Výsledok zmenšíme o hodnotu 1 a následne vynásobíme číslom 2, čo docielime aj bitovým posunom vľavo o hodnotu 1. Takto upravená perióda určí veľkosť posunu vpravo, ktorý uplatníme na delenec – číslo 2, a takto dostaneme koeficient lambda. Celý postup môžeme zapísať ako: $\lambda = 2.0 >> [(log_2\,(perioda) - 1) * 2]$. Ďalej nasleduje volanie funkcie `ema_computation` pre výpočet novej hodnoty priemeru.

4.4.3 Parametre syntézy

Pri nastavovaní parametrov syntézy sú naše možnosti v porovnaní s SMA obmedzené. S parametrami je možné experimentovať jedine pri delení a hlavnom cykle programu. Vplyvy modifikácií a zvolených parametrov na latenciu ako aj spotreba zdrojov sú zobrazená v tabuľke 4.8.

V tabuľkách sa nám opäť sa nám potvrdzuje, že po rozbalení delenia latencia výrazne klesá a použitím zreťazenia sa priepustnosť dostáva do priaznivejších čísel. Pri takto navrhnutej funkcii `ema` je najlepšou voľbou parameter zreťazenia II=2, pri použití II=1 je syntéza neúspešná. Dôvodom je, že program Catapult nedokáže načasovať požadovanú sériu operácií (chyba SCHD-3). Takéto zlyhanie syntézy nastáva pri uvedení neplatných parametrov alebo práve pri nevhodnom stupni zreťazenía. Modifikovaná funkcia – `ema_mod` ma nízku latenciu, no spotreba zdrojov neklesla. Pri nahliadnutí do detailnejších správ, ktoré sú k dispozícií po vygenerovaní RTL zistíme príčinu tejto spotreby. Vyššie nároky na zdroje spôsobuje volanie funkcie `fastlog2`, kde sú použité 4 multiplexory (každý obsadí 231 LUT). Ďalšou komponentou s výraznou spotrebou zdrojov je posuvný register vpravo (493 LUT), ktorý realizuje bitový posun. Vo všetkých modifikáciách je použitá násobička, ktorá je umiestnená v 4 blokoch DSP (DSP48E).

Funkcia	Cyklus		Vlastnosti		
	main	div#1:for	latencia	priepustnosť	zdroje
ema	–	–	510,00	520,00	3 608,36
ema	–	U=48 (max)	70,00	80,00	4 683,07
ema	II=2	U=48 (max)	70,00	20,00	4 822,99
ema_mod	–	–	20,00	30,00	4 657,47
ema_mod	II=1	————	20,00	10,00	4 618,55
ema_computation	–	————	20,00	30,00	3 197,02

Tabuľka 4.8: Prehľad vplyvu parametrov syntézy na latenciu a spotrebu zdrojov vo funkciách `ema` a `ema_mod`.

4.4.4 Zhodnotenie

V exponenciálnom kĺzavom priemere sme predstavili dva možné spôsoby výpočtu parametra lambda. Kým v prvom nie sme obmedzený periódou sme zaťažený delením. V druhom sme delenie kompletne vynechali a použili len bitové posuny, sme však obmedzený periódou čo nám nemusí vždy vyhovovať. Pri násobení je požitá rýchla násobička z DSP blokov (oneskorenie 0.653 ns) pomocou ktorej vieme efektívne znížiť latenciu.

4.5 Black – Scholes model a Delta indikátor

Spolu vytvárajú najzložitejšiu skupina funkcií. K výpočtu sa využívajú takmer všetky implementované matematické funkcie (`qlog, qexp, qphi`). O zložitostí svedčí aj fakt, že generovanie RLT popisu trvalo v niektorých prípadoch vyše 1 hodiny. Implementované sú tu funkcie na počítanie teoretických cien opcií a Delta indikátory pre kúpne (*call*) aj predajné (*put*) opcie.

4.5.1 Rozhranie

Jednotlivé funkcie sú implementované tak ako to popisuje sekcia 3.2.1. Všetky funkcie pracujú z rovnakými vstupmi, preto je aj ich rozhranie jednotné. Návratová hodnota funkcií je dátového typu `q16_16` alebo `q16_16_u` pretože hodnota niektorých funkcií, napríklad `C` a `P` ani záporná byť nemôže. Taktiež všetky vstupné parametre sú kladné. Tieto špecifiká sme brali na zreteľ aj pri tvorení rozhrania. Toto je v podobe hlavičiek funkcií nasledovné:

```
q16_16_u
C(q16_16_u S, q16_16_u K, q16_16_u r, q16_16_u s, q16_16_u T);
q16_16_u
P(q16_16_u S, q16_16_u K, q16_16_u r, q16_16_u s, q16_16_u T);
q16_16
d1(q16_16_u S, q16_16_u K, q16_16_u r, q16_16_u s, q16_16_u T);
q16_16
d2(q16_16_u S, q16_16_u K, q16_16_u r, q16_16_u s, q16_16_u T);
q16_16
delta_call(q16_16_u S, q16_16_u K, q16_16_u r, q16_16_u s, q16_16_u T);
q16_16
```

```
delta_put(q16_16_u S, q16_16_u K, q16_16_u r, q16_16_u s, q16_16_u T);
```

- `S` – hodnota aktuálnej ceny.
- `K` – hodnota realizačnej ceny.
- `r` – bezriziková úroková miera v percentách.
- `s` – percentuálne vyjadrenie volatility.
- `T` – splatnosť opcie.

