Izvedba in optimizacija hitre Fourierjeve transformacije v Vitis HLS za obdelavo radarskih podatkov

Luka Podbregar, Andrej Trost, Aljaž Blatnik, Andrej Žemva, Boštjan Batagelj

Univerza v Ljubljani, Fakulteta za elektrotehniko
E-pošta: luka.podbregar@fe.uni-lj.si

Implementation and Optimization of Fast Fourier Transformation in Vitis HLS for Radar Data Processing

This paper presents the implementation and optimization of the Cooley-Tukey Fast Fourier Transform (FFT) for application on FPGA boards. We used the Vitis HLS tool to build digital circuits at the register-transfer level (RTL) using the C/C++ high-level programming language. Furthermore, the tool allowed for the optimization of the implemented algorithm using directives.

The FFT is essential for implementation of radar systems. It enables the transformation of radar data from time to frequency domain, which allows for simpler analysis and recognition of targets in radar systems. We used directives to optimise algorithm execution, aiming to reduce latency and resource utilisation on the FPGA. With the low-latency solution achieved, it is possible to use FPGA implementations of the FFT for radar systems with higher sample rates. This allows for more accurate target acquisition.

1 Uvod

V prispevku je opisana izvedba in optimizacija Cooley-Tukey hitre Fourierjeve transformacije (angl. Fast Fourier Transform - FFT) za uporabo v programabilema polju logičnih vrat (angl. Field Programmable Gate Arrays - FPGA). Uporabljeno je bilo orodje Vitis HLS, ki omogoča enostavno načrtovanje digitalnih vezij na ravni prenosa in transformacije podatkov med registri (angl. Register-transfer level - RTL) v visokonivojskem programskem jeziku C/C++. Orodje dovoljuje tudi optimizacijo izvedenih algoritmov z uporabo direktiv.

FFT je ključna operacija za pretvorbo radarskih podatkov iz časovne v frekvenčno domeno, kar omogoča analizo in razpoznavo tarč v radarskih sistemih. Uporabljene direktive so bile namenjene optimizaciji izvajanja algoritma FFT s ciljem zmanjšanja zakasnitev in izrabe virov na FPGA. Z doseženo nizko latenčno rešitvijo je možna uporaba FPGA izvedba FFT za radarske sisteme z višjimi vzorčnimi frekvencami, kar omogoča bolj natančno zaznavanje tarč.

2 Vitis HLS

Vitis, orodje za visokonivojsko sintezo (angl. High-level synthesis - HLS) ki ga ponuja Xilinx, omogoča preprosto načrtovanje FPGA algoritmov. Temelji na sintezi visokonivojske C/C++ kode v model digitalnega vezja na nivoju RTL. Takšno načrtovanje algoritmov je hitrejše in manj kompleksno kot neposredno načrtovanje v RTL, kar zmanjšuje možnosti za napake. Izhodni model vezja je v jeziku VHDL ali Verilog.

S tem orodjem načrtujemo digitalna vezja, prilagojena specifičnim primerom uporabe, kar omogoča optimalno izvedbo v smislu zakasnitev in izrabe namenskih ter splošnih enot naprave, kot so registri, vpogledne tabele (angl. Lookup table - LUT), flip-flopi (FF), bloki statičnega pomnilnika (BRAM) in enote za digitalno obdelavo signalov (angl. Digital Signal Processor - DSP). Program razvijalcu omogoča vstavljanje direktiv oziroma navodil za nadzor procesa sinteze v visokonivojsko kodo. Te omogočajo optimizacijo generirane RTL kode s prilagajanjem različnih vidikov načrtovanja, s čimer lahko razvijalec izboljša učinkovitost, zmogljivost in porabo virov na ciljni FPGA napravi.

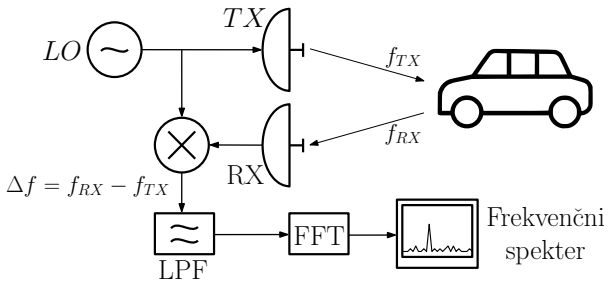
Glavne metrike za optimizacijo algoritmov v Vitis HLS so zmogljivost, zakasnitve (latenca in interval ponovitve), poraba virov in energijska učinkovitost. Zmogljivost se nanaša na število operacij, ki jih sistem lahko izvede v določenem času, kar vpliva na hitrost izvajanja algoritma. Latenca je čas, ki je potreben za dokončanje ene operacije od začetka do konca. Poraba virov vključuje količino logičnih enot, pomnilniških blokov in drugih FPGA virov, ki jih algoritem uporablja. Energijska učinkovitost pa se nanaša na porabo energije glede na izvedeno delo. Optimizacija teh metrik lahko vodi do hitrejših, bolj učinkovitih in ekonomičnih strojnih rešitev.

