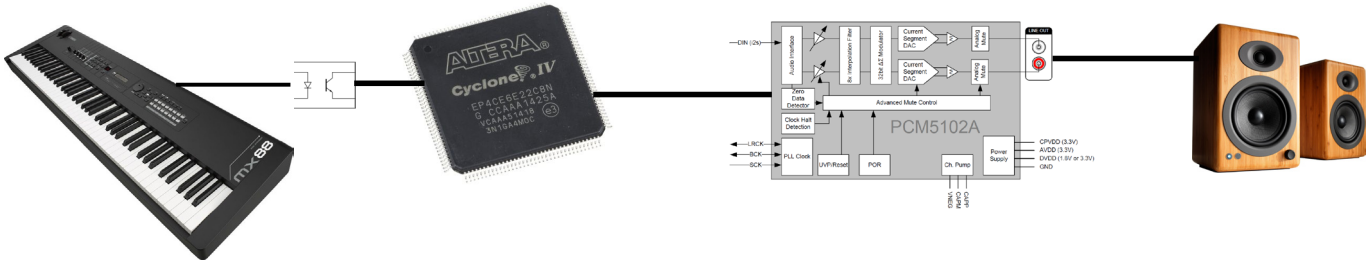
POROČILO za PROJEKT PRI PREDMETU DIVS

NAPISAL: Žiga Šmelcer
DATUM, KRAJ: 3. februar 2020, Ljubljana
KRATEK OPIS: MIDI -> sinus (CORDIC) -> i2s pretvornik
KLJUČNE BESEDE: Verilog, Intel ModelSim, Intel Quartus, Altera Cyclone IV, MIDI, CORDIC, i2s, TI PCM5102A DAC kartica

Ideja

Pri projektu je bil cilj narediti na Alterini Cyclone IV FPGA ploščici pretvornik iz MIDI serijskega protokola, ki ga uporablja večino elektronkskih inštrumentov na tipke, v i2s protokol, ki ga uporabljajo kvalitetne nizkonivojske audio DAC kartice. Iz MIDI signala se prebere pritisnjeno tipko in nato se v ROM celici naredi poizvedbo o frekvenci (bolj natančno o inkrementu normirane krožne frekvence) in pošlje v CORDIC obdelati podatek za trenutno višino tona. Logiko sem si otežil s tem, da omogoča pritisk katerekoli izmed 88 tipk, katerih signal se sešteje in pošlje na i2s enoto.

Zgradba sistema



Slika 1: Shematski prikaz priklopa naprav

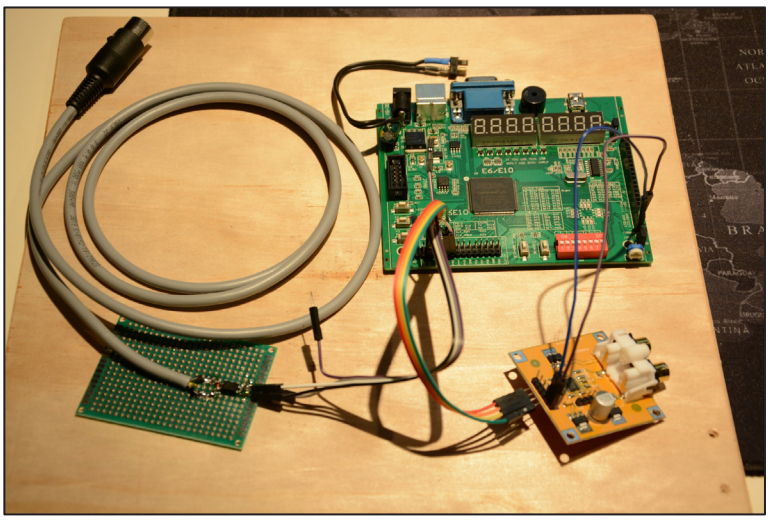
Priklop različnih naprav in sistemov je prikazan na sliki 1. Vhodna naprava je sintisajzer z MIDI izhodom, izhodna naprava pa zvočniki, ki dobijo signal iz TI PCM5102A DAC kartice. Za zaščito vhodnih pinov FPGA zaradi dolge povezave MIDI se uporabi optospojnik. Za izhodne pine nisem uporabil nobene zaščite, ker je kartica postavljena blizu razvojne ploščice.

