Sommario

[0. Project specifications 2](#_Toc106981589)

[1. Introduction 2](#_Toc106981590)

[1.1 Field of applications and examples 2](#_Toc106981591)

[1.2 Algorithm Description 3](#_Toc106981592)

[1.3 Possible Architectures 3](#_Toc106981593)

[2. Architecture Description 4](#_Toc106981594)

[2.1 Counter 4](#_Toc106981595)

[2.2 Subtractor 5](#_Toc106981596)

[2.3 Accumulator 5](#_Toc106981597)

[2.4 General SAD architecture 6](#_Toc106981598)

[3. VHDL code 6](#_Toc106981599)

[3.1 DFF\_N 6](#_Toc106981600)

[3.2 Counter 7](#_Toc106981601)

[3.3 Subtractor 7](#_Toc106981602)

[3.4 Accumulator 8](#_Toc106981603)

[3.5 SAD 9](#_Toc106981604)

[4. Test Plan 11](#_Toc106981605)

[4.1 DFF\_N Testbench 11](#_Toc106981606)

[4.2 Counter Testbench 12](#_Toc106981607)

[4.3 Subtractor Tesbench 13](#_Toc106981608)

[4.4 Accumulator Testbench 14](#_Toc106981609)

[4.5 SAD Testbench 16](#_Toc106981610)

[4.6 SAD Freeze Testbench 17](#_Toc106981611)

[4.7 SAD Testbench with python script 19](#_Toc106981612)

[5. Logic Synthesis for the standard case (16x16 images) 23](#_Toc106981613)

[5.1 RTL Analysis 23](#_Toc106981614)

[5.2 Synthesis 23](#_Toc106981615)

[5.2.1 Clock period setting 24](#_Toc106981616)

[5.2.2 Critical Path 24](#_Toc106981617)

[5.2.3 Power consumption 25](#_Toc106981618)

[6.1 RTL analysis in 64x64 images case 26](#_Toc106981619)

[6.2 Synthesis on 64x64 images case 26](#_Toc106981620)

[6.2.1 Clock period setting 26](#_Toc106981621)

[6.2.2 Utilization report 27](#_Toc106981622)

[6.2.3 Power consumption report 27](#_Toc106981623)

[7. Conclusions 27](#_Toc106981624)

[References 28](#_Toc106981625)

# 0. Project specifications

1) Design a synchronous digital circuit that implement the SAD calculation, defined as the sum of pixel values absolute differences between two blocks of monochromatic images A and B. Consider 2 blocks of size 16 pixel x 16 pixel, each pixel has an integer value between 0 and 255 represented in 8 bit. The inputs are the clock, the reset, the enable and PA and PB signals that will be provided by an external source, the 256 pixel values of A and B are provided sequentially in different clock cycles. The output is a SAD signal in 16 bits and a data\_valid signal of 1 bit. On reset SAD=0 and data\_valid=0. Data\_valid is set at the end of the SAD calculation. If enable=0 the circuit keeps its state independently from the input values.

2) Define a VHDL parametrized version of point 1 for the case of NxN image blocks, with N power of 2. Note: The input and output signals are the same such that:

The timing constraints must be changed because there are pixel couples.

# 1. Introduction

The sum of absolute differences is a measure of similarity between image blocks, it consists in calculating the absolute differences between each pixel value of the two images and summing them to obtain a similarity metric. The higher the value the lower the similarity between the 2 blocks.

Immagine che contiene testo, orologio

Descrizione generata automaticamente

Figure 1: SAD computation of two R × C matrices **A** and **B**

## 1.1 Field of applications and examples

SAD is used for different purposes like object recognition, video compression, disparity maps and others.

For example, in the case of **object detection**, the environment light changing, and other factors can lead to misinterpretation of objects. In this context sad is used as a complementary method working with other type of object detection, like edge detection, to cope with these problems and to improve the reliability of the results.

In **digital image processing** the sum of absolute differences is used to identify which part of a search image is most similar to a template image. In the following example we have the comparison of a 3x3 template and a 3x5 search image where each pixel is represented with an integer value from 0 to 9.

Immagine che contiene testo

Descrizione generata automaticamente

Figure 2: Template and Search Image pixel values

The template can fit in the left side of the search image, in the center and in the right side, shifting the block by one column for each case. Calculating the absolute differences between each pixel:

Immagine che contiene testo

Descrizione generata automaticamente

Figure 3: left, center and right absolute differences

For each case we can add the 9 absolute differences to get the SAD values, respectively: 20, 25, 17. We can state that the right side of the search image is the most similar to the template, that’s because it’s the lower value.

## 1.2 Algorithm Description

The algorithm used is the basic one and it consists in three main operations:

* Taking the pixel values couples sequentially
* Calculate the absolute difference
* Sum to the total value (SAD)

Those steps repeat until the counter threshold is reached, for example considering 16x16 blocks the total number of sums is 256. Further improvements can be made considering parallel solution (see next paragraph).

## 1.3 Possible Architectures

In “*Efficient sum of absolute difference computation on FPGAs”* (see References)the circuit exploits parallelism in the absolute difference calculation, in the image below there is the proposed architecture.

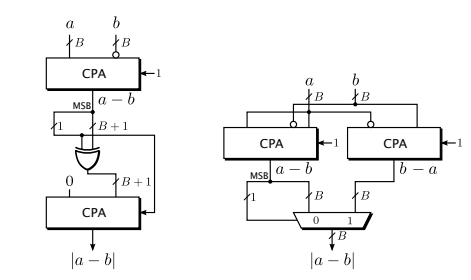


Figure 4: Sequential vs Parallel Absolute Difference Calculation circuits

In this case, instead of negating the result if negative, both and are calculated in parallel and the positive result is selected by the most significant bit. In this way the complexity is higher, but the critical path is shorter. A third solution is evaluating if and then perform one of the 2 operations, this is simpler and faster due to the carry-chain MUXes of the Xilinx architecture.

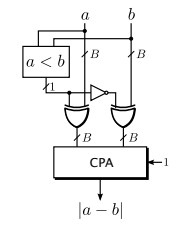


Figure 5: FPGA optimized absolute difference circuits

To improve the performances in the SAD calculation some basic SAD core units can be put in parallel like shown in *Figure 5*.

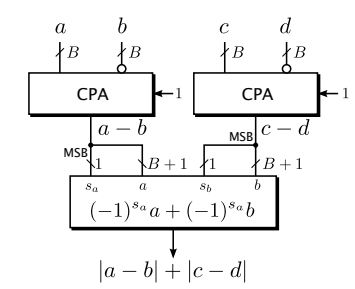


Figure 6: SAD core units in parallel

*Note:* CPA refers to carry propagate adder.

# 2. Architecture Description

Three different components have been considered to implement the SAD calculation: clock counter, subtractor, accumulator. A classical D-flip-flop implementation has been used to keep the circuit state.

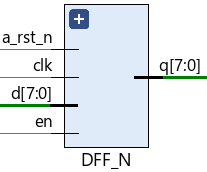


Figure: DFF\_N schematic

## 2.1 Counter

This component is used to understand the end of the SAD calculation. When the enable signal is set it counts the clock cycles and so the number of sums performed in total, otherwise the counting is freeze. When the counting value overflows it means that the SAD calculation is finished, the overflow signal triggers the accumulator to stop summing the inputs and to set the data\_valid signal.

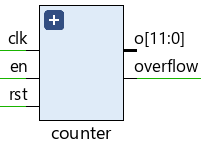


Figure: counter schematic

## 2.2 Subtractor

The subtractor is a combinational circuit that calculate the absolute difference of the 2 inputs, it is implemented following the logic of the parallel absolute difference presented before.

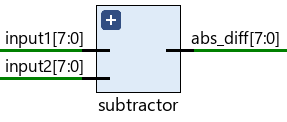


Figure: subtractor schematic

## 2.3 Accumulator

The accumulator adds the output coming from the subtractor to the total SAD value, when it receives the overflow signal from the counter it adds the last absolute difference and then set the data\_valid signal.

