

SEMINARSKI RAD

PREDMET: PROJEKTOVANJE VLSI SISTEMA

USER-MANUAL

UBRZAVANJE ZAKLJUČIVANJA KONVULACIONIH NEURONSKIH MREŽA I IMPLEMENTACIJA NA PYNQ- Z2 (FPGA)PLOČI

Predmetni Profesor:

Dr Aleksandar Peulić

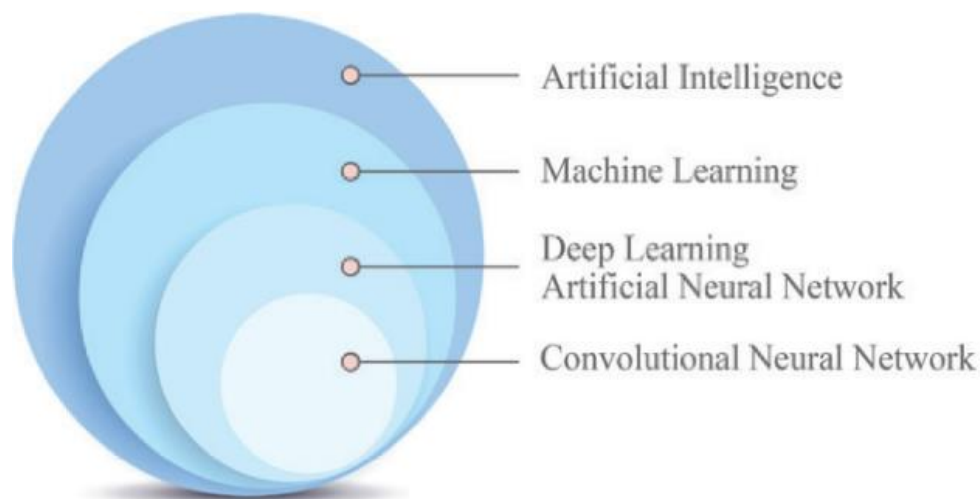
Student:

Luka Jevtić 64-2017

SADRŽAJ

Uvod	3
Pravljenje Projekta u Vivado-u	5
Kod u Vivado-u	7
Simulacija Accelerator-a.....	14
Sinteza i implementacija	16
Pokretanje Bitstream-a.....	17
Izgled PYNQ-Z2 nakon BITSTREAM-a	19

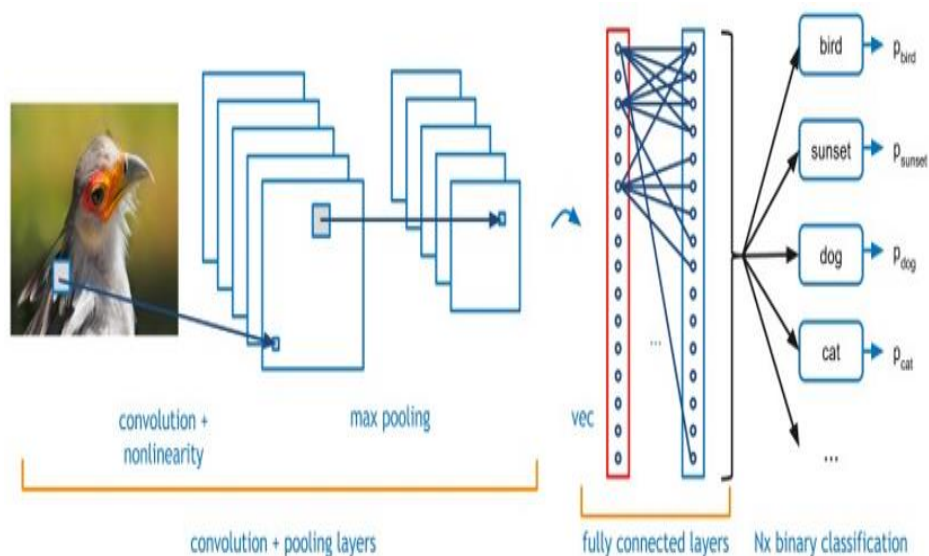
UVOD



Slika 1 – Podskupovi veštačke inteligencije

Duboko učenje, koje se uglavnom sastoji od aplikacija koje uključuju određene specijalizovane strukture zvane Neuralne mreže, dalo je najveći podsticaj čitavoj oblasti mašinskog učenja i istraživanja veštačke inteligencije. Neuralne mreže su pokazale ogromne sposobnosti u obavljanju zadataka koji su tradicionalno bili veoma teški za računare, do te mere da se pokazalo da neki pristupi dubokog učenja rade čak i bolje od ljudi u nekim zadacima kao što su vid i prepoznavanje govora.

Imajući u vidu ovaj ogroman potencijal, postojao je veliki interes za ubrzavanje zaključivanja neuronskih mreža, posebno od strane velikih internet kompanija sa alatima u rasponu od glasovnih asistenata do autonomne navigacije koja rukuje gomilom korisničkih podataka u realnom vremenu koji treba da prođe kroz ogromne neuronske mreže i taj rezultat treba da se servira korisniku u roku od nekoliko milisekundi.



Slika 2 – Prepoznavanje slike uz pomoć CNN

Na slici iznad možemo primetiti prepoznavanje slike ptice uz pomoć metoda **konvolucionih neuronskih mreža**, naime ovde je prikazan process ubrzanja takvog čitanja uz pomoć raznih slojeva.

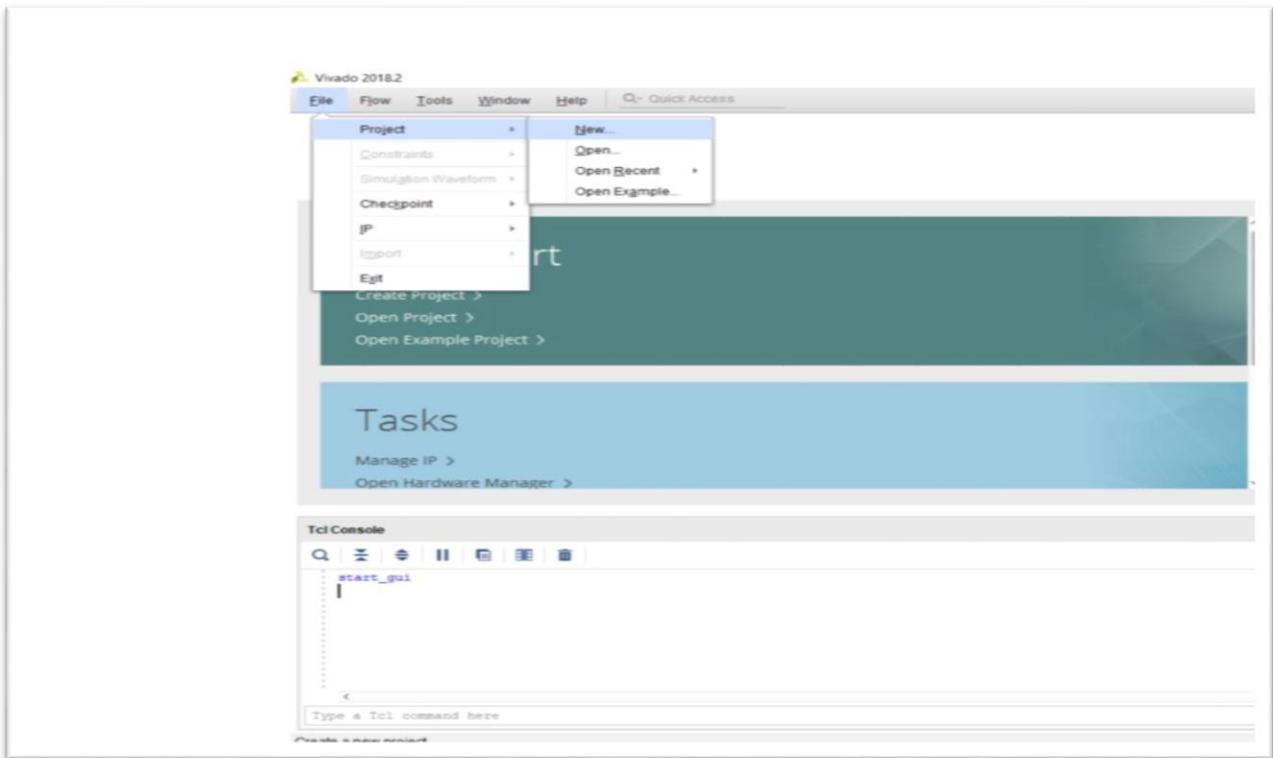
Na početku imamo sloj:

Konvolucije(Convolation) – gde se vrši savijanje ulazne slike sa skupom filtera, od kojih svaki filter daje jednu mapu karakteristika na izlaznoj slici.

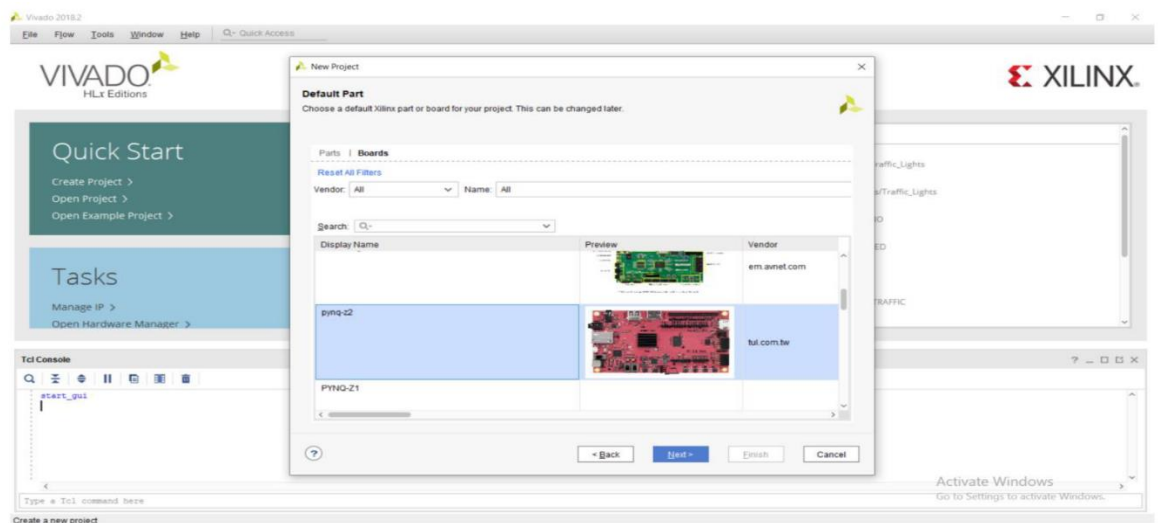
Objedinjavanje(Pooling) - je u suštini operacija 'spuštanja uzorkovanja' koja ima za cilj smanjenje broja parametara i složenosti kako se podaci šire u mreži. Ovaj proces uključuje pokretanje 'prozora za okupljanje' preko ulaza i smanjenje veličine ulaza pomoću nekog algoritma. Daleko najčešći algoritmi su **Max-Pooling** i **Average - Pooling**.

