**Gamma\_Aff IP**

**Inputs**

1. S00\_AXI
   1. [IP: axi\_interconnect\_1] M03\_AXI → S00\_AXI
2. ready\_Params\_0 [1-bit]
   1. [IP: ParametersMulti\_0] param\_done\_0 → ready\_Params\_0
3. ready\_Grad\_0 [1-bit]
   1. [IP: GradientsMulti\_1] grad\_done\_0 → ready\_Grad\_0
4. ready\_Coord\_0 [1-bit]
   1. [IP: SubsetCoordsMulti] sub\_done\_0 → ready\_Coord\_0
5. num\_of\_subsets\_0 [32-bits]
   1. [IP: ParametersMulti\_0] num\_of\_subsets\_0 → num\_of\_subsets\_0
6. optimization\_method\_0 [32-bits]
   1. [IP: ParametersMulti\_0] optimization\_method\_0 → optimization\_method\_0
7. correlation\_routine\_0 [32-bits]
   1. [IP: ParametersMulti\_0] correlation\_routine\_0 → correlation\_routine\_0
8. num\_pixels\_0 [32-bits]
   1. [IP: ParametersMulti\_0] num\_of\_pxl\_0 → num\_pixels\_0
9. cx\_0 [32-bits]
   1. [IP: Gam\_Interface\_0] gam\_cx\_0 → cx\_0
10. cy\_0 [32-bits]
    1. [IP: Gam\_Interface\_0] gam\_cy\_0 → cy\_0
11. base\_address\_0 [32-bits]
    1. [IP: Gam\_Interface\_0] base\_addr\_out\_0 → base\_address\_0
12. num\_pxl\_Int\_0 [32-bits]
    1. [IP: Gam\_Interface\_0] num\_pxl\_Int\_out\_0 → num\_pxl\_Int\_0
13. num\_pxl\_FP\_0 [32-bits]
    1. [IP: Gam\_Interface\_0] num\_pxl\_FP\_out\_0 → num\_pxl\_FP\_0
14. dout\_ref\_ints\_0 [32-bits]
    1. [IP: MUX\_0] ref\_img\_out\_0 → dout\_ref\_ints\_0
15. dout\_def\_ints\_0 [32-bits]
    1. [IP: MUX\_0] def\_img\_out\_0 → dout\_def\_ints\_0
16. dout\_grad\_x\_0 [32-bits]
    1. [IP: blk\_mem\_gen\_3] doutb → dout\_grad\_x\_0
17. dout\_grad\_y\_0 [32-bits]
    1. [IP: blk\_mem\_gen\_4] doutb → dout\_grad\_y\_0
18. x\_0 [32-bits]
    1. [IP: blk\_mem\_gen\_6] doutb → x\_0
19. y\_0 [32-bits]
    1. [IP: blk\_mem\_gen\_7] doutb → y\_0
20. gam\_interface\_done\_0 [1-bit]
    1. [IP: Gam\_Interface\_0] gam\_interface\_done\_0 → gam\_interface\_done\_0
21. s00\_axi\_aclk [1-bit]
    1. [IP: zynq\_ultra\_ps\_e\_0] pl\_clk0 → s00\_axi\_aclk
22. s00\_axi\_aresetn [1-bit]
    1. [IP: rst\_ps8\_0\_100M] peripheral\_aresetn → s00\_axi\_aresetn

**Associated IPs (inputs):**

1. zynq\_ultra\_ps\_e\_0
2. rst\_ps8\_0\_100M
3. axi\_interconnect\_1
4. ParametersMulti\_0
5. GradientsMulti\_1
6. SubsetCoordsMulti\_0
7. Gam\_Interface\_0
8. MUX\_0
9. blk\_mem\_gen\_3 [BRAM 3]
10. blk\_mem\_gen\_4 [BRAM 4]
11. blk\_mem\_gen\_6 [BRAM 6]
12. blk\_mem\_gen\_7 [BRAM 7]

