ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ψηφιακά Συστήματα VLSI

5η Εργαστηριακή Άσκηση

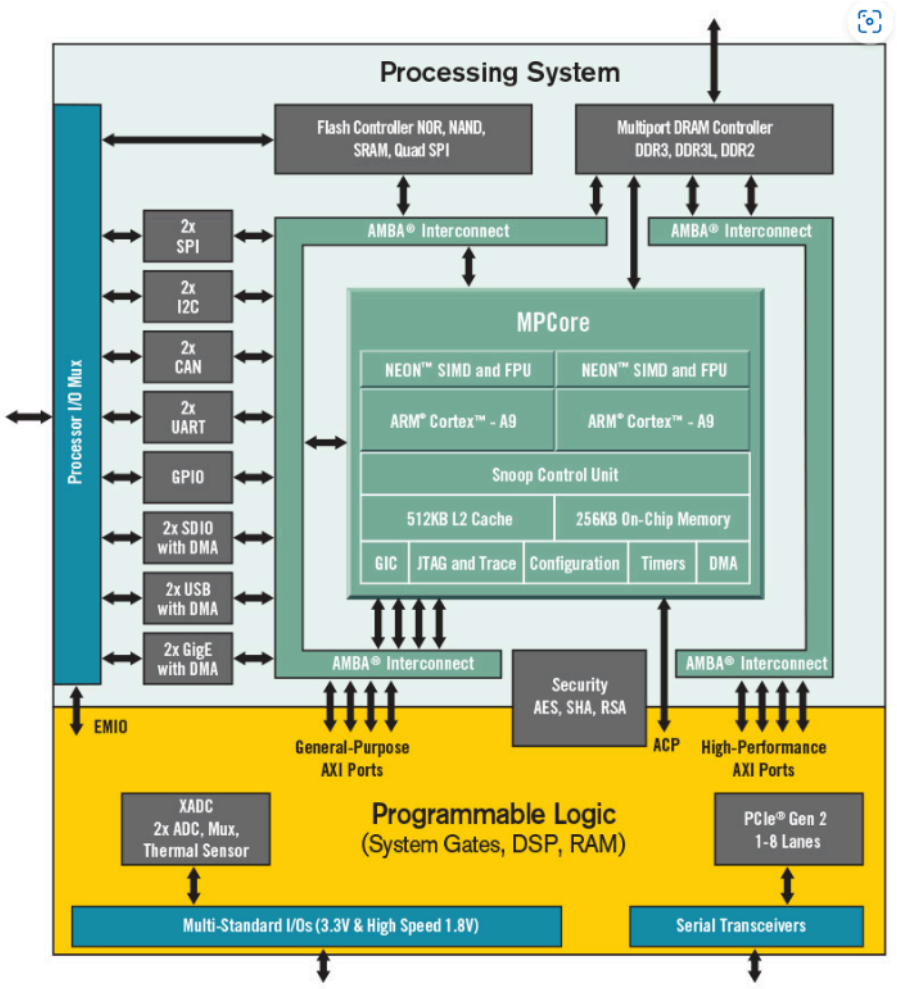
Δημήτρης Γεωργακόπουλος - 03119207 Γιώργος Τριανταφύλλου - 03120005

➢ **Εισαγωγή.....2**
➢ **AXI4-Lite slave διεπαφή + FIR φίλτρο ως custom IP block.....3**
➢ Υλοποίηση και διαχείριση επικοινωνίας για το AXI4 - Lite FIR filter custom IP..... 4
1) AXI4 Write operation..... 4
2) AXI4 Read operation..... 4
3) AXI4 - FIR filter operation and reading input data..... 5
➢ **Προσθήκη AXI4-Lite slave διεπαφή + FIR φίλτρο ως custom IP block στο synthesis design - AXI interconnect με το PS (ARM processor)..... 6**
➢ Υλοποίηση block design συστήματος PS - FPGA logic..... 7
➢ Υλοποίηση firmware για επικοινωνία PS - PL, μέσω Vivado SDK..... 7
➢ **Προσομοίωση λειτουργίας AXI4 διεπαφής + FIR φίλτρου, με αντίστοιχα αποτελέσματα στο terminal..... 10**

➢ Εισαγωγή

Σε αυτήν την εργαστηριακή άσκηση, θα αποτυπώσουμε (synthesis & implementation) το FIR φίλτρο, όπως υλοποιήθηκε στην προηγούμενη αναφορά, στο Programmable Logic (PL, FPGA logic) του Zynq-7000 SoC (ARM dual-core processor + FPGA logic) που υποστηρίζει η πλακέτα Zybo.

