TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Bezztrátové komprese obrazu v embedded zařízeních

Bakalářská práce

Studijní program: B2612 Elektrotechnika a informatika
Studijní obor: Elektronické informační a řídicí systémy

Autor práce: **David Ulman**
Vedoucí práce: Ing. Martin Rozkovec, Ph.D.
Ústav informačních technologií a elektroniky

Konzultant práce: Ing. Jiří Jeníček, Ph.D.
Ústav informačních technologií a elektroniky

Liberec 2020

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Bezztrátové komprese obrazu v embedded zařízeních

Jméno a příjmení: **David Ulman**
Osobní číslo: M15000122
Studijní program: B2612 Elektrotechnika a informatika
Studijní obor: Elektronické informační a řídicí systémy
Zadávající katedra: Ústav informačních technologií a elektroniky
Akademický rok: **2019/2020**

Zásady pro vypracování:

1. Seznamte se základními a pokročilými metodami bezztrátové komprese obrazu. Dále se seznamte s platformou APSoC Zynq (Ultrascale+), vývojovou deskou ZCU106 a ZedBoard a balíkem vývojových nástrojů Xilinx Vivado.
2. Dle rešerše v jazyce C#/C/C++ na PC implementujte vhodné algoritmy komprese – enkodér a dekodér a porovnejte je.
3. Některé z algoritmů naprogramujte pro vnořená jádra v Zynq a porovnejte rychlost s PC.
4. Vyberte algoritmus vhodný pro implementaci v HW a v jazyce VHDL jej implementujte na hradlovém poli v APSoC. Výsledky porovnejte s předchozími měřeními.

Rozsah grafických prací: 5
Rozsah pracovní zprávy: 30-40
Forma zpracování práce: tištěná/elektronická
Jazyk práce: Čeština

Seznam odborné literatury:

[1] Pinker, J., Poupa, M., Číslicové systémy a jazyk VHDL, 2006, BEN – technická literatura, ISBN:8073001985
[2] Harris, D., Harris, S., Digital Design and Computer Architecture, 2nd Edition, 2012, Morgan Kaufmann, ISBN:9780123944245
[3] Khalid Sayood. Indtroduction to Data Compresssion, 5th Edition, 2017, Morgan Kaufmann, ISBN: 9780128094747

Vedoucí práce: Ing. Martin Rozkovec, Ph.D.
Ústav informačních technologií a elektroniky

Konzultant práce: Ing. Jiří Jeníček, Ph.D.
Ústav informačních technologií a elektroniky

Datum zadání práce: 9. října 2019
Předpokládaný termín odevzdání: 18. května 2020

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 18. října 2019

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 - školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

1. června 2020 David Ulman

Poděkování

Tímto bych rád poděkoval svému vedoucímu práce Ing. Martinu Rozkovci, Ph.D. za podnětné rady a konzultace při řešení a zpracování této práce.

Abstrakt

Tato bakalářská práce se zabývá bezztrátovými kompresními metodami digitálního obrazu a jejich implementací pro procesor osobních počítačů a pro platformu Zynq, která v sobě integruje procesor a programovatelný obvod typu hradlové pole. Práce mezi sebou některé metody nejdříve porovnává z pohledu jejich schopnosti komprimovat zvolené šedotónové obrazy a jeden konkrétní algoritmus si volí pro následnou implementaci do hradlového pole. Vlastní návrh obvodu vybraného algoritmu je stěžejní částí této práce a je proveden v prostředí Vivado v popisném jazyce VHDL a otestován na vývojové desce ZedBoard, která obsahuje platformu Zynq-7020.

Klíčová slova: bezztrátová komprese, digitální obraz, hradlová pole, Zynq, procesor, VHDL

Abstract

This Bachelor thesis deals with the Lossless compression methods of digital images and with their implementation for Personal Computer Processors and for Zynq platform, which integrates a processor and a programmable circuit FPGA. This work at first compares some for the methods from the standpoint of their ability to compress chosen grayscale images, and it chooses one specific algorithm for the following implementation into the FPGA. The design of the chosen algorithm is the crucial part of this work, and it was done in the Vivado environment in descriptive language VHDL. The circuit was tested on a ZedBoard development board, which includes the Zynq-7020 platform.

Keywords: lossless compression, digital image, FPGA, Zynq, processor, VHDL

Obsah

1. Úvod ... 12

2. Komprese obrazu ... 13

2.1 Komprese obecně ... 13

2.2 Digitální Obraz ... 14

2.2.1 Charakteristiky digitálního obrazu ... 15

2.2.2 Reprezentace digitálního obrazu ... 16

2.2.3 Specifika digitálního obrazu vhodná pro jeho kompresi ... 16

2.3 Ztrátová komprese obrazu ... 17

2.4 Bezztrátová komprese obrazu ... 17

2.4.1 RLE ... 18

2.4.2 Metody s nulovým znakem ... 19

2.4.3 Slovníkové algoritmy LZ ... 19

2.4.4 Huffmanovo kódovaní ... 20

2.4.5 Aritmetické kódovaní ... 21

2.4.6 FELICS ... 22

3. Programovatelné obvody FPGA ... 24

3.1 Programovatelný logický blok CLB ... 25

3.2 Návrh obvodů pro FPGA ... 25

3.2.1 VHDL ... 25

3.2.2 Vivado ... 26

3.3 Platforma Zynq ... 26

4. Kompresní algoritmy na procesoru ... 27

4.1 Zvolené metody ... 27

4.1.1 RLE ... 27

4.1.2 Huffmanovo kódování ... 27

4.1.3 FELICS ... 27

4.2 Dosažené výsledky ... 28

4.2.1 RLE1, RLE2 ... 28

4.2.2 HUFFMAN ... 29

4.2.3 FELICS1, FELICS2, DELTA ... 30

4.3 Výběr metody pro implementaci ... 32
4.3.1 FELICS3 ... 33

5. Návrh hardwarové implementace ... 34

5.1 Axis blok ... 34

5.2 IP jádro ... 35
5.2.1 ARBITER_A ... 37
5.2.2 ARBITER_B ... 38
5.2.3 ARBITER_C ... 39
5.2.4 SUMS_COUNTER ... 40
5.2.5 K_FINDER (A, B, C, D) ... 42
5.2.6 PRIORIT_ENCODER ... 43
5.2.7 METACODER ... 44
5.2.8 CODER ... 47
5.2.9 PRESS ... 48

5.3 Zhodnocení výsledků implementace ... 50

5.4 Návrh na úpravu ... 52

6. Závěr ... 53

Použitá literatura ... 54

Přílohy ... 56

Seznam obrázků

Obrázek 1: *RGB model původně virážované (sépiové) fotografie z roku 1858, na které je zachycen jeden z posledních Napoleonových granátníků. [25]* 14
Obrázek 2: *Elastograický snímek, který zobrazuje Youngův model pružnosti, kterým je zde indikována rakovina prostaty. [26]* 14
Obrázek 3: *Ukázka histogramu na šedotónovém portrétu prezidenta Miloše Zemana od Herberta Slavíka* 15
Obrázek 4: *Ukázka konstrukce Huffmanova stromu* 21
Obrázek 5: *Orientace algoritmus FELICS v bitové mapě* 22
Obrázek 6: *Model pravděpodobnostního rozložení algoritmu FELICS* 22
Obrázek 7: *Obecné schéma obvodu FPGA [20]* 24
Obrázek 8: *Zjednodušené schéma CLB [20]* 24
Obrázek 9: *Vývojová deska Zedboard se Zynq-7020 SoC [24]* 26
Obrázek 10: *Blokové zapojení IP jádra s DMA* 34
Obrázek 11: *Časový diagram AXI protokolu* 35
Obrázek 12: *Obecné schéma axis bloku* 35
Obrázek 13: *Axis bloky tvořící hlavní část IP jádra* 36
Obrázek 14: *Možné kombinace v bitové mapě* 37
Obrázek 15: *„Cik-Cak“ reprezentace hodnoty vzdálenosti od středu S* 39
Obrázek 16: *Princip řazení hodnot v axis bloku PRESS* 48

Seznam tabulek

Tabulka 1: *Ukázka kódových slov pro zkrácený binárního kód při u=3* 23
Tabulka 2: *Ukázka kódových slov pro Ricův kód* 23
Tabulka 3: *Průměrný kompresní zisk pro algoritmus RLE1 a RLE2* 29
Tabulka 4: *Průměrný kompresní zisk pro algoritmus HUFFMAN* 30
Tabulka 5: *Průměrný kompresní zisk pro algoritmy FELICS1, FELICS2 a DELTA* 32
Tabulka 6: *Průměrný kompresní zisk pro algoritmy FELICS2 a FELICS3* 33
Tabulka 7: *Entita hlavní části IP jádra* 36
Tabulka 8: *Entita axis bloku ARBITER_A* 37
Tabulka 9: *Entita axis bloku ARBITER_B* 38
Tabulka 10: *Entita axis bloku ARBITER_C* 40
Tabulka 11: *Entita axis bloku SUMS_COUNTER* 41
Tabulka 12: *Entita axis bloků K_FINDER (A, B, C, D)* 42

Tabulka 13: *Entita axis bloku PRIORIT_ENCODER* 43
Tabulka 14: *Entita axis bloku METACODER* 44
Tabulka 15: *Entita axis bloku CODER* 47
Tabulka 16: *Entita axis bloku PRESS* 49
Tabulka 17: *Syntézní odhad použitých zdrojů* 51
Tabulka 18: *Kompresní čas hw. implementace a některých algoritmů* 51

Seznam grafů

Graf 1: *Kompresní zisk algoritmu RLE pro část vzorků* 28
Graf 2: *Kompresní zisk algoritmu HUFFMAN pro sadu „běžné výjevy“* 29
Graf 3: *Kompresní zisk algoritmu HUFFMAN pro sadu „šum“* 30
Graf 4: *Kompresní zisky algoritmů FELICS1, FELICS2 a DELTA* 31
Graf 5: *Porovnání algoritmu HUFFMAN, DELTA a FELICS2* 32

Seznam blokových schémat

Blokové schéma 1: *Obvod pro axis blok ARBITER_A* 38
Blokové schéma 2: *Obvod pro axis blok ARBITER_B* 39
Blokové schéma 3: *Obvod pro axis blok ARBITER_C* 40
Blokové schéma 4: *Obvod pro axis blok SUMS_COUNTER* 41
Blokové schéma 5: *Obvod pro axis bloky K_FINDER (A, B, C, D)* 42
Blokové schéma 6: *Obvod pro axis blok PRIORIT_ENCODER* 43
Blokové schéma 7: *Obvod pro axis blok METACODER (zkrácený binární kód)* 45
Blokové schéma 8: *Obvod pro axis blok METACODER (Ricův kód)* 46
Blokové schéma 9: *Obvod pro axis blok CODER* 48
Blokové schéma 10: *Obvod pro axis blok PRESS* 50

Seznam zkratek

APSoC	plně programovatelný systém na čipu (All Programmable Systém on Chip)
ASCII	standardizovaná tabulka znaků (American Standard Code for Information Interchange)
ASIC	zákaznický integrovaný obvod (Application Specific Integrated Circuit)
AXI	typ komunikačního rozhraní (Advanced eXtensible Interface)
CCITT	komise pro telegrafii a telefonii (Consultative Committee for International Telephony and Telegraphy)
CLB	programovatelný logický blok (Configurable Logic Block)
CMOS	typ technologie logických obvodů (Complementary Metal Oxide Semiconductor)
DMA	metoda přímého přístupu do paměti (Direct Memory Access)
DVB-T	standart digitálního televizního vysílání (Digital Video Broadcasting–Terrestrial)
FFD	klopný obvod senzitivní na hranu (Flip-Flop D register)
FIFO	způsob manipulace s daty (First In First Out)
FPGA	programovatelné hradlové pole (Field Programmable Gate Array)
HDL	jazyk pro popis hardwarových struktur (Hardware Description Language)
HiRISE	polychromatická kamera pro sondu MRO (High Resolution Imaging Science Experiment)
IEEE	institut pro elektrotechnické a elektronické inženýrství (Institute of Electrical and Electronics Engineers)
IOB	vstupně výstupní blok (Input/Output Block)
JPEG	metoda ztrátové komprese (Joint Photographic Experts Group)
LUT	vyhledávací tabulka (Look-Up Table)
LZW	metoda bezztrátové komprese (Lempel Ziv Welch)
MPEG	kolekce metod pro kompresi od společnosti MPEG (Moving Picture Experts Group)
MRO	vesmírná sonda na orbitě planety Mars (Mars Reconnaissance Orbiter)
PLA	typ programovatelného obvodu (Programmable Logic Array)
RAM	polovodičová paměť s přímým přístupem (Random Access Memory)
RLE	metoda bezztrátové komprese (Run-Length Encoding)
ROM	nevolativní elektronická paměť (Read-Only Memory)
RTL	Abstrakce digitálního návrhu (Register Transfer Level)
TTL	Standart pro implementaci integrovaných digitálních obvodů (Transistor Transistor Logic)
VHDL	standardizovaná jazyk pro popis hardwaru (Very high speed integrated circuit HDL)

1. Úvod

Již od nástupu prvních komunikačních technologií je lidskou snahou úspora místa pro účely ukládání a přenosu informace. Za jeden z prvních výsledků tohoto snažení se například považuje Morseova abeceda z roku 1838, použitá v telegrafii, která umožnila kompresi běžné abecedy. Od té doby a zejména po zformulování teorie informace Claudem Shanonem, byla vytvořena řada dalších kompresních algoritmů, které nalézaly uplatnění pro konkrétní aplikace dané éry. Pro moderní svět zacházející s enormním množstvím dat ve všech odvětvích své činnosti, je potřeba komprese stále velmi aktuální a proto se stále vyvíjí nové algoritmy, ty ale velmi často vycházejí z již desítek let známých metod.

Jako prostředek pro realizaci algoritmů lze využít například procesor, ten je ale jen nejviditelnější použitelnou technologií nikoli ojedinělou. Lze využít takzvané zakázkové obvody ASIC, které realizují specifickou neměnnou logickou funkci danou výrobou. Takové zařízení je pak často pro vybranou aplikaci mnohem rychlejší než použití univerzálního procesoru. Podskupinou zakázkových obvodů jsou programovatelné obvody, které se dají uživatelem plně nakonfigurovat. Sem patří zejména od 80. let používaná programovatelná hradlová pole, známá pod zkratkou FPGA.

Hlavním cílem této práce je vlastní návrh hardwarové implementace algoritmu pro bezztrátovou kompresi digitálního obrazu do hradlového pole platformy APSoC Zynq. Pro výběr algoritmu poslouží první část práce, která se zabývá rešerší bezztrátových metod. Té předchází pojednání o tom, co pojem komprese znamená pro výpočetní techniku a co to vlastně je digitální obraz. V druhé části bude přiblížena funkce a práce s hradlovými poli a samotná platforma APSoC Zynq. Ve třetí části budou některé metody bezztrátové komprese naprogramovány a ozkoušeny na procesoru a následně porovnány. V poslední části bude popsáno a zhodnoceno samotné provedení hardwarového návrhu zvoleného algoritmu.

2. Komprese obrazu

2.1 Komprese obecně

Komprese nebo také komprimace je, ve výpočetní technice, proces zpracování informace, kterým se dosahuje změny objemu dat (obsahujících danou informaci), z původní velikosti L(X) na novou velikost O(X). Informaci v takto zpracovaných datech ovšem nelze přímo interpretovat a pro její správnou interpretaci musí být data nejdříve zpracována opačným procesem (dekomprese dat). Hlavní motivací ke kompresi je snížení potřebného úložného prostoru a urychlení datového toku. Z těchto důvodů plyne snaha o to, aby nová velikost O(X) byla co možná nejmenší oproti původní L(X). [8][10]

Z pohledu teorie informace [11] se jedná o snižování redundance obsažené v datech, čímž dochází k jejich efektivní reprezentaci. Je-li dosaženo maximální možné komprese, respektive minimální možné délky O(X) a to při současném zachování původní informace, mělo by se jednat o ideální stav. Toto tvrzení je nicméně omezeno jen na případ, kdy se jedná o takzvanou kompresi bezztrátovou, u níž se odstraňuje pouze statistická nadbytečnost a je tak možné komprimovaná data plně rekonstruovat. Komprese dat se tedy dělí na kompresi bezztrátovou a kompresi ztrátovou. Informace jako obraz či zvuk, jsou totiž často určeny lidským smyslům a ty do jisté míry tolerují zkreslení a nepřesnosti. Toho se využívá právě při takzvané ztrátové kompresi, při níž dochází k nenávratné ztrátě jisté části informace, která je ale nechtěná či irelevantní pro danou aplikaci. [8][10]

Dle [10] jsou kvantitativními veličinami, kterými lze zhodnotit efektivitu dané komprese:

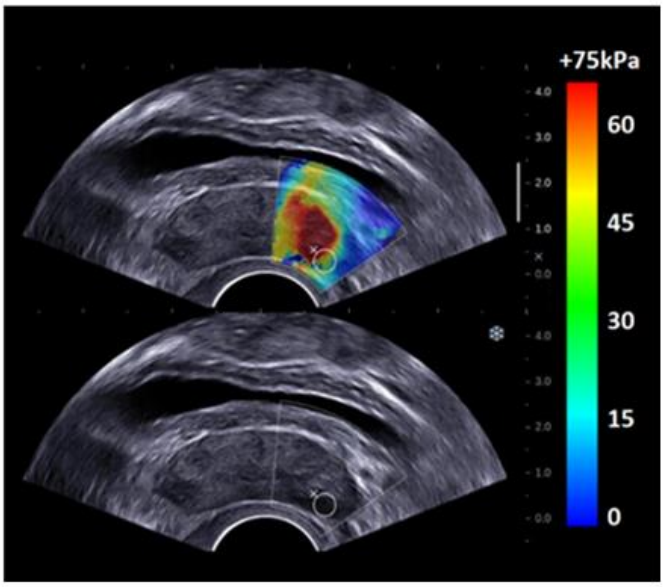
- kompresní poměr: L(X) / O(X)
- kompresní faktor: O(X) / L(X)
- kompresní zisk: (O(X) – L(X) / O(X))

Neméně důležitými kritérii je pak i náročnost na operační paměť, čas potřebný ke kompresi/dekompresi a veličina spíše kvalitativního charakteru, algoritmická náročnost.