Funkcija aktivacije se uvodi u neuronskoj mreži sa namerom da se unese nelinearnost u celu mrežu. Bez toga, cela neuronska mreža bi se pojednostavila i ne bi mogla da pročita sliku. Koriste se različite funkcije za ovu svrhu, ali najčešće (i prve) korišćene bi bile funkcije **ReLu** (Ispravljачka linearna jedinica) i **Tanh** (Hiperbolična tangenta).

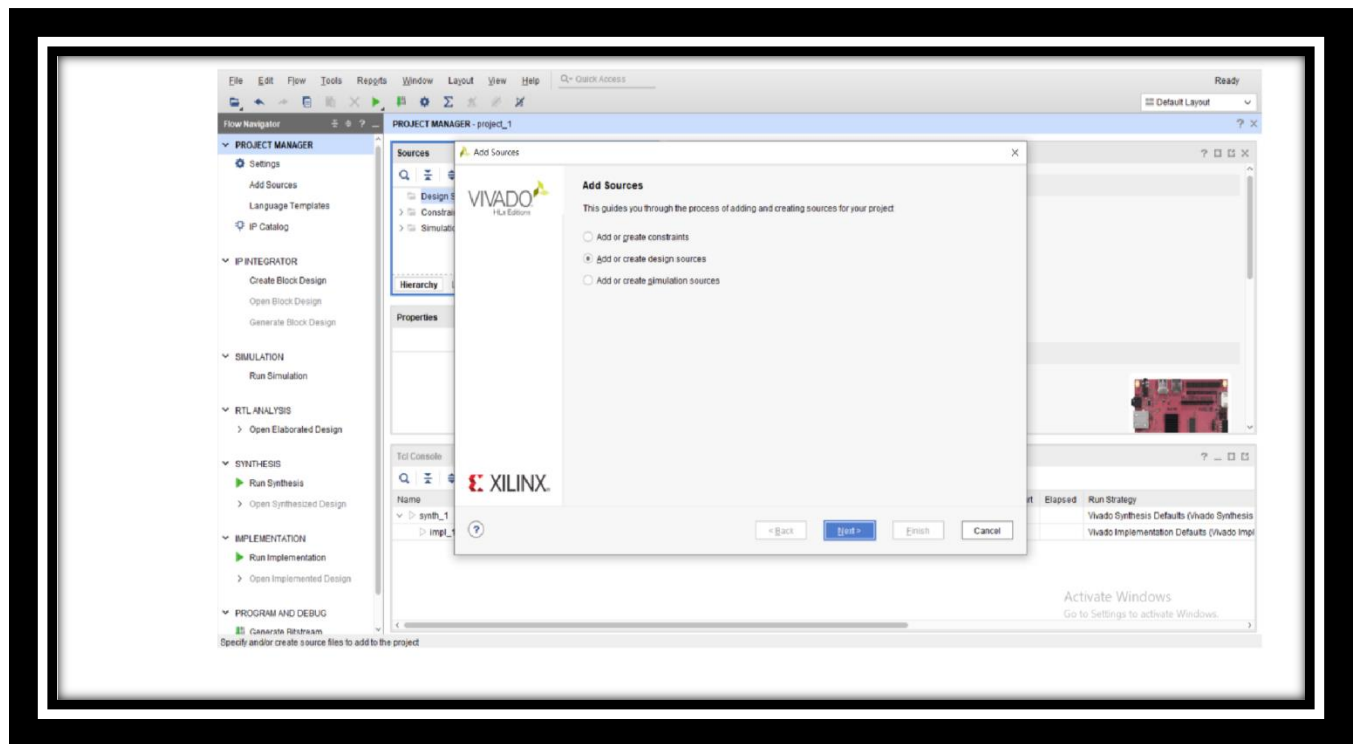
PRAVLJENJE PROJEKTA U VIVADO-U



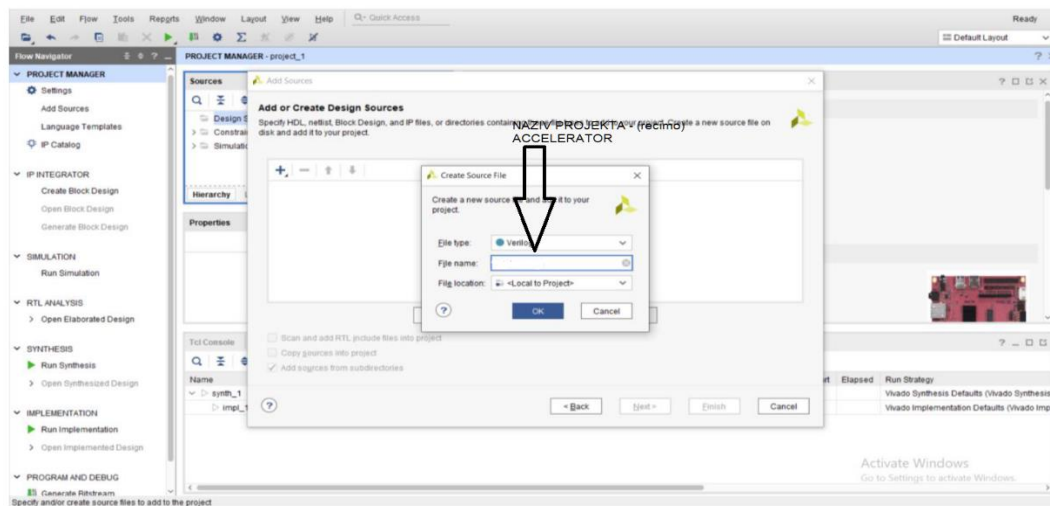
Slika 3 – Pravljenje projekta u Vivado(Project->New Project)



Slika 4 – Odabir razvojne ploče(PYNQ-Z2)



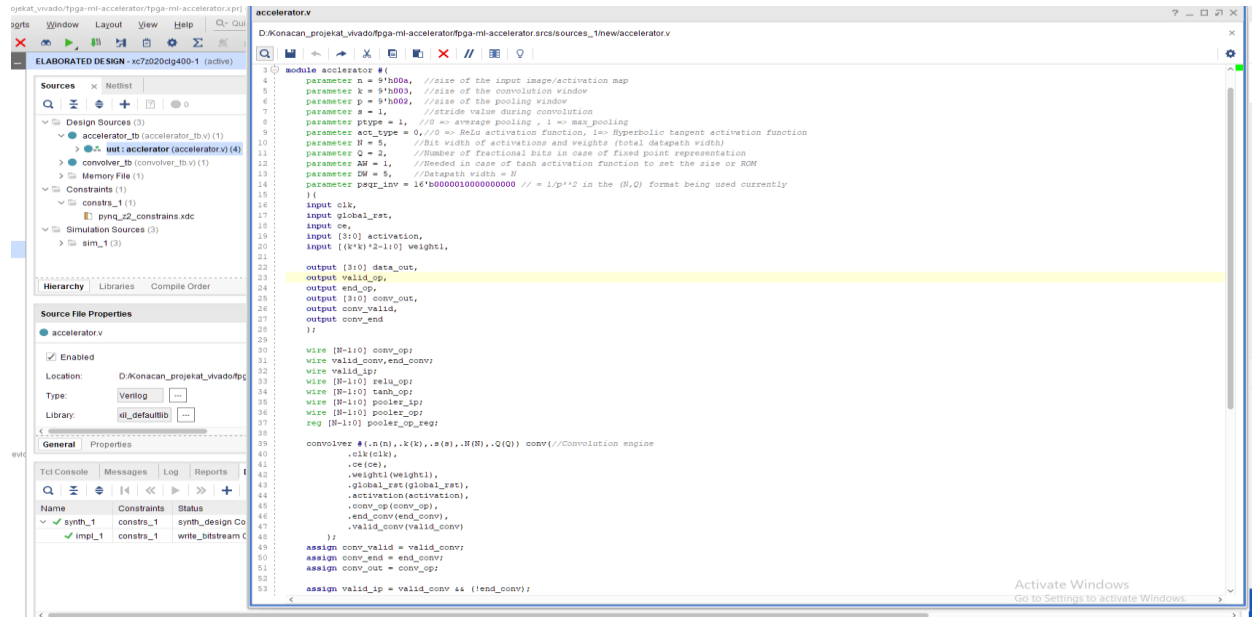
Slika 5 – Pravljenje .v fajla (Desni klik na Design Sources)



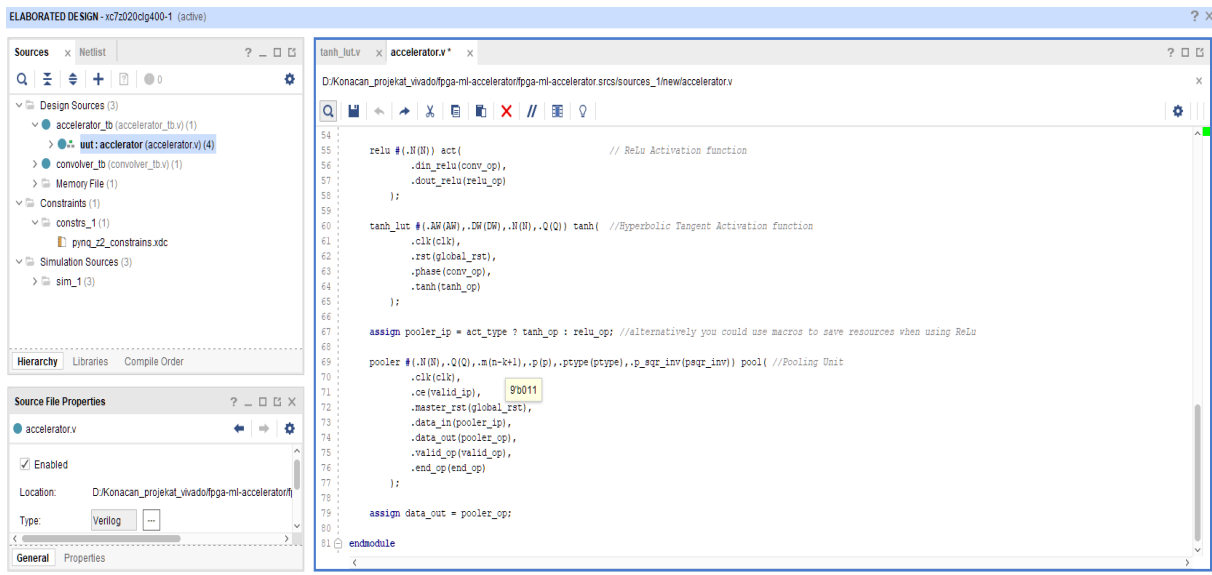
Slika 6 – Pravljenje .v fajla - postavljanje imena

KOD U VIVADO-U

Pre svega za realizovanje ovog koda sam koristio verziju Vivado 2018.2. U nastavku ću prikazati kod i u kasnijim poglavljima simulaciju...



