UNIVERSITÀ DI PISA

Computer Engineering

SAD Image Calculation

Electronics and Communications Systems Project

Gaia Anastasi

Academic Year 2021/2022

Contents

1 Introduction

The aim of this work is to design a digital circuit to implement the sum of absolute differences (SAD), which is a measure of the similarity between image blocks, monochromatic images in digital image processing.

1.1 Similarity estimation

SAD calculates the similarity taking into consideration the absolute difference between pixels of two monochromatic images, pixels by pixels. These differences are summed to create a simple metric of block similarity, the **L1 norm** of the difference image or **Manhattan distance** between two image blocks.

1.1.1 Manhattan distance

A taxicab geometry or a *Manhattan geometry* is a geometry in which the distance between two points is the sum of the absolute differences of their Cartesian coordinates. In two dimensions, the distance between two points $(x_1, y_1), (x_2, y_2)$ is:

$$d = |x_1 - x_2| + |y_1 - y_2| \tag{1}$$

This method is also known as Manhattan geometry because it alludes to the grid layout of most streets on the island of Manhattan, where a taxi's shortest path is the total of the absolute values of the lengths it traverses on avenues and streets to go from a place to another.

1.2 Application usage

SAD may be used for a variety of purposes, such as *object recognition, generation of disparity maps for stereo images*, and *motion estimation for video compression.* The latter one is the major use case of SAD. It consists in determining motion vectors that describe the transformation from one 2D image to another, usually from adjacent frames in a video sequence.

2 Architecture

In this section the general architecture of the system will be discussed.

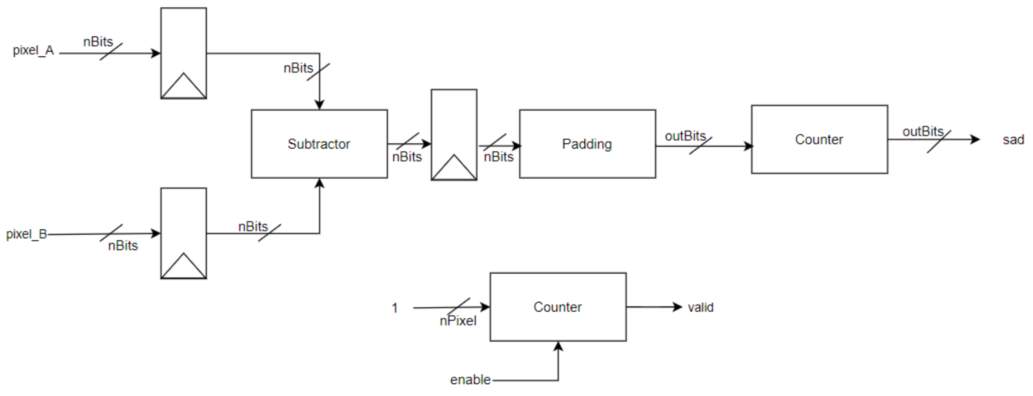


Figure 1: System architecture

2.1 Subtractor

Subtraction can be seen as a sum with the second addend reversed in sign and input carry equals to 1 so the subtractor structure is implemented as a *Ripple Carry Adder* which takes two inputs: the minuend and the subtrahend of the subtraction, and an input carry.

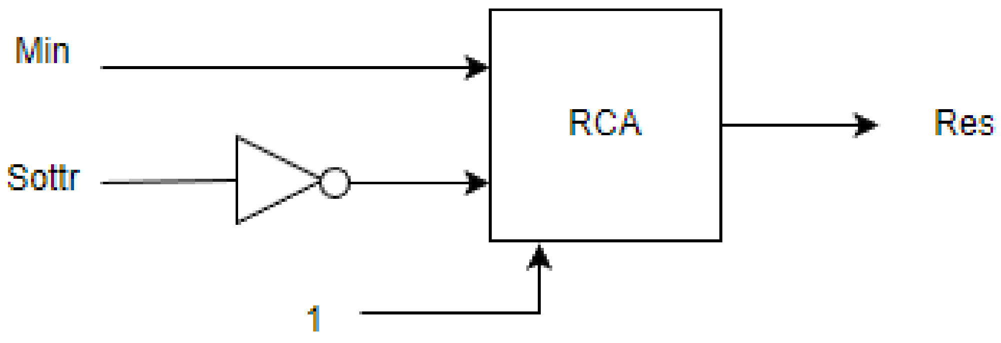


Figure 2: Subtractor architecture

2.2 Padding

The padding is introduced to expand the inputs of the **SAD** counter on *outBits* bits in order to avoid computations overflows.

