Načrtovanje lidarja s konstantno modulacijo svetlobe s programskim orodjem Vivado HLS

Luka Pogačnik, Andrej Trost, Andrej Žemva, Marko Munih

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

Designing a CW lidar with Vivado HLS

Abstract. *This article talks about design and validation of a continuous wave (CW) lidar application on an FPGA board. VHDL code was generated using Vivado HLS software. Due to the nature of the problem, each individual sub component was designed and validated individually and then joined to form a single cohesive software design. To optimize power draw and speed requirements, a VHDL controller element was coded to govern the flow lidar sensing. Software was written with deployment on Red Pitaya development board in mind.*

1 Uvod

Prispevek govori o načrtovanju in validaciji programske opreme za lidar s konstantno modulacijo svetlobe (CW lidar) na FPGA plošči. VHDL koda je bila generirana s pomočjo programskega paketa Vivado HDL. Zaradi narave delovanja CW lidarja je bila vsaka podkomponenta razvita ter validirana posamično, na koncu pa so bile združene v kohezivno celoto. Za optimizacijo porabe energije ter potrebe po hitrosti delovanja, je bil nadzorni element napisan neposredno v jeziku VHDL. Čeprav je bilo razvitih že več lidarjev, ki delujejo na FPGA ploščah [1][2], so se lotili razvoja lastnega sistema. Programska oprema je bila načrtovana z mislijo na uporabo z razvojnim orodjem Red Pitaya.

2 Ozadje delovanja lidarjev

Lidar deluje na podlagi merjenja časa od oddaje svetlobnega pulza do prejema odbite svetlobe. Zvezo med oddaljenostjo do tarče (d), časom preleta (t) ter hitrostjo propagacije svetlobe v mediju (c) podaja enačba (1).

$$d = (t \cdot c)/2 \tag{1}$$

Groba ocena pove, da svetloba za vsak milimeter potrebuje 3,3 ps, torej 6,6 ps na milimeter za povratno pot. Merjenje z ločljivostjo v milimetrskem redu se izkaže za tehnološko velik, ne pa nerešljiv izziv. Za FPGA čip Zinq 7010, ki smo ga izbrali za našo aplikacijo, je bil izdelan časovnik, ki je lahko meril z ločljivostjo 22 ps [3].

Namesto izdelave preciznega časovnika se lahko čas preleta svetlobe meri tudi posredno, in sicer na podlagi faznega zamika (θ) med oddajano in prejeto modulirano svetlobo. Meritev je odvisna tudi od modulacijske frekvence (f), kot to opisuje enačba (2)

$$d = (c \cdot \theta)/(4 \cdot \pi \cdot \mathrm{f}) \tag{2}$$

Ne glede na realno oddaljenost tarče, bo meritev CW lidarja vedno prikazala meritev znotraj valovne dolžine moduliranega signala (r), ki ga opisuje enačba (3).

$$r = c/(2 \cdot \mathrm{f}) \tag{3}$$

Iz enačb je razvidno, da nižja modulacijska frekvenca omogoča velik domet, prednost višje modulacijske frekvence pa je večja natančnost oziroma manjša občutljivost na fazni šum (enaka napaka v meritvi faze prinese manjšo napako v meritvi razdalje).

Enolično določanje razdalje izven tega intervala je s CW lidarjem možno, če meritev izvedemo pri uporabi več modulacijskih frekvenc. Pravo razdaljo tarče nato določimo tako, da ugotovimo, katera razdalja odgovarja dobljenim meritvam [4]. Za največji domet je smotrno, da so si modulacijske frekvence čim bolj tuje.

2.1 Meritev fazne razlike

Merjenje fazne razlike se lahko realizira z IQ demodulatorjem. Sistem generira dva signala, sinus in kosinus, pri modulacijski frekvenci. Sinusni signal gre na oddajnik, prejeti signal pa se pomnoži enkrat s sinusom s sinusom, pri čemer dobimo I komponento, drugič pa s kosinusom, da dobimo Q komponento. I in Q komponenti gresta skozi nizko prepustno sito, po zadostnem času vzorčenja pa se fazno razliko izračuna kot $atan2(Q, I)$.

