پروژه VHDL

مقدمه

هدف از این پروژه، طراحی و پیاده‌سازی یک ماژول VHDL برای انجام تبدیل ویولت روی تصویر ورودی است. در این مسیر، ابتدا الگوریتم مورد نظر با استفاده از زبان Python و سپس زبان C شبیه‌سازی شد تا درک بهتری از جریان داده‌ها و عملیات مورد نیاز به‌دست آید. پس از آن، پیاده‌سازی نهایی با استفاده از زبان HLS C برای پیاده‌سازی روی FPGA صورت گرفت.

در این پروژه، از تصاویر به‌صورت پیکسل‌های سریالی استفاده شد تا پیاده‌سازی حالت استریمی مشابه سخت‌افزار میسر شود. همچنین به کمک مراجع علمی ارائه شده، طراحی معماری تولید آرایه‌های محلی (پنجره‌ای) و پردازش گرادیان دنبال شد.

مراجع مورد استفاده

دو مقاله‌ی زیر به عنوان مرجع معماری و پیاده‌سازی در سخت‌افزار مورد استفاده قرار گرفتند:

- Hardware Description of Multi-Directional Fast Sobel Edge Detection Processor by VHDL for Implementing on FPGA
- Implementation of edge detection algorithms in real time on FPGA

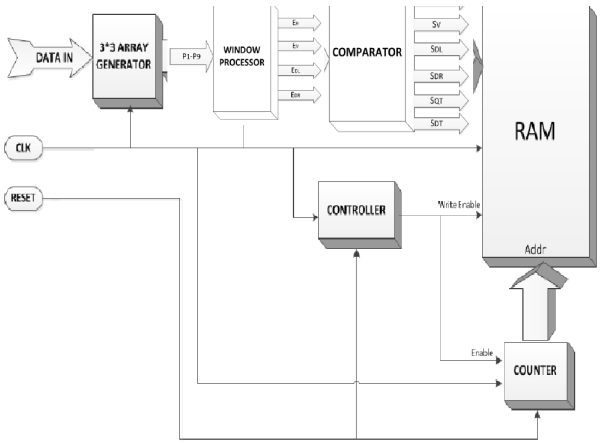
مراحل پیاده‌سازی

۱. پیاده‌سازی اولیه در Python

در ابتدا با هدف درک بهتر فیلترهای تصویر و تشخیص لبه، چندین پیاده‌سازی ساده با زبان Python انجام شد. این پیاده‌سازی‌ها شامل فیلترهای پایه، ماتریس‌های هموارسازی، و همچنین تشخیص لبه با روش‌های کلاسیک بودند. در انتها نیز خروجی‌ها در پوشه‌ای به‌نام python ذخیره شدند.

۲. پیاده‌سازی در زبان C

در ادامه، منطق شبیه‌سازی تولید پنجره‌های 2x2 و 3x3 به زبان C پیاده‌سازی شد تا پایه‌ای برای تبدیل به محیط سخت‌افزاری فراهم گردد. در این مرحله، با استفاده از آرایه‌های خطی شبیه Shift Register، داده‌ها به‌صورت سریالی خوانده شده و پنجره‌های محلی ساخته شدند. سپس پردازش اولیه روی این پنجره‌ها انجام گرفت.



شکل ۱: نقشه‌ی کلی استخراج‌شده از مقاله برای شروع و مسیر پردازش تصویر

۳. طراحی مولد پنجره (Window Generator)

پس از آن، تمرکز بر روی طراحی تابعی برای تولید پنجره‌های 2x2 به‌صورت استریمی قرار گرفت. در این مرحله، برای هر پیکسل ورودی، یک پنجره‌ی ۲ در ۲ تولید می‌شود که برای پردازش‌های بعدی مورد استفاده قرار می‌گیرد.