Slika 7 – Kod u top modelu accelerator - part 1

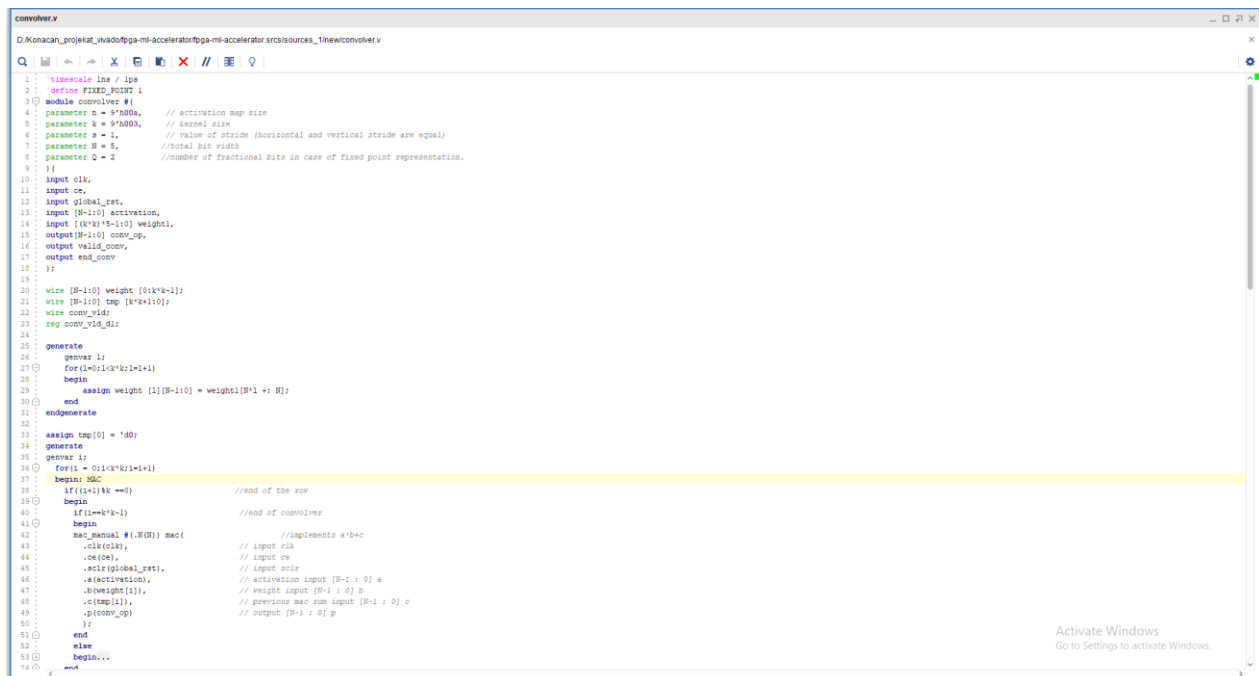


Slika 8 – Kod u top modelu accelerator – part 2

Ukratko objašnjenje koda u **accelerator.v** – To je top model koji u sebi sadrži još 4 podmodela (conv(convolver), relu, tanh, pooler), njegove ulazne vrednosti su clk(clock), global_rst(reset), ce, trobitnu reg promenljivu activation, reg promenljivu weight1(kojoj vrednost zavisi od k(konvolucioni prozori)) I Izlazne vrednosti data _out, valid_op, end_op, conv_out, conv_valid, conv_end.

Accelerator dalje prosledjuje vrednosti podmodelima funkcijama relu,tanh(**Funkcije Aktivacije**) kao I convolver-u(**Convolution**) I pooler-u(**Pooling**).

Kod koji se nalazi u podmodelu convolver:



```

1 timescale 1ns / 1ps
2 `define FIXED_POINT 1
3 module convolver #(
4     parameter N = 9'h000, // activation map size
5     parameter K = 9'h003, // kernel size
6     parameter S = 1, // value of stride (horizontal and vertical stride are equal)
7     parameter W = 5, // total bit width
8     parameter Q = 2 // number of fractional bits in case of fixed point representation.
9 )
10 input clk,
11 input ce,
12 input global_rst,
13 input [9:1:0] activation,
14 input [(K*S)-1:0] weight1,
15 output [9:1:0] conv_out,
16 output valid_conv,
17 output end_conv
18
19 wire [9:1:0] weight [(K*K)-1:0];
20 wire [9:1:0] tmp [(K*K)-1:0];
21 wire conv_valid;
22 reg conv_valid;
23
24 generate
25     generate_1:
26     for (i=0; i<K*K; i+=1)
27     begin
28         assign weight[i] = weight1[(K*K)-1-i];
29     end
30 endgenerate
31
32 assign tmp[0] = 'd0;
33 generate
34     generate_1:
35     for (i = 0; i<K*K; i+=1)
36     begin: MAC
37         if ((i+1)%K == 0) //end of the row
38         begin
39             if (i==K*K-1) //end of convolve
40             begin
41                 mac_macout1 #(.SDT) mac(
42                     .clk(clk), // implements a*b+c
43                     .ce(ce), // input ce
44                     .actr(global_rst), // input actr
45                     .a(activation), // activation input [9:1:0] a
46                     .b(weight[i]), // weight input [9:1:0] b
47                     .c(tmp[i]), // previous mac sum input [9:1:0] c
48                     .p(conv_out) // output [9:1:0] p
49                 );
50             end
51         end
52     else
53         begin:
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

Slika 9 – Kod u convolver-u – part 1


```

convolver.v
D:\Konacan_projekat_vivado\tpga-mi-accelerator\tpga-mi-accelerator.srcs\sources_1\twe\convolver.v

41 | begin
42 |   mac_manual #(1:B(0)) mac(
43 |     .clk(clk),           // input clk
44 |     .ce(ce),             // input ce
45 |     .sclr(global_rst),   // input sclr
46 |     .active_valid(),     // activation input [B-1 : 0] a
47 |     .b(weight[i]),       // weight input [B-1 : 0] b
48 |     .c(tmp[i]),          // previous mac sum input [B-1 : 0] c
49 |     .p(count_op)         // output [B-1 : 0] p
50 |   );
51 | end
52 | else
53 | begin
54 |   wire [B-1:0] tmp2;
55 |   //make a mac unit
56 |   mac_manual #(1:B(0)) mac(
57 |     .clk(clk),
58 |     .ce(ce),
59 |     .sclr(global_rst),
60 |     .active_valid(),
61 |     .b(weight[i]),
62 |     .c(tmp[i]),
63 |     .p(tmp2)
64 |   );
65 |
66 |   variable_shift_reg #(1:WIDTH(0),.SIZE(n-k)) SR (
67 |     .d(tmp2),           // input [10 : 0] d
68 |     .clk(clk),          // input clk
69 |     .ce(ce),            // input ce
70 |     .rst(global_rst),   // input sclr
71 |     .out(tmp[i+1])      // output [32 : 0] q
72 |   );
73 | end
74 | end
75 | else
76 | begin
77 |   mac_manual #(1:B(0),.Q(2)) mac2(
78 |     .clk(clk),
79 |     .ce(ce),
80 |     .sclr(global_rst),
81 |     .active_valid(),
82 |     .b(weight[i]),
83 |     .c(tmp[i]),
84 |     .p(tmp[i+1])
85 |   );
86 | end
87 | end
88 | endgenerate
89 |
90 |
91 | reg [31:0] count,count2,count3,row_count;
92 | reg en1,en2,en3;
93 |
94 | always @(posedge clk)
95 | begin
96 |   if(global_rst)
97 |   begin
98 |     count <= 0;           //master counter: counts the clock cycles
99 |     count2 <= 0;          //counts the valid convolution outputs
100 |     count3 <= 0;          // counts the number of invalid convolutions where the kernel wraps around the next row of inputs.
101 |     row_count <= 0;       //counts the number of rows of the output.
102 |     en1 <= 0;
103 |     en2 <= 1'b1;
104 |     en3 <= 0;
105 |   end
106 |   else if(ce)
107 |   begin
108 |     if(en1 && en2)
109 |     begin
110 |       if(count2 == n-k)
111 |       begin
112 |         count2 <= 0;
113 |         en2 <= 0;
114 |         row_count <= row_count + 1'b1;
115 |       end
116 |       else
117 |       begin
118 |         count2 <= count2 + 1'b1;
119 |       end
120 |     end
121 |   end
122 |   if(en2)
123 |   begin
124 |     if(count3 == k-2)
125 |     begin
126 |       count3 <= 0;
127 |       en2 <= 1'b1;
128 |     end
129 |     else
130 |     begin
131 |       count3 <= count3 + 1'b1;
132 |     end
133 |   end
134 |   if(((count2 + 1) & 0 == 0) && (row_count & 0 == 0)) || (count3 == k-2) && (row_count & 0 == 0) || (count == (k-1)*n+k-1))
135 |   begin
136 |     en3 <= 1;
137 |   end
138 |   else
139 |   begin
140 |     en3 <= 0;
141 |   end
142 |   assign count_val = (en1 && en2 && en3);
143 |   assign end_count = (count == n+k-2) ? 1'b1 : 1'b0;
144 |   always @(posedge clk or posedge global_rst)
145 |   begin
146 |     if(global_rst)

```

Slika 10 – Kod u convolver-u – part 2

