Univerza *v Ljubljani*
Fakulteta *za elektrotehniko*

Poročilo pri predmetu načrtovanje digitalnih elektronskih sistemov

Osnovna digitalna obdelava slike s FPGA

Avtor: Gašper Šavle
Program: Aplikativna elektrotehnika
3. letnik – Avtomatika
Vpisna številka: 64190318

Koper, januar 2023

Povzetek

Naloga zaključnega projekta je bila nadgradnja mikroračunalnika, ki smo ga pri vajah naredili na FPGA razvojni ploščici Altera de0 - nano. Moj cilj je bil predstaviti barvno sliko z ASCII znaki, glede na svetlost posameznega piksla. Kasneje sem se projekt odločil nadgraditi z osnovno izvedbo digitalne obdelave slike.

Kazalo

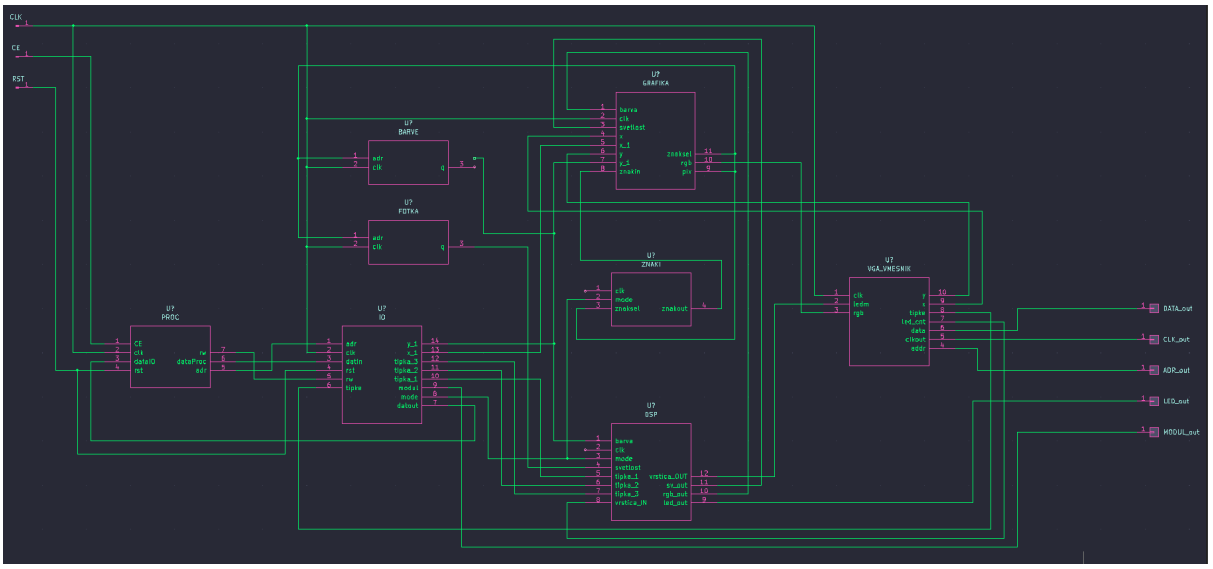
1. Delovanje sistema

Sistem je sestavljen iz 8 blokov, od katerih so 4-je bili dodani kot razširitev za moj zaključni projekt. Dodal sem bloke:

- dsp
- znaki
- barve
- fotka

Med bloki je bilo treba uskladiti tudi ustrezne povezava, pri tem pa je bilo treba paziti na podatkovno širino informacij, ki smo jih želeli prenašati med posameznimi moduli.

2. Blokovna shema sistema



Shema blokov in povezav

Blokovna shema je precej zajetna, zato priporočam ogled izvorne datoteke v orodju KiCAD ali vključen posnetek zaslona z orodjem za ogledovanje slik.

3. Bloki v sistemu

5.1. PROCPAK

"PROCPAK" opravlja konverzijo iz ukazov v zbirniku v strojno kodo, ki jo lahko razume naš procesor. To nam omogoča programiranje v nekoliko višjem programskem jeziku. Komponenta je bila vključena že tekom izvajanja laboratorijskih vaj.

5.2. PROC

Usmerja signale med IO in procesorjem, skrbi za zelo nizkonivojske signale, kot je na primer RESET. Tako, kot blok pred njim, je tudi proc že bil vključen v osnovo za projekt.