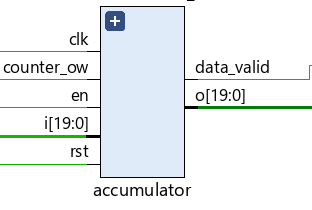


Figure: accumulator schematic

## 2.4 General SAD architecture

The general architecture can be summarized using the following scheme:

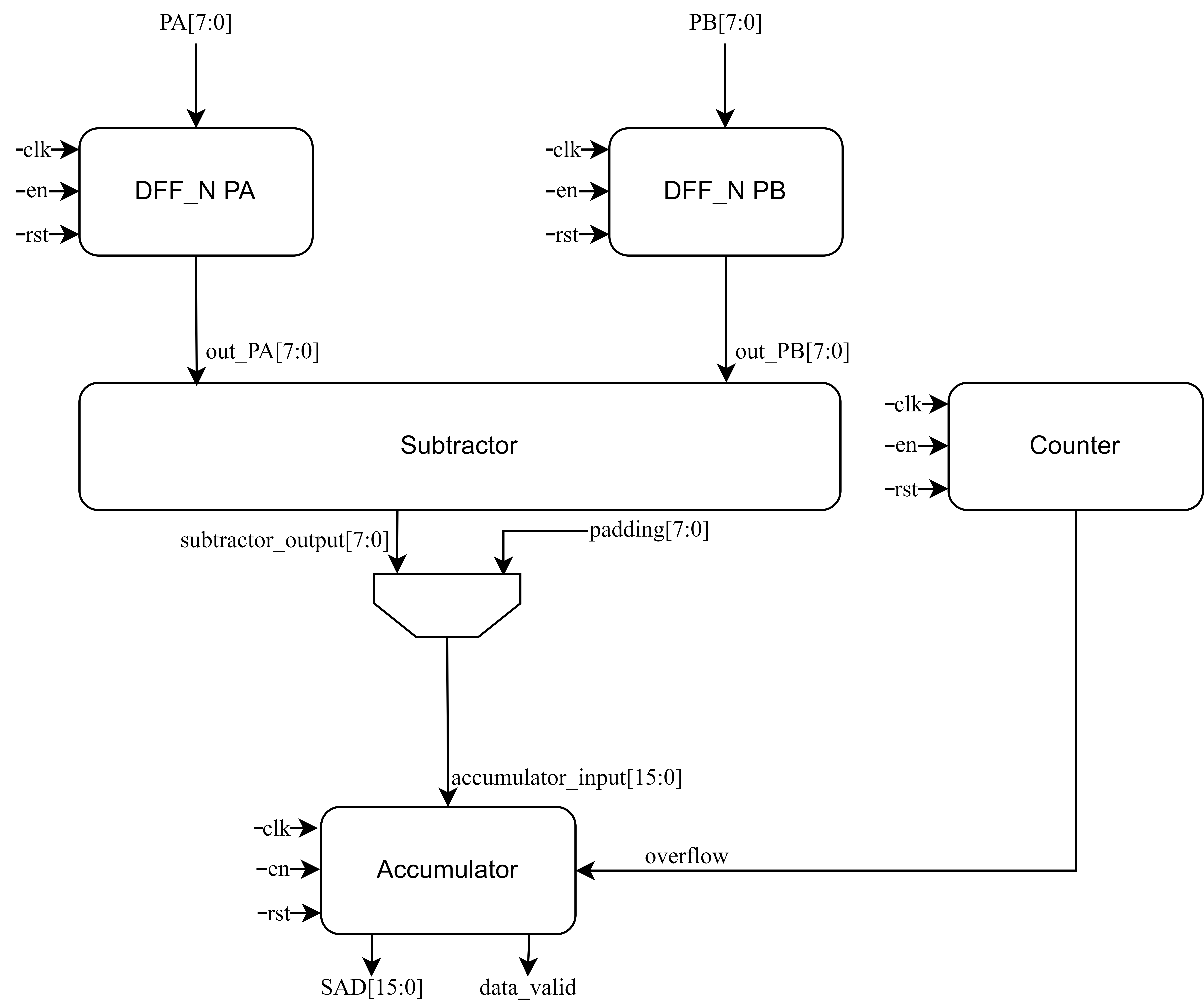


Figure 7: General SAD architecture

# 3. VHDL code

The architecture described in VHDL follows a behavioral approach to keep code as simple as possible and to have an explicit view on the circuit behavior. The proposed solutions are inspired by the architectures presented at the introduction.

The developing of the project follows a bottom-up approach where starting from the implementation of the main components we put them all together to build the complete circuit.

## 3.1 DFF\_N

This is a classical implementation of d-flip-flop with set/reset signals.

**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.all;**

**entity** DFF\_N **is**

**generic(** NBit **:** positive **:=** 8**);**

**port(**

clk **:** **in** std\_logic**;**

a\_rst\_n **:** **in** std\_logic**;**

en **:** **in** std\_logic**;**

d **:** **in** std\_logic\_vector**(**NBit **-** 1 **downto** 0**);**

q **:** **out** std\_logic\_vector**(**NBit **-** 1 **downto** 0**)**

**);**

**end** DFF\_N**;**

**architecture** struct **of** DFF\_N **is**

**begin**

ddf\_n\_proc**:** **process(**clk**,** a\_rst\_n**)**

**begin**

**if(**a\_rst\_n **=** '1'**)** **then**

q **<=** **(others** **=>** '0'**);**

**elsif(rising\_edge(**clk**))** **then**

**if(**en **=** '1'**)** **then**

q **<=** d**;**

**end** **if;**

**end** **if;**

**end** **process;**

**end** struct**;**

## 3.2 Counter

The counter architecture is equipped with an output register to maintain the counting value, the latter is freeze if the enable signal is not set. When max\_val threshold is reached the counter overflows and set the corresponding signal.

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**STD\_LOGIC\_ARITH**.all;**

**use** IEEE**.**STD\_LOGIC\_UNSIGNED**.all;**

**entity** counter **is**

**generic** **(**

NBit **:** positive **:=** 8

**);**

**port** **(**

en**:** **in** std\_logic**;**

rst**,**clk **:** **in** std\_logic**;**

overflow**:** **out** std\_logic**;**

o**:** **out** std\_logic\_vector**(** NBit**-**1 **downto** 0 **)**

**);**

**end** counter**;**

-- when en = '1' it counts the clock cycles, otherwise the counting is freezed

-- set overflow to 1 when max\_val is reached

**architecture** beh **of** counter **is**

**signal** output\_reg **:** std\_logic\_vector **(**NBit**-**1 **downto** 0**);**

**constant** max\_val **:** std\_logic\_vector **(**NBit**-**1 **downto** 0**)** **:=** **(others** **=>** '1'**);**

**begin**

counter\_output\_reg**:** **process(**clk**,** rst**)**

**begin**

**if(**rst **=** '1'**)** **then**

output\_reg **<=** **(others** **=>** '0'**);**

overflow **<=** '0'**;**

**elsif((rising\_edge(**clk**)))** **then**

**if** **(**en**=**'1'**)** **then**

-- when max\_val is reached the counter overflows and overflow is set to '1'

**if** **(**unsigned**(**output\_reg**)** **=** unsigned**(**max\_val**))** **then**

output\_reg **<=** **(others** **=>** '0'**);**

overflow **<=** '1'**;**

**else**

output\_reg **<=** output\_reg **+** 1**;**

**end** **if;**

**end** **if;**

**end** **if;**

**end** **process** counter\_output\_reg**;**

-- mapping the output

o **<=** output\_reg**;**

**end** beh**;**

## 3.3 Subtractor

The subtractor implementation is given by a combinational process where two inputs are fed into the circuit and the absolute difference is done depending on the value of those.

**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.all;**

**use** ieee**.**numeric\_std**.all;**

**entity** subtractor **is**

**generic** **(**

NBit**:** positive **:=** 8

**);**

**port** **(**

input1**:** **in** std\_logic\_vector **(**NBit**-**1 **downto** 0**);**

input2**:** **in** std\_logic\_vector **(**NBit**-**1 **downto** 0**);**

abs\_diff**:** **out** std\_logic\_vector **(**NBit**-**1 **downto** 0**)**

**);**

**end** subtractor**;**

-- Perform the difference between the 2 inputs and get the abs value

**architecture** beh **of** subtractor **is**

**begin**

sub\_process**:** **process(**input1**,** input2**)**

**begin**

**if** unsigned**(**input1**)** **>=** unsigned**(**input2**)** **then**

abs\_diff **<=** std\_logic\_vector**(** unsigned**(**input1**)-**unsigned**(**input2**)** **);**

**else**

abs\_diff **<=** std\_logic\_vector**(** unsigned**(**input2**)-**unsigned**(**input1**)** **);**

**end** **if;**

**end** **process** sub\_process**;**

**end** **architecture;**

## 3.4 Accumulator

The accumulator sums the inputs on clock edges if it is enabled and if the counter has not already overflowed. When it receives the overflow signal from counter, the accumulator performs the last sum and then set data\_valid to 1 to advertise that the result is ready.

