Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

7η Σειρά Ασκήσεων

Μάθημα: Ψηφιακά Συστήματα VLSI

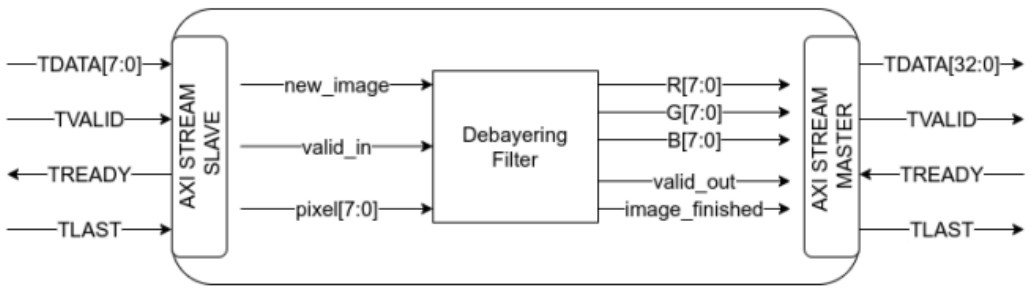
Εξάμηνο: 8ο

Ονοματεπώνυμα: Αλεξοπούλου Γεωργία, Γκενάκου Ζωή

Προγραμματισμός SoC FPGA

Στο Β Μέρος της εργαστηριακής άσκησης κληθήκαμε να προγραμματίσουμε την πλακέτα ZYBO, ώστε να υλοποιεί ένα Debayering φίλτρο, στο οποίο τα δεδομένα εισόδου θα αποστέλλονται από τον ενσωματωμένο επεξεργαστή (ARM) προς το FPGA για επεξεργασία, και αντίστροφα για τα αντίστοιχα αποτελέσματα. Καλούμαστε να συνδέσουμε το Debayering φίλτρο που έχουμε ήδη υλοποιήσει στο Μέρος Α της άσκησης με AXI4-Stream διεπαφές (AXI-Stream Slave/Master Interface). Η διασύνδεση των σημάτων εισόδου και εξόδου του φίλτρου με την AXI διεπαφή να υλοποιηθεί όπως φαίνεται παρακάτω:

- Slave_TDATA[7:0] = pixel[7:0]
- Slave_TVALID = valid_in
- Master_TDATA[32:0] = “00000000” & R[7:0] & G[7:0] & B[7:0]
- Master_TVALID = valid_out
- Master_TLAST = image_finished



Επιπλέον αναπτύσσουμε την ανάλογη εφαρμογή λογισμικού για την αποστολή των σημάτων εισόδου και την λήψη των σημάτων εξόδου του φίλτρου από τον ενσωματωμένο επεξεργαστή. Η επικοινωνία μεταξύ PS-PL θα υλοποιείται μέσω Direct Memory Access (DMA) λογικής. Η εφαρμογή λογισμικού θα είναι υπεύθυνη για την προετοιμασία του DMA ώστε να στέλνει τα pixel προς το φίλτρο και να δέχεται τα αποτελέσματα. Επίσης, η εφαρμογή λογισμικού υλοποιεί και reference software το οποίο θα υπολογίζει και αυτό τα αποτελέσματα του Debayering φίλτρου και θα τα συγκρίνει με τα αποτελέσματα που θα λαμβάνονται από το FPGA.

Για την διευκόλυνση μας θα βασιστούμε στο project dvlsi2021_lab5 που παρέχεται από το εργαστήριο. Αφού κατεβάσουμε το αρχείο και κάνουμε export τα αρχεία του φακέλου, ανοίγουμε το Vivado. Στο αρχικό παράθυρο επιλέγουμε: Tools → Run Tcl Script. Στο παράθυρο που θα ανοίξει πηγαίνουμε στο φάκελο που έχουμε κάνει export και μέσα στο φάκελο scripts (πχ. /Desktop/dvlsi2021_lab5/scripts) επιλέγουμε το αρχείο dvlsi2021_lab5_prj.tcl. Πατάμε OK. Το Vivado αυτόματα θα δημιουργήσει ένα καινούργιο project για να υλοποιήσετε την άσκηση. Από αυτό το σημείο και μετά προσθέτουμε τα δικά μας αρχεία.