Pri percentuálnom vyjadrení sa myslí hodnota z intervalu $< 0, 1 >$. V prípade hodnoty `T` je tento čas vyjadrený v rokoch. Napríklad pre opciu, ktorá expiruje za 50 dní je to: $T = 50/365 = 0.13699$. Bezriziková úroková miera je teoretický pojem, pretože každá investícia so sebou riziko prináša.

4.5.2 Implementácia

V tomto štádiu máme všetky potrebné matematické funkcie k dispozícií. Implementácia tak spočíva v kombinácií aritmetických operácií a volaní týchto funkcií. Pri tvorbe algoritmu sme využili jednu z funkcií matematickej knižnice programu Catapult, ktorou je odmocnina. Označená je rovnako ako v C++ (`sqrt`), vyžaduje kladný vstup a obsahuje jeden cyklus. Cyklus má poľa špecifikácie vstupu rôzny počet iterácií, vždy však ide o vopred známy počet. Ak je na vstupe číslo typu `q16_16_u` má tento cyklus 24 iterácií. Pri výpočte sme postupovali spôsobom, že sme si najprv vyjadrili čiastkové výsledky na základe ktorých sme stanovili konečný výsledok. Tento postup umožňuje zdieľanie zdrojov v procese plánovania.

4.5.3 Parametre syntézy

Pri takto zložitých funkciách je nájsť vhodné parametre veľmi obtiažne. Pri zle zvolených obmedzeniach syntézy končíme väčšinou neúspešným plánovaním. Spravidla končíme s chybou SCHD-3, kedy Catapult nedokáže naplánovať zvolenú sériu operácií alebo s chybou SCHD-4 kedy Catapult nedokáže uskutočniť plánovanie ani s neobmedzeným množstvom zdrojov. Tieto chyby súvisia predovšetkým s parametrom zreťazenia. V prípade, že aplikujeme len rozbaľovanie, vyššie spomenutým chybám sa dokážeme vyhnúť.

Pri týchto funkciách sú možnosti na experimentovanie veľké. Parametre nemusíme uplatniť na všetky cykly, môžeme si vybrať len tie s vysokým počtom iterácií. Ak je možné aplikovať zreťazenie, pôjde len o niektoré časti algoritmu, inak skončíme s chybou už pri plánovaní.

Prehľady spotreby zdrojov pre jednotlivé funkcie po aplikovaní rôznych obmedzení sú spracované v tabuľkách 4.9, 4.10 a 4.11. Tabuľky v tejto sekcii už nezobrazujú konkrétne cykly s nastaveniami parametrov, pretože pre všetky cykly aplikujeme zhodné nastavenia. Uvedením všetkých cyklov (v niektorých prípadoch aj 15 cyklov) by tabuľka stratila svoju prehľadnosti. Každý riadok predstavuje jednu funkciu z Black – Schles modelu a stĺpce nás informujú o vlastnostiach vygenerovaného obvodu. Okrem všeobecného ohodnotenia zdrojov sú tu pridané informácie o spotrebovaných DSP blokoch, LUT a MUX_CARRY. Tieto informácie sú dostupné po vygenerovaní RTL v reporte *RTL* v časti *Bills of Materials*.

Tabuľka 4.9 informuje o spotrebe zdrojov a latencii bez aplikovania akýchkoľvek obmedzení. Ukazuje nám, že veľký počet delení spôsobuje vysokú latenciu. Preto som sa rozhodol

Funkcia	Vlastnosti		Spotreba			
	Latencia	Priepustnosť	DSP48E	LUT	MUX_CARRY	zdroje
d1	5 730 ns	5 380 ns	14	1 707	340	12 779,87
d2	5 390 ns	5 400 ns	14	2 010	340	13 082,58
delta_call	6 000 ns	6 010 ns	42	3 825	777	37 009,08
delta_put	6 010 ns	6 020 ns	42	3 832	777	37 016,17
P	7 050 ns	7 060 ns	74	7 108	1 230	65 439,24
C	7 080 ns	7 090 ns	53	6 975	1 019	48 878,11

Tabuľka 4.9: Vlastnosti funkcií bez aplikovania rozbalenia a zreťazenia

Funkcia	Vlastnosti		Spotreba			
	Latencia	Priepustnosť	DSP48E	LUT	MUX_CARRY	zdroje
d1	840 ns	850 ns	8	6 100	3 012	12 534,96
d2	840 ns	850 ns	14	6 867	3 378	17 999,59
delta_call	1 160 ns	1 170 ns	37	10 594	4 857	40 084,57
delta_put	1 160 ns	1 170 ns	37	10 620	4 857	40 096,33
P	1 230 ns	1 240 ns	72	20 688	8 985	77924,22
C	1 220 ns	1 230 ns	72	20 691	9 001	77953,13

Tabuľka 4.10: Vlastnosti funkcií po aplikovaní plného rozbalenia

zížiť latenciu aplikovaním plného rozbalenia na všetky cykly delenia a výpočtu odmocniny. Dosiahnuté výsledky sú zobrazené v tabuľke 4.10. Toto rozbalenie preukázalo pozitívny vplyv na latenciu a priepustnosť. Posledný experiment spočíval v aplikovaní iba čiastočného rozbalenia. Rozbalenie s parametrom U=8 bolo opäť aplikované na všetky cykly delenia a cykly výpočtu odmocniny. Výsledky s čiastočným rozbalením U=8 sú znázornené v tabuľke 4.11. Pri pohľade na tabuľky 4.10 a 4.11 zisťujeme, že rozdiely v latencii predstavujú zanedbateľnu hodnotu. Plné rozbalenie sa teda neosvedčilo a zbytočne by nám zaberalo zdroje na čipe.