Πιο συγκεκριμένα, θα χρειαστεί να εντάξουμε την υλοποίηση του FIR φίλτρου σε VHDL, ως τμήμα σε ένα packaged IP block, ώστε να συνδεθεί το FIR synthesized design από το FPGA logic του SoC, με τον ARM processor (PS), μέσω μιας AXI4-Lite διεπαφής. Η AXI4 διεπαφή είναι απαραίτητη ώστε να επικοινωνεί κατάλληλα το custom IP block του FIR φίλτρου με τον επεξεργαστή ARM του SoC και εν τέλει με τον χρήστη, ανταλλάσσοντας δεδομένα και αποτελέσματα. Το FIR φίλτρο ως custom IP block θα είναι αποτυπωμένο στο PL (Programmable logic) του SoC και θα "καλύπτεται" από την διεπαφή AXI4, καθιστώντας το custom IP block σαν AXI slave device. Έπειτα, το FIR + AXI4 packaged IP block θα συνδέεται μέσω ενός AXI AMBA interconnect block στο διάδρομο δεδομένων που συνδέει το PS (ARM processor) με το PL τμήμα του Zynq SoC.



Έχοντας αποτυπώσει πλήρως το FIR φίλτρο στο PL του Zynq ως AXI4 slave device στο AXI interconnect bus που συνδέει PS - PL, απομένει να υλοποιήσουμε κατάλληλες εντολές (firmware) για να επικοινωνούμε δεδομένα, μέσω του ARM processor, προς το AXI bus για αποστολή στο FIR φίλτρο (master - slave επικοινωνία). Αυτό θα γίνει με την βοήθεια του SDK tool που περιέχεται ήδη

στο Vivado suite, με σκοπό να παραχθεί το κατάλληλο εκτελέσιμο και να φορτωθεί στην Flash μνήμη ώστε οι εντολές να τρέξουν στον ARM processor.

➢ AXI4-Lite slave διεπαφή + FIR φίλτρο ως custom IP block

Αρχικά, ξεκινήσαμε εντάσσοντας το FIR φίλτρο που υλοποιήσαμε στην προηγούμενη άσκηση, σε ένα custom IP block. Ειδικότερα, αφού ξεκινήσαμε να κανουμε package το νέο custom IP, επιλέξαμε να γίνει customized ως AXI4 peripheral. Επιλέγοντας Slave interface mode για το IP και χρήση 4 εσωτερικών καταχωρητών για την αποθήκευση δεδομένων στο peripheral, ετοιμάστηκαν, μέσω του Vivado, έτοιμα wrapper αρχεία σε VHDL με όλες τις πληροφορίες που χρειάζονται για να χρησιμοποιήσουμε και να διαχειριστούμε δεδομένα από/προς το AXI4 slave device που υλοποιούμε.

Στην συνέχεια, προσθέσαμε τα κατάλληλα αρχεία του FIR φίλτρου και εντάξαμε το component αυτού εντός της slave AXI διεπαφής (“fir_filter_ip_v1_0_S00_AXI.vhd”), μαζί με τα κατάλληλα σήματα που χρειάζονται για την χρήση αυτού.

```
    -- FIR filter component to be used inside with AXI4 interconnect bus
    signal reset_fir, valid_in, valid_out: std_logic;
    signal fir_out: std_logic_vector(23 downto 0);
    signal x: std_logic_vector (7 downto 0);

    component FIR_filter_unit
        generic(
            data_width: integer := 8
        );
        Port ( clk : in STD_LOGIC;
               rst : in STD_LOGIC;
               valid_in : in STD_LOGIC;
               x : in STD_LOGIC_VECTOR (data_width-1 downto 0);
               valid_out : out STD_LOGIC;
               y : out STD_LOGIC_VECTOR (23 downto 0));
    end component;

    -- for debugging...
    signal valid_out_counter, valid_in_counter:
std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0) := (others => '0');
```

➢ Υλοποίηση και διαχείριση επικοινωνίας για το AXI4 - Lite FIR filter custom IP

Έπειτα, αφού εντάξαμε το component θα χρειαστεί να χρησιμοποιήσουμε κατάλληλα το hardware που περιγράφει (μέσω VHDL) τα read/write operations στο AXI4 peripheral, με σκοπό να μπορέσουμε να εκτελέσουμε κατάλληλο διάβασμα δεδομένων από τον ARM processor (PS), αλλά και όποτε ζητείται (μέσω του κατάλληλου address) να μπορεί να αποστέλλει πίσω στο ARM processor, τα αποτελέσματα του FIR φίλτρου.
Όσον αφορά την εγγραφή δεδομένων προς το PL μέσω του AXI bus, αποφασίσαμε να λαμβάνουμε τα δεδομένα ως πακέτο στον 32-bit "slv_reg0", ενώ θα εξάγουμε τα αποτελέσματα του FIR αφού τα αποθηκεύσουμε - package (μέχρι να ζητηθούν από το PS στο κατάλληλο address) στον "slv_reg1".

1) AXI4 Write operation

Το τμήμα που αφορά το hardware εγγραφής δεδομένων προς τους registers ("slv_reg{X}") δεν επηρεάζεται από την υλοποίηση που έχουμε και για αυτό δεν αλλάζει καθόλου. Το κομμάτι αυτό ενεργοποιείται στα rising edges του S_AXI_ACLK και εφόσον το σήμα "slv_reg_wren" οδηγηθεί στο λογικό '1'. Έπειτα, αυτόματα το AXI peripheral αναλόγως του "loc_addr" που μεταφέρεται στο bus, μέσω του σήματος S_AXI_AWADDR (write address που γίνεται cast από το PS μέσω του AXI interconnect), θα "καταλάβει" ακριβώς σε ποιον register να εγγράψει τα δεδομένα που κυκλοφορούν στο AXI bus (S_AXI_WDATA).