2.2 Digitální Obraz

Obecný trojrozměrný obraz s rozměry x, y, z proměnný v čase t lze matematicky definovat pomocí takzvané obrazové funkce $f(x, y, z, t)=H$, kde hodnota obrazové funkce H je parametr popisující například barvu či jas [13]. Obraz v digitální podobě vznikne tak, že se tato spojitá obrazová funkce nejdříve navzorkuje a vzniklé vzorky se přiřadí k některé z možných diskrétních hodnot (kvantování). [9]

Pro digitalizovaný barevný obraz se v počítačové grafice velmi často jako parametr obrazové funkce volí vektorově reprezentovaná barevná složka modelu RGB (R-červená, G-zelená, B-modrá). Například pro dvourozměrný statický obraz bude $f(x, y)=[Hr, Hg, Hb]$. V případě šedotónového obrazu je zase často parametrem jas reprezentovaný nejčastěji na osmi bitech, tedy v 256 úrovních šedi. Nejedná se ale o jediné používané způsoby. Setkat se lze například i s barevným modelem CMYK (C-azurová, M-purpurová, Y-žlutá, K-černá) využívaným pro tisk, anebo s HSL (H-barevný tón, S-sytost, L-jas), který lépe postihuje funkci lidského oka. Parametrem může být obecně cokoli, v případě termokamery může jít o teplotu, u rentgenového tomografu zase o schopnost pohlcovat záření a například v elastografii, která na základě tuhosti tkání hledá patogeny v lidském těle, je parametrem Youngův modul pružnosti. [12]



Obrázek 1: RGB model původně virážované (sépiové) fotografie z roku 1858, na které je zachycen jeden z posledních Napoleonových granátníků. [25]

Obrázek 2: Elastograický snímek, který zobrazuje Youngův model pružnosti, kterým je zde indikována rakovina prostaty. [26]

2.2.1 Charakteristiky digitálního obrazu

Jas

Z fyzikálního hlediska je jas udáván jako měrná veličina svítivosti. Podobně je tomu i v počítačové grafice, kde je jím vyjádřena svítivost jednoho pixelu. Rozdíl je v tom, že jej nelze vyjádřit jako absolutní hodnotu v cd/m2. Může nabývat jen jakési maximální (bílá barva) a minimální (černá barva) hodnoty a diskrétního množství hodnot mezi tím [16]. Jas celého obrázku je udáván jako aritmetická střední hodnota jasu všech pixelů. [12]

Hloubka obrazu

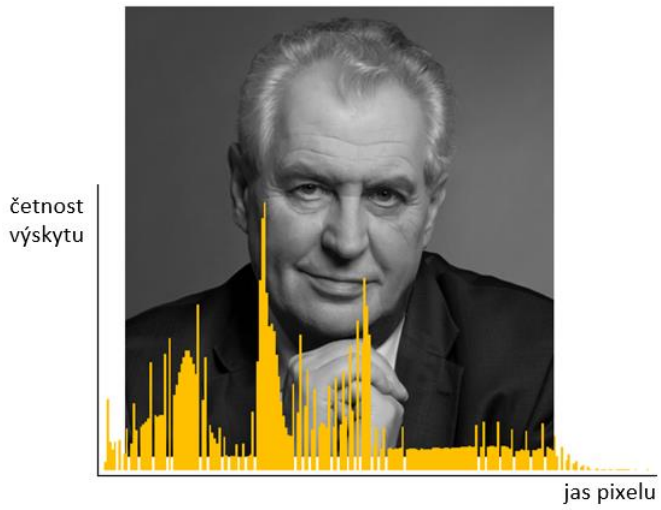
To kolik úrovní jasu bude existovat, udává bitová hloubka obrazu, která vymezuje počet bitů pro jeho reprezentaci. Například pro 8btiovou hloubku bude existovat 256 úrovní a pro 16bitovou hloubku 2^{16} úrovní. [12]

Dynamický rozsah

Ne v každém obraze se musí nutně vyskytovat úrovně jasu v celém rozsahu, jak jej udává bitová hloubka [13]. Tuto skutečnost postihuje dynamický rozsah, který se počítá jako dekadický logaritmus poměru největší a nejmenší hodnoty jasu. Obdobnou veličinou je kontrast, který může být udáván jako podíl dvou extrémů ve vybrané části obrazu. [12]

Histogram

Základní statistickou informaci o pravděpodobnostním rozložení jasu v obrazu poskytuje takzvaný histogram. Ten lze jednoduše graficky znázornit a jde o množinu všech úrovní jasu pixelů a četnosti jejich výskytů. [12]



Obrázek 3: Ukázka histogramu na šedotónovém portrétu prezidenta Miloše Zemana od Herberta Slavíka

2.2.2 Reprezentace digitálního obrazu

Rastrová grafika

Hlavním způsobem, jak reprezentovat digitální obraz je za pomoci bitové mapy. Obraz se reprezentuje matici pixelů, kde má každý pixel přesně danou pozici v prostoru, ve které nabývá hodnoty na základě zvoleného barevného modelu. Například ve výše zmiňovaném RGB modelu bude pro každý pixel existovat vektor [*Hr, Hg, Hb*], což lze vnímat i jako tři samostatné matice, ve kterých je zaznamenána hodnota jasu pro danou barvu. Bitmapová reprezentace je nejčastější pro fotografie a celkově pro všechny netriviální obrazy. [15][16]

Vektorová grafika

K popisu obrazu se využívá obecných geometrických útvarů jako jsou křivky, mnohoúhelníky a body, které lze přesně matematicky popsat. Obraz poskládaný z takto popsaných objektů je přesně definován a lze jej jakkoliv upravovat, aniž by při tom došlo k jeho deformaci či jiné ztrátě kvality. Křivka v takovém obrazu je definována například počátečním vektorem, koncovým bodem, kotevními body, parametry zakřivení, barvou atp. Pro vytváření těchto obrazů slouží speciální nástroje a výsledný obraz je daleko více náročný na výpočetní výkon než pouhá maticová reprezentace. Vektorová grafika slouží zejména pro zobrazování ikon, logotypů, nebo pro vytváření animací. [14][16]

2.2.3 Specifika digitálního obrazu vhodná pro jeho kompresi

Vyjma obrazů zachycujících šum nejsou skutečné obrazy pouhým dílem náhody a vyznačují se tak jistými sémantickými vlastnostmi, které lze využít pro jejich efektivnější kompresi. Hodnoty jasu, kterých pixely nabývají, nejsou v obrazu distribuovány rovnoměrně a velmi často se vyskytuje jen zlomek jejich možných hodnot (histogramy nemívají obdélníkové pravděpodobnostní rozložení). Mezi pixely se také vyskytují prostorové korelace, které jsou patrné jak na lokální, tak i globální úrovni (například mapy) a vytváří se tak shluky podobných/stejných hodnot. Tato statistická závislost existuje jak u jasu, tak i mezi jednotlivými barevnými kanály. Díky těmto skutečnostem lze používat vhodnější kódy, na základě modelů predikovat hodnoty následujících pixelů, odstraňovat informace statisticky nadbytečné a v neposlední řádě i informace, které nejsou relevantní z důvodu omezenosti lidského vnímání. [8]

2.3 Ztrátová komprese obrazu

Tato práce si neklade za cíl popsat existující algoritmy ztrátové komprese, nelze se ale alespoň nezmínit o tom, jakým způsobem toto samostatné odvětví k obrazové kompresi přistupuje. Zejména se využívá již zmiňované omezenosti lidského vnímání, kdy komprimovaný obraz neobsahuje kompletní informaci v takové podobě, jako originál, ale pozorovatel by při jeho vnímání neměl pociťovat žádný, nebo jen nepatrný rozdíl.

Lze se setkat s trojím přístupem. První přístup některé pixely trvale odstraňuje, přičemž počítá s jejich obnovou přes dokreslení (z anglického inpaiting), například využitím vlnkové transformace. Dalším možným přístupem je získat odpovídající matematický model obrazu a z hodnot okolních pixelů predikovat hodnotu pixelu, který má být zakódován. Posledním principem je použití kosinových, nebo vlnkový transformací, které se často kombinují i s metodami patřících do bezztrátové komprese. [8]

Třetí zmiňovány přístup se týká například známého algoritmu JPEG. Ten nejdříve obraz převede do barevného prostoru YCbCr (Y-světlost pixelů, Cb–modrý chromační komponent, Cr–červený chromační komponent), ve kterém provede barevné podvzorkování (oko je citlivější na jas, než na barvu). Na tento výsledný produkt aplikuje diskrétní kosinovou transformaci, tedy převedení do frekvenční oblasti. Na závěr se provádí komprese Aritmetickým, nebo Huffmanovým kódováním, což jsou zástupci komprese bezztrátové.

2.4 Bezztrátová komprese obrazu

Algoritmy, které spadají do této kapitoly nejsou často určeny pouze pro obrazovou kompresi. Některé z nich jsou totiž stejně (někdy i více) vhodné pro zvuk, nebo text. U bezztrátové komprese obrazu vychází algoritmy z předpokladu, že způsob uložení informace o obrazu v bitové mapě na specifickém počtu bitů, bude obsahovat velké množství redundance, kterou lze minimalizovat. S jistou mírou zjednodušení lze konstatovat, že k minimalizaci redundance existují dva hlavní přístupy. Tím prvním je zakódováním pixelů jako vztahu, k již známým datům. Tím druhým je snaha o optimalizací použitého kódu pro reprezentaci daného pixelu. Algoritmy, které dosahují dobrých výsledků pak často výhodným způsobem kombinují oba přístupy.

2.4.1 RLE

Metoda RLE je v češtině označována jako proudové kódování, nebo také bytový proud. Jde o bezztrátový kompresní algoritmus, který se od roku 1967 používal při kódování televizního vysílání. Donedávna byl například jako součást formátu MPEG-2 [30] používán Českou televizí v digitálním standartu DVB-T [31]. Algoritmus je jednoprůchodový, snadno implementovatelný, vyznačuje se velkou rychlostí a pokud se v komprimovaném obrazu často opakuje hodnota mezi sousedními pixely, pak jde i o efektivní způsob komprese obrazu s tímto rysem.

Princip algoritmu je v tom, že opakující se posloupnost stejných hodnot pixelů nahradí hodnotou reprezentující počet opakovaní daných hodnot a hodnotou samotnou. Z proudu příchozích pixelů se tedy vytvoří dvojce [počet opakování; hodnota]. [10]

Příklad:

- Vstupní proud: **AAAACGPPPPPP## (**proud pixelů jejichž hodnota je 8bitová a reprezentována znaky v ASCII tabulce)
- Výstupní proud: **4A1C1G6P2#** (čísla jsou pro ilustraci uvedena jako skutečná čísla nikoli jako znaky z ASCII tabulky)
- proud pixelů o velikosti 14 bytů se tak sníží na 10 bytů.

Pokud k opakování sousedních hodnot nedochází anebo se hodnoty opakují jen velmi málo, stává se metoda velmi neúčinnou a dochází k záporné kompresi. Výsledná velikost zkomprimovaného obrazu může být v krajním případě až dvojnásobná. RLE je tedy i algoritmem velmi silně závislým na charakteru vstupních dat. [10]

Existuje několik možností, jak tento algoritmus modifikovat tak, aby se stal vhodnějším pro vybranou aplikaci a vylepšil se tak jeho kompresní poměr. Dochází-li například u 8bitového obrazu k častému opakování, které se ale neblíží k maximální hodnotě 255, není vhodné používat pro reprezentaci počtu opakování 8 bitů, nýbrž počet, kterým se i tak zajistí že většina počtu opakování bude moci být na těchto bitech reprezentována. Je-li algoritmus použit i pro obrazy, jejichž povaha není pro metodu RLE vhodná, je dobré neopakující se hodnoty zapsat v původní podobě a jeden bit vyhradit jako identifikátor, který určuje, zda je v danou chvíli metoda RLE použita či nikoli. [10]

2.4.2 Metody s nulovým znakem

Tyto metody jsou zaměřeny na obrazy s velkým výskytem jedné zvolené hodnoty. Velmi často se jedná právě o nulu, která reprezentuje nějakou nevýznamnou informaci. Nevýznamnou informací u obrazu může být například bílé/průhledné pozadí.

Potlačení nul

Potlačení nul pracuje obdobně jako RLE, kdy se nulové znaky zakódují do dvojce [identifikátor; počet nul], ale ostatní hodnoty se zachovají v původní podobě. Jako identifikátor je vhodné použít právě zvolený nulový znak. [17]

Bitové mapy

Hodnoty pixelů v obrazu se nejdříve rozdělí na dva typy. První typ reprezentován bitem o hodnotě 0 udává, že se na příslušné pozici nachází nulový znak. Druhý typ reprezentovaný bitem o hodnotě 1 signalizuje, kde se vyskytuje znak nenulový. Takto vznikne bitová mapa, která nese informaci o pozici nenulových znaků. Výstupem je pak sekvence nenulových znaků a bitová mapa. Pro 8bitový obraz se metoda stane efektivní, pokud počet nulových znaků přesáhne 1/8 celkového množství znaků. [17]

2.4.3 Slovníkové algoritmy LZ

Samostatnou kapitolou bezztrátové komprese jsou takzvané slovníkové metody. Pod zkratkou LZ (iniciály příjmení původních autorů Lempel, Ziv) je zahrnut soubor několika si podobných metod tohoto typu. Patří sem původní algoritmus LZ77 a jeho následné modifikace LZ78, nebo LZW84, která se používala jako základ pro starší verze archivačních programů typu ZIP. Modernějšími modifikacemi jsou například DEFLATE, či z něj vycházející LZMA, která se dnes používá v archivačním programu 7-ZIP. [22][17]

Myšlenkou slovníkových metod je skládání příchozích znaků do řetězců, ze kterých se vytváří slovník. S obsahem slovníku jsou pak nově příchozí řetězce porovnávány a kódovaný jako odkazy do jeho odpovídající části.

Tvorba slovníku

Jednou z možností je nejdříve inicializovat slovník pro všechny možné hodnoty jednoho znaku. Podle této počáteční inicializace se bude kódovat první, druhý a pak i každý znak, pro který nebude ve slovníku nalezena odpovídající kombinace. Kombinace prvního

a druhého znaku tvoří první řetězec, který se uloží do slovníku. Pokud následující dvojce pixelů (3 a 4) nebude shodná, vytvoří se ve slovníku ze znaků 2 a 3 další řetězec. Pokud, se ale znaky 1,2 a 3,4 shodují, tak se na výstup zapíše odkaz na odpovídající část slovníku a současně se do slovníku zavede další řetězec, tentokrát ale vytvořený z trojce znaků 3,4 a 5. Obdobným způsobem se pokračuje i dále a záleží na tom jaká maximální délka řetězce slovníku je zvolena. [27]

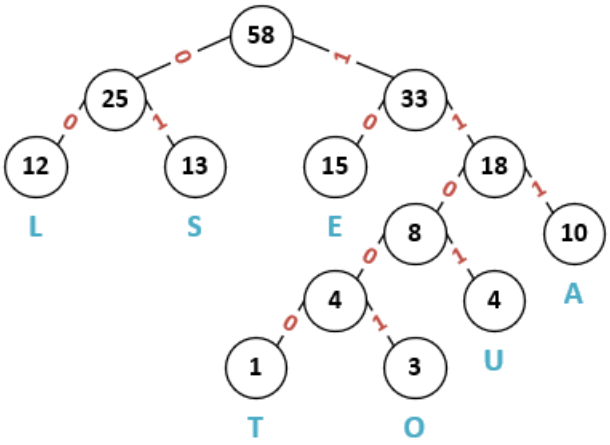
2.4.4 Huffmanovo kódovaní

Jde o entropické kódovaní, jehož princip byl například použit komisí CCITT pro tvorbu protokolu na faxování černobílých dokumentů [7]. Jeho nejefektivnější varianta (semiadaptivní kódování) pracuje na základě znalosti histogramu pro daný obraz. Z tohoto důvodu je metoda označována jako dvouprůchodová, kdy se nejdříve musí získat právě histogram a až poté lze obraz komprimovat. Metoda využívá faktu, že četnost výskytu pixelů není v obrazu prakticky nikdy rovnoměrná, a proto není vhodné pro jejich reprezentaci využívat kód se statickou délkou. Místo toho metoda hledá pro každou hodnotu optimálně dlouhý kód. Tedy pro často se opakující hodnoty pixelů bude použit kód s kratší délkou a pro pixely, které se vyskytují jen velmi málo zase kód delší. [4] [8]

Aby bylo možné kódovat hodnoty rozdílnými délkami, musí se zajistit, aby byl použitý kód takzvaně prefixový (jednoznačně dekódovatelný). Kód se proto vytváří podle Huffmanova stromu, který tuto skutečnost zajišťuje a zároveň se podle něj získá i minimální možný kód [10]. Obdobně například funguje i Shannon-Fanovo kódování, to ale nezaručuje optimální délku kódu, a tak se u něj dosahuje horších výsledků. [6]

Konstrukce Huffmanova stromu

Z histogramu se hodnoty pixelů (dále uvedeny jako znaky) seřadí, podle pravděpodobnosti jejich výskytu. Dvě nejnižší pravděpodobnosti se mezi sebou sečtou a výsledek se zařadí zpět na odpovídající pozici, přičemž bude odkazovat právě na hodnoty ze kterých byl vytvořen (s těmi se nadále již nijak nepočítá). Takto se postupuje do té doby, než zbydou jen dva členy, které odkazují na všechny předchozí hodnoty. Vzniklá struktura pak připomíná korunu stromu. Znaky jsou koncovými body a součty pravděpodobností které na ně, anebo již na vzniklé pravděpodobností odkazují, budou tvořit uzlové body. Prefixový kód pro pixel se vytvoří tak, že se postupuje od počátečního uzlu k danému znaku. Mezi jednotlivými body se podle zvolené konvence, přiřadí směr, kterým se ke znaku sestupuje hodnota '0' nebo '1'. [4]



Obrázek 4: Ukázka konstrukce Huffmanova stromu

Algoritmus lze modifikovat i tak, aby byl jednoprůchodový. První možností je binární strom vytvořit z histogramu podobných obrazů (statické kódovaní), tím se ale metoda stane závislou na přesnosti zvoleného modelu. Strom se může vytvářet i za chodu algoritmu (adaptivní kódovaní), v takovém případě je ale potřeba strom vždy podle potřeby přeuspořádat a již nelze zajistit, že se bude jednat o minimální možný kód pro celý obraz. [5]

2.4.5 Aritmetické kódovaní

Dalším případem tvorby entropického kódu je aritmetické kódování, které je ve většině ohledů lepší než Huffmanovo kódování. Stejně jako u Huffmanova kódování se nejdříve musí získat pravděpodobnostní rozdělení jednotlivých pixelů. Na jeho základě se ovšem nevytváří kód pro každý pixel zvlášť, nýbrž dojde k zakódování celého obrazu jednou hodnotou. Tato hodnota odpovídá reálnému číslu z intervalu ⟨0, 1), které obraz přesně popisuje a je získáno postupným zpřesňováním tohoto intervalu. [3]

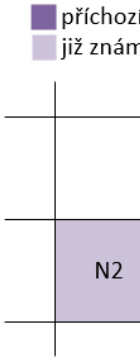
Zpřesňování intervalu

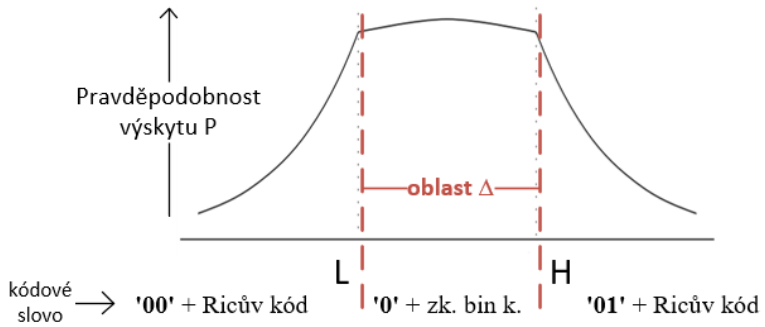
Interval se nejdříve rozdělí podle pravděpodobnosti výskytů na části, jejichž velikost je poměrná pravděpodobnosti výskytu příslušné hodnoty. Při samotné kompresi se ta část intervalu, která přísluší pixelu, který má být zakódován, rozdělí na stejné části jako původní interval. Tím se interval omezí z obou stran. Nové rozdělení se vždy stává základem pro další příchozí pixel. S rostoucím množstvím kódovaných pixelů se interval stále zmenšuje a pro jeho zápis je potřeba větší množství bitů. Více pravděpodobná hodnota interval zmenšuje méně a méně pravděpodobná hodnota více. [3]

2.4.6 FELICS

Anglický text zkratky FELICS lze do češtiny přeložit jako „rychlá a efektivní bezztrátová obrazová komprese“. Je to metoda, kterou v roce 1993 vymysleli pánové Paul G. Howard a Jerey Scott Vitter a byla například implementována do programovatelného hradlového pole pro polychromatickou kameru HiRISE [2]. HiRISE je jeden z vědeckých experimentů na palubě planetární sondy MRO, která obíhá a studuje planetu Mars. Metoda je jednoprůchodová, její kompresní poměr je srovnatelný s bezztrátovou verzí JPEG (se kterou ji ve své práci [1] porovnávají autoři), přičemž je ale až pětinásobně rychlejší.