2.3 Counter

The counter architecture is the following:

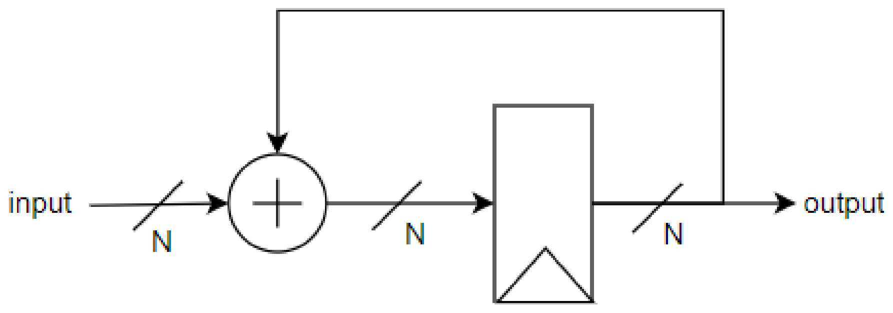


Figure 3: Counter architecture

The counter is implemented as a *Ripple Carry Adder* followed by a *D Flip-Flop* with an asynchronous reset active low and with an enable. Counter is used twice in the circuit: **SAD counter** and **Pixel counter**

2.3.1 SAD Counter

The SAD counter is used to compute the SAD output value. It takes as an input a value on *outBits* bits, the padded absolute difference between the two pixels, and it returns as output the sum of the provided input with the value previously computed and now stored in the *D Flip-Flop*. If valid is high, the computation is over and the final results is in *SAD*.

2.3.2 Pixel Counter

The Pixel Counter is used to count the number of pixels processed and to determine when to set the **valid** signal. It takes always 1 as an input and it sums it to the previous computed result.

3 VHDL Implementation

In the following chapter only the main sections of architecture will be presented, but for futher details, it is possible to check the contents of *src* directory.

3.1 SAD

The system implememtation could be checked in *sad.vhd* file. This module connects all the system components.

```
entity sad is
    generic (
        nPixel   :  positive:=5;   --total pixels (2^nPixel)
        nBits    :  positive:=8;    --bits in each pixel
        outBits  :  positive:= 16     --bits for the output
    );
    port(
        pixel_A       : in std_logic_vector(nBits-1 downto 0);
        pixel_B       : in std_logic_vector(nBits-1 downto 0);
        clk           : in std_logic;
        rst           : in std_logic;
        enable        : in std_logic;
        sad           : out std_logic_vector(outBits-1 downto 0);
        valid         : out std_logic;
        new_comp      : in std_logic
    );



end sad;
```

It was used a generic stucture in order to make easier some future modifications on the system. The system works in the following way: it takes as input 2 pixels: **pixel_A** and **pixel_B**, and it performs an absolute difference between these two. The result of this computation is then expanded, in order to avoid overflow, and then summed to the sum of the absolutes differences computed during the previous stages or with 0 if the pixels are the first ones of a new computation. When all the pixels have been processed the final results is returned on the **sad** signal and **valid** signal is raised to let the user know that the computation is over. The valid signal remains up for a cycle clock, then it is reset. An user can require the elaboration of more than a couple of images. For each new elaboration it is required to raise the **new_comp** signal to let the system know that a new elaboration is starting. When **new_comp** signal is raised the *DFF* inside the *SAD counter* and the *Pixel Counter* are both reset in order to start a new computation.

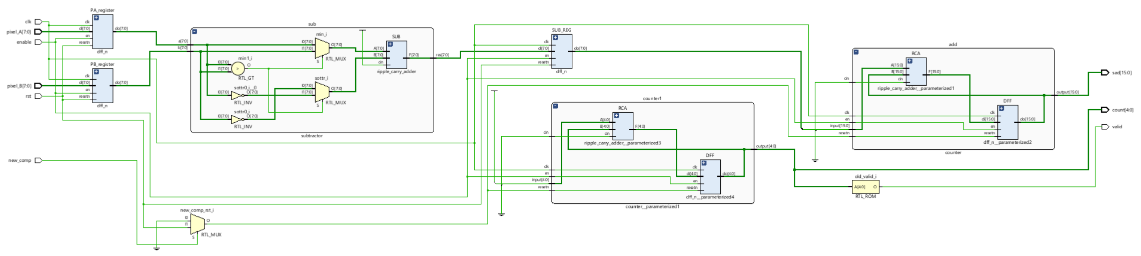


Figure 4: SAD architecture

4 Test plan

In order to verify the correctness of the system the following tests were performed:

- **Unit tests:** each submodules has a dedicated testbench in order to check the correctness of the single submodule.
- **Complete test:** The whole system is tested with a testbench in order to check that each components works well and the system do what is required.