2) AXI4 Read operation

Αντίστοιχα, για το τμήμα που αφορά το hardware αποστολής δεδομένων προς τον ARM processor (PS) λαμβάνουμε διαφορετική αντιμετώπιση. Πιο συγκεκριμένα, εφόσον το σήμα "valid_out" που δίνει το FIR φίλτρο γίνει λογικό '1' και άρα τελειώσει ο υπολογισμός του FIR, "πακετάρονται" στον 'slv_reg1' τα δεδομένα εξόδου του FIR, στον αμέσως επόμενο κύκλο ρολογιού.

Στην συνέχεια, ασύγχρονα με το S_AXI_ACLK αλλά μόλις το σήμα "slv_reg_rden" που δείχνει ότι ζητείται διάβασμα δεδομένων από το PS στο AXI interconnect bus, γίνει λογικό '1' τα δεδομένα που εντάχθηκαν στο 'slv_reg1' μεταφέρονται στον τελικό register του AXI bus προς αποστολή "reg_data_out".

Στον κύκλο ρολογιού αμέσως μετά και επειδή το σήμα "slv_reg_rden" είναι λογικό '1', το αρχικό τμήμα που έβαλε τα δεδομένα εξαρχής από το FIR στον 'slv_reg1' είναι υπεύθυνο να αποστείλει την πληροφορία από τον "reg_data_out" προς το σήμα axi_rdata, το οποίο τελικά οδηγεί τα δεδομένα προς το AXI bus για αποστολή στο PS.

Οι περιγραφές παραπάνω φαίνονται στα παρακάτω τμήματα σε VHDL εντός του AXI peripheral:

```
    -- Implement memory mapped register select and read logic generation
    -- Slave register read enable is asserted when valid address is available
    -- and the slave is ready to accept the read address.
    slv_reg_rden <= axi_arready and S_AXI_ARVALID and (not axi_rvalid) ;

    process (slv_reg0, slv_reg1, slv_reg2, slv_reg3, axi_araddr, S_AXI_ARESETN,
slv_reg_rden)
    variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
    begin
```

```
        -- Address decoding for reading registers
        loc_addr := axi_araddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
        case loc_addr is
          when b"00" =>
            reg_data_out <= slv_reg0;
          when b"01" =>
            reg_data_out <= slv_reg1;
          when b"10" =>
            reg_data_out <= valid_in_counter;
          when b"11" =>
            reg_data_out <= valid_out_counter;
          when others =>
            reg_data_out  <= (others => '0');
        end case;
    end process;

    -- Output register or memory read data
    process( S_AXI_ACLK ) is
    begin
      if (rising_edge (S_AXI_ACLK)) then
        if ( S_AXI_ARESETN = '0' ) then
          axi_rdata  <= (others => '0');
        else
          if (slv_reg_rden = '1') then
            -- When there is a valid read address (S_AXI_ARVALID) with
            -- acceptance of read address by the slave (axi_arready),
            -- output the read dada
            -- Read address mux
              axi_rdata <= reg_data_out;     -- register read data
              slv_reg1(24) <= '0';
          elsif (valid_out = '1') then       -- Give data through AXI bus to ARM core
only if valid_out is high.
              slv_reg1 <= "0000000" & valid_out & fir_out;
          else
              slv_reg1 <= slv_reg1;
          end if;
        end if;
      end if;
    end process;
```

Ξεκάθαρα, για λόγους debugging μεταφέρουμε ασύγχρονα και στο 1ο process και 2 ειδικούς counter οι οποίοι κάνουν track τις μεταβολές τόσο σε valid_in που δίνουμε αλλά και στα valid_out σήματα που εξάγει το FIR. Οι counter αυτοί μπορούν να διαβαστούν από το PS μόλις ζητηθούν από τους registers: slv_reg2 & slv_reg3.

3) AXI4 - FIR filter operation and reading input data

Τέλος, απομένει να εντάξουμε το component του FIR προς εκτέλεση και να του αναθέσουμε τις κατάλληλες εισόδους μέσω του “slv_reg0” σε κάθε rising edge του `S_AXI_ACLK`. Αυτό φαίνεται στο

παρακάτω τελικό τμήμα της περιγραφής σε VHDL, ενώ παραθέτουμε και την διαχείριση σε processes των debugging counters που αναφέραμε προηγουμένως.

```
-- Add user logic here
FIR: FIR_filter_unit generic map( data_width => 8 )
                     port map ( clk => S_AXI_ACLK, rst => reset_fir,
                                valid_in => valid_in, x => x,
                                valid_out => valid_out, y => fir_out);

input_mode: process (S_AXI_ACLK)
begin

    x <= slv_reg0(7 downto 0);
    valid_in <= slv_reg0(8);
    reset_fir <= slv_reg0(9);

end process;

-- Scan valid_in/out signal changes and track counter
in_count: process (reset_fir, valid_in)
begin
    if reset_fir = '1' then
        valid_in_counter <= (others => '0');
    elsif rising_edge(valid_in) then
        valid_in_counter <= valid_in_counter + 1;
    end if;
end process;

out_count: process (reset_fir, valid_out)
begin
    if reset_fir = '1' then
        valid_out_counter <= (others => '0');
    elsif rising_edge(valid_out) then
        valid_out_counter <= valid_out_counter + 1;
    end if;
end process;

-- User logic ends
```

