POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Elettronica

Tesi di Laurea Magistrale

VLSI architecture for Depth Image-Based Rendering

Relatore
prof. Guido Masera

Correlatore:
prof. Maurizio Martina

Laureanda
Roberta Marino
matricola: 237690

Anno accademico 2018 – 2019

Abstract

The video quality is everyday improving considering always more often three dimensional ultra high definition contents and video compression with multi-view video plus depth option, which can compress many views of the same scene. For each view, a depth map and two camera calibration matrices are needed, in order to obtain a partial 3D reconstruction of the scene. It is possible to mathematically build a virtual camera able to capture the scene from a different prospectives. This is called free-viewpoint video synthesis. This work starts by the software model of the algorithm presented in 1, that is the optimization of the Depth Image-Based Rendering (DIBR) algorithm, and then from the software model developed in 1 a MATLAB model nearer to hardware is built. Taking in consideration the high computational and high performance requirements of a real-time, the Very Large Scale Integration (VLSI) architecture free viewpoint video synthesis is developed. The structure is parallelized and pipelined to make the execution as fast as possible. For this purpose the frame is considered divided in sections called, in this work, "tiles". The data obtained by input memory enter in structure in tiles and the resulted data obtained by the architecture go out to the main out memory in tiles. Warping is the main part of the architecture and it computes the new coordinates for the input pixel in order to create the free virtual three dimensional vision between the two frames visions in input of the system. An occlusion-compatible warping order is presented to reduce the memory requirements and the amount of operations per pixel. In particular, the tile, read from main memory, is divided in four part and saved in four buffers. These data, one per clock cycle, are used by the warping to compute the correct coordinate of each pixel. The architecture is developed in fixed point and with a generic parallelism, but considering the dataset and then the imposted parameters, in this case,the tile dimension is equal to 256 pixels. So there is four warping blocks that work in parallel and each of them computes 64 pixels for tile, then the algorithm takes a quarter of the cycles to complete all the necessary operations of the warping. The result of the warping is saved with the corresponding pixel value in the a memory because the calculation of the correct order inside the tile is lower than calculation of the positions. Moreover reading the input small memory with modified raster order, the operations of warping are reduced to the incremental sum, except for the first pixel. Then the output data of the warping are saved in memory and they are read in the correct order respect to the occlusion-compatibility. The ordered data are stored in a custom designed cache, which works as an intermediary with the main memory.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduction

Depth Image Based Rendering (DIBR) is a algorithm that reconstructs a virtual view of the same scene starting by different point views.

1.1 Previous DIBR works and Optimizated DIBR

A previous and common software for a DIBR is the view Synthesis Reference Software (VSRS) and it is able to synthesize virtual views for both a simplified case where the cameras are placed in a parallel fashion as a linear array supposing that between views only horizontal shift is possible so that virtual views can be generated only between the two cameras. The algebra involved is also much simpler but the result is too constrained for a true free viewpoint content generation. The majority of the works in the previous literature about VLSI implementations of this algorithm are focused on this particular case scenario. The basic DIBR algorithm is slow and presents a heavy computational burden, then in[1] some algebraic transformations are proposed in order to achieve real-time performances, and to reduce the intermediary word width and thus save silicon real estate. The implementation of the algorithm requires for each warping stage various matrix-vector multiplication. The result of this warping is represented in an homogeneous coordinates system so a further division is required to find the destination of each pixel in the target frame. In particular, in the forward approach, for every input pixel, the resulting location in the arrival frame can be computed starting from **??** equantion.

$$u_v = \frac{d_1}{d_3} \cdot \frac{\frac{E_{1,1}}{d1} \cdot u_r + \frac{E_{1,2}}{d1} \cdot v_r + \frac{E_{1,3}}{d1} + d}{\frac{E_{3,1}}{d3} \cdot u_r + \frac{E_{3,2}}{d3} \cdot v_r + \frac{E_{3,3}}{d3} + d} \tag{1.1}$$

In 1.1 equation the matrix E is obtained merged the matrix A, camera parameters, and the vector b that are constant for each couple of input-output frames, u and v are the coordinates of the pixel in the image coordinates system. The matrix

A is composed by intrinsic and rotation matrices. Condidering that adjacent pixels share similar ur and vr coordinates the final formule to perform the incremental forward warping are then summarized here.

$$u^i = (\tfrac{b_1}{b_2} \pm n_u) + \tfrac{NUM_U^{i-1} + step\_\_U_u \cdot \ \Delta\_\_u_r + step\_\_V_u \cdot \ \Delta\_\_v_r \pm n_u \cdot \ d}{DEN^(i-1) + step\_\_U_D \cdot \ \Delta\_\_u_r + step\_\_V_D \cdot \ \Delta\_\_v_r + d} \tag{1.2}$$

$$v^i = (\tfrac{b_1}{b_2} \pm m_v) + \tfrac{NUM_V^{i-1} + step\_\_U_u \cdot \ \Delta\_\_u_r + step\_\_V_u \cdot \ \Delta\_\_v_r \pm m_u \cdot \ d}{DEN^(i-1) + step\_\_U_D \cdot \ \Delta\_\_u_r + step\_\_V_D \cdot \ \Delta\_\_v_r + d} \tag{1.3}$$

In 1.3 and 1.2 equations the first term $(\frac{b_1}{b_2} \pm m_v)$ is referred to the epipole coordinates. The terms with step prefix, instead, are referred to the incremental value corresponding to the movement in the frame. The two homogeneous coordinates are u for horizontal direction, and v for vertical direction. The d̈ïs reffered to the depth, in particular, it is the disparity value for each pixel in an unsigned 8 bit value, where the higher values are closer to the camera and the lower values are the most distant ones. NUM_U^{i-1}, NUM_V^{i-1} and DEN^{i-1} are the accumulation variables that store the partial computation of the numerators and the denominator.

$$\begin{pmatrix} u_v \\ v_v \\ 1 \end{pmatrix} z_v = A \begin{pmatrix} u_r \\ v_r \\ 1 \end{pmatrix} z_r + b \tag{1.4}$$

The 1.4 matrix equation is the main equation to compute the virtual coordinates, indicated in formula as u_v and v_v, starting from the real coordinates u_r and v_r. Matrix A and vector b depend on the physical camera setup. The camera is described by a 3x3 matrix, called intrinsic matrix K, a position in space (a column vector t) and a rotation matrix R. For more details of all the formulas and simplifications starting from this matrix that are made to obtain the incremental final result, refer to the reference reading[1].

Capitolo 2

Software implementation

In order to understand the real behaviour of the algorithm it was developed, in first time, a "golden model" on Matlab, very near to hardware implementation. The software model presented in [1], that cosidered the order of process of the pixel increasing the columns row by row, is changed considering the process of pixel for tile. In particular considering the main memory divided in blocks, for us tile, composed by 16 rows for 16 columns of pixels, then 256 words by 32 bit. As explained in the following part, two levels of ordering are considering. First level, that in this work will be called "tile level", processes the tile respect to the radius between the corners of the frame and the epipole coordinates computing by the camera parameter. Considering that the frame of dataset 2 contains 1024X768 pixels, then in a frame there are 3072 tiles. The other order level is inner of the tile. Then, when the tile is read from the main memory, in this level, it is established the order of processig pixel of this tile respect to the radius between the corners of single tile and the epipole coordinates. Established the order of processing , the main part of algorithm is the Warping. The warping is the computing of position of the considered input pixel of input tile in the output frame. This computing is done on the depth base. The depth has a important role, respect to the occlusion principle, explained in following part,as clear by the formulas 1.3 and 1.2. Merging the mathematical results obtained by the input images of left camera and right camera it is obtained the image of central view respect these two. The two input images of dataset 2 are reported in 2.2 and 2.1 with colors and the relative depths, in particular, the considered frame is the frame 95 and the two choosen cameras are the camera 3, cam3, and the camera 5 , cam 5, to obtained the virtual camera, cam4 .

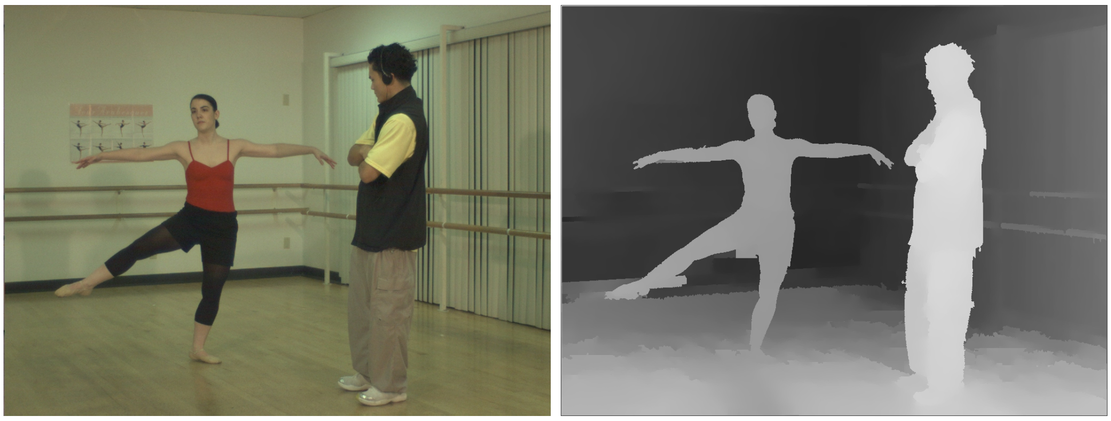


Figura 2.1. The input image obtained by left camera(cam3) of sequence frame 95 , the color on the left and the depth on the right