**library** ieee**;**

**use** ieee**.**std\_logic\_1164**.all;**

**use** ieee**.**numeric\_std**.all;**

**use** IEEE**.**std\_logic\_unsigned**.all;**

**entity** accumulator **is**

**generic** **(**

NBit**:** positive **:=** 16

**);**

**port(**

i**:** **in** std\_logic\_vector**(**NBit**-**1 **downto** 0**);**

rst**,**clk **:** **in** std\_logic**;**

en**:** **in** std\_logic**;**

counter\_ow**:** **in** std\_logic**;**

o**:** **out** std\_logic\_vector**(** NBit**-**1 **downto** 0 **);**

data\_valid**:** **out** std\_logic

**);**

**end** accumulator**;**

-- Sum the input at each clock if en = '0', set data\_valid to 1 when counter overflows

**architecture** beh **of** accumulator **is**

**signal** output\_oreg **:** std\_logic\_vector **(**NBit**-**1 **downto** 0**);**

**signal** data\_valid\_oreg **:** std\_logic**;**

**begin**

accumulator\_p**:** **process(**rst**,** clk**)**

**begin**

**if(**rst **=** '1'**)** **then**

output\_oreg **<=** **(others** **=>** '0'**);**

data\_valid\_oreg **<=** '0'**;**

**elsif(rising\_edge(**clk**))** **then**

**if(**en**=**'1' **and** data\_valid\_oreg**=**'0'**)** **then**

output\_oreg **<=** std\_logic\_vector**(**unsigned**(**output\_oreg**)** **+** unsigned**(**i**));**

-- check on counter overflow value to prevent the restart of summing

-- when overflow

**if(**counter\_ow **=** '1'**)** **then**

data\_valid\_oreg **<=** '1'**;**

**end** **if;**

**end** **if;**

**end** **if;**

**end** **process** accumulator\_p**;**

-- Mapping the output

data\_valid <= data\_valid\_oreg;

o <= output\_oreg;

**end** beh**;**

## 3.5 SAD

The code basically consists in the interconnection of the single components, in addition there is the process of padding addition used to expand the output of the subtractor in order to be the right size for the accumulator input. To start a new SAD calculation the restart signal must be set to 0 and then to 1 every time, in this way we clean all components and the circuit is ready to receive new images.

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**STD\_LOGIC\_ARITH**.all;**

**use** IEEE**.**STD\_LOGIC\_UNSIGNED**.all;**

**entity** SAD **is**

**generic** **(**

NBit\_input: positive **:=** 8**;**

NBit\_output: positive **:=** 16**;**

NBit\_counter **:** positive **:=** 8**;**

NBit\_subtractor **:** positive **:=** 8**;**

NBit\_DFF\_N**:** positive **:=** 8**;**

NBit\_accumulator**:** positive **:=** 16

**);**

**port** **(**

clk **:** **in** std\_logic**;**

rst **:** **in** std\_logic**;**

en **:** **in** std\_logic**;**

PA **:** **in** std\_logic\_vector**(**NBit\_input**-**1 **downto** 0**);** -- value of pixel A

PB **:** **in** std\_logic\_vector**(**NBit\_input**-**1 **downto** 0**);** -- value of pixel B

sad **:** **out** std\_logic\_vector**(**NBit\_output**-**1 **downto** 0 **);** -- output of |A - B|

data\_valid **:** **out** std\_logic -- set to 1 when SAD calculation is over

**);**

**end** SAD**;**

**architecture** Structural **of** SAD **is**

-- counter

**component** counter **is**

**generic** **(**

NBit **:** positive **:=** 8

**);**

**port** **(**

en**:** **in** std\_logic**;**

rst**,**clk **:** **in** std\_logic**;**

overflow**:** **out** std\_logic**;**

o**:** **out** std\_logic\_vector**(** NBit**-**1 **downto** 0 **)**

**);**

**end** **component;**

-- subtractor

**component** subtractor **is**

**generic** **(**

NBit**:** positive **:=** 8

**);**

**port** **(**

input1**:** **in** std\_logic\_vector **(**NBit**-**1 **downto** 0**);**

input2**:** **in** std\_logic\_vector **(**NBit**-**1 **downto** 0**);**

abs\_diff**:** **out** std\_logic\_vector **(**NBit**-**1 **downto** 0**)**

**);**

**end** **component;**

-- DFF\_N

**component** DFF\_N **is**

**generic(** NBit **:** positive **:=** 8**);**

**port(**

clk **:** **in** std\_logic**;**

a\_rst\_n **:** **in** std\_logic**;**

en **:** **in** std\_logic**;**

d **:** **in** std\_logic\_vector**(**NBit **-** 1 **downto** 0**);**

q **:** **out** std\_logic\_vector**(**NBit **-** 1 **downto** 0**)**

**);**

**end** **component;**

-- accumulator

**component** accumulator **is**

**generic** **(**

NBit**:** positive **:=** 16

**);**

**port(**

i**:** **in** std\_logic\_vector**(**NBit**-**1 **downto** 0**);**

rst**,**clk **:** **in** std\_logic**;**

en**:** **in** std\_logic**;**

counter\_ow**:** **in** std\_logic**;**

o**:** **out** std\_logic\_vector**(** NBit**-**1 **downto** 0 **);**

data\_valid**:** **out** std\_logic

**);**

**end** **component;**

-- padding used to format the subtractor output to the accumulator input, Default: padding is composed by 8 zeros in the default case

**constant** padding **:** std\_logic\_vector**(**NBit\_accumulator**-**NBit\_subtractor**-**1 **downto** 0**)** **:=** **(others** **=>** '0'**);**

-- Signals for interconnections

**signal** counter\_overflow**:** std\_logic**;**

**signal** counter\_output**:** std\_logic\_vector **(**NBit\_counter**-**1 **downto** 0**);**

**signal** subtractor\_output**:** std\_logic\_vector **(**NBit\_subtractor**-**1 **downto** 0 **);**

**signal** out\_PA**:** std\_logic\_vector **(**NBit\_DFF\_N**-**1 **downto** 0**);**

**signal** out\_PB**:** std\_logic\_vector **(**NBit\_DFF\_N**-**1 **downto** 0**);**

**signal** accumulator\_input**:** std\_logic\_vector **(**NBit\_accumulator**-**1 **downto** 0**);**

**begin**

COUNTER\_DEF**:** counter

**generic** **map(**NBit **=>** NBit\_counter**)** **port** **map(**en **=>** en**,** rst **=>** rst**,** clk **=>** clk**,** overflow

**=>** counter\_overflow**,** o **=>** counter\_output**);** -- if (en = '1') clock\_count = clock\_count

+ 1

PA\_REG**:** DFF\_N **generic** **map** **(**NBit **=>** NBit\_DFF\_N**)** **port** **map** **(**clk **=>** clk**,** a\_rst\_n **=>** rst**,**

en **=>** en**,** d **=>** PA**,** q **=>** out\_PA**);** -- PA

PB\_REG**:** DFF\_N **generic** **map** **(**NBit **=>** NBit\_DFF\_N**)** **port** **map** **(**clk **=>** clk**,** a\_rst\_n **=>** rst**,**

en **=>** en**,** d **=>** PB**,** q **=>** out\_PB**);** -- PB

SUBTRACTOR\_DEF**:** subtractor **generic** **map(**NBit **=>** NBit\_subtractor**)** **port** **map** **(**input1 **=>**

out\_PA**,** input2 **=>** out\_PB**,** abs\_diff **=>** subtractor\_output**);** -- |PA - PB|

ACCUMULATOR\_DEF**:** accumulator

**generic** **map(**NBit **=>** NBit\_accumulator**)**

**port** **map** **(**i **=>** accumulator\_input**,** clk **=>** clk**,** rst **=>** rst**,** en **=>** en**,** counter\_ow **=>**

counter\_overflow**,** o **=>** sad**,** data\_valid **=>** data\_valid**);** -- += |PA - PB|, data\_valid

-- Process to set the padding of the subtractor output

add\_padding\_process**:** **process(**subtractor\_output**)**

**begin**

-- concatenating NBit\_accumulator-NBit\_subtractor zeros to the subtractor output

accumulator\_input **<=** padding **&** subtractor\_output**;**

**end** **process** add\_padding\_process**;**

**end** Structural**;**

# 4. Test Plan

In this chapter we will see the verification of the circuit implemented in VHDL, to accomplish this aim we will use testbenches along with python scripts. The approach is again the bottom-up so starting from the verification of the components we then verify the entire implementation.