Funkcia	Vlastnosti		Spotreba			
	Latencia	Priepustnosť	DSP48E	LUT	MUX_CARRY	zdroje
d1	1 090 ns	1,100 ns	14	3 076	813	14 145,55
d2	1 100 ns	1 110 ns	12	4 032	813	13 535,51
delta_call	1 290 ns	1 300 ns	37	5 967	1 603	35 231,40
delta_put	1 300 ns	1 310 ns	33	6 057	1 603	32 190,19
P	1 480 ns	1 490 ns	74	10 067	2 011	68 870,33
C	1 480 ns	1 490 ns	74	10 499	2 054	68 442,64

Tabuľka 4.11: Vlastnosti funkcií pri aplikovaní čiastočného rozbalenia (U = 8)

4.5.4 Zhodnotenie

Aj napriek zložitosti funkcií sme boli schopný s HLS dosiahnuť uspokojivých výsledkov. Latencia vo všetkých funkciách dosahuje jednotky mikrosekúnd. Najdôležitejší indikátor Delta poskytuje spoľahlivé a presné hodnoty. Najväčší priestor na optimalizáciu predstavujú použité matematické funkcie. Tie sa síce vyznačujú vysokou presnosťou, no spotreba zdrojov má potenciál klesnúť. Za zváženie by stálo preskúmať aproximáciu funkcie `qphi`, ktorá je uskutočnená polynómom. V spotrebe zdrojov sú najhoršie funkcie P a C. Využívajú 11 druhov násobičiek, ktoré obsadzujú všetky alokované DSP bloky.

Kapitola 5

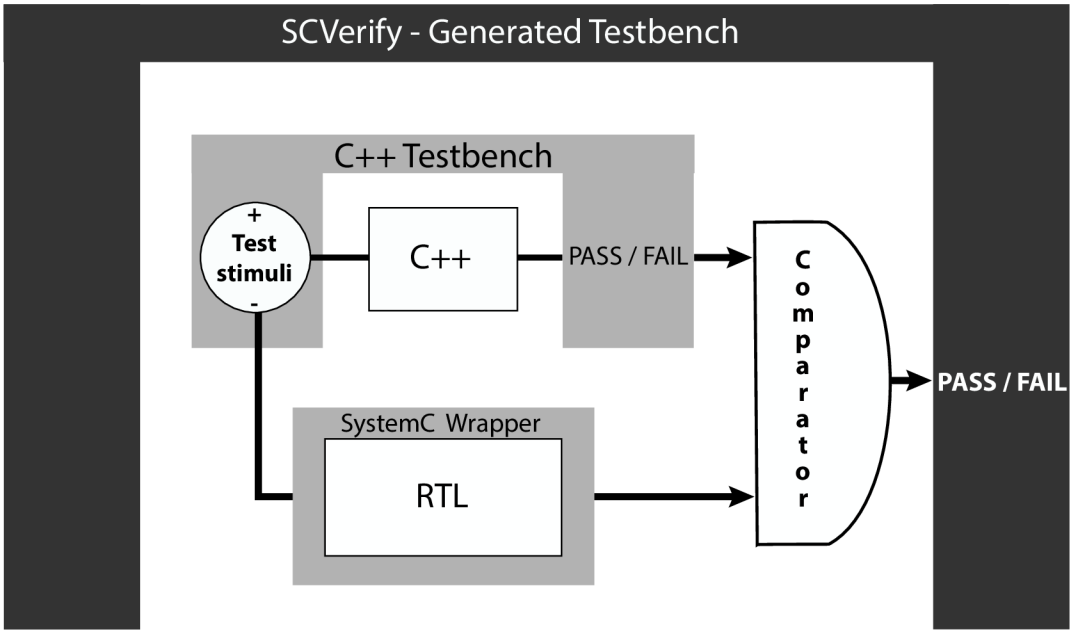
Overovanie funkčnosti

Overovanie funkčnosti ako aj verifikácia je veľmi dôležitou časťou pri vytváraní obvodov. Dôkladné testovanie môže zavčas odhaliť mnohé chyby. Funkčnosť dizajnov sa overuje v prvotných fázach vývoja prostredníctvom simulácií, kde sa odhalí väčšina bežných chýb. Po simuláciách nasleduje testovanie v hardware na konkrétnom čipe. Tam sa dizajn už len doladí, prípadne sa odstránia ďalšie menšie chyby. V rámci tejto práce je funkčnosť demonštrovaná prostredníctvom simulácií. Na verifikáciu a validáciu slúži v programe Catapult prostredie *SCVerify*. Pokiaľ je toto prostredie aktivované, všetky súbory potrebné pre simuláciu sú vytvorené počas generovania RTL popisu a samotné simulácie sú vykonávané v prostredí ModelSim.