Αρχικά, προσθέτουμε το αρχείο με την υλοποίηση σου Debayering φίλτρου (Add Sources). Στο project υπάρχουν τα αρχεία dvlsi2021_lab5_top και design_1_wrapper. Εμείς θα τροποποιήσουμε τον κώδικα του dvlsi2021_lab5_top αρχείου.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity dvlsi2021_lab5_top is
  port (
        DDR_cas_n         : inout STD_LOGIC;
        DDR_cke           : inout STD_LOGIC;
        DDR_ck_n          : inout STD_LOGIC;
        DDR_ck_p          : inout STD_LOGIC;
        DDR_cs_n          : inout STD_LOGIC;
        DDR_reset_n       : inout STD_LOGIC;
        DDR_odt           : inout STD_LOGIC;
        DDR_ras_n         : inout STD_LOGIC;
        DDR_we_n          : inout STD_LOGIC;
        DDR_ba            : inout STD_LOGIC_VECTOR( 2 downto 0);
        DDR_addr          : inout STD_LOGIC_VECTOR(14 downto 0);
        DDR_dm            : inout STD_LOGIC_VECTOR( 3 downto 0);
        DDR_dq            : inout STD_LOGIC_VECTOR(31 downto 0);
        DDR_dqs_n         : inout STD_LOGIC_VECTOR( 3 downto 0);
```

```
        DDR_dqs_p         : inout STD_LOGIC_VECTOR( 3 downto 0);
        FIXED_IO_mio      : inout STD_LOGIC_VECTOR(53 downto 0);
        FIXED_IO_ddr_vrn  : inout STD_LOGIC;
        FIXED_IO_ddr_vrp  : inout STD_LOGIC;
        FIXED_IO_ps_srstb : inout STD_LOGIC;
        FIXED_IO_ps_clk   : inout STD_LOGIC;
        FIXED_IO_ps_porb  : inout STD_LOGIC
       );
end entity; -- dvlsi2021_lab5_top

architecture arch of dvlsi2021_lab5_top is

  constant N : integer := 1024;

  component design_1_wrapper is
    port (
          DDR_cas_n         : inout STD_LOGIC;
          DDR_cke           : inout STD_LOGIC;
          DDR_ck_n          : inout STD_LOGIC;
          DDR_ck_p          : inout STD_LOGIC;
          DDR_cs_n          : inout STD_LOGIC;
          DDR_reset_n       : inout STD_LOGIC;
          DDR_odt           : inout STD_LOGIC;
          DDR_ras_n         : inout STD_LOGIC;
          DDR_we_n          : inout STD_LOGIC;
          DDR_ba            : inout STD_LOGIC_VECTOR( 2 downto 0);
          DDR_addr          : inout STD_LOGIC_VECTOR(14 downto 0);
          DDR_dm            : inout STD_LOGIC_VECTOR( 3 downto 0);
          DDR_dq            : inout STD_LOGIC_VECTOR(31 downto 0);
          DDR_dqs_n         : inout STD_LOGIC_VECTOR( 3 downto 0);
          DDR_dqs_p         : inout STD_LOGIC_VECTOR( 3 downto 0);
          FIXED_IO_mio      : inout STD_LOGIC_VECTOR(53 downto 0);
          FIXED_IO_ddr_vrn  : inout STD_LOGIC;
          FIXED_IO_ddr_vrp  : inout STD_LOGIC;
          FIXED_IO_ps_srstb : inout STD_LOGIC;
          FIXED_IO_ps_clk   : inout STD_LOGIC;
          FIXED_IO_ps_porb  : inout STD_LOGIC;