Za FPGA sem uporabil razvojno ploščico z *Altera Cyclone IV* EP4CE6E22C8N. To je najnižji razred *Cyclone IV* FPGA-ja, zaradi česar sem imel pri sintezi težave, ker je napisan program vzel preveč logičnih elementov. Zaradi tega je bil končni projekt narejen s 16 bitnim CORDIC generatorjem namesto 24 binim.

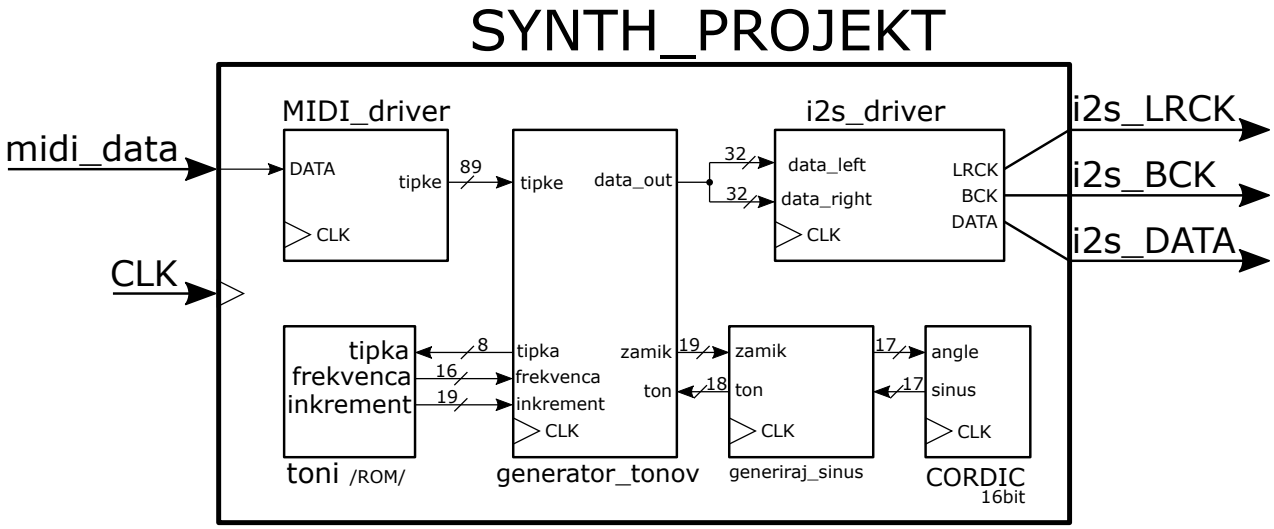
Opuščen je bil tudi LED segmentni prikaz nazadnje pritisnjene tipke.

Resources	EP4CE6	EP4CE10	EP4CE15	EP4CE22	EP4CE30	EP4CE40	EP4CE55	EP4CE75	EP4CE115
Logic elements (LEs)	6,272	10,320	15,408	22,320	28,848	39,600	55,856	75,408	114,480
Embedded memory (Kbits)	270	414	504	594	594	1,134	2,340	2,745	3,888
Embedded 18 × 18 multipliers	15	23	56	66	66	116	154	200	266
General-purpose PLLs	2	2	4	4	4	4	4	4	4
Global Clock Networks	10	10	20	20	20	20	20	20	20
User I/O Banks	8	8	8	8	8	8	8	8	8
Maximum user I/O (1)	179	179	343	153	532	532	374	426	528