Princip metody spočívá v zakódování každého nového pixelu P za použití jeho dvou již zakódovaných nejbližších sousedů N1 a N2. Sousední pixel s větší hodnotou je označen jako H a soused s nižší hodnotou jako L. Oblast mezi hodnotami pixelů H a L se nazývá Δ a pro příchozí pixel P se v ní odhaduje pravděpodobnost výskytu na 50 %. Je-li pixel P v dané oblasti Δ či mimo ní, bude v kódovém slovu indikováno jedním bitem ('1'–v oblasti, '0'–mimo oblast). V případě, že je pixel P mimo oblast Δ, použije se v kódovém slově ještě jeden bit, kterým se indikuje, zda je menší či větší. Podle toho, do jaké oblasti pixel P skutečně spadá, bude jeho hodnota zakódovaná jako vzdálenost od pixelu L (pixel P je menší než L), nebo od pixelu H (pixel P je vetší než H), nebo od středové hodnoty oblasti Δ (pixel P je mezi H a L, nebo je roven právě H, či L). Protože pravděpodobnostní rozložení, se kterým metoda pracuje, je v oblasti Δ téměř obdélníkové (pravděpodobnost u středu je mírně vyšší než u krajů), použije se na tuto oblast zkrácený binární kód. Rozložení pravděpodobnosti mimo oblast už má ale geometrický charakter, a tak se použije exponenciální Ricovo kódování. [1]





Obrázek 5: Orientace algoritmus FELICS v bitové mapě

Obrázek 6: Model pravděpodobnostního rozložení algoritmu FELICS

Zkrácené binární kódování

Jde o způsob kódování, které generuje pro zvolený rozsah *h* možných hodnot optimálně dlouhý prefixový kód. Jedná se o modifikaci binárního kódování, se kterým se plně shoduje, pokud platí, že $\log_2(h)$ je celé číslo. Pokud logaritmus není celé číslo, odečte se od sebe mocnina o základu dvou z jeho horní celé časti (2^ [$\log_2$(h)]) a hodnota *h*. Rozdíl se nazve jako *u*. Pro počet bitů rovných hodnotě horní celé části logaritmu, se vytvoří všechny možnosti binárního kódu, ze kterých se pro zkrácený binární kód použije prvních *u* hodnot, které se ale oříznou o nevýznamnější bit. Následujících *u* hodnot se přeskočí a zbytek hodnot se použije v nezměněné podobě. [1][32]

Tabulka 1: Ukázka kódových slov pro zkrácený binárního kód při u=3

binární kódování	n =	Kód	zkrácené binární k.	n =	kód
	0	000		0	~~0~~00
	1	001		1	~~0~~01
	2	010		2	~~0~~10
	3	011		-	~~011~~
	4	100		-	~~100~~
	5	101		-	~~101~~
	6	110		3	110
	7	111		4	111

Ricovo kódování

Ricovo kódování vytváří prefixový kód, který je vhodný zejména tam, kde je pravděpodobnost malých hodnot daleko vyšší než těch velkých. Metoda pracuje s parametrem m, pro který platí že $m=2^k$, kde *k* je libovolné kladné číslo, které je potřeba zvolit. Hodnota *n*, která má být zakódována se vydělí parametrem *m*. Celá část po dělení *q* se zakóduje unárně (posloupností odpovídajícího počtu jedniček a oddělující nulou) a bude tak tvořit první část kódu (unární část). Druhá část kódu (binární část) je binární hodnota, zbytku po dělení *r*, pro kterou bude potřeba *k* bitů. [1]

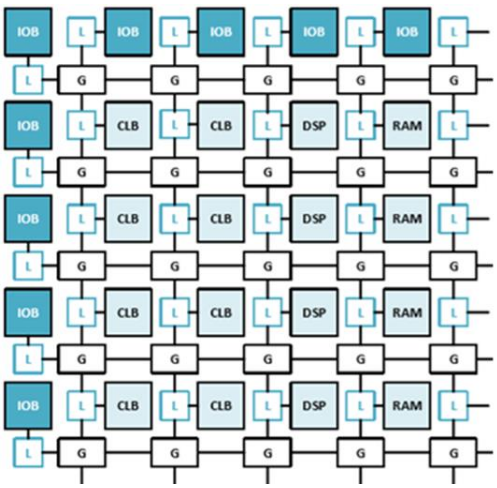
Tabulka 2: Ukázka kódových slov pro Ricův kód

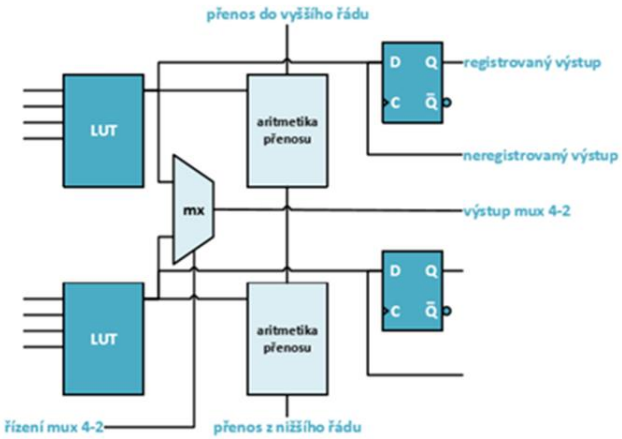
n =	m=1 k=0	m=2 k=1	m=4 k=2	m=8 k=3
0	0	0 ● 0	0 ● 00	0 ● 000
1	10	0 ● 1	0 ● 01	0 ● 001
2	110	10 ● 0	0 ● 10	0 ● 010
3	1110	10 ● 1	0 ● 11	0 ● 011
4	11110	110 ● 0	10 ● 00	0 ● 100
5	111110	110 ● 1	10 ● 01	0 ● 101
6	1111110	1110 ● 0	10 ● 10	0 ● 110
7	11111110	1110 ● 1	10 ● 11	0 ● 111

3. Programovatelné obvody FPGA

Obvody FPGA v češtině označované jako programovatelná hradlová pole, jsou součástky, které patří mezi nejuniverzálnější číslicově programovatelné obvody. Jednodušší programovatelné obvody, jako jsou například PLA, se sestávají ze vstupů, výstupů a matice programovatelných hradel AND a OR, jejichž vstupy lze volit pomocí programovatelných spínačů a lze tak realizovat libovolnou logickou funkci. Hradlová pole mohou realizovat jak jednotlivé logické funkce, tak složité soubory funkcí. [20][29]

FPGA mají vstupně výstupní obvody IOB, které obsahují registr, budič a multiplexor a splňují napěťové standardy jako jsou CMOS a TTL. Hradlová pole, realizují logické operace soustavou programovatelných logických bloků CLB, které mohou být libovolně propojeny, jak mezi sebou, tak se vstupně výstupními obvody. Propojení je provedeno propojovací maticí, kterou ovládají globální (G) a lokální (P) přepínače. Nejdůležitějšími parametry FPGA jsou počet logických a vstupně výstupních bloků, počet registrů a frekvence se kterou jsou jednotlivé bloky schopné pracovat. S FPGA jsou často integrovány i prvky jako paměť RAM, násobičky, řadiče externích pamětí či mikroprocesory. Jejich funkce by si totiž vyžádala příliš mnoho zdrojů, pokud by byla realizována CLB bloky. [20]





Obrázek 7: Obecné schéma obvodu FPGA [20]

Obrázek 8: Zjednodušené schéma CLB [20]

3.1 Programovatelný logický blok CLB

Blok CLB může vykonávat jakoukoli logickou funkci. V nejjednodušší variantě se skládá z vyhledávací tabulky LUT, přepínače a registru. Vyhledávací tabulka je realizována například jako paměť typu ROM s *n* bitovou adresou, na kterou vedou vstupní hodnoty logického bloku. Obsahem paměti ROM, který je dán uživatelským návrhem, je definována funkce, kterou daný blok vykonává. Podle počtu výstupů *m* může LUT realizovat právě *m* funkcí, které sdílejí logické proměnné. Přepínačem se zajišťuje, že mezi sebou lze kombinovat více LUT. Výstupní registr může reagovat na úroveň (latch), ale častěji je senzitivní na hranu (flip-flop) a jde nejčastěji o registr typu D (FFD). Logické bloky se v praxi často spojují do vyšších celků, ve kterých spolu s dodatečnou logikou uskutečňují funkci sčítaček, multiplexorů a dalších často používaných funkcí. [20]

3.2 Návrh obvodů pro FPGA

Jednoduší obvody se dají navrhnout popisem schématu pomocí základních obvodových prvků. Pro složitější funkce se nejčastěji používá popisných jazyků pro hardwarové struktury HDL, jako je například Verilog založený na syntaxu jazyku C, nebo v Evropě rozšířenější VHDL. Vytvořený návrh se zpracovává syntézou, která vytvoří takzvaný netlist. Netlist je orientovaný graf tvořený základními prvky, kterým se říká primitiva (LUT, FFD aj.) a jejich vzájemného propojení. Podle netlistu se provede implementace, ta pro danou technologii primitiva namapuje, umístí a propojí. Do hradlového pole je výstup implementace (bitstream) nahrán speciálním programátorem. [20]

3.2.1 VHDL

VHDL je volně přístupný, otevřený jazyk pro popis hardwaru, založený na syntaxi programovacího jazyku ADA (podobný jako Pascal). Byl vyvinut ministerstvem obrany USA za účelem sjednocení popisu zákaznických integrovaných obvodů ASIC a v roce 1987 byl standardizován jako IEEE 1076-1987 [19]. Jeho nejnovější revize je z roku 2011. VHDL je paralelní jazyk, který je typově velmi striktně orientovaný. Jazykem se popisuje jednak struktura obvodu, tedy vstupy a výstupy (popis entity), tak chování obvodu (popis architektury). [18] [21]

3.2.2 Vivado

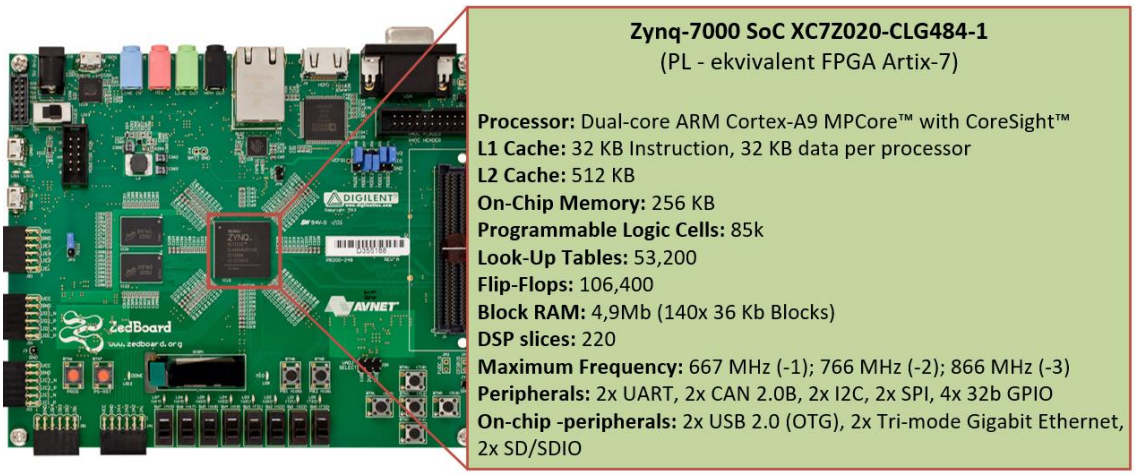
Viavado je vývojové prostředí od společnosti Xilinx, které bylo zveřejněno v roce 2012. Prostředí umožňuje návrh obvodů jak pomocí jazyku VHDL, tak Verilog. V prostředí lze provádět simulaci návrhu, abstrakci popsaného obvodu (RTL analýzu) a lze pomocí něj provést kompletní zpracování návrhu od syntézy až po nahrání do vývojové desky. [28]

3.3 Platforma Zynq

Jde o platformu od společnosti Xilinx, ve které se na jednom čipu integruje FPGA a procesor. Architektura je dána dvěma hlavními částmi. Částí procesorového systému PS a částí programovatelné logiky PL. Obě části lze využít tak aby pracovali společně, nebo zvlášť. [23]

U rodiny Zynq-7000 je část PS tvořena dvoujádrovým procesorem ARM Cortex-A9 a jde o takzvaně „tvrdý“ procesor. Procesorovou funkci může obstarávat ještě „měkký“ procesor MicroBlaze, který je stvořen ze základních prvků v části PL. Protože MicroBlaze je instance jádra, tak lze v části PL vygenerovat i více těchto „měkkých“ procesorů. Část PL je založena na FPGA Artix-7 / Kintex-7. Jeden CLB se skládá ze dvou podbloků (slice) z nichž každý obsahuje čtyři LUT a osm FFD registrů. [23]

Platformu lze zakoupit samostatně v různých variantách, nebo již osazenou ve vývojové desce. K dispozici jsou desky přímo od společnosti Xilinx, nebo například deska ZedBoard od firmy Digilent, která je cílovým zařízením této práce.



Obrázek 9: Vývojová deska Zedboard se Zynq-7020 SoC [24]

4. Kompresní algoritmy na procesoru

4.1 Zvolené metody

Pro účely práce byly naprogramovány v jazyce C kodéry a dekodéry několika vybraných kompresních algoritmů. Kodéry posloužili pro výběr vhodného algoritmu pro následnou hardwarovou implementaci a na dekodérech se ověřila funkčnost naprogramovaných kompresních algoritmů. Zvoleny byli následující algoritmy.

4.1.1 RLE

RLE algoritmus se jevil jako ideální učebnicový příklad. Byl naprogramován ve dvou verzích, kdy první verze (RLE1) nepoužívá příznakový bit pro indikaci opakování, druhá verze (RLE2), již tento bit využívá.

4.1.2 Huffmanovo kódování

Huffmanovo kódování bylo naprogramováno v jeho semiadaptivní verzi a její pracovní název je HUFFMAN. Algoritmus byl vybrán, protože se u něj předpokládalo, že dosahuje konzistentně dobrých výsledků.

4.1.3 FELICS

Algoritmus FELICS byl od začátku brán jako hlavní kandidát pro implementaci a byl naprogramován v několika verzích. Základní rozdíl mezi dvěma hlavními verzemi je způsob, jakým se volí parametr k potřebný právě pro tvorbu Ricova kódu. V první verzi (FELICS1) je parametr zvolen pevně. V druhé verzi (FELICS2) je tento parametr adaptivní a volí se podle vzdálenosti Δ mezi sousedními pixely H a L (Δ=H-L). Předpokládá se totiž že vhodnost k pro zakódování příchozího pixelu koreluje se vzdáleností mezi jeho dvěma sousedy [1]. Adaptivní verze funguje tak, že pro každou možnou Δ se nejprve zvolí libovolný výchozí parametr k. S každým příchozím pixelem si algoritmus pro odpovídající Δ zpětně dopočítává potenciální velikosti kódu pro daný případ, a to samostatně pro všechny varianty k. Tyto velikosti si kumulativně ukládá a ve chvíli, kdy má opět zakódovat pixel s odpovídající vzdáleností Δ, použije právě to k, které mělo do této chvíle pro stejnou Δ nejmenší kumulativní hodnotu.

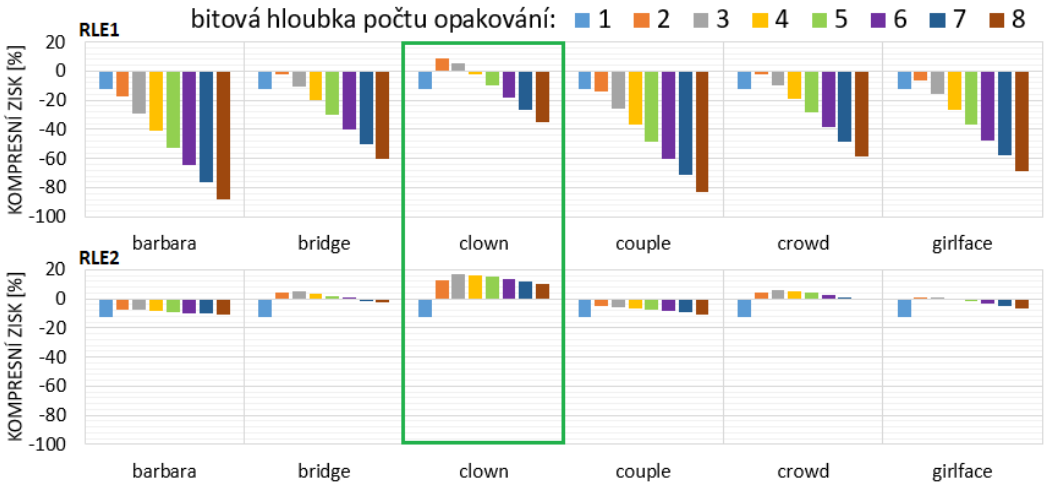
Další variantou je algoritmus dále nazývaný jako DELTA. Ten vznikl jako vlastní nápad, kdy záměrem bylo nějakým způsobem zakódovat příchozí pixel jako rozdíl od předešlé hodnoty. Postupným vylepšováním se ale z této metody de facto stal právě FELICS algoritmus, který ale používá binární kód s proměnou délkou.

4.2 Dosažené výsledky

Pro testování byli vytvořeny dvě sady dvanácti šedotónových obrazů s bitovou hloubkou 8 bitů (256 úrovní šedi) a s rozlišením 512x512. První sada obrazů dostala název „běžné výjevy“ a zachycovala objekty, postavy a prostory. Každý vzorek má svůj jednoslovný název (Příloha A). Obrazy v druhé sadě pak byly vybrány tak aby zachycovaly nahodilejší jevy a mezi jednotlivými pixely tudíž neměly tak velkou závislost jako tomu je u první sady, díky čemuž by měly být náročnější pro kompresi. Tato sada je označena jako „šum“ a vzorky jsou označeny písmeny abecedy od „a“ do „l“ (Příloha B).