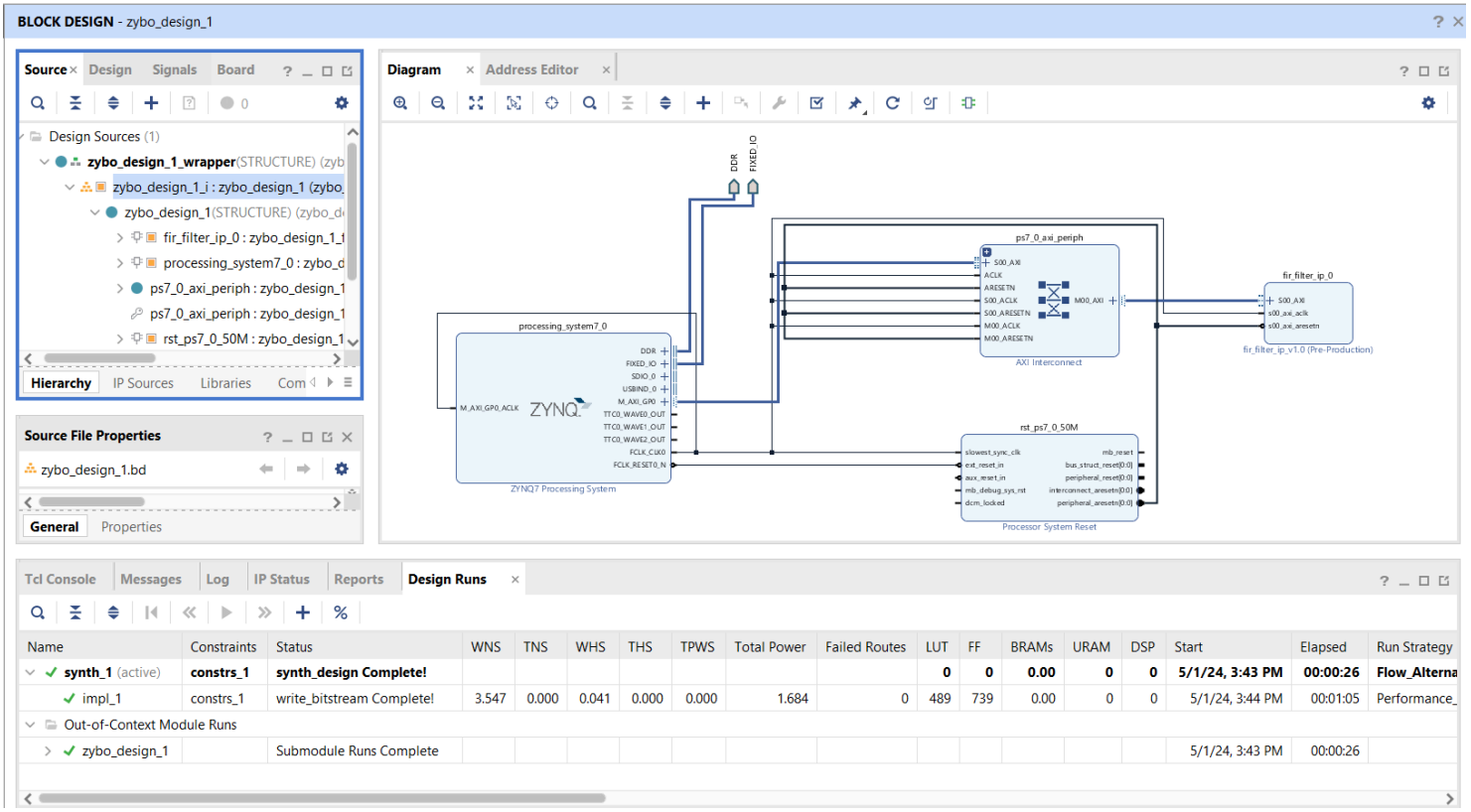
➢ Προσθήκη AXI4-Lite slave διεπαφή + FIR φίλτρο ως custom IP block στο synthesis design - AXI interconnect με το PS (ARM processor)

Στην συνέχεια, αφού κάναμε package το IP block που υλοποιήσαμε ανωτέρω, σε νέο project στο Vivado, εντάξαμε το FIR + AXI4 IP block μαζί με το Zynq PS σε ένα κοινό block design. Μέσω

αυτοματισμών του Vivado suite προστέθηκε αντίστοιχα το απαραίτητο AXI interconnect με σκοπό το AXI bus να διαχειρίζεται κατάλληλα την επικοινωνία με το AXI slave device στο PL.