Slika 2: Specifikacije Cyclone IV



Slika 3: Fizična postavitev

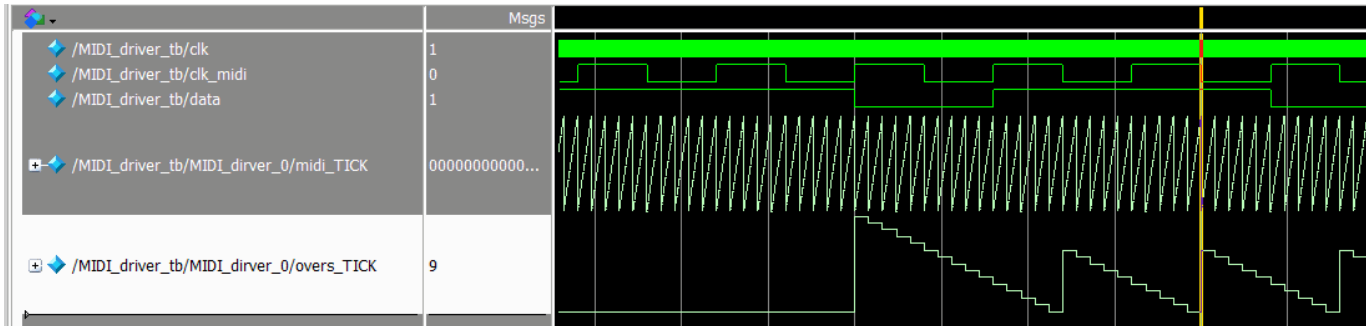


Slika 4: FPGA shema projekta

Projekt sem razdelil na manjše celote zaradi lažjega programiranja, debagiranja in optimizacije kode. Podenot je 6, združuje pa jih najvišja enota SYNTH_PROJEKT. Ta enota sprejema signal iz MIDI vmesnika in oddaja signale za i2s. Oba komunikacijska protokola sta serijska, zato imata le enobitne povezave.

MIDI_DRIVER

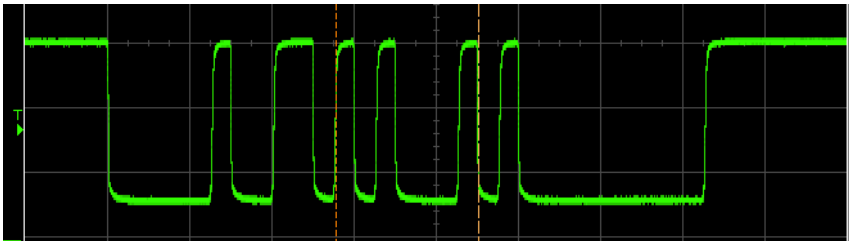
Ta podenota je sestavljena iz CLK dividerja za MIDI uro (31.250 kbps), ki sem ga pospešil za faktor 10, za boljšo zaznavo start bita in sredinsko branje logične vrednosti. Ko se zazna start bit, se naredi 15 praznih ciklov, nato po 10, da imamo vedno odčitavanje podatka na sredini urinega cikla.



Slika 5: MIDI_driver usklajevanje timingov

Drugi del te enote je state machine. Zgrajen je na podlagi MIDI podatkovnega paketka, ki sem ga moral ročno dešifrirati, ker na spletu ni bilo podobna zapisa. Zanimivo je na primer tudi to, da so tipke zamaknjene za 20 napravem klaviaturskim/klavirskim tipkam.

Vsako spremembo tipke se zapiše v 89 bitni register tipk. 89 zato, ker se indeksi upoštevajo od 0-88 in želimo preslikanje tipk 1:1.



