ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ (MICROLAB)

5η Εργαστηριακή Αναφορά στο μάθημα "**ΨΗΦΙΑΚΑ ΣΥΣΤΗΜΑΤΑ VLSI "** του 8ου Εξαμήνου

των φοιτητών της **ομάδας 17**,

Εμμανουήλ Αναστάσιου Σερλή, Α.Μ. 03118125
Κωνσταντίνου Ιωάννου, ΑΜ: 03119840

Υλοποίηση FIR φίλτρου με AXI διεπαφή σε ZYNQ SoC FPGA

Πριν αρχίσουμε την περιγραφή των βημάτων για την υλοποίηση του φίλτρου FIR (της 4ης εργαστηριακής άσκησης) στο **ZYNQ SoC FPGA,** αξίζει να σημειωθεί ότι τροποποιήσαμε το φίλτρο FIR σε ορισμένα σημεία. Συγκεκριμένα**,** τροποποιήσαμε την μονάδα **Control Unit** έτσι ώστε ο Counter (που δίνει τις διευθύνσεις στις ram/rom) να παγώνει ενώ όταν λάβει valid_in να μετράει μέχρι την τιμή 8 αγνοώντας ενδιάμεσα valid_in. Το φίλτρο μας έβγαζε σωστά αποτελέσματα πριν την εν λόγω αλλαγή, αλλά μόνο για ιδανικές περιπτώσεις (δηλαδή valid_in και counter να ξεκινάνε μαζί), κάτι το οποίο είναι εφικτό στο testbench αλλά όχι στην πραγματικότητα για την υλοποίηση στο FPGA, λόγω των overhead κύκλων που επιφέρει η επικοινωνία με τον επεξεργαστή.

Παρακάτω φαίνεται σε κώδικα η τροποποιημένη υλοποίηση του **control unit**

```
178  use IEEE.STD_LOGIC_UNSIGNED.ALL;
179
180  entity Control_Unit is...
192
193  architecture Behavioral of Control_Unit is
194      signal count_reg : STD_LOGIC_VECTOR (2 downto 0) := (others => '0');
195      signal  waitt : std_logic ;
196      signal calculate : std_logic := '0';
197      signal flag : std_logic;
198      signal valid_out_temp:  STD_LOGIC;
199      signal s1,s2,s3,s4,s5,s6 :std_logic ;
200
201
202     --comp7
203         component dff2 is...
211
212  begin
213
214      process (clk)
215      begin
216
217         if rst = '1' then
218            count_reg <= (others => '0');
219           valid_out_temp<='0';
220            mac_init<='1';

221           elsif rising_edge(clk) then
222           if (valid_in = '1') and (calculate = '0') then
223               flag<='1';
224               --count_reg <= count_reg + 1;
225               end if;
226             if count_reg = "000" then
227              mac_init<='1';
228  --            flag <= '0';
229               if calculate = '1' then
230                  valid_out_temp <= '1';
231               else   valid_out_temp <= '0';
232                end if;
233
234               if(valid_in = '0') then
235                we<= '0';
236               calculate <= '0';
237              --  count_reg <= "000";
238
239               else
240                 we<= '1';
241                 calculate <= '1';
242               end if;
243            else -- counter = mid
244            we <='0';
245            valid_out_temp<='0';
246            mac_init<='0';
247            end if;
```

```
248
249           if count_reg = "111" then
250              flag <= '0' ;
251              end if;
252
253          if flag='1' then
254              count_reg <= count_reg+1;
255          else
256               count_reg <= "000";
257          end if;
258        ram_addr <= count_reg;
259        rom_addr <= count_reg;
260        counter <= count_reg;
261
262       -- valid_out_dff: dff_big port map(d=>val
263      end if;
264      end process;
265
```

Αφού τελειώσαμε με αυτήν την υποσημείωση αρχίζουμε τα βήματα, ώστε να υλοποιήσουμε το FIR φίλτρο στο Zynq.