Pre názorné pochopenie je spôsob validácie a verifikácie obvodov znázornený na obrázku 5.1. Program Catapult porovnáva výstupy RTL netlistu oproti originálnemu C/C++ kódu, za použitia manuálne vytvoreného testovacieho scenára (*testbench*) [5]. Prvým krokom je vytvorenie testovacieho scenára – *testbench*. Obvykle ide o cyklus v ktorom voláme funkciu v dizajne s rôznymi vstupmi a sledujeme výstupne hodnoty. Výsledky simulácie sú znázornené v časovom diagrame a ďalšie informácie nájdeme v samotnom prepise simulácie (*transcript*). V prípade neúspechu tam nájdeme aj podrobne informácie o chybách. Do prepisu sú uložené všetky textové reťazce poslané počas priebehu simulácie na štandardný výstup. Tento prepis ešte môže obsahovať informácie o latencii a priepustnosti, pokiaľ sa nezhoduje s odhadom programu Catapult. Hodnoty latencie môžu byť odlišné v prípade, že v dizajne máme cykly s vopred neurčeným počtom opakovaní. Simulačný scenár je popísaný jazykom C++ s použitím simulačnej knižnice `mc_scverify.h`. Podľa potreby importujeme ďalšie knižnice. Neplatia tu už žiadne obmedzenia, keďže tento súbor slúži na popis simulácie, nesyntetizuje sa.

Simulačný scenár je jediná časť ktorú dodáva programátor, ďalšie časti sú vygenerovaný HLS nástrojom, ktoré zabezpečia synchronizáciu a prepojenie všetky blokov. Počas testovania je ešte vhodné aktivovať signál nazývaný `TRANSACTION_DONE_SIGNAL`. Tento slúži na synchronizáciu transakcií a bez jeho aktivácie sa nám môže stať, že jednotlivé výstupy nebudú synchronizované v správnom poradí a simulácia bude aj napriek správnym výsledkom označená ako neúspešná. Takéto správanie môžeme očakávať pri používaní cyklov s vopred neurčeným počtom opakovaní. Počet iterácií v týchto cykloch je častokrát väčší ako program Catapult pri tvorbe simulácie očakával. `TRANSACTION_DONE_SIGNAL` signál by sa nemal vyskytovať vo finálnych dizajnoch kde zbytočne spotrebováva zdroje.

Všetky testovacie scenáre sú k dispozícii na priloženom CD nosiči. Každá funkcia ma svoj vlastný testovací scenár a TCL skript ktorý sa postará o načítanie zdrojových kódov, nastavenie parametrov, spustí plánovanie, generovanie RTL a nakoniec aj vlastnú simuláciu.



Obr. 5.1: Verifikácia C/C++ a RTL [5]

5.1 Testovanie matematických funkcií

Pri testovaní matematických funkcií máme tu výhodu, že výsledok získaný z dizajnu môžeme porovnať s výsledkom matematickej funkcie zo štandardnej knižnice. Funkcie zo štandardnej knižnice a ich výsledky budeme ďalej označovať ako referenčné. Do testovacích scenárov tak zavádzame ďalšie porovnávanie výsledkov. Okrem porovnania výsledkov získaných z architektúry a vlastnej implementácie v C++ zavádzame aj porovnanie výsledku z vygenerovanej architektúry a referenčnej hodnoty zo štandardnej knižnice. Preukážeme tak, že aproximačný algoritmus funguje korektne.

Matematické funkcie sú testované na väčšej vzorke vstupov. Vstupy sú volené náhodne z celého prípustného intervalu. Musíme brať do úvahy rozsah dátového typu `q16_16`, ktorý je vstupom pre všetky funkcie s výnimkou `fastlog2`. Okrem týchto náhodných čísel do testovaniu môžeme pridať aj iné špecifické hodnoty.

Pre každú funkciu je napísaný vlastný testovací scenár a v každej iterácií cyklu je zvolené nové náhodné číslo slúžiace ako vstup pre dizajn a referenčnú funkciu. Výstupné hodnoty z dizajnu a referenčnej funkcie porovnáme a z ich rozdielu dostaneme odchýlku aproximačného algoritmu. Treba si uvedomiť, že tieto dva výstupy nikdy nebudú zhodné a to z viacerých dôvodov. Vstupy pre dizajn a referenčnú funkciu sú na bitovej úrovni reprezentované odlišne a z ich odlišnej reprezentácie vyplýva aj ich odlišná presnosť. Ďalšie chyby vznikajú pri aritmetických operáciach, čo súvisy opäť súvisí s ich bitovou reprezentáciou. Koniec koncov ide o aproximačné algoritmy.

V každej iterácií je vyhodnocovaná odchýlka od referenčnej hodnoty a v prípade veľkého rozdielu je simulácia označená ako neúspešná. Na konci simulácie je užívateľovi poskytnutá maximálna, minimálna a priemerná odchýlka od referenčnej hodnoty zistená počas simulácie.

Vstupom pre funkciu `fastlog2` sú náhodné kladné celé číslo z intervalu $< 0, 2^{32} >$ čo predstavuje rozsah možných vstupov. Na konci simulácie je otestované číslo 0 a 1. Logarit-

mus nie je definovaný pre záporné hodnoty a zo zápornou hodnotou nepočíta ani dátový typ vstupného čísla `qint_u`.

