



Projekt

Uklanjanje šuma medijan filtrom



1. Sadržaj

2. Uvod	1
2.1. Projektni zadatak	1
3. Medijan filter	2
4. Postavljanje računalnog sustava	4
4.1. Fpga	5
5. Aplikacija	8
5.1. Grafičko sučelje	8
5.2. Funkcionalnost aplikacije	9
6. Rad projekta	18
7. Zaključak	21
8. Literatura	22



2. Uvod

Projektni zadatak je dizajnirati računalni sustav zasnovan na MicroBlaze procesoru koji uklanja šum pomoću medijan filtra. Sustav je dizajniran u Xilinx Platform Studiu (XPS) , dok je popratno grafičko sučelje realizirano pomoću aplikacije pisane u C# programskom jeziku.

Korisnik pomoću grafičkog sučelja komunicira sa sustavom, gdje izabire sliku i šalje je sustavu. Sustav sliku, nakon obrade medijan filtrom, šalje natrag u aplikaciju gdje se prikazuje obrađena slika.

2.1. Projektni zadatak

U ovom projektnom zadatku potrebno je dizajnirati računalni sustav zasnovan na MicroBlaze procesoru te napraviti grafičko sučelje korištenjem proizvoljnog programskog jezika (pr. C#, C/C++, java). Grafičko sučelje treba imati mogućnost učitavanja slike koja se nakon toga šalje MicroBlaze računalnom sustavu putem UART-a. Računalni sustav treba napraviti uklanjanje šuma na slici korištenjem medijan filtra. Veličina medijan filtra postavlja se u grafičkom sučelju. Nakon uklanjanja šuma, računalni sustav treba sliku poslati natrag u grafičko sučelje, gdje se prikazuje rezultat.




3. Medijan filter

Medijan filter je nelinearna digitalna tehnika filtriranja koja se često koristi za uklanjanje šuma sa slike jer, pod određenim uvjetima, čuva korisne detalje slike dok uklanja šum. Osobito je djelotvoran u uklanjanju šuma vrste „salt and pepper“.

Filtar radi na način da se odabere blok (prozor) piksela određene veličine. Blok prolazi kroz cijelu sliku. Unutar bloka računa se vrijednost svakog piksela te se vrijednosti piksela razvrstaju po veličini. Nakon razvrstavanja, odabire se medijan piksela koji će zamijeniti središnji piksel unutar bloka. Na slici 2.1 nalazi se primjer medijan filtriranja. U tablici s lijeve strane nalaze se nerazvrstane vrijednosti piksela u bloku 3x3. Vrijednosti se razvrstavaju po veličini (0, 2, 3, 3, 4, 6, 10, 19, 97) kako bi se mogao odrediti medijan. Nakon što se odredi medijan (4), zamjenjuje se središnji piksel s medijan vrijednošću što je prikazano u tablici s desne strane.

6	2	0
3	97	4
19	3	10



*	*	*
*	4	*
*	*	*

Slika 1: Primjer medijan filtriranja

Prednosti medijan filtriranja su te što na filter ne utječu pikseli čija se vrijednost znatno razlikuje od vrijednosti ostalih piksela unutar bloka te što čuva rubove slike, dok je najveći nedostatak brzina filtriranja i što ne prepoznaje detalje slike.



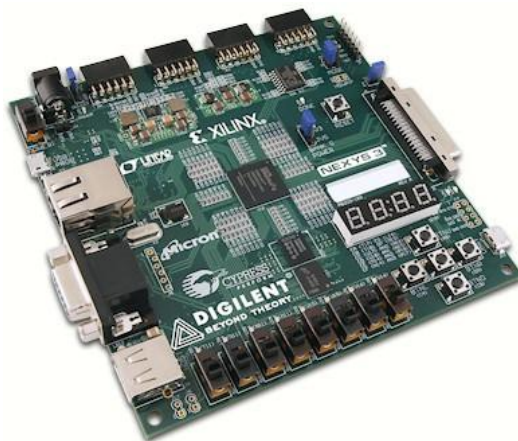
Slika 2: Primjer uklanjanja šuma „salt and paper“ pomoću medijan filtra

Medijan filter se koristi u sistemima potrošačke elektronike kada se želi prikazati slika vrhunske kvalitete ili u obradi rezultata različitih skenera i senzora kod kojih ponovljeno snimanje podrazumijeva veliku cijenu ili je čak neizvodljivo, npr. medicina, vojna industrija itd. Još jedna od mogućih primjena ove grane je i prilikom restauracije starih umjetničkih djela, kada je potrebno ukloniti oštećenja nastala s vremenom. Osnovni zadatak tehnike za otklanjanje šuma je uklanjanje nepoželjnog dijela signala slike ili videa uz očuvanje ostalih detalja.



4. Postavljanje računalnog sustava

Za izradu projekta korišten je Nexys3 razvojni sustav prikazan na slici 3, zasnovan na MicroBlaze procesoru. Ovaj sustav se sastoji od FPGA fizičkog sklopa te programskog paketa Xilinx Platform Studio koji sadrži Embedded Development Kit (EDK) i Software Development Kit (SDK).



Slika 3: Nexys3 razvojni sustav

U EDK odabiru se i dodaju potrebni periferni sklopovi računalnog sustava. Ovaj računalni sustav zahtjeva slijedeće komponente:

1. MicroBlaze procesor
2. MDM (eng. MicroBlaze Debug Module)
3. UART (serijska komunikacija)
4. BRAM (eng. Block RAM)
5. LMB (eng. Local Memory Bus) kontroleri za BRAM
6. GPIO (eng. General Purpose Input Output)
7. Micron RAM

Nexys 3 razvojni sustav omogućuje maksimalno korištenje 64 KB Block RAM memorije. Zbog potrebe za privremenom pohranom slike mora se proširiti memorija korištenjem Micron RAM-a čija je maksimalna veličina 16 MB (koliko je i upotrijebljeno). Postupak u EDK završava stvaranjem .bit datoteke i otvaranjem SDK. U SDK je napisan program u C programskom jeziku i njime je programiran FPGA.