Slika 6: Dešifriranje MIDI signala s pomočjo osciloskopa (sproščena tipka)

IME	TIPKA	CONV	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	B16	B17	B18	B19	B20	B21	B22	B23	B24	B25	B26	B27	B28	B29	
H	15		0	0	0	0	0	1	0	0	1	1	0	LSB		TIPKA				MSB						GLASNOST							
C Veliki	16	36	0	0	0	0	0	1	0	0	1	1	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	1	OFF TIPKA
C # Veliki	17	37	0	0	0	0	0	1	0	0	1	1	0	1	0	1	0	0	1	0	0	1	0	0	1	0	1	1	0	1	0	1	ON TIPKA
	18	38	0	0	0	0	0	1	0	0	1	1	0	0	1	1	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	1	TR rahlo
	19	39	0	0	0	0	0	1	0	0	1	1	0	1	1	1	0	0	1	0	0	1	0	0	0	1	1	1	1	1	0	1	TR močno
	20	40	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	0	1	0	0	1	0	1	1	1	1	1	0	0	0	1	TR srednje

Slika 7: Tabela podatkov za različne tipke in moči pritiska

TONI

Ta enota je sestavljena le iz enega vhoda in dveh izhodov. En izhod vrne frekvenco, drugi pa inkrement za krožno frekvenco za CORDIC enoto. S prva sem mislil preračunavati inkrement sproti, vendar sem ugotovil, da je možno že v naprej poračunati inkremente in si jih shraniti za vsako tipko posebej.

$$f(n) = 440 \times 2^{\frac{n-49}{12}} \qquad inkrement = \frac{2\pi\ f_{tipka}}{f_{vzorcna}}$$

Slika 8: Enačba za izračun frekvence tipke (levo) in inkrementa tipke (desno)

GENERIRAJ_SINUS

Je enota, ki pretvori inkrement ali krožno frekvenco tona v potrebno za CORDIC enoto. CORDIC enota namreč sprejema vhod le za [0, π/2]. Tako ima **generiraj_sinus** enota 5 različnih pogojev za vhod in izhod na podlagi inkrementa: [0, 0.5*π], [0.5*π, π], [π, 1.5*π], [1.5*π, 2*π] in [>2*π]. Na podlagi vhoda inkrementa je potrebno pravilno polarizirati tudi izhod za omenjene razdelke.

CORDIC

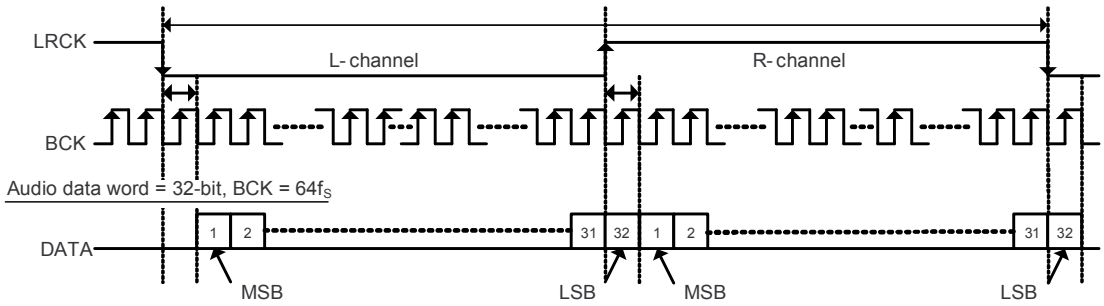
Enoto CORDIC sem generiral s pomočjo vgrajene kljižice na 16 bitno ločljivost decimalnih vrednosti. To pomeni, da je največja vhodna številka π/2 oz. binarno: 1.100100100001111110 (17 bitno število!).

Izhod ima enako največjo ločljivost decimalnik vrednosti in je podobno zapisan. Ima največje število 1 oz. binarno: 1.000000000000000000 (17 bitno število!).

I2S_DRIVER

Omenjena enota ima dva 32 bitna vhoda, ki sta vrednosti signala ob vzorčni frekvenci. En vhod je za levi in drugi za desni kanal. Na podlagi teh vhodov je potrebno generirati zaporedje izhodnih signalov LRCK, BCK in DATA.