## 4.1 DFF\_N Testbench

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.all;**

**use** IEEE**.**std\_logic\_unsigned**.all;**

**use** IEEE**.**NUMERIC\_STD**.all;**

**entity** DFF\_N\_tb **is**

**end** DFF\_N\_tb**;**

**architecture** beh **of** DFF\_N\_tb **is**

--const def

**constant** clk\_period **:** time **:=** 100 ns**;**

**constant** NBit **:** positive **:=** 8**;**

--component dut

**component** DFF\_N

**generic(** NBit **:** positive **:=** 8**);**

**port(**

clk **:** **in** std\_logic**;**

a\_rst\_n **:** **in** std\_logic**;**

en **:** **in** std\_logic**;**

d **:** **in** std\_logic\_vector**(**NBit **-** 1 **downto** 0**);**

q **:** **out** std\_logic\_vector**(**NBit **-** 1 **downto** 0**)**

**);**

**end** **component;**

--signal of testbench

**signal** clk\_ext **:** std\_logic **:=** '0' **;**

**signal** rst\_ext **:** std\_logic **:=** '0' **;**

**signal** en\_ext **:** std\_logic **:=** '1'**;**

**signal** d\_ext **:** std\_logic\_vector**(**NBit**-**1 **downto** 0**)** **:=** **(others** **=>** '0'**);**

**signal** q\_ext **:** std\_logic\_vector**(**NBit**-**1 **downto** 0**);**

**signal** testing **:** boolean **:=** true **;**

--testbench

**begin**

clk\_ext **<=** **not** clk\_ext **after** clk\_period**/**2 **when** testing **else** '0'**;**

--component instantiation

dut**:** DFF\_N

**generic** **map(**

NBit **=>** NBit

**)**

**port** **map(**

clk **=>** clk\_ext**,**

a\_rst\_n **=>** rst\_ext**,**

en **=>** en\_ext**,**

d **=>** d\_ext**,**

q **=>** q\_ext

**);**

stimulus**:** **process**

**begin**

rst\_ext **<=** '1'**;**

**wait** **for** 30 ns**;**

rst\_ext **<=** '0'**;**

**wait** **until** **rising\_edge(**clk\_ext**);**

d\_ext **<=** b"00001000"**;**

**wait** **until** **rising\_edge(**clk\_ext**);**

d\_ext **<=** b"00001010"**;**

**wait** **until** **rising\_edge(**clk\_ext**);**

d\_ext **<=** b"00001001"**;**

**wait** **until** **rising\_edge(**clk\_ext**);**

rst\_ext **<=** '0'**;**

**wait** **until** **rising\_edge(**clk\_ext**);**

testing **<=** false**;**

**end** **process;**

**end** beh**;**

In this simple testbench we can see the simple functionality of a DFF\_N.

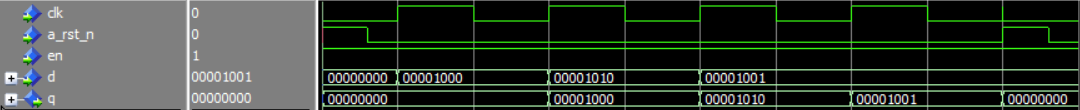


Figure 8: DFF\_N testbench result

## 4.2 Counter Testbench

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.all;**

**use** IEEE**.**std\_logic\_unsigned**.all;**

**use** IEEE**.**NUMERIC\_STD**.all;**

**entity** counter\_tb **is**

**end** counter\_tb**;**

**architecture** beh **of** counter\_tb **is**

--const def

**constant** clk\_period **:** time **:=** 100 ns**;**

**constant** NBit **:** positive **:=** 8**;**

--component dut

**component** counter

**generic** **(**

NBit **:** positive **:=** 8

**);**

**port** **(**

en**:** **in** std\_logic**;**

rst**,**clk **:** **in** std\_logic**;**

overflow**:** **out** std\_logic**;**

o**:** **out** std\_logic\_vector**(** NBit**-**1 **downto** 0 **)**

**);**

**end** **component;**

--signal of testbench

**signal** clk\_ext **:** std\_logic **:=** '0' **;**

**signal** rst\_ext **:** std\_logic **:=** '0' **;**

**signal** en\_ext **:** std\_logic **:=** '1'**;**

**signal** overflow\_ext**:** std\_logic**;**

**signal** o\_ext **:** std\_logic\_vector**(**NBit**-**1 **downto** 0**)** **;**

**signal** testing **:** boolean **:=** true **;**

--testbench

**begin**

clk\_ext **<=** **not** clk\_ext **after** clk\_period**/**2 **when** testing **else** '0'**;**

--component instantiation

dut**:** counter

**generic** **map(**

NBit **=>** NBit

**)**

**port** **map(**

clk **=>** clk\_ext**,**

en **=>** en\_ext**,**

rst **=>** rst\_ext**,**

overflow **=>** overflow\_ext**,**

o **=>** o\_ext

**);**

stimulus**:** **process**

**begin**

rst\_ext **<=** '1'**;**

**wait** **for** 30 ns**;**

rst\_ext **<=** '0'**;**

**wait** **for** 25600 ns**;**

testing **<=** false**;**

**end** **process;**

**end** beh**;**

The testbench of the counter should show the wave of the output increasing until it overflows after 256 clocks because the register is 8 bits.

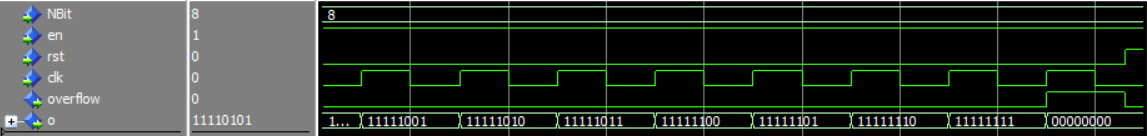


Figure 9: Counter Testbench result

## 4.3 Subtractor Tesbench

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.all;**

**use** IEEE**.**std\_logic\_unsigned**.all;**

**use** IEEE**.**NUMERIC\_STD**.all;**

**entity** subtractor\_tb **is**

**end** subtractor\_tb**;**

**architecture** beh **of** subtractor\_tb **is**

--const def

**constant** NBit **:** positive **:=** 8**;**

--component dut

**component** subtractor

**generic** **(**

NBit**:** positive **:=** 8

**);**

**port** **(**

input1**:** **in** std\_logic\_vector **(**NBit**-**1 **downto** 0**);**

input2**:** **in** std\_logic\_vector **(**NBit**-**1 **downto** 0**);**

abs\_diff**:** **out** std\_logic\_vector **(**NBit**-**1 **downto** 0**)**

**);**

**end** **component;**

--signal of testbench

**signal** input1\_ext **:** std\_logic\_vector**(**NBit**-**1 **downto** 0**)** **:=** b"00000001"**;**

**signal** input2\_ext **:** std\_logic\_vector**(**NBit**-**1 **downto** 0**)** **:=** b"00000010"**;**

**signal** abs\_diff\_ext **:** std\_logic\_vector**(**NBit**-**1 **downto** 0**);**

**signal** testing **:** boolean **:=** true**;**

--testbench

**begin**

--component instantiation

dut**:** subtractor

**port** **map(**

input1 **=>** input1\_ext**,**

input2 **=>** input2\_ext**,**

abs\_diff **=>** abs\_diff\_ext

**);**

stimulus**:** **process**

**begin**

**wait** **for** 10 ns**;**

input1\_ext **<=** b"00001000"**;**

input2\_ext **<=** b"00010100"**;**

**wait** **for** 10 ns**;**

testing **<=** false**;**

**end** **process;**

**end** beh**;**

The Subtractor testbench should show that absolute difference between the 2 inputs is calculated correctly.