Da bi FPGA primio sliku potrebno je inicijalizirati Micron RAM i UART komponente te omogućiti primanje podataka. Funkcija „XUartLite_RecvByte“ prima podatke jedan po jedan bajt, a „XUartLite_SendByte“ ih šalje na isti način. Za interakciju sa Micron RAM-om upotrijebljene su funkcije „Xil_Out32LE“ i „Xil_In32LE“.

4.1. Fpga

Periferni sklopovi MicronRAM i UART, kako bi se ispravno koristili i kako bi lakše mogli upravljati memorijom, moraju se postaviti na njihove osnovne adrese. Funkcija „XUartLite_RecvByte“ služi za primanje podataka koje aplikacija šalje bajt po bajt. Prvo se primaju podaci o dimenziji slike (stupci i redovi), a zatim i sama slika. Podaci o veličini slike se spremaju u Block RAM jer nisu veliki i zbog jednostavnog dohvaćanja. Podaci dolaze u ASCII kodu te se moraju pretvoriti u bročane vrijednosti kako bi ih mogli upotrijebiti. Pretvorba se izvršava na način da se podacima oduzme broj 48 (ASCII: '0') i zatim se zbroje znamenke.

Linija:

Kod:

```
55      adresa = XPAR_MICRON_RAM_MEMO_BASEADDR;
56      u32 addUART= XPAR_UARTLITE_1_BASEADDR;

63      data1 = XUartLite_RecvByte(addUART);
64      data2 = XUartLite_RecvByte(addUART);
65      data3 = XUartLite_RecvByte(addUART);
66      columns=(data1-'0')*100+(data2-'0')*10+(data3-'0');
67      data1 = XUartLite_RecvByte(addUART);
68      data2 = XUartLite_RecvByte(addUART);
69      data3 = XUartLite_RecvByte(addUART);
70      rows=(data1-'0')*100+(data2-'0')*10+(data3-'0');
```

Programski kod 1. Spremanje podataka

Zastavicu za analizu stanja programa postavljamo na nulu. Micron RAM se ponovno postavlja na osnovnu adresu kako bi izbjegli moguće pogreške kod pristupanja memoriji i kako bi znali s koje adrese krećemo. U varijablu „data“ pohranjujemo memorijsku adresu i varijablu u kojoj se nalazi podaci, to radimo iz razloga što funkcija „Xil_Out32LE“ to zahtjeva. Postupak primanja slike se u cijelosti nalazi unutar „for“ petlje gdje se nakon svakog primljenog podatka adresa pomiče za četiri mjesta kako bi podaci bili odvojeni i uzastopno zapisani. Zastavica se postavlja na jedinicu i adresa Micron RAM memorije vraća se na osnovnu ako je slika primljena i pohranjena u cijelosti.

**Linija:****Kod:**

```
73      i=0;
74      flag=0;
75      adresa=XPAR_MICRON_RAM_MEM0_BASEADDR;
76      adresa2=XPAR_MICRON_RAM_MEM0_BASEADDR+(rows*columns*4)+4;
77
78      for(i=0;i<rows*columns;i++){
79          data = XUartLite_RecvByte(addUART);
80          Xil_Out32LE(adresa, data);
81          Xil_Out32LE(adresa2, data);
82          adresa+=4;
83
84          adresa2+=4;
85      }
86
87      if(i==rows*columns) flag=1;
88      adresa = XPAR_MICRON_RAM_MEM0_BASEADDR;
89      adresa2=XPAR_MICRON_RAM_MEM0_BASEADDR+(rows*columns*4)+4;
```

Programski kod 2. Spremanje slike

Kada se zastavica nalazi u jedinici, odvija se medijan filtriranje. Prozor veličine 3x3 kreće se kroz originalnu sliku. Pomoću funkcije „Xil_In32LE“ čitamo vrijednosti piksela unutar prozora te ih u sljedećem koraku razvrstavamo po veličini. Izračunati medijan se sprema u varijablu „adresa2“ gdje će biti spremljeni i ostali rezultati filtriranja.

Linija:**Kod:**

```
93      for ( m = 1; m < rows - 1; ++m)
94          for ( n = 1; n < columns - 1; ++n)
95              {
96                  int k = 0;
97                  int window[9];
98                  for ( j = m - 1; j < m + 2; ++j)
99                      for ( i = n - 1; i < n + 2; ++i)
100                          window[k++] = Xil_In32LE((adresa+(4*(j*columns+i))));
101
102                  for ( j = 0; j < 5; ++j)
103                      {
104                          int min = j;
105                          for ( l = j + 1; l < 9; ++l)
106                              if (window[l] < window[min])
```




```
107             min = l;
108             const int temp = window[j];
109             window[j] = window[min];
110             window[min] = temp;
111         }
112
113         Xil_Out32LE((adresa2+(4*(m * columns + n))), window[4])
114     };
```

Programski kod 3. Medijan filtriranje

Ukoliko je slika u cijelosti obrađena, zastavica se postavlja u dvojku. Slanje se odvija unutar „for“ petlje. Pomoću funkcije „Xil_In32LE“ čitamo podatke iz RAM-a. Funkciji je potrebna samo adresa kao parametar i pohranjujemo ih u varijablu „data11“. Funkcijom „XUartLite_SendByte“ šaljemo podatke aplikaciji na način da joj predamo adresu UART-a i varijablu sa slikom. Adresu pomičemo za četiri mjesta sa svakim poslanim podatkom, kako bi podatke poslali jedan po jedan točnim redoslijedom.