- **LRCK** je enostaven signal, ki pove, ali pošiljamo levi ali desni kanal.
- **BCK** je ura, ki jo moramo generirati na podlagi vzorčne frekvence in tabele dokumentacije, da *PCM5102A* pravilno zazna vzorčno frekvenco podatkov. V mojem primeru je ta ura 12,288,000 Hz kar predstavlja problem, ker ima razvojna ploščica kristal s 50 MHz uro in uri nista večkratnih druga druge. BCK ura je izvedena z deljenjem sistemske ure s 4.
- **DATA** signal se generira na podlagi trenutnega kanala in je zaporedje bitov signala od MSB do LSB.



Slika 9: I2S protokol

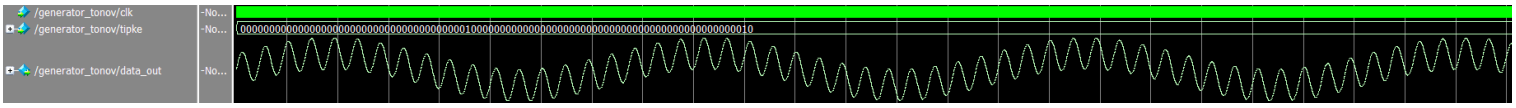
GENERATOR_TONOV

To je glavna enota, ki je zaslužna za pretvorbo stanj tipk (**tipke[88:0]**) v izhodni signal, ki se ga pošlje na **i2s_driver** enoto. Ker sem si zaželel možnost pritiska katerekoli tipke hkrati, sem kompleksnost te enote zelo zvišal. Ne le, da si potrebuje shranjevati stanje inkrementov (krožnih frekvenc) vseh 88 tipk, ampak mora vse to posredovati enoti **generiraj_sinus**, ki posreduje **CORDIC** enoti. Za vsako tipko ni mogoče narediti svojega CORDIC generatorja, ker že ena samo CORDIC enota vzame več kot polovico FPGA kombinatornih enot. Hkrati CORDIC enota potrebuje nekaj ciklov (5 za 16 bitni podatek) za izračun rezultata.

Zato sem se lotil narediti sekvenčno logiko, ki z vzorčno frekvenco (48 kHz) preleti čez vse tipke. Pri vsaki tipki pogleda, če je še pritisnjena, če je prištejemo registru inkrementov tipk potrebni inkrement za trenutno tipko, če ne, nastavimo register inkrementov tipke na 0. Pri prištevanju moramo paziti tudi na overflow, zato odštejemo vrednost 2π, če je inkrement večji od te vrednosti.

Nato sprožimo zakasnitev zaradi **CORDIC** enote. Ko se začne zakasnitev damo na vhod **generiraj_sinus** enote inkrement trenutne tipke. Počakamo potrebno število zakasnitev in tik pred koncem zakasnitve (`TICK_CORDIC==1`) prištejemo začasnemu signalu **data_signal** za **i2s_driver** trenutno vrednost tona.

Ob vzorčni frekvenci nastavimo **data_out_signal**, ki je vhod **i2s_driver** enote, na začasni signal **data_signal**, ki ga ponastavimo na 0.



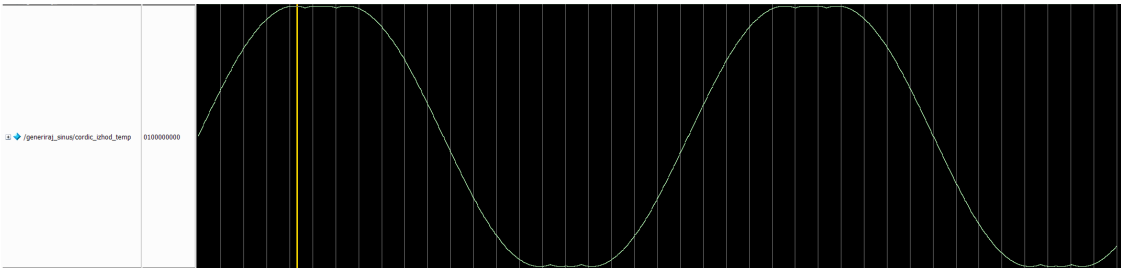
Slika 10: Preizkušanje delovanja enote generator_tonov v ModelSim

Težave pri izvedbi projekta

Tekom izdelave sem naletel na več težav, ki bi jih rad izpostavil. Težave je bilo v veliki meri uspešno odpraviti s temeljitim pregledom kode in simularanjem, tiste ki so bile strojne narave, je bilo potrebno odpraviti s kompromisom.