5.3. CPU

Blok CPU smo začeli izdelovati že na vajah, kjer je bil cilj implementirati osnovne assembly funkcije (LDA, ADD, STA…). CPU upravlja z akumulatorjem in registri glede na ukaz, ki ga prejme. Deluje v 2 stanjih, naloži in izvedi (fetch, execute). Glede na ti dve stanji upravlja s signaloma "read enable" in "write enable", ima pa še tretje stanje "reset", ki nastavi programski števec nazaj na 0. Ob vsakem zajemu novega ukaza se programski števec (PC) poveča za 1.

5.4. VGA vmesnik

Ta blok skrbi za takt VGA signala in premikanje po naslovih zaslona. Skrbi, da se nahajamo v vidnem območju ekrana in skrbi za povezavo podatkov na ustrezen pin VGA konektorja. Zadolžen je tudi za prikazovanje podatkov na LED matriki, torej opravlja 7X5 multipleksiranje podatkov.

5.5. SISTEM

Blok "SISTEM" je najvišji v hierarhiji našega projekta, v njem so izvedene vse povezave med signali posameznih komponent. Vsebuje tudi delilnik ure, ki upravlja s "clock enable" signalom komponente "PROC".

```
-- Grafika
U3: entity work.Grafika port map (
    clk => clk,
    x => x,
    y => y,
    x1 => x1,
    y1 => y1,
    trk => trk,
    pix => naslov,
    svetlost => svout,
    barva => rgbout,
    znaksel => znaksel,
    znaksel1 => znaksel1,
    znakin => znakout,
    znakin1 => znakout1,
    rgb => rgb );
```

```
-- DSP
U7: entity work.dsp port map(
    clk => clk,
    mode => mode,
    tipka_1 => tipka_1,
    tipka_2 => tipka_2,
    tipka_3 => tipka_3,
    svetlost => svetlost,
    barva =>  barva,
    svout => svout,
    rgbout => rgbout,
    ledout => ledout,
    vrsticain => numvrst,
    vrsticout => vrsticin
    );
```

Povezava signalov med bloki

5.6. IO

Blok je zadolžen za pretok podatkov v procesor in iz njega, v njem je opisana logika komunikacije s tipkami in izmenjava podatkov z assembly programom.

V mojem primeru na 4 spremenljivke vpisuje vrednosti iz assembly programa, te se nato glede na vrednost signala “mode” različno tolmačijo v komponenti “DSP”

```
case rw is
when "01" =>
  case adr is

  -- Ob naslovu 1 se posodobi nacin delovanja dsp
  when "00000001" =>
    mode <= datin(1 downto 0);

  -- Ob naslovu 2 se v spremenljivko tipka_1 zapise njena vrednost
  when "00000010" =>
    tipka_1 <= std_logic_vector(datin(3 downto 0));

  -- Ob naslovu 3 se v spremenljivko tipka_2 zapise njena vrednost
  when "00000011" =>
    tipka_2 <= std_logic_vector(datin(3 downto 0));

  -- Ob naslovu 4 se v spremenljivko tipka_3 zapise njena vrednost
  when "00000100" =>
    tipka_3 <= std_logic_vector(datin(3 downto 0));

  -- Ko je naslov 0 se procesorju sporoci modul delilnika frekvence.
  when "00000000" =>
    modul <= datin;

  when others => null;
end case;
```

Razvrščanje signalov po naslovih

5.7. GRAFIKA

Komponenta “GRAFIKA” je najobsežnejša v celotnem projektu, skrbi za prikazovanje vsega na zaslonu. Ob urinem pulzu preverja lokacijo signalov X in Y, ki ju prejme iz komponente “VGA vmesnik”. Sliko začne prikazovati šele, ko se nahajamo znotraj definiranega vidnega območja. Opravlja transformacije signalov za premikanje komponent po zaslonu in naslavljanje spominskih blokov. S transformiranima in bitno zamaknjenima vrednostima X in Y program naslavlja pomnilnik, ki hrani svetlosti posameznega piksla na sliki. Nato glede na svetlost določi gostoto ASCII znaka, s katerim bo prikazal ta piksel. Sočasno z branjem svetlosti, komponenta grafika bere še drug spominski blok, ki pa hrani barve pikslov. Svetlosti in barve nato posreduje komponenti “DSP”, ki jih obdelane vrne nazaj v komponento “GRAFIKA”, ta nato izpiše znak ustrezne gostote in ustrezne barve.

5.8 . DSP

Komponenta “DSP” je nadgradnja projektu, deluje kot vmesnik med podatki, prebranimi iz spominskih blokov in komponento “GRAFIKA”. Glede na vrednost dvobitnega signala “mode” obdela podatek na drugačen način, poznamo 4.