Linija:

Kod:

```
123         if(x>rows-3 && y>columns-3) flag=2;
124         adresa = XPAR_MICRON_RAM_MEM0_BASEADDR;
125         adresa2=XPAR_MICRON_RAM_MEM0_BASEADDR+(rows*columns*4)+4;
126
127         if(flag==2){
128             for(i=0;i<rows*columns;i++){
129                 data11 = Xil_In32LE(adresa);
130                 XUartLite_SendByte(addUART,data11);
131                 adresa+=4;
132             }
```

Programski kod 4. Slanje slike

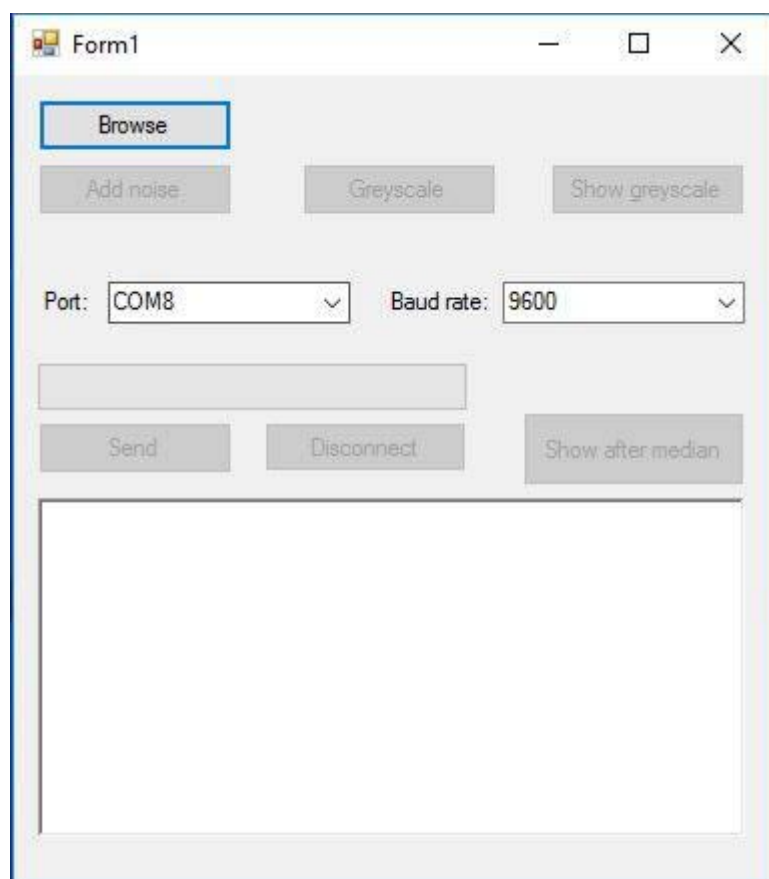


5. Aplikacija

Prilikom razvoja aplikacije korišteno je razvojno okruženje Microsoft Visual Studio i C# programski jezik. C# programski jezik je korišten radi jednostavnosti pravljenja formi i povezivanja aplikacije sa uređajem zbog mnogobrojnih ugrađenih funkcija. Aplikacija je ostvarena na principu da se učitava slika koju želimo filtrirati medijom filtrom. Sljedeće je da se slika pretvori u crno bijelu sliku te se ista pošalje na fpga te nakon završetka filtriranja aplikacija primi sliku. Slika koja se šalje nalazi se u bitmap formatu. Bitmap format pogodan je za slike manjih dimenzija jer pohranjuje sliku u izvornom obliku bez kompresija i pretvorbi zbog čega je takve slike jednostavnije obraditi i izvući iz njih potrebne podatke.

5.1. Grafičko sučelje

Pokretanjem aplikacije otvara se sljedeći prozor:



Slika 3: Grafički prikaz aplikacije

Na slici 3 možemo vidjeti od čega se grafički dio aplikacije sastoji. Vidimo da je izgledom jednostavna te da se sastoji od 7 gumbova (eng. Button), 2 kombinirana okvira (eng. ComboBox), 1 traka učitavanja (eng. Progress Bar) i 1 okvira za tekst (eng. RichTextBox).



Za upravljanje aplikacijom koriste se:

- „Browse“ gumb pomoću kojeg se pretražuje i učitava slika s računala
- „Port“ i „Baud rate“ kombinirani okviri pomoću kojih odabiremo aktivni kanal i biramo brzinu slanja i primanja podataka
- „Greyscale“ gumb koji pretvara originalnu sliku u crno-bijelu
- „Send“ gumb pomoću kojeg se otvara serijska komunikacija i šalje crno-bijela slika
- „Disconnect“ gumb koji zatvara serijsku komunikaciju

Za prikaz rada aplikacije koriste se:

- „Add noise“, „Show greyscale“ i „Show after median“ gumbovi pomoću kojih redom dodajemo šum na učitano izvornu sliku, konstruiramo crno-bijelu sliku i prikazujemo primljenu sliku nakon obrade.
- Okvir za tekst na koji se ispisuje ono što se događa u aplikaciji.
- Traka napretka koja vizualno prikazuje napredak pri slanju slike

5.2. Funkcionalnost aplikacije

Prilikom pokretanja aplikacije otvara se nova forma u kojoj je gumb „Browse“ jedini omogućen dok su ostali onemogućeni te kako se ide kroz aplikaciju će se omogućiti. Pomoću kombiniranog okvira „Port“ moguće je odabrati kanal kojim će se podaci prenositi i pomoću kombiniranog okvira „Baud rate“ jednu od normiranih brzina prijenosa podataka.