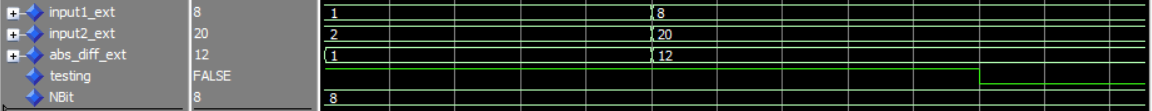


Figura 10: Subtractor Testbench result

## 4.4 Accumulator Testbench

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.all;**

**use** IEEE**.**std\_logic\_unsigned**.all;**

**use** IEEE**.**NUMERIC\_STD**.all;**

**entity** accumulator\_tb **is**

**end** accumulator\_tb**;**

**architecture** beh **of** accumulator\_tb **is**

--const def

**constant** clk\_period **:** time **:=** 100 ns**;**

**constant** NBit **:** positive **:=** 16**;**

**constant** NBit\_counter**:** positive **:=** 8**;**

--component dut

**component** accumulator

**generic** **(**

NBit**:** positive **:=** 16

**);**

**port(**

i**:** **in** std\_logic\_vector**(**NBit**-**1 **downto** 0**);**

rst**,**clk **:** **in** std\_logic**;**

en**:** **in** std\_logic**;**

counter\_ow**:** **in** std\_logic**;**

o**:** **out** std\_logic\_vector**(** NBit**-**1 **downto** 0 **);**

data\_valid**:** **out** std\_logic

**);**

**end** **component;**

--signal of testbench

**signal** i\_ext **:** std\_logic\_vector**(**NBit**-**1 **downto** 0**)** **:=** b"0000000000000001"**;**

**signal** clk\_ext **:** std\_logic **:=** '0' **;**

**signal** rst\_ext **:** std\_logic **:=** '0' **;**

**signal** en\_ext **:** std\_logic **:=** '1'**;**

**signal** counter\_ow\_ext **:** std\_logic **:=** '0'**;**

**signal** o\_ext **:** std\_logic\_vector**(**NBit**-**1 **downto** 0**);**

**signal** data\_valid\_ext **:** std\_logic**;**

**signal** testing **:** boolean **:=** true**;**

--testbench

**begin**

clk\_ext **<=** **not** clk\_ext **after** clk\_period**/**2 **when** testing **else** '0'**;**

--component instantiation

dut**:** accumulator

**port** **map(**

i **=>** i\_ext**,**

clk **=>** clk\_ext**,**

rst **=>** rst\_ext**,**

en **=>** en\_ext**,**

counter\_ow **=>** counter\_ow\_ext**,**

o **=>** o\_ext**,**

data\_valid **=>** data\_valid\_ext

**);**

-- accumulator must stop summing at b"0000000010000000"=256

stimulus**:** **process**

**begin**

rst\_ext **<=** '1'**;**

**wait** **for** 30 ns**;**

rst\_ext **<=** '0'**;**

**wait** **for** 25500 ns**;**

counter\_ow\_ext **<=** '1'**;**

**wait** **until** **rising\_edge(**clk\_ext**);**

**wait** **until** **rising\_edge(**clk\_ext**);**

**wait** **until** **rising\_edge(**clk\_ext**);**

testing **<=** false**;**

**wait** **until** **rising\_edge(**clk\_ext**);**

**end** **process;**

**end** beh**;**

The testbench of the accumulator should show the counting value reaching 256 and then stop because of counter overflow (in this case simulated with an external signal). The input at the accumulator is b"0000000000000001"**.**

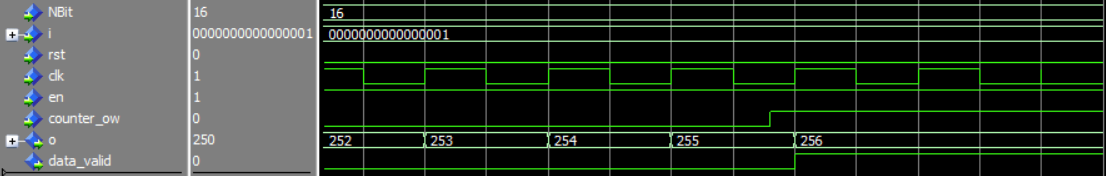


Figure 11: Accumulator testbench result

## 4.5 SAD Testbench

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.all;**

**use** IEEE**.**std\_logic\_unsigned**.all;**

**use** IEEE**.**NUMERIC\_STD**.all;**

**entity** SAD\_tb **is**

**end** SAD\_tb**;**

**architecture** beh **of** SAD\_tb **is**

--const def

**constant** clk\_period **:** time **:=** 100 ns**;**

**constant** NBit\_input **:** positive **:=** 8**;**

**constant** NBit\_output**:** positive **:=** 16**;**

--component dut

**component** SAD

**generic** **(**

-- when counter reaches this value data\_valid is set to 1.

counter\_threshold**:** positive **:=** 256**;**

NBit\_input**:** positive **:=** 8**;**

NBit\_output**:** positive **:=** 16

**);**

**port** **(**

clk **:** **in** std\_logic**;**

rst **:** **in** std\_logic**;**

en **:** **in** std\_logic**;**

PA **:** **in** std\_logic\_vector**(**NBit\_input**-**1 **downto** 0**);** -- value of pixel A

PB **:** **in** std\_logic\_vector**(**NBit\_input**-**1 **downto** 0**);** -- value of pixel B

sad **:** **out** std\_logic\_vector**(**NBit\_output**-**1 **downto** 0 **);** -- output of |A - B|

data\_valid **:** **out** std\_logic -- set to 1 when SAD calculation is over

**);**

**end** **component;**

--signal of testbench

**signal** clk\_ext **:** std\_logic **:=** '0' **;**

**signal** rst\_ext **:** std\_logic **:=** '0' **;**

**signal** en\_ext **:** std\_logic **:=** '1'**;**

**signal** PA\_ext **:** std\_logic\_vector**(**NBit\_input**-**1 **downto** 0**)** **:=** b"00000001"**;**

**signal** PB\_ext **:** std\_logic\_vector**(**NBit\_input**-**1 **downto** 0**)** **:=** b"00000010"**;**

**signal** sad\_ext **:** std\_logic\_vector**(**NBit\_output**-**1 **downto** 0**)** **;**

**signal** data\_valid\_ext **:** std\_logic**;**

**signal** testing **:** boolean **:=** true **;**

--testbench

**begin**

clk\_ext **<=** **not** clk\_ext **after** clk\_period**/**2 **when** testing **else** '0'**;**

--component instantiation

dut**:** SAD

**port** **map(**

clk **=>** clk\_ext**,**

rst **=>** rst\_ext**,**

en **=>** en\_ext**,**

PA **=>** PA\_ext**,**

PB **=>** PB\_ext**,**

sad **=>** sad\_ext**,**

data\_valid **=>** data\_valid\_ext

**);**

stimulus**:** **process** --no sensitivity list

**begin**

rst\_ext **<=** '1'**;**

**wait** **for** 30 ns**;**

rst\_ext **<=** '0'**;**

-- standard working experiment, the result must be the sum of clocks

**wait** **until** **rising\_edge(**clk\_ext**);**

PA\_ext **<=** b"00000011"**;**

PB\_ext **<=** b"00000100"**;**

**wait** **until** **rising\_edge(**clk\_ext**);**

PA\_ext **<=** b"00000101"**;**

PB\_ext **<=** b"00000110"**;**

**wait** **for** 26000 ns**;**

testing **<=** false**;**

**end** **process;**

**end** beh**;**

The total value of sad should be 256 because the counter is 8 bits and overflows at that value, the circuit then should set data\_valid.

Immagine che contiene testo, verde, tabellonesegnapunti

Descrizione generata automaticamente

Figura 12: SAD Standard Testbench result

We can use a similar testbench to verify the SAD behavior in the case of NxN images, for example 64x64. This time we have 64x64 couples, so the counter must be 12 bits (), and the size of the accumulator output should be 20 bits because the highest value is . The circuit works again as expected:

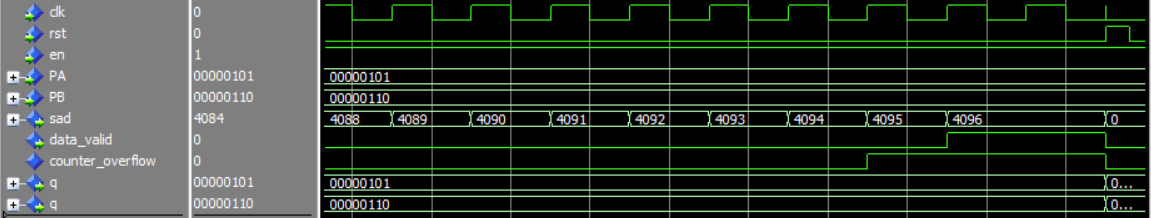


Figure 12.5: SAD NxN Testbench result

## 4.6 SAD Freeze Testbench

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.all;**

**use** IEEE**.**std\_logic\_unsigned**.all;**

**use** IEEE**.**NUMERIC\_STD**.all;**

**entity** SAD\_freeze\_tb **is**

**end** SAD\_freeze\_tb**;**

**architecture** beh **of** SAD\_freeze\_tb **is**

--const def

**constant** clk\_period **:** time **:=** 100 ns**;**

**constant** NBit\_input **:** positive **:=** 8**;**

**constant** NBit\_output**:** positive **:=** 16**;**

--component dut

**component** SAD

**generic** **(**

-- when counter reaches this value data\_valid is set to 1.

counter\_threshold**:** positive **:=** 256**;**

NBit\_input**:** positive **:=** 8**;**

NBit\_output**:** positive **:=** 16

**);**

**port** **(**

clk **:** **in** std\_logic**;**

rst **:** **in** std\_logic**;**

en **:** **in** std\_logic**;**

PA **:** **in** std\_logic\_vector**(**NBit\_input**-**1 **downto** 0**);** -- value of pixel A