CORDIC IP knjižica

Zelo prikladno je, da Intel/Altera v programskem okolju Quartus omogoča generacijo različnih enot. Jaz sem uporabil CORDIC enoto. Vendar vzaradi nerazločne dokumentacije nisem pravilno implementiral CORDIC generiranja. S prva sem mislil, da gredo vrednosti od [0,N] in ne [0, π/2].



Slika 11: Napačno generiranje sinusa

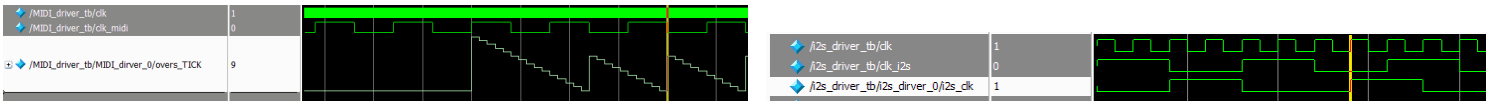
Table 7. Sin Cos Function Signals

Name	Type	Configuration	Range	Description
a	Input	Signed input	$[-\pi,+\pi]$	Specifies the number of fractional bits (F_{IN}). The total width of this input is F_{IN}+3.Two extra bits are for the range (representing π) and one bit for the sign. Provide the input in two's complement form.
		Unsigned input	$[0,+\pi/2]$	Specifies the number of fractional bits (F_{IN}). The total width of this input is w_{IN}=F_{IN}+1. The one extra bit accounts for the range (required to represent π/2). ČE GENERIRAMO N bitni CORDIC, IMAMO N+1 BITNI VHOD!
s, c	Output	Signed input	$[-1,1]$	Computes sin(a) and cos(a) on a user-specified output fraction width(F). The output has width w_{OUT}= F_{OUT}+2 and is signed.
		Unsigned input	$[0,1]$	Computes sin(a) and cos(a) on a user-specified output fraction width(F_{OUT}). The output has the width w_{OUT}= F_{OUT}+1 and is unsigned. ČE GENERIRAMO N bitni CORDIC, IMAMO N+1 BITNI IZHOD!

Slika 12: Intel CORDIC dokumentacija

Neusklajenost časov

S pomočjo *testbench* simulacij, v katerih sem generiral točno uro, sem odpravil nepravilnosti pri delilnikih ur. Če želimo generirati 4-krat počasnejšo uro, moramo šteti od 0-3 sistemske urine cikle, torej do ene manj kot želimo deliti, če ne dobimo petkratni delilnik ure!



Slika 13: Usklajeni uri midi_driver in TB (levo) in neusklajeni uri i2s_driver in TB (desno)

ROM celica

Če želimo pravilno narediti **ROM celico**, torej da sintetizator kode prepozna da želimo narediti ROM blok na FPGA-ju, moramo definirati vse vrednosti bloka. Pri case izvedbi to naredimo tako, da vključimo na koncu stavek **default**, ki priredi vrednost še za nenapisane naslove. V nasprotnem primeru sintentizator generira latche in nam porabi dodatne bloke ter naredi nekaj, kar si nismo zamislili.

```
always @(*) begin
  case (tipka)
    8'd001: begin fq <= 27; ink <= (`PI_2 * 27 )/ F_vzorc; end

    ...

    8'd088: begin fq <= 4186; ink <= (`PI_2 * 4186 )/ F_vzorc; end

    default: begin fq <= 0; ink <= 0; end // default potreben, če ne naredi latche!
  endcase
end
```

Slika 14: Koda za izdelavo ROM celice in ne latch elementov

Premajhen FPGA

Ob nakupovanju Alterine FPGA razvojne ploščice nisem bil pozoren koliko kombinacijskih enot je možno narediti na njej. Predvsem zato, ker nisem vedel, koliko jih vzame nek povprečen FPGA projekt.

