Elektrotehniško-računalniška strokovna šola in gimnazija Ljubljana

Pong v FPGA tehniki

Raziskovalna naloga

Elektrotehnika

Avtor: Simon Lipovšek, 4. letnik SŠ

Mentor: Aleš Volčini, prof.

Ljubljana, april 2023

Kazalo vsebine

Kazalo slik

Kazalo tabel

Povzetek

V tej raziskovalni nalogi sem predstavil uporabo FPGA integriranega vezja in primer, razumljiv za srednješolce. Osredotočil sem se na praktični vidik in sicer preko izdelavo igre Pong. Pri tem sem uporabil opisni jezik Verilog HDL, za prikaz slike uporabil VGA, vse skupaj pa poganja mikrokrmilnik MKR Vidor 4000 z integriranim vezjem FPGA. Analiziral sem generirane signale in preveril delovanje z dodano igralno konzolo ter zaslonom VGA. Potrdil sem primernost uporabe FPGA integriranega vezja ter z igro uspešno demonstriral delovanje.

Ključne besede: FPGA, MKRD Vidor 40000, Pong, VGA, Verilog HDL, VHDL, grafični vmesnik

Abstract

In this paper, I was researching implementation and limitations of field programmable gate array (FPGA). After researching it, I decided to extend my knowledge by building an implementation of the game Pong. My game is written in Verilog hardware description language (VHDL) and rendered on standardized implementation of video graphic array (VGA) display. For the programming board I used Arduino MKR Vidor 4000, which has an Intel Cyclone 10 series FPGA processor.

Keywords: FPGA, MKR Vidor 4000, Pong, VGA, Verilog HDL, VHDL, graphical interface

1. Uvod

V tej raziskovalni nalogi bom poizkusil dokazati, da lahko s FPGA-jem generiramo signal po VGA grafičnem standardu. Z njim bom generiral sliko za VGA zaslon. Uporabil bom FPGA Intel Cyclone 10, ki je vključen na razvojni plošči Arduino MKR Vidor 4000. Za boljši izziv bom v FPGA poizkusil realizirati osnovne gradnike igre Pong.

V preteklosti so bila FPGA integrirana vezja uporabljena izrecno v industrijske namene, saj so izredno specifična in je za vsako opravilo potrebno rekonfigurirati celotno vezje ali pa vsaj njegov del. Kljub temu pa FPGA postaja zadnje čase čedalje bolj popularen tudi v potrošni digitalni industriji. Njegova zmožnost paralelnega in zaporednega procesiranja mu podata prednost pred mikrokrmilnikih pri razširjanju in stiskanju datotek, zajemanju podatkov, pospeševanju industrijskih komunikacijskih protokolov in zahtevnih vzporednih računskih operacijah. Zato avtomobilska, telefonska in računalniška podjetja razmišljajo o implementaciji majhnega FPGA vezja poleg centralne procesne enote, ki bo pospešil računanje ponavljajočih računskih operacij. Tudi največji podjetji potrošnih mikroprocesorjev Intel in AMD sta že pred časom na trgu kupili vsako svoje podjetje, ki se ukvarja z razvojem in proizvodnjo FPGA integriranih vezij.

Izvorno kodo si bralec lahko ogleda na povezavi https://github.com/Pizmuh/FPGA_PONG.git .

Glede na temo naloge sem si izbral naslednje hipoteze, ki so :

1. Delo z FPGA vezji je dovolj enostavno, da bi ga lahko uporabljali tudi že v srednji šoli.
2. FPGA vezja so dovolj hitra, da lahko z njimi implementiramo VGA grafični standard.
3. Dosegli bomo resolucijo vsaj 640 x 480 pikslov v osmih barvah.
4. Vezje je mogoče pospešiti z dodatno manipulacijo časovnega pulza.
5. Predvidevam, da bom lahko generiral sliko in izvajal računske operacije sočasno s prikazovanjem signala po VGA grafičnem standardu.

2. Izbira strojne in programske opreme

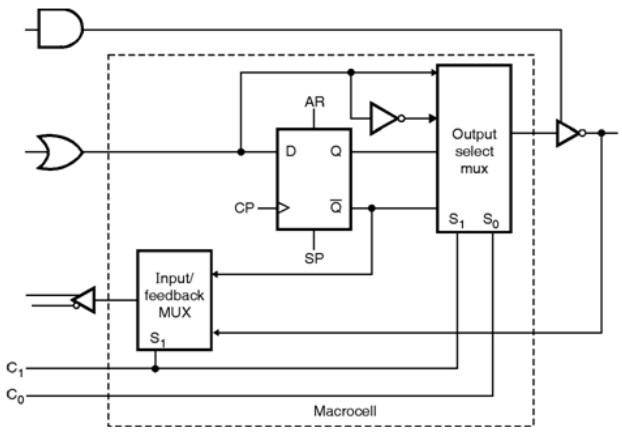
Pred eksperimentalnem delom si poglejmo nekaj pojmov potrebnih za njegovo razumevanje.

2.1 Vrste programirljivih vezij

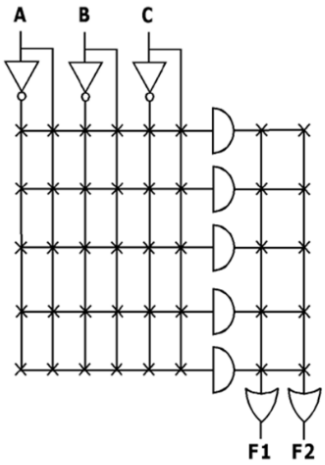
Programirljiva vezja so v svoji osnovi logična vezja katerim je mogoče preurediti povezavo med posameznimi logičnimi vrati. Ker vse vrste arhitektur ne morejo spreminjati povezav med vsemi logičnimi vrati ampak samo v nekemu skupku le teh, ta skupek imenujemo logične enote. Namen teh vezij je sestava rekonfigurativnih ali reprogramirljivih digitalnih vezij. Konfigurativna vezja se od programirljivih vezji razlikujejo v tem, da spremenijo povezave med kombinatoričnimi enotami, med tem ko pa pri programirljivih vezjih obnašanje spremenijo navodila v spominu vezja. Ta vezja skupaj s preostalimi komponentami tvorijo različne tipe programirljivih naprav.

2.1.1 PLD

So najpreprostejša in najcenejša vezja znotraj družine programirljivih vezij. Sem spadajo PROM, EPROM, PLA, PAL in GAL. V vezjih so velikokrat uporabljeni kot nadomestek za 7400 serijo TTL integriranih vezij. Sestavljeni so iz 4 do 22 povezanih mikrocelic (slika 1), ki so sestavljene iz aritmetične logike in pomnilniških vezij. Znotraj teh mikrocelic se lahko sestavi preproste Boolove enačbe s katerimi lahko pretvorimo binarne vhodne informacije v binarne izhodne informacije. Omogočajo tudi shranitev izhodnih bitov s pomočjo sekvenčnih vezij (flip-flopov), vendar je v večini primerov to mogoče samo do naslednjega časovnega pulza. Kljub temu sta sama arhitektura in delovanje še vedno močno odvisno od proizvajalca, kar pomeni da se med seboj razlikujejo tudi v načinu programiranja. Z razliko od drugih tipov programirljivih naprav se SPLD lahko programira samo enkrat. Glede na tip programiranja in shranjevanje konfiguracije obstajajo podvrste kot so PAL, GAL, FPLA in PLD (slika 2). Nekatere uporabljajo princip varovalk (*ang. fuses*), ki so po prvem programiranju prežgane in tako ustvarijo potrebno logično vezje, druga pa standardne spominske enote kot so EPROM, EEPROM, FLASH in druge. ***[14], [26], [16]***



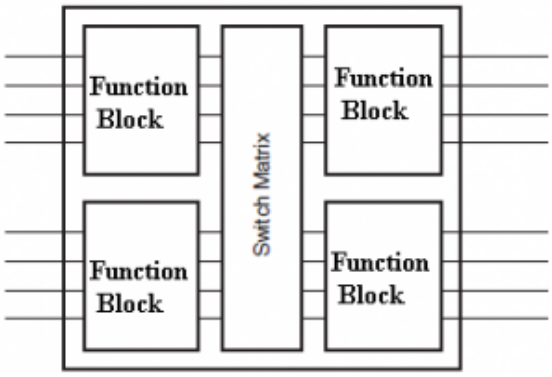
Slika 1: Mikrocelica



Slika 2: Primer PLD vezja

2.1.2 CPLD

Je kompleksnejša verzija SPLD-ja, saj vsebuje podobna vezja in makrocelice, z razliko v tem da so te celice med seboj povezane s preklopno matriko. Tako kot same makrocelice so tudi povezave med njimi programirljive. Na sliki 3 lahko vidimo funkcijske bloke, ki predstavljajo makrocelice, podobne kot v SPDL-ju.



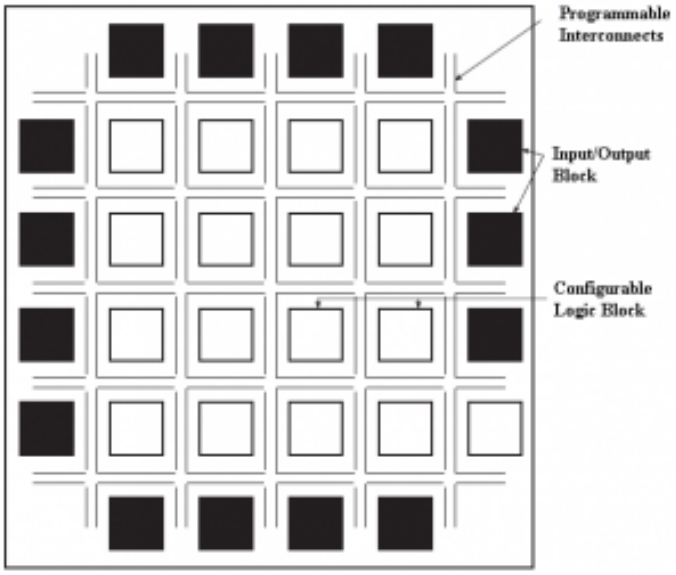
Slika 3: Blok diagram CPLD vezja

Slika prikazuje, kako so posamezni funkcijski bloki povezani s preklopno matriko, kar omogoča izmenjavo podatkov med zgornjimi in spodnjimi funkcijskimi bloki v naslednji procesni stopnji. Kljub temu, da podatke lahko premikamo po vzporednih funkcijskih blokih in naprej skozi procesne stopnje, jih ne moremo premakniti skozi procesne stopnje nazaj, saj so med seboj ločene s preklopno matriko, ki ne omogoča reverznega propagiranja podatkov. Zaradi njihove predvidljive časovne karakteristike, zanesljivost in možnosti programiranja zahtevnejših operacij so velikokrat uporabljeni v kritični infrastrukturi in napravah, kjer sta potrebna visoka natančnost in učinkovitost. Poleg tega pa jim

nizka električna poraba in nizka cena omogočata vgradnjo v prenosne naprave, kot so telefoni, ure in podobno. ***[14]***

2.1.3 FPGA

FPGA so integrirana vezja, ki vsebujejo logične bloke obdane s programirljivimi vhodno-izhodnimi enotami. Lahko vsebujejo od deset do sto tisoč logičnih blokov. Idealno naj bi FPGA ponujal popolno povezljivost med posameznimi logičnimi bloki, ampak zaradi stroškov izdelave se takih komponent ne najde na trgu. Vezje FPGA uporabljamo povsod, kjer je potreben hiter odziv in izvedba velikega števila ponavljajočih ali vzporednih operacij. Pri programiranju FPGA-ja določamo povezave med vhodnimi, izhodnimi enotami, logičnimi bloki in s tem implementiramo določeno vezje.