3 Struktura sistema

Sistem je bil zasnovan za delovanje na razvojni plošči Red Pitaya. V njenem jedru se nahaja Xilinxov Zynq-7010 FPGA, ima pa tudi po dva kanala 14-bitnih AD ter DA pretvornikov z vzorčno frekvenco 125 MHz. Poleg tega ima tudi zadostno število digitalnih izhodov, ki jih lahko uporabimo za diskretno modulacijo signala. Digitalni sistem smo načrtovali z visokonivojskim HLS [5].

Sistem je bil prvotno zastavljen v enem kosu. Glavna funkcija za sintezo v Vivado HLS je dobila podatek, kako dolgo naj modulira ter vzorči signal, ob koncu izvajanja pa vrne fazno razliko do tarče. Tovrstna struktura je primerna samo za simulacijo, saj ne dopušča kontrole nad časi izvajanja posameznih delov funkcije.

Inicializacija se lahko izvede poljubno počasi, saj ni časovno kritična. Modulacija ter demodulacija se morata začeti na vsak urin cikel, pri čemer pa ni potrebe, da se izvede do konca. Enako velja za filtriranje I ter Q komponent. Zadnji korak, izračun faze s funkcijo $atan2$, se mora začeti izvajati šele, ko vse predhodne komponente končajo i izvajanjem, od tam naprej pa se lahko funkcija izvede poljubno počasi. Sistem mora komunicirati tudi z vhodno-izhodnimi enotami za oddajanje ter sprejemanje signala.

Takšnega sistema se s HLS ne da realizirati v eni sami funkciji, zato smo problem razbili na posamezne logične enote. Te smo individualno validirali, nato pa združili v delujočo celoto.

Za namen testiranja smo vhodno-izhodne enote nadomestili z nastavljivo zakasnilno linijo, s čimer smo lahko pravilnost delovanja preverili tudi v celoti programsko.

3.1 Nadzornik

Nadzornik je komponenta, ki skrbi za tok izvajanja meritve. Kot parameter prejme trajanje vzorčenja N, ima pa še vhode za začetek vzorčenja ter vnovični zagon sistema. Od signala za začetek vzorčenja ob vsakem urinem ciklu izhodni števec poveča za 1 do maksimalne vrednosti 127, po čemer se ponastavi na 0. Ko se cikel ponovi 2^N-krat, na izhodu sporoči, da je bilo doseženo zadostno število ciklov oddajanja, sistem čaka, da z delovanjem končajo še ostali deli lidarja.

Nadzornik kot komponenta je po naravi primerna za realizacijo v FPGA in dokaj enostavna za kodiranje, zaradi česar je bila napisana neposredno v VHDL. S tem smo imeli nad njenim delovanjem popoln nadzor.

3.2 Modem

Modem kot vhod iz nadzornika prejme števec, ki ga uporabi kot parameter za generacijo sinusa ter kosinusa. 7-bitni števec, ki teče na intervalu $[0,128)$, se smatra kot parameter za kotni funkciji $[0,2\pi)$.

Poleg tega iz AD pretvornika prejme tudi amplitudo prejetega signala. Tega pomnoži s sinusom ter kosinusom, da generira pripadajoči I ter Q komponenti.

Velikost sinusa ter kosinusa je 14b predznačeno celo število, enako velikosti AD ter DA pretvornika Red Pitaye, platforme, na kateri je namen realizirati naš sistem. Komponenti I ter Q sta predznačeni 27b celi števili, tako da na prvem koraku še ne izgubljamo ločljivosti.

Za modulacijsko frekvenco smo sprva izbrali $1\ MHz$. Maksimalna razdalja, preden se začne signal ponavljati, bi bila torej malo manj kot $150\ m$. Red Pitaya ima AD ter DA pretvornik, ki delujeta s $125\ MHz$. To pomeni, da bi bila vsaka perioda sinusa ter kosinusa sestavljena iz 125 vzorcev. Za poenostavitev nadaljnjih procesov smo se odločili, da bomo uporabili 128 vzorcev na periodo, s čemer je modulacijska frekvenca padla na $0{,}977\ MHz$.

Sinus ter kosinus bi se v teoriji dalo sproti generirati s CORDIC algoritmom, je pa ta nadvse procesno potraten. Bolj smotrno je uporabiti vpogledno tabelo. Za sinus s 128 vzorci na periodo potrebujemo le 32 vzorcev, preostanek pa dosežemo s preslikavami po kvadrantih. Dodaten bonus je to, da se isto tabelo uporabi tudi za kosinus, kjer parameter le povečamo za $\pi/2$ (32 vzorcev) in poskrbimo, da števec ostane na intervalu $[0,128)$. Pri uporabi 7b števca, je to zagotovljeno že samo od sebe. Zaradi očitnih prednosti uporabe vpogledne tabele, smo se odločili za ta pristop.