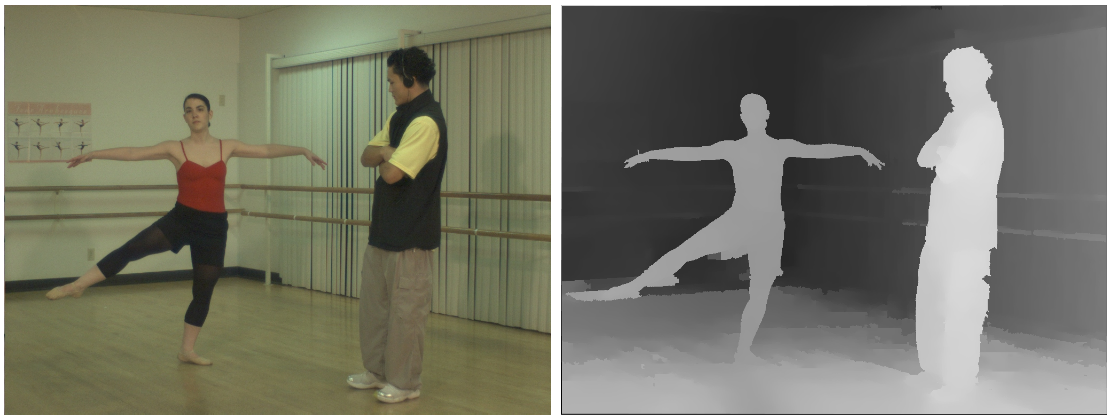


Figura 2.2. The input image obtained by right camera(cam 5) of sequence frame 95, the color on the left and the depth on the right

The resulting image on the left part of the 2 image is comparated with the image obtained by the real camera located beetween this two cameras to prove the good result of the considered algorithm.

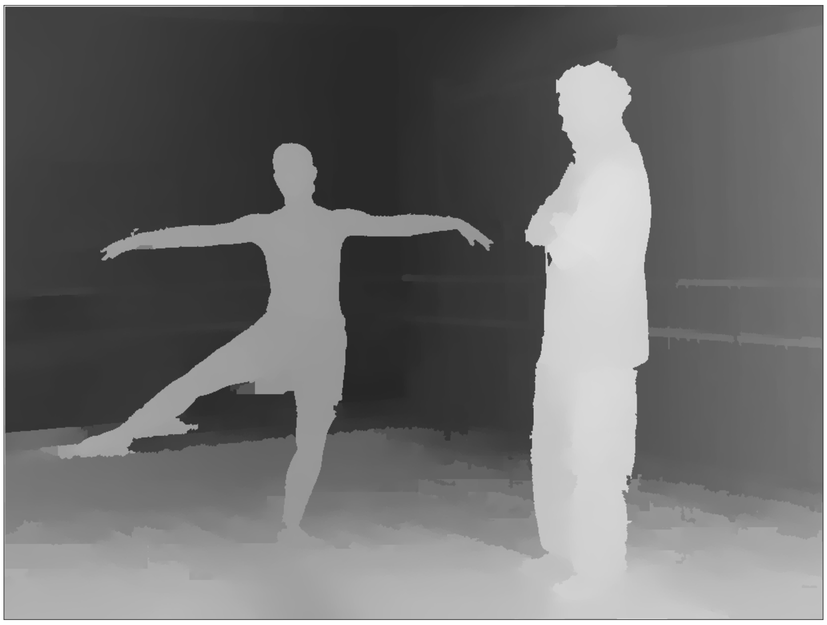


Figura 2.3. The obtained depth of virtual camera(cam 4) of sequence frame 95

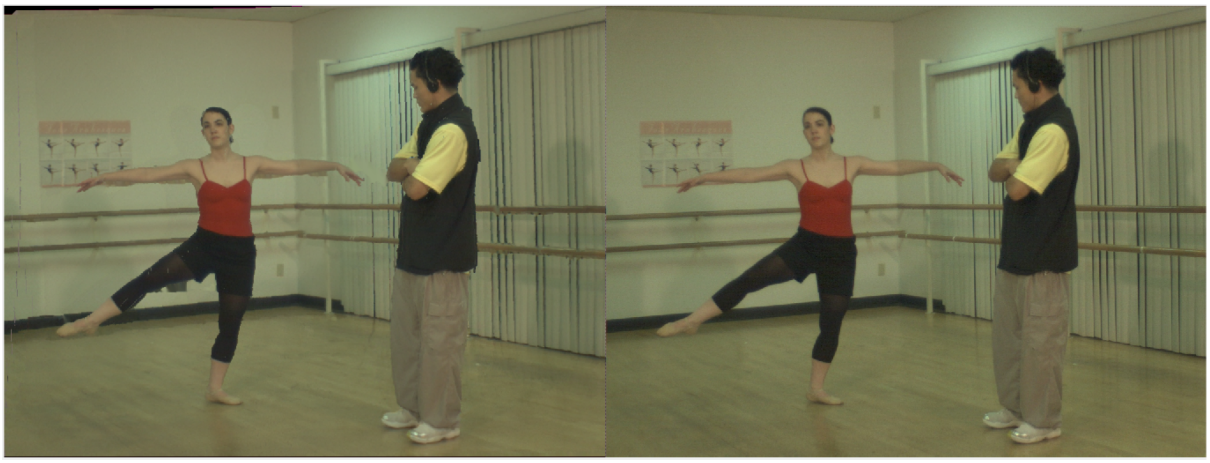


Figura 2.4. The computed image of virtual camera on the left, the real image of cam 4 located between left cam 3 and right cam 5 on the right

The parameters obtained by imput camera characteristics and by rotation and transformation matrices, descrived in[1], for the right camera,are in this case :

- u_param_1_r: it is the epipole u (horizontal)coordinate

$$u\_param\_1\_r = \frac{b\_r(1)}{b\_r(3)} \tag{2.1}$$

Ĺ

- v_param_1_r: it is the epipole v (vertical)coordinate

$$v\_param\_1\_r = \frac{b\_r(2)}{b\_r(3)} \tag{2.2}$$

Ĺ

- u_param_2_r: it is the parameter that considers the horizontal movement to compute the pixel u coordinate

$$u\_param\_2\_r = Z\_far\_r \cdot \ Z\_norm\_r \cdot \ \frac{A\_r(1,1)}{b_r(1)} \tag{2.3}$$

Ĺ

- v_param_2_r: it is the parameter that considers the horizontal movement to compute the pixel v coordinate

$$v\_param\_2\_r = Z\_far\_r \cdot \ Z\_norm\_r \cdot \ \frac{A\_r(2,1)}{b_r(2)} \tag{2.4}$$

Ĺ

- u_param_3_r: it is the parameter that considers the vertical movement to compute the pixel u coordinate

$$u\_param\_3\_r = Z\_far\_r \cdot \ Z\_norm\_r \cdot \ \frac{A\_r(1,2)}{b_r(1)} \tag{2.5}$$

Ĺ

- v_param_3_r: it is the parameter that considers the vertical movement to compute the pixel v coordinate

$$v\_param\_3\_r = Z\_far\_r \cdot \ Z\_norm\_r \cdot \ \frac{A\_r(2,2)}{b_r(2)} \tag{2.6}$$

Ĺ

- u_param_4_r: it is the parameter that considers the initial point to compute the pixel u coordinate

$$u\_param\_4\_r = Z\_far\_r \cdot \ Z\_norm\_r \cdot \ \frac{A\_r(1,3)}{b_r(1)} + Z\_norm\_r \tag{2.7}$$

Ĺ

- v_param_4_r: it is the parameter that considers the initial point to compute the pixel v coordinate

$$v\_param\_4\_r = Z\_far\_r \cdot \ Z\_norm\_r \cdot \ \frac{A\_r(2,3)}{b_r(2)} + Z\_norm\_r \tag{2.8}$$

Ĺ

- den_param_7_r: it is the parameter that considers the initial point to compute the denominator for coordinates u and v of the pixel

$$den\_param\_7\_r = Z\_far\_r \cdot \; Z\_norm\_r \cdot \; \tfrac{A\_r(3,3)}{b_r(3)} + Z\_norm\_r \tag{2.9}$$

Ĺ

- den_param_6_r: it is the parameter that considers the vertical movements to compute the denominator for coordinates u and v of the pixel

$$den\_param\_6\_r = Z\_far\_r \cdot \; Z\_norm\_r \cdot \; \tfrac{A\_r(3,2)}{b_r(3)} \tag{2.10}$$

Ĺ

- den_param_5_r: it is the parameter that considers the horizontal movements to compute the denominator for coordinates u and v of the pixel

$$den\_param\_5\_r = Z\_far\_r \cdot \; Z\_norm\_r \cdot \; \tfrac{A\_r(3,1)}{b_r(3)} \tag{2.11}$$

Ĺ

All these parameters are taken by the MATLAB computation and inserted in the registers as input of the structure. As explained in the following part in the section 3.1.1 on occlusion compatibility, there are two possibilities regarding the processing order of the pixels. Both solutions have been studied to understand which one was really the most convenient. The first option consists in considering the epipole itself as starting point and then starting from the calculated smaller radius, consequently we start to cover the nearest pixels and then the ones in the foreground and proceed always increasing the radius of one (in our If we refer to a tile of 16x16 pixels, we move 16 units). For each pixel it is necessary to check if that position has already been obtained in previous calculations, because, in that case, the new calculation must be discarded, being further from the epipole, due to the occlusion principle the current pixel would be covered by the previous one and not the opposite. The second option consists, instead, in starting from the greater radius then from the pixels furthest from the epipole and progressively decreasing the radius, getting closer and closer to the epipole. So every position already calculated is rewritten, because closer to the epipole due to the principle of occlusion. From a study on MATLAB of the two different systems the second approach results be the most convenient method, because there is a memory savings due to the fact that in the first method each pixel must be accompanied by a flag that tells us if it has already been written or less. Although the advantage that results from this analysis is that with the first method the resulting matrices are much less sparse than those obtained that the second method, but nevertheless the memory savings obtained with the second method is considerable. The resulted image obtained from Matlab to simultate the behaviour of structure in software is one of 2.5, because it is the left image obtained after the warping, before the fusiong and the filter part necessary to obtain the result shown in 2.