Slika 4: FPGA blok shema

FPGA naprave omogočajo kompleksnejše operacije in so primerljive s takimi, ki imajo milijone logičnih vrat. Poleg osnovnih logičnih in vhodno-izhodnih enot vsebuje tudi integriran procesor, spomin z visoko kapaciteto, časovno vezje in podpira različne signalizacijske tehnologije. FPGA se zato uporablja pri procesiranju in shranjevanju velike količine podatkov, procesiranju digitalnih signalov, natančnih merilnih inštrumentih in telekomunikacijskih sistemih. Med seboj se razlikujejo v arhitekturi in načinu proizvodnje tranzistorjev. Tako TTL (ang. transistor-transistor logic) in Schottky TTL arhitektura uporabljata tranzistorje kot stikala, ECL (ang. emitter occupied

logic) pa uporablja tranzistorje kot usmerjevalnike toka skozi logična vrata, ki izvajajo logične operacije. Za izvedbo MOSFET in implementacijo logičnih vrat ter drugih digitalnih vezij CMOS uporablja P in N tip polprevodnega kovinskega oksida. Obstajajo tudi druge tehnologije kot so: CBT (Crossbar switch technology), GaAs (Gallium Arsenide), I^2L (Integrated injection logic), SOS (Silicon on sapphire). ***[1], [2], [16]***

2.2 Verilog HDL

Verilog HDL (hardware description language) je opisni jezik, ki sta ga razvila Phil Moorby in Prabhu Goel leta 1984 in sicer za podjetje Gateway Design Automation Inc. Način načrtovanja in pravila jezika so se spreminjala do leta 1990, standardizirala pa so se ko je pravico do uporabe in distribucije odkupilo podjetje Cadence Design System. V tem podjetju so opazili vrednost programa, zato so se odločili da ga bodo poizkusili vpeljati kot standard. Leto po pridobitvi programa so ga uredili in javno dokumentirali, kmalu za tem so poslali prošnjo za standardizacijo na IEEE. Ta je prošnjo odobril kot standard 1364-1995 in poimenoval jezik Verilog-95. Leta 2001 je podjetje ponovno poslalo prošnjo za nov standard z novimi definicijami in razširitvami. Najbolj pomembna razširitev je bila omogočanje predznačenih spremenljivk, kar je tudi omogočil vzpon in dominanco Veriloga-2001. 2005 je IEEE ponovno izdalo novi standard (IEEE standard 1364-2005), kjer so bili objavljeni majhni popravki in modifikacije. Poleg tega so dodali tudi podprogram za pomoč pri načrtovanju vezja, ki je popularizacijo in adaptacijo tega jezika še dodatno pospešil. Danes je najpopularnejši jezik SystemVerilog 2009, ki je nastal leta 2009 z združitvijo SystemVeriloga in Veriloga. ***[4][5]***

2.2.1 HDL

HDL je pripomoček s katerim lahko inženir opiše logično delovanje digitalnih vezij na različnih abstraktnih nivojih. Zajame lahko Boolove enačbe, časovna vezja, tabele stanj in sheme s pomočjo notacij, ki so hitro berljive in podobne programskemu jeziku. Uporabljen je za testiranje in simuliranje različnih digitalnih vezij, saj pri delu s FPGA-ji omogoča hitro testiranje in preverjanje delovanja digitalnega vezja. ***[5]***

2.2.2 Sintaksa in oblikovna struktura Verilog HDL-ja

Oblikovna struktura Verilog HDL-ja temelji na modulih, ti moduli se lahko med sabo kličejo podobno kot funkcije pri programiranju. Večina stavkov, ki opisujejo delovanje vezja mora ležati znotraj teh modulov, med tem ko stavki, ki spreminjajo pogoje oz. parametre pod katerimi naj bi delovalo vezje, ležijo izven njih. Stavek zaključimo s podpičjem.

Vrste računskih operacij

Jezik pozna različne operacije, prikazane na tabeli spodaj. Te operacije lahko uporabljamo na Boolovih spremenljivkah ali na številih.

Operacija	*Razlaga*
**	Znak za potenciranja.
*	Znak za množenje.
/	Znak za deljenje.
%	Znak za izračun ostanka.
+	Znak za seštevanje.
-	Znak za odštevanje.
<<	Znak za premik bitvo v levo.
>>	Znak za premik bitvo v desno.
<<<	Znak za premik števk v levo.
>>>	Znak za premik števk v desno.
<	Znak za manjše kot.
<=	Znak za manjše ali enako kot.
>	Znak za večje kot.
>=	Znak za večje ali enako kot.
==	Znak za je enako.
!=	Znak za ni enako.
===	Znak za popolno binarno enakost.
!===	Znak za popolno binarno ne enakost.
&	Bitna in vrata.
~&	Bitna ne in vrata.
^	Bitna XOR vrata.
^~	Bitna XNOR vrata.
~^	Bitna XNOR vrata (druga vrsta zapisa).

/	Logična ali vrata.
~/	Logična ne ali vrata.
&&	Logična in vrata.
||	Logična ali vrata.
or	Logična ali operacija v primeru stavka.

Tabela 1: Računski operatorji v Verilog HDL opisnem jeziku

Deklaracija signalov

Verilog HDL pozna vhodne, izhodne in kombinirane signale. Vhodni signal deklariramo s ključno besedo *input*, izhodne *output*, vhodno-izhodne pa z *inout*, kot vidimo na primeru spodaj.

```
input vhod,
inout vhod_in_izhod,
output izhod;
```

Slika 5: Prikaz deklaracije vhodnih, izhodnih in vhodno-izhodnih signalov

Ker se lahko vhodne in izhodne vrednosti spreminjajo skozi izvajanja programa, je potrebno vrednosti začasno shraniti v registrih. Za shranjevanje začasnih vrednosti v registre se uporablja ključna beseda »reg«, ki ji sledita končni in začetni naslov bita, zavito v oglatem oklepaju, ločena z dvopičjem:

```
reg [2:0]vhod = 0; /// Spremenljivki vhod se vrednost 0 vpiše v register
                   /// dolg 3 bite.
```

Slika 6: Shranjevanje podatkov v register

Nekatere spremenljivke so samostojne ali pa so računsko odvisne druge od druge. Ker spremenljivki ne moremo spreminjati vrednosti izven njenega modula, za to uporabljamo konstantni gonilnik, ki je zakrit v ključni besedi *wire (slika 7)*. V primeru, kjer hočemo izvajati matematične operacije na podatkih uporabimo konstantni gonilnik s ključno besedo *assign*.

```
wire [15:0] novi_vhod, stari_izhod;
/// Stari_izhod je primer vrednosti, ki jo vrne implementirani podmodul.
Novi_vhod je pa spremenljivka, katera je sedaj lahko uporabljena v višjem modulu.

assign naslov = celotni_naslov && maska;
/// Stari_izhod je primer vrednosti, ki jo vrne implementirani podmodul.
Novi_vhod je pa spremenljivka, katera je sedaj lahko uporabljena v višjem modulu.
```

Slika 7: Konstantni gonilnik

Deklaracija modulov

Pri deklariranju modula je najprej potrebno napisati ključno besedo *module*, s katero povemo, da smo začeli opisovati vezje. Sledi ime modula in v oklepaju še imena vhodnih in izhodnih signalov. Imena spremenljivk se ločuje z vejico. Modul se zaključi z ukazom *endmodule*, kateremu ni potrebno dodati podpičja na koncu (slika 8).

```
module multiplekser(
  input clock,
  input reg [2:0] vhodna_podatkovna_linija,
  input reg selektor,
  output reg podatki
);
/// Tukaj vmes lahko dodamo kodo, ki jo želimo dodatno implementirati
endmodule
```

Slika 8: Pimer praznega modula

Sprožilni pulz

Sprožilni pulz je vezje, ki se uporablja, ko hočemo izvajati več stvari sočasno. Pri deklaraciji je najprej potrebno napisati ključno besedo *always@*, sledi oklepaj znotraj katerega povemo način proženja (npr. *posedge* pove, da se proženje izvaja na sprednji fronti) in ime spremenljivke s katero hočemo prožiti vezje. Najbolj uporabljena je sintaksa aktivacija proženje na prednjo fronto zato je ta oblika najkrajša in opisana takole:

```
always@(posedge clk)
  begin
  end
/// Ob vsaki zaznavi prednje fronte na signalu clk se izvede koda znotraj begin
in end.
```

Slika 9: Sprožilno vezje

2.2.3 Primeri delovanja: 16-bitni seštevalnik

Takole seštejemo dve 16-bitni vrednosti:

```
module signed_adder
#(parameter WIDTH=16)
(
    input signed [WIDTH-1:0] podatki_a,
input signed [WIDTH-1:0] podatki_b,
```

