ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΙΣΤΩΝ

ΨΗΦΙΑΚΑ VLSI – 8ο ΕΞΑΜΗΝΟ

Μαρκέτος Νικόδημος – ΑΜ: 03117095

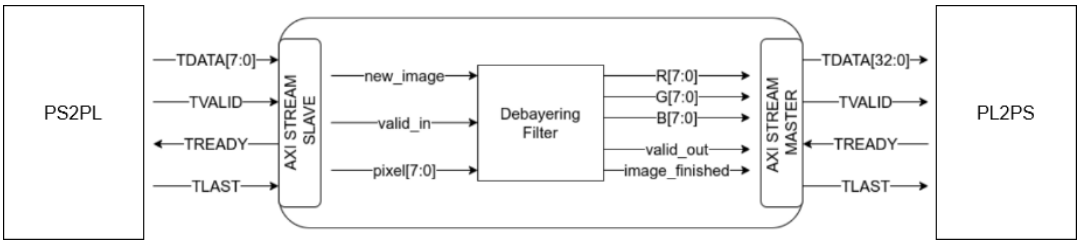
Τζομάκα Αφροδίτη – ΑΜ: 03117107

Ομάδα Β2

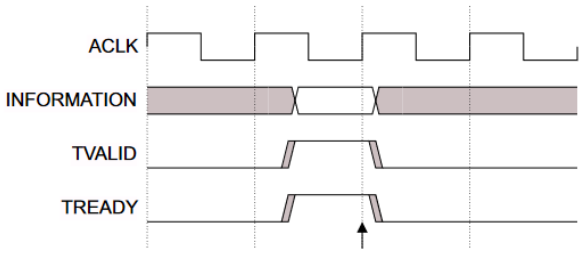
Υλοποίηση Debayering φίλτρου με AXI διεπαφή σε Zynq SoC FPGA

Ζητούμενο της παρούσας εργασίας ήταν ο προγραμματισμός της αναπτυξιακής πλακέτας ZYBO, ώστε να υλοποιεί το Debayering φίλτρο της προηγούμενης εργασίας, στο οποίο τα δεδομένα εισόδου θα αποστέλλονται από τον ενσωματωμένο επεξεργαστή προς το FPGA με DMA λογική.

Η επικοινωνία επεξεργαστή (PS) και FPGA (PL) βασίζεται στο πρωτόκολλο AXI και η αρχιτεκτονική είναι η ακόλουθη:



Πιο συγκεκριμένα για να επικοινωνήσουν τα δύο DMA engines με την AXI4-Stream διεπαφή του Debayering χρειάζεται να λάβει χώρα το handshake μεταξύ TVALID και TREADY:



Στη συνέχεια τα δεδομένα στέλνονται στο Debayering φίλτρο, εκείνο τα επεξεργάζεται και τα προωθεί στο PL2PS DMA engine με την προϋπόθεση ότι συμβαίνει και εκεί το παραπάνω handshake.

Εκείνα που είχαμε να υλοποιήσουμε εμείς ήταν:

- από την μεριά του PS2PL να βεβαιωθούμε ότι όταν δεχόμαστε TVALID θα πρέπει να παράξουμε αντίστοιχο TREADY προκειμένου να μπορέσουμε να λάβουμε τα δεδομένα. Επομένως, (όσον αφορα στον κώδικα) το TREADY γίνεται mapped στο valid_in όσο δεν

έχουμε valid_out. Στην αντίθετη περίπτωση, 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 να παράγουμε το 32-bit σήμα TDATA με τα δεδομένα από το Debayering ως '00000000'&R&G&B, το σήμα TLAST δηλαδή το image_finished με το οποίο σηματοδοτείται η λήξη της λήψης δεδομένων και το σήμα TVALID δηλάδη το αντίστοιχο valid_out του Debayering. Θα πρέπει επίσης να ελέγχουμε το σήμα TREADY έτσι ώστε σε περίπτωση που αυτό έρθει 0 να ειδοποιούμε το κύκλωμα ώστε να κάνει stall. Γι' αυτό και το TREADY αυτό γίνεται mapped στο σήμα halt του Debayering που υλοποιεί την λειτουργία αυτή.