۴. پیاده‌سازی در محیط HLS C

سپس کد به زبان HLS C منتقل شد تا برای پیاده‌سازی سخت‌افزاری آماده شود. در این پیاده‌سازی، ابتدا یک نسخه ساده از تابع تولید پنجره نوشته شد که تنها از آرایه‌های داخلی استفاده می‌کرد و حالت پیکسل به پیکسل را شبیه‌سازی می‌کرد.

```
#include <stdio.h>

#define WIDTH 5
#define HEIGHT 5

// HLS-compatible function to generate window from pixel stream
void array_generator(
    int pixel,
    int *valid_window,
    int window[3][3])
{
#pragma HLS STREAM variable=pixel dim=1
    static int row1[WIDTH] = {0};
    static int row2[WIDTH] = {0};
    static int curr_row[WIDTH] = {0};
    static int row_idx = 0;
    static int col_idx = 0;

    // Save current pixel
    curr_row[col_idx] = pixel;
    *valid_window = 0; // default no window yet

    if (row_idx >= 2 && col_idx >= 2) {
        // Center at (row_idx - 1, col_idx - 1)
        int center_col = col_idx - 1;

        for (int r = 0; r < 3; r++) {
            for (int c = 0; c < 3; c++) {
                if (r == 0)
```

```
                    window[r][c] = row1[center_col - 1 + c];
                else if (r == 1)
                    window[r][c] = row2[center_col - 1 + c];
                else
                    window[r][c] = curr_row[center_col - 1 + c];
            }
        }
        *valid_window = 1;
    }

    col_idx++;
    if (col_idx == WIDTH) {
        col_idx = 0;

        for (int i = 0; i < WIDTH; i++) {
            row1[i] = row2[i];
            row2[i] = curr_row[i];
            curr_row[i] = 0;
        }

        row_idx++;
    }
}
```

در نسخه پیشرفته‌تر، از کتابخانه‌های آماده مانند hls::stream استفاده شد تا کد قابلیت پشتیبانی از ورودی‌های استریمی را داشته باشد. استفاده از دستورات pragma HLS نیز بهینه‌سازی بیشتری برای پیاده‌سازی سخت‌افزاری فراهم کرد.

```
#include <stdio.h>

#define WIDTH 5
#define HEIGHT 5


void array_generator(
```

```
    int image[HEIGHT][WIDTH],
    int *valid_window,
    int window[2][2],
    int row_idx,
    int col_idx)
{
    *valid_window = 0;

    // Ensure the window fits inside the image
    if (row_idx < HEIGHT - 1 && col_idx < WIDTH - 1) {
        genout: for (int r = 0; r < 2; r++) {
            genin: for (int c = 0; c < 2; c++) {
#pragma HLS PIPELINE
                window[r][c] = image[row_idx + r][col_idx + c];
            }
        }

        *valid_window = 1;
    }
}
```

۵. طراحی ماژول پردازشگر پنجره (Window Processor)

در این بخش، یک پردازشگر برای خروجی‌های پنجره‌ی 2x2 طراحی شد که از آن، چهار مقدار LL, LH, HL, HH به‌دست می‌آیند. این مقدارها طبق روابط زیر محاسبه شده‌اند:

$$\text{LL (Approximation)}: \frac{1}{2}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} \qquad \text{LH (Horizontal detail)}: \frac{1}{2}\begin{bmatrix}1 & -1\\ 1 & -1\end{bmatrix}$$