Ko sem želel generirati 24 bitni CORDIC sinusni generator, je že sam zavzel skoraj celoten prostor elementov na FPGA-ju. Tu sem naredil kompromis manjše ločljivosti.

Na žalost nisem mogel implementirati še prikaza spremembe zadnje tipke na LED segmentnem zaslonu. Ta bi zelo dobro služil pri preverjanju MIDI_driver enote v celotnem sistemu.

Fitter Resource Utilization by Entity

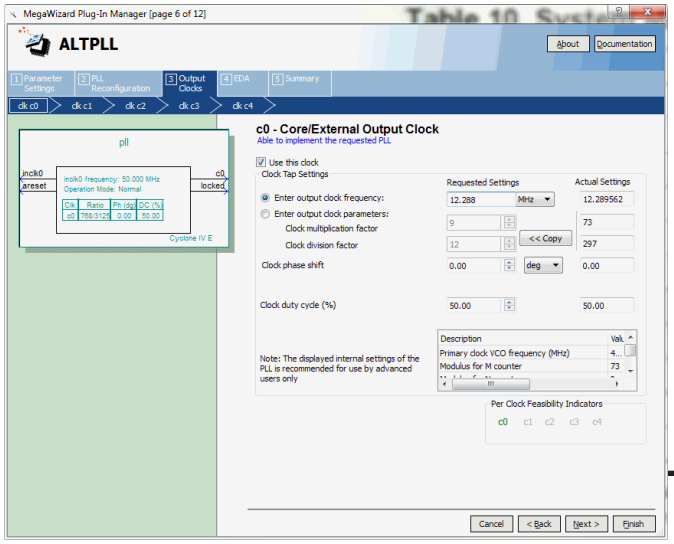
<<Filter>>

	Compilation Hierarchy Node	Logic Cells	Dedicated Logic Registers	I/O Registers	Memory Bits	M9Ks	DSP Elements	DSP 9x9	DSP 18x1
1	|SYNTH_PROJEKT	5470 (1)	2309 (0)	0 (0)	0	0	0	0	0
1	|MIDI_driver:MIDI_driver_0|	291 (291)	143 (143)	0 (0)	0	0	0	0	0
2	|generator_tonov:generator_tonov_0|	5107 (2366)	2115 (1761)	0 (0)	0	0	0	0	0
1	|generiraj_sinus:generiraj_sinus_0|	2501 (172)	354 (0)	0 (0)	0	0	0	0	0
1	|cordic_CORDIC_0:cordic_0|	2329 (717)	354 (0)	0 (0)	0	0	0	0	0
2	|toni:toni_0|	244 (244)	0 (0)	0 (0)	0	0	0	0	0
3	|i2s_driver:i2s_driver_0|	76 (76)	51 (51)	0 (0)	0	0	0	0	0

Slika 15: Poraba elementov na Cyclone IV razvojni plošči

Napačna sistemska ura

Razvojna ploščica z Altera FPGA je prišla s 50 MHz kristalom. Problem tega je, da je nemogoče dobiti točno frekvenco 12.288 MHz za 48 kHz vzorčno frekvenco. V mojem primeri delim 50 MHz s štiri in dobim 12.250 MHz, s čimer sem se sprijaznil. Da o zlobnih 44.1 kHz ne govorim. FPGA ima vgrajena tudi 2 PLL modula, in bi se z njim lahko še bolje približal frekvenci, a ne točno.