PB **:** **in** std\_logic\_vector**(**NBit\_input**-**1 **downto** 0**);** -- value of pixel B

sad **:** **out** std\_logic\_vector**(**NBit\_output**-**1 **downto** 0 **);** -- output of |A - B|

data\_valid **:** **out** std\_logic -- set to 1 when SAD calculation is over

**);**

**end** **component;**

--signal of testbench

**signal** clk\_ext **:** std\_logic **:=** '0' **;**

**signal** rst\_ext **:** std\_logic **:=** '0' **;**

**signal** en\_ext **:** std\_logic **:=** '1'**;**

**signal** PA\_ext **:** std\_logic\_vector**(**NBit\_input**-**1 **downto** 0**)** **:=** b"00000001"**;**

**signal** PB\_ext **:** std\_logic\_vector**(**NBit\_input**-**1 **downto** 0**)** **:=** b"00000010"**;**

**signal** sad\_ext **:** std\_logic\_vector**(**NBit\_output**-**1 **downto** 0**)** **;**

**signal** data\_valid\_ext **:** std\_logic**;**

**signal** testing **:** boolean **:=** true **;**

--testbench

**begin**

clk\_ext **<=** **not** clk\_ext **after** clk\_period**/**2 **when** testing **else** '0'**;**

--component instantiation

dut**:** SAD

**port** **map(**

clk **=>** clk\_ext**,**

rst **=>** rst\_ext**,**

en **=>** en\_ext**,**

PA **=>** PA\_ext**,**

PB **=>** PB\_ext**,**

sad **=>** sad\_ext**,**

data\_valid **=>** data\_valid\_ext

**);**

stimulus**:** **process** --no sensitivity list

**begin**

-- report "The value of 'testing' is " & boolean'image(testing);

rst\_ext **<=** '1'**;**

**wait** **for** 30 ns**;**

rst\_ext **<=** '0'**;**

-- freeze experiment, the sum is freezed for 2 clocks and then resumed

-- In the default case the result is 257 because we have introduce a

-- couple with a difference of 2 in the pixel value

**wait** **until** **rising\_edge(**clk\_ext**);**

PA\_ext **<=** b"00000100"**;**

PB\_ext **<=** b"00000110"**;**

**wait** **until** **rising\_edge(**clk\_ext**);**

en\_ext **<=** '0'**;**

**wait** **until** **rising\_edge(**clk\_ext**);**

**wait** **until** **rising\_edge(**clk\_ext**);**

en\_ext **<=** '1'**;**

PA\_ext **<=** b"00000011"**;**

PB\_ext **<=** b"00000100"**;**

**wait** **for** 26000 ns**;**

testing **<=** false**;**

**end** **process;**

**end** beh**;**

This testbench is to verify if the freeze functionality works, to do that we can stop the accumulation of absolute differences for one clock. The first clock sums the absolute difference between b"00000110"and b"00000110"**,** that is 2, and the for the remaining time after the freezing it sums the absolute difference between b"00000011” and b"00000010"that is 1. If the circuit works the total sad until overflow should be 257.

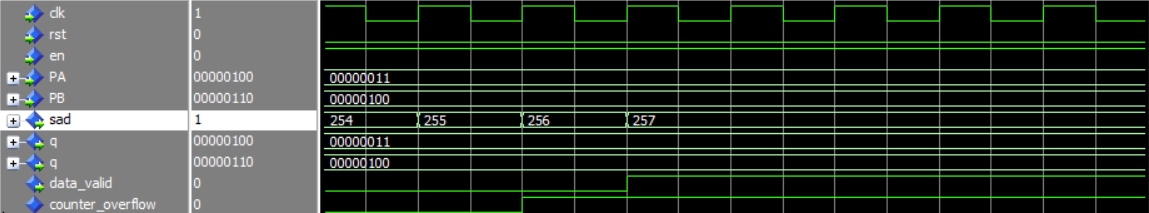


Figura 13: SAD Freeze Tesbench result

## 4.7 SAD Testbench with python script

The last test is to verify the SAD calculation on a real case, we can generate a random set of pixel values for 2 different images and then calculate the total sad value. Two python script has been used to generate the sequences and then calculate the total sad, after that the same sequence is fed into the circuit to verify if the sum of absolute difference coincides.

default\_test.py:

This script is to generate the sequences of random pixels and to calculate the SAD values after each sum.

import random  
  
# generating "amount" random pixel values  
def random\_pixel\_values(id, amount, max\_pixel\_value):  
 randomlist = ""  
 randomlist\_integer = []  
 for i in range(0, amount):  
 n = random.randint(0, max\_pixel\_value)  
 randomlist += str(n) +"\n"  
 randomlist\_integer.append(n)  
  
 with open("./../../SAD\_Project\_Files/tb/test/default\_list"+str(id)+".txt", 'w') as f:  
 f.write(randomlist)  
  
 return randomlist\_integer  
  
# calculate the sum of absolute differences between the 2 vectors  
def SAD(PAs, PBs):  
  
 if len(PAs) != len(PBs):  
 return -1  
  
 SAD\_str = ""  
 sad = 0  
 for i in range (0, len(PAs)):  
 diff = abs(PAs[i] - PBs[i])  
 sad += diff  
 SAD\_str += str(sad) + "\n"  
  
 # write the resulting vector sad on file  
 with open("./../../SAD\_Project\_Files/tb/test/default\_list\_sad.txt", 'w') as f:  
 f.write(SAD\_str)  
 return sad  
  
def main():  
 # default: images of 16x16 pixels = 256 pixels  
 # default: monochrome [0, 255] pixel  
 amount = 256  
 max\_pixel\_value = 255  
  
 # generating PAs and PBs random values  
 PAs = random\_pixel\_values(1, amount, max\_pixel\_value)  
 PBs = random\_pixel\_values(2, amount, max\_pixel\_value)  
  
 # calculating the result of SAD  
 sad = SAD(PAs, PBs)  
  
 if sad == -1:  
 print("sad not generated correctly")  
 else:  
 print("sad generated correctly: "+str(sad))  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

test\_sad.py:

This script is to compare the real sad calculated in the previous script by python and the one produced by the circuit (simulated one), this is obtained exporting the values of the wave as an event list.

# load the simulated sad and extract data  
def load\_simulated\_sad():  
 with open("./../../SAD\_Project\_Files/tb/test/list.txt", 'r') as f:  
 sim\_sad = f.readlines()  
  
 #reading only the odd lines  
 sim\_sad = [v for i, v in enumerate(sim\_sad) if i % 2 != 0]  
  
 #dropping element 0, 1 and last  
 sim\_sad = sim\_sad[2:]  
 sim\_sad = sim\_sad[:-1]  
  
 #extract the value from each line and convert it to an integer  
 for i in range(0, len(sim\_sad)):  
 sim\_sad[i] = sim\_sad[i].split(" ")[1].replace("\n", "")  
 sim\_sad[i] = int(sim\_sad[i], 2)  
  
 #extract the value of sad from the line and then convert it to an integer  
 return sim\_sad  
  
# load real sad  
def load\_real\_sad():  
 with open("./../../SAD\_Project\_Files/tb/test/default\_list\_sad.txt", 'r') as f:  
 real\_sad = f.readlines()  
 for i in range(0, len(real\_sad)):  
 real\_sad[i] = int(real\_sad[i])  
 return real\_sad  
  
# calculate the error on the 2 list, return 0 if not ok, 1 if ok  
def check\_error(i1, i2):  
 if len(i1) != len(i2):  
 print("error")  
 return False  
  
 for i in range(0, len(i1)):  
 if (i1[i] - i2[i]) != 0:  
 return False  
  
 return True  
  
def main():  
 sim\_sad = load\_simulated\_sad()  
 real\_sad = load\_real\_sad()  
  
 print(real\_sad)  
 print(sim\_sad)  
  
 if check\_error(sim\_sad, real\_sad):  
 print("no errors, simulation success")  
 else:  
 print("errors, simulation failure")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

SAD\_final\_tb.vhd:

This script reads the sequence of pixel values produced by default\_test.py and fed them sequentially into the circuit.