➢ Υλοποίηση block design συστήματος PS - FPGA logic

Αφού έγιναν τα παραπάνω και ελέγξαμε ότι είναι upgraded η τελική version του custom IP block για το FIR φίλτρο που υλοποιήσαμε ανωτέρω, έχουμε καταλήξει στο παρακάτω block design:

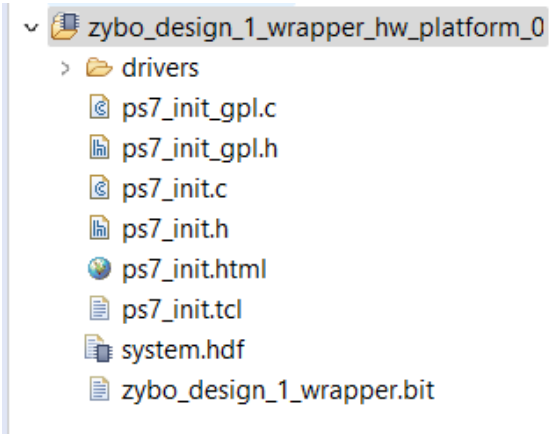


Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy
synth_1 (active)	constrs_1	synth_design Complete!								0	0	0.00	0	0	5/1/24, 3:43 PM	00:00:26	Flow_Alterna
impl_1	constrs_1	write_bitstream Complete!	3.547	0.000	0.041	0.000	0.000	1.684	0	489	739	0.00	0	0	5/1/24, 3:44 PM	00:01:05	Performance_
Out-of-Context Module Runs																	
zybo_design_1		Submodule Runs Complete													5/1/24, 3:43 PM	00:00:26	

Στην συνέχεια το Vivado suite αφού ζητήσαμε να γίνει generate το απαραίτητο bitstream για να φορτώσουμε στο FPGA το απαραίτητο (placed & routed) RTL design που υλοποιήσαμε, το Vivado suite έτρεξε αυτόματα Synthesis & Implementation πάνω στο block design, και δίχως errors (όπως timing issues ή synthesis issues) ετοίμασε το bitstream. Το bitstream αυτό είναι ενδεικτικό των component που χρησιμοποιήθηκαν για το implemented RTL design του ψηφιακών κυκλωμάτων (π.χ AXI4 slave + FIR filter unit) που χρησιμοποιήθηκαν και πως ακριβώς θα “ενωθούν” (placement & routing) συνολικά για την δημιουργία του παραπάνω συστήματος στο PL τμήμα του SoC.

➢ Υλοποίηση firmware για επικοινωνία PS - PL, μέσω Vivado SDK

Τέλος, αφού γίνει export hardware για να δημιουργηθούν τα κατάλληλα tcl, hdf, κ.α αρχεία, μαζί με το bitstream (.bit), θα κάνουμε launch το SDK tool και θα δημιουργήσουμε ένα απλό application project, το οποίο θα φορτωθεί στον ARM processor (εσωτερική flash - Programming) και θα εξετάζει την λειτουργία του FIR φιλτρου πάνω στο FPGA - PL hardware του Zynq SoC.



Παρακάτω, παραθέτουμε τον application κώδικα σε C που υλοποιήσαμε για να ελέγξουμε τη λειτουργία του FIR φίλτρου:

```
/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xbasic_types.h"
#include "xparameters.h"
#include "sleep.h"

Xuint32 *baseaddr_ip = (Xuint32 *)XPAR_FIR_FILTER_IP_0_S00_AXI_BASEADDR;

Xuint8 valid_in, fir_reset, valid_out = 0;
Xuint32 packet, fir_in, fir_out = 0;

int main()
{
    init_platform();

      while (1)
      {
            xil_printf("FIR: Provide input sample x[n]: ");
            scanf("%d", &fir_in);
```

```
            xil_printf("\n\rFIR: x[n] given is: %d \n\r", fir_in);

            xil_printf("FIR: Is input valid? (1/0) : ");
            scanf("%d", &valid_in);

            if (valid_in == 0x01)
               xil_printf("\n\rFIR: Input is valid... \n\r");
            else{
               valid_in = 0;
               xil_printf("\n\rFIR: Input is invalid... \n\r");
            }

            xil_printf("FIR: Do you wish to reset FIR? (1/0) : ");
            scanf("%d", &fir_reset);

            if (fir_reset == 0)
                  xil_printf("\n\rFIR: FIR won't reset... \n\r");
            else{
                  fir_reset = 0x01;
                  xil_printf("\n\rFIR: FIR will now reset... Next print will give
x[n]...\n\r");
            }

            packet = (fir_reset << 9) | (valid_in << 8);
            packet = (packet | fir_in);
            xil_printf("FIR: Data sent to FIR are packaged us: 0x%08x \n\r",
packet);

            // Discard valid_in bit, if reset is given...
            if(fir_reset == 1)
                  *(baseaddr_ip) = (fir_reset << 9) | fir_in;
            else
                  *(baseaddr_ip) = packet;

            // Pulse valid_in signal for some clock cycles, until valid_out is
given...
            if(valid_in == 1){
                  while (valid_out == 0 && fir_reset == 0) {
                        valid_out = (*(baseaddr_ip + 1) &  0x1000000) >> 24;
                        *(baseaddr_ip) = fir_in & 0xFF;
                        usleep(500000);
                  }
            }

            if (fir_reset == 0){
                  xil_printf("\nvalid_out is: 0x%08x \n\r",  valid_out);
                  xil_printf("FIR: slv_reg[] registers contain: \n\r --- x[n] =
0x%08x, y[n] = 0x%08x \n\r --- valid_in_counter: 0x%08x,  valid_out_counter: 0x%08x
\n\r", *(baseaddr_ip), *(baseaddr_ip + 1), *(baseaddr_ip + 2), *(baseaddr_ip + 3));

                  fir_out = (*(baseaddr_ip + 1) & 0xFFFFFF);
```

```
                    xil_printf("FIR: Data y[n] calculated  %d (in bytes: 0x%08x) \n\r",
fir_out, *(baseaddr_ip + 1));
                }


xil_printf("------------------------------------------------------------------------
\n\r");

                valid_out = 0;
                usleep(1000000);    // 1sec delay until new sample x[n] from user...
        }

    cleanup_platform();
    return 0;
}
```

