

# SAD calculation of two monochrome images

Francesco Urbani

May 27, 2018

## Contents

<b>1</b>	<b>Project specifications</b>	<b>4</b>
<b>2</b>	<b>Algorithm description</b>	<b>4</b>
<b>3</b>	<b>Architecture</b>	<b>4</b>
<b>4</b>	<b>VHDL</b>	<b>6</b>
4.1	Test benches . . . . .	6
4.2	Compiling instructions . . . . .	7
<b>5</b>	<b>Vivado</b>	<b>7</b>

## List of Figures

1	SAD architecture . . . . .	5
---	----------------------------	---

## 1 Project specifications

Design a synchronous digital system that calculates the SAD value, defined as the sum of the magnitude of the differences of each pixel of two monochrome images, image A and image B. Each pixel's value is stored in N-bits and each images has  $px$  pixels per side. The circuit has a clock, a reset and an enable signals as input together with the signals PA and PB representing one pixel of each image getting into the system from the outside, sequentially. The circuit has a M-bits output signal called SAD, where:

$$2^M \geq (2^N - 1) \cdot px^2 \quad (1)$$

and a DATA\_VALID signal which is set when the SAD calculation is completed.

When the reset is active, SAD and DATA\_VALID go to 0. If the enable signal is zero the system does not change its state.

## 2 Algorithm description

In order to meet the specification's constraints, I've designed – using the hardware description language VHDL – a synchronous digital circuit, that takes as input two monochrome images modeled as two N-bits signals and it outputs the value of the SAD once the calculation was over.

The signals representing the images enters into two PIPO registers whose output is then subtracted and made positive.

The output of the subtraction is then fed into a M-bits phase accumulator which increases its value at every clock cycle.

The output of the phase accumulator represents the value of the *current* SAD.

## 3 Architecture

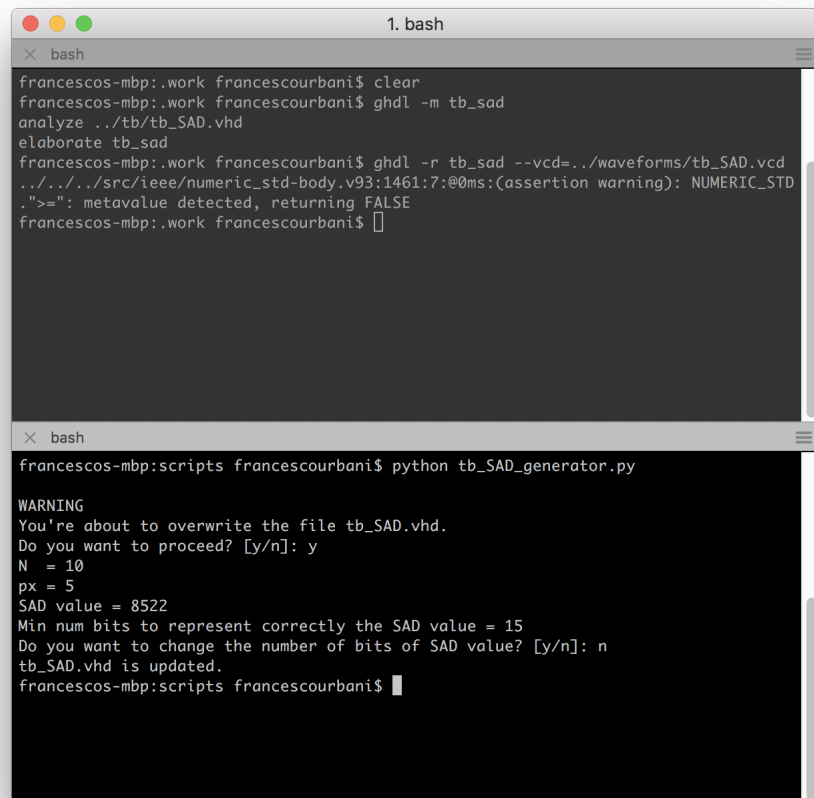
The architecture chosen is the following:

The images are fed into the system by means of two signals, called PA and PB, whose size is N, i.e. the number of bits representing the pixel grade on a scale from 0 (i.e. 100% black) to  $2^N - 1$ , typically 255 (i.e. 100% white).