Linija	Kod
29	<code>public Form1 ()</code>
30	<code>{</code>
31	<code> InitializeComponent();</code>
32	<code> var ports = SerialPort.GetPortNames();</code>
33	<code> if (ports.Any())</code>
34	<code> {</code>
35	<code> comboBox1.DataSource = ports;</code>
36	<code> comboBox2.DataSource =</code>
	<code>EnumBaudRates.SupportedBaudRates;</code>
37	<code> }</code>
38	<code>}</code>
39	<code>private void Form1_Load(object sender, EventArgs e)</code>
40	<code>{</code>
41	<code> serialPort2.DataReceived += new</code>
	<code>SerialDataReceivedEventHandler(DataReceivedHandler);</code>
42	<code> serialPort2.DataBits = 8;</code>
43	<code> serialPort2.StopBits = StopBits.One;</code>
44	<code> serialPort2.Parity = Parity.None;</code>



```
45         serialPort2.DtrEnable = true;
46     }
```

Programski kod 1

Radi lakšeg snalaženja u programskom kodu na početku smo učitali datoteku „helpers“ u koju smo smjestili gotove funkcije. Programski kod 2. je funkcija za prikaz mogućih brzina prijenosa podataka koju smo koristili u programskom kodu 1.

Linija	Kod
9	public static class EnumBaudRates
10	{
11	public static readonly List<string>
	SupportedBaudRates = new List<string>
12	{
13	"9600",
14	"14400",
15	"19200",
16	"38400",
17	"57600",
18	"115200",
19	};
20	}

Programski kod 2. Funkcija „EnumBaudRates“

Gumb „Browse“ je jedini aktivan od učitavanja forme i pomoću njega je moguće učitati sliku za slanje u raznim oblicima. Nakon što je gumb pritisnut, otvara se novi prozor pomoću kojeg je potrebno učitati sliku (programski kod 3). Nakon što je uspješno učitana slika, gumb „Add noise“ i gumb „Greyscale“ su postali omogućeni. Pritiskom na gumb „Add noise“ dodajemo šum na učitano izvornu sliku (programski kod 4). Pritiskom na gumb „Greyscale“ učitano sliku pretvaramo u crno-bijelu sliku. U okvir za tekst ispisat će se „Grayscale finished!“ kada pretvorba bude završena, te će gumb „Show grayscale“ i gumb „Send“ postati omogućeni (programski kod 5).

Linija	Kod
61	private void btn_browse_Click(object sender, EventArgs e)
62	{
63	using (var dialog = new OpenFileDialog())
64	{
65	dialog.Filter = "Image files (*.jpg, *.jpeg,
	*.bmp, *.png) *.jpg; *.jpeg; *.bmp; *.png";
66	dialog.Title = "Select an image";
67	if (dialog.ShowDialog() == DialogResult.OK)
68	{
69	originalImg = new
	Bitmap(dialog.FileName);
70	}



```
71         if (originalImg != null)
72         {
73             showImage(originalImg);
74         }
75     }
76     btn_showoriginal.Enabled = true;
77     btn_greyscale.Enabled = true;
78 }
```

Programski kod 3. Gumb „Browse“

Pritiskom na gumb „Add noise“ pokreće se funkcija „GenerateNoise“ koja služi generiranju šuma na slici. Kao argumente prima sliku u „Bitmap“ formatu i vjerojatnost da će se piksel pretvoriti u šum.

Linija	Kod
96	<code>private Bitmap GenerateNoise(Bitmap image, double prob)</code>
97	<code>{</code>
98	<code> double tresh = 1 - prob;</code>
99	<code> Bitmap finalBmp = image;</code>
100	<code> Random r = new Random();</code>
101	<code> for (int x = 0; x < image.Width; x++)</code>
102	<code> {</code>
103	<code> for (int y = 0; y < image.Height; y++)</code>
104	<code> {</code>
105	<code> double num = r.NextDouble();</code>
106	<code> if (num < prob)</code>
107	<code> {</code>
108	<code> finalBmp.SetPixel(x, y,</code>
	<code>Color.FromArgb(255, 0, 0, 0));</code>
109	<code> }</code>
110	<code> else if (num > tresh){</code>
111	<code> finalBmp.SetPixel(x, y,</code>
	<code>Color.FromArgb(255, 255, 255, 255));</code>
112	<code> }</code>
113	<code> else continue;</code>
114	<code> }</code>
115	<code> }</code>
116	<code> return finalBmp;</code>
117	<code>}</code>

Programski kod 4. Gumb „Add noise“

Linija	Kod
129	<code>private void btn_greyscale_Click(object sender, EventArgs e)</code>
130	<code>{</code>
131	<code> if (originalImg != null)</code>
132	<code> {</code>
133	<code> grayImg = originalImg.MakeGrayScale();</code>



```
134         richTextBox1.AppendText("Grayscale finished!\n");
135         grayImg.Save(ms, ImageFormat.Bmp);
136     }
137     btn_showgreyscale.Enabled = true;
138     btn_send.Enabled = true;
139 }
```

Programski kod 5. Gumb „Greyscale“

U programskom kodu 3 koristimo funkciju „showImage“ koju smo definirali na početku aplikacije te ju koristimo kroz cijelu aplikaciju za prikazivanje slika u novom prozoru (programski kod 6). U programskom kodu 5 koristimo funkciju „MakeGrayScale“ koju smo smjestili u datoteku „helpers“. Funkcija prvo stvori praznu bitmap sliku iste veličine kao izvorna slika te stvara novi „graphics“ objekt. Potrebno je originalnu sliku precrtati preko novostvorenog bitmap-a koristeći grayscale matricu. Grayscale matrica boja se inicijalizira i stvaraju se njeni atributi kako bi se isti mogli primijeniti prilikom precrtavanja slike (programski kod 7).