$$\text{HL (Vertical detail)}: \frac{1}{2}\begin{bmatrix}1 & 1\\ -1 & -1\end{bmatrix} \qquad \text{HH (Diagonal detail)}: \frac{1}{2}\begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix}$$

```
#define WIDTH 372
#define HEIGHT 286
#include <stdio.h>
const int LL_kernel[2][2] = { {1, 1}, {1, 1} };
const int LH_kernel[2][2] = { {1, -1}, {1, -1} };
const int HL_kernel[2][2] = { {1, 1}, {-1, -1} };
const int HH_kernel[2][2] = { {1, -1}, {-1, 1} };

void window_processor(
    int image[HEIGHT][WIDTH],
    int row_idx,
    int col_idx,
  int *valid_window,
    float *LL,
    float *LH,
    float *HL,
    float *HH)
{

    int window[2][2];
    array_generator(image, valid_window, window, row_idx, col_idx);

    if (*valid_window) {
        int sum_LL = 0, sum_LH = 0, sum_HL = 0, sum_HH = 0;

        for (int r = 0; r < 2; r++) {
            for (int c = 0; c < 2; c++) {
#pragma HLS PIPELINE
                sum_LL += window[r][c] * LL_kernel[r][c];
                sum_LH += window[r][c] * LH_kernel[r][c];
                sum_HL += window[r][c] * HL_kernel[r][c];
                sum_HH += window[r][c] * HH_kernel[r][c];
            }
        }
```

```
35
36          *LL = sum_LL * 0.5f;
37          *LH = sum_LH * 0.5f;
38          *HL = sum_HL * 0.5f;
39          *HH = sum_HH * 0.5f;
40      } else {
41          *LL = 0.0f;
42          *LH = 0.0f;
43          *HL = 0.0f;
44          *HH = 0.0f;
45      }
46  }
```

۶. طراحی تابع Comparator

تابع comparator وظیفه‌ی دریافت خروجی‌های خام چهار کانال Haar Wavelet شامل LL، LH، HL و HH را از تابع window_processor دارد. این تابع با بررسی و ترکیب مقادیر این چهار خروجی، یک مقدار نهایی به صورت پیکسل ۸ بیتی uint8 تولید می‌کند.

ایده‌ی اصلی در این مرحله، بهره‌گیری از اطلاعات موجود در کانال‌های ویولت برای استخراج ویژگی‌های مهم تصویر مانند لبه‌ها و حذف نویز بود. در طی روند پیاده‌سازی و آزمون، ترکیب‌های مختلفی مانند موارد زیر بررسی و ارزیابی شدند:

- HH + LL : ترکیب جزییات با بخش هموار تصویر
- LH + HL : ترکیب اطلاعات لبه‌ی افقی و عمودی
- HH + LH + HL : تمرکز کامل بر روی لبه‌ها و جزئیات

در نهایت، با آزمون‌های مختلف روی تصاویر تست، ترکیب‌هایی که بیشترین وضوح لبه‌ها را بدون تولید نویز اضافی ارائه می‌دادند، انتخاب شدند.

یکی از چالش‌های اصلی در این مرحله، جلوگیری از تولید خروجی‌های یکنواخت و خاکستری بود، به‌ویژه زمانی که فیلترهای کاهش نویز اعمال می‌شدند. از آن‌جایی که در طراحی سخت‌افزاری نمی‌توان به پیکسل‌های مجاور دسترسی داشت (بر خلاف روش‌های پردازش تصویر در نرم‌افزار)، تابع comparator

باید به‌گونه‌ای عمل می‌کرد که خروجی تنها با استفاده از همان پنجره‌ی ۲X۲ تولید شود.
در نهایت، منطق تابع به گونه‌ای پیاده‌سازی شد که پیکسل نهایی، ترکیبی نرمال‌سازی‌شده و بهبودیافته از مقادیر ویولت بوده و در اکثر موارد تصویر واضح و لبه‌ها مشخص باقی بماند. کلیه‌ی پیاده‌سازی‌های مربوط به این مرحله نیز با فایل‌های تست جداگانه بررسی و اعتبارسنجی شدند.

```
#include <stdio.h>
#include <stdint.h>

#define WIDTH 372
#define HEIGHT 286
#define ABS_LIMIT 255.0f
#define MIN_STRENGTH 30.0f

uint8_t edge_map_single_value(float val, float abs_limit, float min_strength) {
^^Ifloat v = (val < 0) ? -val : val;
    if (v > abs_limit) v = abs_limit;
    float norm = (v / abs_limit) * 255.0f;
    return (norm >= min_strength) ? 255 : 0;
}
void comparator(
^^I^^Iint image[HEIGHT][WIDTH],
^^I    int row_idx,
^^I    int col_idx,

^^I  int *valid_window,
^^I  uint8_t *combined_lh_hl_hh,
^^I  uint8_t *combined_lh_hl,
^^I  uint8_t *combined_ll_hh,
^^I  uint8_t *norm_HH,
^^I  uint8_t *norm_HL,
^^I  uint8_t *norm_LH,
^^I  uint8_t *norm_LL
^^I  ){