➢ Προσομοίωση λειτουργίας AXI4 διεπαφής + FIR φίλτρου, με αντίστοιχα αποτελέσματα στο terminal

Στις παρακάτω εικόνες από το terminal θα φανει η εκτέλεση του παραπάνω κώδικα μαζί με τις αποκρίσεις που λαμβάνουμε από το PL, δηλαδή τα αποτελέσματα του FIR φίλτρου, ακριβώς όπως αυτά στέλνονται μέσω της AXI slave διεπαφής.

Το πείραμα ελέγχου για το FIR χωρίζεται στα παρακάτω βασικά σημεία για τον έλεγχο της λειτουργίας του:

1) Ξεκινάμε και δίνουμε χαμηλές τιμές x[n] στο FIR φίλτρο για εύκολους υπολογισμούς.

2) Συνεχίζουμε δίνοντας μόνιμα την τιμή 255 (0xFF) που αποτελεί την μέγιστη τιμή που μπορεί να λάβει το φίλτρο. Η τιμή αυτή μόλις δοθεί αρκετές φορές στο φίλτρο και γεμίζει την RAM αυτού, τότε η έξοδος y[n] θα πρέπει να συγκλίνει στην τιμή: 255*36 = 9180.

3) Θα δώσουμε άλλη μια φορά είσοδο 255 με σκοπό να δούμε ξανά ότι το FIR φίλτρο δεν ξεφεύγει από την τιμή σύγκλισης.

4) Στην συνέχεια, θα ελέγξουμε την reset λειτουργία αυτού και θα δώσουμε είσοδο reset ίση με ΄1΄. Πράγματι, για να επιβεβαιώσουμε πως το FIR φίλτρο έχει γίνει πλήρως reset, ξανδίνουμε ως valid είσοδο την τιμή x[n] = 0x01 (decimal: 1) και αναμένουμε να δούμε έξοδο y[n] από το φίλτρο ίση με (1*x[n] + 2*0 + … + 8*0) = y[n]. Αν το reset είχε αποτύχει λόγω λανθασμένης σχεδίασης, τότε η έξοδος που θα άμβάναμε θα ήταν ίση με (1*x[n] + 2*255 + … + 8*255) = y[n] >> x[n].

5) Τέλος, για να επιβεβαιώσουμε και την τελευταία λειτουργία του FIR φίλτρου, θα συνεχίζουμε να δίνουμε μη-μηδενικές τιμές x[n] στο φίλτρο, αλλά με valid_in ίσο με 0 (invalid τιμές). τότε θα

πρέπει σε κάθε διάβασμα του slv_reg1, η τιμή εξόδου του φίλτρου που θα βλέπουμε να μην αυξάνεται αλλά να παραμείνει στο ‘1’, καθώς οι επιπλέον τιμές είναι invalid.

Εξετάζοντας παρακάτω τα βήματα 1-5 παρατηρουμε πράγματι την ορθή συμπεριφορά του FIR φίλτρου, καθώς και την κατάλληλη αποστολή δεδομένων, μέσω της διεπαφής AXI.