<i>Linija</i>	<i>Kod</i>
80	<code>private void showImage(Image img)</code>
81	<code>{</code>
82	<code> Form fm = new Form();</code>
83	<code> PictureBox pb = new PictureBox();</code>
84	<code> pb.Image = img;</code>
85	<code> pb.Size = new Size(img.Width + 300, img.Height +</code>
	<code>300);</code>
86	
87	<code> pb.SizeMode = PictureBoxSizeMode.StretchImage;</code>
88	<code> fm.Controls.Add(pb);</code>
89	<code> fm.Size = new Size(img.Width + 300, img.Height +</code>
	<code>300);</code>
90	
91	<code> fm.Show();</code>
92	<code>}</code>

Programski kod 6. Gumb „Show Image“

<i>Linija</i>	<i>Kod</i>
11	<code>public static class ImageHelper</code>
12	<code>{</code>
13	<code> public static Bitmap MakeGrayScale(this Image image)</code>
14	<code>{</code>
15	<code> Bitmap newBitmap = new Bitmap(image.Width,</code>
	<code>image.Height);</code>
16	<code> Graphics g = Graphics.FromImage(newBitmap);</code>
17	<code> ColorMatrix colorMatrix = new ColorMatrix(</code>
18	<code> new float[][]</code>
19	<code>{</code>



```
20         new float[] { .3f, .3f, .3f, 0, 0 },
21         new float[] { .59f, .59f, .59f, 0, 0 },
22         new float[] { .11f, .11f, .11f, 0, 0 },
23         new float[] { 0, 0, 0, 1, 0 },
24         new float[] { 0, 0, 0, 0, 1 }
25     });
26     ImageAttributes attributes = new
ImageAttributes();
27     attributes.SetColorMatrix(colorMatrix);
28     g.DrawImage(image, new Rectangle(0, 0,
image.Width, image.Height),
29         0, 0, image.Width, image.Height,
GraphicsUnit.Pixel, attributes);
30     g.Dispose();
31     return new Bitmap;
32 }
33 }
```

Programski kod 7. Funkcija „MakeGrayScale“

Pritiskom na gumb „Show greyscale“ prikazujemo crno-bijelu sliku dobivenu nakon greyscale-a (programski kod 8). Za prikaz slike koristimo gotovu funkciju koju smo već opisali (programski kod 6).

<i>Linija</i>	<i>Kod</i>
11	<code>private void btn_showgreyscale_Click(object sender, EventArgs e)</code>
12	<code>{</code>
13	<code> if (grayImg != null)</code>
14	<code> {</code>
15	<code> showImage(grayImg);</code>
16	<code> }</code>
17	<code>}</code>

Programski kod 8. Gumb „Show greyscale“

Gumb „Send“ koristi se kako bi se učitana slika poslala na FPGA na daljnju obradu sa median filtrom. Pritiskom na gumb provjerava se je li serijski port otvoren. Prije slanja same slike šalju se prvo njene dimenzije, odnosno prvo broj stupaca pa broj redaka. Slika koja se šalje spremljena je u bitmap formatu te se šalje kao 1D polje bajt po bajt u „for“ petlji. Prvo moramo odvojiti bitmap zaglavlje od tijela slike. Tijelo slike spremamo u „byte“ polje koje zatim šaljemo bajt po bajt. Prilikom slanja slike traka napretka vizualno prikazuje koliko smo podataka slike poslali te koliko još ima do završetka slanja. Kada se slanje završi trenutna vrijednost trake napretka je maksimalna, u okviru za tekst ispisuje se poruka „Send finished!“ te se omogućuje gumb „Disconnect“(programski kod 9).

<i>Linija</i>	<i>Kod</i>
131	<code>private void btn_send_Click(object sender, EventArgs e)</code>
132	<code>{</code>



```
133         if (!serialPort2.IsOpen)
134         {
135             serialPort2.Open();
136             comboBox1.Enabled = false;
137             btn_disc.Enabled = true;
138             send_pointer = 0;
139             progressBar1.Step = 1;
140             progressBar1.Value = 0;
141             ms.Position = 0;
142             byte[] stupci =
Encoding.ASCII.GetBytes(grayImg.Width.ToString());
143             byte[] retci =
Encoding.ASCII.GetBytes(grayImg.Height.ToString());
144             byte[] imgConverted = ms.ToArray();
145             byte[] imgByteArray = new byte[grayImg.Width
* grayImg.Height * 4];
146             var x = 0;
147             var start = imgConverted[10];
148             for (int i = start; i < (imgConverted.Length
- start); i++)
149             {
150                 imgByteArray[x] = imgConverted[i];
151                 x++;
152             }
153             progressBar1.Maximum = grayImg.Width *
grayImg.Height * 4;
154             progressBar1.Value = 0;
155             serialPort2.Write(stupci, 0, 3);
156             serialPort2.Write(retci, 0, 3);
157             for (int i = 0; i < grayImg.Width *
grayImg.Height * 4; i += 4)
158             {
159                 serialPort2.Write(imgByteArray, i, 1);
160                 progressBar1.Value = i;
161                 send_pointer++;
162             }
163             progressBar1.Value = progressBar1.Maximum;
164             richTextBox1.AppendText("Send finished! \n");
165             btn_disc.Enabled = true;
166         }
167     }
```