4.2.1 RLE1, RLE2

Charakter vzorků v obou sadách nebyl pro tuto metodu výhodný. Pro obě naprogramované verze bylo vyzkoušeno, jaké důsledky na kompresní zisk, bude mít změna počtu bitů pro hodnotu reprezentující počet opakování. Verze bez příznakového bitu dosáhla téměř ve všech případech silné záporné komprese. Verze s příznakovým bitem už dopadla o něco lépe. Nevhodnost zvolených obrazů již nebyla tak znát, a pro obraz „clown“ byl dokonce v nejlepší konfiguraci kompresní zisk 16,8 %. Vzorky představující šum nedopadly dobře v žádné z konfigurací. Metoda se pro svou neefektivnost ukázala jako nevhodná pro implementaci.



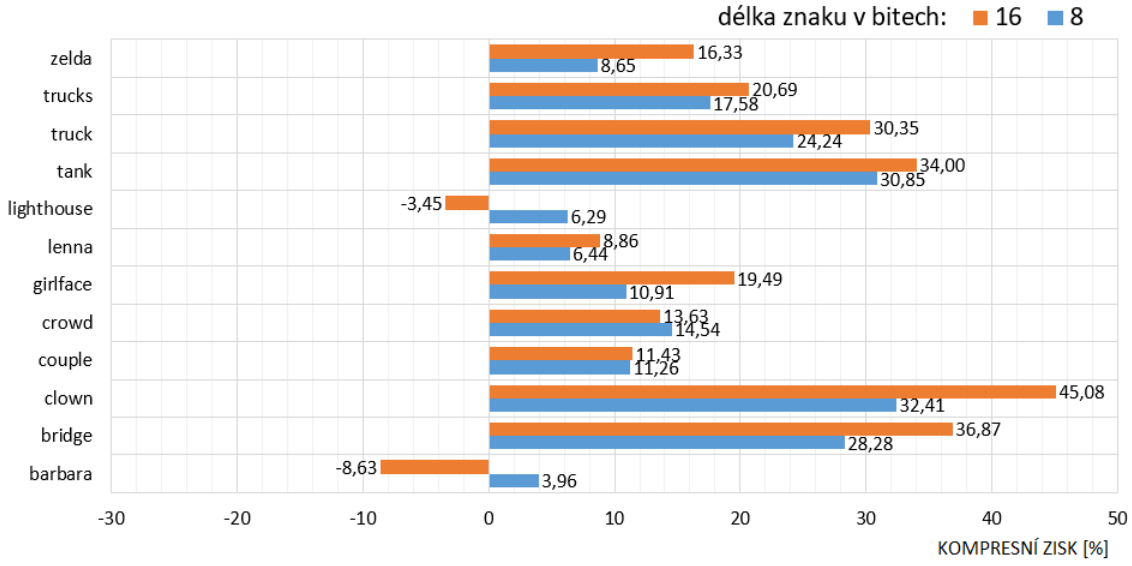
Graf 1: Kompresní zisk algoritmu RLE pro část vzorků

Tabulka 3: Průměrný kompresní zisk pro algoritmus RLE1 a RLE2

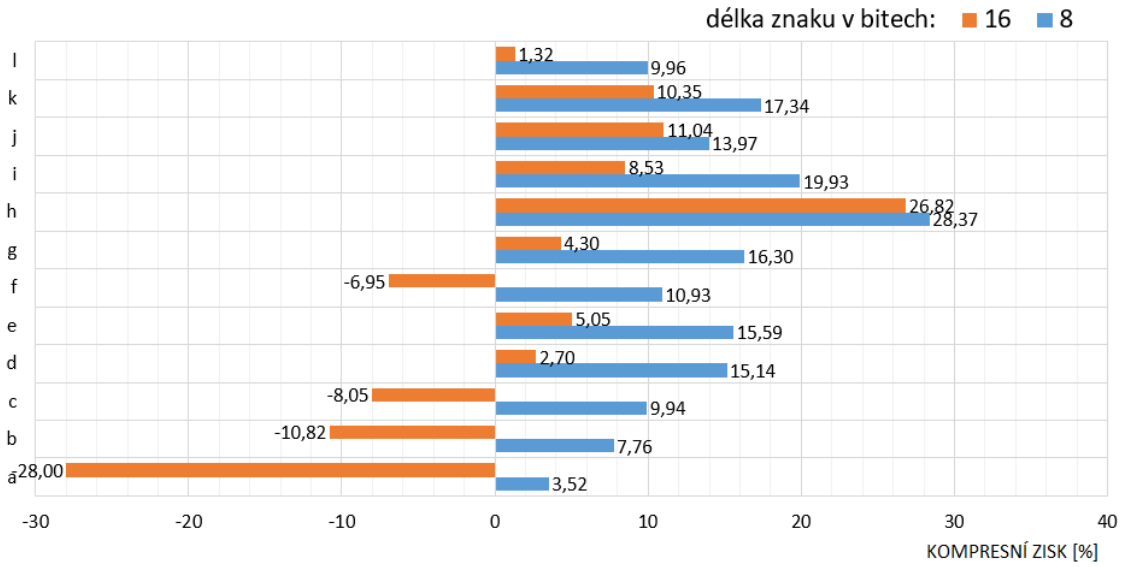
bitová hloubka počtu opakování:	2 bity	3 bity	4 bity	5 bitů	6 bitů	7 bitů	8 bitů	
průměrné výsledky sady obrazů: „běžné výjevy“								
kompresní zisk [%]	-9,39	-19,09	-29,71	-40,46	-51,24	-62,04	-72,84	RLE1
	-0,96	-1,01	-1,98	-3,09	-4,25	-5,41	-6,58	RLE2
průměrné výsledky sady obrazů: „šum“								
kompresní zisk [%]	-21,04	-33,02	-45,07	-57,14	-69,21	-81,29	-93,38	RLE1
	-9,66	-9,90	-10,20	-10,53	-10,86	-11,21	-11,55	RLE2

4.2.2 HUFFMAN

Huffmanovým kódováním bylo v první sadě obrazů dosaženo průměrného kompresního zisku 16,3 % a v druhé sadě 14,1 %. Zajímavých výsledků bylo dosaženo, když se experimentálně za znaky binárního stromu nepoužil jen jeden pixel (znak=8 bitů), nýbrž dvojce pixelů (znak=16 bitů). V takovou chvíli se u první sady obrazů dosáhlo výsledků o několik procent lepších, a to v průměru 18,7 %. Pro 16bitové znaky bylo ovšem potřeba ukládat rozsáhlejší histogramy, a tak se u několika obrazů zisk ve výsledku zhoršil. Pro druhou sadu obrazů, která nemá mezi pixely tak výraznou korelaci se zisk zhoršil výrazně, a to na pouhých 1,4 %. Výsledky této metody byly uspokojivé, protože se ale jedná o algoritmus dvouprůchodový, tak není vhodný pro hardwarovou implementaci.



Graf 2: Kompresní zisk algoritmu HUFFMAN pro sadu „běžné výjevy“



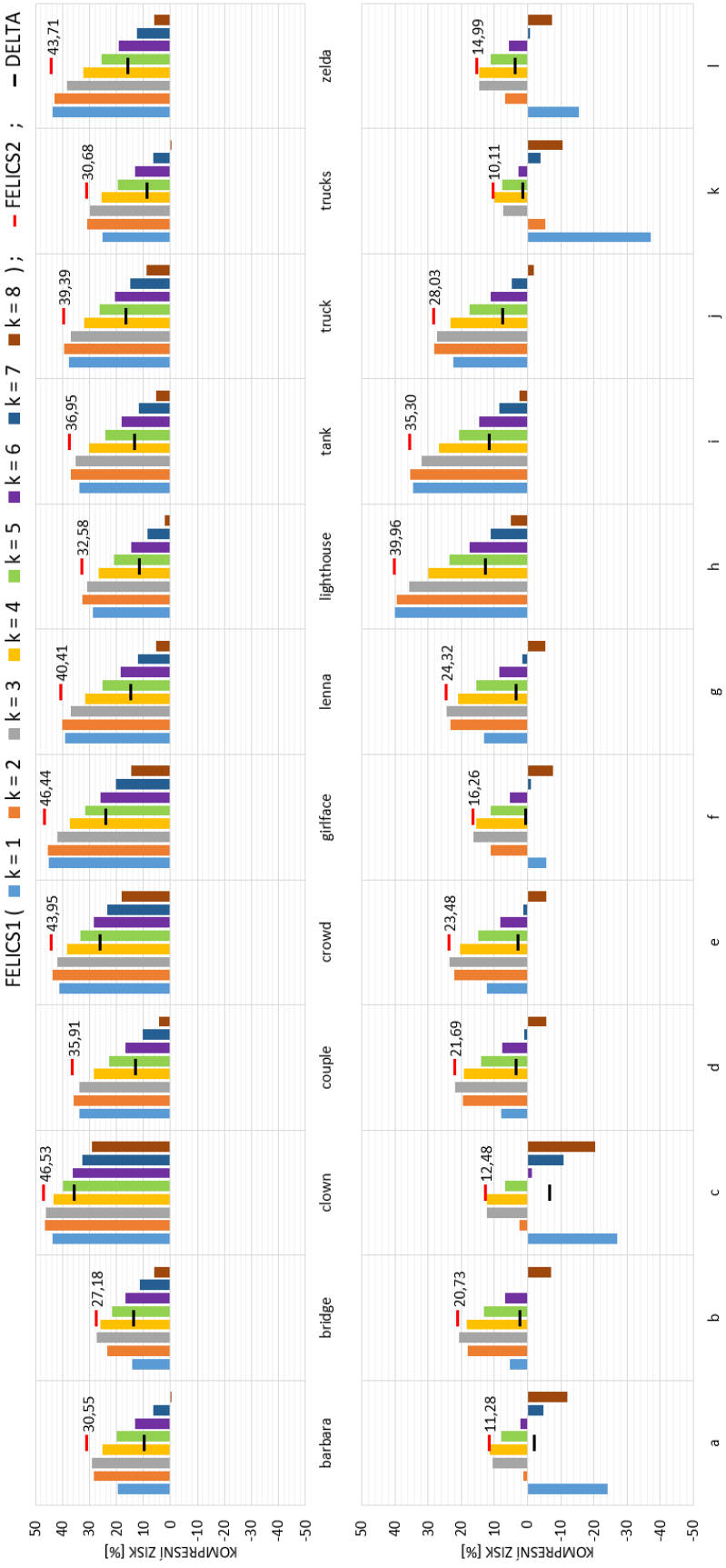
Graf 3: Kompresní zisk algoritmu HUFFMAN pro sadu „šum“

Tabulka 4: Průměrný kompresní zisk pro algoritmus HUFFMAN

bitová hloubka znaků v binárním stromu:		8 bitů	16 bitů
průměrné výsledky sady obrazů: „běžné výjevy“	Kompresní zisk [%]	16,28	18,72
průměrné výsledky sady obrazů: „šum“		14,06	1,36

4.2.3 FELICS1, FELICS2, DELTA

Verze s pevným parametrem *k* ukázala že správná volba *k* má zásadní vliv na výsledek komprese. Například s *k*=1 bylo u první sady, obrazů často dosaženo velmi dobrých výsledků (33,8 %), ale u druhé sady byl pro stejné *k* zisk několikrát záporný a v průměru byl kompresní zisk pouze 2,2 %. Parametry *k*=2, *k*=3 a *k*=4 dopadli téměř vždy velice dobře, žádný ale nebyl pro všechny případy dominantní (viz Graf 4). Pro první sadu byl v průměru nejvýhodnější parametr k=2 (37,2 %), zatímco pro druhou sadu to byl *k*=3 (20,5 %). Adaptivní volba *k* se ve verzi FELICS2 ukázala jako velmi účinná a konzistentně zisková pro všechny vzorky a pokaždé překonala nejlepší pevný parametr pro daný obraz. V první sadě byl průměrný zisk 37,9 %, v druhé sadě 21,6 %, což jsou nejlepší výsledky ze všech zkoušených metod. Verze DELTA dopadla hůře. Zisk v první sadě byl průměrně 17 % a v druhé pouhé 3 %.



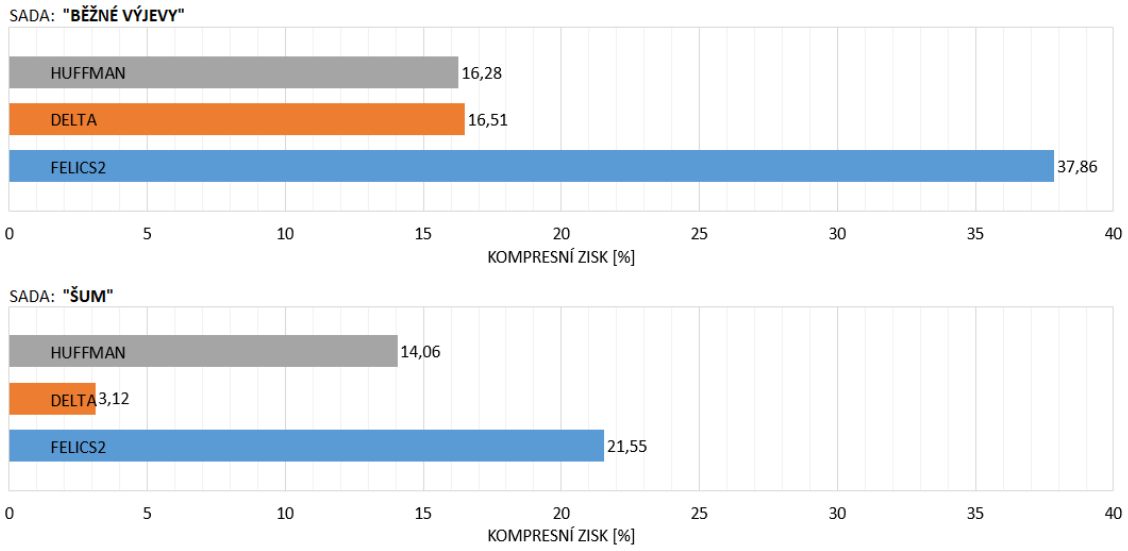
Graf 4: Kompresní zisky algoritmů FELICS1, FELICS2 a DELTA

Tabulka 5: Průměrný kompresní zisk pro algoritmy FELICS1, FELICS2 a DELTA

FELICS1:	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	FELICS2	DELTA
průměrné výsledky sady obrazů: „běžné výjevy“										
Zisk [%]	33,78	37,19	35,65	31,35	25,91	20,09	14,18	8,27	37,86	16,51
průměrné výsledky sady obrazů: „šum“										
Zisk [%]	2,19	16,85	20,51	18,52	13,65	7,37	0,56	-6,31	21,55	3,12

4.3 Výběr metody pro implementaci

Metody RLE a HUFFMAN byly z výše zmiňovaných důvodů shledány jako nevhodné pro další implementaci. Nejlepších výsledků pro metodu FELICS bylo dosaženo s verzí FELICS2. Verze DELTA by byla, vzhledem k jednoduššímu kódování hodnot, snadnější na následnou implementaci, a stejně tak verze FELICS1 by byla se svým pevným parametrem *k* pro logické obvody praktičtější. Výborná účinnost adaptivního hledání *k*, ale byla na konec shledána jako rozhodující, a proto byla pro hardwarovou implementaci vybrána verze FELICS2.



Graf 5: Porovnání algoritmu HUFFMAN, DELTA a FELICS2

Praktické zkušenosti z prvotních pokusů o přesnou implementaci verze FELICS2 vedly k několika zpětným modifikacím této verze a proto vznikla ještě verze FELICS3, která vyšla z těchto modifikací a byla elegantnější pro implementaci. Obě verze spolu byli porovnány na šesti snímcích pořízených infračervenou kamerou, aby se zjistilo, jak bude metoda FELICS účinná pro vícebitové snímky a jestli bude mít nová verze vliv na kompresní zisk. Sada s těmito testovacími snímky je nazvána jako „hlavní“ (Příloha C).

Snímky jsou originálně uloženy v rozlišení 2048x2048 jako 16bitové, ale poslední 4 bity jsou vždy nulové (efektivně jde tedy o 12bitové snímky). K sadě se při testování přistupuje jako k 16bitové, 12bitové a 8bitové a to v původním rozlišení a v rozlišení 256x256, aby se vyzkoušelo jaký vliv má na kompresní zisk rozlišení.

4.3.1 FELICS3

Tato verze se od verze FELICS2 liší hlavně v zjednodušeném hledání optimálního parametru *k*. Hledání parametru samostatně pro každou vzdálenost Δ se totiž ukázalo jako zbytečně náročné a nadbytečné. Pokud se vede společná kumulativní statistika bez ohledu na vzdálenosti Δ, je pro zkoušené obrazy dosaženo téměř totožných výsledků (viz Tabulka 6).

Jedním z možných problémů se ukázalo určení počtu bitů, na kterých se má zaznamenávat potenciální velikost obrazu pro daná *k*. V této verzi je proto nastavitelná stopka, která při překročení požadované velikosti zastaví další hledání parametru a zbytek obrazu se bude kódovat už jen s parametrem, který byl do této chvíle nejvýhodnější. Experimentálně se pro hlavní sadu obrazů ukázalo, že použití 20 bitů bude mít na výsledek jen nepatrný vliv, zatímco například 15 bitů by už vedlo k jistému zhoršení.

Pro hlavní sadu obrazů se metoda FELICS ukázala jako velmi účinná. Větší rozlišení obrazu vždy vedlo k lepšímu výsledku a rozdíl byl znát zejména s rostoucí bitovou hloubkou. Pokud se při původním rozlišení přistupovalo k sadě jako k 8bitové, bylo pro verzi FELICS3 při neomezovaném hledání *k*, dosaženo v průměru kompresního zisku 60 %, pro 12bitový přístup to bylo 45 %, a pro 16bitový přistup 37 %.

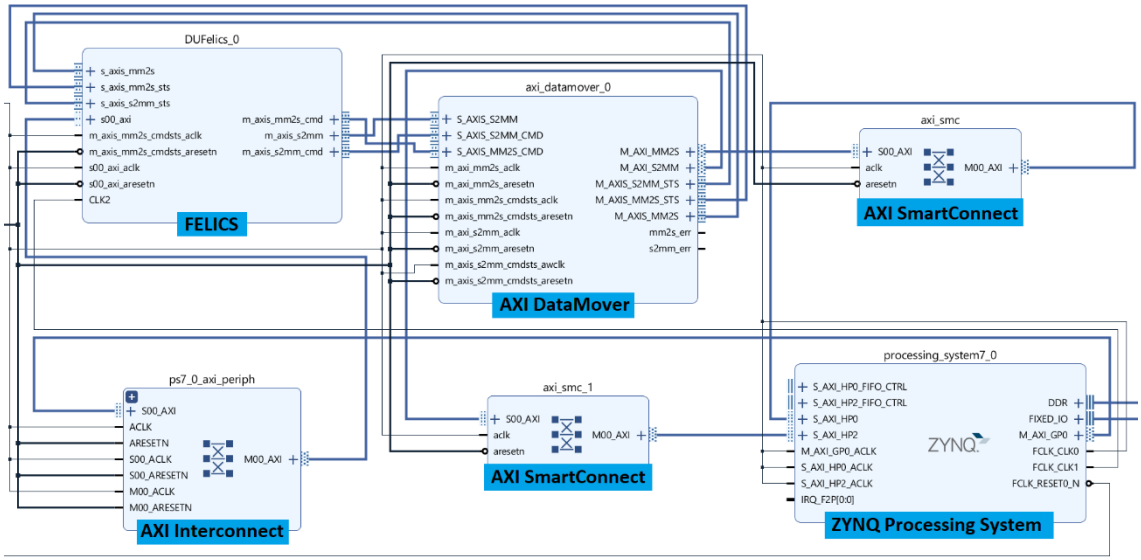
Tabulka 6: Průměrný kompresní zisk pro algoritmy FELICS2 a FELICS3

rozlišení	x-bitový přístup k „hlavní“ sadě:			x=8	x=12	x=16
2048 x 2048	FELICS2	---	kompresní zisk [%]	60,27	45,55	37,19
	FELICS3	neomezované hledání *k*		60,22	45,34	37,03
		omezení na 20 bitů		60,22	45,30	36,82
		omezení na 15 bitů		60,22	44,10	34,86
256 x 256	FELICS2	---	kompresní zisk [%]	60,07	44,75	31,28
	FELICS3	neomezované hledání *k*		59,54	43,19	33,32
		omezení na 20 bitů		59,54	43,19	32,07
		omezení na 15 bitů		59,34	36,72	28,22

Porovnání času potřebného ke kompresi některých algoritmů na PC a na procesoru platformy Zynq, jak jej požaduje zadání, je provedeno v kapitole 5.3. Tam je uvedeno spolu s rychlostí hardwarové implementace.