4.1 Complete Test

In order to verify the correctness of the signal some tests were performed proving many different input combinations and special cases. For the following tests we have used these configuration:

```vhdl
constant nPixel   :   positive:=5;     --total pixels 2^
  nPixel
constant nBits    :   positive:=8;     --pixel bits
    constant outBits :   positive:=16;    --output bits
    constant CLK_PERIOD : time  := 100 ns;
    constant FINISH : positive  := 2**(nPixel-1)+1;
```

In all the tests we have also output a *count* signal in order to show the **Pixel counter** value at each clock cycle.

4.1.1 Test 1

In this particular test the input are kept still for the whole test and only one elaboration is performed. This test checks the behaviour of the system in the most simple case ever.

```vhdl
   stimulus : process
    begin
        pixel_A_ext <= (others => '1');
        pixel_B_ext <= "00000001";
        rst <= '1';
        new_comp_ext<= '1';

        wait until rising_edge(clk);
        new_comp_ext <= '0';

        wait for FINISH*CLK_PERIOD;
        wait for 200 ns;
        testing <= false;

    end process;
```

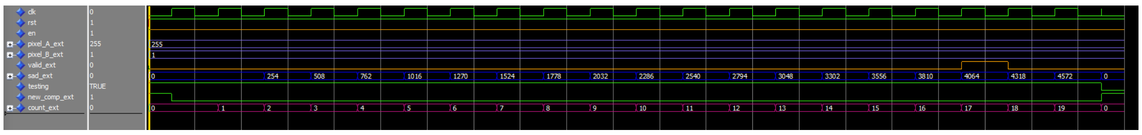


Figure 5: Test1 results on ModelSim

4.1.2 Test 2

In this particular test the input are kept still again for the whole test but more elaborations are performed. After 2 consecutive elaborations the *enable* signal is set for a clock cycle and then another elaboration is performed.

```
   stimulus : process
    begin
        pixel_A_ext <= (others => '1');
        pixel_B_ext <= "00000001";
        rst <= '1';
        new_comp_ext<= '1';

        wait until rising_edge(clk);
        new_comp_ext <= '0';

        wait for FINISH*CLK_PERIOD;
        wait for 200 ns;
        new_comp_ext <= '1';
        wait until rising_edge(clk);
        new_comp_ext <= '0';
        wait for FINISH*CLK_PERIOD;
        new_comp_ext <= '1';
        wait until rising_edge(clk);
        en<='0';
        new_comp_ext <= '0';
        wait until rising_edge(clk);
        en<='1';

        wait for FINISH*CLK_PERIOD;
        testing <= false;

    end process;
```

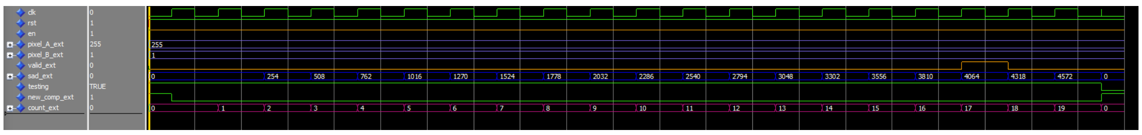


Figure 6: Test1 results on ModelSim

The following image is a zoomed picture in order to see the results better, in particular, the attention is focused on the beginning of a new elaboration and the reset of enable signal.