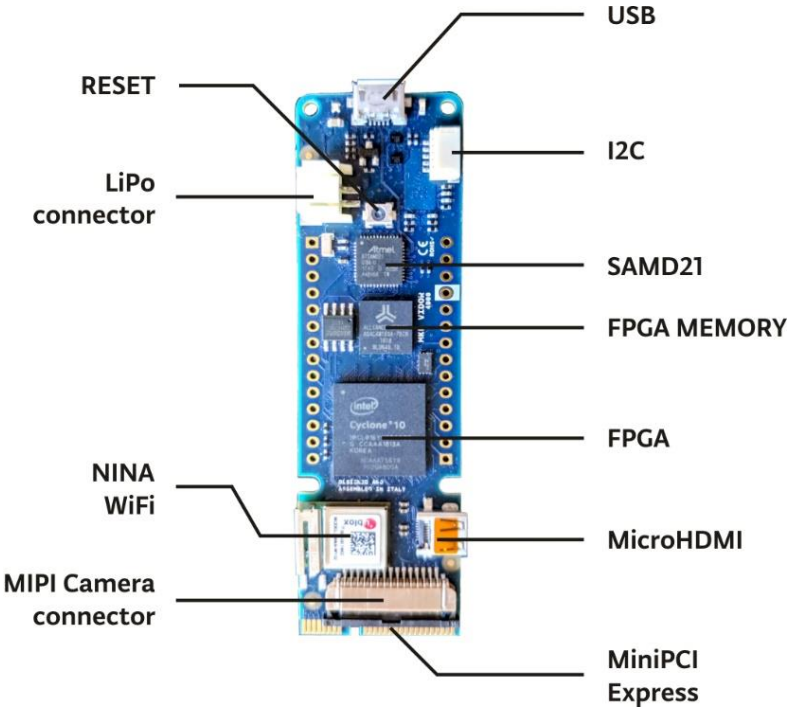
```
output [WIDTH:0] rezultat
)
begin
	assign rezultat = podatki_a + podatki_b;
endmodule
```
Slika 10: 16-bitni seštevalnik

2.2.4 Uporaba programa Quartus Prime

Intel Quartus Prime je program narejen za načrtovanje SOPC (system on a programmable chip). Z njim lahko simuliramo vezje, pripravljamo datoteke glede na izbrano programirljivo napravo in tvorimo časovne analize. Za načrtovanje podpira več različnih opisnih jezikov in datotečnih zapisov. Poleg opisnih jezikov podpira tudi načrtovanje vezja v obliki diagramov ali električnih shem. Program je razvilo podjetje Altera, ki je bilo ustanovljeno leta 1983 in razvilo takrat imenovani program Quartus. Leta 2015 pa je Altero kupilo podjetje Intel in spremenilo ime programa Altera Quartus II v Quartus Prime, pri čemer je ohranilo le uporabniški vmesnik.

2.2.5 Arduino MKR Vidor 4000

Arduino Vidor MKR 4000 je družba Arduino izdala z namenom približati razvoj in uporabo FPGA širši publiki.



Slika 11: Arduino MKR Vidor 4000

Na sliki 11 lahko vidimo, da poleg FPGA-ja MKR Vidor 4000 vsebuje tudi več različnih komponent, saj je bil namenjen eksperimentiranju in raziskovanju. Najbolj pomembna stvar je seveda integrirano vezje 10CL016, to je vezje FPGA iz serije Cyclone, ki jo proizvaja podjetje Intel. Vsebuje 504 KB integriranega RAM-a in 56 18x18 bitnih množilnikov za visoko-hitrostne DSP računske operacije. Najvišja priporočena frekvenca celotnega vezja je 150 MHz. FPGA-ju so dodatno dodelili 8 MB SRAM-a, kar omogoča manipulacijo video ali zvočnih posnetkov. Koda s katero je FPGA programiran se shrani na posebej dodeljenem 2 MB QSPI flash pomnilniku. Plošča vsebuje tudi priključek za MIPI kamero in mikro HDMI. Na plošči najdemo še mikrokrmilnik Arm Cortex M0 32 bitni SAMD21, kateri si deli izhodne priključke z FPGA-jem, WiFi in Bluetooth komunikacija pa je izvedena z nizko potrošnim integriranim vezjem NINA-W10, ki deluje na 2,4 GHz frekvenčnem spektru. ***[23] [25]***

2.3 VGA Grafični standard

VGA je ime za grafično kartico, način prikazovanja slike in priključek. Standardiziran je bil leta 1987, razvilo ga je podjetje IBM za PS/2 linijo računalnikov. VGA priključek je standardni D-SUB DB 15 priključek (sliki 12 in 13), ki ni omejen na zaslonske oz. projekcijske naprave, je pa tam največkrat uporabljen.

Slika 12: VGA priklop

Slika 13: VGA priključek

Za prenos signalov VGA uporablja 15 pinov, za pričvrstitev priključka pa se dodatno uporabljata dva vijaka. Kabel vsebuje zaščito pred elektromagnetnim valovanjem z ozemljenim prevodnim ovitkom, ki je ovit okoli podatkovnih žic in ustvari Faradayevo kletko, ko pritrdimo priključek na napravo.

Za prikaz slike sem se odločil uporabiti resolucijo 640 x 480 in osveževanje slike s frekvenco 60 Hz, zato se bom v nadaljevanju posvetil razlagi tega VGA načina.

Generiranje slike temelji na standardiziranem številu pulzov. Starejši zasloni so omogočali popolnoma analogen pristop in izbrane frekvence osveževanja. Večina zaslonov prenese majhno napako v frekvenci, je pa potrebno natančno preračunati časovni pulz posameznega piksla izvorne naprave za boljšo vertikalno in horizontalno sinhronizacijo.

$$f_p = N_{vp} \cdot N_{Hp} \cdot f_o$$

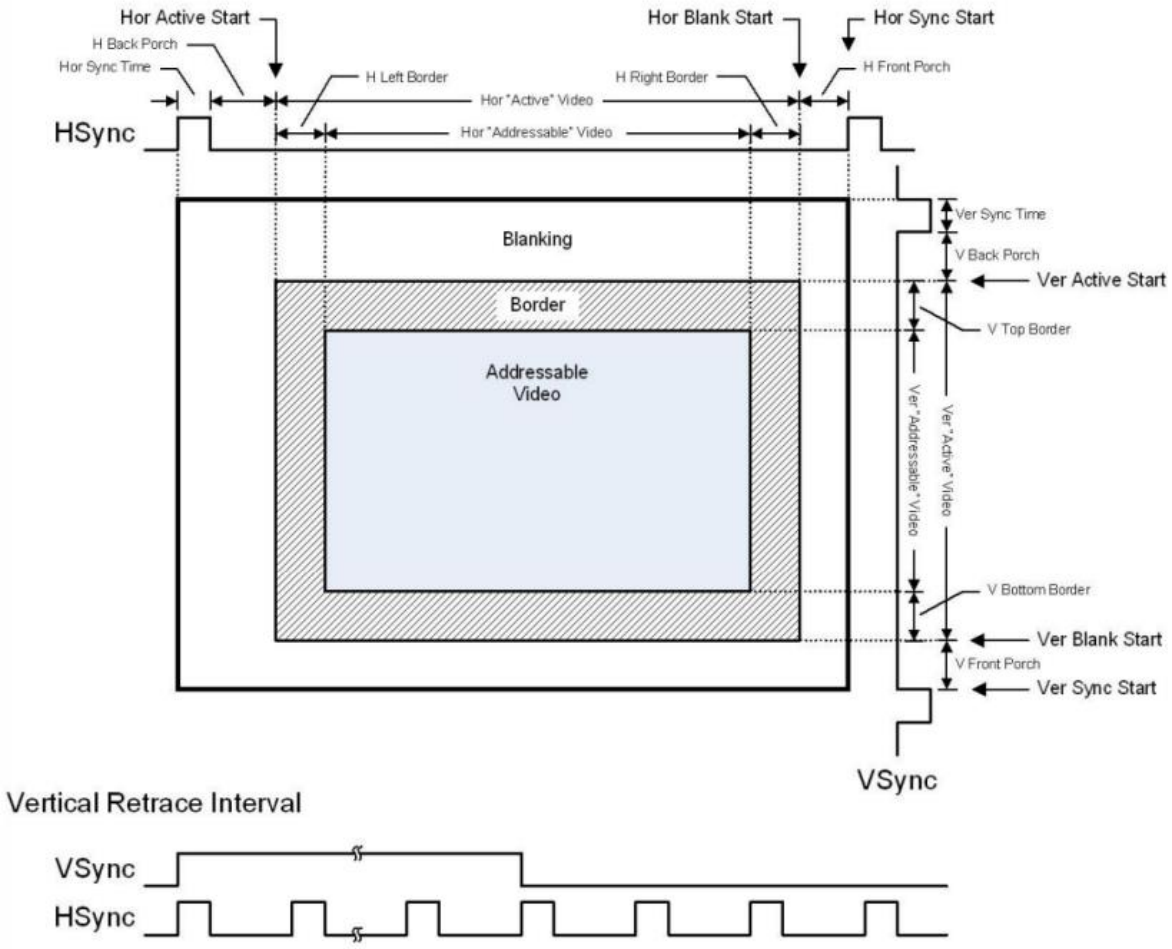
f_p je frekvenca s katero je potrebno pošiljati podatke o enem pikslu.

f_o je frekvenca osveževanja celotnega zaslona.

N_{Vp} je število vertikalnih pikslov.

N_{Hp} je število horizontalnih pikslov.

Z zgornjo enačbo lahko ugotovimo kakšna je frekvenca s katero moramo pošiljati podatke preko VGA priklopa. Upoštevati je treba tudi mrtvi čas, pri katerem je potrebno pošiljati prazen signal na začetku in na koncu. Dolžina mrtvega časa se spreminja od zaslona do zaslona ampak po navadi se uporablja 16 pikslov na začetku in 48 pikslov na koncu. V našem primeru pri resoluciji 640 x 480 pride frekvenca posameznega piksla 25,2 MHz, če upoštevamo da potrebujemo nekaj časa za horizontalni in vertikalni povratek.



Slika 14: Primer delovanja VGA grafičnega vmesnika

Na zgornjem delu slike 14 lahko vidimo vizualizacijo pomenske vsebine vsega signala, ki je potreben za prikaz slike na VGA zaslonu. Na spodnjem delu pa vidimo simbolični prikaz vertikalnega in horizontalnih sinhronizacijskih pulzov.