3 Radarski modul

Prvotno vojaški sistem za zaznavanje sovražnika se je v zadnjih desetletjih razširil na številna nova področja. Med njimi so avtomobilski radarji [1], senzorji za nadzor prometa [2], senzorji za zaznavo in prekljapljanje luči ali vrat, nadzor avtomatizacije [3], radarji za uporabo v medicini [4], slikanje reliefa [5], zaznavanje vremenskih pojavov [6] ali pa astronomskih objektov [7]. Razvoj in množična proizvodnja integriranih vezij omogočata, da

lahko radarski čip kupimo za nekaj evrov in ga uporabimo kot Dopplerjev radar.

Dopplerjev radar deluje na principu Dopplerjevega pojava, ki je značilen za vsako valovanje, pri katerem se oddajnik ali sprejemnik valovanja premika [8]. V primeru stacionarnega Dopplerjevega radarja je oddajnik stacionaren in meri premikajoče se tarče. Blokovni načrt na sliki (slika 1) prikazuje delovanje preprostega stacionarnega Dopplerjevega radarskega sistema. Če se tarča premika v smeri radarskega sistema, je frekvenca sprejetega signala višja od frekvence oddanega signala. Če se tarča premika stran od radarskega sistema, je frekvenca sprejetega signala nižja od oddane frekvence.



Slika 1: Blokovni načrt delovanja preprostega Dopplerjevega radarja, kjer je LO lokalni oscilator, TX oddajnik, RX sprejemnik in LPF nizko prepustno sito.

Radarske sisteme delimo glede na obliko radarskega signala, ki je odvisna od želenih lastnosti radarskega sistema. Oblika signala se večinoma prilagaja glede na željeno ločljivost izmerjene razdalje in hitrosti. V grobem jih delimo na dve skupini: impulzne radarje in radarje s konstantno oddajo (angl. Continuous-Wave - CW). V obeh primerih lahko radar uporablja signal s konstantno frekvenco ali pa frekvenčno oziroma fazno moduliran signal. Čeprav so frekvenčno modulirani CW radarski sistemi (angl. Frequency-Modulated Continuous-Wave - FMCW) zelo priljubljeni, je bil za namen tega seminarja uporabljen nemoduliran CW radarski sistem.

FMCW omogoča zaznavo relativne hitrosti in razdalje do tarče, medtem ko nemoduliran CW omogoča zaznavo samo relativne hitrosti tarče. Ker je bil radarski sistem načrtovan za nadzor prometa, kjer sta pomembni samo relativna hitrost vozila in velikost odmevne površine (angl. Radar Cross-Section - RCS), je bil izbran nemoduliran CW radarski signal. Velikost RCS se lahko izračuna iz moči sprejetega odbitega signala. Prednost uporabe nemoduliranega CW je v tem, da FMCW potrebuje dodatno zunanjo fazno uklenjeno zanko, kar podraži in zakomplicira izvedbo končnega produkta. Radarski sistem, uporabljen za namen tega seminarja, je bil izveden z uporabo Infineonovega 24 GHz radarskega čipa BGT24LTR11N16.

Radarski čip krmili mikrokrmilnik STM32F301K8U [9], ki preko USART povezave pošilja podatke do čipa CH340G. Ta nato preko serijske komunikacije pošilja podatke do osebnega računalnika, kjer Python skripta zajema podatke in jih shranjuje v CSV datoteke.