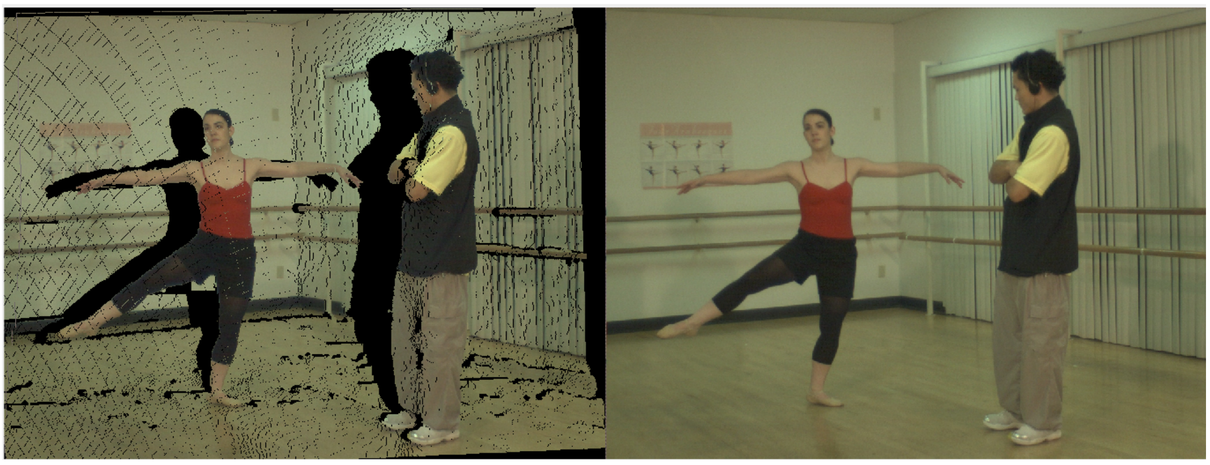


Figura 2.5. The left image otained after the warping before fusing and filter

Capitolo 3

Architecture

3.1 Organization

The entire architecture is parametric, then the parallelism and the dimensions can be changed by used respect the needs. The parameters resulted ready in the registers in input of the architecture for the warping. For the dataset obtained by [2] there are the following considerations. The data on 32 bit are stored in an external memory, 24 bits of color considerig the RGB model and 8 bits of the depth. Each tile could be composed by 16 rows and 16 columns for a total of 256 pixel, then (32bX256) about 1KB for tile.The tile approach, based on the burst access, is very good to increase the velocity of the access of the memory. For example in the considerated data set the dimension of frame is 768X1024, the needed memory is about 30MB,then it results be very big and very slow.

3.1.1 Occlusion compatibility

Warping is the main part of this work and in particular way the order of the warping. If zr is set to zero in equation (1.4) the corresponding obtained point does not depend on the input frame coordinates. This point is, in three dimensions, the centre of the input real camera, the obtained point where the distance from the pinhole of the real camera is nothing, then this point corresponds to the pinhole. The projection of pinhole coordinates by (1.4) to the virtual camera frame is a point called the epipole. It can be seen as the point of start of the real camera radius and it is the central point for an occlusion compatible warping order. The objects near to the epipole will always occlude the other objects placed behind them. This means that if this point is taken as a reference for the warping order there is no need to save each warped pixel the arrival u, v and z coordinates, nor a virtual camera depth map, thus saving a good amount of memory. If the epipole is used as reference, the input camera is correctly calibrated, and the input depth maps present a good quality, it is sure that the pixels that are closer to it will occlude other pixels that would have been warped to the same location. As stated in [1] there are then two methods to perform correctly a Z-buffer free warping. The first consists in following the lines starting from the epipole, called epipolar lines. This is difficult to perform correctly as there is a need to quantize the slope of the lines, especially near the epipole there are various problems as it is the center of a bundle of straight lines. In this region there is a mix of all the possible slopes and it can be very difficult to manage. The method followed in this work is to warp the scene as if it were made of concentric squares whose center is located at the epipole. It is possible to follow two different ways. The first is to start from the epipole and discard all the pixels that warp to an already filled location; the second method consists on the opposite approach, that means to start from the most external square and overwrite the old pixel value with the new one, because the pixels nearest epipole cover the behind part. For example if we observe a landscape with a object in foreground, it masks the parts in background behind it. Both approaches are valued in software with MATLAB and the second approach is chosen. The dimension of the external memory is about similar, the output tiles result less scattered, but it is not neccesary another bit as flag to signal if the pixel is been written or not. The pixel containing the epipole(into or out of the frame)can be set as the reference to choose the tile to load from memory and then another pixel processing would start inside the tile to ensure the occlusion compatible warping.

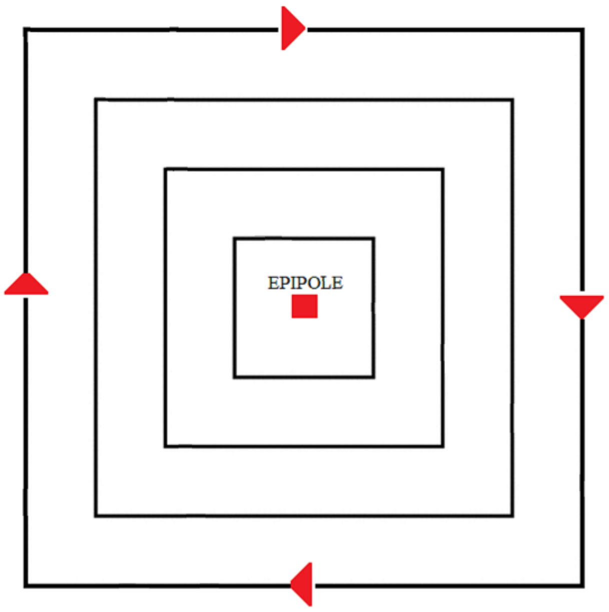


Figura 3.1. Warping order

The coordinates of the epipole are simply:

$$uv\_epi = \frac{b1}{b3} \tag{3.1}$$
Ĺ

$$vv\_epi = \frac{b2}{b3} \tag{3.2}$$
Ĺ

In this work they are ready in the input register and called u_param_1 and v_param_1. The proposed way to move around this point is to firstly calculate the distance between it and the four corners of the frame, as showing in the folowing part. Finding the most distant cornes, the resulted square radius will be obtained by highest difference beetween the coordinates (u or v) of the corner and the coordinates of epipole . At this point, if there is more than one border at that distance, there is the need to warp all their pixel tiles. The warping order can be set at will as long as all the pixels are processed in the correct order. In the proposed architecture a clockwise implementation is used. When all the tiles with the find highest radius have been warped, the radius is decreased by one and the process is repeated until all the tiles have been warped. To ensure the occlusion handling the order of warping is chosen at both tile and pixel levels, calculating the warping order also for the part inside the tile. The maximum radius is computed by the block "COMPUTATION MAIN RADIUS" for the tile level and "COMPUTATION TILE RADIUS" for the pixel level. Instead the block that computes the address of next tile and the address of the next pixel as just explained are "DIRECTION MANAGEMENT" and "TILE MANAGEMENT".

3.1.2 DIRECTIONS

As a result of what has just been explained, there are four possible directions of warping following the clockwise direction.

- DVL:that is the direction that starts by the pixel in the pixel level(or tile in frame level)of corner START_L and arrives up to the pixel before the START_UP;
- DHU:that is the direction that starts by the pixel in the pixel level(or tile in frame level)of corner START_UP and arrives up to the pixel before the START_R;
- DVR:that is the direction that starts by the pixel in the pixel level(or tile in frame level)of corner START_R and arrives up to the pixel before the START_DOWN;

- DHD:that is the direction that starts by the pixel in the pixel level(or tile in frame level)of corner START_DOWN and arrives up to the pixel before the START_L;

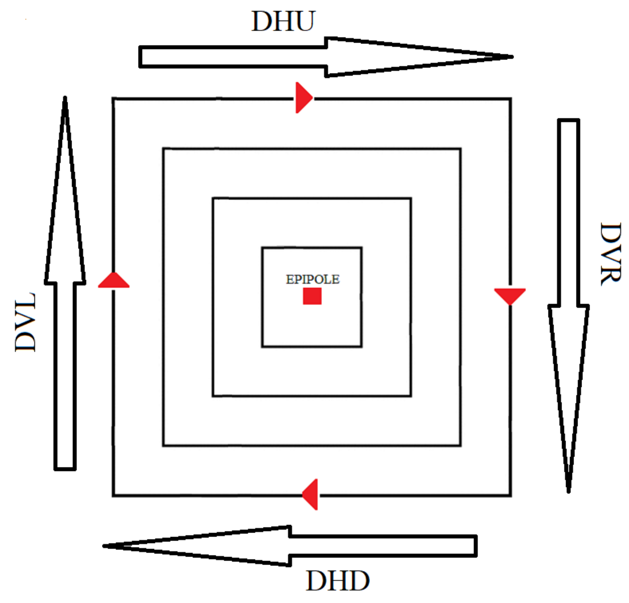


Figura 3.2. Illustration of directions

In figure 3.2 it is shown the possible directions of the architecture, the same considerations are valid for the tile level. The figure rapresents only a example, because the epipole could be in every position.