Ο κώδικας για το hardware φαίνεται παρακάτω:

dvlsi2021_lab5_top.bit

```
library ieee;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
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_VECTOR(0 to 0);
          ------------------------------------------------------------------------------------
          -- 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
      generic( N: integer := 1024 );
      port (
          clk :       IN STD_LOGIC;
          rst_n :     IN STD_LOGIC_vector(0 downto 0);
          new_image : IN STD_LOGIC;
          valid_in :  IN STD_LOGIC;
          halt :      IN STD_LOGIC;
          pixel :     IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          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_vector(0 to 0);

  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_ps2pl  : std_logic;
  signal tmp_tlast_pl2ps  : 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 valid: std_logic;
  signal cnt: 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_ps2pl,
            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_4,
            S_AXIS_S2MM_FROM_ACCELERATOR_tlast  => tmp_tlast_pl2ps,
            S_AXIS_S2MM_FROM_ACCELERATOR_tready => tmp_tready_m,
            S_AXIS_S2MM_FROM_ACCELERATOR_tvalid => tmp_tvalid_out
           );
    ACCELERATOR :DEBAYERING
         port map(
            clk => aclk,
            rst_n => aresetn,
            new_image => new_image,
            valid_in => tmp_tvalid_in,
            halt => tmp_tready_m,
            pixel => tmp_tdata,
            R => R,
            G => G,
            B => B,
            valid_out => tmp_tvalid_out,
            image_finished => tmp_tlast_pl2ps
           );
----------------------------
-- COMPONENTS INSTANTIATIONS

process(aclk)
begin
    if rising_edge(aclk) then
        new_image <= tmp_tlast_ps2pl;
    end if;
end process;
end architecture; -- arch
```

Από την μεριά του Software, γράφουμε σε C πρόγραμμα με το οποίο θα στέλνουμε δεδομένα στο PS ώστε μέσα από το DMA να τα προωθήσει στο PL. Χρησιμοποιούμε την λογική του παραδείγματος που δίνεται για απλή μεταφορά πακέτων σε polling μορφή. Ο κώδικας δίνεται παρακάτω:

helloworld.c

```
#include <stdio.h>
#include <unistd.h>
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"
#include "xparameters_ps.h"
#include "xaxidma.h"
#include "xtime_l.h"
#include "image.h"

#define TX_DMA_ID                XPAR_PS2PL_DMA_DEVICE_ID       //PS2PL
#define TX_DMA_MM2S_LENGTH_ADDR  (XPAR_PS2PL_DMA_BASEADDR + 0x28) // Reports actual number of bytes transferred from
PS->PL (use Xil_In32 for report)

#define RX_DMA_ID                XPAR_PL2PS_DMA_DEVICE_ID       //PL2PS
#define RX_DMA_S2MM_LENGTH_ADDR  (XPAR_PL2PS_DMA_BASEADDR + 0x58) // Reports actual number of bytes transferred from
PL->PS (use Xil_In32 for report)

#define TX_BUFFER (XPAR_DDR_MEM_BASEADDR + 0x10000000) // 0 + 256MByte
#define RX_BUFFER (XPAR_DDR_MEM_BASEADDR + 0x18000000) // 0 + 384MByte

/* User application global variables & defines */
#define N                    0x400
#define MAX_PKT_LEN 0x100000 //(N*N)

/* Device instance definitions*/

XAxiDma TXAxiDma;  //PS2PL DMA Engine
XAxiDma RXAxiDma;  //PL2PS DMA Engine

int main()
{
         Xil_DCacheDisable();

         XTime preExecCyclesFPGA = 0;
         XTime postExecCyclesFPGA = 0;
         XTime preExecCyclesSW = 0;
         XTime postExecCyclesSW = 0;

         print("HELLO 1\r\n");
         // User application local variables

         init_platform();

         /************************DEVICE INITIALIZATION*****************************/

         u8 *TxBufferPtr;
         u32 *RxBufferPtr;
         int Status;

         TxBufferPtr = (u8 *)TX_BUFFER;                //8bit data to accelerator
         RxBufferPtr = (u32 *)RX_BUFFER;               //32bit data from accelerator

         // Step 1: Initialize TX-DMA Device (PS->PL)

         /* Configure & Initialize PS2PL */
         XAxiDma_Config *TXCfgPtr;
         int TXStatus;

         TXCfgPtr = XAxiDma_LookupConfig(TX_DMA_ID);
         if (!TXCfgPtr) {
                  xil_printf("No config found for %d\r\n", TX_DMA_ID);
                  return XST_FAILURE;
         }
         TXStatus = XAxiDma_CfgInitialize(&TXAxiDma, TXCfgPtr);
         if (TXStatus != XST_SUCCESS) {
                  xil_printf("Initialization failed %d\r\n", TXStatus);
                  return XST_FAILURE;
         }

         XAxiDma_IntrDisable(&TXAxiDma, XAXIDMA_IRQ_ALL_MASK,    // Disable interrupts, we use polling mode
                                                  XAXIDMA_DEVICE_TO_DMA);
         XAxiDma_IntrDisable(&TXAxiDma, XAXIDMA_IRQ_ALL_MASK,
                                                  XAXIDMA_DMA_TO_DEVICE);

    // Step 2: Initialize RX-DMA Device (PL->PS)
```

```
	/* Configure & Initialize PS2PL */
	XAxiDma_Config *RXCfgPtr;
	int RXStatus;

	RXCfgPtr = XAxiDma_LookupConfig(RX_DMA_ID);
	if (!RXCfgPtr) {
		xil_printf("No config found for %d\r\n", RX_DMA_ID);
		return XST_FAILURE;
	}
	RXStatus = XAxiDma_CfgInitialize(&RXAxiDma, RXCfgPtr);
	if (RXStatus != XST_SUCCESS) {
		xil_printf("Initialization failed %d\r\n", RXStatus);
		return XST_FAILURE;
	}
	XAxiDma_IntrDisable(&RXAxiDma, XAXIDMA_IRQ_ALL_MASK,    // Disable interrupts, we use polling mode
						XAXIDMA_DEVICE_TO_DMA);
	XAxiDma_IntrDisable(&RXAxiDma, XAXIDMA_IRQ_ALL_MASK,
						XAXIDMA_DMA_TO_DEVICE);

	print("HELLO 2\r\n");

	for(int Index = 0; Index < MAX_PKT_LEN; Index ++) {              //Data to accelerator
		TxBufferPtr[Index] = pixels[Index];
	}

	for(int Index = 0; Index < 10; Index ++) {
			RxBufferPtr[Index] = Index;
		}

/*****************************************************************************/
	print("HELLO 3\r\n");

    XTime_GetTime(&preExecCyclesFPGA);
    int sent, received = 0;
    for(int i = 0; i < 10; i++) printf("%lu \n",(unsigned long)RxBufferPtr[i]);

    // Step 3 : Perform FPGA processing
    //      3a: Setup RX-DMA transaction
    //      3b: Setup TX-DMA transaction
    //      3c: Wait for TX-DMA & RX-DMA to finish
	//for(int Times = 0; Times < 4096; Times ++) {

		Status = XAxiDma_SimpleTransfer(&RXAxiDma,(UINTPTR) RxBufferPtr,
				MAX_PKT_LEN*4, XAXIDMA_DEVICE_TO_DMA);

		if (Status != XST_SUCCESS) {
			return XST_FAILURE;
		}

		print("RX OK\r\n");

		Status = XAxiDma_SimpleTransfer(&TXAxiDma,(UINTPTR) TxBufferPtr,
				MAX_PKT_LEN, XAXIDMA_DMA_TO_DEVICE);

		if (Status != XST_SUCCESS) {
			return XST_FAILURE;
		}

		print("TX OK\r\n");

		/*while ((XAxiDma_Busy(&RXAxiDma,XAXIDMA_DEVICE_TO_DMA)) ||
(XAxiDma_Busy(&RXAxiDma,XAXIDMA_DMA_TO_DEVICE)) ||
				(XAxiDma_Busy(&TXAxiDma,XAXIDMA_DMA_TO_DEVICE)) ||