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.all;**

**use** IEEE**.**std\_logic\_unsigned**.all;**

**use** IEEE**.**NUMERIC\_STD**.all;**

**use** IEEE**.**std\_logic\_textio**.all;**

**use** std**.**textio**.all;**

**entity** SAD\_final\_tb **is**

**end** SAD\_final\_tb**;**

**architecture** beh **of** SAD\_final\_tb **is**

--const def

**constant** clk\_period **:** time **:=** 100 ns**;**

**constant** NBit\_input **:** positive **:=** 8**;**

**constant** NBit\_output**:** positive **:=** 16**;**

**constant** N\_pixels **:** positive **:=** 256**;** -- default: 16x16 = 256 pixels

--component dut

**component** SAD

**generic** **(**

-- when counter reaches this value data\_valid is set to 1.

counter\_threshold**:** positive **:=** 256**;**

NBit\_input**:** positive **:=** 8**;**

NBit\_output**:** positive **:=** 16

**);**

**port** **(**

clk **:** **in** std\_logic**;**

rst **:** **in** std\_logic**;**

en **:** **in** std\_logic**;**

PA **:** **in** std\_logic\_vector**(**NBit\_input**-**1 **downto** 0**);** -- value of pixel A

PB **:** **in** std\_logic\_vector**(**NBit\_input**-**1 **downto** 0**);** -- value of pixel B

sad **:** **out** std\_logic\_vector**(**NBit\_output**-**1 **downto** 0 **);** -- output of |A - B|

data\_valid **:** **out** std\_logic -- set to 1 when SAD calculation is over

**);**

**end** **component;**

--signal of testbench

**signal** clk\_ext **:** std\_logic **:=** '0' **;**

**signal** rst\_ext **:** std\_logic **:=** '0' **;**

**signal** en\_ext **:** std\_logic **:=** '1'**;**

**signal** PA\_ext **:** std\_logic\_vector**(**NBit\_input**-**1 **downto** 0**)** **:=** b"00000000"**;**

**signal** PB\_ext **:** std\_logic\_vector**(**NBit\_input**-**1 **downto** 0**)** **:=** b"00000000"**;**

**signal** sad\_ext **:** std\_logic\_vector**(**NBit\_output**-**1 **downto** 0**)** **;**

**signal** data\_valid\_ext **:** std\_logic**;**

**signal** testing **:** boolean **:=** true **;**

--testbench

**begin**

clk\_ext **<=** **not** clk\_ext **after** clk\_period**/**2 **when** testing **else** '0'**;**

--component instantiation

dut**:** SAD

**port** **map(**

clk **=>** clk\_ext**,**

rst **=>** rst\_ext**,**

en **=>** en\_ext**,**

PA **=>** PA\_ext**,**

PB **=>** PB\_ext**,**

sad **=>** sad\_ext**,**

data\_valid **=>** data\_valid\_ext

**);**

stimulus**:** **process** --no sensitivity list

**file** file\_handler1 **:** text **open** read\_mode **is** "../tb/test/default\_list1.txt"**;**

**file** file\_handler2 **:** text **open** read\_mode **is** "../tb/test/default\_list2.txt"**;**

**variable** row **:** line**;**

**variable** v\_data\_read **:** integer**;**

**begin**

rst\_ext **<=** '1'**;**

**wait** **for** 30 ns**;**

rst\_ext **<=** '0'**;**

-- reading the testing values from file

**for** i **in** 0 **to** N\_pixels**-**1 **loop**

**if** i **>** 0 **then**

**wait** **until** **rising\_edge(**clk\_ext**);**

**end** **if;**

**if** **(not** **endfile(**file\_handler1**)** **and** **not** **endfile(**file\_handler2**))** **then**

**readline(**file\_handler1**,** row**);**

**read(**row**,** v\_data\_read**);**

PA\_ext **<=** std\_logic\_vector**(to\_unsigned(**v\_data\_read**,** PA\_ext'**length));**

**readline(**file\_handler2**,** row**);**

**read(**row**,** v\_data\_read**);**

PB\_ext **<=** std\_logic\_vector**(to\_unsigned(**v\_data\_read**,** PB\_ext'**length));**

**end** **if;**

**end** **loop;**

**wait** **for** 1000 ns**;**

testing **<=** false**;**

**end** **process;**

**end** beh**;**

Immagine che contiene testo, tabellonesegnapunti

Descrizione generata automaticamente

Figure 14: SAD Final Testbench result

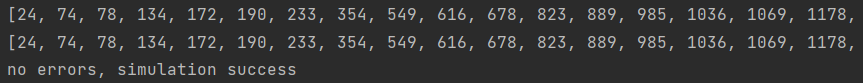
After exporting the sad values of the wave as an event list and save the file as “list.txt” in SAD\_Project\_Files/tb/test we can run test\_sad.py to verify if the calculated value of the script and the simulated one coincide.

default\_test.py output:

The script saves the 2 sequences and the evolution of the sad value, the files are called: default\_list1.txt, default\_list2.txt, default\_list\_sad.txt.



test\_sad.py output:

 … Immagine che contiene testo

Descrizione generata automaticamente

We can see that the SAD evolution is the same, so we can state that the circuit works correctly.

# 5. Logic Synthesis for the standard case (16x16 images)

In this chapter we discuss the Logic Synthesis of the circuit performed on Xilinx Vivado for xc7z010clg400-1.

## 5.1 RTL Analysis

The first step is to perform the RTL analysis, the output of this phase is the schematic design of the circuit:

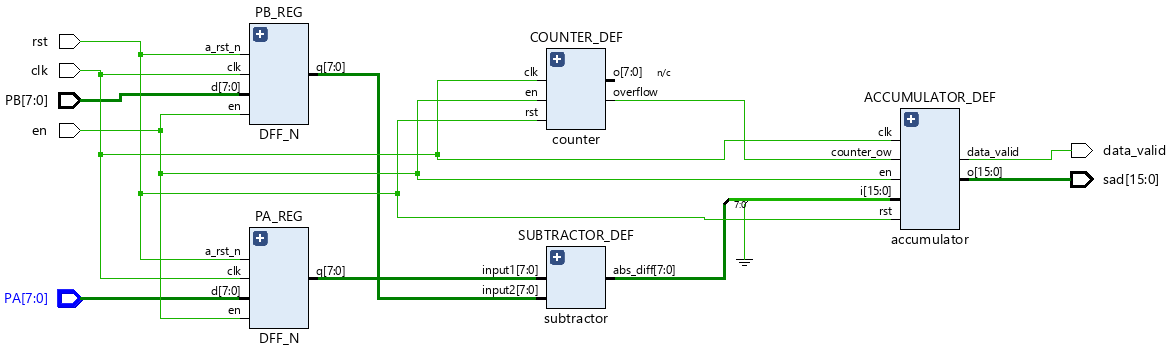


Figure 15: Elaborated Design

## 5.2 Synthesis

The synthesis phase consists in constructing the physical system from the RTL description. No warnings or errors are shown after the running.

Immagine che contiene testo

Descrizione generata automaticamente

Figure 16: Synthesis output

### 5.2.1 Clock period setting

To set the clock period we start from a low value of frequency, like 100 MHz (10 ns), and start increasing it until the WNS reaches a negative value. The WNS must be positive otherwise setup/hold violations occurs, so we take the maximum frequency as the highest possible without having a positive value.

At the value of 167.504 Mhz (5.97ns) we get a WNS of 0.016 ns (as shown in the figure), this means that we are almost there, setting higher frequencies is dangerous for the circuit.

Immagine che contiene testo

Descrizione generata automaticamente

Figure 17: Design Timing Summary with clock period set at 6 ns

In the implementation the slack would be lower than this due to place and route, so the better solution is to maintain a bit of margin error using a slightly lower frequency.

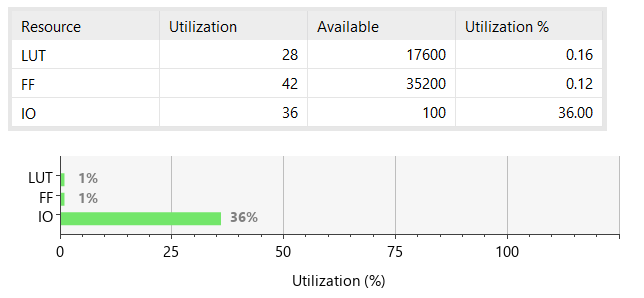


Figure 18: Estimated report of utilization

We can see that the main resource in percentage that has been utilized is I/O, from the scheme we can see that we used 28 LUTS, 42 FF and 36 IO ports.