3.2 Parallelism

In the initial analysis the computed needed parallelism for the structure was very high. Considering the overflow conditions and the presence of different multiplies the parallelism was about 60 bits for the numerators of the divisor in the warping block. For parallelism, the divisor rapresents the most critical component of the warping block, in fact all considerations reported in this section are done to reduce the number of bits of divisor terms. Considering the worst case of the data set the paralellism of numerators for warping of u and v resulted 27 bits plus the fractional part of six bits it was 35 bits and the denominator parallelism was 17 bits plus 6 bit od fractional part, obtaining 23 bits considering the guard bits the chosen values are 25 bits for the denominators and 35 bits for numeratos. The figures 3.4 and

3.3 are reported to undestand the choise of parallelism. The denominator of final division of the warping are considered to understand if possible reduce divisor work. In order to obtain it, the most significant bits of denominator is counted and the numerator is left shifted of this quantity to decrease the work of divisor with the low significant bit full of zeros. To preserve the correct result the two numeratos are left shifted of the same quantity.

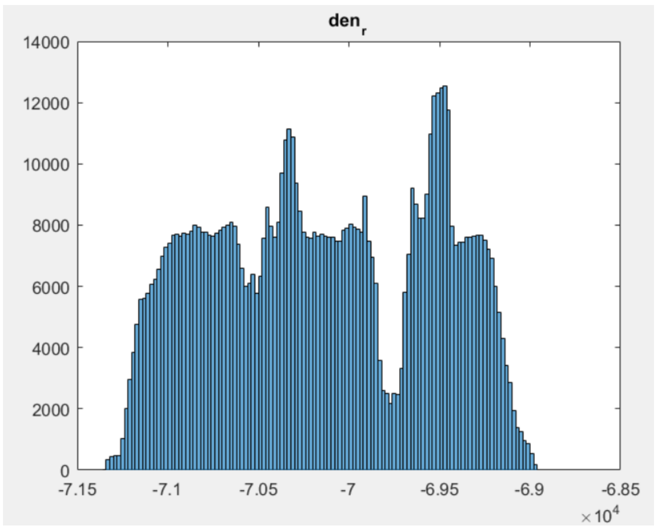


Figura 3.3. Histogram of the denominator in the warping of right vision without shift

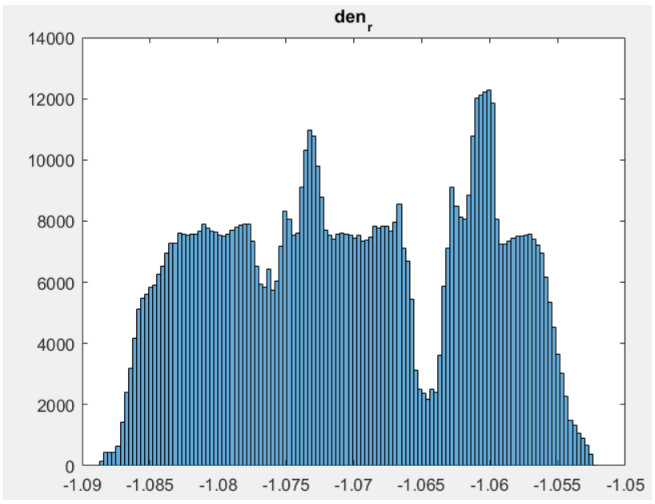


Figura 3.4. Histogram of the shifted denominator in the warping of right vision

3.3 Radiuses

In the structure there are two different radiuses. One to order the tiles and one to order the pixels in to the tile, in this work the first is called "main radius" and the second "tile radius". The chosen radius in both cases is the gratest radius. In first time, the difference beetween the epipole cordinates and the corners of the tile, for radius tile, or of the frame, for main radius, is computed. Between these two difference is chosen the highest module. In this way, the start corner farthest from epipole is detected. Detected the correct start corner the list of directions is uploaded, and the initial direction is selected with a "SR" signal, output of the lut. SR is equal to "00" when the start is START_L,"01" when "START_UP,"10" when START_R,"11" when START_DOWN. In the dataset used in the work, 2, the frame is on 768x1024 pixels then the corners coordinates are: START_L with coordinates (767,0), START_UP with coordinates (0,0), START_R with coordinates (0,1023) and START_DOWN with coordinates (767,1023).

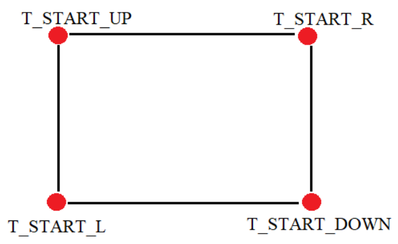


Figura 3.5. Starting corners in the tile

The SR signal and the radius are in input ones of main radius to the direction management and ones of tile radius to the tile management. The radius is decreased when a ring with the same radius is finished. In figure 3.6 is reported only the architecture of the tile radius because it is very similar to the other radius structure.

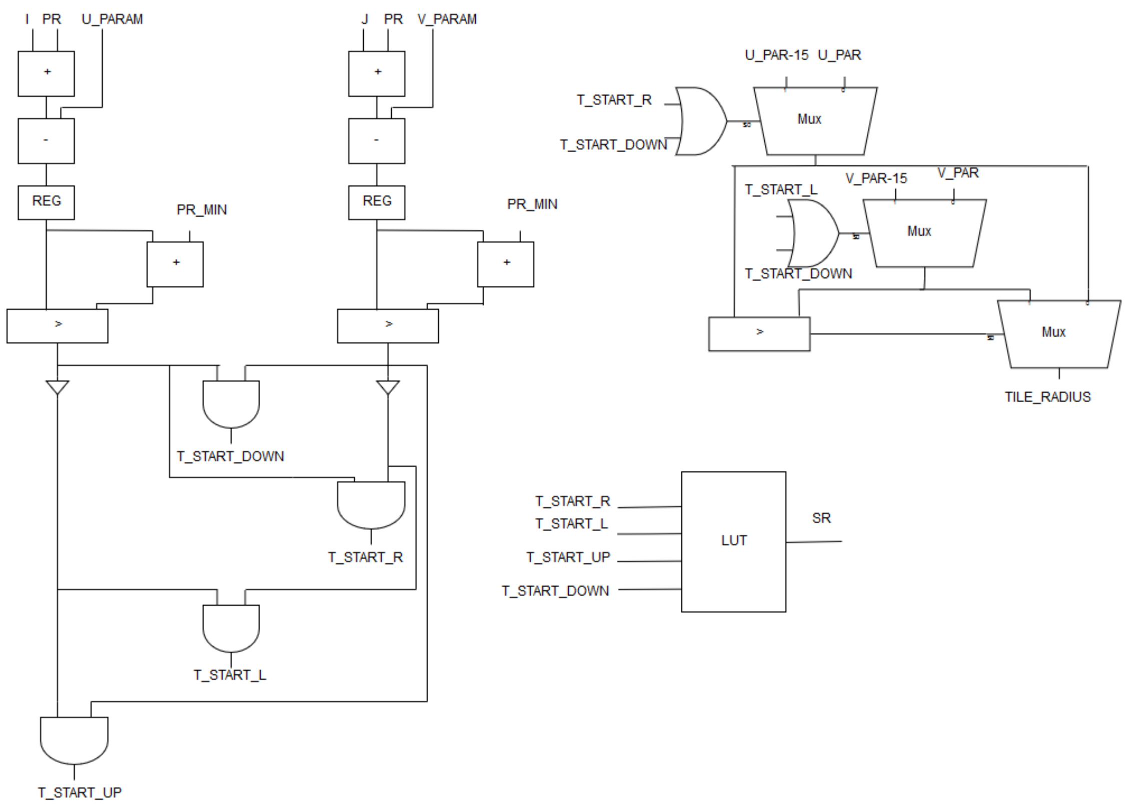


Figura 3.6. The architecture for the computing tile radius

It is the very simply structure,in the first clock cycle, the coordinate of input tile are added to the PR,that is a constant parameters chosen by the user, it depends on the tile dimension, in this case being the tile of 16x16 the PR could be set equal to 15.

$$radius1 = U_{PARAM} - (I + PR) \tag{3.3}$$

$$radius2 = U_{PARAM} - (I + PR) + PR = U_{PARAM} - I \tag{3.4}$$

The $radius1$ and $radius2$ in the formulas 3.4 and 3.3 corresponding to the difference between the coordinate u of epipole and coordinate of the corners of tile. In particular, $radius1$ correspond to difference with the u coordinate of the START_R and START_DOWN, instead $radius2$ correspond to difference with the u coordinate of the START_UP and START_L. The same considerations are valid for the v coordinate. From the comparison between the two radiuses of two different coordinates the starting point.