5. Návrh hardwarové implementace

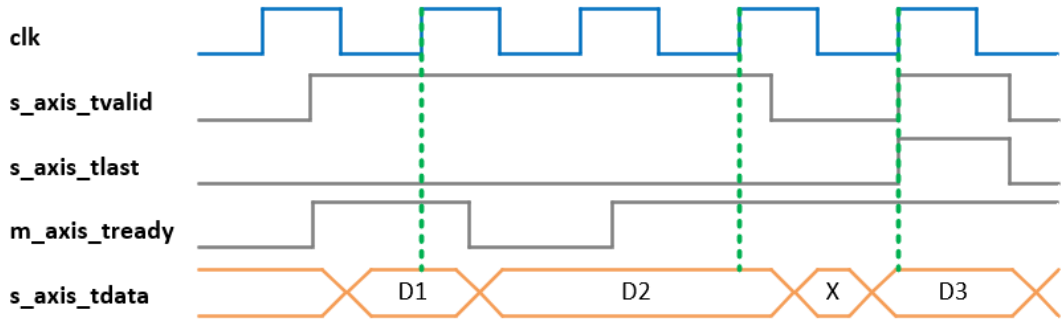
Návrh hardwarové implementace algoritmu FELICS3 (dále označován jako hlavní část IP jádra) byl vytvořen v prostředí Vivado v popisném jazyce VHDL. Celé IP jádro kromě komprese obrazu ještě zajišťuje zápis do řídících registrů, čtení a zápis dat. Návrh byl nejdříve otestován v simulaci prostředí Vivado a poté vyzkoušen na hradlovém poli desky Zedboard, kde byl integrován spolu s dalšími obvody, realizující DMA (přímý přístup do paměti), kterým se zajistilo čtení dat z paměti pro IP jádro a zápis výstupu IP jádra do paměti.



Obrázek 10: Blokové zapojení IP jádra s DMA

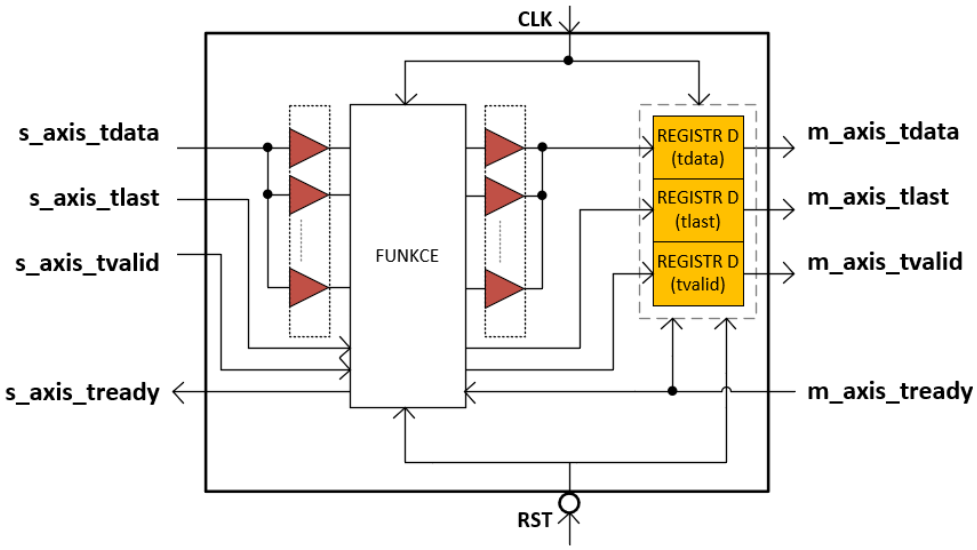
5.1 Axis blok

Hlavní část IP jádra se skládá z dvanácti komponent (axis bloků), které si mezi sebou posílají data podle modelu master/slave na základě AXI protokolu [33]. Každý axis blok má stranu slave, po které k němu chodí data a stranu master, po které data odesílá. Obě strany mají řídící signály *tvalid*, *tready*, *tlast* a datový vstup, respektive výstup *tdata* (viz Obrázek 12). To že data na vstupu *s_axis_tdata* jsou platná, je dáno hodnotou řídícího vstupu *s_axis_tvalid*. Skutečnost, že je blok schopen data přijímat, dává najevo nastavením výstupu *s_axis_tready* do jedničky. Pokud je hodnota vstupu *s_axis_tlast* v jedničce, znamená to, že jde o poslední příchozí data.



Obrázek 11: Časový diagram AXI protokolu

Hodnoty výstupu *m_axis_tdata*, *m_axis_tlast* a *m_axis_tvalid* jsou dány vnitřní funkcí bloku a jsou uloženy do registrů D, čímž se synchronizuje přenos. Vstupní řídící signál *m_axis_tready* povoluje zápis do těchto registrů a zaručuje tak, že nedojde k přepsání hodnot, pokud není následující blok schopen data přijímat. Výstupní řídící signál *s_axis_tready* je dán vnitřní funkcí bloku, ale není synchronizován přes registr.

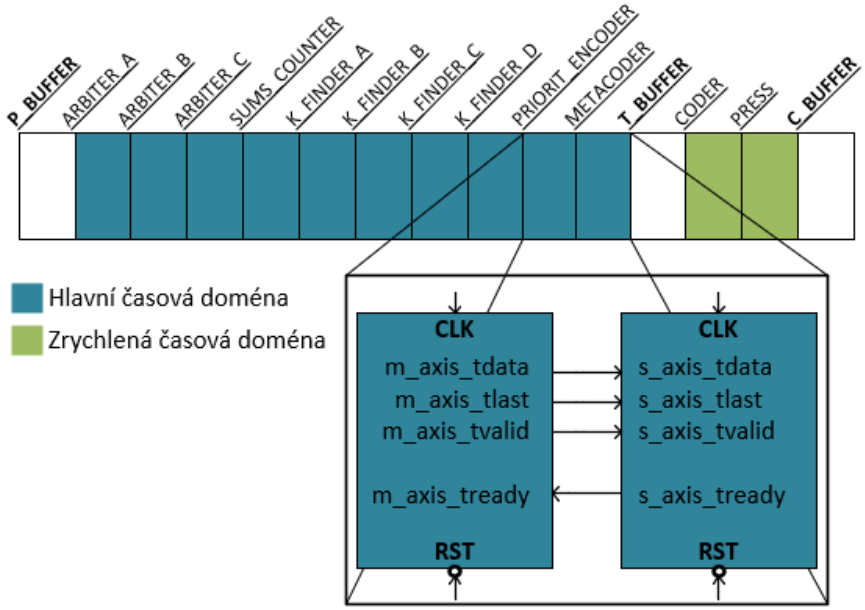


Obrázek 12: Obecné schéma axis bloku

5.2 IP jádro

Funkce každého bloku je unikátní a jsou zařazeny za sebe tak, že zřetězeně zpracovávají vstupní data (tvoří tzv. pipeline, viz Obrázek 13). Pipeline má dvě hodinové domény, základní a zrychlenou. Základní doména je 10ns. Zrychlená doména je 7ns a bloky, které s ní pracují musí být od ostatních odděleny frontou FIFO (T_BUFFER) realizující přechod mezi hodinovými doménami. Celá hlavní část jádra, která zajišťuje funkci algoritmu FELICS3 je z obou stran oddělena přes FIFO (P_BUFFER a C_BUFFER)

a jsou k ní ještě přidruženy části zajišťující komunikaci s pamětí (ty nejsou výstupem této práce a byly mi poskytnuty vedoucím).



Obrázek 13: Axis bloky tvořící hlavní část IP jádra

IP jádro bylo navrženo genericky. Pomocí generického parametru *nBits* lze volit pro jakou bitovou hloubku obrazu bude vygenerováno. Funkční hardware lze generovat pro hodnoty čtyři až šestnáct. V následujících částech práce bude popsána funkce jednotlivých bloků hlavní části, ve které bude pro jednoduchost vynechán popis práce s řídícími signály. Nejdříve bude vždy popsána principiální funkce daného bloku, bude uvedena tabulka, která interpretuje hodnoty vstupu a výstupu (hodnota v závorce udává bitovou délku), slovní popis logického obvodu, který danou funkci realizuje a jeho zjednodušené schéma.

Tabulka 7: Entita hlavní části IP jádra

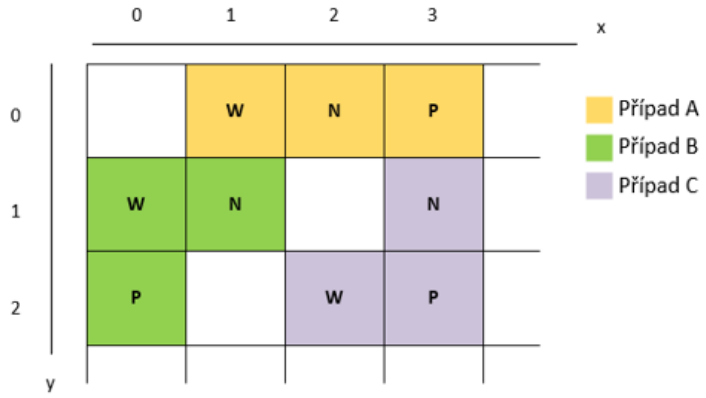
Vstupy do hlavní části IP jádra	
pixel (nBits)	hodnota příchozího pixelu, pro kterou bude vytvořeno odpovídající kódové slovo
width (⌊$\log_2$(maxWidth)⌋ + 1)	vstup daný řídícím registrem, který určuje šířku obrazu
GENERIC – nastavitelné parametry	
nBits	bitová šířka vstupních hodnot (bitová hloubka obrazu)
maxWidth	maximální šířka obrázku v pixelech (viz ARBITER_A)
log_maxSum	při překročení hodnoty $2^{(log\_maxSum-1)}$ v jakémkoli registru, zamrzne výpočet potenciální velikosti ve všech registrech (viz SUMS_COUNTER)

5.2.1 ARBITER_A

Prvním krokem u metody FELICS při kódovaní daného pixelu *P*, je nalezení jeho nejbližších sousedů. Ty budou označeny jako *N* (north) a *W* (west). Mohou nastat celkem tři případy.

- **Případ A** – Příchozí pixel *P* je na prvním řádku (*y*=0) na obecné pozici *x*. Nejbližší známí sousedé jsou na řádku *y*=0, na pozicích *x*-1 a *x*-2.
- **Případ B** – Příchozí pixel *P* je na začátku řádku (*x*=0) a jeho sousedé na pozici *x* a *x*+1, ovšem o řádek výše.
- **Případ C** – Příchozí pixel *P* je na obecné pozici *x* a *y*. Soused *W* se nachází na řádku *y* na pozici *x*-1 a soused *N* na pozici *x* na řádku *y*-1.

Zvláštním případem jsou úplně první dva pixely, které nemají potřebné sousedy, a budou se proto kódovat binárním kódem (zůstanou v nezměněné podobě).



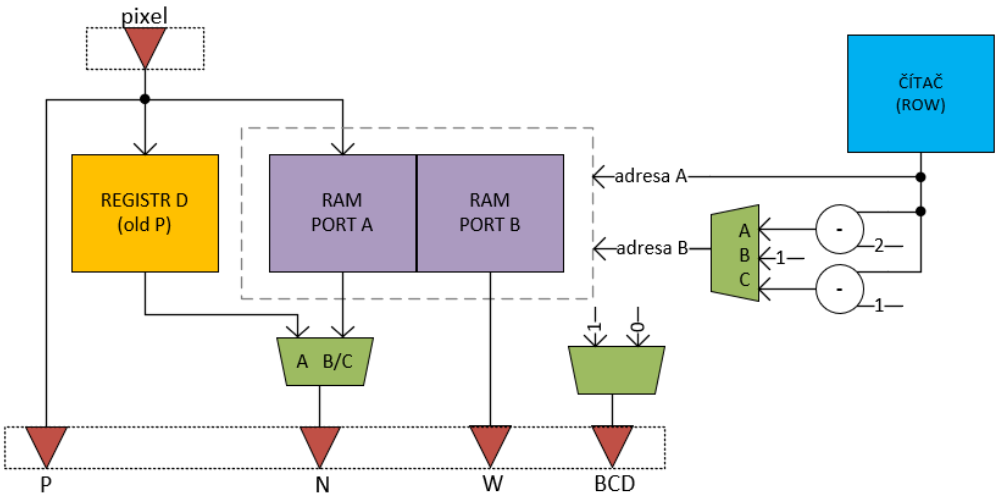
Obrázek 14: Možné kombinace v bitové mapě

Tuto práci s bitovou mapou obstarává první axis blok. Tento blok zajistí že hodnota příchozího pixelu *P* je jednak zachována pro budoucí účely a zároveň pro daný pixel nalezne jeho odpovídající sousedy *N* a *W*.

Tabulka 8: Entita axis bloku ARBITER_A

Použité vstupy	Nové výstupy	
pixel	**P** (nBits)	hodnota příchozího pixelu
width	**N** (nBits)	„severní“ soused pixelu P
	W (nBits)	„západní“ soused pixelu P
	BCD	oznamuje, zda má být použit binární kód
GENERIC – nastavitelné parametry		
maxWidth	maximální šířka obrazu (v pixelech)	

Orientace v bitové mapě je dána hodnou čítače. Pro zapamatování příchozích pixelů je použit registr D a paměť RAM (velikost určuje generický parametr *maxWidth*). V registru se udržuje hodnota pouze předchozího pixelu, v paměti se udržuje celý předchozí řádek (šířka je volena vstupem *width*). Paměť je dvoukanálová, adresy kanálů jsou odvozeny od společného čítače a čtení trvá jeden hodinový takt. Hodnoty sousedů se proto pro současný pixel *P* vyčítají z paměti vždy v předchozím taktu. První kanál se využívá pro čtení i zápis, druhý kanál pouze pro čtení. V případech B a C odpovídá soused *N* hodnotě vyčtené z prvního kanálu, do kterého se současně zapisuje příchozí pixel *P*. V případě A je soused *N* hodnota uložená v registru D. Ve všech třech případech se soused *W* vyčítá z druhého kanálu, jehož adresa se podle potřeby přepíná. Pro první dva pixely se výstup *BCD* nastavuje do nuly.



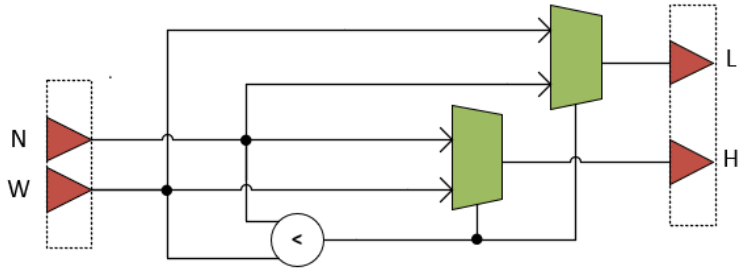
Blokové schéma 1: Obvod pro axis blok ARBITER_A

5.2.2 ARBITER_B

Sousední pixely *N* a *W* je pro další potřeby nutné porovnat a zjistit, který z nich má větší hodnotou. Pixel s větší hodnotou bude nově označen jako *H* a pixel s nižší hodnotou jako *L*. Jde o triviální úkon, který ale může zabrat značný čas, a proto je pro zpracování vyhrazen samostatný axis blok. Čas potřebný k porovnání dvou čísel je závislý na bitové délce porovnávaných hodnot a pokud by byl tento proces přidružen k bloku předchozímu, pak by požadovaný čas 10ns nemusel být dostatečný.

Tabulka 9: Entita axis bloku ARBITER_B

Použité vstupy	Nové výstupy	
N	**H** (nBits)	větší soused pixelu P
W	**L** (nBits)	menší soused pixelu P
Přetrvávající výstupy: P; BCD		

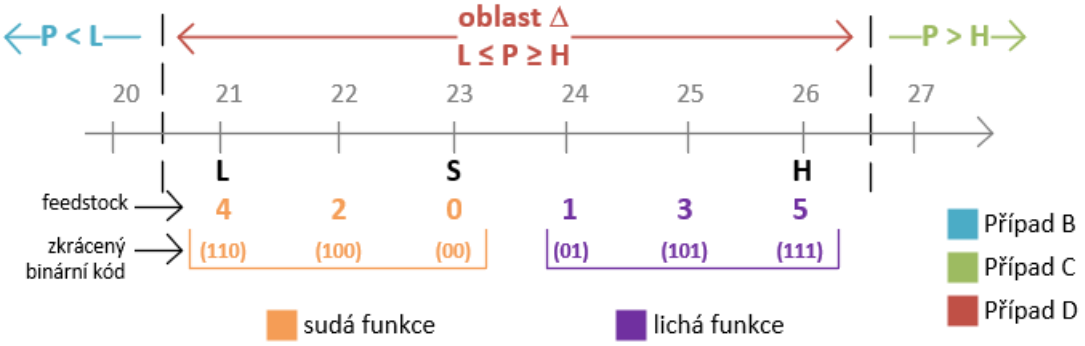


Blokové schéma 2: Obvod pro axis blok ARBITER_B

5.2.3 ARBITER_C

V tomto axis bloku se vypočítá hodnota, která má být dále zakódována, a má tak reprezentovat právě příchozí pixel *P*. Tato hodnota je označena jako *feedstock* (palivo) a může být vypočtena dle tří různých scénářů.

- **1. scénář** – Vstup *BCD* je roven jedné a hodnota *feedstock* se bude rovnat pixelu *P* (případ A).
- **2. scénář** – Pixel *P* je menší než *L*, nebo větší než *H*. Výstup *feedstock* bude roven vzdálenosti *P* od *L* (případ B), respektive vzdálenosti *P* od *H* (případ C).
- **3. scénář** – Pixel *P je* v oblasti Δ (ostrá oblast hodnot mezi *H* a *L*) a hodnota *feedstock* bude záležet na tom, zda je *P* větší nebo menší než středová hodnota *S* oblasti Δ ($S=L+(H-L)/2$). Vzdálenost *P* od *S* bude označena jako *x*. Pro hodnotu větší než *S*, se pro výpočet hodnoty *feedstock* použije lichá funkce $L(x)=2x-1$, pro hodnotu menší nebo rovnou *S* se použije sudá funkce $S(x)=2x$. Tímto postupem se zajistí, že při použití zkráceného binárního kódu se bude délka kódu pro vzdálenost *x* zvětšovat symetricky od střední hodnoty *S*, okolo které se počítá s pravděpodobnějším výskytem pixelu *P*, než u krajních hodnot oblasti Δ (viz kapitola 2.4.6).