```

convolver.v
D:\Konacan_projekat_vivado\tpga-mi-accelerator\tpga-mi-accelerator.srcs\sources_1\twe\convolver.v

92 | reg en1,en2,en3;
93 |
94 | always @(posedge clk)
95 | begin
96 |   if(global_rst)
97 |   begin
98 |     count <= 0;           //master counter: counts the clock cycles
99 |     count2 <= 0;          //counts the valid convolution outputs
100 |     count3 <= 0;          // counts the number of invalid convolutions where the kernel wraps around the next row of inputs.
101 |     row_count <= 0;       //counts the number of rows of the output.
102 |     en1 <= 0;
103 |     en2 <= 1'b1;
104 |     en3 <= 0;
105 |   end
106 |   else if(ce)
107 |   begin
108 |     if(en1 && en2)
109 |     begin
110 |       if(count2 == n-k)
111 |       begin
112 |         count2 <= 0;
113 |         en2 <= 0;
114 |         row_count <= row_count + 1'b1;
115 |       end
116 |       else
117 |       begin
118 |         count2 <= count2 + 1'b1;
119 |       end
120 |     end
121 |   end
122 |   if(en2)
123 |   begin
124 |     if(count3 == k-2)
125 |     begin
126 |       count3 <= 0;
127 |       en2 <= 1'b1;
128 |     end
129 |     else
130 |     begin
131 |       count3 <= count3 + 1'b1;
132 |     end
133 |   end
134 |   if(((count2 + 1) & 0 == 0) && (row_count & 0 == 0)) || (count3 == k-2) && (row_count & 0 == 0) || (count == (k-1)*n+k-1))
135 |   begin
136 |     en3 <= 1;
137 |   end
138 |   else
139 |   begin
140 |     en3 <= 0;
141 |   end
142 |   assign count_val = (en1 && en2 && en3);
143 |   assign end_count = (count == n+k-2) ? 1'b1 : 1'b0;
144 |   always @(posedge clk or posedge global_rst)
145 |   begin
146 |     if(global_rst)

```

Slika 11 – Kod u convolver-u – part 3

```

119 end
120 else if (en1)
121   if (en1 && en2)
122     begin
123       if (count2 == n-k)
124         begin
125           count2 <= 0;
126           en2 <= 0;
127           row_count <= row_count + 1'b1;
128         end
129       else
130         begin
131           count2 <= count2 + 1'b1;
132         end
133       end
134     end
135   if (!en2)
136     begin
137       if (count3 == k-2)
138         begin
139           count3 <= 0;
140           en2 <= 1'b1;
141         end
142       else
143         count3 <= count3 + 1'b1;
144       end
145     end
146   if (((count2 + 1) & n == 0) && (row_count & n == 0)) || (count3 == k-2) && (row_count & n == 0) || (count == (k-1)*n-k-1)
147     //some in every a computations becomes valid
148     //some exceptional cases handled for high when count2 = 0
149     begin
150       en3 <= 1;
151     end
152   else
153     en3 <= 0;
154   end
155   assign conv_vld = (en1 && en2 && en3);
156   assign end_conv = (count == n*k-2) ? 1'b1 : 1'b0;
157   always @(posedge clk or posedge global_reset)
158   begin
159     if (global_reset)
160       conv_vld_di <= 0;
161     else
162       conv_vld_di <= conv_vld;
163   end
164   if (!en3)
165     assign valid_conv = conv_vld_di;
166   else
167     assign valid_conv = conv_vld;
168   end
169 endmodule