4 Izvedba FFT

4.1 Teoretično odzadje

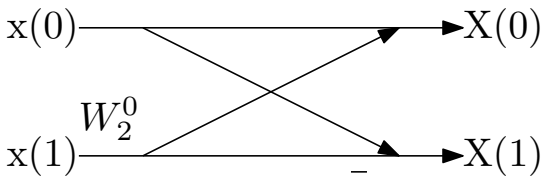
Razpoznavo zaznanih radarskih tarč se najlažje izvede v frekvenčni domeni. Podatke, ki jih zajamemo z radarskim sistemom, moramo zato pretvoriti iz časovne v frekvenčno domeno, kar naredimo s Fourierjevo transformacijo. Za pretvorbo diskretnih signalov se uporablja diskretna Fourierjeva transformacija (DFT). Pretvorbo N števila časovnih vzorcev x_n v frekvenčne vzorce X_k prikazuje (1).

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-j2\pi kn}{N}} \tag{1}$$

Izračun DFT je računsko zahteven in zato neprimeren v primerih, kjer želimo čim hitrejše in optimizirano delovanje kode, kot je v primeru izvedbe na FPGA. Za takšne primere uporabe je bolj primeren eden izmed algoritmov FFT. Ta omogoča zniževanje računske zahtevnosti z izkoriščanjem simetrij, ki nastanejo pri računanju DFT. Najpogosteje uporabljen FFT algoritem je Cooley-Tukey FFT [10], ki računsko zahtevnost za obdelavo N števila vzorcev zmanjša iz $O(N^2)$ na $O(Nlog(N))$.

Cooley-Tukey FFT algoritem temelji na delitvi DFT na manjše DFT. Najpreprostejša delitev vhodnih podatkov je na dva dela, sode in lihe podatke, kar omogoča izkoriščanje simetrije pri računanju FFT. Sortiranje na sode in lihe elemente se lahko izvede z reverzijo bitov. Uporaba slednje, poleg dodatne pohitritve zaradi izrabe simetrije pri računanju, omogoča, da so izhodni podatki FFT v pravilnem vrstnem redu. Sodi in lihi elementi se nato delijo na posamezne pare elementov, nad katerimi izvedemo DFT za dva elementa, in jih nato med sabo kombiniramo. Iz izvedbe algoritma po posameznih parih podatkov izhaja pogoj za uporabo FFT algoritma: število vhodnih podatkov mora biti potenca števila 2.

Slika (slika 2) prikazuje t.i. metulj, ki prikazuje izračun DFT dveh elementov, kjer je utež (angl. Twiddle factor) $W_N^n = e^{\frac{-j2\pi kn}{N}}$. Stičišče dveh črt predstavlja seštevanje oziroma odštevanje, če se poleg stičišča nahaja znak minus, ki predstavlja negacijo spremenljivke nad znakom, med spremenljivkama. Vsaka spremenljivka, ki se nahaja nad črto, predstavlja množenje med dvema vrednostima. Prikaz v obliki metulja je zaradi svoje kompaktnosti pogosto uporabljen za prikaz izračunov FFT v svetu digitalnega procesiranja signalov.



Slika 2: Primer prikaza DFT dveh elementov v obliki metulja.

Orodje Xilinx Vivado sicer ponuja svojo izvedbo FFT kot jedro intelektualne lastnine (angl. Intellectual

Property Core - IP), kjer se z nastavitvijo nekaj parametrov prilagodi izvedbo za željen primer uporabe, vendar ne omogoča takšne prilagodljivosti izvedbe, kot jo omogoča izvedba v Vitis HLS. S slednjim lahko dosežemo drugačne rezultate kot narejeni IP, zahteva pa delo kar nekaj poznavanja orodja in optimalnega kodiranja za sintezo.

4.2 Optimizacija FFT

Osnovno računanje FFT je sekvenčno, metulj za metuljem. Takšna izvedba omogoča hkratno obdelavo elementov, ki jih vsebuje en metulj. Obdelavo se lahko pohitri s paralelno izvedbo. Izvedba Cooley-Tukey FFT na FPGA omogoča paralelizacijo določenih delov algoritma. Ta je lahko izvedena na več načinov [11], kot je paralelizacija s povečanjem velikosti metulja, paralelizacija z računanjem več metuljev hkrati in paralelizacija z računanjem več stopenj hkrati. Izbira vrste paralelizacije je odvisna od zmožnosti ciljne naprave in želja glede zakasnitev izvajanja ter izrabe virov ciljne naprave. Vsaka izmed strategij namreč različno vpliva na izrabo virov FPGA.