Pre funkcie `qlog` a `qlog_mod` platí rovnaká definícia vstupných hodnôt. Ide o náhodné číslo z intervalu $< 0,0; 1000,0 >$. Vstupom tu môže byť aj záporná hodnota na ktorú je dizajn pripravený. Toto otestujeme na konci simulácie tak ako aj hodnoty 0 a 1. Odchýlka od referenčnej funkcie sa pohybuje v rozmedzí 0,00002 – 0,039864 (resp. 0,000023 – 0,091743 pri `qlog_mod`) a priemerne ide o hodnotu 0,000040 (resp. 0,004157 pre `qlog_mod`). V simuláciách sme zistili priepustnosť sa zmenila z 170 ns (resp. 180 ns pre `qlog_mod`) na 240 ns (resp. 320 ns).

Funkcia `qexp` tvorí odlišný typ. Vstupný interval je tu zvolený tak aby výsledok bol ešte v rozsahu dátového typu `q16_16`. Pri veľkých vstupných hodnotách by nám čísla začali pretekať, čo by malo za následok nekorektný výsledok. Ide o chybu vyplývajúcu z dátových typov a nie chybu algoritmu. Pri dodržaní vstupného intervalu $< -11; 10,35 >$ je aj výstup korektný a odchýlka od referenčnej funkcie sa pohybuje v rozmedzí 0,000011 – 1,293857 a priemerne ide o hodnotu 0,073001. Odchýlka dosahuje svoje maxima v krajných hodnotách vstupného intervalu.

Funkcia `qphi` má vstupný interval v celom rozsahu kladných a záporných hodnôt. Z jej matematickej definície nám vyplýva, že jej testovanie je zmysluplné v intervale vstupov $< -3,3 >$. V tomto intervale dostávame hodnoty použiteľné v dalšom výpočte . Tento interval sme použili aj pri testovaní funkcie a k nemu sme na koniec simulácie pridali aj pár náhodných čísel mimo jeho rozsah. Odchýlka od referenčnej funkcie sa pohybuje v rozmedzí 0,0 – 0,000046 a priemerne ide o hodnotu $1,3 \times 10^{-4}$. Latencia sa v simuláciách zvýšila z 53 cyklov na 229, čo ovplyvnilo aj priepustnosť a tá sa zmenila z 53 cyklov na 510 cyklov.

5.2 Testovanie technických indikátorov

Pri testovaní technických indikátorov je oveľa náročnejšie získať referenčný výsledok pre porovnanie. Jednou z možností je, že referenčné hodnoty vypočítame vopred a potom ich porovnáme s výsledkami z architektúry, čím overíme správnosť algoritmov. Ak chceme testovať a použiť iné náhodné hodnoty treba správnosť výsledku overovať dodatočne. Stále však platí že výsledky porovnávame medzi C++ a vygenerovanou architektúrou čo nám zaručuje, že získané z architektúry odpovedajú C++ kódu.

Pre testovanie a demonštráciu funkčnosti som v čo najväčšej miere použiť dáta z reálneho prostredia finančných trhov. Informácie o kurzoch akcií pochádzajú elektronického burzového trhu NASDAQ. Ide o niekoľko IT spoločnosti, s rôznou cenou akcií/opcií. Ďalšie parametre čerpané z reálneho sveta sa využívajú v Black – Scholes modeli. Ide o hodnotu volatility pre konkrétnu spoločnosť a hodnotu bezrizikovej úrokovej miery[1], ktorá je rovnaká pre všetky spoločnosti.

Funkčnosť je demonštrovaná na menšom počte iterácií. Testovací scenár pozostáva z dvoch menších cyklov. V prvom sú vstupy rôzne avšak nie náhodne a k dispozícií je aj referenčné pole výsledkov vypočítaných vopred. V druhom cykle sú vstupné hodnoty náhodné a tu môžeme vidieť reakciu obvodu na rôzne kombinácie. Každá funkcia má svoj vlastný testovací scenár, tak aby presne odpovedal testovaným funkciám. Maximálna, minimálna a priemerná odchýlka je zobrazená v prepise simulácie po ukončení všetkých iterácií.

Pre potreby testovania SMA funkcií boli získané ceny akcií 4 veľkých spoločností (Apple,

[1] hodnota Euribor® Rates – http://www.euribor-ebf.eu/ alebo U.S Treasury Bill Rates – http://www.treasury.gov/

Cisco systems, BASF a Intel) za roky 2013 a 2014 z NASDAQ. V prvej časti testovania je hodnota periódy, indexu, predchádzajúca hodnota SMA a nová cena akcií vopred stanovená. Konkrétne stanovené hodnoty sú uvedené v testovacích scenároch spolu s očakávanými výsledkami. Vždy aspoň v jednej iterácií sú chybné vstupné údaje a vrátenou hodnotou by mala byť 0. V ďalšej časti testovania je perióda a index volený náhodne z intervalu $< 0, 128 >$. Odchýlka hodnôt získaných z dizajnu od referenčných predstavuje v priemere $1,2 \times 10^{-4}$.

Funkčnosť oboch modifikácií exponenciálneho kĺzavého priemeru EMA bola demonštrované nasledovne. Referenčné vstupy a výstupy sú uložené v jednoduchom poli a boli vypočítané nezávisle. Výsledné hodnoty z dizajnu porovnáme s referenčnými a stanovíme odchýlku. V následnej časti už volíme vstupné hodnoty náhodne z rôznych intervalov. Odchýlka hodnôt získaných z dizajnu od referenčných predstavuje v priemere $1,2 \times 10^{-4}$.