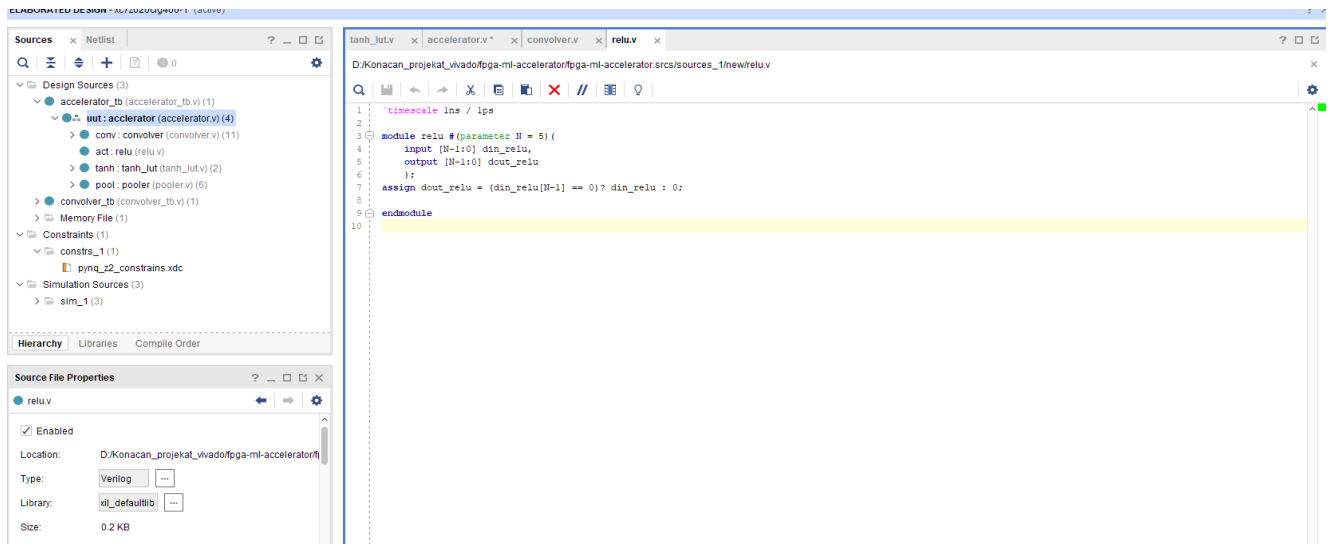
```

Slika 12 – Kod u convolver-u – part 4

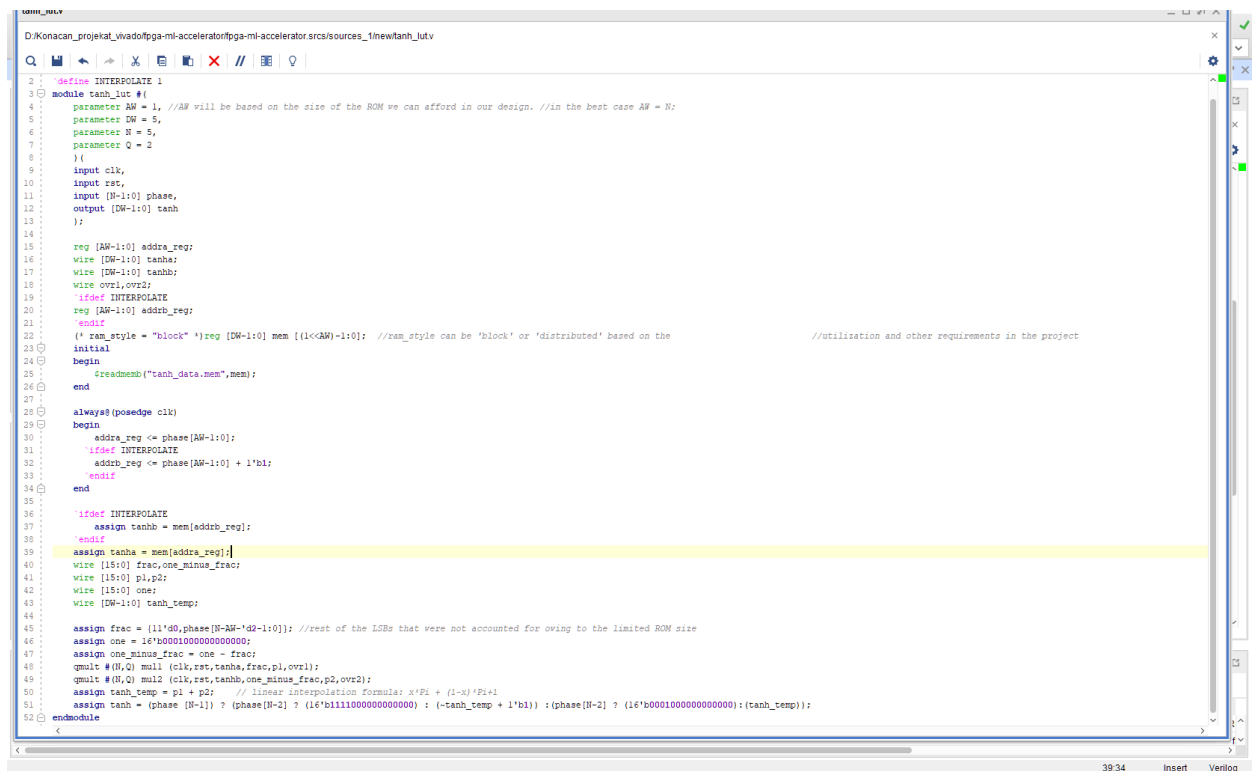
Ukratko objašnjenje koda u **convolver.v** - Primarna ideja koja stoji iza pristupa ovom dizajnu je da se izgradi visoka (pipelined streaming architecture) u kojoj modul za obradu ne mora da stane ni u jednom trenutku. tj. bilo koji deo konvolvera ne čeka da bilo koji drugi deo završi svoj posao i dostavi mu rezultat. Svaka faza obavlja mali deo celokupnog rada kontinuirano na drugom delu ulaza svakog takta. Ovo nije samo karakteristika ovog dizajna, to je opšti princip koji se zove „cevovod“ koji se široko koristi za razbijanje velikih računarskih procesa na manje korake i povećanje najveće frekvencije na kojoj celo kolo može da radi.

Naš dizajn koristi MAC (Multili and Accumulate) jedinice sa ciljem mapiranja ovih operacija u DSP blokove FPGA. Postizanje toga bi učinilo operacije množenja i sabiranja mnogo bržim i trošilo manje energije pošto su DSP blokovi implementirani u čvrstim makroima. što će reći, DSP blokovi su već sintetizovani, postavljeni i preusmereni u silicijum na FPGA uređaju (pyng-z2) na najefikasniji mogući način. Konvolver nije ništa drugo do skup MAC jedinica i nekoliko pomerajućih registara, koji kada se napajaju pravim ulazima, izlaze rezultat konvolucije nakon fiksnog broja ciklusa takta.