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

```
-----
          ---------------------------------------------- PL (FPGA)
COMMON INTERFACE
          ACLK                                    : out STD_LOGIC;
          ARESETN                                 : out STD_LOGIC;

-------------------------------------------------------------------------
---------------
          -- PS2PL-DMA AXI4-STREAM MASTER INTERFACE TO ACCELERATOR
AXI4-STREAM SLAVE INTERFACE
          M_AXIS_TO_ACCELERATOR_tdata         : out
STD_LOGIC_VECTOR(7 downto 0);
          --M_AXIS_TO_ACCELERATOR_tkeep         : out
STD_LOGIC_VECTOR( 0    to 0);
          M_AXIS_TO_ACCELERATOR_tlast         : out STD_LOGIC;
          M_AXIS_TO_ACCELERATOR_tready        : in  STD_LOGIC;
          M_AXIS_TO_ACCELERATOR_tvalid        : out STD_LOGIC;

-------------------------------------------------------------------------
---------------
          -- ACCELERATOR AXI4-STREAM MASTER INTERFACE TO PL2P2-DMA
AXI4-STREAM SLAVE INTERFACE
          S_AXIS_S2MM_FROM_ACCELERATOR_tdata  : in
STD_LOGIC_VECTOR(31 downto 0);
          --S_AXIS_S2MM_FROM_ACCELERATOR_tkeep  : in
STD_LOGIC_VECTOR( 3 downto 0);
          S_AXIS_S2MM_FROM_ACCELERATOR_tlast  : in  STD_LOGIC;
          S_AXIS_S2MM_FROM_ACCELERATOR_tready : out STD_LOGIC;
          S_AXIS_S2MM_FROM_ACCELERATOR_tvalid : in  STD_LOGIC
         );
  end component design_1_wrapper;

  component debayering is
    generic (N: integer := 1024);
    Port (
          clock : in std_logic;
          reset : in std_logic;
          valid_in : in std_logic;
          pixel : in std_logic_vector(7 downto 0);
```

```
            new_image : in std_logic;
            R : out std_logic_vector(7 downto 0);
            G : out std_logic_vector(7 downto 0);
            B : out std_logic_vector(7 downto 0);
            valid_out : out std_logic;
            image_finished : out std_logic
        );
end component;

-----------------------------------------
-- INTERNAL SIGNAL & COMPONENTS DECLARATION

  signal aclk    : std_logic;
  signal aresetn : std_logic;

  signal tmp_tdata  : std_logic_vector(7 downto 0);
  signal R : std_logic_vector(7 downto 0);
  signal G : std_logic_vector(7 downto 0);
  signal B : std_logic_vector(7 downto 0);
  signal tmp_tdata_32 : std_logic_vector(31 downto 0);
  signal tmp_tlast_ps_to_pl : std_logic;
  signal tmp_tlast_pl_to_ps : std_logic;
  signal tmp_tready_s : std_logic;
  signal tmp_tready_m : std_logic;
  signal tmp_tvalid_in : std_logic;
  signal tmp_tvalid_out : std_logic;
  signal tmp_tkeep  : std_logic_vector(0 downto 0);
  signal tmp_tlast  : std_logic;
  signal tmp_tready : std_logic;
  signal tmp_tvalid : std_logic;
  signal count : std_logic_vector(3 downto 0) := (others => '0');
  signal new_image : std_logic;

begin

  tmp_tready_s <= tmp_tready_m and tmp_tvalid_in when tmp_tvalid_out
= '1' else tmp_tvalid_in;
  tmp_tdata_32 <= "00000000" & R & G & B;
```

```
  PROCESSING_SYSTEM_INSTANCE : design_1_wrapper
    port map (