Αρχικά, μετά την δημιουργία new_project, πάμε tools -> Create and Package new IP, ώστε να δημιουργήσουμε ένα νέο IP με το FIR. Στο νέο παράθυρο που ανοίγει το vivado για να δημιουργήσουμε ένα νέο IP βάζουμε αρχικά μέσω του add_sources ->design_sources και προσθέτουμε το VHDL αρχείο για το FIR (που φτιάξαμε στην προηγούμενη άσκηση).

Στην συνέχεια χρειάζεται να τροποποιήσουμε τον κώδικα για AXI, παρακάτω παραθέτουμε ΜΟΝΟ τα κομμάτια του κώδικα που τροποποιήσαμε.

Αρχικά χρειάζεται να ορίσουμε μερικά βοηθητικά σήματα που θα χρησιμοποιήσουμε και να βάλουμε ως component το FIR φίλτρο.

```
122           --extras
123         signal A_ip : std_logic_vector (31 downto 0); --A_temp = 000..00&rst&valid_in&X
124         signal B_ip : std_logic_vector(31 downto 0); --B_temp = 000..00&valid_out&y
125         signal temp_out: std_logic_vector(31 downto 0);
126         signal out_flag: std_logic := '0';
127         --used from FIR
128         signal rst_ip : std_logic;
129         signal valid_in_ip: std_logic;
130         signal x_ip: std_logic_vector(7 downto 0);
131         signal valid_out_ip: std_logic;
132         signal y_ip: std_logic_vector(18 downto 0);
133         --not used from FIR
134         signal en_ram_rom_ip: std_logic;
135         signal rom_out_ip,ram_out_ip: std_logic_vector(7 downto 0);
136         signal rom_add_ip,ram_add_ip,counter_control_ip: std_logic_vector(2 downto 0);
137         signal mac_init_ip,we_out_ip: std_logic;
138
139          component FIR is -- custom FIR added
140                Port ( clk : in std_logic;
141                       rst : in std_logic;
142                       valid_in : in std_logic;
143                       en_ram_rom: in std_logic;
144                       x : in std_logic_vector(7 downto 0);
145                       valid_out : out std_logic;
146                       y_final : out std_logic_vector (18 downto 0);
147                       rom_out,ram_out:out STD_LOGIC_VECTOR (7 downto 0);
148                       rom_add,ram_add,counter_control:out STD_LOGIC_VECTOR (2 downto 0);
149                       mac_init : out std_logic;
150                       we_out:out std_logic
151                       );
```

Μετά κάνουμε mapping τα variables του φίλτρου με τα signals.

```
-- Add user logic here
FIR_bb : FIR  port map (
              --used from A,B
              clk => S_AXI_ACLK,
              rst => rst_ip,
              x => x_ip,
              valid_in => valid_in_ip,
              valid_out => valid_out_ip,
              y_final => y_ip,
              --not used from A,B
              en_ram_rom => '1', --en_ram_rom_ip ,
              we_out => we_out_ip,
              mac_init => mac_init_ip,
              rom_out => rom_out_ip,
              ram_out => ram_out_ip,
              rom_add => rom_add_ip,
              ram_add => ram_add_ip,
              counter_control => counter_control_ip
              );
```

Και ενώνουμε τα bits μέσα σε ένα process ώστε να δέχεται είσοδο A 32bit και να βγάζει έξοδο B 32bit όπως ζητάει η εκφώνηση.

```
451         process (S_AXI_ACLK) is
452                 begin
453                                 --get input from reg0
454                                 A_ip <= slv_reg0;
455                                 x_ip <= A_ip(7 downto 0);
456                                 valid_in_ip <= A_ip(8);
457                                 rst_ip <= A_ip(9);
458                                 --write output to reg1
459                                 B_ip <= "000000000000"&valid_out_ip & y_ip;
460
461
462                 end process;
```