### 5.2.2 Critical Path

The WNS is determined by the critical path of the architecture:

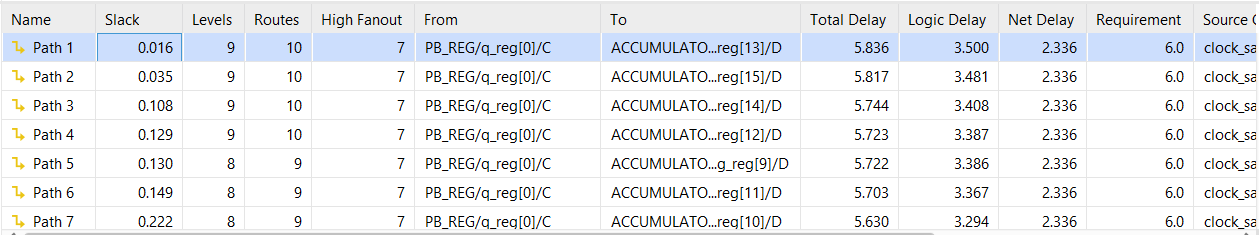


Figure 19: Max delay path

The path goes from PB\_reg to the output\_reg of the accumulator as shown in this figure:

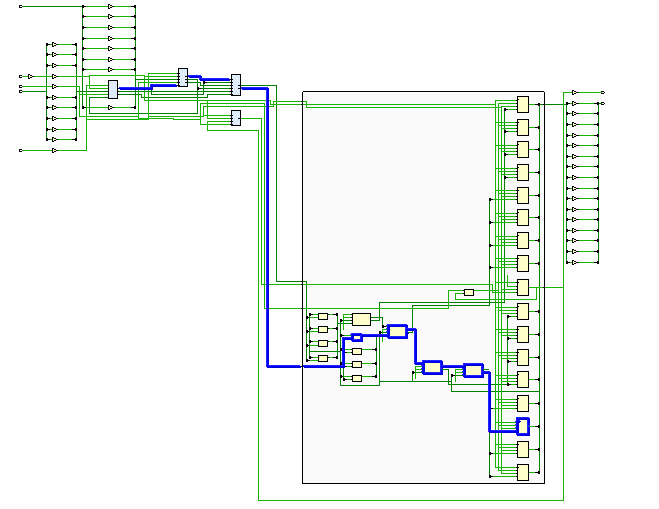


Figure 20: Critical Path

### 5.2.3 Power consumption

We can see that the power consumption

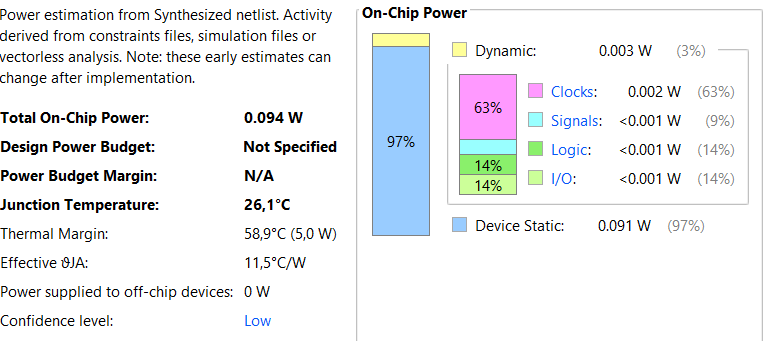


Figure 21: Power consumption estimation report

A total of 0.094 W is needed, the main contribution is given by static power consumption, so where there is no switching activity, and it is around 97% of the total.

## 6.1 RTL analysis in 64x64 images case

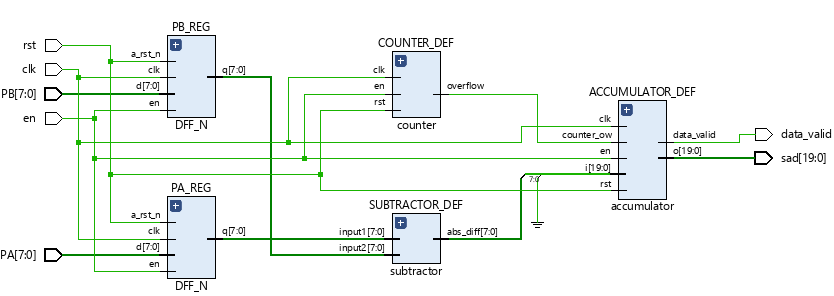


Figure 22: Schematic design 64x64 case

## 6.2 Synthesis on 64x64 images case

### 6.2.1 Clock period setting

We can see the timing report indicating a violation, a clock frequency of 167.504 Mhz (5.97ns) is too high in this case. This is caused by the increase in the size of inputs and outputs in the circuit.

Immagine che contiene testo

Descrizione generata automaticamente

Figure 23: Timing Report with violation 64x64 case

Observing the total delay in the path report we can see that it is worst than before, it goes from 5.836 ns to 5.950 ns:

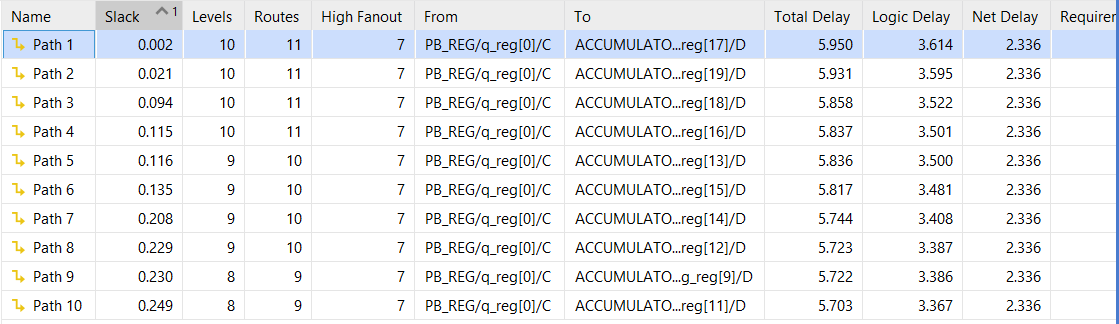


Figure 24: Critical path 64x64 case

We must set a lower frequency to avoid this problem, again we can try to decrease the frequency and find the bound value where the slack becomes positive. The limit value of frequency is 164,74 Mhz (6.07 ns):

Immagine che contiene testo

Descrizione generata automaticamente

Figure 25: Timing Report 64x64 case

Again, the clock should be set on a lower value of frequency to have a bit of margin error.

### 6.2.2 Utilization report

From this report we can see that the major increase is in IO resources because of bigger inputs to the counter and accumulator components.

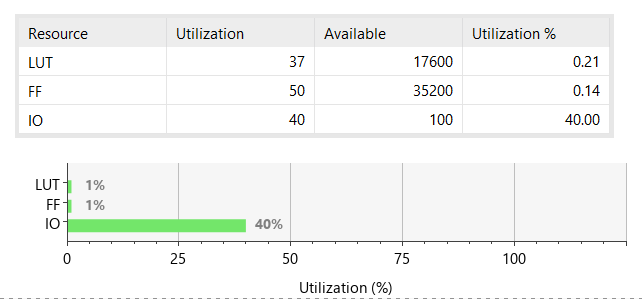


Figure 26: Utilization report 64x64 case

### 6.2.3 Power consumption report

The consumption report is pretty much the same as expected.

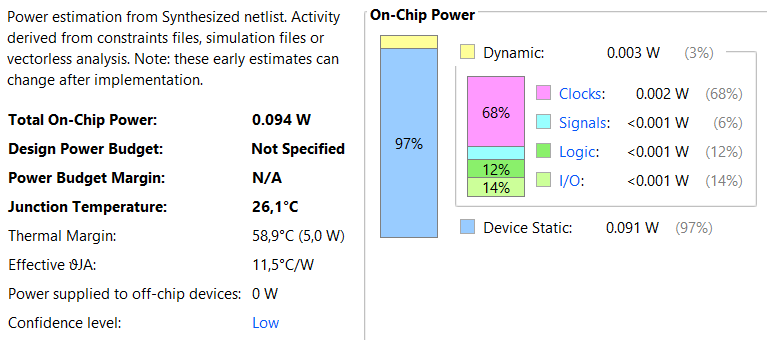


Figure 27: Power consumption report 64x64 case

# 7. Conclusions

We have achieved our goal to develop a circuit for the sum of absolute differences on NxN block of images, this is verified by the test presented before that show the expected behavior.

By the comparison of the synthesis of 16x16 case and 64x64 case we understand the importance of setting a different clock frequency, the bigger will be the images the lower should be the frequency to avoid violations.

From the performance view we have seen that the design leads to a low usage of resources and a low power consumption that can increase if the characteristic of the blocks change.

As a result, we can deploy an implementation of the design to a ZyBo board and perform the SAD calculation in different field of applications like the ones presented in the introduction.

# References

“Efficient sum of absolute difference computation on FPGAs” (2016), Martin Kumm; Marco Kleinlein; Peter Zipf