- Βήματα 1 εως 4: Διαρκής εκτέλεση FIR με εισόδους x[n]

```
valid_out is: 0x00000001

FIR: slv_reg[] registers contain:

 --- x[n] = 0x00000002, y[n] = 0x00000004

 --- valid_in_counter: 0x00000002,  valid_out_counter: 0x00000002

FIR: Data y[n] calculated  4 (in bytes: 0x00000004)

------------------------------------------------------------------------

FIR: Provide input sample x[n]:

FIR: x[n] given is: 10

FIR: Is input valid? (1/0) :

FIR: Input is valid...

FIR: Do you wish to reset FIR? (1/0) :

FIR: FIR won't reset...

FIR: Data sent to FIR are packaged us: 0x0000010A


valid_out is: 0x00000001

FIR: slv_reg[] registers contain:

 --- x[n] = 0x0000000A, y[n] = 0x00000011

 --- valid_in_counter: 0x00000003,  valid_out_counter: 0x00000003

FIR: Data y[n] calculated  17 (in bytes: 0x00000011)

------------------------------------------------------------------------
```

```
--------------------------------------------------------------------

FIR: Provide input sample x[n]:

FIR: x[n] given is: 255

FIR: Is input valid? (1/0) :

FIR: Input is valid...

FIR: Do you wish to reset FIR? (1/0) :

FIR: FIR won't reset...

FIR: Data sent to FIR are packaged us: 0x000001FF


valid_out is: 0x00000001

FIR: slv_reg[] registers contain:

 --- x[n] = 0x000000FF, y[n] = 0x0000011D

 --- valid_in_counter: 0x00000004,  valid_out_counter: 0x00000004

FIR: Data y[n] calculated  285 (in bytes: 0x0000011D)

--------------------------------------------------------------------

FIR: Provide input sample x[n]:

FIR: x[n] given is: 255

FIR: Is input valid? (1/0) :

FIR: Input is valid...

FIR: Do you wish to reset FIR? (1/0) :

FIR: FIR won't reset...

FIR: Data sent to FIR are packaged us: 0x000001FF
```

```
valid_out is: 0x00000001

FIR: slv_reg[] registers contain:

 --- x[n] = 0x000000FF, y[n] = 0x00000328

 --- valid_in_counter: 0x00000005,  valid_out_counter: 0x00000005

FIR: Data y[n] calculated  808 (in bytes: 0x00000328)

------------------------------------------------------------------------

FIR: Provide input sample x[n]:

FIR: x[n] given is: 255

FIR: Is input valid? (1/0) :

FIR: Input is valid...

FIR: Do you wish to reset FIR? (1/0) :

FIR: FIR won't reset...

FIR: Data sent to FIR are packaged us: 0x000001FF


valid_out is: 0x00000001

FIR: slv_reg[] registers contain:

 --- x[n] = 0x000000FF, y[n] = 0x00000632

 --- valid_in_counter: 0x00000006,  valid_out_counter: 0x00000006

FIR: Data y[n] calculated  1586 (in bytes: 0x00000632)

------------------------------------------------------------------------
```

Connected to: Serial (COM27, 115200, 0, 8)

```
FIR: Provide input sample x[n]:

FIR: x[n] given is: 255

FIR: Is input valid? (1/0) :

FIR: Input is valid...

FIR: Do you wish to reset FIR? (1/0) :

FIR: FIR won't reset...

FIR: Data sent to FIR are packaged us: 0x000001FF


valid_out is: 0x00000001

FIR: slv_reg[] registers contain:

 --- x[n] = 0x000000FF, y[n] = 0x00000A3B

 --- valid_in_counter: 0x00000007,  valid_out_counter: 0x00000007

FIR: Data y[n] calculated  2619 (in bytes: 0x00000A3B)

------------------------------------------------------------------------

FIR: Provide input sample x[n]:

FIR: x[n] given is: 255

FIR: Is input valid? (1/0) :

FIR: Input is valid...

FIR: Do you wish to reset FIR? (1/0) :

FIR: FIR won't reset...

FIR: Data sent to FIR are packaged us: 0x000001FF


valid_out is: 0x00000001
```

```
valid_out is: 0x00000001

FIR: slv_reg[] registers contain:

 --- x[n] = 0x000000FF, y[n] = 0x00000F43

 --- valid_in_counter: 0x00000008,  valid_out_counter: 0x00000008

FIR: Data y[n] calculated  3907 (in bytes: 0x00000F43)

------------------------------------------------------------------------

FIR: Provide input sample x[n]:

FIR: x[n] given is: 255

FIR: Is input valid? (1/0) :

FIR: Input is valid...

FIR: Do you wish to reset FIR? (1/0) :

FIR: FIR won't reset...

FIR: Data sent to FIR are packaged us: 0x000001FF


valid_out is: 0x00000001

FIR: slv_reg[] registers contain:

 --- x[n] = 0x000000FF, y[n] = 0x00001541

 --- valid_in_counter: 0x00000009,  valid_out_counter: 0x00000009

FIR: Data y[n] calculated  5441 (in bytes: 0x00001541)

------------------------------------------------------------------------
```

Connected to: Serial (COM27, 115200, 0, 8)

```
FIR: Provide input sample x[n]:

FIR: x[n] given is: 255

FIR: Is input valid? (1/0) :

FIR: Input is valid...

FIR: Do you wish to reset FIR? (1/0) :

FIR: FIR won't reset...

FIR: Data sent to FIR are packaged us: 0x000001FF


valid_out is: 0x00000001

FIR: slv_reg[] registers contain:

 --- x[n] = 0x000000FF, y[n] = 0x00001C34

 --- valid_in_counter: 0x0000000A,  valid_out_counter: 0x0000000A

FIR: Data y[n] calculated  7220 (in bytes: 0x00001C34)

------------------------------------------------------------------------

FIR: Provide input sample x[n]:

FIR: x[n] given is: 255

FIR: Is input valid? (1/0) :

FIR: Input is valid...

FIR: Do you wish to reset FIR? (1/0) :

FIR: FIR won't reset...

FIR: Data sent to FIR are packaged us: 0x000001FF


valid_out is: 0x00000001
```