```

```
^^Ifloat LL_val, LH_val, HL_val, HH_val;
^^I window_processor(image, row_idx, col_idx
     , valid_window, &LL_val, &LH_val, &HL_val, &HH_val);
^^I *norm_LL = edge_map_single_value(LL_val,510.0f,40.0f);
^^I *norm_LH = edge_map_single_value(LH_val,ABS_LIMIT,MIN_STRENGTH);
^^I *norm_HL = edge_map_single_value(HL_val,ABS_LIMIT,MIN_STRENGTH);
^^I *norm_HH = edge_map_single_value(HH_val,ABS_LIMIT,MIN_STRENGTH);
^^I float f_norm_LL = (float)(*norm_LL);
^^I float f_norm_LH = (float)(*norm_LH);
^^I float f_norm_HL = (float)(*norm_HL);
^^I float f_norm_HH = (float)(*norm_HH);

^^I float norm_ll_hh = (f_norm_LL + f_norm_HH);
^^I float norm_lh_hl = (f_norm_LH + f_norm_HL);
^^I float norm_lh_hl_hh = (f_norm_HH + f_norm_LH + f_norm_HL);

^^I *combined_ll_hh = (norm_ll_hh > 255.0f) ? 255 : (uint8_t)norm_ll_hh;
^^I *combined_lh_hl = (norm_lh_hl > 255.0f) ? 255 : (uint8_t)norm_lh_hl;
^^I *combined_lh_hl_hh = (norm_lh_hl_hh > 255.0f) ? 255 : (uint8_t)norm_lh_hl_hh;
}
```

شکل ۲: عکس اصلی

(آ) پایتون کد از HH کانال خروجی

(ب) پایتون کد از HL کانال خروجی

(ج) خروجی کانال LH از کد پایتون

(د) خروجی کانال map = lh+hl+hh از کد پایتون

شکل ۳: نمایش افقی کانال‌های مختلف ویولت

۷. تابع نهایی SaveToRamAndOutput

در این تابع، پردازش تصویر در حلقه‌های for انجام شده و خروجی نهایی در قالب تصویر با فرمت PGM ذخیره می‌شود. این تابع با دریافت تصویر ورودی و اعمال ماژول‌های مختلف، تصویر خروجی را تولید می‌کند.

```
#include <stdio.h>
#include <stdint.h>

#define WIDTH 372
#define HEIGHT 286

void comparator(
    int image[HEIGHT][WIDTH],
    int row_idx,
    int col_idx,
    int *valid_window,
    uint8_t *combined_lh_hl_hh,
    uint8_t *combined_lh_hl,
    uint8_t *combined_ll_hh,
```

```
    uint8_t *norm_HH,
    uint8_t *norm_HL,
    uint8_t *norm_LH,
    uint8_t *norm_LL
);


void SaveToRamAndOutput(int input_image[HEIGHT][WIDTH],
uint8_t output_image[HEIGHT - 1][WIDTH - 1]) {

#pragma HLS STREAM variable=input_image dim=2
  uint8_t combined_lh_hl_hh, combined_lh_hl, combined_ll_hh;
    uint8_t norm_HH, norm_HL, norm_LH, norm_LL;
    for (int i = 0; i < HEIGHT - 1; i++) {
        for (int j = 0; j < WIDTH - 1; j++) {
#pragma HLS PIPELINE
      int valid = 0;
            comparator(
                input_image,
                i,
                j,
                &valid,
                &combined_lh_hl_hh,
                &combined_lh_hl,
                &combined_ll_hh,
                &norm_HH,
                &norm_HL,
                &norm_LH,
                &norm_LL
            );

            output_image[i][j] = combined_lh_hl_hh;
        }
    }
```

```
}
```