Programski kod 9. Gumb „Send“

Funkcija „DataReceivedHandler“ (programski kod 10) se inicijalizira na početku programa u programskom kodu 1 te se pokreće svaki put kada računalo primi bilo kakve podatke preko komunikacijskog kanala. Stvara se novi komunikacijski kanal „sp“ na koji dolaze informacije. Varijabla „indata“ predstavlja podatke koje računalo prima preko kanala te se ti podaci nadodaju na varijablu „sb“ koja je tipa „StringBuilder“. Ako je veličina primljenih podataka jednaka veličini poslanih podataka, u okviru za tekst ispisuje se poruka „Receive finished!“ te pokreće se funkcija „ConvertReceived“ kojoj predajemo varijablu „sb“ pretvorenu u string.



Prilikom primanja podataka sa serijskog port-a koristimo ugrađenu „sp.ReadExisting()“ metodu. Problem kod nje je taj što ona forsira kodiranje primljenih bajtova u ASCII formatu. Pošto je ASCII format 7-bitni format, sa serijskog port-a možemo primiti brojeve najviše vrijednosti 127. Pošto mi moramo primiti brojeve od 0-255 to nam predstavlja problem. Jedan od rješenja je promijeniti vrstu formata kodiranja. U ovom slučaju koristimo „ISO-8859-1“ format.

Linija	Kod
170	<code>private void DataReceivedHandler(</code>
171	<code>object sender,</code>
172	<code>SerialDataReceivedEventArgs e)</code>
173	<code>{</code>
174	<code>SerialPort sp = (SerialPort)sender;</code>
175	<code>sp.Encoding = Encoding.GetEncoding(28591);</code>
176	<code>string indata = sp.ReadExisting();</code>
177	<code>sb.Append(indata);</code>
178	<code>if (sb.Length == send_pointer)</code>
179	<code>{</code>
180	<code>Invoke(new MethodInvoker(delegate ()</code>
181	<code>{</code>
182	<code>richTextBox1.AppendText("Receive</code>
	<code>finished! \n");</code>
183	<code>}});</code>
184	<code>ConvertReceived(sb.ToString());</code>
185	<code>}</code>
186	<code>}</code>

Programski kod 10. Funkcija „DataReceivedHandler“

Funkcija „ConvertReceived“ (programski kod 11) poziva se kada je slika uspješno primljena od FPGA. Njezin cilj je rekonstruirati novu sliku sa podacima dobivenim median filtrom. U „byte“ polje image kopiramo originalnu sliku koju smo spremili u „MemoryStream“ kako bi imali kostur slike. Zatim trebamo samo mijenjati RGB bajtove stare slike. U okviru za tekst ispisuje se poruka kada je pretvorba započela porukom „Starting conversion..“ i kada se završi porukom „Conversion finished!“. Kada se pretvorba završi gumb „Show after median“ je omogućen.

Linija	Kod
188	<code>private void ConvertReceived(string input)</code>
189	<code>{</code>
190	<code>Invoke(new MethodInvoker(delegate ()</code>
191	<code>{</code>
192	<code>richTextBox1.AppendText("Starting</code>
	<code>conversion.. \n");</code>
193	<code>}});</code>
194	<code>byte[] input_image = new byte[input.Length];</code>
195	<code>Encoding iso = Encoding.GetEncoding("ISO-8859-</code>
	<code>1");</code>
196	<code>input_image = iso.GetBytes(input);</code>



```
197         byte[] image = ms.ToArray();
198         int pocetak = image[10];
199         int velicina = grayImg.Height * grayImg.Width *
200             4;
201         int j = 0;
202         for (int i = pocetak; i < velicina; i += 4)
203         {
204             image[i] = input_image[j];
205             image[i + 1] = input_image[j];
206             image[i + 2] = input_image[j];
207             j++;
208         }
209         var ms_return = new MemoryStream(image);
210         receivedImg = Image.FromStream(ms_return);
211         sb.Clear();
212         Invoke(new MethodInvoker(delegate ()
213         {
214             richTextBox1.AppendText("Conversion finished!
215                                     \n"));
216             btn_showaftermedian.Enabled = true;
217         }));
218     }
219 }
```

Programski kod 11. Funkcija „ConvertReceived“

Pritiskom na gumb „Disconnect“ zatvaramo komunikacijski kanal ako je bio otvoren (programski kod 12).

Linija	Kod
218	<code>private void btn_disc_Click(object sender, EventArgs e)</code>
219	<code>{</code>
220	<code> if (serialPort2.IsOpen)</code>
221	<code> {</code>
222	<code> serialPort2.Close();</code>
223	<code> }</code>
224	<code>}</code>

Programski kod 12. Gumb „Disconnect“

Pritiskom na gumb „Show after median“ prikazujemo novu filtriranu sliku koja je primljena od strane FPGA (programski kod 13). Za prikaz slike koristimo gotovu funkciju koju smo već opisali (programski kod 6).