Στο σημείο του κώδικα που το hardware λαμβάνει δεδομένα από το software (δηλαδή γράφουμε στο register – εμείς επιλέξαμε να γράφουμε στον slv_reg0), κάνουμε τις εξής αλλαγές:

- Σβήνουμε τα σημεία που γράφουμε στον slv_reg1 τον οποίο εμείς χρησιμοποιούμε για να διαβάζουμε, δηλαδή για να στέλνουμε από το hardware στο software.Αυτό το κάνουμε ώστε να μην έχουμε multi_drive error.
- Χρειάζεται να προσθέσουμε στην συνθήκη if (έτοιμο να διαβάσει από το software) ένα else ώστε να κάνουμε το valid_in =0, αυτό συμβαίνει ώστε το FIR να μην διαβάζει πολλές φορές την ίδια είσοδο αλλά να λαμβάνει και να επεξεργάζεται κάθε είσοδο μόνο μια φορά.

```
244     process (S_AXI_ACLK)
245     variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
246     begin
247       if rising_edge(S_AXI_ACLK) then
248         if S_AXI_ARESETN = '0' then
249           slv_reg0 <= (others => '0');
250          -- slv_reg1 <= (others => '0');
251           slv_reg2 <= (others => '0');
252           slv_reg3 <= (others => '0');
253         else
254           loc_addr := axi_awaddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
255           if (slv_reg_wren = '1') then
256             case loc_addr is
257               when b"00" =>
258                 for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
259                   if ( S_AXI_WSTRB(byte_index) = '1' ) then
260                     -- Respective byte enables are asserted as per write strobes
261                     -- slave registor 0
262                     slv_reg0(byte_index*8+7 downto byte_index*8) <= S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
263                   end if;
264                 end loop;
265               when b"01" =>
266                 slv_reg2 <= (others => '0'); -- ?????
267               --  for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
268                 --  if ( S_AXI_WSTRB(byte_index) = '1' ) then
269                     -- Respective byte enables are asserted as per write strobes
270                     -- slave registor 1
271                 --    slv_reg1(byte_index*8+7 downto byte_index*8) <= S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
272                 --   end if;
273                -- end loop;
282               when b"11" =>
283                 for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
284                   if ( S_AXI_WSTRB(byte_index) = '1' ) then
285                     -- Respective byte enables are asserted as per write strobes
286                     -- slave registor 3
287                     slv_reg3(byte_index*8+7 downto byte_index*8) <= S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
288                   end if;
289                 end loop;
290               when others =>
291                 slv_reg0 <= slv_reg0;
292                 slv_reg1 <= slv_reg1;
293                 slv_reg2 <= slv_reg2;
294                 slv_reg3 <= slv_reg3;
295             end case;
296             else  slv_reg0(8) <=  '0'; --dont write on reg0 if write enable is zero
297           end if;
298         end if;
299       end if;
300     end process;
```

Ομοίως στο process όπου κάνουμε Send_Read_data δηλαδή όπου το hardware στέλνει δεδομένα στο software ακολουθούμε την παρακάτω λογική :