۸. آزمون و تست ماژول‌ها

برای هر یک از ماژول‌ها، یک Testbench مجزا نوشته شد که عملکرد آن را بررسی کرده و خروجی‌های مناسب را ذخیره می‌نماید. این تست‌ها به رفع خطاها و صحت عملکرد کمک زیادی کردند.

array$_{g}eneratorpixeltb$

```c
#include <stdio.h>

#define WIDTH 5
#define HEIGHT 5

int main() {
    int image[HEIGHT][WIDTH] = {
        { 1, 2, 3, 4, 5 },
        { 6, 7, 8, 9,10 },
        {11,12,13,14,15 },
        {16,17,18,19,20 },
        {21,22,23,24,25 }
    };

    int window[3][3];
    int valid;

    for (int i = 0; i < HEIGHT; i++) {
        for (int j = 0; j < WIDTH; j++) {
            int pixel = image[i][j];
            array_generator(pixel, &valid, window);
            if (valid) {
                printf("\n3x3 window:\n");
                for (int r = 0; r < 3; r++) {
                    for (int c = 0; c < 3; c++) {
```

```
                        printf("%2d ", window[r][c]);
                    }
                    printf("\n");
                }
            }
        }
    }
    return 0;
}

```

window$_{p}rocessortb$

```
#define WIDTH 5
#define HEIGHT 5
#include <stdio.h>

// Example usage
int main() {
    int image[HEIGHT][WIDTH] = {
        {10, 20, 30, 40, 50},
        {15, 25, 35, 45, 55},
        {20, 30, 40, 50, 60},
        {25, 35, 45, 55, 65},
        {30, 40, 50, 60, 70}
    };

    int valid;
    float LL, LH, HL, HH;

    // Process window at position (1,1)
    window_processor(image, 1, 1, &valid, &LL, &LH, &HL, &HH);

    if (valid) {
```

```
        printf("LL = %.2f\n", LL);
        printf("LH = %.2f\n", LH);
        printf("HL = %.2f\n", HL);
        printf("HH = %.2f\n", HH);
    } else {
        printf("Invalid window\n");
    }

    return 0;
}
```

comparator tb

```
#define WIDTH 5
#define HEIGHT 5
#include <stdio.h>
#include <stdint.h>

int main() {
    int valid_window;

    int test_image[HEIGHT][WIDTH] = {
    ^^I^^I    { 100, 2, 10, 50, 8 },
    ^^I^^I    { 4,  1, 28, 38, 12 },
    ^^I^^I    { 7,  14, 22, 44, 9 },
    ^^I^^I    { 6,  30, 16, 48, 11 },
    ^^I^^I    { 9,  20, 13,  7, 14 }
    };
    uint8_t combined_lh_hl_hh, combined_lh_hl, combined_ll_hh;
    uint8_t norm_HH, norm_HL, norm_LH, norm_LL;
    // Test comparator on a few pixels
    comparator(test_image, 2, 3, &valid_window,
                &combined_lh_hl_hh, &combined_lh_hl, &combined_ll_hh,
                &norm_HH, &norm_HL, &norm_LH, &norm_LL);
    printf("At (%d,%d):\n", 2, 3);
```

```
    printf("  HH = %u, HL = %u, LH = %u, LL = %u\n"
    , norm_HH, norm_HL, norm_LH, norm_LL);
    printf("  combined_ll_hh = %u\n", combined_ll_hh);
    printf("  combined_lh_hl = %u\n", combined_lh_hl);
    printf("  combined_lh_hl_hh = %u\n", combined_lh_hl_hh);
    comparator(test_image, 0, 0, &valid_window,
                &combined_lh_hl_hh, &combined_lh_hl, &combined_ll_hh,
                &norm_HH, &norm_HL, &norm_LH, &norm_LL);
    printf("At (%d,%d):\n", 0, 0);
    printf("  HH = %u, HL = %u, LH = %u, LL = %u\n"
    , norm_HH, norm_HL, norm_LH, norm_LL);
    printf("  combined_ll_hh = %u\n", combined_ll_hh);
    printf("  combined_lh_hl = %u\n", combined_lh_hl);
    printf("  combined_lh_hl_hh = %u\n", combined_lh_hl_hh);
    comparator(test_image, 1, 0, &valid_window,
                &combined_lh_hl_hh, &combined_lh_hl, &combined_ll_hh,
                &norm_HH, &norm_HL, &norm_LH, &norm_LL);
    printf("At (%d,%d):\n", 1, 0);
    printf("  HH = %u, HL = %u, LH = %u, LL = %u\n"
    , norm_HH, norm_HL, norm_LH, norm_LL);
    printf("  combined_ll_hh = %u\n", combined_ll_hh);
    printf("  combined_lh_hl = %u\n", combined_lh_hl);
    printf("  combined_lh_hl_hh = %u\n", combined_lh_hl_hh);

    return 0;
}
```