**Outputs**

1. sub\_coord\_ea\_x\_0 [1-bit]
   1. sub\_coord\_ea\_x\_0 → enb [IP: blk\_mem\_gen\_6]
2. sub\_coord\_ea\_y\_0 [1-bit]
   1. sub\_coord\_ea\_y\_0 → enb [IP: blk\_mem\_gen\_7]
3. sub\_coord\_wea\_x\_0 [1-bit]
   1. sub\_coord\_wea\_x\_0 → web [IP: blk\_mem\_gen\_6]
4. sub\_coord\_wea\_y\_0 [1-bit]
   1. sub\_coord\_wea\_y\_0 → web [IP: blk\_mem\_gen\_7]
5. sub\_coord\_addr\_x\_0 [16-bits]
   1. sub\_coord\_addr\_x\_0 → addrb [IP: blk\_mem\_gen\_6]
6. sub\_coord\_addr\_y\_0 [16-bits]
   1. sub\_coord\_addr\_y\_0 → addrb [IP: blk\_mem\_gen\_7]
7. addr\_ref\_ints\_0 [17-bits]
   1. addr\_ref\_ints\_0 → gamma\_addr\_ints\_ref\_0 [IP: Interface\_0]
8. addr\_def\_ints\_0 [17-bits]
   1. addr\_def\_ints\_0 → gamma\_addr\_ints\_def\_0 [IP: Interface\_0]
9. addr\_grad\_x\_0 [17-bits]
   1. addr\_grad\_x\_0 → addrb [IP: blk\_mem\_gen\_3]
10. addr\_grad\_y\_0 [17-bits]
    1. addr\_grad\_y\_0 → addrb [IP: blk\_mem\_gen\_4]
11. gam\_done\_0 [1-bit]
    1. gam\_done\_0 → gamma\_done\_0 [IP: Results]
    2. gam\_done\_0 → probe\_in# [IP: VIO]
12. disp\_x\_0 [32-bits]
    1. disp\_x\_0 → dis\_X\_0 [IP: Results]
    2. disp\_x\_0 → probe\_in# [IP: VIO]
13. disp\_y\_0 [32-bits]
    1. disp\_y\_0 → dis\_Y\_0 [IP: Results]
    2. disp\_y\_0 → probe\_in# [IP: VIO]
14. disp\_z\_0 [32-bits]
    1. disp\_z\_0 → dis\_Z\_0 [IP: Results]
    2. disp\_z\_0 → probe\_in# [IP: VIO]
15. gam\_new\_subset\_0 [1-bit]
    1. gam\_new\_subset\_0 → gam\_new\_subset\_0 [IP: Gam\_Interface\_0]
    2. gam\_new\_subset\_0 → probe\_in# [IP: VIO]
16. gid\_0 [32-bits]
    1. gid\_0 → gam\_subset\_number\_0 [IP: Gam\_Interface\_0]
    2. gid\_0 → probe\_in# [IP: VIO]
17. reuslts\_done\_0 [1-bit]
    1. reuslts\_done\_0 → reuslts\_done\_0 [IP: Results]
18. gam\_idle\_counter\_0 [128-bits]
    1. gam\_idle\_counter\_0 → probe\_in# [IP: VIO]
19. gam\_busy\_0 [1-bit]
    1. gam\_busy\_0 → gam\_busy\_0 [IP: Counter\_0]
    2. gam\_busy\_0 → probe\_in# [IP: VIO]

**Associated IPs (outputs):**

1. blk\_mem\_gen\_6 [BRAM 6]
2. blk\_mem\_gen\_7 [BRAM 7]
3. Interface\_0
4. blk\_mem\_gen\_3 [BRAM 3]
5. blk\_mem\_gen\_4 [BRAM 4]
6. Results\_0
7. Gam\_Interface\_0
8. Counter\_0
9. VIO

**IP Description**

Since this IP is complex, we start by explaining the process of DIC operation in this IP and a detailed description for each function of the Gamma IP is also provided below.