Simulacije so pokazale, da je vseeno, če oddajamo signal s sinusno modulacijo, ali pa le pripadajoč dvonivojski signal (prižgan/ugasnjen), saj filter odstrani visokofrekvenčne komponente signala. To poenostavi tudi izhodno stopnjo, saj lahko le preprosto odpiramo in zapiramo tranzistor, brez potrebe po zvezni regulaciji.

3.3 Nizkoprepustni filter

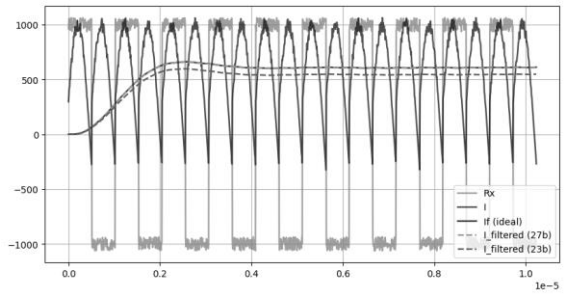
Teorija pravi, da lahko povprečno vrednost periodičnega signala določimo s povprečenjem preko ene periode signala. V teoriji bi torej lahko I ter Q komponenti poslali skozi filter za računanje tekočega povprečja 128 elementov (kolikor je elementov v eni periodi sinusa). Signal bi morali oddajati le tako dolgo, da se zajame ena perioda odbite svetlobe. V našem primeru, kjer smo pričakovali, da bo amplituda odbite svetlobe padla pod šumni nivo še preden presežemo razpon za enolično določljivo razdaljo, bi torej zagotovo zadoščalo oddajanje svetlobe preko dveh period.

Simulacija to teorijo potrdi, s pripombo, da deluje le v odsotnosti šuma. Ob njegovi prisotnosti je smotrno vzorčiti preko več period signala. Še boljše od računanja tekočega povprečja je filtriranje z nizkoprepustnim filtrom, ker nam omogoča daljše vzorčenje za boljšo eliminacijo motenj. FIR filtri niso primerni, saj vedno zajamejo enako število vzorcev, za pravilno delovanje pa morajo zajeti večkratnik števila vzorcev ene periode. Poleg tega zaradi velikega števila zahtevanih računskih operacij in zahtevi, da se lahko funkcija pri uri $125\ MHz$ kliče vsak cikel (pipeline je dovoljen), porabi izjemno veliko količino razpoložljivih sredstev. V eksperimentu se je izkazalo, da bi za filtriranje preko dveh ciklov z mejno frekvenco $1\ kHz$ porabili skoraj vsa razpoložljiva sredstva v uporabljenem FPGA. Računanje tekočega povprečja bi bilo seveda bolj učinkovito, še učinkovitejša pa je uporaba IIR filtra.

Simulacija je pokazala, da bomo za doseganje željene karakteristike potrebovali filter četrtega reda. Dobro razmerje med glajenjem in hitrostjo odziva dobimo pri mejni frekvenci $600\ kHz$. Ugotovili smo, da lahko za reprodukcijo enake filtrske karakteristike, kot jo dobimo s filtrom na števila s plavajočo vejico, uporabimo koeficiente s fiksno vejico s 27 biti (od tega 3 za celo število ter predznak). Pri uporabi manjših koeficientov začne filter dušiti tudi DC komponento. Zakasnilna linija za vhodne podatke je bila enakega tipa kot vhodni parameter, torej predznačeno celo število s 27 biti, za izhodne podatke pa so bila izbrana števila s fiksno vejico, 32b, od tega 6b za celi del ter predznak. Pri izbiri smo pazili, da pri normalnem delovanju ne bo prišlo do preliva, ter da ne bi izgubljali resolucije na decimalnih mestih. Izhodni podatek je enakih dimenzij kot vhod filtra.

Slika 3.1 prikazuje simulacijo obnašanja signala na sprejemni strani. S svetlo sivo je prikazan sprejeti signal, temno siv valovit signal pa prikazuje produkt prejetega signala s sinusom. Zgornja dva odziva filtra, ki se prekrivata, prikazujeta odziv za filter realiziran s plavajočo vejico, in simulacijo delovanja filtra z 27b koeficienti. Za demonstracijo obnašanja pri premajhni

resoluciji koeficientov je s prekinjeno črto prikazan še odziv filtra s 23b koeficienti.