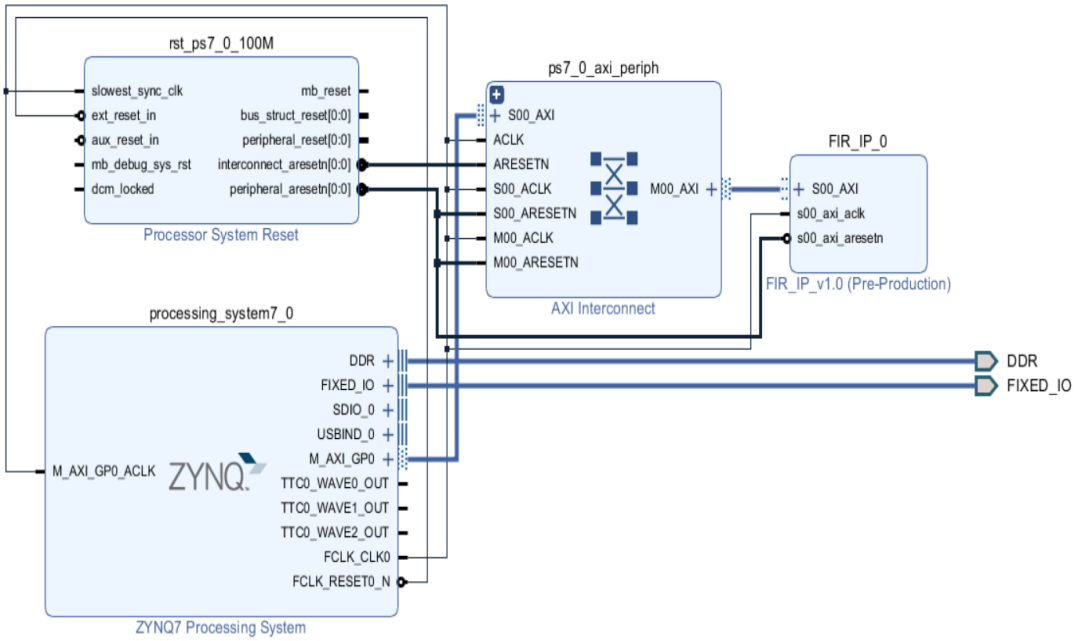
Όταν το software(master) στείλει έγκυρα δεδομένα στο hardware(slave), το FIR βγάζει δεδομένα εξόδου με valid_out =1 και το hardware τα αποθηκεύει στον slv_reg1, στην συνέχεια μέχρι ο master (software) να στείλει **Read_ready** δηλαδή ότι είναι έτοιμος να λάβει δεδομένα απλώς κρατάμε την valid τιμή εξόδου του FIR φίλτρου τιμή στο slv_reg1. Τέλος, όταν έρθει το σήμα ότι ο master είναι έτοιμος να διαβάσει, ο slave(hardware) στέλνει τα δεδομένα του slv_reg1 μέσω του axi_rdata στον master(software) και μηδενίζει το bit που αντιπροσωπεύει valid_out για να μην αποθηκεύσει ξανά την ίδια τιμή, και ουσιαστικά ο slave περιμένει μέχρι ο master να στείλει δεδομένα με valid_in =1, ώστε το FIR να παράγει δεδομένα εξόδου με valid_out =1.

```
403       -- Output register or memory read data
404     process( S_AXI_ACLK ) is
405     begin
406       if (rising_edge (S_AXI_ACLK)) then
407         if ( S_AXI_ARESETN = '0' ) then
408           axi_rdata  <= (others => '0');
409         --  slv_reg1 <= (others => '0'); --new code
410         else
411
412             -- When there is a valid read address (S_AXI_ARVALID) with ...
416                  if( valid_out_ip = '1')   then
417                      slv_reg1 <= B_ip ; --"000000000000"&valid_out_ip & y_ip; --B_ip;
418
419                  elsif(slv_reg_rden = '1') then
420                     axi_rdata <= reg_data_out; --slv_reg1;
421                     slv_reg1(19) <= '0';
422
423                   else  slv_reg1 <= slv_reg1;
424               end if;
425           end if;
426         end if;
427     end process;
428
```

Αφού αποθηκεύσουμε τις αλλαγές μας έχουμε έτοιμο το νέο IP για το FIR φίλτρο.

Επιστρέφουμε λοιπόν στο αρχικό μας project και μέσω setting->IP->repository προσθέτουμε το path που δημιουργήσαμε για το FIR_IP ώστε να μπορέσουμε να το προσθέσουμε στο design.