Kod koji se nalazi u podmodelu relu i tanh_lut(Funkcije Aktivacije):

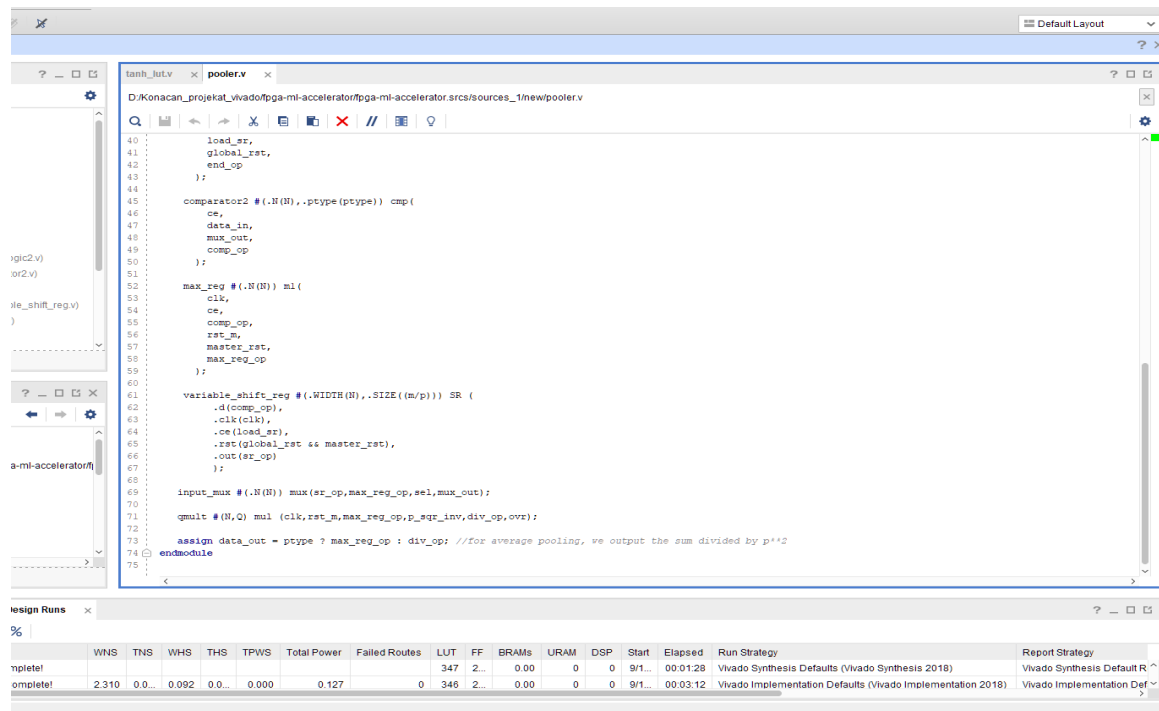


Slika 13 – Kod(implentacija) funkcije RELU(Ispravljačka funkcija)

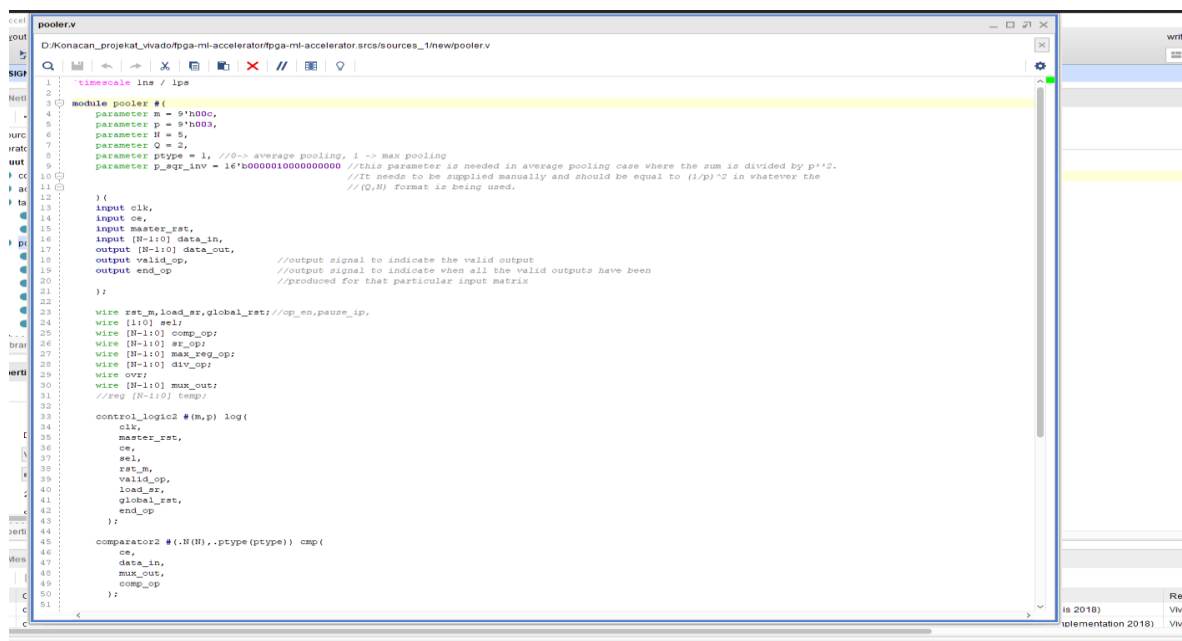


Slika 14 – Kod(implentacija) funkcije tanh(Hiperbolički tangens)

Kod koji se nalazi u podmodelu pooler(Objedinjavanje(Pooling)):

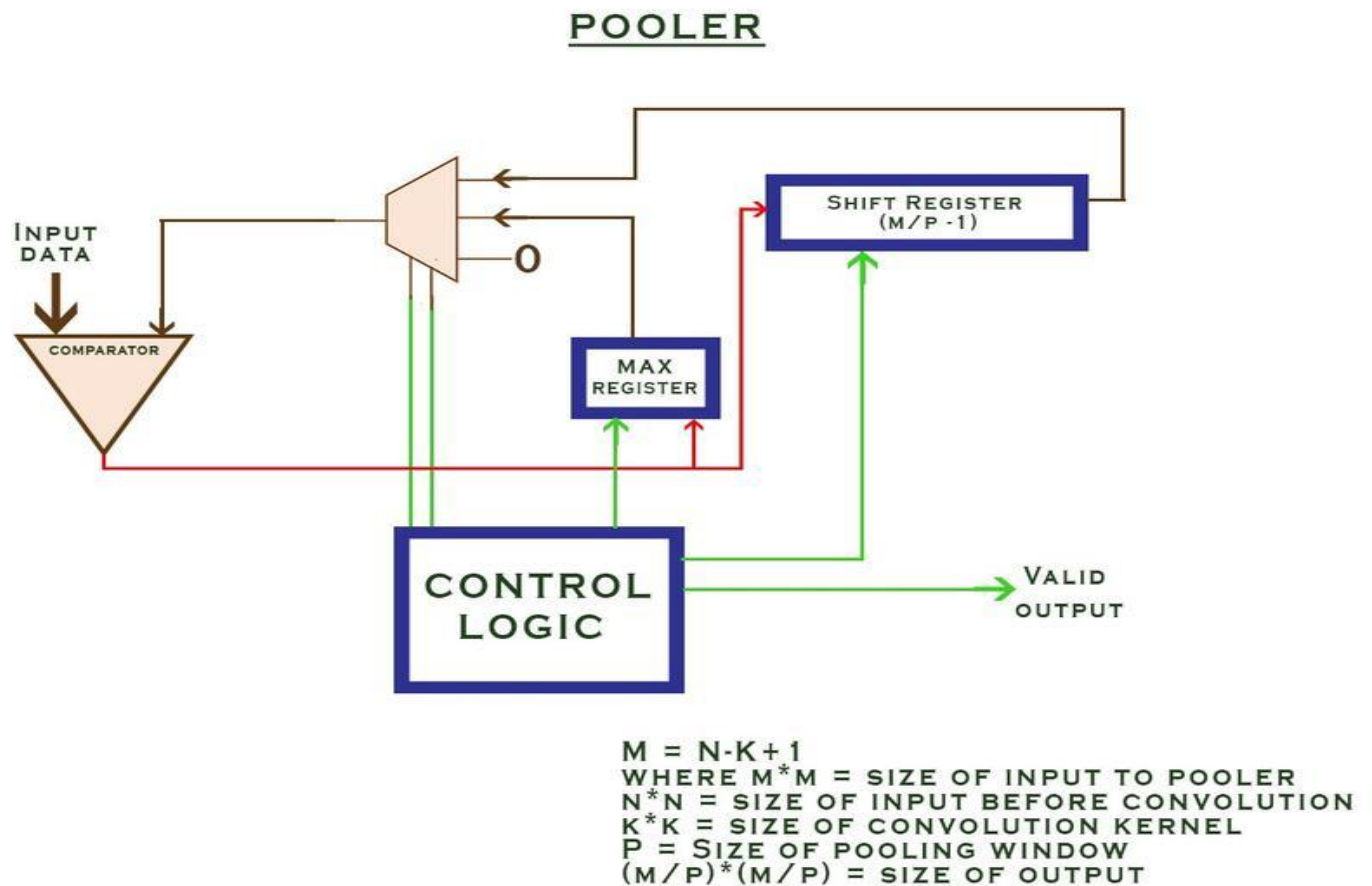


Slika 15 – Kod pooler-u – part 1



Slika 16 – Kod pooler-u – part 2

Puler je deo ovog dizajna koji implementira operaciju Max-Pooling, kod za Max-Pooling to je uglavnom gomila kontrolnih izraza koji generišu kontrolne signale za sve pojedinačne jedinice u prikupljanju podataka na osnovu određenih uslova koji variraju sa vrednostima N , K i P .



Slika 17 – Pooler objašnjen kroz logička kola, register-e, kontrolne jedinice

SIMULACIJA ACCELERATOR-A

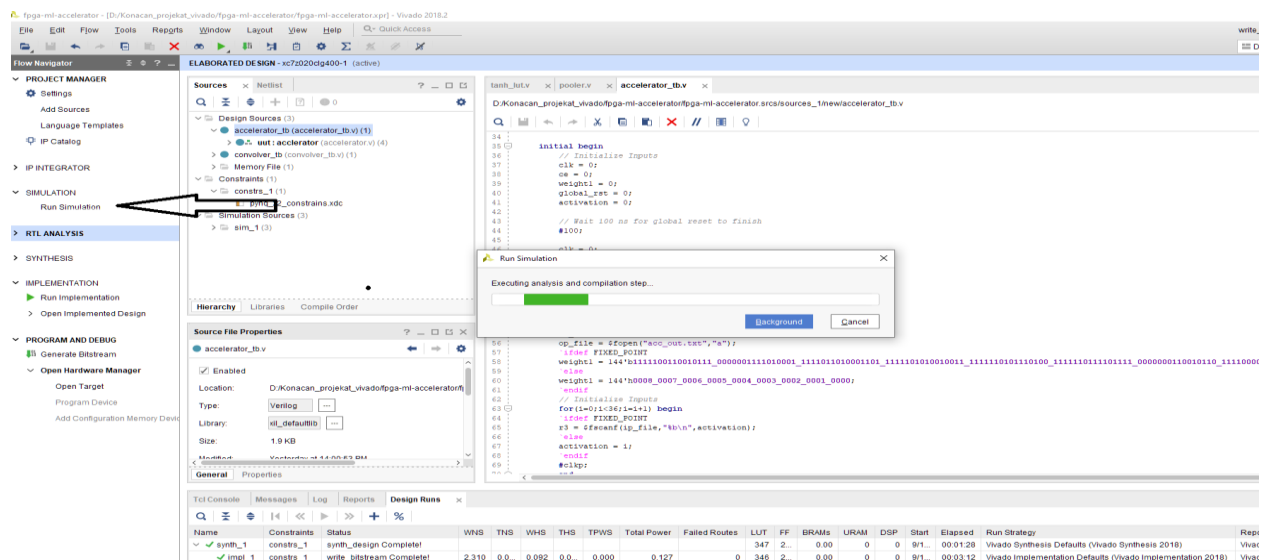
Analiza I pokretanje glavnog modela accelerator.v gde se kreira simulacija accelerator_tb.v, gde imamo pristup vrednostima promenljivih u vremenskim intervalima. Možemo podesiti vrednosti promenljivih(input-a) kroz vreme u accelerator_tb.v.