Connected to: Serial (COM27, 115200, 0, 8)

```
FIR: Data sent to FIR are packaged us: 0x000001FF


valid_out is: 0x00000001

FIR: slv_reg[] registers contain:

 --- x[n] = 0x000000FF, y[n] = 0x000023DC

 --- valid_in_counter: 0x0000000B,  valid_out_counter: 0x0000000B

FIR: Data y[n] calculated  9180 (in bytes: 0x000023DC)

--------------------------------------------------------------------------

FIR: Provide input sample x[n]:

FIR: x[n] given is: 255

FIR: Is input valid? (1/0) :

FIR: Input is valid...

FIR: Do you wish to reset FIR? (1/0) :

FIR: FIR won't reset...

FIR: Data sent to FIR are packaged us: 0x000001FF


valid_out is: 0x00000001

FIR: slv_reg[] registers contain:

 --- x[n] = 0x000000FF, y[n] = 0x000023DC

 --- valid_in_counter: 0x0000000C,  valid_out_counter: 0x0000000C

FIR: Data y[n] calculated  9180 (in bytes: 0x000023DC)

--------------------------------------------------------------------------
```

- Βήμα 4: Δίνουμε reset signal στο παραπάνω run αφού γίνει σταθεροποίηση δεδομένων στην RAM (όλα 0xFF → 255):

```
FIR: slv_reg[] registers contain:

 --- x[n] = 0x000000FF, y[n] = 0x000023DC

 --- valid_in_counter: 0x0000000C,  valid_out_counter: 0x0000000C

FIR: Data y[n] calculated  9180 (in bytes: 0x000023DC)

------------------------------------------------------------------------

FIR: Provide input sample x[n]:

FIR: x[n] given is: 1

FIR: Is input valid? (1/0) :

FIR: Input is invalid...

FIR: Do you wish to reset FIR? (1/0) :

FIR: FIR will now reset... Next print will give x[n]...

FIR: Data sent to FIR are packaged us: 0x00000201

------------------------------------------------------------------------

FIR: Provide input sample x[n]:

FIR: x[n] given is: 1

FIR: Is input valid? (1/0) :

FIR: Input is valid...

FIR: Do you wish to reset FIR? (1/0) :

FIR: FIR won't reset...

FIR: Data sent to FIR are packaged us: 0x00000101


valid_out is: 0x00000001
```

```
FIR: slv_reg[] registers contain:
 --- x[n] = 0x00000001, y[n] = 0x00000001
 --- valid_in_counter: 0x00000000,  valid_out_counter: 0x00000001
FIR: Data y[n] calculated  1 (in bytes: 0x00000001)
--------------------------------------------------------------------------
```

- Βήμα 5: Ολοκληρώνουμε την προσομοίωση με παροχή 2 επιπλέον invalid εισόδων στο FIR.

```
--------------------------------------------------------------------------
FIR: Provide input sample x[n]:
FIR: x[n] given is: 10
FIR: Is input valid? (1/0) :
FIR: Input is invalid...
FIR: Do you wish to reset FIR? (1/0) :
FIR: FIR won't reset...
FIR: Data sent to FIR are packaged us: 0x0000000A
valid_out is: 0x00000000
FIR: slv_reg[] registers contain:
 --- x[n] = 0x0000000A, y[n] = 0x00000001
 --- valid_in_counter: 0x00000000,  valid_out_counter: 0x00000001
FIR: Data y[n] calculated  1 (in bytes: 0x00000001)
--------------------------------------------------------------------------
FIR: Provide input sample x[n]:
FIR: x[n] given is: 10
FIR: Is input valid? (1/0) :
FIR: Input is invalid...
FIR: Do you wish to reset FIR? (1/0) :
FIR: FIR won't reset...
FIR: Data sent to FIR are packaged us: 0x0000000A
valid_out is: 0x00000000
```

```
FIR: slv_reg[] registers contain:
 --- x[n] = 0x0000000A, y[n] = 0x00000001
 --- valid_in_counter: 0x00000000,  valid_out_counter: 0x00000001
FIR: Data y[n] calculated  1 (in bytes: 0x00000001)
--------------------------------------------------------------------------

FIR: Provide input sample x[n]:
FIR: x[n] given is: 10
FIR: Is input valid? (1/0) :
FIR: Input is invalid...
FIR: Do you wish to reset FIR? (1/0) :
FIR: FIR won't reset...
FIR: Data sent to FIR are packaged us: 0x0000000A
valid_out is: 0x00000000
FIR: slv_reg[] registers contain:
 --- x[n] = 0x0000000A, y[n] = 0x00000001
 --- valid_in_counter: 0x00000000,  valid_out_counter: 0x00000001
FIR: Data y[n] calculated  1 (in bytes: 0x00000001)
--------------------------------------------------------------------------
```