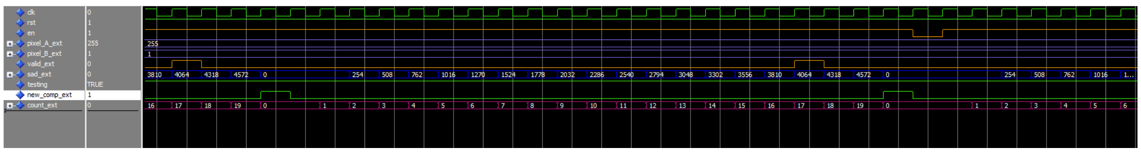


Figure 7: Detailed view of the test results

4.1.3 Test 3

The same test perfomed in the previous section is now performed varying the input at each clock cycle. The test is been generated usign a *Python* script. For further details check the **test3_script.py** script in the *tb* directory.

```
  for a in range(finish*NUM_TEST):
   if a % finish == 0:
       out_file.write("    new_comp_ext <= '1';\n")
   else:
       out_file.write("    new_comp_ext <= '0';\n")

   pixelA = pixel()    #random pixel
   pixelB = pixel()    #random pixel
   out_file.write("    pixel_A_ext <= \"" +pixelA + "
     \"; \n" )
   out_file.write("    pixel_B_ext <= \"" +pixelB + "
     \"; \n" )
```

```
        out_file.write("wait until rising_edge(clk);\n")
```

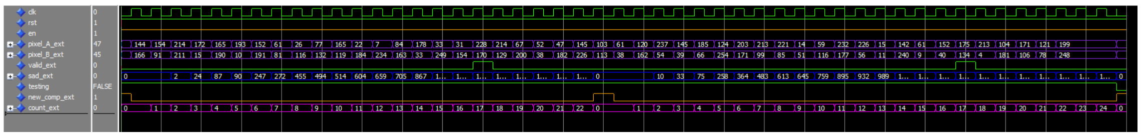


Figure 8: Test3 results on ModelSim

The following image is a zoomed picture in order to see the results better

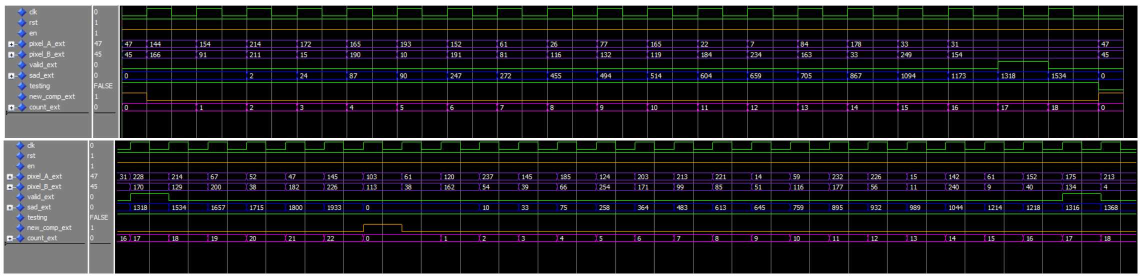


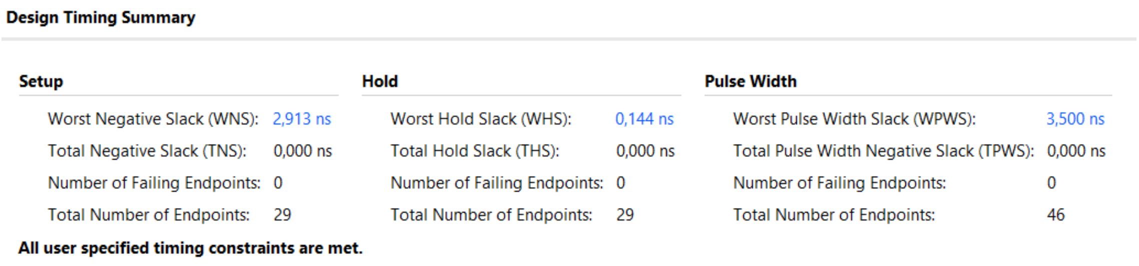
Figure 9: Detailed view of test3 results

5 Synthesis

In this chapter the results obtained creating the project with Xilinx Vivado are presented. The device selected as a working device is the **Zybo Zynq-7000(xc7z010clg400-1)**, the clock period is set to 8ns and a timing constraint is added.