```

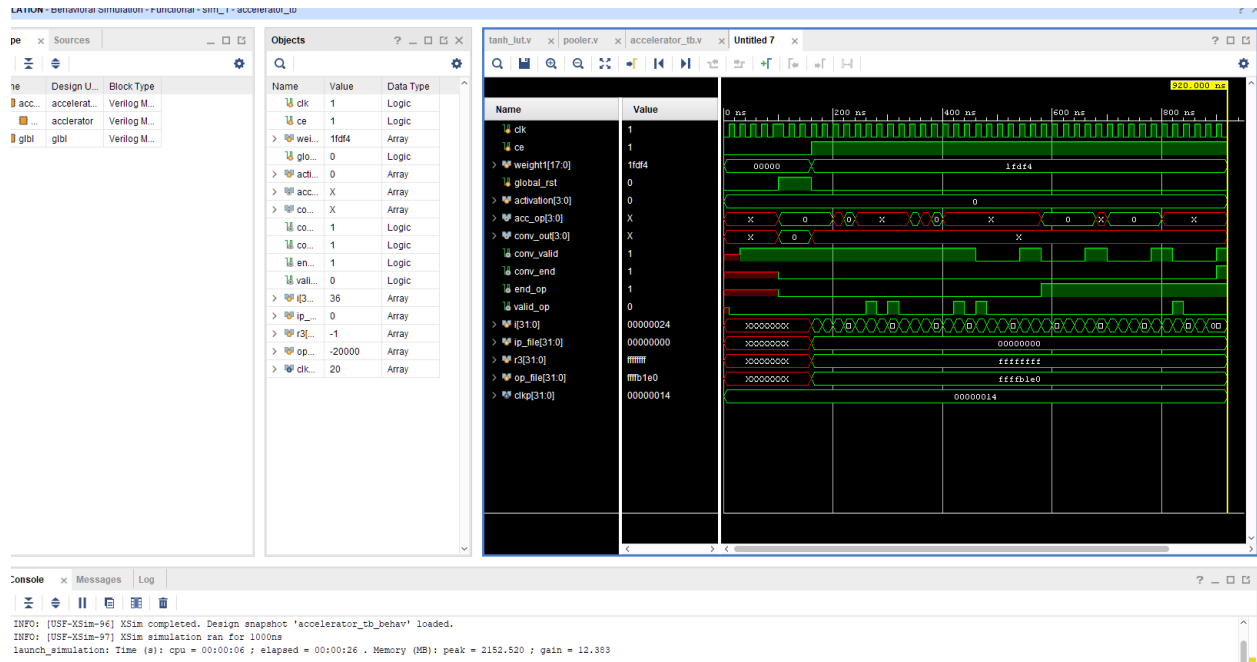
34:
35:
36:   initial begin
37:       // Initialize Inputs
38:       clk = 0;
39:       ce = 0;
40:       weight1 = 0;
41:       global_rst = 0;
42:       activation = 0;
43:
44:       // Wait 100 ns for global reset to finish
45:       #100;
46:
47:       clk = 0;
48:       ce = 0;
49:       weight1 = 0;
50:       activation = 0;
51:       global_rst = 1;
52:       #60;
53:       global_rst = 0;
54:       // #100;
55:       ce=1;
56:       op_file = $fopen("activations.txt","w");
57:       op_file = $fopen("acc_out.txt","a");
58:       //end FIXED_POINT
59:       weight1 = 14'b111100110010111_000000111101001_11101101001101_111101010010011_111101011101100_11110101110111_0000000110010110_1111000011010101_11111011110110100;
60:       //ce
61:       weight1 = 14'h0008_0007_0006_0005_0004_0003_0002_0001_0000;
62:       //end if
63:       // Initialize Inputs
64:       for (i=0; i<36; i=i+1) begin
65:           if (i==2 FIXED_POINT
66:               r3 = $fscanf(op_file, "%b\n", activation);
67:           else
68:               activation = 1;
69:           end if
70:           #clkp;
71:       end
72:
73:   always # (clkp/2) clk = ~clk;
74:
75:   always#(posedge clk) begin
76:       if (valid_op == 'end_op) begin
77:           $fdisplay(op_file, "%b", acc_op);
78:       end
79:       if (conv_end) begin
80:           if (ce)
81:               begin
82:                   $fdisplay(op_file, "%sk0d", "end", 0);
83:                   $finish;
84:               end
85:       end
86:   endmodule

```

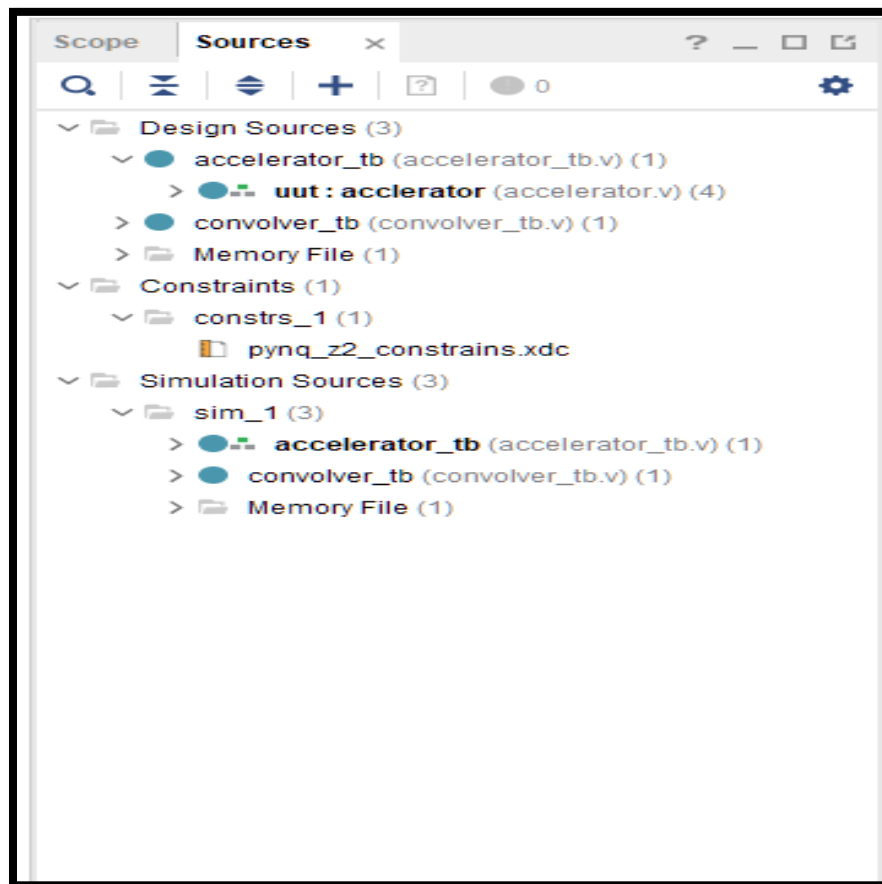
Slika 18 – Kod simulacije



Slika 19 – Učitavanje i pokretanje simulacije (Run Simulation)



Slika 20 – konačna simulacija, vrednost promeljivih kroz vremenski interval do 800 ns

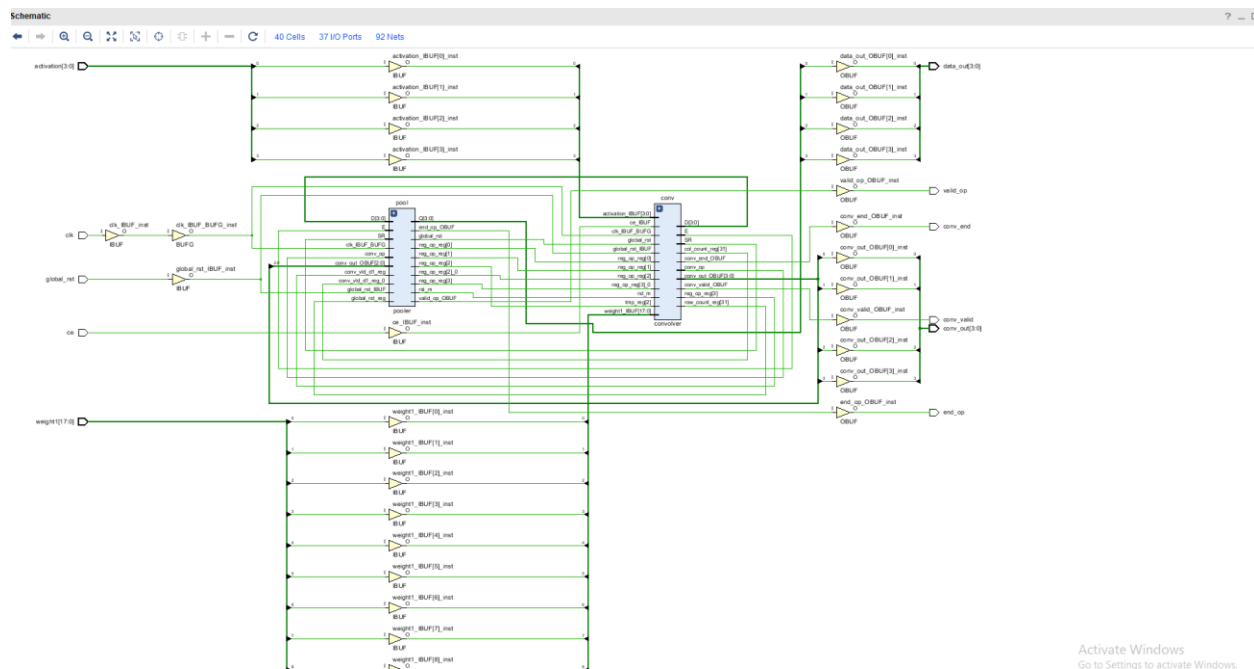


Slika 21 – Top model simulacije accelerator

SINTEZA I IMPLEMENTACIJA

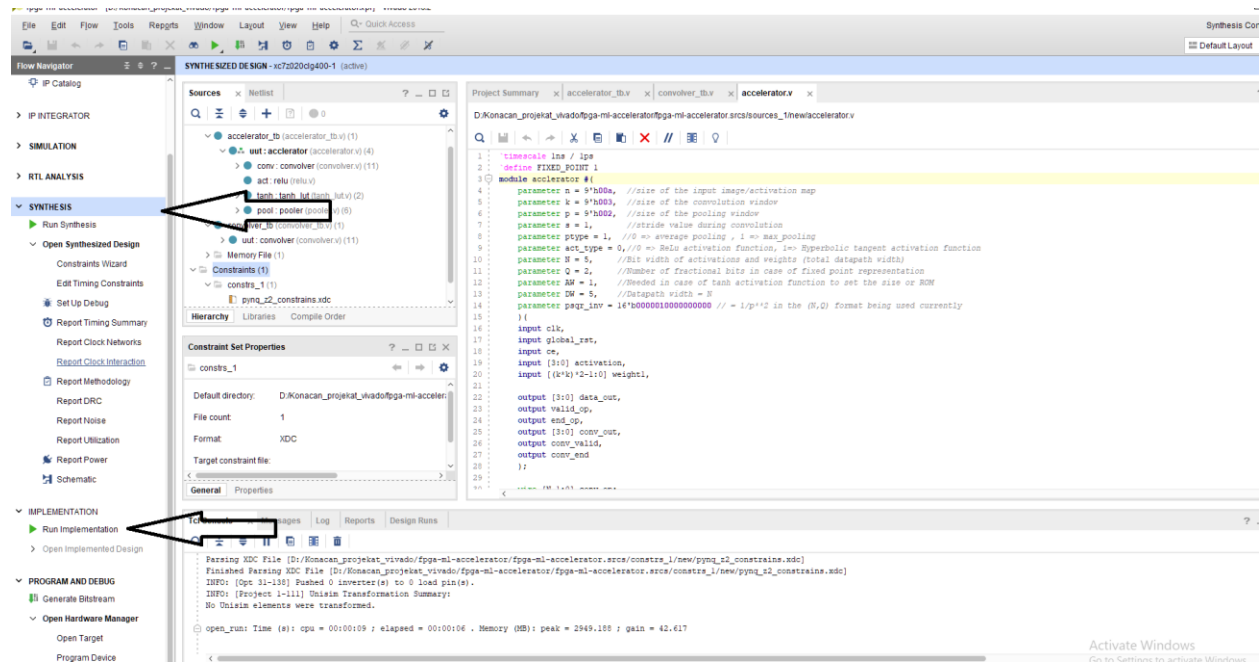
Za uspešno povezivanje i projektovanje koda u vivado-u na ploču PYNQ-Z2 mora da se odrade dva koraka, to je da naš izvorni kod prođe sinezu(Synthesis) i implementaciju(Implementation), pre početka izvršenja sinteze obavezno je povezati ploču PYNQ-Z2 sa računarom, koristeći USB kabal. Kada povežemo ploču sa računarom moramo aktivirati switch na gore i u tom trenutku se pali crvena led dioda(CR-8) , što indicira da je uspešna konekcija izvršena. Sledeći korak jeste ubacivanje Constraints-a, to su kodovi koji se nalaze na sajtu imi-ja(Odeljak treća godina->Projektovanje VLSI Sistema) koristimo ih kako bi implemenirali ulazne i izlazne vrednosti u program na ploču PYNQ-Z2.