Delta a všetky ďalšie funkcie, ktoré sú súčasťou Black – Scholes modelu používajú rovnaké rozhranie. Testovacie hodnoty tak môžu byť zhodné pre všetky funkcie. Bezriziková úroková miera predstavuje podľa U.S Treasury Bill k 01.05.2014 3%. Súčasný (9.5.2015) odhad volatility predstavuje pre Apple (AAPL – NASDAQ) 46,0% pre Google (GOOGL-NASDAQ) 26,4% pre Cisco (CSCO – NASDAQ) 28,7% a pre Intel (INTC – NASDAQ) je volatilita 22,1% [2]. Realizačnú cenu (*strike price*) a cenu podkladového aktíva (*spot price*) si buď určíme sami alebo znovu prevezmeme z trhu. To isté platí aj pre splatnosť opcie (*time to maturity*). Pre overenie funkčnosti tak máme dáta, ktoré pomerne presne vyjadrujú situáciu na trhu. Referenčné vstupné a výstupné hodnoty boli stanovené nezávisle na vlastnej implementácií a sú uložené v poli. Porovnaním výsledkov získaných z architektúry a referenčných výsledkov dostaneme odchýlku, ktorá je v priemere predstavuje v prípade delta $9,1 \times 10^{-5}$. Vo funkcií `C` má rovnako určená odchýlka hodnotu 0,00349 a vo funkcii `P` 0,00514.

[2] data pochádzajú z http://www.abg-analytics.com

Kapitola 6

Záver

Cieľom práce bolo implementovať vybrané funkcie z oblasti technickej analýzy finančných trhov. Funkcie boli HLS nástrojom transformované do popisu vhodného pre hardware. Počas vlastnej transformácie sme mali možnosť experimentovať s rôznymi parametrami syntézy, čo viedlo k preskúmaniu výkonnostných možností rady vygenerovaných architektúr. Podľa zadania sme sa mali sústrediť pri experimentovaní na nízku latenciu a minimálnu spotrebu zdrojov. Z výsledkov je badateľné, že na latenciu má výrazný vplyv miera použitej paralelizácie. Zreťazenie cyklov umožňuje efektívne využívanie zdrojov, no jeho aplikácia je niekedy obtiažna. Prevzaté algoritmy sme aj skúšali modifikovať s cieľom nižšej odozvy obvodu.

Pri tvorbe týchto indikátorov sme narazili na problém v podobe neexistencie syntetizovateľnej implementácie niekoľkých matematických funkcií. Práca sa teda rozšírila o tvorbu logaritmu s prirodzeným základom, exponenciálnej funkcie a distribučnej funkcie normálneho rozloženia. Tieto využívajú rôzne metódy aproximácie a ich implementácia by bez použitia HLS bola oveľa náročnejšia.

Správnosť výstupných hodnôt technických indikátorov a matematické funkcií bola preverená v simuláciách. Dosiahnute výsledky sa vyznačujú vysokou presnosťou (dosiahnutá odchýlka od referenčných hodnôt je $6,615 \times 10-3$), čo nás len môže utvrdzuje o ich vhodnosti pre ďalšie nasadenie. Po menšom prispôsobení by bolo možné všetky funkcie technických indikátorov zakomponovať do existujúcich FPGA kariet v podobe samostatných aplikácií. V budúcnosti sa môžeme zamerať na optimalizáciu a implementáciu dalších matematických funkcií a vytvoriť tak komplexnú knižnicu. Okrem nich by bolo možné rozšíriť dostupné riešenie o ďalšie technické indikátory a v spolupráci s investormi vypracovať kompletnú HFT obchodnú stratégiu.

Literatúra

[1] ABRAMOWITZ, M.; STEGUN, I. A. (editoři): *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables*. New York: Dover Publications, 1972, ISBN 0-486-61272-4, s. 296–299.

[2] ANDERSON, S. E.: Bit Twiddling Hacks [online]. 2009-01-05 [cit. 2014-04-20].
URL
http://graphics.stanford.edu/~seander/bithacks.html#IntegerLogLookup

[3] BLACK, F.; SCHOLES, M.: The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, ročník 81, č. 3, 1973: s. 637–654, ISSN 00223808.
URL http://www.jstor.org/stable/1831029

[4] Calypto Desing Systems: *Algorithmic C^{TM} Datatypes*. 2011.

[5] Calypto Desing Systems: *Catapult® C Synthesis User's and Reference Manual*. 2011.

[6] Calypto Desing Systems: *Catapult® Synthesis C++ to Hardware Concepts*. 2011.

[7] CHINNADURAI, M.: Survey On Scheduling And Allocation In High Level Synthesis. *International Journal of Computer Science*, ročník vol. 3, č. issue 5, 2012-10-31: s. 43–51, doi:10.5121/ijcses.2012.3503.
URL http://dx.doi.org/10.5121/ijcses.2012.3503

[8] CONG, J.; LIU, B.; NEUENDORFFER, S.; aj.: High-Level Synthesis for FPGAs: From Prototyping to Deployment. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, ročník 30, č. 4, April 2011: s. 473–491, ISSN 0278-0070, doi:10.1109/TCAD.2011.2110592.
URL http://dx.doi.org/10.1109/TCAD.2011.2110592

[9] COUSSY, P.; MORAWIEC, A. (editoři): *High-level synthesis: from algorithm to digital circuit*. Springer, 2008, ISBN 978-1-4020-8587-1, 297 s.