general tb that output edged detected image as a pmg

```
#define WIDTH 372
#define HEIGHT 286
#include <stdio.h>
#include <stdint.h>

void save_pgm(const char *filepath, uint8_t image[HEIGHT - 1][WIDTH - 1]) {
```

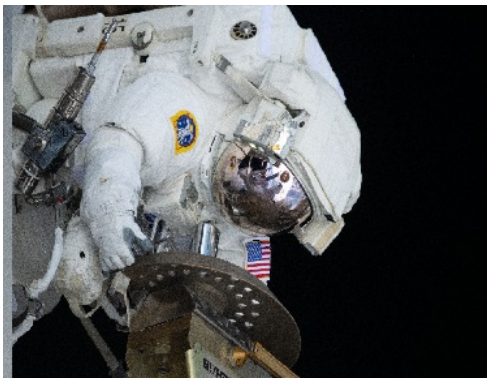
```
    FILE *f = fopen(filepath, "wb");
    if (!f) {
        perror("Failed to open file");
        return;
    }

    fprintf(f, "P5\n%d %d\n255\n", WIDTH - 1, HEIGHT - 1);

    for (int i = 0; i < HEIGHT - 1; i++) {
        fwrite(image[i], sizeof(uint8_t), WIDTH - 1, f);
    }

    fclose(f);
    printf(" Image saved to: %s\n", filepath);
}

int load_pgm(const char *filepath, int image[HEIGHT][WIDTH]) {
    FILE *f = fopen(filepath, "rb");
    if (!f) {
        perror(" Failed to open input PGM file");
        return -1;
    }

    // Read and ignore PGM header
    char magic[3];
    int maxval, widtht, heightt;

    fscanf(f, "%2s", magic);
    if (magic[0] != 'P' || magic[1] != '5') {
        printf(" Not a P5 PGM file.\n");
        fclose(f);
        return -1;
    }

```

```
    fscanf(f, "%d %d", &widtht, &heightt);
    fscanf(f, "%d", &maxval);
    fgetc(f); // consume newline after header


    // Read pixel data
    for (int i = 0; i < HEIGHT; i++) {
        for (int j = 0; j < WIDTH; j++) {
            int pixel = fgetc(f);
            if (pixel == EOF) pixel = 0;
            image[i][j] = pixel;
        }
    }

    fclose(f);
    return 0;
}

int main() {
    int input_image[HEIGHT][WIDTH];
    uint8_t output_image[HEIGHT - 1][WIDTH - 1];
    load_pgm("X:\\pro\\New folder (2)\\outputpgmpython.pgm", input_image);
uint8_t image_copy[HEIGHT - 1][WIDTH - 1];

for (int i = 0; i < HEIGHT - 1; i++) {
    if (i % 512 == 0) {  // Adjust frequency as needed
        printf("Progress: %.2f%%\n", (100.0 * i) / (HEIGHT - 1));
    }
    for (int j = 0; j < WIDTH - 1; j++) {
        int val = input_image[i][j];
        if (val < 0) val = 0;
        if (val > 255) val = 255;
        image_copy[i][j] = (uint8_t)val;
    }
```