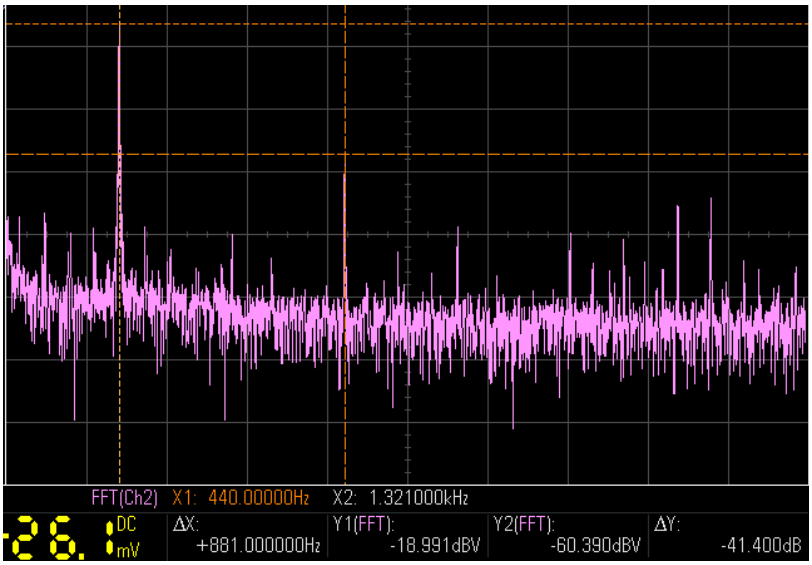
Sampling Frequency	System Clock Frequency (f_{SCK}) (MHz)								
	256 f_s	384 f_s	512 f_s	768 f_s	1024 f_s	1152 f_s	1536 f_s	2048 f_s	3072 f_s
8 kHz	2.048	3.072	4.096	6.144	8.192	9.216	12.288	16.384	24.576
16 kHz	4.096	6.144	8.192	12.288	16.384	18.432	24.576	36.864	49.152
32 kHz	8.192	12.288	16.384	24.576	32.768	36.864	49.152	–(1)	–(1)
44.1 kHz	11.2896	16.9344	22.5792	33.8688	45.1584	–(1)	–(1)	–(1)	–(1)
48 kHz	12.288	18.432	24.576	36.864	49.152	–(1)	–(1)	–(1)	–(1)

Slika 16: Seznam BCK ur glede na vzorčno frekvenco

Slika 17: Altera PLL čarovnik za generiranje ure

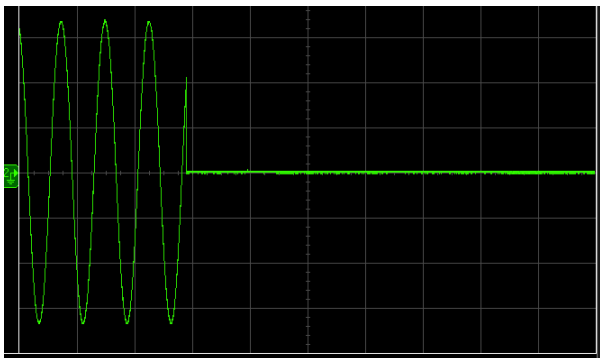
Zaključek

S samo izvedbo projekta sem zadovoljen kljub nekaterim težavam. V prihodnje bi izboljšal kvaliteto zvoka z uporabo FPGA-ja višjega razreda, da bi lahko implementiral 24 bitni CORDIC, in kristala namenjenega vzorčni frekvenci 48 kHz kot je na primer 49.152 MHz. Neusklajenost ur najverjetneje povzroči višje harmonska popačenja. Kot primer vidimo za 440 Hz oz. tipka A1, da ima le 40 dB rezerve, kar je slišno. Za kakovosten zvok si želimo nad 100 dB.



Slika 18: FFT analiza 440 Hz sinusnega signala

Izboljšati bi se dalo tudi začetek in konec pritiska tipke. Sinus se začne od začetka ob pritisku tipke, kar povzroči precej velik odvod (višje harmonska popačenja), še večji se zgodi ob sprostitvi tipke. To zelo negativno vpliva na kakovost sintetiziranega sinusa in je neprijetno poslušati.



Slika 19: Velika sprememba ob izklopu tipke

Slika 20: Potek razvoja