Standard za 640 x 480 zaslonsko resolucijo pri 60 Hz določa naslednje parametre ***[11]***:

Screen refresh rate	**60 Hz**
Vertical refresh[(glej opombo)]	31.46875 kHz
Pixel freq.	25.175 MHz

Tabela 2: Frekvence potrebne za grafični standard 640 x 480 @60Hz

Na zgornji tabeli lahko vidimo, da za osveževanja zaslona pri frekvenci 60 Hz potrebujemo horizontalno frekvenco približno 31 kHz[1]. Pikslom se vrednost spreminja približno s frekvenco 25 MHz.

Scanline part	*Pixels*	*Time [µs]*
Visible area	640	25.422045680238
Front porch	16	0.63555114200596
Sync pulse	96	3.8133068520357
Back porch	48	1.9066534260179
Whole line	800	31.777557100298

Tabela 3: Število pikslov in njihov pomen na horizontalni osi

Zgornja tabela prikazuje koliko začetnih in končnih pulzov je potrebno za sinhronizacijo zaslona z grafičnim generatorjem na horizontalni osi.

Frame part	*Lines*	*Time [ms]*
Visible area	480	15.253227408143
Front porch	10	0.31777557100298
Sync pulse	2	0.063555114200596
Back porch	33	1.0486593843098
Whole frame	525	16.683217477656

Tabela 4: Število pikslov in njihov pomen na vertikalni osi

Tako kot prej lahko tukaj vidimo število začetnih in končnih pulzov potrebnih za sinhronizacijo, z razliko, da gre za vertikalo zaslona.

3. FPGA v praksi

Sedaj, ko vemo kaj potrebujemo, si lahko pogledamo praktično uporabo.

3.1 Sestavljanje izdelka

Glavni del naloge se nanaša na FPGA integrirano vezje. Zanj sem uporabil že obstoječo testno ploščo Arduino Vidor 4000. Najprej sem nanj povezal priključke, potrebne za generiranje VGA

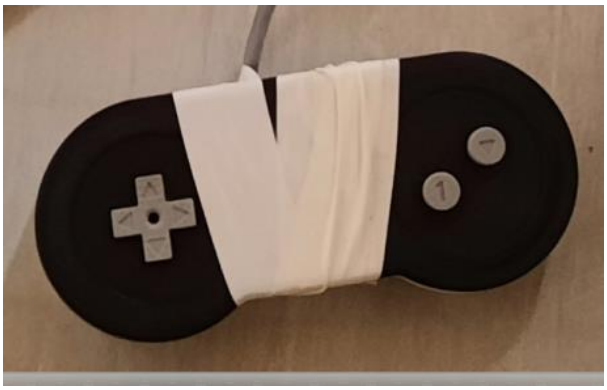
[1] Vir napačno navaja da gre za “Vertical refresh”, kar je napačno. Gre za horizontalno frekvenco.

signal, upore, ki so prevzeli vlogo digitalno analognega pretvornika (DAC) in priključke za igralno konzolo.

3.1.1 Igralna konzola

Za ohišje igralne konzole sem uporabil prosto dostopni 3D model na sliki 16 in ga rahlo preuredil, kot je vidno na sliki 15.

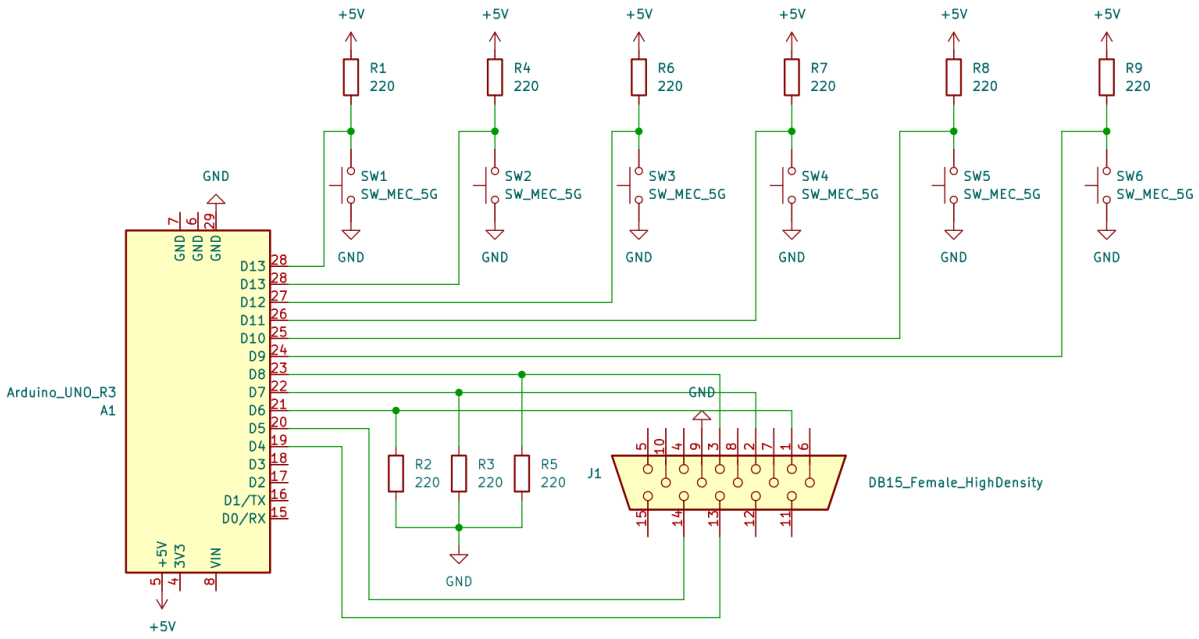
Slika 16: 3D model uporabljene igralne konzole



Slika 15: Izdelana igralna konzola

Ohišje igralne konzole je natisnjeno na 3D tiskalniku. Gumbe sem z krmilnim vezjem povezal s pomočjo »ethernet« kabla, na katerega sem priključil vezje z gumbi vezanimi v negativni logiki. Konzola vsebuje šest gumbov, štiri na levi strani in dva na desni. Pod smernim gumbom sem položil peno, ki v sproščenem stanju uravna gumb. Kabel vsebuje osem žic, od katerih sem jih šest uporabil za prenos stanja gumbov in dve za napajanje. Predvidel sem tudi stacionaren del ohišja ki naj bi vseboval MKR Vidor 4000, del vezja za igralno konzolo in vezje DAC in priključek za VGA kabel (D-SUB DB15).

3.1.2 Vezava komponent



Slika 17: Shema izdelanega vezja

Na zgornji sliki lahko vidimo, da sem za generiranje in prenos VGA signala uporabil pet povezav: R, G, B, VS ter HS. Gumbe na igralni konzoli sem se odločil vezati v negativni logiki, saj sem se s tem izognil šumom, ki bi lahko povzročili napačno interpretacijo stanja gumba.

V tabeli spodaj vidimo spisek in število uporabljenih komponent.

Komponente	*Količina*
Arduino MKR Vidor 4000	1
VGA vmesnik	1
220 ohm upori	9
Gumbi	6

Tabela 5: Kosovnica

Za prikaz slike sem uporabil ekran, ki še podpira VGA standard in brez problema prikazuje resolucijo 640 x 480 (resolucija je nizka in je lahko za nekatere ekrane tudi problematična). Da je to lahko problem sem ugotovil, ko sem poizkusil stabilizirati sliko, saj moja implementacija ne deluje na frekvenci osveževanja 60 Hz ampak 72 Hz. Ugotovil sem, da so v te namene primerni zasloni, ki podpirajo širok nabor osveževanjih frekvenc :).

3.2 Pong

Kot praktični pristop sem si zadal implementacijo igre Pong z manjšimi poenostavitvami. Za razliko od originalne igre, ki je po vsej verjetnosti napisana v nekem programskem jeziku sem si zadal igro sestaviti v obliki vezja z diskretnimi logičnimi elementi in z minimalnim programom, ki teče na mikrokontrolerju. Vezje sem načrtoval v okolju Intel Quartus Prime, kjer sem tudi generiral strojno kodo za konfiguracijo vezja FPGA. Vezja nisem risal ampak sem ga opisal v opisnem jeziku Verilog HDL. Z razliko od drugih razvojnih plošč je v samem postopku potrebno še opraviti nekaj vmesnih korakov, npr. preden sem naložil strojno kodo na FPGA, jo je bilo potrebno zrcaliti, kar sem naredil s pomočjo prostega dostopnega Java programa po imenu Reversebit. Po zrcaljenju bitov sem jo lahko integriral v Arduino program, ki jo ob resetu nalaga na FPGA.

3.2.1 Implementacija VGA grafičnega signala

Grafični signal sem implementiral s pomočjo prožilnega pulza, ki mi je omogočal natančno nadziranje časovnih intervalov med vsakim poslanim bitom.

```
always @ (posedge clock)
begin
  if (counter == 3)
  begin
    counter <= 1'b0;
    enable <= 1'b1;
  end
  else
  begin
    counter <= counter + 1'b1;
    enable <= 1'b0;
  end
end
```

Slika 18: Verilog HDL opis vezja za nižanje frekvence

S prvo vejitvijo na sliki 18 sem vhodno frekvenco 120 MHz znižal na 30 MHz in sicer z deljenjem s 4. Na vsak četrti pulz sem spremenil vrednost spremenljivke *enable*, ki je s svojim spreminjanjem narekovala aktivnost vezja za generiranje VGA signala.

```
always @(posedge clock)
begin
  if (enable == 1)
  begin
    if(hcount == 799)
```

```
    begin
      hcount <= 0;
      if(vcount == 524)
        vcount <= 0;
      else
        vcount <= vcount+1'b1;
    end
    else
      hcount <= hcount+1'b1;

  if (vcount >= 490 && vcount < 492)
    vsync <= 1'b0;
  else
    vsync <= 1'b1;

  if (hcount >= 656 && hcount < 752)
    hsync <= 1'b0;
  else
    hsync <= 1'b1;
  end
end
```

Slika 19: Implementacija VGA grafičnega vmesnika

V nadaljevanju (na sliki 19) sem s pomočjo funkcije prožilnega pulza in vejitve izločil mrtve zone in generiranje barv, tako da sem ostal samo z vertikalno in horizontalno pozicijo pikslov pri katerih smem pošiljati želeno vrednost.