For conducting the image correlation algorithm on the input frames, some precomputed values are required. Subset coordinates (the pixel indexes that are inside the pixels) are computed using SubsetCoordsMulti\_0 IP and the gradients (the intensities changes in each frame) are computed in GradientsMulti\_1 IP. So the Gamma\_Aff\_0 should start after the user defined, the subset coordinates, and the gradients are computed. This is checked at the first state of the IP. When these three sets of data were ready, the next states are started to loop over the total number of subsets (num\_of\_subsets\_0) and do the correlation. At first the initializations that should be done per subset are performed in state ‘b101110010 and then based on the user defined parameters for the correlation routine and optimization routine (currently we have tracking mode with the Fast method). The process for each subset starts at state ‘b000000101 in which the initial\_guess() function is called to start the optimization with a good first guess. Then in the next state, the displacements from the initial\_guess() function are saved and the computeUpdateFast() is called. This function is not in the form of a Verilog function and is implemented in states ‘b000000111 to ‘b1110100001. In the computeUpdateFast() function, a main for loop is implemented as 25 times (because of using Fast method) to find the best match of the current processing subset in the deformed image. at the first state after the start of the main for loop, initialize() function is called to initializes the work variables and based on the interpolation method (this is done at state ‘b000001000. At the next step, the mean() function for both reference and deformed image is called. Then it starts a loop as the number of pixels per subset (num\_pxl\_Int\_0) to find the region of interest by calling the residulas\_aff() function. After exiting this loop, some internal values are computed and the last step is saved and the convergence is checked. At the end of each iteration when the subset is found, the test\_for\_convergence\_aff() function is called at state ‘b000101110 to check we have reached the required accuracy or not. A tolerance value of 0.0001 is used for this aim. If we converged sooner (when the loop iterator is less than 25) the function exists and does not have to continue the algorithm. The next state will send the resulted displacements for the current subset to the Results\_0 IP, and it will jump to the ‘b101110001 state to process the next subset and so on. When we are done with all the subsets of the current frames, the displacements are accumulated at state ‘b000110010 to ‘b000110101 by a while loop over the total number of subsets per current frame and calling the Adder function from the floating point library. After that, we jump to state ‘b000110110 and wait for a new\_frmae signal from the client to reset the states and start from the beginning to process the new frames.

Note: The beelow functions are not implemented as the formal definition of the function in Verilog. It is because of some limitations that the functions have on their number of inputs and outputs. So, in this report we call them as functions but they are actually part of the FSM block from state X to Y. We we refer to calling a function, it means jumping to the first state of that part of FSM which implements the function. The method we use for jumping to a specific state of the FSM and then returning to the point we left is explained in the following paragraph in Arithmetic functions section.

* Arithmetic functions

Verilog states: ‘b1000000 - ‘b1111011

C++ file: all over the code

They include addition, subtraction, multiplication, division, sin, cos, asin, and acos. They are implemented in a FSM-based like the rest of the design. Each time that we call these functions, the present state (called “state”) is set to the starting state of that function to jump into it and do the process. At the end when the function is done, the return address (called “temp\_state”) is set to the next state that should be executed after returning form that arithmetic function. So, let’s say we are currently at state A, adder function starts at state B to E, and after calling the adder we should do the rest of the process by jumping to state F. So the state transitions should be like this: assuming we are in state A: state = B (to call addition function) and temp\_state = F. then we addition is done and we are in state E we have state = temp\_state, to jump back to the next state (F). The trigonometric functions have been implemented using Taylor series expansion and for the proposed basic arithmetic functions are utilized for computing each term of the series.

* gamma\_() function

Verilog states: ‘b1111100 - ‘b10000111

C++ file: DICe\_SubsetSerial.cpp L: 259

This function returns the ZNSSD gamma correlation value between the reference and deformed subsets. It has a for-loop over the number of pixels per subset and computes the difference between the intensities of the pixels and the mean value of the intensities for that subset. This function calls the mean() function for both the reference and the deformed subsets.