MODE “00”: threshold

DSP primerja svetlost prebrano iz spominskega bloka z vrednostjo tipk 2 in 3. Tipka 2 v tem načinu delovanja predstavlja spodnjo mejo svetlosti, DSP torej “zavrže” prebran podatek, če je njegova svetlost manjša od vrednosti tipke 2. Zavrže pa ga tako, da namesto njegove prave svetlosti, komponenti “GRAFIKA” sporoči svetlost “0”. Prav tako zavrže svetlosti, ki so višje od vrednosti tipke 3, ki v tem načinu delovanja predstavlja zgornjo mejo prikazanih svetlosti. V tem načinu delovanja torej prikazujemo le svetlosti, ki so višje od spodnje meje in nižje od zgornje meje, sicer je prikazan piksel črn.

```
if (tipka_2 <= svetlost(6 downto 3) and svetlost(6 downto 3) <= tipka_3)then
    svout <= svetlost;
else
    svout <= "00000000";
end if;
rgbout <= barva;
```

Primerjava signala svetlost z mejnima vrednostima

MODE “01”: filtri

DSP primerja vrednosti posamezne komponente barve, prebrane iz spomina. Barva je predstavljena, kot 6 - bitni vektor, kjer prva 2 bita predstavljata rdečo, druga 2 zeleno in zadnja 2 bita modro barvo. Skupno nam to omogoča 31 barv (plus črno). Prebrano vrednost torej “zrezemo” na 3 posamezne dvo- bitne komponente, ki jih potem DSP primerja z vrednosti tipk 1, 2 in 3.
Če je vrednost posamezne komponente manjša od vrednosti doticne tipke, DSP to komponento nastavi na “00”, torej efektivno odreže to komponento barve.

```
-- Primerjava rdece
if barva(5 downto 4) <= red then
    rgbout(5 downto 4) <= "00";
else
    rgbout(5 downto 4) <= barva(5 downto 4);
end if;
```

Prireditev posameznih bitov signala barva

MODE “10”: svetlost

DSP svetlosti, prebrani iz spomina, prišteje oziroma odšteje vrednost tipk 2 ali 3.

```
when "10" =>
-- Svetlost
    visaj(7 downto 4) <= unsigned(tipka_3);
    nizaj(7 downto 4) <= unsigned(tipka_2);

    svout <= std_logic_vector(unsigned(svetlost) + visaj - nizaj);
    vrsticout <= matrika2(to_integer(vrsticain));
    rgbout <= barva;
```

Povečanje in zmanjšanje svetlosti

MODE “11”: negativ

V tem načinu delovanja DSP zamenja prebrano svetlost in barvo izvornega piksla iz spominske komponente z njunima negiranima vrednostima.

```
when "11" =>
-- Negativ
    svout <= not(svetlost);
    rgbout <= not(barva);

    vrsticout <= matrika3(to_integer(vrsticain));
```

Negacija svetlosti in barve

6. Program za procesor v zbirniku

```
zac:
    lda 4095
    sta mod
    outp 0

loopmod: inp btnmod
    sta novomod
    lda novo3
    sta staro3
    lda staromod
    nota staromod
    anda novomod
    sta rezmod

    jze loopbtn1
    lda mode
    add 1
    sta mode
    outp 1
    lda novomod
    sta staromod
    jmp loopbtn1
```

Assembly koda za eno tipko

Program ob vstopu nastavi spremenljivko “mod” na maksimalno vrednost in jo izvozi na izhod 0, ki je v VHDL kodi povezana z delilnikom ure grafike. Nato vstopi v zanko “loopmod”, ki v spremenljivko “novomod” shrani stanje tipke 0 (btnmod), nato opravi logično operacijo AND z negirano vrednostjo spremenljivke “staromod”, ki predstavlja prejšnje stanje tipke 0 (btnmod). Rezultat operacije je 1, če sta vrednosti različni in 0, ce sta vrednosti enaki. Ta rezultat se zdaj shrani v spremenljivko “rezmod”. Naslednja operacija je vejitev, če je trenutno naložena vrednost enaka 0, torej program preskoči na preverjanje naslednje tipke, če staro in novo stanje tipke 0 nista različni. Če sta vrednosti različni vstopi v del zanke, ki opravlja dejansko funkcijo zanke. Naloži spremenljivko mode, jo inkrementira za 1 in jo spremenjeno nazaj shrani, ter jo izvozi na naslov 1, ki je v VHDL kodi povezana s signalom “mode”, nato shrani novo stanje tipke na mesto starega stanja, s tem poskrbi, da se zanka ne more izvesti večkrat zapored pri istih pogojih, kar v tem primeru pomeni, da je stikalo občutljivo na fronto in se program ne bo izvajal, ce tipko držimo pritisnjeno. Po končani zamenjavi novega in starega stanja program preskoči na zanko, ki preverja naslednjo tipko na podoben način.