               DDR_cas_n         => DDR_cas_n,
               DDR_cke           => DDR_cke,
               DDR_ck_n          => DDR_ck_n,
               DDR_ck_p          => DDR_ck_p,
               DDR_cs_n          => DDR_cs_n,
               DDR_reset_n       => DDR_reset_n,
               DDR_odt           => DDR_odt,
               DDR_ras_n         => DDR_ras_n,
               DDR_we_n          => DDR_we_n,
               DDR_ba            => DDR_ba,
               DDR_addr          => DDR_addr,
               DDR_dm            => DDR_dm,
               DDR_dq            => DDR_dq,
               DDR_dqs_n         => DDR_dqs_n,
               DDR_dqs_p         => DDR_dqs_p,
               FIXED_IO_mio      => FIXED_IO_mio,
               FIXED_IO_ddr_vrn  => FIXED_IO_ddr_vrn,
               FIXED_IO_ddr_vrp  => FIXED_IO_ddr_vrp,
               FIXED_IO_ps_srstb => FIXED_IO_ps_srstb,
               FIXED_IO_ps_clk   => FIXED_IO_ps_clk,
               FIXED_IO_ps_porb  => FIXED_IO_ps_porb,

-------------------------------------------------------------------
-----
               ------------------------------------------------ PL
(FPGA) COMMON INTERFACE
               ACLK                                => aclk,    --
clock to accelerator
               ARESETN                             => aresetn, --
reset to accelerator, active low

-------------------------------------------------------------------
---------------
               -- PS2PL-DMA AXI4-STREAM MASTER INTERFACE TO
ACCELERATOR AXI4-STREAM SLAVE INTERFACE
               M_AXIS_TO_ACCELERATOR_tdata         => tmp_tdata,
               --M_AXIS_TO_ACCELERATOR_tkeep         => tmp_tkeep,
```

```
              M_AXIS_TO_ACCELERATOR_tlast         =>
tmp_tlast_ps_to_pl,
              M_AXIS_TO_ACCELERATOR_tready        => tmp_tready_s,
              M_AXIS_TO_ACCELERATOR_tvalid        => tmp_tvalid_in,

-------------------------------------------------------------------
---------------
              -- ACCELERATOR AXI4-STREAM MASTER INTERFACE TO
PL2P2-DMA AXI4-STREAM SLAVE INTERFACE
              S_AXIS_S2MM_FROM_ACCELERATOR_tdata  => tmp_tdata_32,
              --S_AXIS_S2MM_FROM_ACCELERATOR_tkeep  => tmp_tkeep &
tmp_tkeep & tmp_tkeep & tmp_tkeep,
              S_AXIS_S2MM_FROM_ACCELERATOR_tlast  =>
tmp_tlast_pl_to_ps,
              S_AXIS_S2MM_FROM_ACCELERATOR_tready => tmp_tready_m,
              S_AXIS_S2MM_FROM_ACCELERATOR_tvalid => tmp_tvalid_out
             );

ACCELERATOR : debayering
    port map(
        clock => aclk,
        reset => aresetn,
        new_image => new_image,
        valid_in => tmp_tvalid_in,
        pixel => tmp_tdata,
        R => R,
        G => G,
        B => B,
        valid_out => tmp_tvalid_out,
        image_finished => tmp_tlast_pl_to_ps
    );
----------------------------
-- COMPONENTS INSTANTIATIONS

process(aclk)
begin
    if (aclk'event and aclk = '1') then
        new_image <= tmp_tlast_ps_to_pl;
    end if;
```

```
end process;
end architecture; -- arch
```