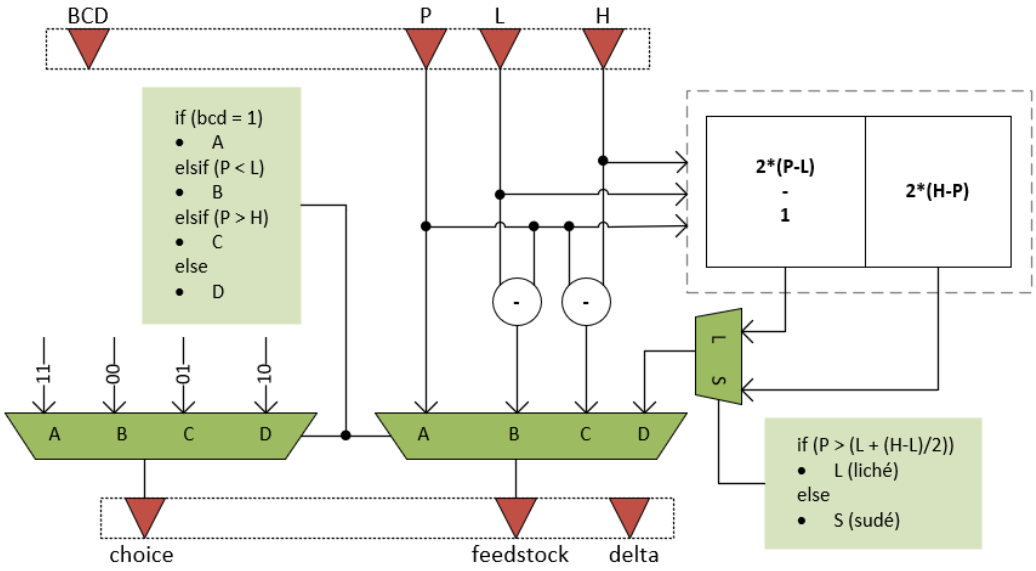
Obrázek 15: „Cik-Cak“ reprezentace hodnoty vzdálenosti od středu S

Protože pro jednotlivé scénáře se hodnota *feedstock* bude kódovat jiným kódem a v druhém scénáři se do kódového slova uvádí, zda jde o případ B, nebo C, je pro tyto účely zaveden výstup *choice*, který oznamuje, jaký případ pro danou hodnotu *feedstock* platí. V dalších částech bude potřeba znát rozdíl hodnot *H* a *L*, nikoli však hodnoty samotné, a proto je vytvořen výstup *delta*, pro který platí že *delta=H-L*.

Tabulka 10: Entita axis bloku ARBITER_C

Použité vstupy	Nové výstupy	
P	**choice** (2)	oznamuje, jaký případ nastal (A=11; B=00; C=01; D=10)
H	**delta** (nBits)	rozdíl pixelů H a L
L	**feedstock** (nBits)	hodnota, která má být dále zakódována
BCD		

Možné výstupy pro výstup *choice* jsou předem nadefinované. Možné výstupy pro výstup *feedstock* jsou získány tak, jak bylo výše nastíněno. Pro případ A je použit vstup *P*, pro případy B a C se použije odpovídající rozdíl vstupů *H*, *L* a *P* a pro případ D se přepíná sudá a lichá funkce podle toho, zda je, či není pixel *P* větší než hodnota *S*. Oba výstupy jsou získány přepínáním podle nastalého případu.



Blokové schéma 3: Obvod pro axis blok ARBITER_C

5.2.4 SUMS_COUNTER

V tomto axis bloku začíná hledání optimálního parametru *k* potřebného pro tvorbu Ricova kódu (viz kapitola 4.3.1). Blok zajišťuje výpočet potenciální velikosti kódu při použití daných parametrů *k* na příchozí hodnotu *feedstock*. Tyto velikosti si kumulativně ukládá a na výstupu jsou uvedeny v poli s názvem *sums*. Výpočet je k uloženým hodnotám

přičten, jen pokud se pro danou konstelaci pixelů má následně Ricův kód použít, a také pokud uložené hodnoty zatím nepřekročily předem definovanou velikost danou generickým parametrem *log_maxSum*.

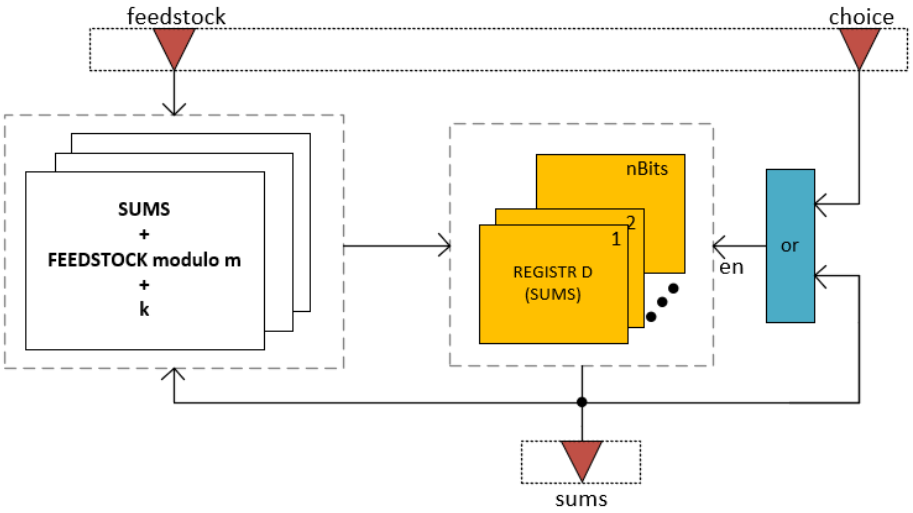
Tabulka 11: Entita axis bloku SUMS_COUNTER

Použité vstupy	Nové výstupy	
choice	**sums** (nBits*log_maxSum)	pole ve kterém je pro každý parametr *k* vypočtená dosavadní potenciální velikost
feedstock		
Přetrvávající výstupy: choice, delta, feedstock		
GENERIC – nastavitelné parametry		
log_maxSum	při překročení hodnoty $2^{(log\_maxSum-1)}$ v jakémkoli registru, zamrzne výpočet potenciální velikosti ve všech registrech	

Pro každý parametr *k* je určen vlastní registr D. Zápis do registrů je povolen pouze pokud žádný z registrů nemá na nejvýznamnějším bitu hodnotu '1' a pokud na vstupu *choice* je hodnota '00', nebo '01'. Do každého registru se přičte hodnota *feedstock* oříznutá o *k* dolních bitů a samotná hodnota *k*. Oříznutá hodnota *feedstock* je formálně rovna podílu po dělení hodnotou 2^k a jde tedy o číslo reprezentující počet jedniček, kterými se v Ricově kódu reprezentuje podíl po dělení parametrem *m* (viz Ricovo kódování na str. 23). Hodnota *k* pak odpovídá počtu bitů, na kterých se v Ricově kódu uvádí zbytek po dělení. Takto se určí potenciální velikost při použití příslušných parametrů *k*, a to bez toho, aniž by se musel pro každý registr vytvářet odpovídající Ricův kód.

Poznámka:

Parametr *k*=0 se nepoužívá a první hodnota *k* je ve všech částech popisu IP jádra vždy rovna jedné (*k*=1 je na nulté pozici).



Blokové schéma 4: Obvod pro axis blok SUMS_COUNTER

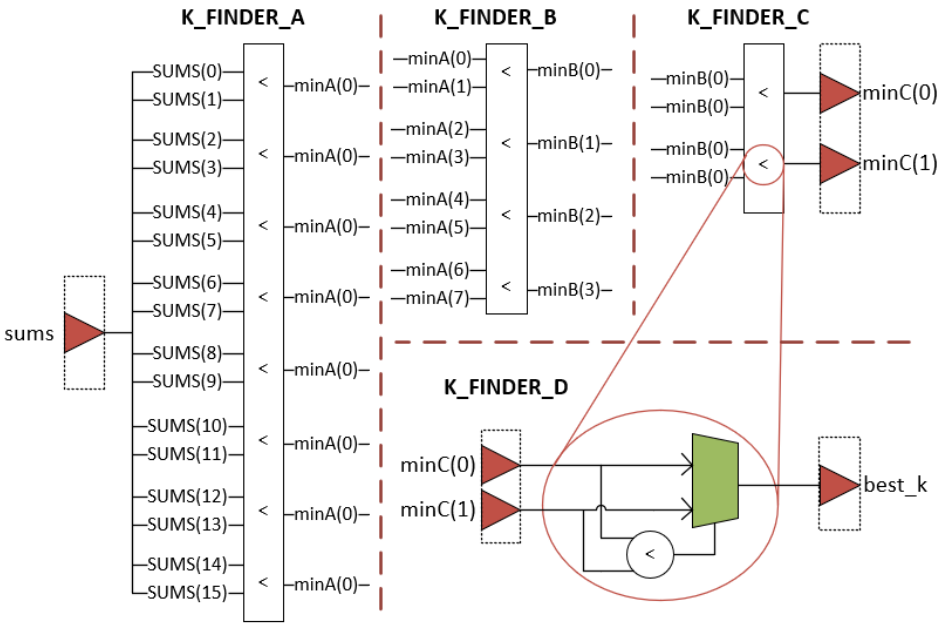
5.2.5 K_FINDER (A, B, C, D)

Aby se zjistilo, která hodnota pole *sums*, reprezentující dané *k*, je nejmenší, je potřeba mezi sebou všechny hodnoty porovnat. K tomuto účelu jsou vyhrazeny tyto čtyři axis bloky. V prvním bloku se porovná osm dvojic hodnot z pole *sums*, a ty s menší hodnotou se přiřadí na výstup. V druhém bloku se porovnají čtyři výsledné dvojce bloku prvního a ve třetím zase dvě dvojce bloku třetího. Ve čtvrtém bloku se provede výsledné porovnání a parametr *k* příslušící nejmenší hodnotě, bude na výstupu označen jako *best_k* (best_k=0 odpovídá k=1). Tento porovnávací pavouk je navrhnut tak, aby se v jednom taktu neporovnávalo více hodnot za sebou, a velikost registrů D v bloku *SUMS_COUNTER*, tak byla omezena jen dobou potřebnou právě pro jedno porovnání.

Tabulka 12: Entita axis bloků K_FINDER (A, B, C, D)

Použité vstupy	Nové výstupy	
sums	**best_k** (4)	parametr *k*, který by byl pro dosavadní kódování nejvýhodnější
Přetrvávající výstupy: choice, delta, feedstock		

Axis bloky, které dohromady tvoří *K_FINDER* vždy porovnávají 16 hodnot. Je-li generický parametr *nBits* menší než 16, nastaví se nepotřebné porovnávací vstupy do maximální hodnoty, díky čemuž neovlivní výsledek. Každé porovnání je prováděno tak, jako tomu bylo u bloku *ARBITER_B*, jen s tím rozdílem, že vyšší hodnota není potřeba a se samotnou porovnávanou hodnotou se musí udržovat informaci o tom, ke kterému *k* přísluší.



Blokové schéma 5: Obvod pro axis bloky K_FINDER (A, B, C, D)

5.2.6 PRIORIT_ENCODER

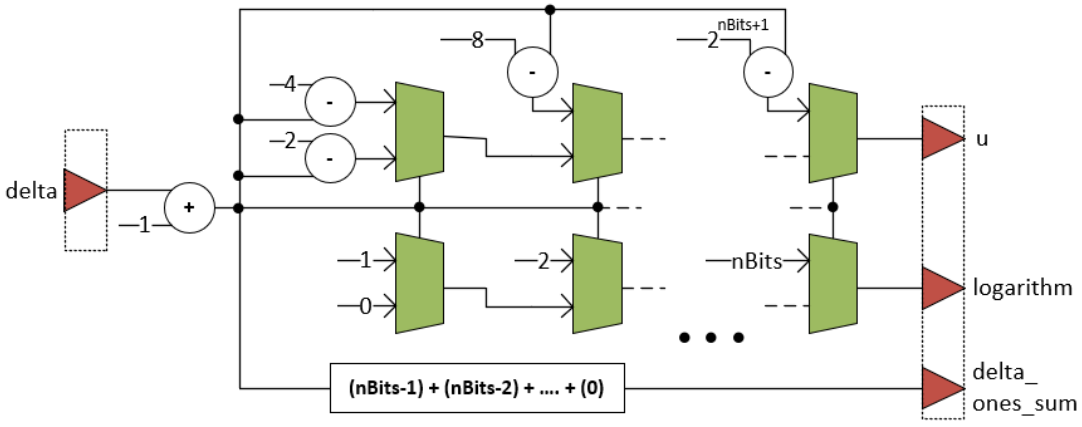
V tomto axis bloku jsou předpočteny hodnoty, které jsou zapotřebí pro tvorbu zkráceného binárního kódu. Rozsah hodnot *h*, pro které bude zkrácený binární kód vznikat, odpovídá vstupu *delta* zvětšeného o hodnotu jedna. Je potřeba vypočíst hodnotu *u*, která například udává kolik hodnot z binárního kódu se nepoužije (viz zkrácené binární kódování na str. 23), dolní celou část dvojkového logaritmu z hodnoty rozsahu *h* a počet aktivních bitů hodnoty *h*. Počet aktivních bitů je na výstupu označen jako *delta_ones_sum* a pokud je roven jedné, znamená to, že hodnota rozsahu *h* je mocnina dvou, což je skutečnost, která ovlivňuje tvorbu zkráceného binárního kódu.

Tabulka 13: Entita axis bloku PRIORIT_ENCODER

Použité vstupy	Nové výstupy	
delta	**delta_ones_sum** (5)	počet bitů v jedničce u hodnoty delta+1
	u (nBits+1)	počet hodnot kratším kodem
	logarithm (5)	$\lfloor \log_2(\text{delta}+1) \rfloor$
Přetrvávající výstupy: best_k, choice, feedstock		

Možné hodnoty výstupu *logarithm* jsou předem známy. Dolní celá část logaritmu hodnoty *delta*+1 totiž odpovídá váze právě toho bitu hodnoty *delta*+1, který je ze všech aktivních bitů nejvýznamnější, a formálně se tedy jedná o funkci prioritního enkodéru. Každý přepínač enkodéru je řízen příslušným bitem hodnoty *delta*+1 a každému bitu této hodnoty, přísluší právě jedna hodnota vstupu. Označí-li se priorita vstupu jako *x* a pro nejnižší prioritu bude platit, že *x*=0, bude hodnota vstupů v takovémto případě stejná jako jejich priorita.

Protože hodnota *u* souvisí s hodnotou *logarithm*, bude výstup nalezen podle stejných pravidel a hodnoty vstupů do enkodéru jsou vypočteny podle formule $f(x)= 2^{x+1} - delta+1$.



Blokové schéma 6: Obvod pro axis blok PRIORIT_ENCODER

5.2.7 METACODER

V této části jsou již známy všechny potřebné hodnoty nutné pro vygenerování kódového slova. Aby bylo možné generovat různě dlouhé výstupní hodnoty, bude v tomto axis bloku vytvořen řetězec hodnot, ve kterém bude podle zvolené předlohy dané kódové slovo reprezentováno. Podle zvolených pravidel pak následující blok bude v několika krocích kódové slovo generovat. Řetězec se skládá ze šesti částí, přičemž první dvě části slouží pro všechny kódy, zbylé jen pro Ricův kód. U něj je v těchto částech řetězce vyřešeno generování unární části, která může v okrajových případech potřebovat stovky bitů.

- **1. část řetězce** – Skládá z 18 bitů a v případě binárního a zkráceného binárního kódu se na ni ukládá celé kódové slovo. V případě Ricova kódu se na tuto část uloží vstup *choice* a maximálně 16 jedniček z unární části kódu.
- **2. část řetězce** – Oznamuje, kolik bitů v první části je platných.
- **3. část řetězce** – Pokud není 16 bitů v první části dostatečných pro reprezentaci unární části kódu a je potřeba generovat dalších, více jak 16 jedniček, bude v této části uloženo kolikrát je ještě potřeba 16 jedniček vygenerovat.
- **4. část řetězce** – Oznamuje, kolik jedniček mezi 0 a 15 je potřeba vygenerovat nad rámec první, nebo třetí části.
- **5. část řetězce** – Slouží pro binární část Ricova kódu, kdy jeden bit je určen pro oddělující nulu a zbylé pro část reprezentující zbytek po dělení.
- **6. část řetězce** – Oznamuje, kolik bitu v páté části je platných.

Tabulka 14: Entita axis bloku METACODER

Použité vstupy	Nové výstupy	
delta_ones_sum	**codeA** (18)	první část řetězce
u	**sizeA** (5)	druhá část řetězce
logarithm	**codeX16** (nBits-4)	třetí část řetězce
best_k	**codeXR** (4)	čtvrtá část řetězce
choice	**codeB** (nBits+1)	pátá část řetězce
feedstock	**sizeB** (5)	šestá část řetězce

Podle vstupních hodnot je paralelně vytvořen řetězec pro binární, Ricův a zkrácený binární kód. Volba řetězce se provádí přepínáním podle vstupní hodnoty *choice*. Pro hodnoty '01' a '00' se použije řetězec Ricova kódu, pro '10' řetězec zkráceného binárního kódu a pro '11' řetězec binárního kódu, který vzniká pouhým zkopírováním vstupu *feedstock* na výstup *codeA* a zapsání hodnoty *nBits* na výstup *sizeA*.

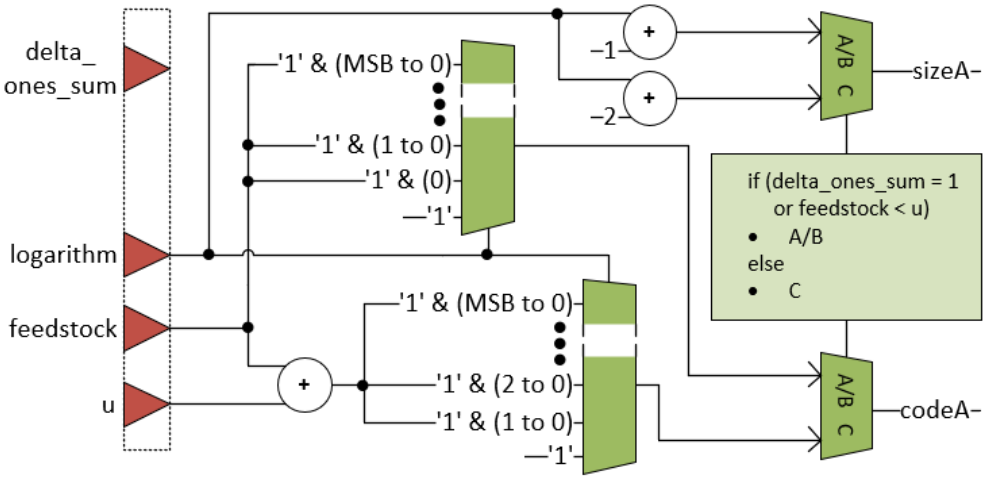
Princip pro zkrácený binární kód

- **Případ A** – Pokud je hodnota *feedstock* menší než hodnota *u*, bude výsledný kód totožný se vstupem *feedstock*, přičemž počet platných bitů této hodnoty bude odpovídat hodnotě *logarithm*+1.
- **Případ B** – Pokud je vstup *delta_ones_sum* roven jedné jsou výstupy totožné jako v případě A.
- **Případ C** – Pro hodnoty vstupu *feedstock*, které jsou větší nebo rovny hodnotě *u*, bude potřeba o jeden bit více než v případě A a ke vstupu *feedstock* se bude přičítat hodnota vstupu *u*.