[10] DE MICHELI, G.; KU, D.; MAILHOT, F.; aj.: The Olympus synthesis system. *Design Test of Computers, IEEE*, ročník 7, č. 5, Oct 1990: s. 37–53, ISSN 0740-7475, doi:10.1109/54.60605.
URL http://dx.doi.org/10.1109/54.60605

[11] FINGEROFF, M.: *High-level synthesis: blue book*. United States: Mentor Graphics Corporation, 2010, ISBN 14-500-9723-5.

[12] FORENCICH, A.; RHEE, J.: Fixed Point Filtering Library [online]. 2010-11-29 [cit. 2014-04-20].
URL http://ieee.ucsd.edu/wiki/tutorials:fixed_point_filtering_library

[13] GIOVANNI, D. D.; ORTOBELLI, S.; RACHEV, S.: Delta hedging strategies comparison. *European Journal of Operational Research*, ročník 185, č. 3, 2008: s. 1615 – 1631, ISSN 0377-2217.
URL http://dx.doi.org/10.1016/j.ejor.2006.08.019

[14] HRICA, J.: Floating-Point Desing with Xilinx's Vivado HLS [online]. *Xcell Journal*, , č. 81, fourth quarter 2012 [cit. 2014-04-25]: s. 28–37.
URL http://www.xilinx.com/publications/archives/xcell/Xcell81.pdf

[15] HULL, J.: *Options, Futures and Other Derivatives*. Options, Futures and Other Derivatives, Pearson/Prentice Hall, 2009, ISBN 9780136015864, s. 349–380.

[16] LIAO, S.; TJIANG, S.; GUPTA, R.: An Efficient Implementation Of Reactivity For Modeling Hardware In The Scenic Design Environment. In *Design Automation Conference, 1997. Proceedings of the 34th*, June 1997, ISSN 0738-100X, s. 70–75, doi:10.1109/DAC.1997.597119.
URL http://dx.doi.org/10.1109/DAC.1997.597119

[17] MEEUS, W.; Van BEECK, K.; GOEDEMÉ, T.; aj.: An overview of today's high-level synthesis tools. *Design Automation for Embedded Systems*, ročník 16, č. 3, 2012: s. 31–51, ISSN 0929-5585, doi:10.1007/s10617-012-9096-8.
URL http://dx.doi.org/10.1007/s10617-012-9096-8

[18] METGHALCHI, M.; MARCUCCI, J.; YUNG-HO, C.: Are moving average trading rules profitable? Evidence from the European stock markets. *Applied Economics*, ročník 44, č. 12, 2012: s. 1539–1559, doi:10.1080/00036846.2010.543084.
URL http://dx.doi.org/10.1080/00036846.2010.543084

[19] POIGNÉ, A.; MORLEY, M.; MAFFEÏS, O.; aj.: The Synchronous Approach to Designing Reactive Systems. *Formal Methods in System Design*, ročník 12, č. 2, 1998: s. 163–187, ISSN 0925-9856, doi:10.1023/A:1008697810328.
URL http://dx.doi.org/10.1023/A%3A1008697810328

[20] REJNUŠ, O.: *Finanční trhy*. Ostrava: Key Publishing, třetí vydání, 2011, ISBN 978-80-7418-128-3.

Dodatok A

Obsah CD

`/doc` – technická správa a manuál
- `/latex` – zdrojové kódy technickej správy pre systém LaTeX
- `/xsinal00.pdf` – technická správa vo formáte pdf
- `/manual.pdf` – manuál pre úspešnú transformáciu zdrojových kódov a spustenie simulácií.

`/mathlib` – zdrojové kódy a testovacie scenáre pre implementované matematické funkcie
- `/mathlib.cc` – zdrojové kódy pre všetky implementované matematické funkcie.
- `/mathlib.h` – hlavičkový súbor matematických funkcií.
- `/tb_fastlog2.cc` – testovací scenár pre funkciu `fastlog2`.
- `/tb_qlog.cc` – testovací scenár pre logaritmickú funkciu `qlog`.
- `/tb_qlog_mod.cc` – testovací scenár pre modifikovaný logaritmus `qlog_mod`.
- `/tb_qexp.cc` – testovací scenár pre exponenciálnu funkciu `qexp`.
- `/tb_qphi.cc`– testovací scenár pre distribučnú funkciu normálneho rozdelenia `qphi`.

`/sma` – zdrojové kódy a testovacie scenáre pre výpočet jednoduchého kĺzavého priemeru.
- `/sma.cc` – zdrojové kódy funkcií jednoduchého kĺzavého priemeru.
- `/sma.h` – hlavičkový súbor pre funkcie jednoduchého kĺzavého priemeru.
- `/tb_sma.cc` – testovací scenár pre funkciu priemeru `sma`.
- `/tb_sma_mod.cc`– testovací scenár pre modifikovanú funkciu priemeru `sma_mod`.
- `/tb_sma_next.cc`– testovací scenár pre funkciu `sma_next`.
- `/tb_sma_next_mod.cc`– testovací scenár pre modifikovanú funkciu SMA `sma_next_mod`.
- `/tb.h` – spoločný hlavičkový súbor pre testovacie scenáre.