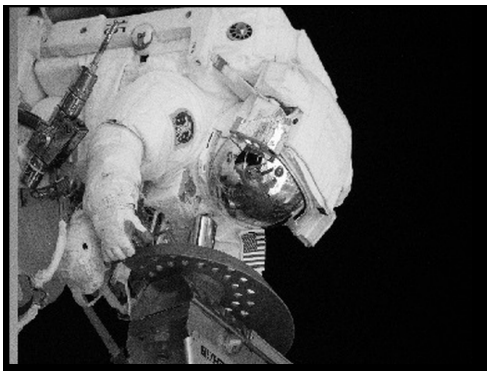
```
}
    save_pgm("X:\\pro\\New folder (2)\\read_back_check.pgm", image_copy);
    SaveToRamAndOutput(input_image, output_image);
    save_pgm("X:\\pro\\New folder (2)\\output.pgm", output_image);
    printf(" Done writing output image.\n");
    return 0;
}
```

تصاویر نمونه خروجی

در ادامه، تصاویر نمونه‌ی ورودی و خروجی ارائه شده‌اند که نشان‌دهنده‌ی عملکرد الگوریتم هستند.

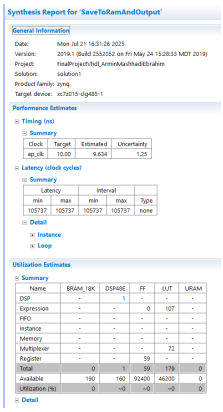


شکل ۴: تصویر اصلی



شکل ۵: تصویر ورودی اولیه بعد از سیاه و سفید کردن و تغییر پسوند به pmg.

شکل ۶: خروجی پس از اعمال تبدیل ویولت



Synthesis Report for 'SaveToRamAndOutput'

General Information

Date: Mon Jul 21 16:31:26 2025
Version: 2019.1 (Build 2552052 on Fri May 24 15:28:33 MDT 2019)
Project: FinalProject/hdl_ArminMaslhadiEbrahim
Solution: solution1
Product family: zynq
Target device: xc7z015-clg485-1

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	9.634	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
105737	105737	105737	105737	none

Detail

Instance

Loop

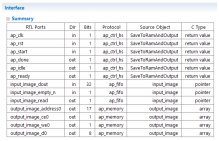
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	1	-	-
Expression	-	-	-	0	107	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	72	-
Register	-	-	-	59	-	-
Total	0	0	0	59	179	0
Available	190	160	92400	46200	0
Utilization (%)	0	0	0	0	0

Detail

(آ) Vivado HLS نرم‌افزار از سنتز کامل گزارش



Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	SaveToRamAndOutput	return value
ap_rst	in	1	ap_ctrl_hs	SaveToRamAndOutput	return value
ap_start	in	1	ap_ctrl_hs	SaveToRamAndOutput	return value
ap_done	out	1	ap_ctrl_hs	SaveToRamAndOutput	return value
ap_idle	out	1	ap_ctrl_hs	SaveToRamAndOutput	return value
ap_ready	out	1	ap_ctrl_hs	SaveToRamAndOutput	return value
input_image_dout	in	32	ap_fifo	input_image	pointer
input_image_empty_n	in	1	ap_fifo	input_image	pointer
input_image_read	out	1	ap_fifo	input_image	pointer
output_image_address10	out	17	ap_memory	output_image	array
output_image_ce0	out	1	ap_memory	output_image	array
output_image_we0	out	1	ap_memory	output_image	array
output_image_d0	out	8	ap_memory	output_image	array

(ب) طراحی زمان‌بندی و منابع از استفاده خلاصه

شکل ۷: نمایش گزارش‌های مربوط به سنتز طراحی در HLS