Βάζουμε στο design το Zynq και το FIR_IP και οι συνδέσεις γίνονται αυτόματα οπότε καταλήγουμε στο τελικό design όπως φαίνεται παρακάτω



Στην συνέχεια κάνουμε HDL_wrapper το design μας, RTL_analysis και Run_implementation ώστε να σιγουρευτούμε ότι δεν έχουμε κάποιο error ή κάποιο σημαντικό warning. Τέλος δημιουργούμε το bitstream (generate_bitstream), File->Export->Export_hardware και ανοίγει αυτόματα το SDK.

Απομένει να σχολιάσουμε τον κώδικα του master-software που γράψαμε στο sdk σε γλώσσα C.

Αφού βρούμε από το xparametres.h το MY_IP_BASEADDR για την συγκεκριμένη εφαρμογή και ορίσουμε τις μεταβλητές μας, ζητάμε από τον χρήστη να δώσει τις τιμές rst,valid_in,data_in και στην συνέχεια αφού τα κάνουμε shift για να τοποθετηθούν στην σωστή θέση και τα ενώνουμε στην μεταβλητή A των 32 bit. Με την εντολή Xil_out32 στέλνουμε τα δεδομένα στον slave (hardware), προσθέτουμε 0 στην διεύθυνση γιατί γράφουμε στον slv_reg0.

```c
#define MY_IP_BASEADDR 0x43C00000 //from  xparameters.h

int main() {
  init_platform();

      unsigned int A,data_in,y,valid_in ,rst,valid_out,B;
      int RST;

     while(1) {
       valid_out =0;


       xil_printf("Give input N\n");
       scanf("%d",&data_in);

       xil_printf("give reset\n");
       scanf("%d",&rst);

       xil_printf("give valid in\n");
       scanf("%d",&valid_in);

       valid_in = valid_in << 8;
       RST =rst;
       rst =rst << 9;
       A = rst | valid_in | data_in;


        Xil_Out32((MY_IP_BASEADDR+0x00), A); //write to FIR to reg0
```

Στην συνέχεια, διαβάζουμε την τιμή εξόδου του φίλτρου από τον slv_reg1, η διεύθυνση του οποίου αντιστοιχεί στην διεύθυνση του MY_IP_BASEADDR + 4. Αν λάβουμε valid_in = 0 ή reset = 1, τότε μέσω της εντολής xil_in32 διαβάζουμε τον slv_reg1 και απομονώνουμε το valid_out το οποίο μέσα σε αυτό το if condition περιμένουμε προφανώς να είναι μηδέν.

Εάν τώρα δεχτήκαμε μια έγκυρη τιμή valid_in =1 (και προφανώς δεν έχουμε ενεργοποιημένο το reset), τότε σε μια εσωτερική while{} διαβάζουμε συνεχώς τον slv_reg1 μέχρι να μας δώσει μια valid τιμή εξόδου ο slave(FIR_IP), δηλαδή μέχρι να γίνει το valid_out =1, τέλος με μια απλή μάσκα απομονώνουμε τα data_out και τα τυπώνουμε ώστε να τα δει ο χρήστης μέσω σειριακής επικοινωνίας στο terminal του SDK. Μετά το πρόγραμμα συνεχίζει και ζητά από τον χρήστη νέες τιμές εισόδου.

Σημειώνουμε ότι χρησιμοποιούμε την εντολή xil_printf και όχι την printf γιατί επιβαρύνει λιγότερο το zynq.

```
            if( valid_in == 0  || RST == 1 )  {
                  B = Xil_In32(MY_IP_BASEADDR+0x04);
                 valid_out = B &  0x80000;
                   xil_printf("\nvalid_out   is: %d \n",  valid_out);
                  }
          else  {
            while (valid_out == 0 ) {
                 B = Xil_In32(MY_IP_BASEADDR+0x04);
                 valid_out = B &  0x80000;
                   xil_printf("\nB  is: %d \n",  B);

                }
                y = B & 0x7FFFF; //mask for y
                xil_printf("%u\n",y);
            }
      }
    cleanup_platform();
    return 0;
}
```