3.4 DIRECTION MANAGEMENT

This block manages the direction of reading of the tile in the MAIN MEMORY. There are four counters, one for each direction, in order to count the number of time that it is follow that direction. It is very important, because, for example if the tile is covered by the previous direction, it is not covered two times. The outputs of counters go directly in the multiplexer, called MUX1, for the HU and VL directions, instead for the VR and HD are subtracted by the maximum address of the tile, in this case 1023 for VR and 767 for HD, these results are called "SM" and "SM1". The maximum value because, when it is covered one time, for example, the VR direction the next directions do not start by the column with address 1023 because they are get covered by VR direction. After these subtractions the results go to multiplexer with the other two out of the counters. This multiplexer is controlled in base to the actual direction: for the first time this direction is detected by computing radius block, by SR, in the following times, instead, by a unit that respect to the input radius detects the correct direction. This part starts from the output of MUX1, it is firstly shifted by 16 because, considering the dimension of tile, the coordinates increase of 16 and the result is subtracted by epipole coordinate, u or v, respect to the direction and then this new radius is subtracted by the radius. This radius comes from a counter that has in input radius of computing radius unit and this counter decrements the radius each time the ring with same radius is completed and then it is decresed by 16, 16 because a tile is compesed by 16x 16 pixels. When the result of subtraction with epipole coordinates is equal to the radius of actual ring, the output of comparator is equal to one and then the corresponding direction flag is put to one. The output of initial counters are called CVL, CVR, CHD, CHU. When a direction is detected a new counter is actived in order to compute how many tiles are processed for this direction before passing to other directions, in particular these counters are increased by 16, in the case of CNT_J_P_16 and CNT_I_P_16, or decrease by 16 in the case CNT_J_m_16 and CNT_I_m_16. The starting value of shifted CNT_J_P_16 is CHU, of shifted CNT_J_m_16 is CHD, of shifted CNT_I_P_16 is CVL, and of shifted CNT_I_m_16 is CVR. In the final part there are two multiplexer to detect the correct address of column and row, callend i and j, in this work. Respect to the direction four values of i and j can be selected. If the direction is HU, j is equal to the shifted CHU and i to the output of CNT_I_P_16. If the direction is HD, j is equal to SM and i to the output of CNT_I_m_16. If the direction is VL, j is equal to output of CNT_J_m_16 and i to the shifted CVL. If the direction is VR, j is equal to the output of CNT_J_P_16 and i to SM1. whenever a direction is finished, it is verified that the next clockwise direction has the same radius, if the equality is not verified, it is passed to the other direction clockwise. When all 4 directions have been checked the radius is decremented and the first direction is re-examined. In order to ensure that during the control of all directions not followed the outputs i and j are not considered valid there is a signal that is asserted when the outputs are valid. The enable of block comes from control unit and it is the only signal by it, because the other signals are logically internally produced.

3.5 Warping

3.5.1 Parallelization

In order to obtain the higher velocity and a throughtput about one pixel position for clock cycle the read input tile is considered divided in four parts and then the four different data for clock cycle are saved in four different fifos (FIFO1,FIFO2,FIFO3,FIFO4) and then they are put in input of the four different warping blocks. For each clock cycle there are four positions of different pixels. In output of each the Warping block(WARPING) there is the position of considered pixel, the data of color and depth, contained, initially, in the fifo, and the validity of the computed positions. This least significant bit is sent to the control unit, that enable the writing of the DATA MEMORY. The DATA MEMORY is the tile dimension memory that contains the all the output data of the warping and when the all data of the actual tile are written in this memory the signal of control unit is actived and the entire content of the Data memory is written in FIFO_DATA. In the final part of the architecture there are a tile management that by the first moment produces the order of the address of single pixel of the tile, considering the radius of different single tile and then the direction for the occlusion compatibility. These addresses is the addresses of the REORDER MEMORY that write the data reading the entire tile from FIFO_DATA and in output read data are the order data and they are written in the cache and then when a tile is completed in the MAIN memory. Thanks to this structure the data in input of the cache are one for clock cycle, because despite the first part of structure is more speed and in only 64 clock cycles it produces the warping of the entire tile the FIFO_DATA is quickly full and then, when it is full, the first part of the architecture is stopped, but the second one continues to work.

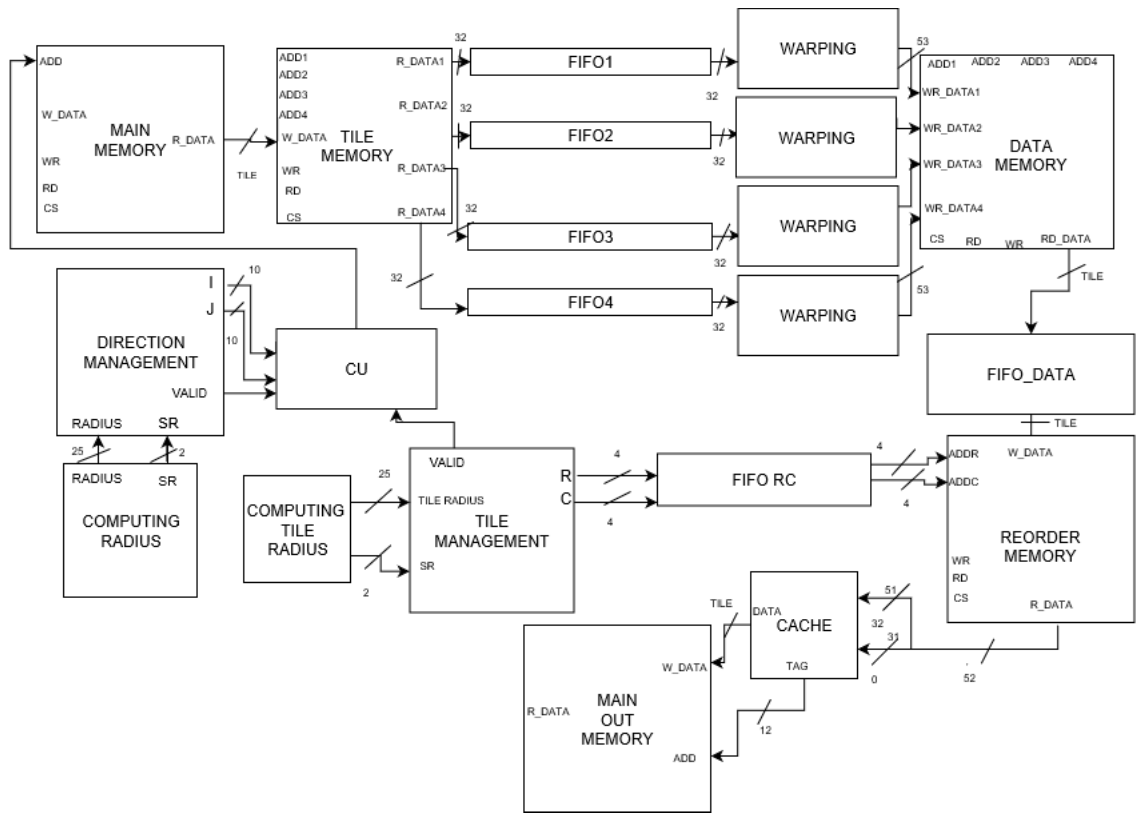


Figura 3.7. The Highest hierarchical vision of Architecture

In the scheme all signals not connected come from the Control Unit.

3.5.2 Warping order and Tile Memory

Considering that the slower block is the TILE MENAGEMENT, the data by tile memory do not warped in order respect to the occlusion considerations established by TILE MENAGEMENT block, but in the modified raster. The first row is read for raster, increasing the column, but when the columns of the row are finished the row is increased by one and the columns are decreased one by one, the following order is similar to trasing a z. In this way the number of operations of warping is reduced to one sum, except for the first pixel. As showed in figure 3.8 it is as the TILE MEMORY is divided in four parts. Each part has a initial poit (in figure it is the red dot) and the Warping is done in increasing order for first and third rows, then decreasing order for second and fourth rows. In this way the calculations are very speed because they are incremental computations. Then the outputs of the Warping block are saved in the DATA MEMORY and only, in last part, they are reorder respect the occlusion compatibility. In this way the algorithm is more efficient.