* mean() function

Verilog states: ‘b11000010- ‘b11000111

C++ file: DICe\_SubsetSerial.cpp L: 221

This function returns the mean intensity value for the reference or deformed subset. In the verilog implementation, in addition to the mean value of the intensities, the summation is also returned. Since it is used in the rest of the algorithm ans so to avoid an extra computation delay, we return both values each “mean” function is called. The outputs are stored in mean[31:0] and mean[63:32] for the summation and mean values respectively. The num\_pxl\_Int\_0 and num\_pxl\_FP\_0 input signals are used within this function as a loop counter and the denominator of the division for divining summation of the pixels to the number of pixels within the subset. Since the division is in floating point, so we need to have the number of pixels in floating point format (num\_pxl\_FP\_0) and in integer format (num\_pxl\_Int\_0) for the loop counter.

* initial\_guess() function

Verilog states: ‘b11001000- ‘b11001010

C++ file: DICe\_Initializer.cpp L:183

Within this function initialize method, called by the objective function to start the optimization with a good first guess. At first we save the previous displacements of the current subset and it also initializes the initial guess displacements with these values to call initial\_guess\_4 function.

* initialize\_guess\_4() function

Verilog states: ‘b11001011 - ‘b11010010

C++ file: DICe\_Initializer.cpp L:199

Initialize the subset in the vicinity of the previous guess. This function is called in “initial\_guess” function with the previous displacements as input. It starts with the given guess and calls “insert\_motion”, “initialize”, “gamma\_”, and “map\_to\_u\_v\_th\_aff” functions in order.

* initialize() function

Verilog states: ‘b11010011 - ‘b11100110

C++ file: DICe\_SubsetSerial.cpp L:300

Initializes the work variables and based on the interpolation method (BILINEAR, BICUBIC, …) calls the related function.

* interpolate\_bilinear() function

Verilog states: ‘b100000001 - ‘b100100011

C++ file: DICe\_ImageSerial.cpp L:354

The easiest of ways to interpolate the pixel intensity values for non-pixel locations.

Bilinear interpolation uses values of only the 4 nearest pixels, located in diagonal directions from a given pixel, in order to find the appropriate color intensity values of that pixel and tries to achieve a best approximation of a pixel's intensity based on the values at surrounding pixels

* interpolate\_grad\_x\_bilinear() function

Verilog states: ‘b100100100 - ‘b101000101

C++ file: DICe\_ImageSerial.cpp L:368

The gradients also need to be interpolated when the values are requested at non-integer locations.

* interpolate\_grad\_y\_bilinear() function

Verilog states: ‘b101000110 - ‘b101101000

C++ file: DICe\_ImageSerial.cpp L:382

The gradients also need to be interpolated when the values are requested at non-integer locations.

* map\_aff() function

Verilog states: ‘b101111111-’b110010110

C++ file: DICe\_LocalShapeFunction.cpp L:186

This function maps the input coordinates to the output coordinates. The inputs to this function are subset centerpoints (cx\_0 , cy\_0), reference and deformed intensities (dout\_ref\_ints\_0, dout\_def\_ints\_0), gradients (dout\_grad\_x\_0, dout\_grad\_y\_0), and subset coordinates (x\_0, y\_0) and the outputs are the mapped locations.

* map\_to\_u\_v\_theta\_aff() function

Verilog states: ‘b110010111- ‘b110010111

C++ file: DICe\_LocalShapeFunction.cpp L:297

The function converts the current map parameters to u, v, and theta.

* residuals\_aff() function

Verilog states: ‘b1110010000- ‘b1110001001

C++ file: DICe\_LocalShapeFunction.cpp L:309

This is a method that computes the residuals for this shape function (affine). It is called in the computeUpdateFast function for each subset.

* test\_for\_convergence\_aff() function

Verilog states: ‘b1110001011 - ‘b1110001111

C++ file: DICe\_LocalShapeFunction.cpp L:358

The function returns true if the solution is converged.

* save\_fileds() function

Verilog states: ‘b110100000 - ‘b110100000

C++ file: DICe\_LocalShapeFunction.cpp L:62

It saves off the parameters to the correct fields. The inputs are the schema that has the mesh with associated fields and the global id of the subset to save the fields for.