Being the system synchronous, it works on the rising-edge clock.

The other two inputs are the reset (active high) and the enable, that prevents the SAD value to be updated when it's low.

The two outputs are SAD and DATA\_VALID.



The image shows two terminal windows. The top window, titled '1. bash', shows the following commands and output:

```
francescos-mbp:work francescoubani$ clear
francescos-mbp:work francescoubani$ ghdl -m tb_sad
analyze ../tb/tb_SAD.vhd
elaborate tb_sad
francescos-mbp:work francescoubani$ ghdl -r tb_sad --vcd=../waveforms/tb_SAD.vcd
../../../../src/ieee/numeric_std-body.v93:1461:7:@0ms:(assertion warning): NUMERIC_STD
.">=": metavalue detected, returning FALSE
francescos-mbp:work francescoubani$
```

The bottom window, also titled 'bash', shows the execution of a Python script:

```
francescos-mbp:scripts francescoubani$ python tb_SAD_generator.py

WARNING
You're about to overwrite the file tb_SAD.vhd.
Do you want to proceed? [y/n]: y
N = 10
px = 5
SAD value = 8522
Min num bits to represent correctly the SAD value = 15
Do you want to change the number of bits of SAD value? [y/n]: n
tb_SAD.vhd is updated.
francescos-mbp:scripts francescoubani$
```

Figure 1: SAD architecture

SAD represent the actual information we are interested in, and it's defined as the sum of the absolute differences of each pixel of the two images. Therefore it's value is updated at each clock cycle and its only freezed when the enable is low or the calculation is completed.

DATA\_VALID goes high when the SAD calculation is completed and stays high until the system is reset.

## 4 VHDL

A structural approach has been followed. After writing the `.vhd` files describing all the components that I would have been using in the top-level component, I've simply wired them up together to create the final design. The only two components described with a *behavioral* approach have been `counter.vhd` and `subtractor.vhd`.

### 4.1 Test benches

In order to create a relevant test bench for `SAD.vhd` that would replicate an actual real-life situation, which means two images where every pixel is different to the adjacent pixel, a Python script has been developed.

The script (see `./scripts/tb_SAD_generator.py`) takes as input the number of bits representing each pixel value ( $N$ ) and the number of pixels/side, i.e. the square root of the total number of pixels composing each image ( $px$ ). It also calculates the minimum number of bits necessary to store the SAD value without losing information which is:

$$\lceil \log_2((2^N - 1) \cdot px^2) \rceil \quad (2)$$

and asks the user whether he wants to increase it.

After that the script generates two square matrices with integer elements in the range  $[0, 2^N)$ , calculates a third matrix whose elements are the magnitude of the difference of the former two matrices' elements and eventually calculates the overall sum of it's elements, i.e. the expected SAD value.

Afterwards it generates the test bench file `tb_SAD.vhd` adherent to the model so that comparing the model to the waveform result would be straightforward.

## 4.2 Compiling instructions

I've used the open-source compiler GHDL to compile the VHDL code in conjunction with the waveform viewer Scansion which is a nice alternative to GTKWave for macOS.

Once organized the folder like this:

```
.
├── .work
├── scripts
├── src
├── tb
├── vivado
└── waveforms
```

move into `.work/` and from your terminal type:

```
ghdl -i ../src/*
ghdl -i ../tb/*
```

to import all the `.vhd` files into the working directory. Then:

```
ghdl -m sad
```

to compile the top-level component. That will compile all the components instantiated into the top-level component as well. Then type:

```
ghdl -m tb_sad
```

to compile the test bench of the top-level component. And finally:

```
ghdl -r tb_sad --vcd=../waveforms/tb_SAD.vcd
```

to create the waveform file `tb_SAD.vcd` which can be afterwards opened with one of those two above mentioned waveform viewers.

## 5 Vivado