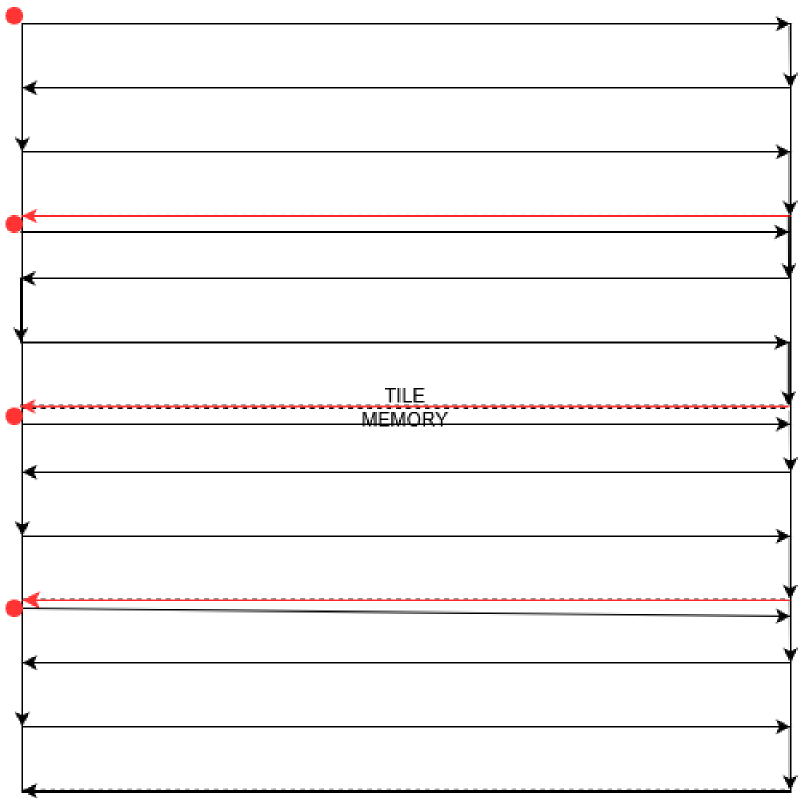


Figura 3.8. order warping

3.5.3 Warping block

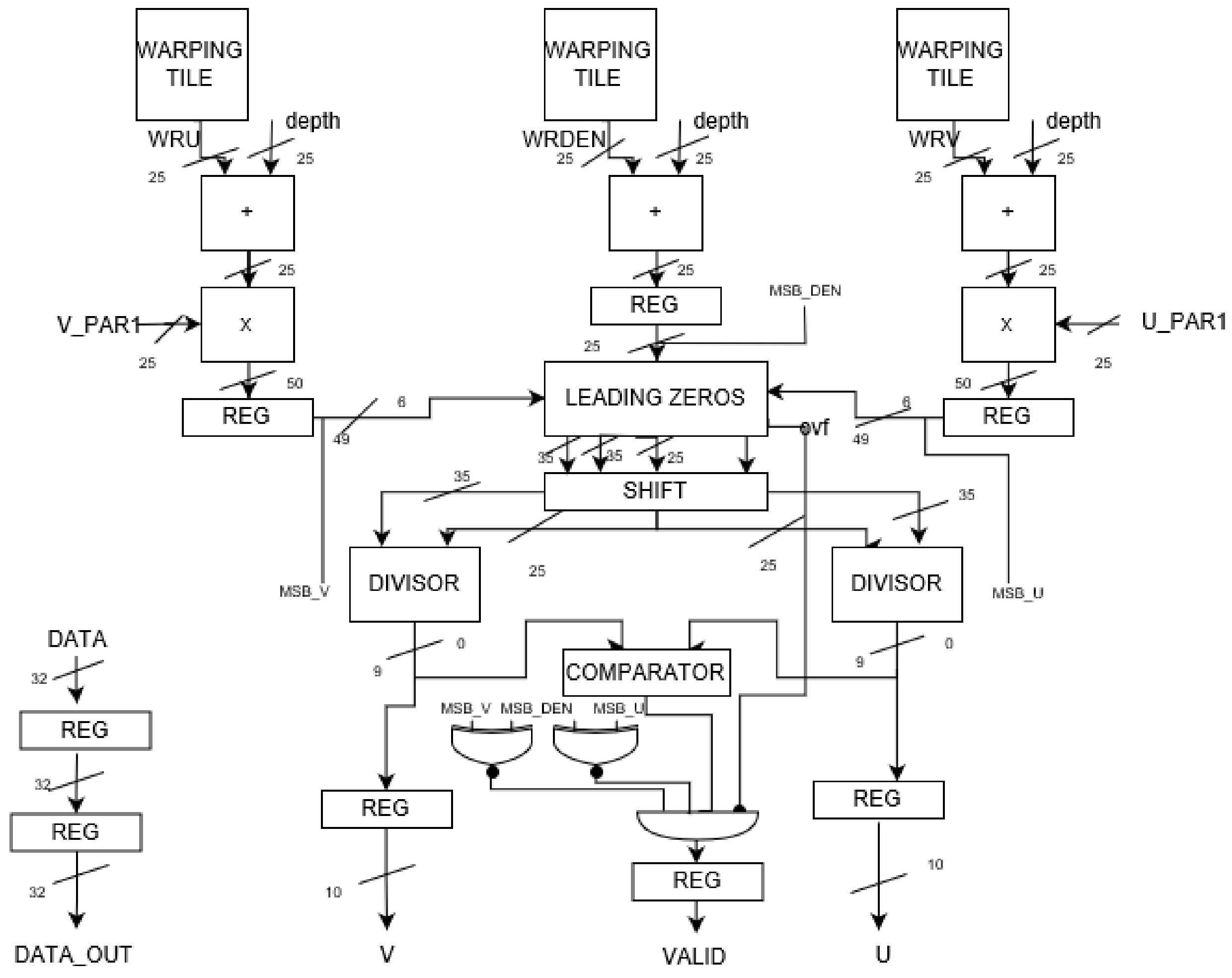


Figura 3.9. Internal structure of Warping block

This block is the heart of all the architecture developed in this work. It is the unit that calculates the final positions of the pixels read in input in the output memory to create the image of a virtual camera positioned between the two captured images of two cameras. It consists of several parts. The first part is the WARING TILE block that provides a first important result for the calculation of the numerators and denominators that will be input to the divisor to produce the coordinates u and v, it will be explained in detail in the appropriate section. The necessary blocks of WARPING TILE in each WARPING block are three: one for preliminary calculation of the numerator u, WRU, one for preliminary calculation of the numerator v, WRV, and one for preliminary calculation of denominator, WRDEN. WRU, WRV and WRDEN are added at the depth read by the corresponding fifo input. The depth corresponding to the eight least significant bits of the read data of the corresponding input fifo. To perform this, firstly, the depth on 8 bits are estended to 25 bits to add two input with the same parallelism. The partial denominator is saved in a pipe register, while the two partial numerators are multiplied by the epipole coordinates, partial numerator u by u epipole coordinate and partial numerator v by v epipole coordinate. Being the two inputs of the multiply on 25 bits the output of it will be on 50 bits and these results will be saved in the pipe registers to decrease the critical path. So, the two actual numerators and the denominator in the pipe registers are put in input to the LEADING ZEROS, in particular in the two numerators the least significant six bits are truncated, since the denominator that has not been multiplied has only 6 bits of fractional part, while the two numerators on 50 bits have 12 bits of fractional part, so the truncation is done to keep the same 6 bits of fractional part of the denominator. This last block computes the numbers of bits that are only the sign extension in the denominator and it checks if there is the possibility of the overflow at least in one of the two shifted numerators. The numbers of leading zeros of the block and the two numerators and denominator are put in input of SHIFT that shifts of numbers of leading zeros both numerators and denominator. The 35 least significant bits of shifted numerators are put in input of the corresponding divisor. Just 35 bits because from the study of the data on matlab the maximum parallelism required to correctly represent the largest and the smallest number entering the divisor was on 27 bits and considering the six fractional part bits and the two guard bits to be added, the 35 bits are more than sufficient. The read data by the corresponding input fifo are delayed by two clock cycles, because of the two layer of pipeline, inserted in the block, one just described between the multiplier and the divider and another before the output, to clean the signals from any glitches. The each components of warping is explained in the relevant sections.

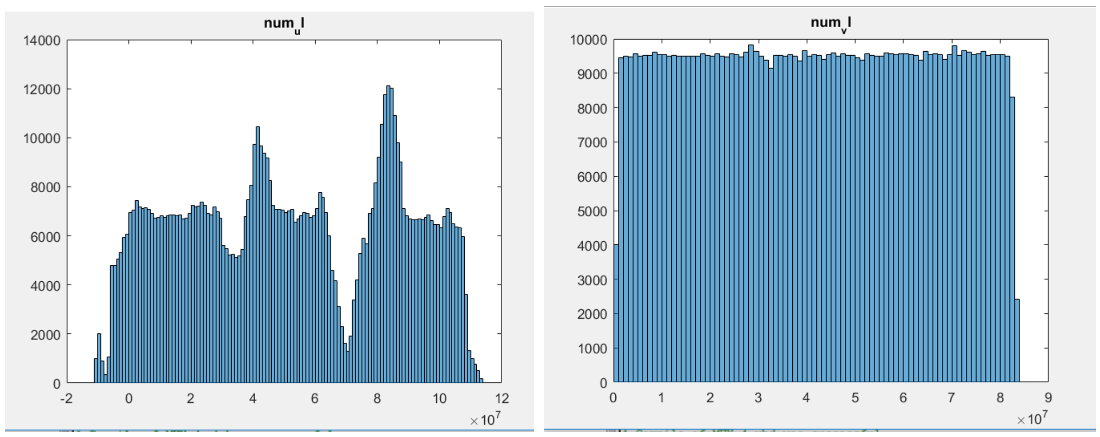


Figura 3.10. Histogram to study of maximum parallelism of numerators in input of divisor for left vision

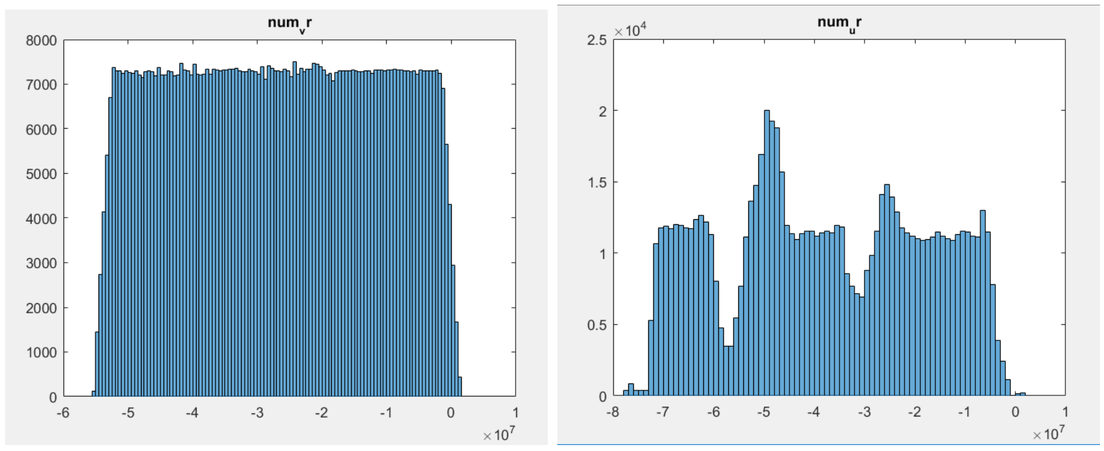


Figura 3.11. Histogram to study of maximum parallelism of numerators in input of divisor for right vision