U nastavku ću pokazati kako izgleda uspešna sinteza ...



Slika 22 – Shematic našeg koda

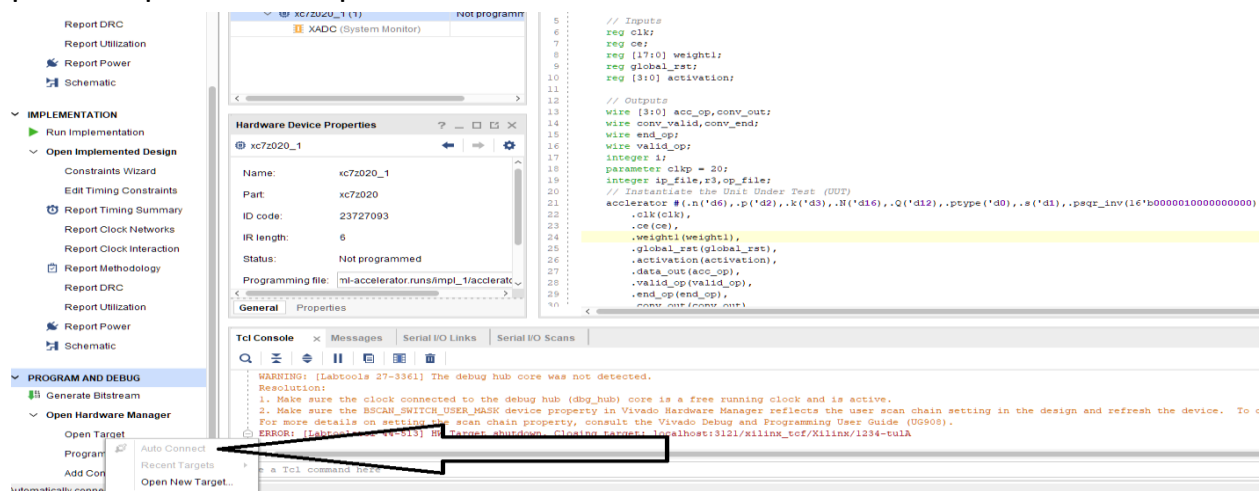
Shematic je prikaz našeg koda kroz veze i logička kola i vrednostima ulaznih i izlaznih vrednosti iz programa.



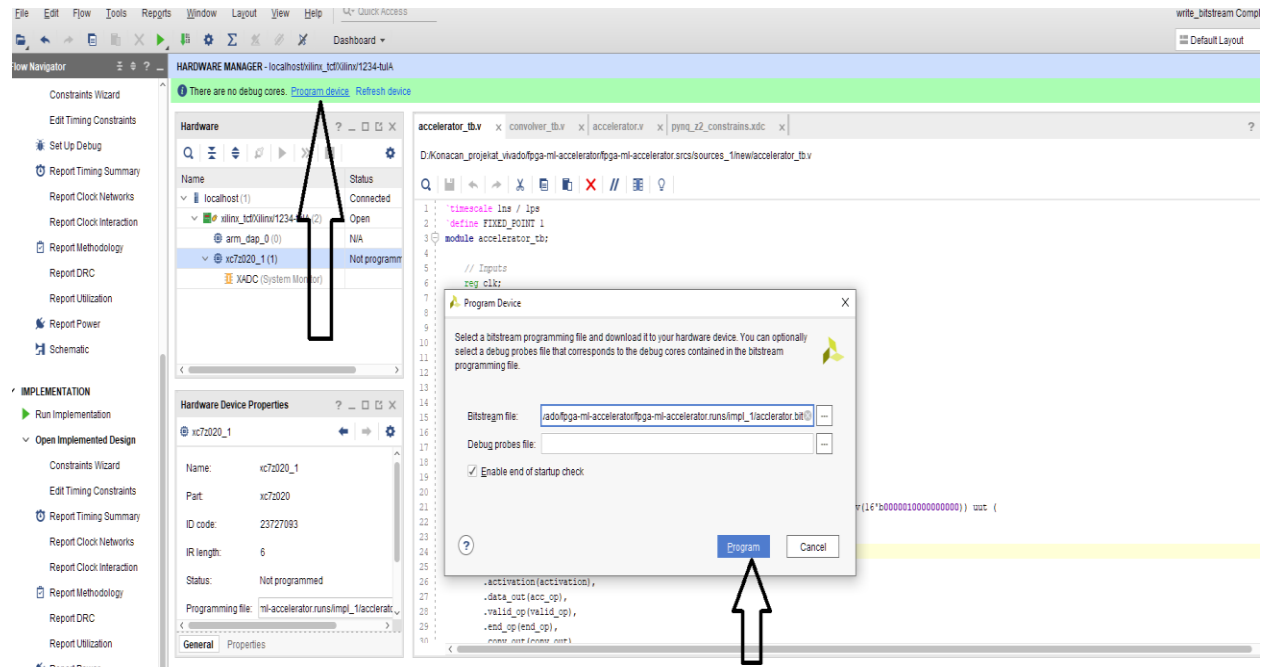
Slika 23 – Pokretanje sineze i implementacije

POKRETANJE BITSTREAM-A

Ukoliko prolazi sinteza i implementacija, onda možemo da pokrenemo i Bitstream ukoliko je prepoznalo ploču idemo na Open Hardware Manager -> Open Target -> auto connect. Ploča se povezuje, Bitstream je binarna sekvenca. Služi za uspostavljanje komunikacionog kanala programa i ploče u našem slučaju PYNQ-Z2. Nakon završetka kreira se Bitstream datoteka koja sadrži programske podatke povezane sa pločom.

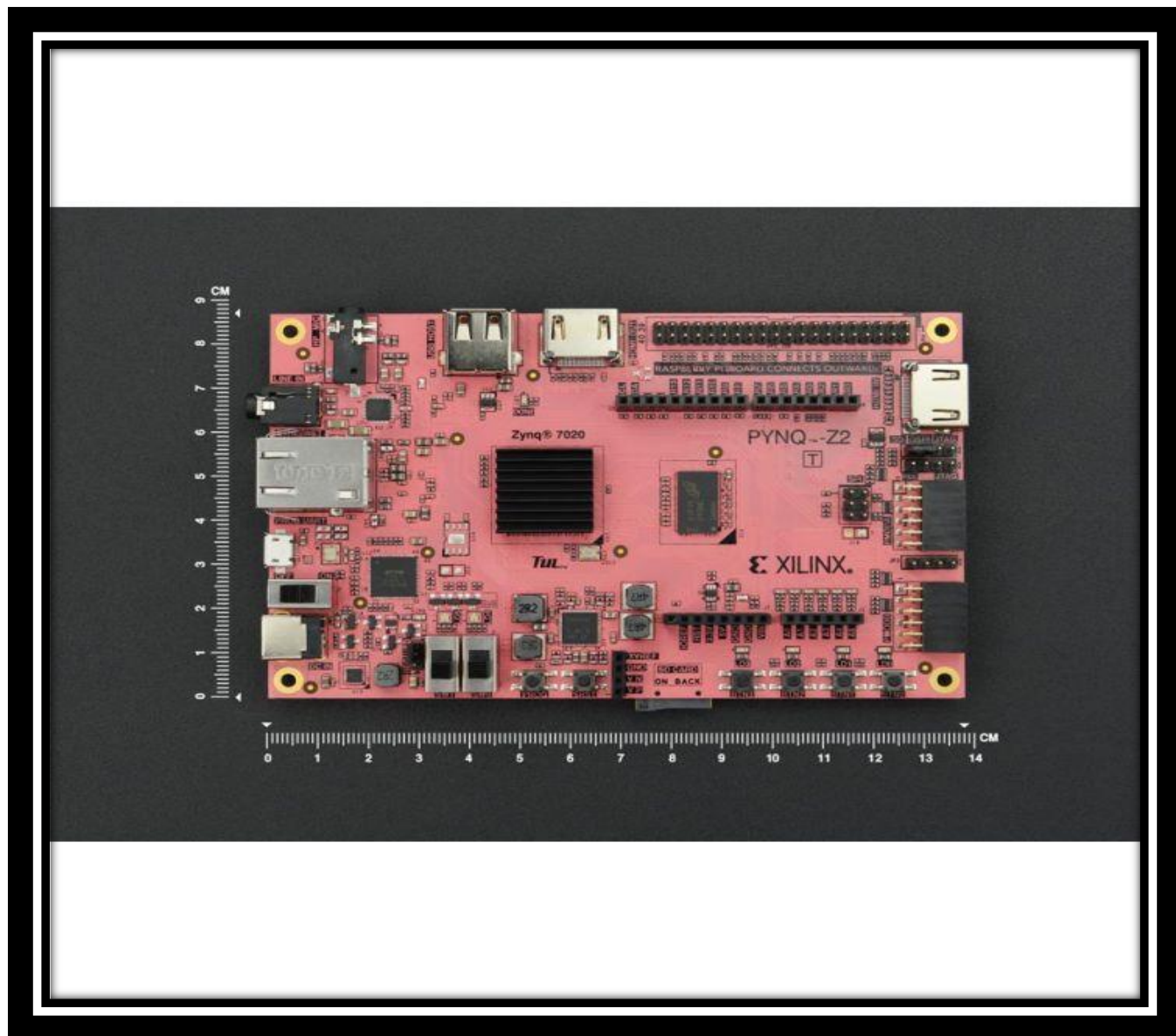


Slika 24 – Izvršena provera za BitStream- auto connect opcija

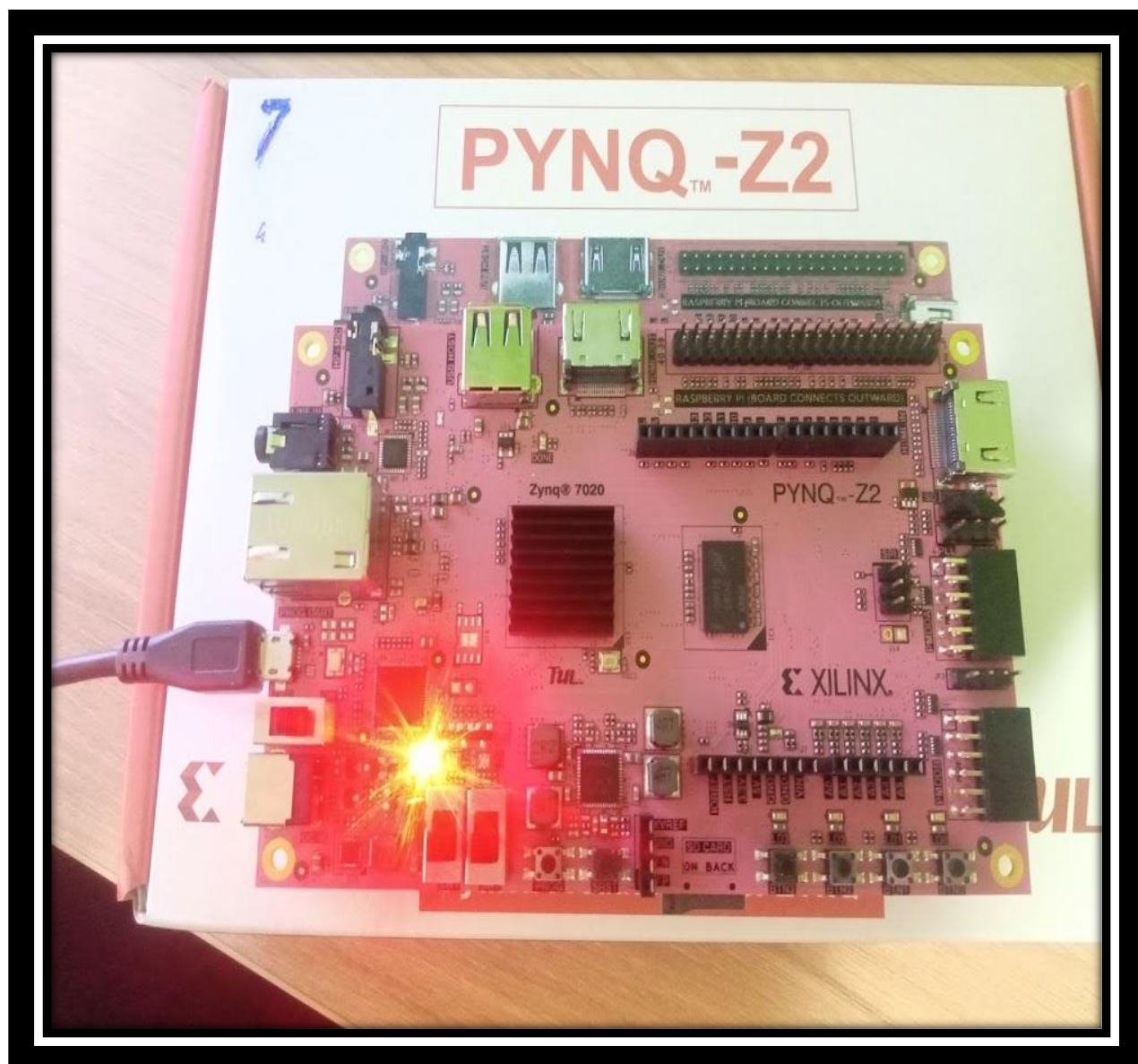


Slika 25 – Nakon auto connect koraci u pokretanju BitStream-a

IZGLED PYNQ-Z2 NAKON BITSTREAM-A



Slika 26 –Izgled ploče koja nije povezana



Slika 27 –Izgled ploče koja je povezana

Jasno na slici možemo uočiti da je ploča povezana sa računarom preko Micro-USB-a, DIP Switches su ugašeni, ipak radi led diode na LD5 boja je crvena. Power switch je uključen, da nije ne bi postojala konekcija. Imamo dva button-a za programming I system reset, znači ponovno konektovanje. Nakon pokretanja dva switch-a vrednosti na ploči se menjaju(promena boje), pritiskom na button-e menjaju se led diode kod switch-eva(plava,crvena, zelena) u zavisnosti koji button se pritisne(BTN0,BTN1,BTN2,BTN3).

Literatura

1. <https://www.bittware.com/resources/cnn/>
2. <https://cs231n.github.io/convolutional-networks/#conv>
3. Izvorni kod github - <https://github.com/thedatabusdotio/fpga-ml-accelerator>
4. VLSI sistemi - <https://imi.pmf.kg.ac.rs/moodle/course/view.php?id=555>
5. <https://www.amd.com/en/products/xilinx>
6. <https://www.semanticscholar.org/paper/FPGA-based-Acceleration-for-Convolutional-Neural-on-Huynh/be1733f215cef2bc0bb0be731c3aca009795e4b0?p2df>