Vitis HLS omogoča uporabo vrsto različnih strategij paralelizacije [12]; podrobneje bodo opisane le tiste, ki so bile uporabljene v izvedbi tega seminarja. Najpogostejši tip paralelizacije je z uporabo direktive $pipeline$, ki omogoča cevovodno izvedbo funkcij in zank. S paralelno izvedbo večih zaporednih ponovitev funkcije ali zanke zmanjša začetni interval (angl. Initiation Interval - II). II predstavlja čas med obdelavo novih vhodnih podatkov in se meri v urinih ciklih. Vitis HLS privzeto strmi k minimalnem možnem II, omogoča tudi izbiro poljubne vrednosti II. Podobno deluje direktiva $dataflow$, ki omogoča cevovodno izvedbo več funkcij ali zank hkrati. Direktiva za razvoj zank $unroll$ omogoča paralelno izvedbo zank. To naredi s kopiranjem telesa zanke v poljubno število kopij, kar omogoča izvajanje večih zank hkrati. Pri paralelizaciji so pomembne tudi direktive, ki se nanašajo na shranjevanje spremenljivk v pomnilnik in dostop do njih. Ena izmed njih je $array\_partition$, ki se nanaša na nize števil (angl. Array). Te razdeli v manjše enote pomnilnika, ki so med seboj neodvisne, kar omogoča istočasen dostop do podatkov v pomnilniku in zmanjšuje t.i. ozko grlo. Uporaba te direktive posledično omogoča večjo paralelizacijo algoritmov, kjer je pomemben dostop do podatkov v pomnilniku. Omogoča uporabo več različnih tipov enot pomnilnika poljubnih dolžin, kot sta blokovni tip in ciklični tip.

Izvedbo FFT lahko dodatno pohitrimo s predhodnim računanjem koeficientov uteži W_N^n, ki jih lahko shranimo v pomnilnik. Do njih dostopamo pri računanju posameznih metuljev in so v tej izvedbi, zaradi uporabe kompleksnih vhodnih vrednosti, definirani kot dva statična arraya, eden za realna in eden za kompleksna števila, oba velikosti 512 elementov. Z definiranjem spremenljivk kot statične spremenljivke se izognemo nepotrebnim inicializacijam spremenljivk v izvedbi funkcij. Za izračun FFT potrebujemo $\log_2(N)$ stopenj, vsako z $\frac{N}{2}$ metulji. To za uporabljen primer FFT velikosti 512 elementov z dvema vhodnima nizoma števil pomeni 9 stopenj in 4608 metuljev. Vhodni podatki za FFT in podatki za ovrednotenje rezultatov se, v namen tega seminarja, v izvedbo prenašajo iz datoteke tipa .dat, v katero se s Python skripto zapiše radarske podatke. Branje datotek tipa .dat omogoča C++ knjižnica cstdio. Za uporabo na pravi ploščici FPGA bi bil uporabljen vmesnik, ki omogoča uporabo toka podatkov.

Če želimo optimizirati algoritem, ki se bo izvajal na FPGA, je zelo pomembna izbira tipa spremenljivk. Vitis HLS omogoča izbiro med klasičnimi C/C++ tipi kot so int, $double$ ali $float$. Omogoča pa tudi vključitev knjižnic, ki omogočajo uporabo tipov spremenljivk s poljubno natančnostjo, kot sta ap_int in ap_fixed. Slednje je število s fiksno vejico, kar pomeni, da ima fiksno število bitov pred in za decimalno vejico. Primerjavo uporabe spremenljivk s fiksno vejico in spremenljivk s plavajočo vejico prikazuje tabela 1 [13], [14]. Z uporabo števil s fiksno vejico je možno znižati latenco izvajanja, ceno končnega izdelka in povečati energijsko učinkovitost za ceno nižje natančnosti, nižjega dinamičnega območja in daljšega časa načrtovanja algoritma. Slednji je posledica predvsem kvantizacijske napake zaradi zaokroževanja števil, kar zahteva posebno pozornost pri načrtovanju algoritma. Dodatno se z računanjem s spremenljivkami s fiksno vejico izognemo prelivu zaradi predčasno definirane dolžine bitov za rezultat računske operacije.

Tabela 1: Primerjava med števili s fiksno vejico in števili s plavajočo vejico.

Lastnosti	**Fiksna vejica**	**Plavajoča vejica**
Cena procesorjev	Nižja	Višja
Čas načrtovanja algoritma	Daljši	Krajši
Dinamično območje	Nižje	Višje
Energijska učinkovitost	Višja	Nižja
Latenca operacij	Nižja	Višja
Natančnost	Nižja	Višja
Preprostost uporabe	Slabša	Boljša

Algoritem FFT je bil v končni verziji razdeljen na tri funkcije: funkcijo prve stopnje FFT, funkcijo zadnje stopnje FFT in funkcijo vmesnih stopenj. Ta delitev je omogočila izogib nepotrebnim izračunom in posledično pohitritev izvedbe. Za dodatno pohitritev se je pred izračunom FFT izvedla še reverzija bitov.