Linija	Kod
226	<code>private void btn_showaftermedian_Click(object sender,</code>
	<code>EventArgs e)</code>
227	<code>{</code>
228	<code> if (receivedImg != null)</code>
229	<code>{</code>



```
230             showImage(receivedImg);  
231         }  
232     }
```

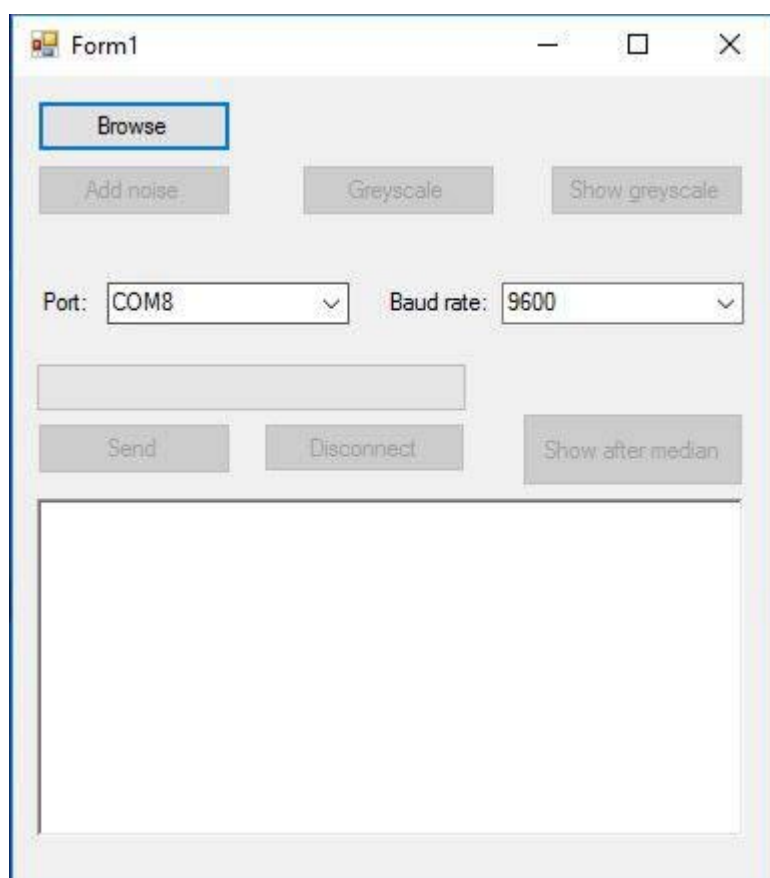
Programski kod 13. Gumb „Show after median“



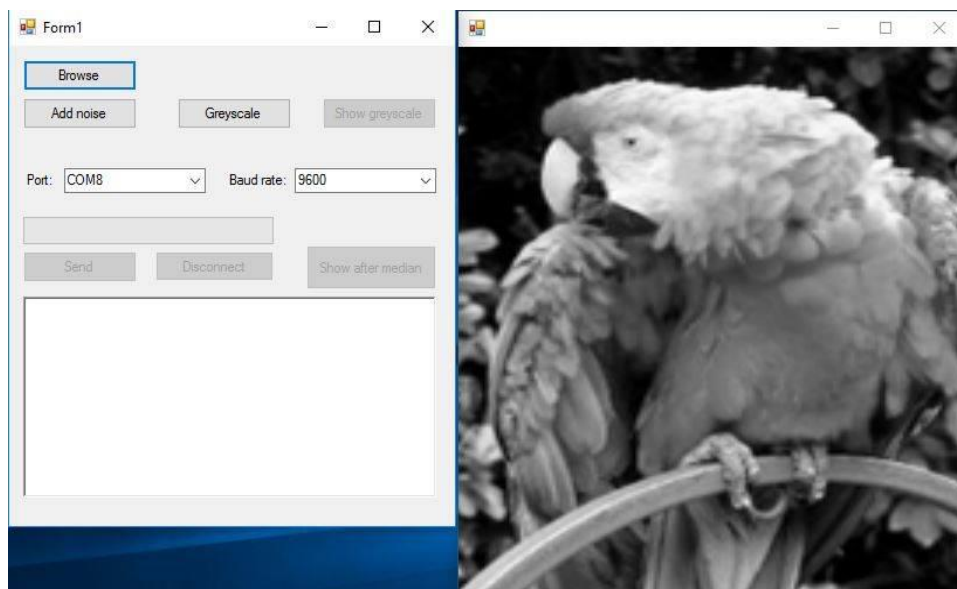
6. Rad projekta

Na narednim slikama ovoga odjeljka prikazana je uspješna realizacija projekta, te rad C# aplikacije. Slika 4 prikazuje samu C# aplikaciju koja je principijelni dio samog projekta. Prilikom otvaranja aplikacije klikom na gumb „Browse“ proizvoljno odabiremo sliku koju želimo obraditi. Dozvoljene dimenzije slike su 1000 piksela u širinu, te 1000 piksela u duljinu. Ako je učitana slika veća od predviđenih vrijednosti, obrađen će biti onaj dio koji je u odgovarajućem rasponu, dok će ostatak biti eliminiran. Nakon odabira slike, potrebno je odabrati „Greyscale“ u slučaju da je slika u boji.