3.2.2 Delovanje igre Pong

Pri izdelavi igre sem se odločil, da bom za vsak objekt naredil svoj modul. Tem modulom sem posredoval informacijo o časovnem pulzu, stanju gumbov (kjer je to potrebno) in horizontalno ter vertikalno pozicijo piksla, ki se trenutno izrisuje na zaslonu.

Logika, potrebna za implementiranje pravil igre

```
ball ball_inst
    (
    .clock(clock),
    .red(R4),
    .green(G4),
    .blue(B4),
    .hcount(hcount),
    .vcount(vcount),
    .enable(enable),
    .layer(L4),
    .inp3(inp3),
```

```
    .inp4(inp4),
    .maks(maks),
    .smerx(smerx),
    .score1(score1),
    .score2(score2)
    );
```

Slika 20: klic modula za žogo

Modul na sliki 20 je lahko nazaj vrnil R, B in G vrednost posameznih pikslov, plast na kateri se ta barva nahaja in določene globalne spremenljivke (npr. rezultata igre *score1* in *score2*).

Spodaj je prikazan primer delovanja žogice, ki se od odbija od robov ustvarjenega poligona in loparjev, razdeljenega na tri dele. Pred samimi odločitvami sem ob vsakem pulzu odštel ena in po petih milijonih odštetih pulzov števec pride do ničle in takrat se izvede vejitev.

```
always @(posedge clock)
begin
    accumx <= (ppsx ? 5_000_000 : accumx) - 1;

    if (ppsx)
     begin
```

Slika 21: Implementacija frekvenčnega delilnika za objekte v igri

Znotraj te vejitve sem uporabil še dve. Prva od njiju, prikazana na naslednji sliki se nanaša na premikanje žogice. Njeno gibanje se spreminja glede na to ali se je odbila od zgornjega ali od spodnjega roba, kar pove spremenljivka *smery*.

```
if (smery == 1)
begin
    county <= county + 3;
end
else if (smery == 0)
begin
    county <= county - 3;
end
```

Slika 22: Premikanje objekta v igri

V naslednjem delu pa pregledujem ali je šla žogica skozi izbrano mejno koordinato. V tem primeru prišteje točko nasprotniku in žogico prestavi na sredino zaslona (slika 23).

```
//score
```

```
if ((countx <= 55 ) )
begin
    score1 <= score1 + 1;
    countx <= 250;

end
else if ((countx >= 570) )
begin
    score2 <= score2 + 1;
    countx <= 250;
end

if ((score1 >= 5) && (inp4 == 0))
begin
    score1 <= 0;
    score2 <= 0;
end
else if ((score2 >= 5) && (inp3 == 0))
begin
    score1 <= 0;
    score2 <= 0;
end
```

Slika 23: Štetje rezultata igre

Določanje pozicije in barve objekta

Za izrisovanje žogice na zaslonu je potrebno povedati pri katerih horizontalnih in vertikalnih vrednostih naj bo objekt izbrane barve. To sem izvedel z vejitvijo na katero sem podal zgornji levi in spodnji desni rob kvadratka, za poenostavitev sem izbral žogico »kvadratne« oblike. Če z izrisovanjem piksla nismo ravno v žogici, za barvo ozadja privzamemo črno (slika 24).

```
always @ (posedge clock)
begin
  if (enable)
  begin
      if ( (60 < hcount && hcount < 70) &&  (vcount > count) && (vcount < 100 +
count ) ) //Kvadratek
    begin
      green <= 3'b111;
      blue <= 2'b11;
      red <= 3'b111;
    end
```

```
    else
    begin
      green <= 3'b000;
      blue <= 2'b00;
      red <= 3'b000;
    end
  end
end
```
Slika 24: Primer implementacije izrisovanja objekta

Premikanje loparčkov

Ker igra Pong oponaša klasično namizno igro Ping-pong imamo seveda dva loparčka, enega na levi in enega na desni strani.

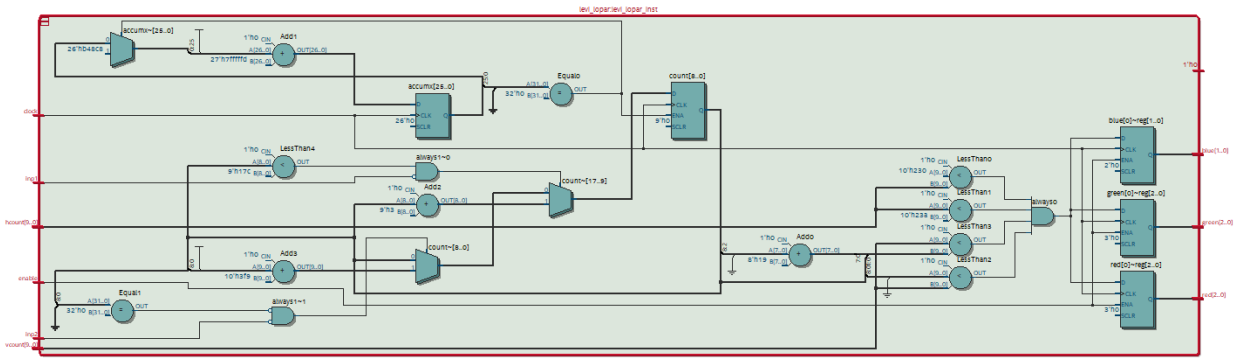
Sedaj lahko pogledamo primer v katerem so združeni nekateri zgoraj razloženi principi in sicer s primerom modula levi_lopar (slika 19). V prvi vrstici lahko vidimo stavek: »*`timescale 1ns / 1ps«*. Ta ne vpliva na delovanje modula oz. generiranje strojne kode, saj je namenjen simulatorju vezja. Naprej sledi deklaracija modula in signalov, ki jih modul potrebuje. Preden sem nadaljeval z logiko vezja, sem napisal lokalne spremenljivke in njihove velikosti v register. Znotraj modula vidimo dve sprožilni vezji. Prvo vezje je namenjeno prenašanju barve in koordinate objekta modulu za VGA protokol, drugo pa opisuje gibanje objekta. Prvo je sinhronizirano z VGA modulom, kateremu poda informacije o barvah piksla pri danih koordinatah. Koordinate objekta se lahko spremenijo, saj je vertikalna pozicija odvisna od spremenljivke *count.* Spremenljivko spreminjamo s pomočjo drugega sprožilnega vezja na približno 0,2 µs. Spreminjanja spremenljivke nisem sinhroniziral z VGA modulom, saj je njegovo delovanje prehitro in bi povzročilo prehitro spreminjanje vrednosti ter posledično navidezno zameglitev prikazanega objekta na zaslonu. Zameglitev je navidezna, saj objekt ni zamegljen, ampak se prehitro premika in s tem zamenja na videz tekoče gibanje z odsekovnim premikanjem objekta. Pri vsaki meritvi najprej preverim vrednost prvega gumba in pozicije loparja. Ko je objekt znotraj spodnje meje poligona in ko je pritisnjen prvi gumb *inp1*, prištejem spremenljivki *count* vrednost 3 in s tem objekt premaknem za tri piksle navzgor. Podobno sem naredil tudi s premikanjem navzdol, samo da tokrat preverim stanje gumba za navzdol *inp2* in s tem odštejemo od spremenljivke *count vrednost 3*. V primeru, kjer sta obe tipki pritisnjeni, bo vedno prevladala tipka za navzgor *inp1*.

```
`timescale 1ns / 1ps
module levi_lopar(
  input clock,
  output [2:0] red,
  output [2:0] green,
  output [1:0] blue,
  input reg [9:0] hcount,
  input reg [9:0] vcount,
  input enable,
  output layer,
  input inp1,
  input inp2
);

reg [25:0] accumx = 0;
reg [8:0] count = 0;

wire ppsx = (accumx == 0);
assign layer = 0;

always @ (posedge clock)
begin
  if (enable)
  begin
      if ( (560 < hcount && hcount < 570) &&  (vcount > count) && (vcount < 100 +
count ) ) //Kvadratek
    begin

      green <= 3'b111;
      blue <= 2'b11;
      red <= 3'b111;
    end
    else
    begin

      green <= 3'b000;
      blue <= 2'b00;
      red <= 3'b000;
    end
  end
end

always @(posedge clock)
begin
    accumx <= (ppsx ? 5_000_000 : accumx) - 1;
```

```
    if (ppsx)
     begin

        if ((inp1 == 0) && (count < 380)) // 480
        begin

            count <= count + 3 ;
        end
        else if ((inp2 == 0) && !(count == 0))
        begin

            count <= count - 3 ;
        end
    end
end
endmodule
```

Slika 25: Levi loparček

Verilog HDL je opisni jezik, v kateremu povemo katere elemente vezje ima in kako so med seboj povezani. Vsako tako vezje lahko tudi prikažemo v bolj znani obliki z logičnih vrat in integriranimi logičnimi vezij. Program Quartus Pro nam omogoča tudi prikaz opisanega vezja na sliki 19 v obliki električne sheme, kot prikazuje slika 26.