Στον κώδικα αφού αρχικοποιηθεί το `entity dvlsi2021_lab5_top,` ορίζεται η αρχιτεκτονική που ορίζει όλα τα εσωτερικά σήματα και τα components. Συγκεκριμένα ορίζεται το component του `design_1_wrapper` το οποίο είναι το interface για το FPGA design και περιέχει σήματα για την μεταφορά δεδομένων και το component του `debayering.`

Επίσης αρχικοποιούνται σήματα που θα μας βοηθήσουν για την υλοποίηση μας. Τα σήματα `tmp_tdata, R, G, B` αντιπροσωπεύουν τα data buses για την μεταφορά των pixel και των χρωματικών συνιστωσών. Το σήμα `tmp_tdata_32` είναι το 32-bit data bus που συνδυάζουμε τα R, G και B. Τα σήματα `tmp_tlast_ps_to_pl` και `tmp_tlast_pl_to_ps` είναι σήματα ελέγχου που σηματοδοτούν το τέλος μίας μεταφοράς δεδομένων.

Στην συνέχεια τα block `PROCESSING_SYSTEM_INSTANCE` και `ACCELERATOR` κάνουν map τα σήματα του `dvlsi2021_lab5_top` με τα σήματα των components.

Αυτό που υλοποιούμε στον κώδικα είναι από την μεριά του PS2PL να βεβαιωθούμε ότι όταν δεχόμαστε `TVALID` θα πρέπει να παράξουμε αντίστοιχο `TREADY` προκειμένου να λάβουμε τα απαραίτητα δεδομένα. Επομένως, κάνουμε το `TREADY` map στο `valid_in` όσο δεν έχουμε `valid_out`. Επιπλέον, πρέπει να ληφθεί υπόψιν και το σήμα `TREADY` του master έτσι ώστε αν αυτό γίνει μηδέν, να γίνει μηδέν και το `TREADY` του slave, προκειμένου να μην δεχθούμε άλλα δεδομένα και το Debayering να "παγώσει" μέχρι το PL2PS να είναι έτοιμο να ξαναδεχθεί δεδομένα.

Από την μεριά του PL2PS παράγουμε το 32-bit σήμα `TDATA` με τα δεδομένα από το Debayering ως R (8 bit) & G (8 bit) & B (8 bit) και 8 μηδενικά στην αρχή για να συμπληρωθούν τα 32bit, το σήμα `TLAST` δηλαδή το image_finished με το οποίο σηματοδοτείται η λήξη της λήψης των δεδομένων και το σήμα `TVALID` δηλαδή το αντίστοιχο `valid_out` του Debayering.

Αφού αποθηκεύσουμε τον κώδικα μας, μπορούμε κατευθείαν να κάνουμε Generate Bitstream καθώς το Block Design είναι ήδη υλοποιημένο.

Αφού ολοκληρωθεί αυτό το κομμάτι εκτελούμε τα βήματα File → Export → Export Hardware και τέλος File → Launch SDK και ανοίγει αυτόματα το SDK.

Ακολουθούμε τα εξής βήματα, πάλι: File → New → Application Project, ονομάζουμε το πρόγραμμα μας Next → Hello World για να δημιουργηθεί αυτόματα το .c file μέσα στον φάκελο src που θα τροποποιήσουμε. Ο κώδικας του SDK φαίνεται παρακάτω:

//

Ο κώδικας αυτός είναι υπεύθυνος για την μεταφορά και την επεξεργασία δεδομένων μεταξύ του PL και του PS. Αρχικά βάζουμε τα header files και τα definitions που θα χρειαστούμε. Τα definitions αφορούν τα ID του PS-to-PL και PL-to-PS DMA controllers, τα base addresses του buffer για την λήψη και μετάδοση των δεδομένων, το μέγεθος N όπου είναι οι διαστάση της εικόνας μας και το IMG_LEN, ο συνολικός αριθμός των pixel δηλαδή $N \times N$.

Στο main κομμάτι του κώδικα, ρυθμίζονται τα DMA transactions τόσο για το TX όσο και για το RX. Αυτό περιλαμβάνει τη διαμόρφωση των DMA controllers και την έναρξη μεταφοράς δεδομένων. Τα δεδομένα εισόδου (pixel) μεταφέρονται από το PS στο PL για επεξεργασία. Τα επεξεργασμένα δεδομένα στη συνέχεια μεταφέρονται πίσω από το PL στο PS.

Το FPGA εκτελεί λειτουργίες επεξεργασίας εικόνας στα λαμβανόμενα δεδομένα, ενώ το λογισμικό εκτελεί τις ίδιες λειτουργίες σε ξεχωριστό αντίγραφο των δεδομένων και στην συνέχεια ο κώδικας συγκρίνει τα αποτελέσματα που λαμβάνονται από το FPGA και τις εφαρμογές λογισμικού. Υπολογίζει το ποσοστό σφάλματος μεταξύ των δύο αποτελεσμάτων και μετρά τους χρόνους εκτέλεσης και των δύο υλοποιήσεων για να αξιολογήσει την επιτάχυνση που επιτυγχάνεται από το FPGA.

Τέλος, ο κώδικας απελευθερώνει πόρους μνήμης και καθαρίζει την πλατφόρμα.