5.1 Synthesis

After the synthesis is been performed this was the timing report returned:



Design Timing Summary

Setup

Worst Negative Slack (WNS):	2,913 ns
Total Negative Slack (TNS):	0,000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	29

Hold

Worst Hold Slack (WHS):	0,144 ns
Total Hold Slack (THS):	0,000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	29

Pulse Width

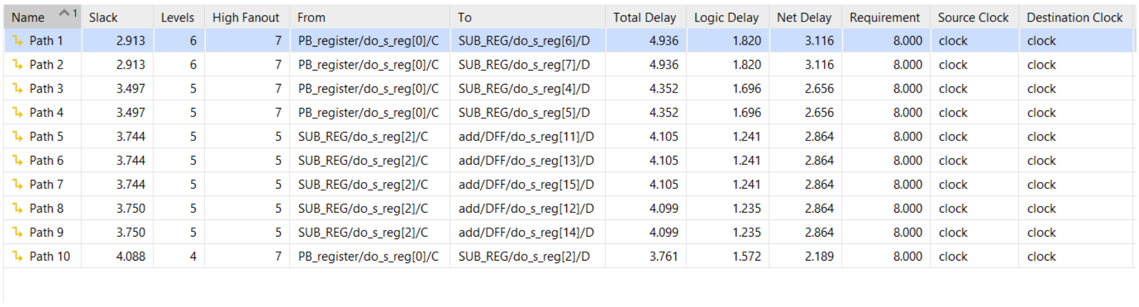
Worst Pulse Width Slack (WPWS):	3,500 ns
Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	46

All user specified timing constraints are met.

Figure 10: Timing report after synthesis

Since the **Worst Negative Slack** is positive the timing constraint is met. The **maximum frequency** can be computes as follows:

$$f_{max} = \frac{1}{T_{clk} - WNS} = 196,6MHz \tag{2}$$

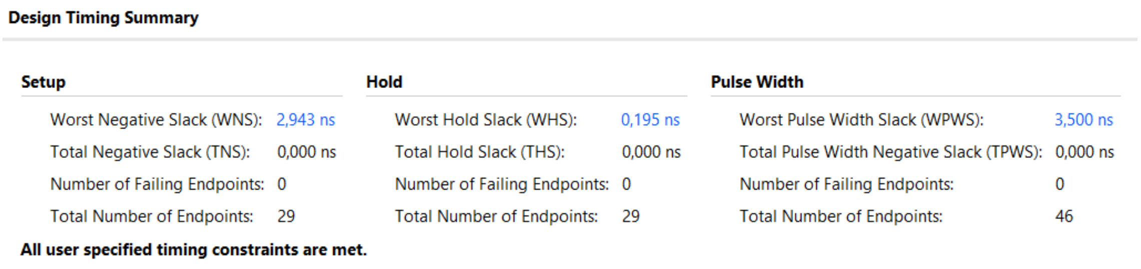


Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 1	2.913	6	7	PB_register/do_s_reg[0]/C	SUB_REG/do_s_reg[6]/D	4.936	1.820	3.116	8.000	clock	clock
Path 2	2.913	6	7	PB_register/do_s_reg[0]/C	SUB_REG/do_s_reg[7]/D	4.936	1.820	3.116	8.000	clock	clock
Path 3	3.497	5	7	PB_register/do_s_reg[0]/C	SUB_REG/do_s_reg[4]/D	4.352	1.696	2.656	8.000	clock	clock
Path 4	3.497	5	7	PB_register/do_s_reg[0]/C	SUB_REG/do_s_reg[5]/D	4.352	1.696	2.656	8.000	clock	clock
Path 5	3.744	5	5	SUB_REG/do_s_reg[2]/C	add/DFF/do_s_reg[11]/D	4.105	1.241	2.864	8.000	clock	clock
Path 6	3.744	5	5	SUB_REG/do_s_reg[2]/C	add/DFF/do_s_reg[13]/D	4.105	1.241	2.864	8.000	clock	clock
Path 7	3.744	5	5	SUB_REG/do_s_reg[2]/C	add/DFF/do_s_reg[15]/D	4.105	1.241	2.864	8.000	clock	clock
Path 8	3.750	5	5	SUB_REG/do_s_reg[2]/C	add/DFF/do_s_reg[12]/D	4.099	1.235	2.864	8.000	clock	clock
Path 9	3.750	5	5	SUB_REG/do_s_reg[2]/C	add/DFF/do_s_reg[14]/D	4.099	1.235	2.864	8.000	clock	clock
Path 10	4.088	4	7	PB_register/do_s_reg[0]/C	SUB_REG/do_s_reg[2]/D	3.761	1.572	2.189	8.000	clock	clock

Figure 11: Critical paths

5.2 Implementation

After performing implementation the following timing report was obtained:



Design Timing Summary

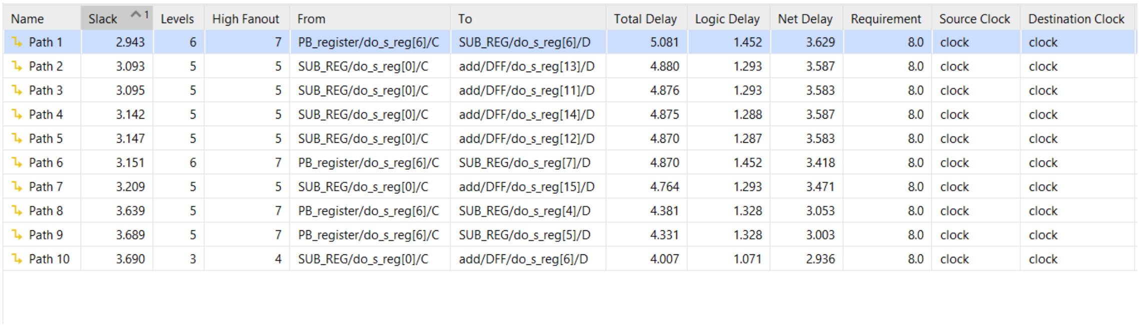
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2,943 ns	Worst Hold Slack (WHS):	0,195 ns	Worst Pulse Width Slack (WPWS):	3,500 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	29	Total Number of Endpoints:	29	Total Number of Endpoints:	46

All user specified timing constraints are met.

Figure 12: Timing report after implementation

Since the **Worst Negative Slack** is positive again the timing constraint is met. The **maximum frequency** is computed as before:

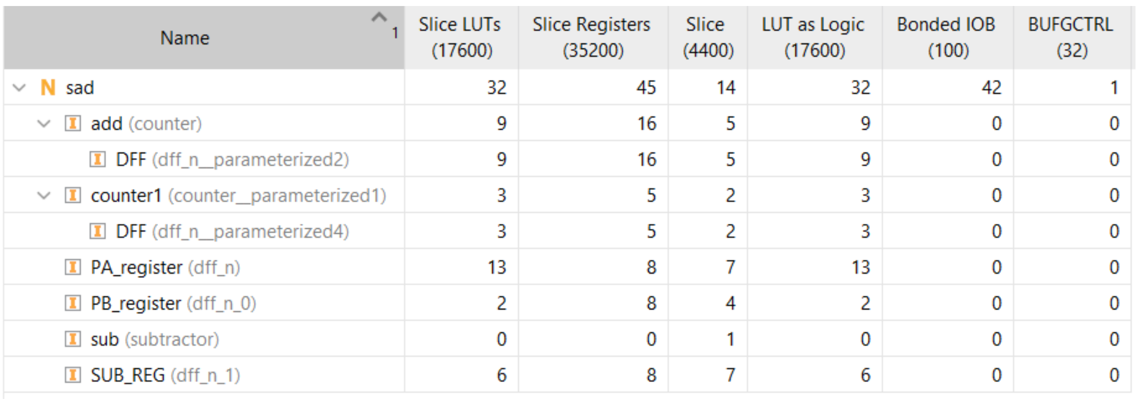
$$f_{max} = \frac{1}{T_{clk} - WNS} = 197,7MHz \tag{3}$$



Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock
Path 1	2.943	6	7	PB_register/do_s_reg[6]/C	SUB_REG/do_s_reg[6]/D	5.081	1.452	3.629	8.0	clock	clock
Path 2	3.093	5	5	SUB_REG/do_s_reg[0]/C	add/DFF/do_s_reg[13]/D	4.880	1.293	3.587	8.0	clock	clock
Path 3	3.095	5	5	SUB_REG/do_s_reg[0]/C	add/DFF/do_s_reg[11]/D	4.876	1.293	3.583	8.0	clock	clock
Path 4	3.142	5	5	SUB_REG/do_s_reg[0]/C	add/DFF/do_s_reg[14]/D	4.875	1.288	3.587	8.0	clock	clock
Path 5	3.147	5	5	SUB_REG/do_s_reg[0]/C	add/DFF/do_s_reg[12]/D	4.870	1.287	3.583	8.0	clock	clock
Path 6	3.151	6	7	PB_register/do_s_reg[6]/C	SUB_REG/do_s_reg[7]/D	4.870	1.452	3.418	8.0	clock	clock
Path 7	3.209	5	5	SUB_REG/do_s_reg[0]/C	add/DFF/do_s_reg[15]/D	4.764	1.293	3.471	8.0	clock	clock
Path 8	3.639	5	7	PB_register/do_s_reg[6]/C	SUB_REG/do_s_reg[4]/D	4.381	1.328	3.053	8.0	clock	clock
Path 9	3.689	5	7	PB_register/do_s_reg[6]/C	SUB_REG/do_s_reg[5]/D	4.331	1.328	3.003	8.0	clock	clock
Path 10	3.690	3	4	SUB_REG/do_s_reg[0]/C	add/DFF/do_s_reg[6]/D	4.007	1.071	2.936	8.0	clock	clock