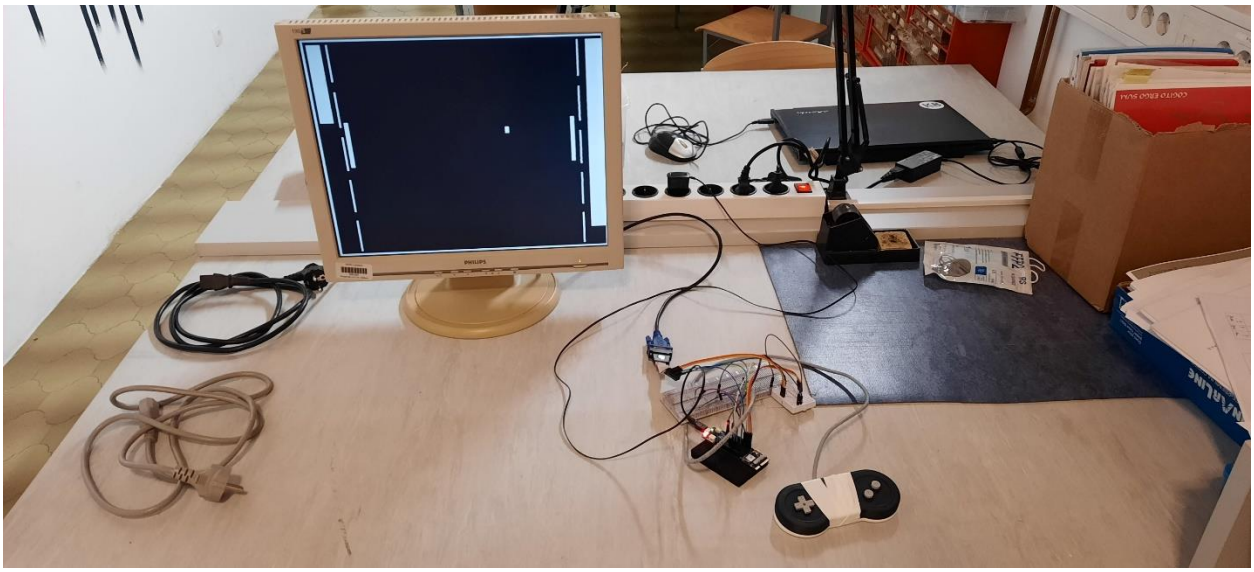
Slika 26: Slika shematike digitalnega vezja levi_lopar

Prednost predstavitve vezja v obliki električne sheme je hitro razumevanje in preprostejša optimizacija, saj lahko vidimo način komponente in procese, ki zavzamejo največ časovnih pulzov. Pri programiranju sem velikokrat tudi naletel na težavo, saj koda ne javi napak ampak opozorila za nezaključene spremenljivke. Pri učenju opisovanja vezja se FPGA velikokrat ni obnašal kot sem predpostavljal. Razlogov je bilo veliko vendar dva glavna sta mi vzela največ časa.

Ena izmed napak je bila, da sem se zmotil v interpretaciji problema. Druga napaka pa nedeklarirane spremenljivke. V obeh primerih prevajalnik kode ne javlja problema, saj pri prvi napaki ni nič narobe s samim pravopisom, v drugi pa nedeklarirane spremenljivke avtomatsko ozemlji. Z boljšim znanjem delovanja kode bi te probleme lahko rešil tudi brez sheme vezja, vendar pa mi je v takih trenutkih shema močno pomagala.

Sheme digitalnega vezja imajo tudi eno glavno slabost in sicer so slabo pregledne. Kompleksnejše sheme so velikokrat nepregledne in jih je poleg tega tudi težje urejati. Večja vezja zasedajo velik predel zaslona, zato nam njihova velikost velikokrat prepreči sočasen pogled na spremenljivko in objekt, na katerega je povezana. To je bila ena izmed prelomnih točk, ki me je prepričala, da sem raje uporabil opisni jezik.

3.3 Pong



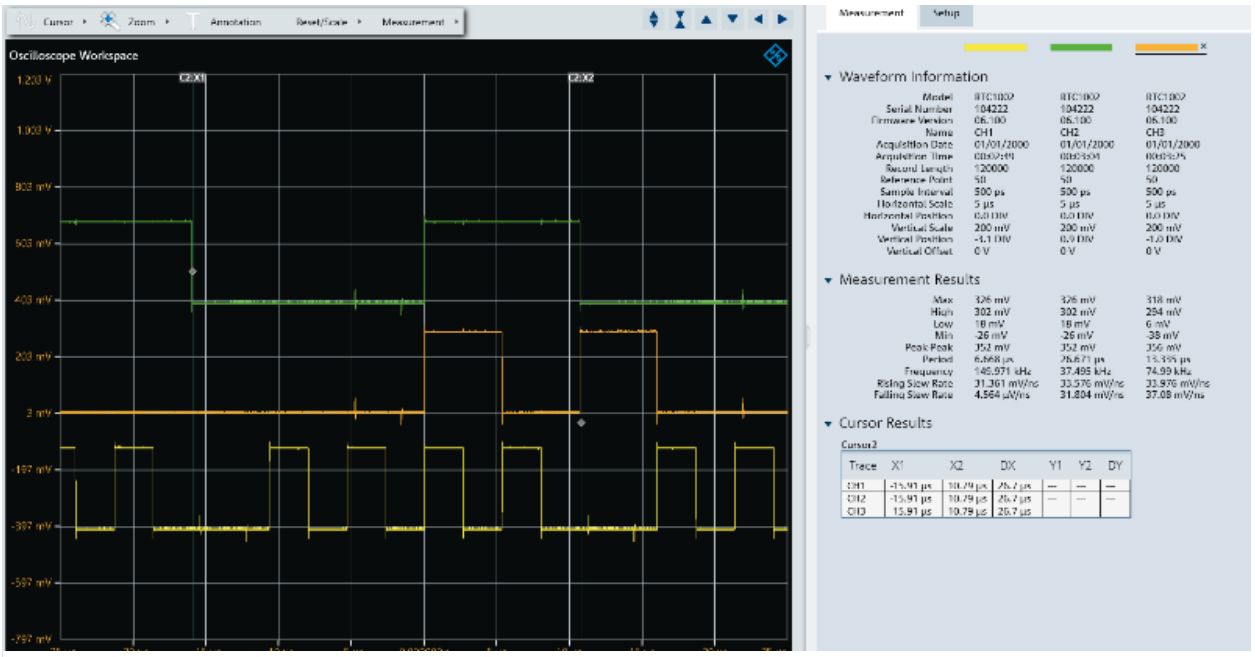
Slika 27: Prikaz delovanja igre Pong

Na zgornji sliki lahko vidimo implementacijo igre Pong, kot je opisana v prejšnjih poglavjih. Vezje sem sestavil na testni plošči in ga priključil na razvojno ploščo Arduino MKR Vidor 4000, saj je namen bil raziskovalen in je zahteval adaptacijo vezja po potrebi. S tem sem tudi žrtvoval stabilnost in čistost signala, saj so slabi stiki v testni ploščici ali med priključki povzročali motnje. Pri testiranju sem uporabljal dva zaslona, enega starejšega (PHILIPS 10S6) in enega novejšega (DELL SE2422H). Na novejšem se teh motenj ni videlo pri starejšem pa so se pojavile v obliki

zamikanja (»plavanja«) vrstic, kar sem pripisal nepravilnostim oz. motnjam v horizontalnem signalu (sliki 29 in 28.

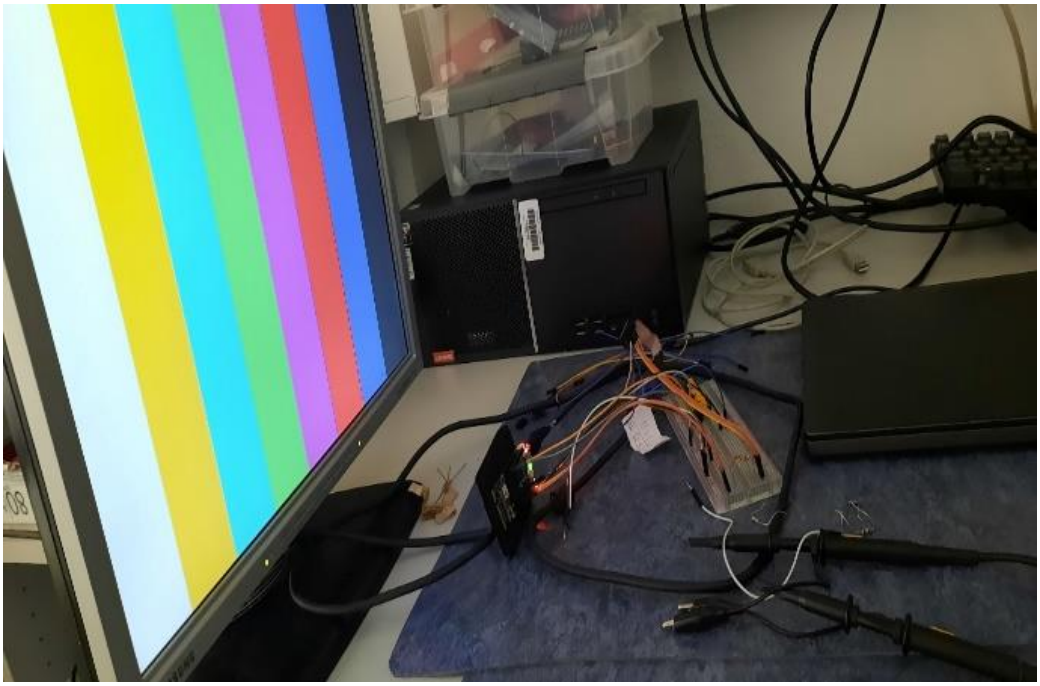
Slika 29: Motnje na starejšem zaslonu

Slika 28: Novejši zaslon, brez motenj



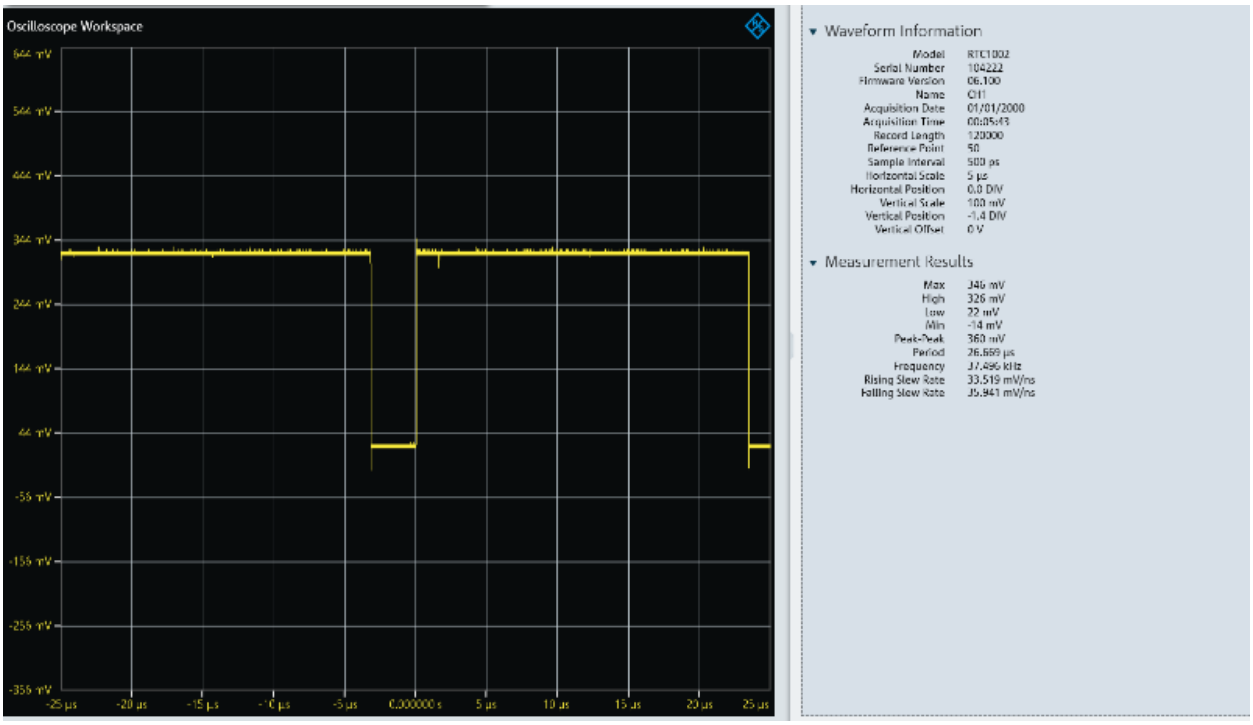
Slika 30: Merjenje R, G, B vrednosti z osciloscokom

Slika 30 prikazuje R, B in G signale barvnega zaslona, kot ga vidimo sliki 31. Za lažjo vizualizacijo imajo signali na osciloskopu imajo ničlo rahlo zamaknjeno.