5 Rezultati

Kot omenjeno, je optimizacija izvedbe FFT temeljila na uporabi različnih direktiv. Pravilnost izvedbe je bila preverjena z izračunom povprečne kvadratne napake (angl. Root Mean Square Error - RMSE) med izhodom algoritma izvedenim v Vitis HLS in izhodom FFT algoritma iz Python knjižnice Numpy. Oba algoritma sta operirala z istimi vhodnimi podatki, zajetimi z radarskim modulom. Ker so bili vhodni in izhodni podatki kompleksna števila, je bila za izračun RMSE uporabljena absolutna vrednost.

Tabela 2: Rezultati optimizacije izvedbe FFT algoritma v Vitis HLS. FPGA plošča Zynq 7000 zc706 ima na voljo 1090 enot BRAM, 900 DSP, 437200 FF in 218600 LUT.

Uporabljene direktive	**Latenca** $[\mu s]$	**Št. BRAM**	**Št. DSP**	**Št. FF**	**Št. LUT**
Števila s fiksno vejico brez direktiv	53,81	18	16	1842	3104
+Reverzija bitov: UNROLL	46,11	18	16	2057	15312
+Reverzija bitov: ARRAY_PARTITION block factor=4	44,00	16	16	2691	18617
+Metulj FFT: UNROLL	39,34	16	16	16476	104931
+Stopnja FFT: ARRAY_PARTITION block factor=4	36,06	0	64	35783	160138
+FFT: DATAFLOW	35,97	0	64	36270	157413
Števila s plavajočo vejico z vsemi direktivami	68,91	290	351	297274	287114

RMSE se z uporabo različnih direktiv ni razlikoval, se je pa razlikoval, kadar je bil spremenjen tip uporabljenih števil. Kadar so bila uporabljena števila s plavajočo vejico, je bil $RMSE = 0,00036$. Z uporabo ap_fixed z 1 bitom pred in 3 biti za decimalno vejico, pa je bil $RMSE = 0,69$. RMSE z uporabo števil s poljubno natančnostjo sicer precej naraste, se pa izvedba FFT precej pospeši. Povečevanje števila bitov pred in za decimalno vejico le rahlo spremeni RMSE, zato je v končni izvedbi uporabljeno število z 1 bitom pred in 3 biti za decimalno vejico, ki omogoča izvedbo z najmanjšo latenco.

Tabela 2 prikazuje rezultate optimizacije FFT izvedbe, testirane na FPGA plošči AMD Zynq 7000 ZC706. Izbrana je bila perioda ure 10 ns. Vsaka zaporedna direktiva je bila dodana verziji s prejšnjimi direktivami. V zadnji vrstici je prikazan še rezultat sinteze z uporabljenimi števili s plavajočo vejico in vsemi izbranimi direktivami. Prikazane so le direktive, ki so izboljšale izvedbo algoritma.

Za delitev nizov števil se je najbolj izkazal blokovni tip s faktorjem 4, kar predstavlja delitev v štiri nove nize števil. Uporaba direktive za razvoj zank je pospešila izvedbo algoritma, je pa zelo povečala izrabo FF in LUT enot ploščice FPGA. Dosežena najnižja latenca izvedbe z uporabo direktiv je bila 35,97 µs. FFT algoritem iz Python knjižnice Numpy je za izvedbo transformacije istih podatkov potreboval 5,76 ms. Optimizirana izvedba Cooley-Tukey FFT algoritma omogoča hitro transformacijo radarskih meritev iz časovne v frekvenčno domeno. Nizka latenca optimiziranega algoritma dovoljuje uporabo radarskih sistemov z višjimi vzorčnimi frekvencami, ki lahko dosežejo višjo natančnost zaznavanja tarč.

6 Zaključek

Xilinxovo programsko okolje Vitis HLS omogoča preprosto izvedbo namenskih FPGA algoritmov v visokonivojskem C/C++ jeziku, ki se nato s sintezo prevedejo v nizkonivojske RTL programske jezike VHDL ali Verilog. S tem se izognemo kompleksnosti in večji možnosti napak pri neposrednem načrtovanju v RTL.