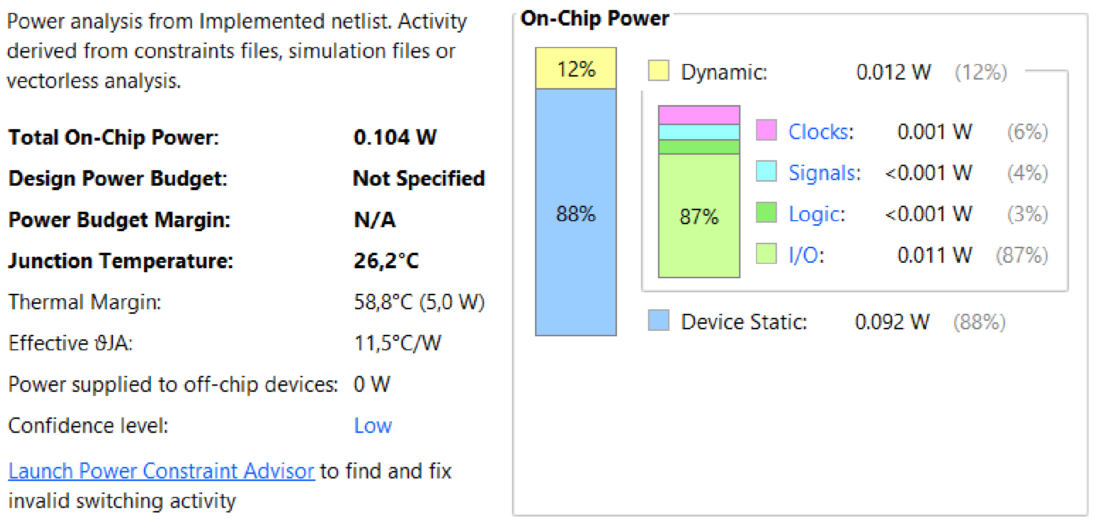
Figure 13: Critical paths

Below the *Resource utilization* and the *Power utilization* reports are shown:



Name	Slice LUTs (17600)	Slice Registers (35200)	Slice (4400)	LUT as Logic (17600)	Bonded IOB (100)	BUFGCTRL (32)
sad	32	45	14	32	42	1
add (counter)	9	16	5	9	0	0
DFF (dff_n__parameterized2)	9	16	5	9	0	0
counter1 (counter__parameterized1)	3	5	2	3	0	0
DFF (dff_n__parameterized4)	3	5	2	3	0	0
PA_register (dff_n)	13	8	7	13	0	0
PB_register (dff_n_0)	2	8	4	2	0	0
sub (subtractor)	0	0	1	0	0	0
SUB_REG (dff_n_1)	6	8	7	6	0	0

Figure 14: Resource utilization


Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.104 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26,2°C**
Thermal Margin:	58,8°C (5,0 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

🟨	Dynamic:	0.012 W	(12%)
🟪	Clocks:	0.001 W	(6%)
🟦	Signals:	<0.001 W	(4%)
🟩	Logic:	<0.001 W	(3%)
🟩	I/O:	0.011 W	(87%)
🟦	Device Static:	0.092 W	(88%)

Figure 15: Power Utilization

5.3 Errors and warnings

During synthesis and implementation phases no relevant warnings or errors were raised other than those relating to non-assigment of I/O pins of the selected board.

6 Conclusion

In conclusion, the digital circuit implemented here is able to take 2 images composed by NxN pixels on 8 bits and to compute the *sum of absolute differences* raising the valid signal when the computation is over. It is also able to process more than a couple of images.

7 References

References

[1] *Sum of absolute differences:*
https://en.wikipedia.org/wiki/Sum_of_absolute_differences

[2] *SIM/Uses/SAD:*
https://wiki.mozilla.org/SIMD/Uses/SAD