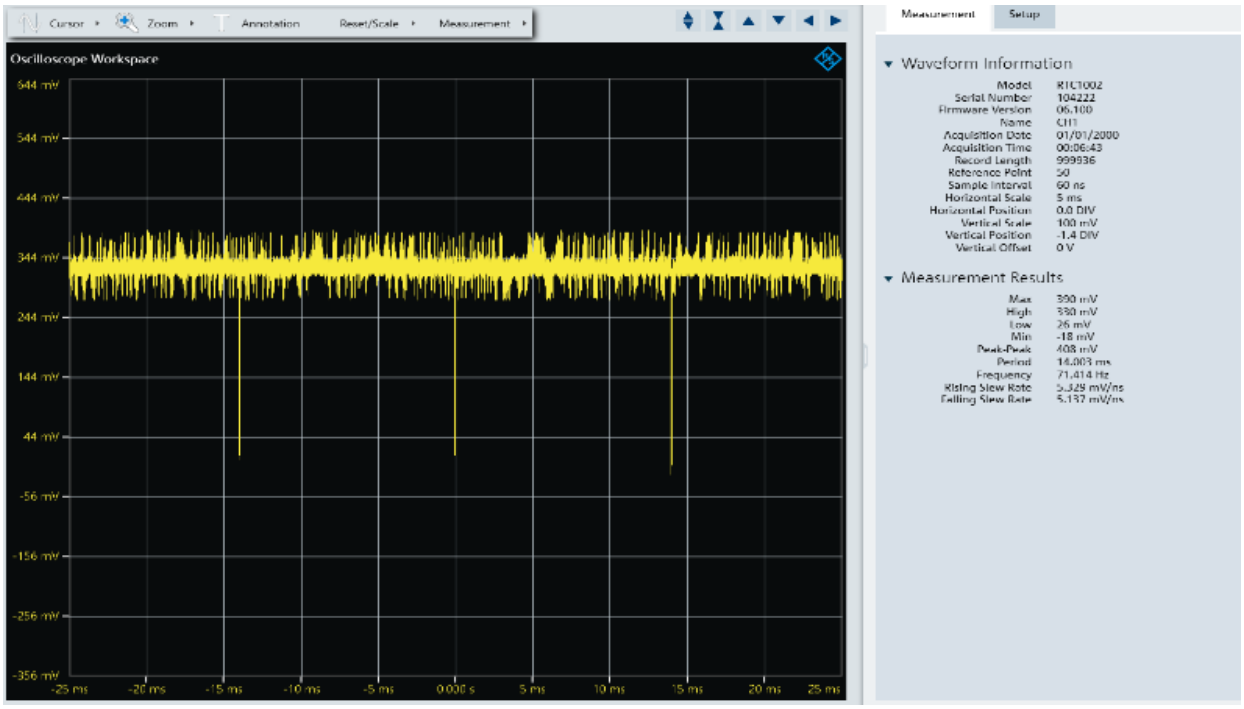


Slika 31: Test R, G, B vrednosti

Na sliki 32 vidimo horizontalen sinhronizacijski pulz, s katerim ločujemo med posameznimi vrsticami slike.

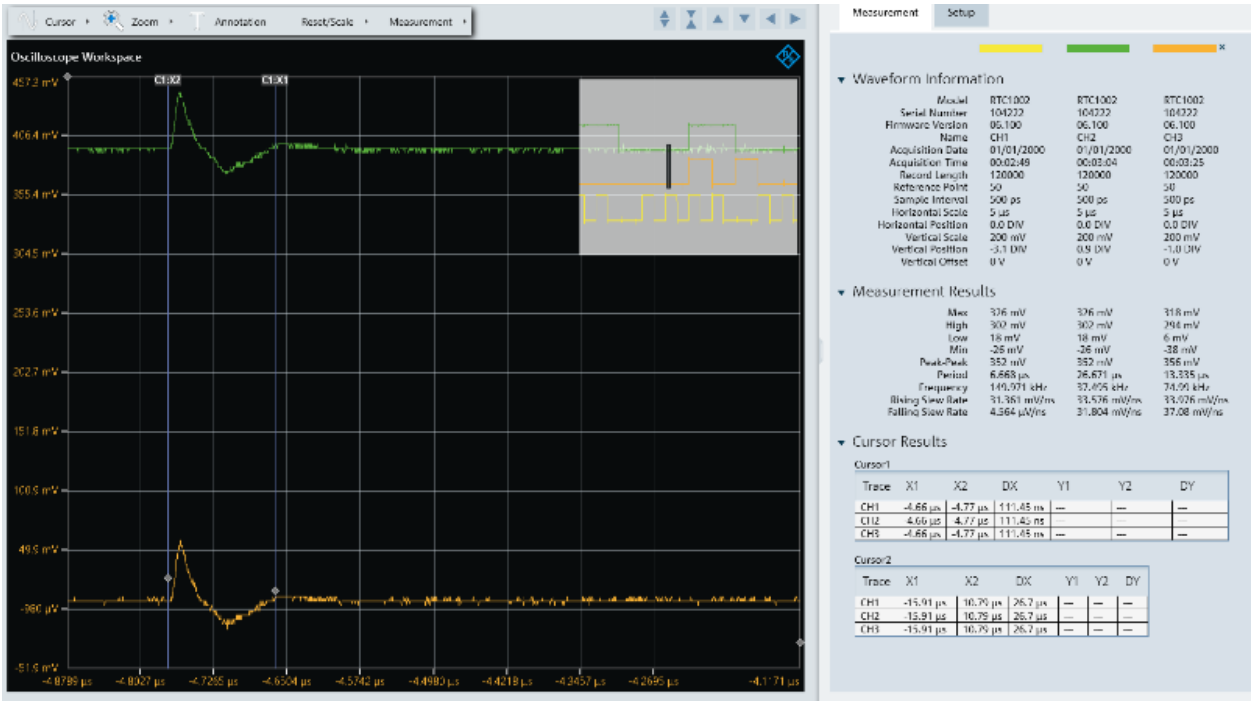


Slika 32: Horizontalen sinhronizacijski pulz



Slika 34: Vertikalni sinhronizacijski pulz

Slika 34 prikazuje osveževalo frekvenco zaslona oz. ti. vertikalni sinhronizacijski signal. Ker je razmerje med prižganim in ugasnjenim stanjem pulza zelo veliko, izgleda ugasnjeno stanje kot daljica. Signal se v resnici spusti dol vse do 5 mV.



Slika 33: Povečava motenj na G in B signalih

Pri opazovanju signala sem opazil periodično pojavljanje enakih motenj na R, G in B signalih (slika 33). Ponavljajoča narava teh šumov in dejstvo da na horizontalnem časovnem pulzu tega šuma ni, pomeni da to niso prisluhi ampak nekaj drugega. Tvoril sem že nekaj osnovnih hipotez ampak, sem se odločil, da bom reševanje tega problema pustil za kasnejši čas.

4. Zaključek

Pri raziskovalni sem se naučil novega programskega jezika Verilog HDL in implementacijo grafičnega standarda VGA. Uspešno sem sprogramiral igrico Pong in jo realiziral z prijetnejšim uporabniškim vmesnikom oz. z igralno konzolo. Projekt mislim nadaljevati in celo razširiti. Najprej bi uporabil višji resolucijski standard in več vrst barv. Potem se bom osredotočil na branje in zapisovanje podatkov v RAM, v katerega bom shranjeval podatke o ozadju uporabljenem v igri.

Tukaj bi močno poudaril predznanje logičnih vrat in digitalnih vezij, ki se jih učimo pri pouku, saj mi je pomagal pri razumevanju delovanja vezja. Razvijalna plošča kot je Arduino MKR 4000 nam omogoča simulacijo in realizacijo zahtevnejših digitalnih vezij brez bojazni, da ima našo vezje probleme s čipi, kabli ali med samimi stiki pri povezovanju le teh. S tem lahko tudi preskočimo predel kjer je potrebno predhodno naročevati testna integrirana vezja, ampak se zanje lahko uporabi kar FPGA.

4.1 Potrjene hipoteze

1. Delo z FPGA vezji je dovolj enostavno, da bi ga lahko uporabljali tudi že v srednji šoli.
   - To hipotezo sem potrdil, saj sem se sam naučil uporabljanja in programiranja FPGA vezja.
2. FPGA vezja so dovolj hitra, da lahko z njimi implementiramo VGA grafični standard.
   - To hipotezo sem potrdil, slika je stabilna in v barvah.
3. Dosegli bomo resolucijo vsaj 640 x 480 pikslov v osmih barvah.
   - Hipotezo sem potrdil s prikazom zaslona z barvnimi progami.
4. Vezje je mogoče pospešiti z dodatno manipulacijo časovnega pulza.
   - Pri ogledovanju kode sem opazil, da je PLL že omogočen znotraj testne kode, tako da se nisem izrecno poglabljal v dodatno pospeševanje vezja.