Pred preverjanjem svojega novega stanja, vsaka zanka ponastavi prejšnjo zanko, s tem, da ji zamenja stanji vhodne spremenljivke. Torej med izvajanjem zanke “loopbtn1” se zamenjata stanji “staromod” in “novomod”, saj se zanka “loopmod” izvaja pred zanko “loopbtn1”.

7. Podporna koda

Za projekt sem napisal program v jeziku Python, ki obdela kakrsnokoli sliko v obliko, ki jo procesor zna brati, v našem primeru je to .mif format, kjer so zaporedno zapisane svetlosti oziroma barve pikslov od leve proti desni, od zgoraj navzdol. Sliko z uporabo knjižnice openCV cv2 odprem v obliki numPy tabele, nato jo skaliram na velikost 64X64, saj uporabljam 12 - biten naslovni signal in vec kot 4096 vrednosti ne morem nasloviti. Sliko razbijem na slike, kjer vsaka predstavlja po eno komponento, R, G in B. Nato se program sprehodi skozi vse slike, rdečo, zeleno, modro in črno-belo. Za vsak piskel črno-bele slike izračuna 8 - bitno binarno vrednosti svetlosti od 0 do 255. To vrednost nato pripne na enodimenzionalni seznam binarnih svetlosti. Za barvne slike je proces nekoliko kompleksnejši, saj moramo 8 - bitno svetlost posamezne komponente predstaviti samo z 2 bitoma, torej z 256 možnih vrednosti pridemo na 4 možne vrednosti, to je 64 - kratna

barvna kompresija. zato vsako svetlost piskla delimo s 64, preden jo predstavimo v binarni obliki.
Binarne oblike vsake komponente nato program “zlepi” skupaj in jih kot 6 - bitno binarno število pripne na seznam binarnih barv. Program nato ustvari novo datoteko formata .mif in ji doda glavo oziroma header, kjer je specificirano v kakšnem formatu so zapisani podatki in njihovi naslovi, podatkovna širina in količina podatkov. Nato se program sprehodi skozi seznama svetlosti in barv, ter vsako, skupaj s svojo zaporedno številko / naslovom zapiše v svojo dotično datoteko. Na koncu se datotekama doda vrstica “END”, ki signalizira konec datoteke. Datoteko nato v programu Quartus z uporabo orodja “In - system memory content editor” zapišemo v spominski blok “fotka” oziroma “barve”.

```
for vrstica in range(64):
    for stolpec in range(64):

        neored = (int(rdeca[vrstica,stolpec]))//64
        red = str(np.binary_repr(neored,2))
        rezultat_r[vrstica,stolpec] = neored

        neogreen = (int(zelena[vrstica,stolpec]))//64
        green = str(np.binary_repr(neogreen,2))
        rezultat_g[vrstica,stolpec] = neogreen

        neoblue = (int(modra[vrstica,stolpec]))//64
        blue = str(np.binary_repr(neoblue,2))
        rezultat_b[vrstica,stolpec] = neoblue



        barve.append(red+green+blue)
        svetlosti.append(np.binary_repr(crnobela[vrstica,stolpec],8))
rezultat = cv2.merge([rezultat_r,rezultat_g,rezultat_b])
```

Kompresija barve in zapis svetlosti

8. Rezultati sinteze

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Memory Bits
1	▾ |sistem	1212 (21)	523 (21)	61440
1	|VGAvmesnik:U1|	92 (92)	75 (75)	0
2	▸ |barve:U6|	61 (0)	39 (0)	24576
3	|dsp:U7|	73 (73)	0 (0)	0
4	▸ |fotka:U5|	63 (0)	41 (0)	32768
5	|grafika:U3|	237 (237)	32 (32)	0
6	|io:U4|	30 (30)	25 (25)	0
7	▸ |proc:U2|	214 (1)	98 (0)	3072
8	▸ |sld_hub:auto_hub|	206 (1)	130 (0)	0
9	▸ |znaki:U8|	215 (215)	62 (62)	1024

Poraba virov

9. Rezultati simulacije