(XAxiDma_Busy(&TXAxiDma,XAXIDMA_DEVICE_TO_DMA)))
		{
				// Wait
		}*/
	//}
		sleep(5);
		sent = Xil_In32(TX_DMA_MM2S_LENGTH_ADDR);
		received = Xil_In32(RX_DMA_S2MM_LENGTH_ADDR);
		printf("Sent = %d, Received = %d \n", sent, received);
		if (sent == N*N && received == N*N*4)
		{
			print("BINGO!\r\n");
		}


		if (XAxiDma_Busy(&RXAxiDma,XAXIDMA_DEVICE_TO_DMA)) print("RX DEVICE TO DMA BUSY\r\n");
		if (XAxiDma_Busy(&RXAxiDma,XAXIDMA_DMA_TO_DEVICE)) print("RX DMA TO DEVICE BUSY\r\n");
		if (XAxiDma_Busy(&TXAxiDma,XAXIDMA_DEVICE_TO_DMA)) print("TX DEVICE TO DMA BUSY\r\n");
		if (XAxiDma_Busy(&TXAxiDma,XAXIDMA_DMA_TO_DEVICE)) print("TX DMA TO DEVICE BUSY\r\n");
```

```
                for(int i = 0; i < 10; i++) printf("%lu \n",(unsigned long)RxBufferPtr[i]);


    XTime_GetTime(&postExecCyclesFPGA);

    /*****************************************************************************/

    print("HELLO 4\r\n");
    XTime_GetTime(&preExecCyclesSW);

    // Step 5: Perform SW processing

    int *r_sw = (int *)malloc(N*N*sizeof(int));
    int *g_sw = (int *)malloc(N*N*sizeof(int));
    int *b_sw = (int *)malloc(N*N*sizeof(int));
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            int right = (j < N - 1) ? (j + 1) : 0;
            int left = (j > 0) ? (j - 1) : 0;
            int up = (i > 0) ? (i - 1) : 0;
            int down = (i < N - 1) ? (i + 1) : 0;

            int r = (j < N - 1) ? pixels[right + N * i] : 0;
            int l = (j > 0) ? pixels[i * N + left] : 0;
            int u = (i > 0) ? pixels[up * N + j] : 0;
            int d = (i < N - 1) ? pixels[down * N + j] : 0;

            int d1 = ((i > 0) && (j > 0)) ? pixels[up * N + left] : 0;
            int d2 = ((i > 0) && j < (N - 1)) ? pixels[up * N + right] : 0;
            int d3 = ((i < N - 1) && (j > 0)) ? pixels[down * N + left] : 0;
            int d4 = ((i < N - 1) && (j < N - 1)) ? pixels[down * N + right] : 0;
            int cc = pixels[i*N+j];
            if (i % 2)
            {

                if (j % 2)
                {   //green
                    r_sw[i*N+j] = (r + l) / 2;
                    g_sw[i*N+j] = cc;
                    b_sw[i*N+j] = (u + d) / 2;
                }

                else
                { //red
                    r_sw[i*N+j] = cc;
                    g_sw[i*N+j] = (u + d + r + l) / 4;
                    b_sw[i*N+j] = (d1 + d2 + d3 + d4) / 4 ;
                }
            }
            else
            {
                if (j % 2)
                {   //blue
                    r_sw[i*N+j] = (d1 + d2 + d3 + d4) / 4;
                    g_sw[i*N+j] = (u + d + r + l) / 4;
                    b_sw[i*N+j] = cc;
                }

                else
                {   //green
                    r_sw[i*N+j] = (u + d) / 2;
                    g_sw[i*N+j] = cc;
                    b_sw[i*N+j] = (r + l) / 2 ;
                }
            }
        }
    }
         print("HELLO 8\r\n");

    XTime_GetTime(&postExecCyclesSW);
    /*****************************************************************************/

    // Step 6: Compare FPGA and SW results
    //     6a: Report total percentage error
    //     6b: Report FPGA execution time in cycles (use preExecCyclesFPGA and postExecCyclesFPGA)
    //     6c: Report SW execution time in cycles (use preExecCyclesSW and postExecCyclesSW)
    //     6d: Report speedup (SW_execution_time / FPGA_exection_time)

    int *r_hw = (int *)malloc(N*N*sizeof(int));
    int *g_hw = (int *)malloc(N*N*sizeof(int));
    int *b_hw = (int *)malloc(N*N*sizeof(int));
    int err = 0;
```

```
    int err_per, FPGA_cycles, SW_cycles, speedup;

    for (int i=0; i<MAX_PKT_LEN; i++)
    {
          r_hw[i] = RxBufferPtr[i] & 0x0000ff;
          g_hw[i] = (RxBufferPtr[i] & 0x00ff00) >> 2;
          b_hw[i] = (RxBufferPtr[i] & 0xff0000) >> 4;
          if (r_hw[i] == r_sw[i] && g_hw[i] == g_sw[i] && b_hw[i] == b_sw[i]) continue;
          else err++;
    }

    err_per = (err/MAX_PKT_LEN)*100;
    FPGA_cycles = postExecCyclesFPGA - preExecCyclesFPGA;
    SW_cycles = postExecCyclesSW - preExecCyclesSW;
    speedup = SW_cycles/FPGA_cycles;
    printf("Total Percentage error: %d %% \n", err_per);
    printf("FPGA execution time in cycles: %d \n", FPGA_cycles);
    printf("SW execution time in cycles: %d \n", SW_cycles);
    printf("Speedup: %d \n", speedup);


    print("BYE\r\n");
    cleanup_platform();
    return 0;
}
```