Slika 3.1: Simulacija obnašanja signala na sprejemni strani.

Realizirano vezje se je po optimizaciji skrčilo na primerno majhno velikost, pri tem pa nismo žrtvovali kakovosti delovanja. Zaradi Xilinx-ove dobro napisane knjižnice za aritmetiko s števili s fiksno vejico, koeficiente vnašamo v obliki števila s plavajočo vejico.

3.4 Arkus tangens

Zadnji korak v meritvi, je izračun funkcije arkus tangens. Ko meritev pride do te točke, so se vse časovno kritične operacije že zaključile, tako da si lahko za dokončanje funkcije vzamemo poljubno veliko časa.

Izračuna bi se lahko lotili s pomočjo interpolacije ter vpoglednih tabel, kar pa je zaradi narave funkcije $atan2$ precej spominsko potratno. Primernejši je pristop s algoritmom CORDIC, kar je tudi osnovna izbira prevajalnika, če se v HLS sintezo vnese funkcijo $atan2$ iz knjižnice $math.h$.

Ta funkcija ne dopušča prostora za optimizacijo, saj se v vsakem primeru sintetizira za tip vhodnih podatkov $double$. Če se vhodne podatke najprej oblikuje v $float$, se funkcija izvaja počasneje, saj prevajalnik osnovno število najprej prevede v $float$, nato pa še v $double$.

Več manevrskega prostora dopušča uporaba namenske funkcije za realizacijo funkcije $atan2$ z algoritmom CORDIC. Uporabili smo tujo implementacijo [6], za vhodne ter izhodne parametre uporablja spremenljivke s fiksno vejico.

Funkcija $atan2$ se mora začeti izvajati šele, ko modulator konča z izvajanjem. To smo zagotovili tako, da smo na vhodni signal za začetek izračuna $atan2$ skozi detektor pozitivne fronte speljali nadzornikov signal za končanje izvajanja. Skladno s pričakovanji je ta funkcija med vsemi porabila še največ razpoložljivih sredstev.

3.5 Simulacija odbite svetlobe

Napisali smo poseben blok, ki kot vhod sprejme izhod lidarja, na izhodu pa simulira meritev zakasnjenega odboja svetlobe. V praksi je ta blok implementiran kot zakasnilna linija z 1b vhodom in 14b predznačenim izhodom. Enobitni signal se pomika po zakasnilni liniji. Na mestu simuliranega odboja se evalvira vrednost signala na zakasnilni liniji. Ničelna vrednost se smatra kot negativna meritev, enica pa se preslika v pozitivno število.

V simulaciji odboja ne upoštevamo upadanja amplitude signala z naraščanjem razdalje. Amplituda prejetega signala je vedno enaka, spreminja se le zakasnitev. V trenutni implementaciji se zakasnitev nastavlja v diskretnih korakih po 1/128 valovne dolžine moduliranega signala. Za oceno delovanja večje ločljivosti ne potrebujemo.

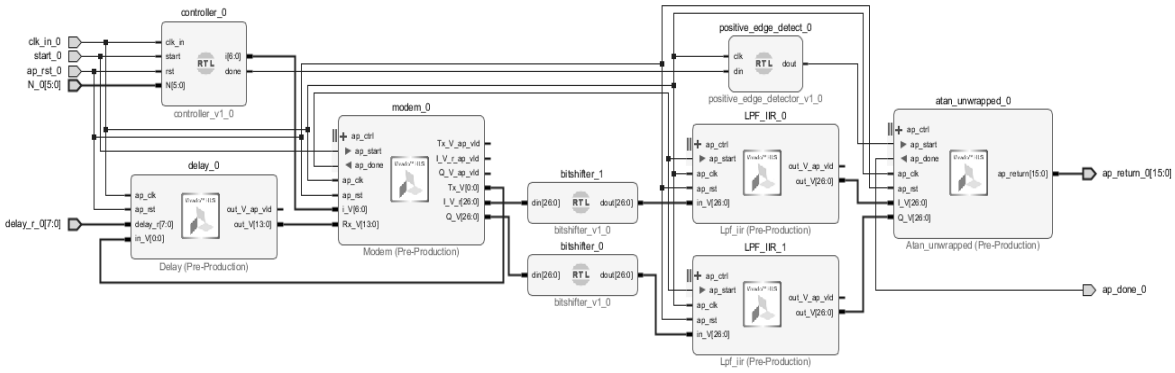
Takšna funkcija za simulacijo vhodno-izhodnih enot je kompaktna, le stežka bi se znašli v situaciji, ko v FPGA ne bi bilo dovolj prostora zanjo. Vseeno pa je potrebno upoštevati, da v sistem vnaša dodatno zakasnitev. V naši implementaciji se odbita svetloba na simuliranem vhodu pojavi z dodatnim zamikom dveh urinih ciklov.