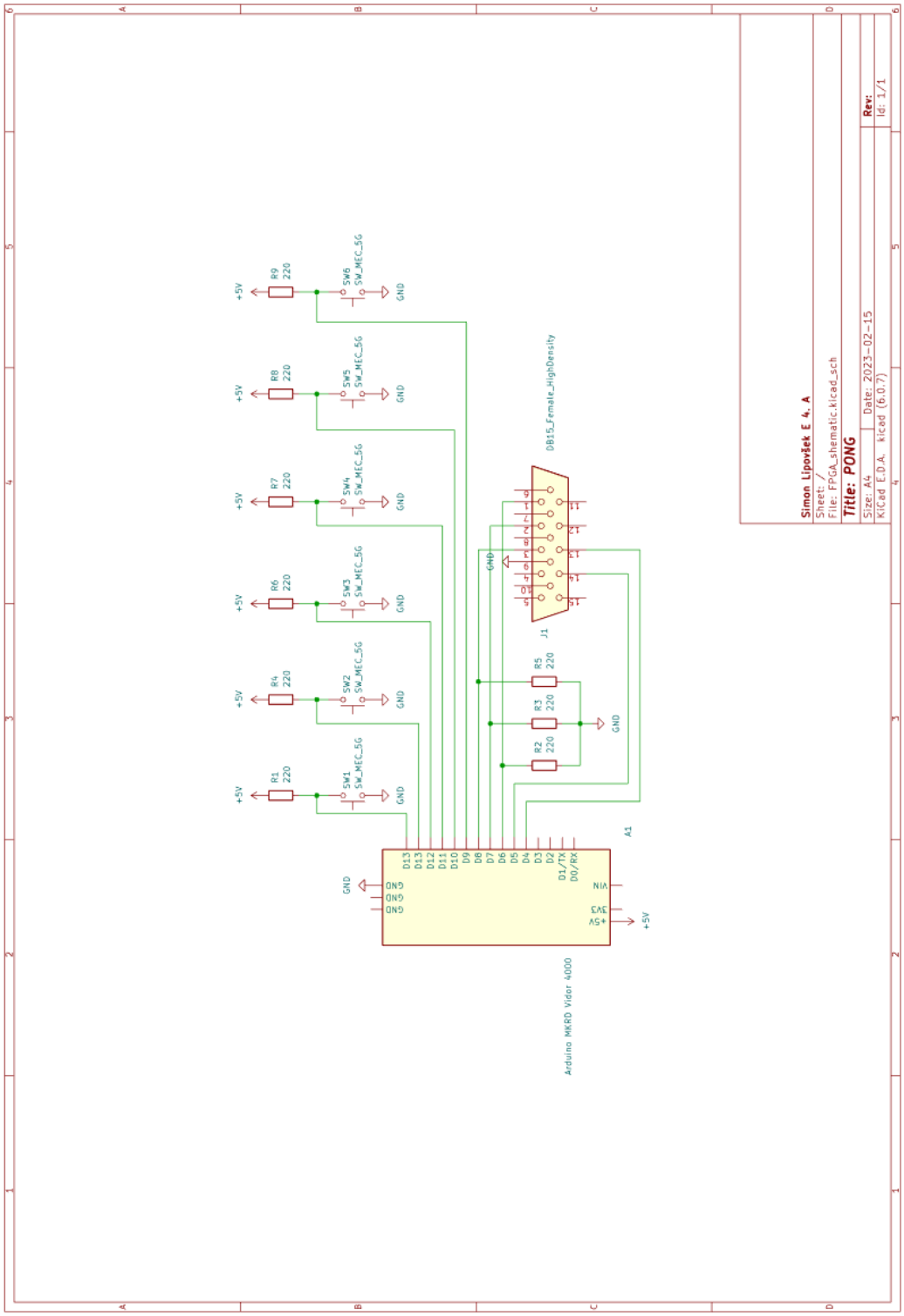
5. Predvidevam, da bom lahko generiral sliko in izvajal računske operacije sočasno s prikazovanjem signala po VGA grafičnem standardu.
    - Hipotezo sem potrdil s prikazom igre Pong. Igra poteka na sliki resolucije 640 x 480 v osmih barvah pri osveževanju 72 Hz.

4.2 Izvorna koda

Izvorna koda je dosegljiva na naslednji povezavi: https://github.com/Pizmuh/FPGA_PONG.git

5. Priloge

5.1 Shematika vezja



6. Viri in literatura

6.1 Literatura

[1] Jari Beguš. ***Pomnilniški vmesnik za pomnilnik DDR3 SDRSAM v vezju FPGA XILINX serije 7***. Magistrsko delo, Univerza v Ljubljani, Fakulteta za elektrotehniko, Ljubljana 2022.

[2] Metod Langus. ***Znakovni terminal za mikroprocesorje v vezju FPGA.*** Diplomsko delo, Univerza v Ljubljani, Fakulteta za elektrotehniko, Ljubljana, 20022.

[3] Samir Palnikat, 10. feb 2023, ***Verilog HDL, 2nd Edition,*** dostopno prek: https://www.informit.com/store/verilog-hdl-9780130449115

[4] **IEEE Standard for Verilog Hardware Description Language**, IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001) , vol., no., pp.1-590, 7 April 2006, doi: 10.1109/IEEESTD.2006.99495.

[5] Clifford E. Cummings, **SystemVerilog - Is This The Merging of Verilog & VHDL?,** Sunburst Design, Inc.

6.2 Viri

[6] Laboratorij za biotehniko. ***Digitalne strukture 7. Vaja.*** Univerza v Ljubljani. Dostopno prek: https://lbk.fe.uni-lj.si/ds/DS-vaja-7-predstavitev.pdf (16. 1. 2023).

[7] Tahustvedt. ***Gamepad for Atari***. Dostopno prek: https://cults3d.com/en/3d-model/various/gamepad-for-atari (16. 1. 2023).

[8] Intel. ***Adder/Subtractor***. Verilog arihmetic functions. Intel. Dostopno prek: https://www.intel.com/content/www/us/en/support/programmable/support-resources/design-examples/horizontal/verilog.html (16. 1. 2023).

[9] TinyVGA. ***VGA Signal 640 x 480 @ 60 Hz Industry standard timing***. VGA Microcontroller projects. SECONS Ltd. Dostopno prek: http://tinyvga.com/vga-timing/640x480@60Hz (16. 1. 2023).

[10] Rohitsingh. ***Simple VGA design example for Telestio***. Getting started with FPGA. Numato lab. Dostopno prek: https://numato.com/kb/simple-vga-design-example-for-telesto/ (16. 1. 2023).

[11] Will Green. ***Video Timings: VGA, SVGA, 720p, 1080p***. Project F. Dostopno prek: https://projectf.io/posts/video-timings-vga-720p-1080p/ (16. 1. 2023).

[12] Intel. ***Adder/Subtractor***. Verilog arihmetic functions. Intel. Dostopno prek: https://www.intel.com/content/www/us/en/support/programmable/support-resources/design-examples/horizontal/verilog.html (16. 1. 2023).

[13] Will Green. ***Begining FPGA graphic.*** Project F: FPGA Dev. Dostopno prek: https://projectf.io/posts/fpga-graphics/ (16. 1. 2023).

[14] FPGA4fun. ***Pong Game***. Basic projects. Dostopno prek: https://www.fpga4fun.com/PongGame.html (16. 1. 2023).

[15] Toshiba. ***What types of transistors are avalible?***. Toshiba Electronic device & storage corporation.
Dostopno prek: https://toshiba.semicon-storage.com/ap-en/semiconductor/knowledge/faq/mosfet_common/what-kinds-of-transistors-exist.html (16. 1. 2023).

[16] Basic electronic tutorials. ***Simple programable logic devices SPLD***. Programable logic devices.
Dostopno prek: https://www.electronics-tutorial.net/Programmable-Logic-Device-Architectures/Programmable-Logic-Devices/Simple-Programmable-Logic-Devices-SPLD/ (16. 1. 2023).

[17] Digilent refrence. ***Verilog HDL background and history***. Digital logic.
Dostopno prek: https://digilent.com/reference/learn/fundamentals/digital-logic/verilog-hdl-background-and-history/start (16. 1. 2023).

[18] Rich Miron. ***Osnove progrmirljivih matrik logičnih vrat (FPGA) – kaj so in kaj je potrebno za njih***. Verilog arihmetic functions. Svet elektronike.
Dostopno prek: https://svet-el.si/revija/predstavljamo/2020-281-24/ (16. 1. 2023).

[19] **Računalniški slovarček**. Univerza v Ljubljani za računalništvo in informatiko.
Dostopno prek: https://dis-slovarcek.ijs.si/list_searched (16. 1. 2023).

[20] ***Osnove programa Quartus Prime***. Univerza v Ljubljani, Fakulteta za elektrotehniko.
Dostopno prek: https://lniv.fe.uni-lj.si/altera/quartus.htm (16. 1. 2023).

[21] Kevin moris. ***Quartus Prime***. Electronic engineering journal.
Dostopno prek: https://www.eejournal.com/article/20151110-quartus/ (16. 1. 2023).

[22] Dr. Shirshendu Roy. **Interfacing VGA display with FPGA**. Digital system design.
Dostopno prek: https://digitalsystemdesign.in/interfacing-vga-display-with-fpga/ (16. 1. 2023).

[23] pndra. ***VidorFPGA***. Github.
Dostopno prek: https://github.com/vidor-libraries/VidorFPGA/ (16. 1. 2023).

[24] Digi-Key. ***Introduction to FPGA****, Part 9: PLL and Glitches*.
Dostopno prek: https://www.digikey.si/en/maker/projects/introduction-to-fpga-part-9-phaselocked-loop-pll-and-glitches/2028ce62001b4cb69335f48e127fa366 (16. 1. 2023).

[25] Arduino. ***Arduino MKR Vidor 4000***.
Dostopno prek: https://store.arduino.cc/products/arduino-mkr-vidor-4000 (16. 1. 2023).

[26] tutorialspoint. ***Programmable logic array PLA***. Programmable logic devices. Learn digital circuits.
Dostopno prek:
https://www.tutorialspoint.com/digital_circuits/digital_circuits_programmable_logic_devices.htm (16. 1. 2023).

[27] Get intimate With cyber!. ***A crypto FPGA project using an Arduino***.
Dostopno prek: https://ubs_csse.gitlab.io/secu_os/tutorials/arduivium.html (1. 3. 2023).