Εθνικό Μετσόβιο Πολυτεχνείο, ΣΗΜΜΥ
Ψηφιακά Συστήματα VLSI

5η Εργαστηριακή Άσκηση “Υλοποίηση FIR φίλτρου με AXI διεπαφή σε ZYNQ SoC FPGA”

Ομάδα 19
Παναγιώτης Μπέλσης, ΑΜ: 03120874
Θεοδώρα Εξάρχου, ΑΜ : 03120865

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

Αρχικά, δημιουργήσαμε 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.
Ουσιαστικά η κύρια αλλαγή ήταν να βάλουμε το φίλτρο fir σαν component και στη συνέχεια να κάνουμε το κατάλληλο mapping τα variables του φίλτρου με τα signals των register που χρησιμοποιούμε.

Επιλέξαμε τους “slv_reg0” για input και τον “slv_reg1” για output

Επιπλέον

• 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.

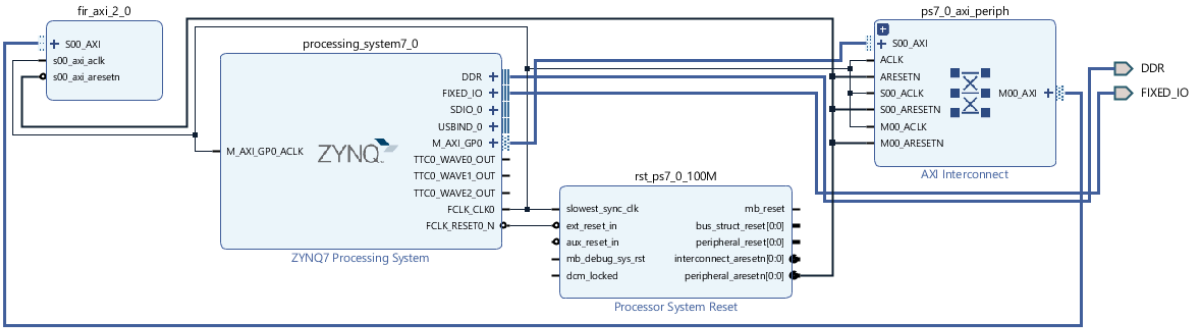
Στο αρχείο fir_axi_2_v1_0_S00_AXI.vhd βλέπουμε αυτά που αναλύσαμε προηγουμένως

```
--           slv_reg1(byte_index*8+7 downto byte_index*8) <= S_AXI_WDATA(byte_index*8+7 downto byte_index*8
          end if;
        end loop;
      when b"10" =>
        for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
          if ( S_AXI_WSTRB(byte_index) = '1' ) then
            -- Respective byte enables are asserted as per write strobes
            -- slave registor 2
            slv_reg2(byte_index*8+7 downto byte_index*8) <= S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
          end if;
        end loop;
      when b"11" =>
        for byte_index in 0 to (C_S_AXI_DATA_WIDTH/8-1) loop
          if ( S_AXI_WSTRB(byte_index) = '1' ) then
            -- Respective byte enables are asserted as per write strobes
            -- slave registor 3
            slv_reg3(byte_index*8+7 downto byte_index*8) <= S_AXI_WDATA(byte_index*8+7 downto byte_index*8);
          end if;
        end loop;
      when others =>
        slv_reg0 <= slv_reg0;
--        slv_reg1 <= slv_reg1;
        slv_reg2 <= slv_reg2;
        slv_reg3 <= slv_reg3;
    end case;
  end if;
 end if;
end if;
end process;
```

```
component fir
port (
clk,valid_in,rst: in std_logic;
x: in std_logic_vector (7 downto 0);
y: out std_logic_vector (18 downto 0);
valid_out: out std_logic
);
end component;
```

```
fir_0 : fir
port map(
clk=> S_AXI_ACLK,
valid_in =>slv_reg0(8) ,
rst =>slv_reg0(9),
x =>slv_reg0(7 downto 0),
y=>slv_reg1(18 downto 0),
valid_out =>slv_reg1(19)
);
    -- User logic ends
```

Το νέο IP για το FIR φίλτρο είναι πλέον έτοιμο και το προσθέτουμε στο αρχικό project fir_zybo που είχαμε δημιουργήσει αρχικά. Βάζουμε στο design το Zynq και το FIR_IP και οι συνδέσεις γίνονται αυτόματα οπότε καταλήγουμε στο τελικό design όπως φαίνεται παρακάτω.



Τέλος το μόνο που απομένει είναι να κάνουμε HDL_wrapper, RTL_analysis και Run_implementation ώστε να σιγουρευτούμε ότι δεν έχουμε κάποιο error.
Τέλος δημιουργούμε το bitstream και ανοίγουμε το SDK αρχείο όπου γράφουμε τον κώδικα του master-software .

Κώδικας του SDK

```
#include <stdio.h>
#include "platform.h"
#include "xil_types.h"
#include "xil_io.h"
#include "sleep.h"

#define XPAR_FIR_AXI_2_0_S00_AXI_BASEADDR 0x43C00000

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;

        // Print input
        xil_printf("A is: %d\n", A);

        // Write to FIR_AXI_2 register
        Xil_Out32((XPAR_FIR_AXI_2_0_S00_AXI_BASEADDR + 0x00), A);

        if (valid_in == 0 || RST == 1) {
            B = Xil_In32(XPAR_FIR_AXI_2_0_S00_AXI_BASEADDR + 0x04);
            valid_out = B & 0x80000;
            xil_printf("\nvalid_out is: %d\n", valid_out);
        } else {
            while (valid_out == 0) {
                B = Xil_In32(XPAR_FIR_AXI_2_0_S00_AXI_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;
}
```

Από το xparametres.h βρήκαμε το Base Address μας

```
#define XPAR_FIR_AXI_2_0_S00_AXI_BASEADDR 0x43C00000
```

Ουσιαστικά με τον παραπάνω κώδικα ζητάμε από τον χρήστη να δώσει τις τιμές rst,valid_in,data_in και στην συνέχεια 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.

Έπειτα 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.

Τελικά αν δεχτούμε μια έγκυρη τιμή valid_in =1 (και reset δεν είναι ενεργοποιημένο), 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. Απομονώνουμε τα data_out και τα τυπώνουμε ώστε να τα δει ο χρήστης μέσω σειριακής επικοινωνίας στο terminal. Στη συνέχεια το πρόγραμμα ζητά από τον χρήστη νέες τιμές εισόδου.