3.6 Decimacija odvečnih bitov

Izhod modema sta 27b predznačeni I ter Q komponenti. S predpostavko, da ne bomo potrebovali toliko bitov, smo ju poslali skozi decimacijski filter, ki na vhodnem nizu izvede SRA operacijo (shift right arithmetic) za N bitov. Izkaže se, da lahko brez opaznejših negativnih učinkov zavržemo 12 bitov, izračun faze pa postane vidno nelinearen pri decimaciji 16 bitov.

3.7 Celoten sistem

Posamezni bloki so bili kot blokovni načrt povezani v funkcionalno celoto, kot to prikazuje Slika 3.2.
Večina ključnih blokov je bila sintetizirana s HLS, kar pomeni, da nismo imeli perfektnega nadzora nad implementacijo, kot bi ga imeli če bi komponente implementirali z lastnoročno napisano HDL kodo. Algoritmu za sintezo smo za vse komponente, razen $atan2$, zahtevali, da imajo inicializacijski interval 1, torej da lahko nov podatek pride v obdelavo v vsakem ciklu. Ker se funkcija $atan2$ izvede šele po končani meritvi, zanjo ta zahteva ne obstaja. Časovne zakasnitve prikazuje tabela 1, porabo sredstev pa tabela 2.



Slika 3.2: blokovni načrt za evalvacijo delovanja sistema

	zakasnitev (cikli)		zakasnitev (ns)		interval (cikli)	
	min.	maks.	min.	maks.	min.	maks.
modem	3	3	24	24	1	1
LPF	3	3	38	38	1	1
atan2	3	285	25	2396	3	285
odboj	2	2	16	16	1	1

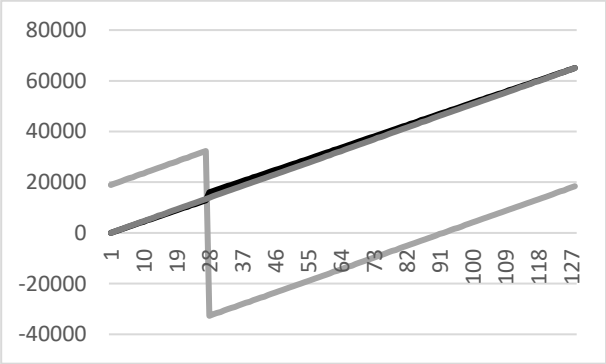
Tabla 1: zakasnitve modulov, sintetiziranih s HLS

	BRAM 18k	DSP48E	FF	LUT	URAM
modem	0	2	155	251	0
LPF	0	12	698	500	0
atan2	0	1	2716	2784	0
odboj	0	0	21	85	0

Tabla 2: poraba sredstev pri s HLS sintetiziranih modulih

4 Analiza delovanja sistema

Izvedli smo simulacijo delovanja sistema pri 128 zamikih, ki predstavljajo enakomerno razporejene zamike na območju med 0 in 2π. Izračunana faza je bila shranjena v datoteko za nadaljnjo obdelavo. Slika 4.1 prikazuje surovo meritev faze (svetlo sivo), nekompenzirano razvito meritev faze (črno) ter kompenzirano razvito meritev faze (temno sovo).