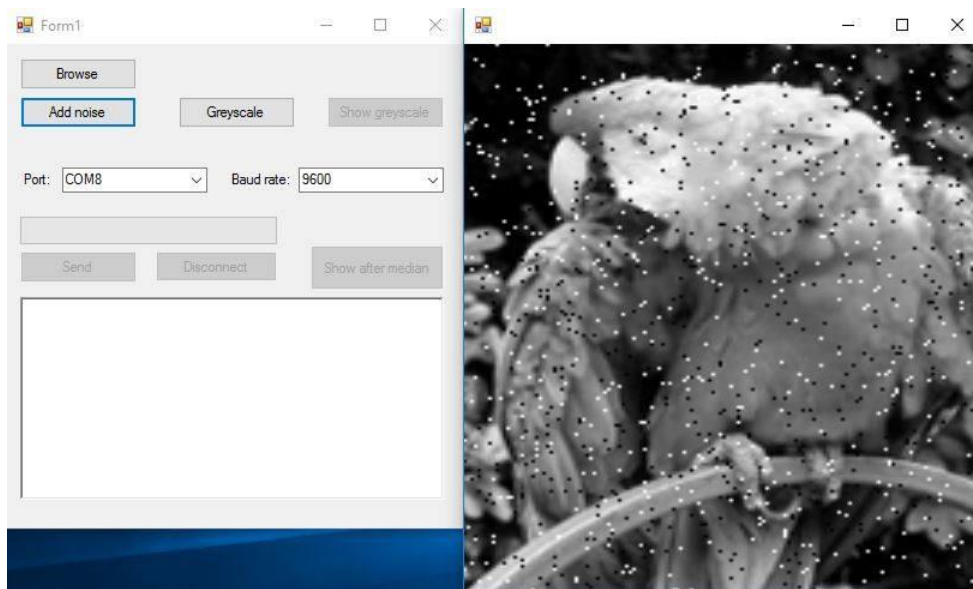
Nakon prethodnih koraka, potrebno je dodati šum slici što je prikazano slikom 6. „Oštećenu“ sliku potrebno je poslati tipkom „send“, te nakon obrade median filtrom kliknuti na gum „Show after median“ što je i prikazano na slici 7.



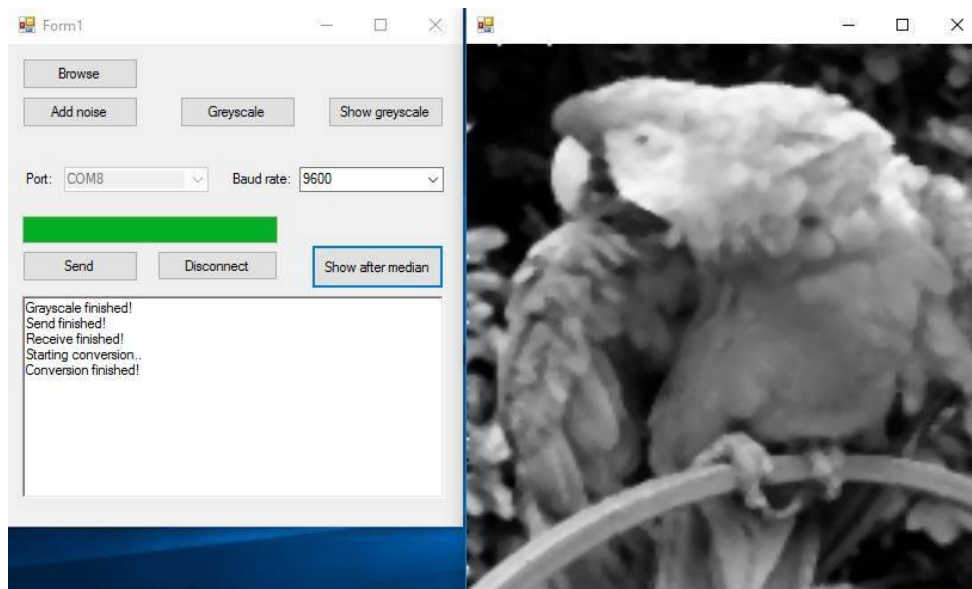
Slika 4: Grafički izgled aplikacije



Slika 5: Grafički izgled aplikacije



Slika 6: Grafički izgled aplikacije



Slika 7: Grafički izgled aplikacije



7. Zaključak

Projektni zadatak je uspješno napravljen, potrebna uspostava komunikacije između razvojnog okruženja i C# aplikacije je uspostavljena – slika je poslana i primljena te je pomoću median filtra uspješno obrađena. Aplikacija radi za slike veličine 100 do 1000 piksela u širini i dužini. Projektni zadatak podijeljen je na manje djelove. Svaki pojedini dio radio je kao zasebni program, no problemi su se pojavili pri njihovom povezivanju u cjelinu. Problem nije bio u samnom algoritmu nego u serijskoj komunikaciji između sustava.

Naime, prilikom primanja podataka sa serijskog port-a koristimo ugrađenu „sp.ReadExisting()“ metodu. Problem kod nje je taj što ona forsira kodiranje primljenih bajtova u ASCII formatu. Pošto je ASCII format 7-bitni format, sa serijskog port-a možemo primiti brojeve najviše vrijednosti 127. Pošto mi moramo primiti brojeve od 0-255 to nam predstavlja problem. Jedan od rješenja je promijeniti vrstu formata kodiranja. U ovom slučaju koristimo „ISO-8859-1“ format.

Nakon brojnih pokušaja, problemi u komunikaciji su uspješno riješeni i nakon što je prvi puta uspješno poslana i primljena slika, dovršavanje projekta i posljednje modifikacije koda su lakše obavljani.



8. Literatura

- [1] Digilent, »Nexys3 Board Reference Manual,« Digilent, Pullman, WA, 2013.
- [2] Xilinx, »Embedded System Tools Reference Manual - Embedded Development Kit,« Xilinx, 2008.
- [3] Xilinx, »MicroBlaze Processor Reference Guide - Embedded Development Kit EDK 10.1i,« Xilinx, 2008.
- [4] P. Marwedel, Embedded System Design - Embedded Systems Foundations of Cyber-Physical Systems, Springer Netherlands, 2011.
- [5] J. O. Hamblen, T. S. Hall i M. D. Furman, Rapid Prototyping of Digital Systems - SOPC Edition, Springer US, 2008.