`/ema` – zdrojové kódy a testovacie scenáre pre výpočet exponenciálneho kĺzavého priemeru.
- `/ema.cc` – zdrojové kódy pre exponenciálne vážený priemer.
- `/ema.h` – hlavičkový súbor funkcie exponenciálneho váženého priemeru.
- `/tb_ema.cc` – testovací scenár pre funkciu `ema`.
- `/tb_ema_mod.cc` – testovací scenár pre modifikovanú funkciu priemeru `ema_mod`.

`/black_scholes` – zdrojové kódy a testovacie scenáre pre Black – Scholes model a delta.
- `/black_scholes.cc` – zdrojové kódy funkcií Black – Scholes modelu a indikátora delta.
- `/black_scholes.h` – hlavičkový súbor funkcií Black – Scholes modelu a indikátora delta.
- `/tb_delta_call.cc` – testovací scenár indikátora Delta pre kúpnu opciu. Funkcia `delta_call`.
- `/tb_delta_put.cc` – testovací scenár indikátora Delta pre predajú opciu. Funkcia `delta_put`.
- `/tb_p.cc` – testovací scenár pre výpočet kupnej ceny opcie. Funkcia `P`.
- `/tb_c.cc` – testovací scenár pre výpočet predajnej ceny opcie. Funkcia `C`.

`/tcl` – TCL skripty pre spustenie jednotlivých funkcií.
`/README.txt` – nápoveda k obsahu CD.

Dodatok B

Základné pojmy a definície

Pre lepšie pochopenie problematiky práce si vysvetlíme niekoľko pojmov a význam použitých skratiek z oblasti technickej analýzy [20] a syntézy obvodov.

- **Aktívum** – je majetok ktorý svojmu vlastníkovi prináša výnos. Aktíva môžu byť finančné (dlhopisy, akcie, penažné prostriedky ...) alebo reálne (budovy, pozemky stroje ...).
- **ASIC** – *Application Specific Integrated Circuit.* Integrovaný obvod prispôsobený špecifikám konkrétnej aplikácií.
- **Delta neutral** – Označuje portfólio zložené z pozícií, ktorých hodnota Delta je rovná nule.
- **DSP** – *Digital Signal Processing* je mikroprocesor, ktorého návrh je prispôsobený pre algoritmy spracujúce digitálne signály.
- **EMA** – *Exponential Moving Average.* Exponenciálny kĺzavý priemer.
- **FPGA** – *Field Programmable Gate Array.* Špeciálne číslicové integrované obvody zložené z programovateľných blokov prepojených konfigurovateľnou maticou spojov.
- **Greeks indikátory** – Nástroje na meranie citlivosti hodnoty opcie na rôzne faktory. Do *Greeks* patria: Delta, Gamma, Théta, Vega, Rhó.
- **HDL** – *Hardware Description Language.* Programovací jazyk na popis štruktúr, komponent a operácií v elektronických obvodoch.
- **Hedging** – je metóda na ochranu investícií pred rizikom strát. Princíp spočíva, že jednu pozíciu zaistíme druhou, otvorenou v opačnom smere.
- **HFT** – *High Frequency Trading.* Je obchodná stratégia, ktorá v krátkom čase zrealizuje veľké množstvo nákupov a predajov akcií alebo iných investícií. Profituje z malých cenových pohybov kurzov.
- **HLL** – *High Level programing Language.* Programovací jazyk vyznačujúci sa veľkou mierou abstrakcie voči detailom cieľového systému.
- **HLS** – *High – Level Synthesis* je automatizovaný proces, ktorý z algoritmického popisu správania vytvorí RTL popis hardware.

- **MACD** – *Moving Average Convergence Divergence*. Technický indikátor zo skupiny oscilátorov, ktoré merajú zmenu ceny za zvolené časové obdobie. Je daný rozdielom krátkodobého (peróda 12 dní) a dlhodobého (perióda 25/26 dní) kĺzavého priemeru.
- **NASDAQ** – *National Association of Securities Dealers Automated Quotations*. Najväčší rýdzo elektronický burzový trh v USA.
- **Opcia** – Zmluva, ktorá poskytuje držiteľovi opcie právo kúpiť alebo predať (v závislosti na type opcie) v dohodnutom okamihu či období predmetné podkladové aktívum za vopred stanovenú cenu.
- **RAM** – *Random Access Memory*. Pamäť umožnujúca náhodný prístup k hodotám v nej uložených.
- **RTL** – *Register – Transfer Level*. Jedna z úrovní popisu dizajnov. Dizajn obvodu je popísaný na úrovni registrov a dalších funkčných jednotiek (sčítačky, násobičky).
- **SMA** – *Simple Moving Average*. Jednoduchý kĺzavý priemer.
- **SoC** – *System on Chip*. Integrovaný obvod, kde všetky komponenty počítača sú umiestnené na jednom čipe.
- **TCL** – *Tool Command Language*. Skriptovací jazyk umožňujúci riadiť časť alebo celú aplikáciu.
- **Validácia** – overenie správnosti produktu (obvodu) vzhľadom k reálnym požiadavkám.
- **Verifikácia** – overenie správnosti produktu (obvodu) vzhľadom k formulovaným požiadavkám.
- **Verilog** – HDL programovací jazyk pre popis a modelovanie integrovaných obvodov.
- **VHDL** – *VHSIC Hardware Description Language*. HDL programovací jazyk na popis digitálnych integrovaných obvodov.
- **Volatilita** – je veličina popisujúca mieru kolísania hodnoty aktíva.