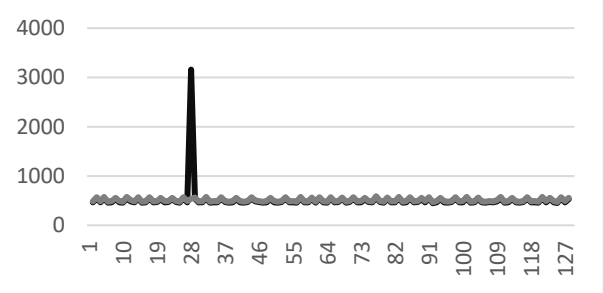


Slika 4.1: rezultat simulacije

V surovi meritvi se vidi, da nima kompenzacije za sistemski zamik faze. Tega odpravimo tako, da od vseh meritev odštejemo fazni zamik pri nastavljeni zakasnitvi 0, nato pa vsem negativnim vrednostim prištejemo 2^{16}. V fizični implementaciji se kalibracija izvede nekoliko drugače, saj nimamo popolnega nadzora nad nastavljenim faznim zamikom. Namesto tega se izvede meritev pri znani oddaljenosti (in s tem pričakovani fazi) tarče, in na izbrani točki določi kompenzacijsko vrednost, ki jo prištejemo meritvi, da dosežemo željeno vrednost. Na koncu negativen vrednosti enako kot prej preslikamo v pozitivne.

Slika 4.2 prikazuje razlike med sosednjimi meritvami s kompenzacijo (sivo) in brez (črno). Vidimo, da se koraki gibljejo okrog 500 (pri čemer 2^{16} predstavlja fazo 2π), v meritvi brez kompenzacije pa je ena špica, ki presega šestkratnik srednje vrednosti korakov. Omeniti gre, da povprečni korak, ki pri izbrani modulacijski frekvenci predstavlja 18 cm, ne predstavlja resolucije sistema, ki je večja. Korak odgovarja le resoluciji zakasnilne linije, ki jo uporabljamo za simulacijo propagacije svetlobe.

Da se znebimo nelinearnosti, moramo kompenzirati meritev faze pred razvojem. Kompenzacija, ki jo tu uporabljamo, je preprosto množenje surove meritve s faktorjem 1,042, kar uspešno odpravi nelinearnost.



Slika 4.2: korak med posameznimi točkami meritev faze

5 Zaključek

Delovanje nastalega CW lidarja je zadovoljivo za nadaljevanje na naslednji korak – implementacijo. Simulacija kaže, da se sistem obnaša skladno z načrtovalskimi cilji, v trenutni obliki pa dopušča še več možnosti za izboljšavo. Prva je decimacija bitov IQ komponent ter optimizacija nizkoprepustnega filtra. Dokler na realnem primeru ne preverim delovanja, je smotrno pustiti tako kot je, nato pa decimirati ob upoštevanju realnih meritev. Druga večja točka je kompenzacija ter korekcija meritev. Trenutno je to izvedeno naknadno, morda pa bi bilo to funkcionalnost smiselno implementirati v FPGA, čeprav bi to porabilo dodatne komponente, večji med katerimi sta 16b odštevalnik ter množilnik.

Literatura

[1] Godbaz, J.P. & Cree, Michael & Dorrington, A.A. & Payne, Andrew. (2009). A Fast Maximum Likelihood Method for Improving AMCW Lidar Precision Using Waveform Shape. Proceedings of IEEE Sensors. 735 - 738. 10.1109/ICSENS.2009.5398544.

[2] Pollastrone, Fabio & Cardarilli, Gian Carlo & Pizzoferrato, Roberto & re, Marco. (2016). Fully digital intensity modulated LIDAR. Defence Technology. 12. 10.1016/j.dt.2016.04.002.

[3] ADAMIČ, Michel, 2020, Časovno digitalni pretvornik visoke ločljivosti na čipu Xilinx Zynq-7010 [na spletu]. Univerza v Ljubljani, Fakulteta za matematiko in fiziko. [Dostopano 2 julij 2021]. Pridobljeno https://repozitorij.uni-lj.si/IzpisGradiva.php?lang=slv&id=117846

[4] Time of Flight System Design—Part 1: System Overview [na spletu]. Analog Devices inc. [Dostopano 20 julij 2021] Pridobljeno https://www.analog.com/en/analog-dialogue/articles/time-of-flight-system-design-part-1-system-overview.html

[5] Kastner, R. & Matai, J. & Neuendorffer, S. (2018). Parallel programming for FPGAs. Kastner, R., Matai, J., & Neuendorffer, S. (2018). Parallel Programming for FPGAs. ArXiv, abs/1805.03648.

[6] Begani, D. (2016). Linear Algebra Library: Atan2() Example, version 1.1. Priodblejno https://github.com/Xilinx/HLx_Examples/tree/master/Math/atan2_cordic