3.5.4 Warping tile

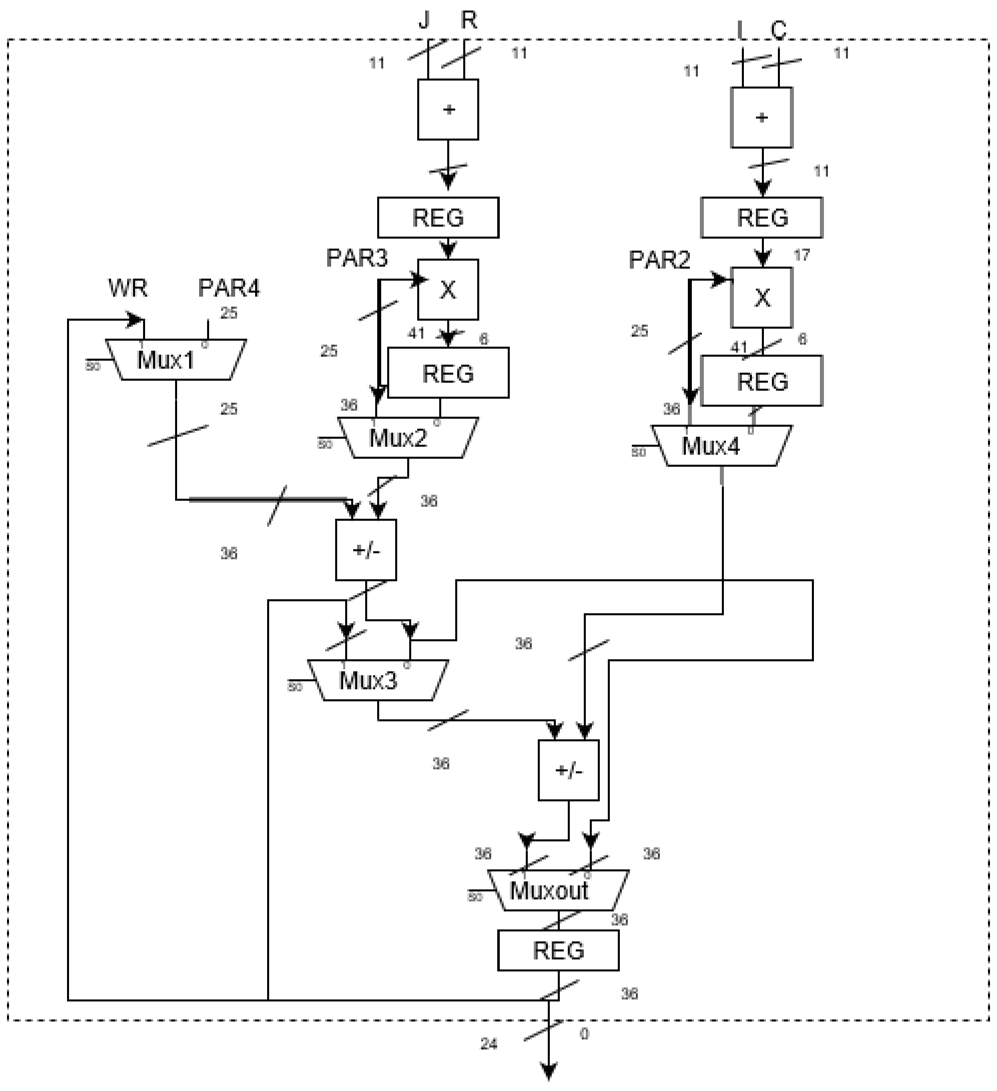


Figura 3.12. Internal structure of block base of Warping block

This block has, in input, parameters already ready in the appropriate registers, the address of the actual tile J and I and the location of processed pixel into the tile R and C, all these inputs are used to compute the partial result following the formulas in the Software implementation section. The R and C addresses are fixed, because they are used only when the position of first pixel of corresponding subtile are computed. In particular, considering figure 3.12, in the WARPING TILE block corresponding to the first quarter of TILE MEMORY, then the WARPING in output of the FIFO1, the fixed R and C are equal to zero, in the second quarter of TILE MEMORY, then the WARPING in output of the FIFO2, the fixed R is equal to four and C is equal to zero,in the third quarter of TILE MEMORY, then the WARPING in output of the FIFO3, the fixed R is equal to eight and C is equal to zero,in the last quarter of TILE MEMORY, then the WARPING in output of the FIFO4, the fixed R is equal to twelve and C is equal to zero. The R and C on four bits are extended to eleven bits and also J and I on ten bits are extended to eleven bits to avoid the overflow in the sum. The J and R and also I and C are added and the results of two adders are saved in corresponding pipe register. The results in output of the registers on 11 bits are extended to 17 bits with a concatenation to the lsb of six bit equal to zero to create the fractional part(left shift for wires). Then they are multiplied by PAR2, the modified result of sum between I and C by PAR3, the modified result of sum between J and R, to rebuild the due incremental value in that location. In fact if the order had been in succession of row and column, then proceeding in succession every time it would be added to the initial parameter PAR4 in the case of numerators or DEN7 in the case of the denominator, as explained in the software implementation section, from time to time the parameter PAR3 or DEN6 each time that a new row was started and parameter PAR2 or DEN5 each time a column was advanced. The results of two multiplications are saved each in a pipe register. In order to obtain the partial numerator or denominator the MUX2 and MUX4 select the input of the longest branch which is the result of the operations just explained (SEL2='1' and SEL4='1' in the secondary control unit), the MUX1 selects the input of PAR4(SEL1='1' in the secondary control unit), then the SUB/ADDER is selected to compute the sum (EN1='0' in the secondary control unit), the MUX3 selects the branch of the result of SUM1 (SEL3='1')and also SUM2 is selected to perform the addition(EN2='0') then MUXOUT chooses the branch of result of SUM1 (SEL='0') and the ready partial numerator or denominator is saved in the occumulation register and then ready for the final part of the Warping. These operation is done only when the tile is changed, then for each WARPING TILE one time on the 64 computations and these selections are done by the FIRST status in the secondary control unit. In all other 63 computations the path is very short. When the reading of TILE MEMORY is made increasing the columns for the same row only the PAR2 is added to the accumulation value WR. So, for this operation MUX3 and MUX4 choose the input0 (SEL3='0' and SEL4='0'), SUM2 remains set to make a sum(EN2='0') and also MUXOUT remains unchanged(SEL='0').These selections are done by the H_UP status in the secondary control unit. When the reading of TILE MEMORY changed only the row with the same last column only the PAR3 is added by the accumulation value WR.So, for this operation MUX1 selects the imput0 with WR (SEL1='0') and MUX2 choose the input0 corresponding to PAR3 (SEL2='0'), SUM1 is selected to perform a sum (EN1='0') and MUXOUT changed (SEL='1') choosing the input1 with directly the result of SUM1. These selections are done by the R_PLUS status in the secondary control unit. When the reading of TILE MEMORY is made decreasing the columns for the same row only the PAR2 is subtracted by the accumulation value WR. So, for this operation MUX3 and MUX4 choose the input0 (SEL3='0' and SEL4='0'), SUM2 is selected to perform a subtraction (EN2='1') and MUXOUT remains unchanged(SEL='0'). These selections are done by the H_DOWN status in the secondary control unit. The result of multiply on 42 bits are truncated of the six least significant bits to report the number of the bits of the fraction part to six. The inputs of MUX2 and MUX4 are, in fact, on 36 bits. The inputs of MUX1 are on 25 bits the for the SUM1 the output of MUX1 is extended on 36 bits and the SUM1 and then there is a possibility of overflow for SUM1. The inputs and output of MUX3, SUM2, MUXOUT and REGOUT remain on 36 bits. But WR is truncated and reported on 25 bits, because, by the study of the data, results already enough and to maintain the parallelism of both MUX1 and output of the block TILE WARPING on 25 bits with 6 bits of fractional part. In all operations the direction of operators are made only if it is necessary, in the other case the operator that do not work during a computation are left unchanged to decrease switching activity.

3.5.5 Divisor

Considering the run time requirements the choice of the divisor is obtained with many consideration. In order to do a speed divider the data input are changed without modifing the results. In first analyse the pseudo-fixed point data are considered and then to reduces the weight of input,in terms of number of significant bits, trailing zeros and leading zeros are considered. Trailing zeros is applied counting the number of zeros in least significant bits and right shifting the data of this quantity, but it resulted not efficient in the odd numbers, because the result of following division is very inexact. Leading zeros constists, for the positive numbers, in counting the number of zeros (or ones in negative numbers)in most significant bits, then the sign extension, and left shifting the numerators and denominator of this quantity, in this way the least significant part of divisor is occupated only by zeros and the velocity increased, because of the carry-out propagation is improved. The structural architecture of the divisor is the non-restoring-arrey, very useful because when the parallelism of numerators and denominator is very different this divisor turns out to be very efficient. The used structure is the following:

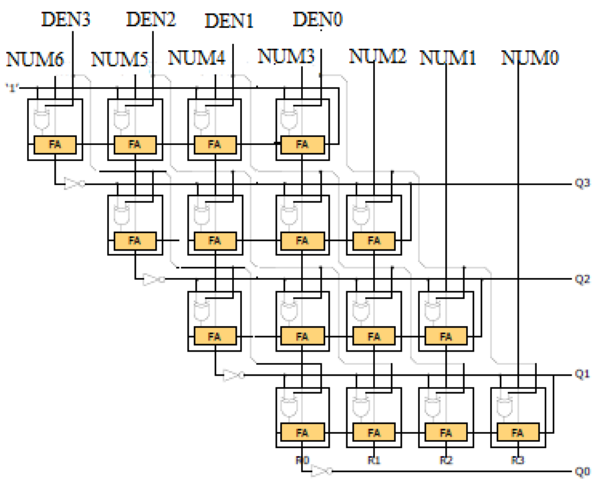


Figura 3.13. The architecture of the non-restorin array divisor

In the structure there are two divisors: one for computing u coordinate and one for computing v coordinate. These results are truncated because the possible coordinate are on 10 bits. In particular the possible range for u coordinate is by 0 to 1023 and for v coordinate by 0 to 767.