Ve všech třech případech je před kód potřeba zapsat na příslušný bit jedničku, kterou se indikuje že jde o Ricův kód.

Výstupy pro zkrácený binární kód

Protože může být kód různě dlouhý, tak je pro všechny možné velikosti vytvořen pro každý z případů A/B a C odpovídající vstup. Výstup *CodeA* bude výsledkem přepínání těchto vstupů na základě vstupu *logarithm* (určuje velikost kódu) a také podle toho o jaký případ se jedná. Výstup *sizeA* je výsledkem přepínání inkrementovaného vstupu *logarithm* pro případ A/B, nebo C.



Blokové schéma 7: Obvod pro axis blok METACODER (zkrácený binární kód)

Princip pro Ricův kód

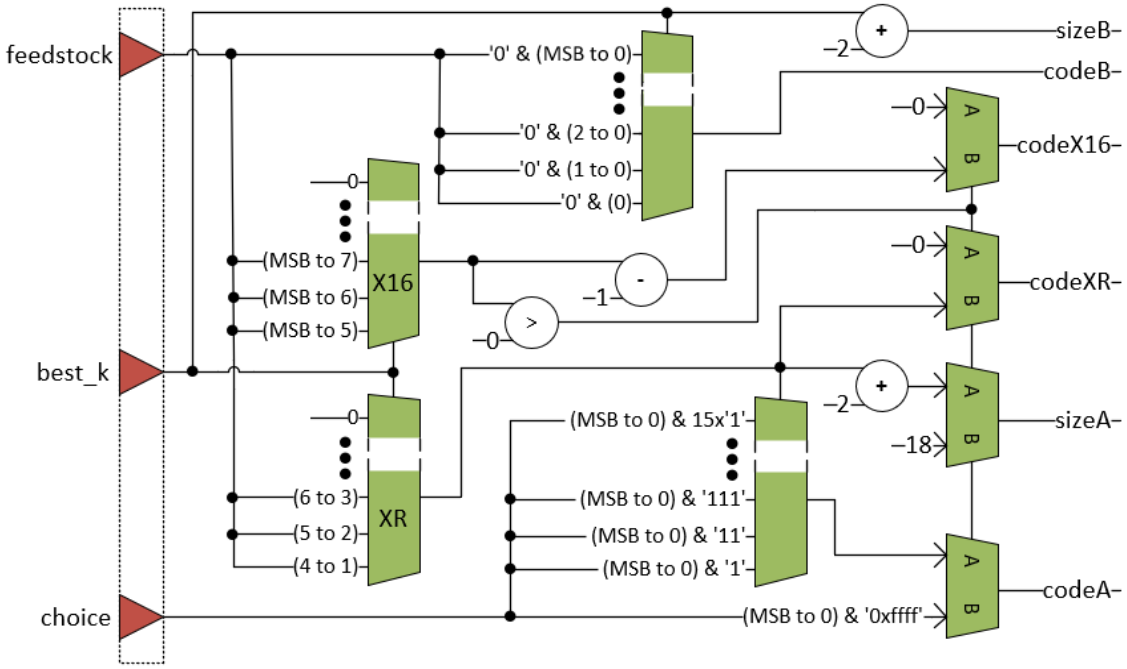
V případě Ricova kódu je pro jeho první část potřeba získat podíl a pro druhou část zbytek po dělení hodnoty *feedstock* parametrem *m*. Zbytek po dělení *m* je roven dolním *k* bitům vstupu *feedstock* a podíl odpovídá vstupu *feedstock* oříznutému o *k* dolních bitů.

Pro tvorbu řetězce, je potřeba z hodnoty podílu po dělení parametrem *m* získat podíl a zbytek po dělení této hodnoty hodnotou 16. Tento nový podíl bude nazván jako *X16* a bude odpovídat oříznutí předchozího podílu o čtyři bity, zbytek podílu je označen jako *XR* a je roven právě těmto čtyřem bitům.

Výstupy pro Ricův kód

Zmíněným principem se získají hodnoty pro každé možné *k* a budou přepínány podle vstupu *best_k*. Výstup *codeB* bude roven příslušnému zbytku po dělení parametrem *k* a výstup *sizeB* hodnotě *best_k+2*.

- **Případ A** – Pokud je *X16* rovno nule, nejsou výstupy *codeX16* a *codeXR* potřeba (celá unární část se totiž vejde na *codeA*) a jsou proto nastaveny do nuly. Výstup *codeA* je výsledkem přepínání na základě *XR*, kdy možné vstupy odpovídají vstupu *choice* následovaného všemi variantami počtu jedniček. Výstup *SizeA* je roven hodnotě *XR+2*.
- **Případ B** – Pokud je *X16* větší jak nula, bude výstup *CodeA* roven vstupu *choice* a šestnácti jedničkám, *SizeA* bude hodnota 18, *CodeX16* bude roven *X16*-1 (prvních šestnáct jedniček je zapsáno na *codeA*) a *codeXR bude* roven *XR*.



Blokové schéma 8: Obvod pro axis blok METACODER (Ricův kód)

5.2.8 CODER

Tento axis blok generuje kódové slovo a je jediný, který nezpracuje vstupní hodnoty během jednoho hodinového taktu a je zde proto použit zrychlený hodinový signál. Jeden hodinový takt bude stačit, pokud byl zvolen řetězec pro binární, nebo zkrácený binární kód a je tak potřeba generovat pouze hodnotu *codeA*. V případě řetězce Ricova kódu je vždy potřeba vygenerovat hodnotu *codeA* a *codeB* a jsou tedy potřeba minimálně dva takty. Pokud je pro Ricův kód potřeba vygenerovat i počet jedniček dle hodnoty *codeXR* bude potřeba další hodinový takt a pokud podle hodnoty *codeX16* bude potřeba *codeX16* taktů. Hodnota se generuje vždy na 18bitovém výstupu s názvem *code*, přičemž počet platných bitů je uveden na výstupu *size*.

Tabulka 15: Entita axis bloku CODER

Použité vstupy	Nové výstupy	
celý řetězec	**code** (18)	část kódového slova, potažmo celé kódové slovo
	size (5)	počet bitů, které ve výstupu *code* tvoří platnou část

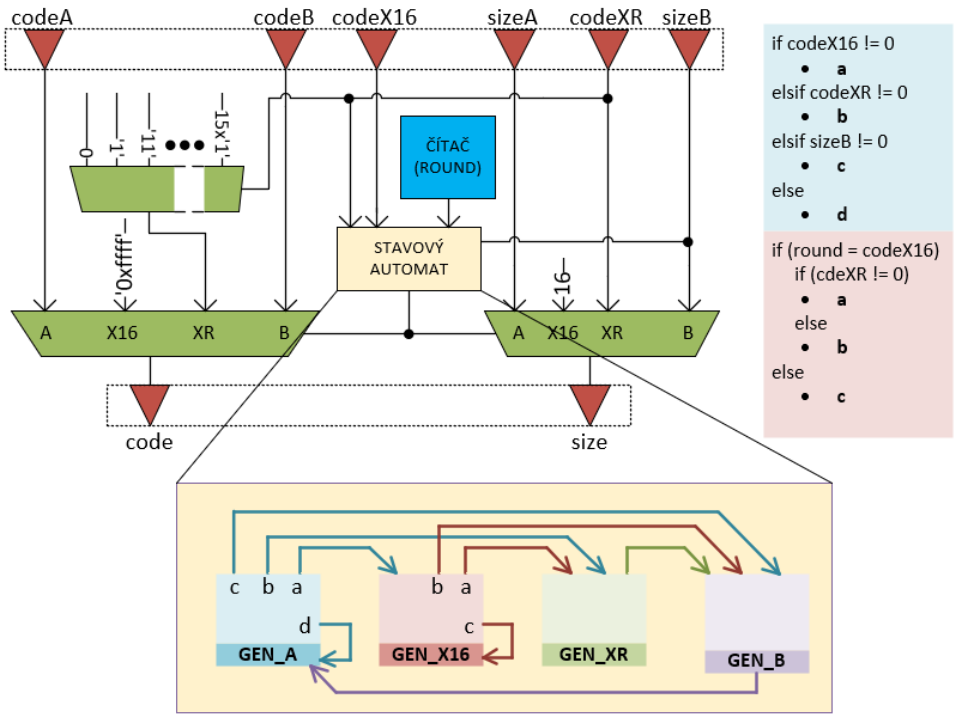
O tom, která část vstupního řetězce se dostane na výstup rozhoduje stavový automat typu Moore. Ten má celkem čtyři stavy, stav *GEN_A*, *GEN_X16*, *GEN_XR* a *GEN_B*. Každý stav slouží pro vygenerování kódového slova podle části řetězce *codeA*, *codeX16*, *codeXR*, nebo *codeB*.

Stavy automatu

Stav *GEN_A* je výchozí a pro každý nový vstupní řetězec se v něm začíná. Ze stavu *GEN_A* se pokračuje do nejbližšího stavu, který generuje nenulovou hodnotu. Stav *GEN_X16* se může opakovat a počet opakování je dán čítačem, který se porovnává se vstupem *code_X16*. Pokud stav *GEN_XR* nebude generovat žádnou hodnotu, přeskočí automat z *GEN_X16* do stavu *GEN_B*. Stav *GEN_XR* vede vždy po jednom taktu na stav *GEN_B*, který vždy po jednom taktu vede na původní stav *GEN_A*, ve kterém se bude generovat výstup již podle nového řetězce.

Výstupy

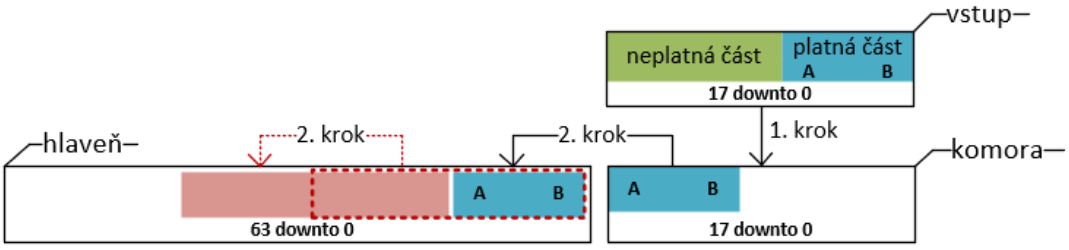
Výstupy *code* a *size* jsou pro stavy *GEN_A* a *GEN_B* dány přímo vstupními hodnotami. Pro stav *GEN_X16* je výstup *code* roven šestnácti jedničkám a výstup *size* hodnotě 16. Pro stav *GEN_XR* je výstup *size* dán přímo vstupem *codeXR* a výstup *code* je počet jedniček o kterém rozhoduje přepínač řízený vstupem *codeXR*.



Blokové schéma 9: Obvod pro axis blok CODER

5.2.9 PRESS

Proud platných hodnot z předchozího bloku, je potřeba pro účely ukládání správně zařadit za sebe do výstupu s pevnou bitovou délkou. Tento nový výstup je označen jako *cipher* a je 64bitový. Aby byl blok schopen vstupní hodnotu od předchozího bloku, který má zrychlenou časovou doménu, zpracovat za jeden hodinový cyklus, je stejná doména použita i v tomto bloku. Blok si vstupní data ukládá (do „komory“) a následně posouvá jejich platnou část o příslušný počet bitů k předchozím, již zpracovaným hodnotám (do „hlavně“).



Obrázek 16: Princip řazení hodnot v axis bloku PRESS

Pokud má blok k dispozici 64 a více platných bitů, vygeneruje příslušný výstup a začne si ukládat nové hodnoty. Pokud poslední příchozí hodnota (*tlast*='1') nezaručí, že se na výstupu vygeneruje 64 bitů beze zbytku, doplní se zbývající uložená hodnota o příslušný počet nul. Toho blok docílí tím, že se bude do chvíle, než sám vygeneruje poslední výstup chovat tak, jako by na jeho vstupu byla nulová hodnota na jednom bitu.

Tabulka 16: Entita axis bloku PRESS

Použité vstupy	Nové výstupy	
code	**cipher** (64)	výstup v podobě ve které je již určen pro ukládání do paměti
size		

První krok

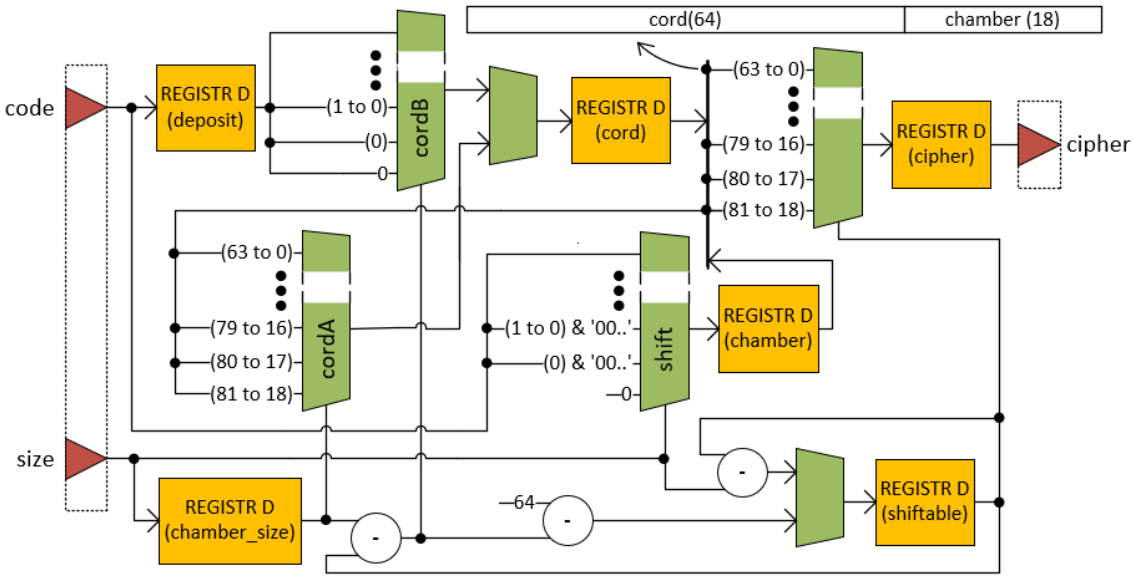
Vstup je nejdříve uložen do registru *deposit* a registru *chambre* (komory). Počet platných bitů vstupu se ukládá do registru *chambre_size*. V registru *deposit* je hodnota v nezměněné podobě, v „komoře“ je hodnota pomoci přepínače (shift) řízeného vstupem *size* posunuta do leva o tolik bitů, aby platilo, že nejvýznamnější platný bit vstupní hodnoty bude na pozici nejvýznamnějšího bitu registru.

Druhý krok

Při nové příchozí hodnotě je obsah „komory“ přesunut do registru *cord* (hlavně) na odpovídající počet spodních bitů, přičemž původní obsah „hlavně“ se o stejnou část bitů posune. Tato operace je provedena pomocí přepínače (cordA), který je řízen hodnotou registru *chambre_size*. To o kolik hodnot lze obsah „hlavně“ posunout, než se zcela zaplní, je vypočteno a uloženo do registru *shiftable*.

Výstup

Pokud by obsah „hlavně“ byl již zaplněn tak, že by její posunutí znamenalo úplné zaplnění nebo přetečení, bude obsah hlavně a příslušná část obsahu „komory“ uložena do výstupního registru *cipher*. Správné spojení hodnot je provedeno přepínačem (cipher), který je řízen registrem *shiftable*. Bity, které se na výstup nevejdou se uloží na odpovídající spodní část „hlavně“. Půjde o bity z registru *deposit*, které se vyberou přepínáním (cordB) podle rozdílu počtu platných bitů v „komoře“ a registru *shiftable*.



Blokové schéma 10: Obvod pro axis blok PRESS

5.3 Zhodnocení výsledků implementace

Výsledky hardwarové implementace se shodují s výsledky algoritmu FELICS3, který zpracoval počítačový procesor. Návrh byl otestován pro genericky parametr *nBits*=8 a *nBits*=16. Pro hlavní část IP jádra při *nBits*=8 byl syntézní odhad zdrojů 1598 LUT a 1614 registrů. Pro *nBits*=16 byl odhad 2428 LUT a 2294 registrů. Pro platformu Zynq-7020 jde o 3,2 (4,6) % všech LUT, 1,5 (2,2) % registrů a 2,2 (2,5) % paměti RAM. Celé IP jádro vyžadovalo přibližně ještě další polovinu těchto zdrojů (viz *Tabulka 17*).

Návrh byl vyzkoušen pro 8bitové a 16bitové snímky v sadě „hlavní“ v rozlišení 256x256. Čas potřebný pro kompresi byl v průměru 1,14ms pro 8bitové snímky a 1,4ms pro 16bitové (rozdílný čas je dán pravděpodobně odlišným plněním front FIFO). Na počítačovém procesoru i5-7300HQ trval algoritmus FELICS3 řádově jednotky ms, na procesoru ARM Cortex-A9 platformy Zynq desítky ms (viz tabulka *Tabulka 18*). V tabulce je ještě navíc pro porovnání verze DELTA, která se ani z hlediska rychlosti neukázala nikterak výhodnější než verze FELICS3 a algoritmus RLE1 (se čtyřmi bity pro počet opakování), který je považován za velmi rychlý. Algoritmus FELIC3 byl oproti RLE1 pomalejší 1,6x na PC a 2,6x na procesoru Zynq.