V tem prispevku je bil izveden Cooley-Tukey FFT algoritem za pretvorbo radarskih podatkov iz časovne v frekvenčno domeno. Algoritem je bil optimiziran z uporabo direktiv, ki omogočajo paralelizacijo prvotno sekvenčnega algoritma. Rezultati so predstavljeni v tabeli 2, kjer sta prikazani latenca in izraba virov glede na uporabljene direktive. Izbira tipa številskih vrednosti se je izkazala za zelo pomembno. Za končno rešitev je bil izbran tip števil s fiksno vejico, kar omogoča znižanje latence izvajanja, cene končnega izdelka in povečanje energijske učinkovitosti, vendar na račun nižje natančnosti, manjšega dinamičnega območja in daljšega časa načrtovanja algoritma.

Algoritem je bil preverjen na podatkih, zajetih z radarskim modulom, in primerjan z rezultatom FFT algoritma iz Python knjižnice Numpy. Doseženi rezultat optimizacije FFT algoritma zaradi nizke latence omogoča uporabo FPGA izvedbe v radarskih sistemih z višjimi vzorčnimi frekvencami, kar bi lahko izboljšalo natančnost zaznavanja tarč.

7 Zahvala

Delo je nastalo pod okriljem projekta J2-50072 in programa P2-0246 Javne agencije za znanstvenoraziskovalno in inovacijsko dejavnost Republike Slovenije.

Literatura

[1] J. Dickmann et al., "Automotive radar the key technology for autonomous driving: From detection and ranging to environmental understanding," 2016 IEEE Radar Conference (RadarConf), Philadelphia, PA, USA, 2016, str. 1-6.

[2] J. Sánchez-Oro, D. Fernández-López, R. Cabido, A. S. Montemayor, and J. J. Pantrigo, "Radar-based road-traffic monitoring in urban environments," Digital Signal Processing, vol. 23, no. 1, Jan. 2013, str. 364–374.

[3] BGT24LTR11 Product Brief. [Online]. Dosegljivo: www.infineon.com/dgdl/Infineon-BGT24LTR11-PB-v0100-EN.pdf?fileId=5546d4625696ed7601569d063799153e. [24. 06. 2024].

[4] S. Pisa, E. Pittella and E. Piuzzi, "A survey of radar systems for medical applications," IEEE Aerospace and Electronic Systems Magazine, vol. 31, no. 11, November 2016, str. 64-81.

[5] W. M. Brown and L. J. Porcello, "An introduction to synthetic-aperture radar," IEEE Spectrum, vol. 6, no. 9, Sept. 1969, str. 52-62.

[6] R. J. Doviak, D. S. Zrnic and D. S. Sirmans, "Doppler weather radar," Proceedings of the IEEE, vol. 67, no. 11, Nov. 1979, str. 1522-1553.

[7] K. v. Klooster, M. Petelin, M. Thumm and M. Aloisio, "Ka-band groundstation antenna aspects for deep space telecommunication and radar," 2013 7th European Conference on Antennas and Propagation (EuCAP), Gothenburg, Sweden, 2013, str. 242-246.

[8] E. J. Barlow, "Doppler Radar," Proceedings of the IRE, vol. 37, no. 4, April 1949, str. 340-355.

[9] STM32F301K8 Product overview. [Online]. Dosegljivo: https://www.st.com/en/microcontrollers-microprocessors/stm32f301k8.html. [24. 06. 2024].

[10] J. W. Cooley, and J. W. Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series." Mathematics of Computation, vol. 19, no. 90, 1965, pp. 297–301.

[11] H. Almorin, B. Le Gal, J. Crenne, C. Jego and V. Kissel, "High-throughput FFT architectures using HLS tools," 2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Glasgow, United Kingdom, 2022, str. 1-4.

[12] Vitis High-Level Synthesis User Guide. [Online]. Dosegljivo: https://docs.amd.com/r/en-US/ug1399-vitis-hls. [24. 06. 2024].

[13] D. Menard, D. Chillet, O. Sentieys, "Floating-to-Fixed-Point Conversion for Digital Signal Processors," EURASIP J. Adv. Signal Process, 2006, 096421 (2006).

[14] M. N. Joshi and D. H. Gawali, "Floating point unit core for Signal Processing applications," 2016 Online International Conference on Green Engineering and Technologies (IC-GET), Coimbatore, India, 2016, str. 1-6.