Tabulka 17: Syntézní odhad použitých zdrojů

Syntézní odhad použitých zdrojů pro jednotlivé části návrhu						
axis blok: (syntéza pro každý zvlášť)	pro nBits = 8			pro nBits = 16		
	LUT	Registry	BRAM	LUT	Registry	BRAM
ARBITER_A	52	58	0,5	60	98	1
ARBITER_B	13	27	0	25	51	0
ARBITER_C	95	20	0	191	36	0
SUMS_COUNTER	41	356	0	145	708	0
K_FINDER_A	91	109	0	179	213	0
K_FINDER_B	97	120	0	97	136	0
K_FINDER_C	49	70	0	49	86	0
K_FINDER_D	14	24	0	14	40	0
PRIORIT_ENCODER	44	33	0	404	51	0
METACODER	82	44	0	159	60	0
CODER	41	31	0	45	39	0
PRESS	651	246	0	651	246	0
∑ =	1270	1138	0,5	2019	1764	1
Hlavní část IP jádra v celku	1598	1614	3	2428	2294	3,5
Celé IP jádro	2262	2969	4	3095	3649	4,5
IP jádro s DMA	5529	6777	6,5	6419	7544	7

Tabulka 18: Kompresní čas hw. implementace a některých algoritmů

bitová hloubka:	8	16	8	16	8	16
FELICS3 pro vzorky 256x256	čas potřebný pro kompresi [ms]					
	Intel® Core™ i5-7300HQ @ 2,5GHz		ARM Cortex-A9 @ 660MHz		hw. implementace	
A	10	26	84	213	1,14	1,48
B	12	25	97	182	1,19	1,42
C	12	25	99	175	1,21	1,39
D	10	24	81	191	1,12	1,46
E	8	16	72	129	1,09	1,20
F	8	23	83	172	1,11	1,41
průměrné výsledky sady obrazů: „hlavní“ s rozlišením 256x256						
FELICS3	10	23	86	177	1,14	1,40
DELTA	11	29	65	181	-	-
RLE1	6	15	37	67	-	-

5.4 Návrh na úpravu

Nevýhodou návrhu je jeho část se zrychlenou časovou doménou. Ta totiž není zrychlená dostatečně a pokud převažuje Ricovo kódování (tomu tak je u všech testovaných snímků) budou postupně zahlceny FIFO a IP jádro nebude schopné v každém taktu přijímat pixel na vstupu, čímž se zbytečně protahuje doba komprese. Pokud by totiž každý blok zpracovával vstupní data vždy za jeden takt, pak při opomenutí problematiky režie odesílání hodnot do paměti, by pro obrazy s rozlišením 256x256 byl kompresní čas roven 0,65ms.

Řešením by bylo použít pro zrychlenou doménu vyšší frekvenci. Použití periody 6ns by již dle výsledků simulace nevedlo k zaplnění FIFO. Výrazně nižší perioda než 7ns, ale nelze pro blok *CODER* použít, protože by již nebyla pro daný logický obvod dostatečná.

Možným řešením je upravit Ricův kód pro extrémní případy, kdy se generuje více jak 16 jedniček. Blok *METACODER* by v těchto případech mohl nahradit oddělující nulu jedničkou v části řetězce *codeA* a do části *codeB* by uvedl požadovanou hodnotu v nezměněné (binární) podobě. Části *codeX16* a *codeXR* by již nebyly potřeba a pro blok *CODER* by se dalo nalézt řešení, které by v jednom taktu spojovalo část *codeA* a *codeB*. V takovémto návrhu by nebyla zrychlená časová doména a každý blok by zpracoval data na svém vstupu právě za jeden takt.

Podle statistiky provedené na algoritmus FELICS3 na PC, bude potřeba vygenerovat více jak 16 jedniček nejčastěji u obrazu „C“ (16bitová hloubka, rozlišení 2048x2048), kde jde o 0,32 % případů a pro jednu hodnotu je dokonce potřeba generovat 133 jedniček. Ricův kód se v takovéto chvíli stává vždy neefektivní a navrhovaná úprava kódu by proto neměla vést k výraznému zhoršení kompresního zisku. Dekodér by tako upravený kód detekoval tím, že posloupnost jedniček u Ricova kódu bude rovna hodnotě 17 a pro dekompresi by používal následnou binární hodnotu.

6. Závěr

Z rešerše bezztrátových kompresních algoritmů se práce zaměřila na Huffmanovo kódování, algoritmus RLE a zejména na algoritmus FELICS. Tyto algoritmy byly testovány vlastnoručně vytvořenými programy pro realizaci kodéru a dekodéru na dvou testovacích sadách. Pro každý algoritmus byl učiněn pokus o optimalizaci kompresního zisku. Z testování vyšel nejlépe algoritmus FELICS pro nějž byly navrženy úpravy, které jej činily vhodnějším pro následnou hardwarovou implementaci. Tato úprava algoritmu byla otestována na vícebitových snímcích pořízených infračervenou kamerou.

Návrh implementace byl proveden v prostředí Vivado a otestován v hradlovém poli platformy Zynq-7020 umístěné na vývojové desce ZedBoard. Návrh samotný se ukázal jako principiálně funkční a jeho princip je v práci podrobně popsán. Návrh samotný je velice skromný, co do počtu zdrojů FPGA potřebných pro jeho implementaci. Porovnání času potřebného ke kompresi ukázalo, že hardwarová implementace je oproti sekvenčnímu zpracování algoritmů rychlejší hned několikrát a to zejména pokud se zvětší bitová hloubka obrazu. V poslední části práce bylo identifikováno slabé místo návrhu a byla navržena jeho možná úprava, která by měla vést ke zrychlení komprese.

Filosofie se kterou je návrh vytvořen, jej omezuje na zpracování právě jednoho pixelu za jeden takt (v nejlepším případě). Do budoucna by bylo prospěšné zaměřit se na možnost paralelního spojení několika instancí návrhu, které by toto omezení odstranilo a vedlo by k několikanásobnému zrychlení. Takové spojení by nutně vyžadovalo významnou modifikaci současného návrhu. Pokud by se ale povedlo paralelně spojit třeba jen dvě instance na úkor dalšího nárůstu potřebných zdrojů FPGA, vedlo by to hned ke dvojnásobnému zrychlení komprese.

Použitá literatura

[1] HOWARD, Paul G. a Scott Jeffrey VITTER. *Fast and Ecient Lossless Image Compression* [online]. Data Compression Conference, Snowbird, Utah, 1993. [vid. 1.5. 2020]. Dostupné z: https://core.ac.uk/download/pdf/213393158.pdf

[2] ELIASON, Eric, HEYD, Rodney a Sarah MATSON. *HiRISE EDR, RDR, and DTM Archive Volumes Software Interface Specification* [online]. University of Arizona, 2012. [vid. 1.5. 2020]. Dostupné z: https://hirise-pds.lpl.arizona.edu/PDS/DOCUMENT/HIRISE_EDR_RDR_VOL_SIS.PDF

[3] WITTEN, Ian H., NEAL, Radford M. a John G. CLEARY. *Arithmetic coding for data compresion* [online]. 1987. [vid. 15.5. 2020]. Dostupné z: https://web.stanford.edu/class/ee398a/handouts/papers/WittenACM87ArithmCoding.pdf

[4] HUFFMAN, David Albert. *A Method for the Construction of Minimum-Redundancy Codes** [online]. 1952 [vid. 16.5. 2020]. Dostupné z: https://www.ic.tu-berlin.de/fileadmin/fg121/Source-Coding_WS12/selected-readings/10_04051119.pdf

[5] *Adaptivní Huffmanovo kódování*. Stringology [online]. [vid. 16.5. 2020]. Dostupné z: http://www.stringology.org/DataCompression/ahv/index_cs.html

[6] *Optimální kódování informačního zdroje*. Wikisofia [online]. [vid. 16.5. 2020]. Dostupné z: https://wikisofia.cz/wiki/Optimální_(entropické)_kódování_informačního_zdroje

[7] *CCITT Encoding.* Higher Institute for Computer & Informtion Technology [online]. [vid. 16.5. 2020]. Dostupné z: https://taahmedsror.files.wordpress.com/2012/04/huffman-encoding.pdf

[8] HLAVÁČ, Václav. *Komprese obrazů* [online]. České vysoké učení technické v Praze [vid 16.5. 2020]. Dostupné z: http://people.ciirc.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/81ImageCompressionCz.pdf

[9] HLAVÁČ, Václav. *Digitální obraz, základní pojmy* [online]. České vysoké učení technické v Praze. [vid 16.5. 2020]. Dostupné z: http://people.ciirc.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/014DigitalImageCz.pdf

[10] SAUDEK, Jan. *Komprese dat* [online]. Fakulta informatiky Masarykovy Univerzity [vid. 16.5. 2020]. Dostupné z: https://www.fi.muni.cz/usr/staudek/vyuka/filesys/02_komprese_dat.pdf

[11] SHANON, Claude Elwood. *A Mathematical Theory of Communication* [online]. 1948. [vid. 16.5. 2020]. Dostupné z: http://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf

[12] ŠRÁMEK, Jaromír, RÁČEK, Ondřej, SEDLÁŘ, Martin a Vojtěch MORNSTEIN, *Získávání a analýza obrazové Informace* [online]. Masarykova univerzita v Brně. [vid. 16.5. 2020]. Dostupné z: https://www.med.muni.cz/biofyz/Image/ucebnice.pdf

[13] *Operace s obrazem*. [online]. Biofyzikální ústav LF MU. [vid. 16.5. 2020]. Dostupné z: https://www.med.muni.cz/biofyz/zobrazovacimetody/files/Obraz.pdf

[14] *Vektorová grafika*. Wikisofia [online]. [vid. 16.5. 2020]. Dostupné z: https://wikisofia.cz/wiki/Vektorov%C3%A1_grafika

[15] *Bitmapová grafika*. Wikisofia [online]. [vid 16.5. 2020]. Dostupné z: https://wikisofia.cz/wiki/Bitmapov%C3%A1_grafika

[16] ČÍKA, Petr. *Multimédia* [online]. Vysoké Technické učení v Brně, 2014. [vid. 17.5. 2020]. Dostupné z: https://vut-vsb.cz/predmet-multimedia-41

[17] *Úvod do kompresních algoritmů* [online]. Mendelova univerzita v Brně. [vid. 17.5. 2020]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=7019

[18] KOLÁŘ, Milan. *Popis systémů pomocí VHDL*. Ústav mechatroniky a technické informatiky, Technická univerzita v Liberci. [vid. 19.5. 2020].

[19] IEEE 1076-1987. *Standart VHDL Language Reference Manual* [online]. The Institue of Electrical and Electronics Engineers, New York, NY, USA. [vid. 17.5. 2020]. Dostupné z: https://perso.telecom-paristech.fr/guilley/ENS/20171205/TP/tp_syn/doc/IEEE_VHDL_1076-1987.pdf

[20] NOVÁK, Ondřej a kolektiv. *Číslicová elektronika*. 1. vydání. Liberec: Technická univerzita v Liberci, Fakulta mechatroniky, informatiky a mezioborových studií, 2014. ISBN 978-80-7494-137-5. [vid 19.5. 2020].

[21] ŠUSTA, Richard. *Příkladný úvod do VHDL* [online]. Katedra řídící techniky ČVUT-FEL, 2013. [vid 21.5. 2020]. Dostupné z: http://dcenet.felk.cvut.cz/edu/fpga/doc/PrikladnyUvodDoVHDL.pdf

[22] *7z Format*. 7-zip [online]. [vid. 21.5. 2020]. Dostupné z: https://www.7-zip.org/7z.html

[23] CROCKETT, Louise H., ELLIOT, Ross A., ENDERWITZ Martin A. a Robert W. STEWART. *The Zynq Book*. 1. vydání. Glasgow, Scotland, UK: Depratment of Electronic and Electrical Engineering, University of Strathclyde. ISBN 0992978709. [vid. 23.5. 2020].

[24] *Zynq-7000 SoC Data Sheet: Overview*. XILINX [online]. 2018. [vid 23.5. 2020]. Dostupné z: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[25] AMARA, Marina. *Sergeant Taria, Napoleonic War Veteran*. Marinamaral [online]. 2016. [vid 23.5. 2020]. Dostupné z: https://marinamaral.com/sergeant-taria-napoleonic-war-veteran/

[26] KUBÁK, Radomír. *Technologické pokroky ultrazvukové diagnostiky 2015-2020* [online]. Fakultní nemocnice Brno. [vid 23.5. 2020]. Dostupné z: https://www.fnbrno.cz/data/files/3908.pdf

[27] *Grafické formáty*. Spseke [online]. [vid. 23.5. 2020]. Dostupné z: https://www.spseke.sk/tutor/prednasky/GrafData.html

[28] *Vivado Design Suite User Guide*. Xilinx [online]. 2012. [vid. 23.5. 2020]. Dostupné z: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_2/ug910-vivado-getting-started.pdf

[29] *Co je FPGA a proč je použít?*. Vyvoj.hw.cz [online]. 2019. [vid 23.5. 2020] Dostupné z: https://vyvoj.hw.cz/zaklady-fpga-co-je-fpga-a-proc-je-pouzit.html

[30] *MPEG-2 Video*. MPEG [online]. [vid. 24.5. 2020]. Dostupné z: https://mpeg.chiariglione.org/standards/mpeg-2/video

[31] TRPÁK, Karel. *Stanovisko České televize k volbě kódovacího systému pro digitální televizní vysílání DVB-T*. Česká televize [online]. 2006. [vid. 24.5.2020]. Dostupné z: https://www.ceskatelevize.cz/vse-o-ct/technika/obraz/stanovisko-ceske-televize-k-volbe-kodovaciho-systemu-pro-digitalni-televizni-vysilani-dvb-t/

[32] *Truncated binary encoding*. Wikipedia [online]. [vid. 24.5. 2020]. Dostupné z: https://en.wikipedia.org/wiki/Truncated_binary_encoding

[33] *AMBA® 4 AXI4-Stream Protocol*. ARM [online]. 2010. [vid. 24.5.2020]. Dostupné z: https://static.docs.arm.com/ihi0051/a/IHI0051A_amba4_axi4_stream_v1_0_protocol_spec.pdf

Příloha A

barbara

bridge

clown

couple

crowd

girlface

lenna

ligthouse

tank

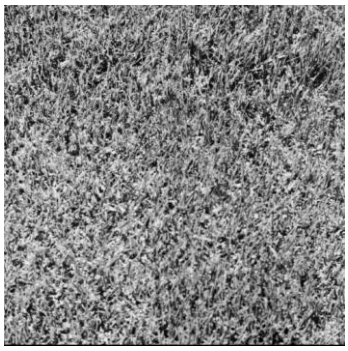
truck

trucks

zelda

Příloha B

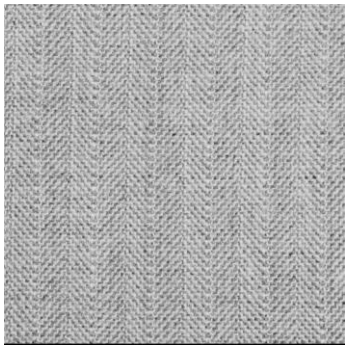
a



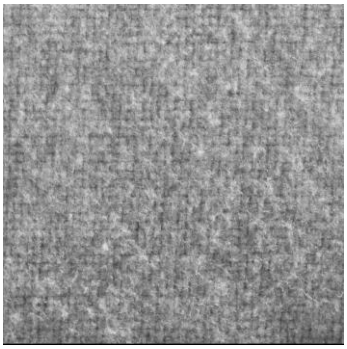
b

c

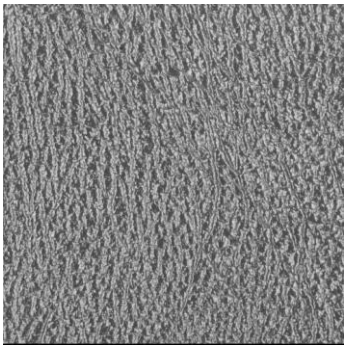
d



e

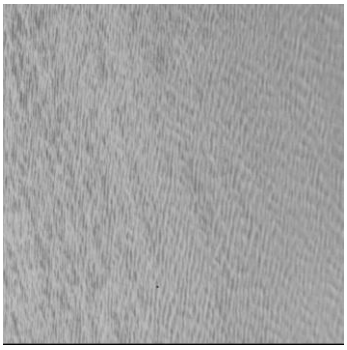


f

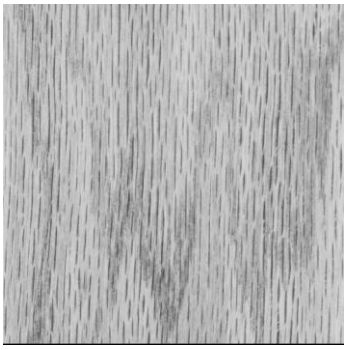


g

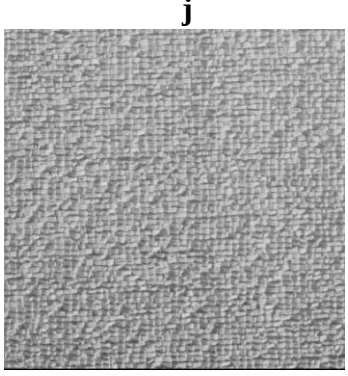
h



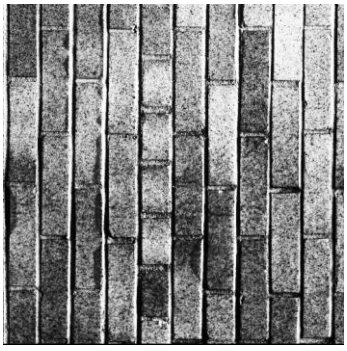
i



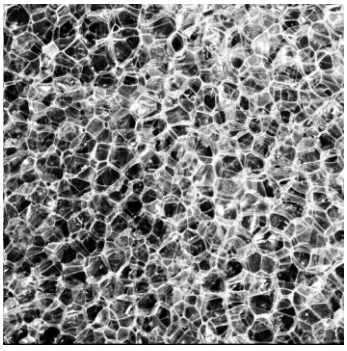
j



k

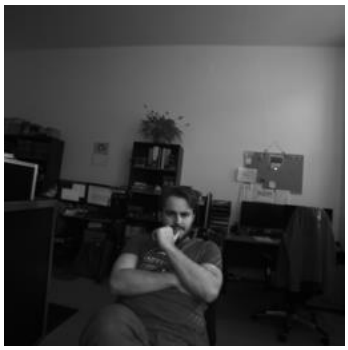


l



Příloha C

A

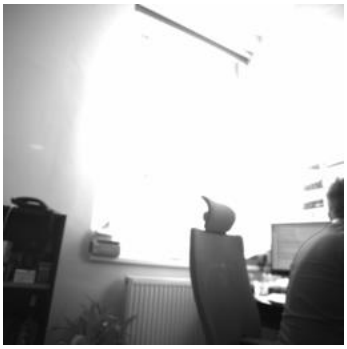


B

C

D

E



F

Obsah CD

FELICS implementace – *Složky s projekty pro prostředí VIVADO.*

- IP – *celé IP jádro*
- Project_000 – *propojené IP jádro s DMA*

Kompresní algoritmy – *Spustitelné programy + zdrojové kódy*

- DELTA-pro-mereni-casu – *pouze pro obrazy s rozlišením 256x256*
- DELTA
- FELICS1
- FELICS2
- FELICS3-pro-mereni-casu – *pouze pro obrazy s rozlišením 256x256*
- FELICS3
- HUFFMAN
- RLE1
- RLE2
- Zdrojové kódy – *složka se zdrojovými kódy v jazyce C*

Statistika – *Excel tabulky.*

- Časy
- Statistika sady hlavní – *složka s měřením pro 16/12/8bitové obrazy a měření pro obrazy v rozlišení 256x256*
- Statistika sady běžné výjevy
- Statistika sady šum

Testovací vzorky – *Složky* s *obrazy ve formátu PGM.*

- Sada běžné výjevy
- Sada hlavní 256x256 (8bit)
- Sada hlavní 256x256 (16bit)
- Sada hlavní 2048x2048 (8bit)
- Sada hlavní 2048x2048 (12bit) – *nelze správně otevřít v daném formátu*
- Sada hlavní 2048x2048 (16bit)
- Sada šum

Text

- ulmanD20_BP.pdf – elektronický text této práce.