3.5.6 LEADING ZEROS & SHIFT

In order to decrease the work of the divider, as mentioned before, there is this block that detects the number of significant bits that are only sign extensions. In fact it does not detect only number of zeros as suggests its name often referred to floating point numbers, but in general all bits of sign extension. This number is computed comparing one by one each bit of the denominator starting from the most significant bit to see with a xnor bit tobt how many are equal to the sign bit (msb), in case the result produced by the xnor is equal to one, a variable is incremented. The first time that one of the bits is no longer equal to the sign bit, the variable is no longer incremented and after its use is reset again. The outputs of LEADING ZEROS are five, one of the computed quantity, two of numerators and one of denominator, they are put in input of SHIFT block in order to shift the numerators and denominator of this computed quantity. The other one is the ovf , a signal that detects if there is a overflow when the numerators are shifted, in fact, the numerators and denominator are passed to the shift by LEADING ZEROS, because they serve in LEADING ZEROS in order to provide the ovf signal. The output of shift are three: The shifted numerator u, the shifted numerator v and the shifted denominator. They are put in input of the divisor to compute the final coordinates.

3.5.7 COMPARATOR & VALID SIGNAL

When the divisor produces the results, u and v coordinates, these have to be in the range of possible positions in output frame. This ranges, as mentioned before, is for u coordinate between 0 and 1023 and for v coordinate between 0 and 767, then on 10 bits. The values, obtained by the divisors, are put in input of the comparator that veriry if they are in ranges. Then in output of Warping there are computed u and v , the VALID signal to indicate if these results are valid and tha data of input tile delayed by two clock cycle, this delay is necessary because, respect to the input data of the tile, the results related to this input, go out after two clock cycle. One clock cycle for pipe register to divide the critical path between multiply and divisor and one clock cycle, because the valid signal is combinatory then the possibilty of the glitches is very high then to avoid them a ragister is put in output of AND gate. The valid signal is obtained, in fact, by the logic AND between four signals:

- one is the result of the logic xnor between the most significant bit of denominator and most significant bit of numerator(u), because the result of the division to be valid has to be positive and for this purpose the sign bit (MSB) of numerator has to be equal to sign bit (MSB) of denominator;
- one is the result of the logic xnor between the most significant bit of denominator and most significant bit of numerator(v),because the result of the division to be valid has to be positive and for this purpose the sign bit (MSB) of numerator has to be equal to sign bit (MSB) of denominator;
- one is the result of the comparator that is one only if both u and v are included in the ranges.
- the last one is the negated ovf signal, mentioned before, produces by the LEADING ZEROS block when there is the overflow in the shift of at least one of the two numerators;

3.6 TILE MANAGEMENT

This block manages the direction of pixel inside of the tile respect to the occlusion compatibility. There are four counters, one for each direction, in order to count the number of times that it follows that direction. It is very important because, for example, if the tile is covered by the previous direction, it is not covered two times. The outputs of counters go directly in the multiplexer, called MUX1, for the THU and TVL directions, instead, for the VR and HD are subtracted by the maximum address of the tile, in this case 15 for TVR and for THD. The maximum value, because, when it is covered one time,for example, the VR direction, the next directions do not start by the column with address 1023, because they are get covered by TVR direction. After these subtractions, the results go to multiplexer with the other two outputs of the counters. This multiplexer is controlled in base of the actual direction: for the first time, this direction is detected by computing radius block, SR, in the following times, instead, by the unit that respect to the input radius detects the correct direction. This part starts by the output of MUX1, that is added to i or j coordinate respect to the direction and then the result is subtracted by epipole coordinate, u or v, respect to the direction and then this new radius is subtracted by the radius. This radius comes from a counter that has in input radius of computing tile radius unit and this counter decrements the radius each time the ring with same radius is completed and then it is decreased by one, when the tile changes the new radius is loaded by the computing tile unit. When the result of subtraction with epipole coordinate is equal to the radius of actual ring, the comparator output is equal to one and then the corresponding direction flag is put to one. The output of initial counters are called CTVL, CTVR, CTHD, CTHU. When a direction is detected a new counter is actived in order to compute how many pixel are processed for this direction before passing to other direction, in particular these conters are increased by one in case of CNT_r_p_1 and CNT_c_p_1, or decreased by one in case of CNT_r_m_1 and CNT_c_m_1 . The starting value of CNT_r_p_1 is CTHU, of CNT_r_m_1 is CTHD, of CNT_c_p_1 is CTVL, and of CNT_c_m_1 is CTVR. In the final part there are two multiplexer to detect the correct address of column and row inside of tile, callend c and r in this work. Respect to the direction, four values of c and r can be selected. If the direction is THU, r is the CTHU and c is the output of CNT_c_p_1. If the direction is THD, r is SM and c is the output of CNT_c_m_1. If the direction is TVL, r is output of CNT_r_m_1 and c is CTVL. If the direction is TVR, r is output of CNT_r_p_1 and c is SM1. Whenever a direction is finished, it is verified that the next clockwise direction has the same radius, if the equality is not verified, it is passed to the other direction clockwise. When all four directions have been checked, the radius is decremented and the first direction is re-examined. In order to ensure that during the control of all directions not followed, the outputs i and j are not considered valid, there is a signal that is asserted when the outputs are valid. The entire block is actived by the signal coming from the control unit. The only signals caming to the control unit are enable and ld_radius, all other signal are logically producted inside the block. This block is always actived for to velocity of the architecture except at the beginning of the execution. When the input tile changes, this is actived, because the velocity of warping is higher than tile management then, inside this block, there is a small fifo that guarantees that despite the tile has already changed, specifically four times when the block ends up producing the right addresses, the block works with the correct values of i and j and the correct radius without losing any of them.

3.6.1 FIFO_RC

FIFO_RC is necessary because TILE MANAGEMENT continuously produces the positions of the pixels inside tile, following the order that guarantees the occlusion compatibility, starting from the radius with respect to the entire tile, but this is a slower calculation than the speed of the WARPING blocks. In fact, the four blocks of WARPING in parallel can calculate the positions of all the pixels of the tile in 64 clock cycles, while the TILE MANAGEMENT produces the internal R and C coordinates in 256 clock cycles more we must consider four more clock cycles for every change of direction as a direction is over the TILE MANAGEMENT checks whether the next direction in clockwise direction is to be traversed or not, as already explained in the appropriate section.

3.7 Control Unit

Most of the control signals come from the control unit. It is the heart of architecture and it is organized in different state machines. The main and other four subordinate to the principal. The first one deals with the reading of the main memory, the filling of the correct tile and the subsequent filling of the four fifo inputs, in the block diagram are called FIFO1, FIFO2, FIFO3, FIFO 4, that provide the data, in particular the depth which is used to calculate the position, to the four warping blocks. In addition, it activates the four machines with sun statuses and when a tile is finished it increases the counter of the number of tiles processed and finally when all the tiles have been processed, it asserts the "done" signal and the functioning of the architecture is interrupted also going to turn off the clock,in the test bench, with a clock gating performed by putting in AND the clock supplied with this signal of done for a lower power consumption. The Control unit, also, provides all the enabling, writing and reading signals of all the memories present in the architecture. It provides thanks to some internal counters the addresses for reading the memory containing the input tile and the addresses for the DATA_MEMORY.It contains, also, the counters for the count of the pixels processed within the tile and the number of tiles processed. The main fsm has six states so organized:

- INIT : in this status all control signal are reset. The computation of main radius by the COMPUTING MAIN RADIUS unit is started and the chip select of MAIN MEMORY is asserted. If the MAIN MEMORY is full the first tile could be read, consequently, the machine can pass to the next status, that is DIRECTION status;
- DIRECTION :In this status the DIRECTION MANAGEMENT is activated and then with the main already calculated radius the address of the next tile to read, for compliance with the occusion, is produces by DIRECTION MANAGEMENT. When the valid address is ready, tha VALID_I_J signal

is asserted by the DIRECTION MANAGEMENT and this signal with the address are put in input to the comtrol unit,because the latter distributes to all blocks and to the main memory current address of the tile and with the asserted VALID_I_J signal can be pass to the next status called LOADTILE;

- LOADTILE: In this status the address to read the correct tile is ready, then the read signal of MAIN MEMORY is asserted and the writing signal of TILE MEMORY is asserted to save the read tile by the MAIN MEMORY. So, the the content of TILE MEMORY is changed, then the load radius signal, put in input of TILE MANAGEMENT, is asserted to read the new radius computed by COMPUTING TILE RADIUS. Then with the TILE MEMORY is full, the machine can be pass to the TILE_EN_CU status;

- TILE_EN_CU: In this status the signal to enable the other four state machines are asserted. Since the TILE MEMORY is full the writing signals of the 4 input fifo(FIFO1, FIFO2, FIFO3, FIFO4) are activated and with them the reading signals of the TILE MEMORY are asserted and the different addresses produces by the control unit block are ready and the four parallel readings are done and each read data are put in input of one of the four fifo memories. The computing tile radius is actived and also the enable of DIRECTION TILE MANAGEMENT is actived. All these signals are actived in this status, because the machine is for the majority of the execution time in this state, in fact all the signals asserted in this state are always active for the duration of the warping of the same tile. When a tile is completed and then the output of the couter of the pixels are equal to 256 then the position of all pixels in the tile are computed then the machine can be pass in the CNTTILE status;

- CNTTILE: In this status the enable of the counter of number of processed tile are activated, when the output of this counter is equal to 3072, the number of tile in the MAIN MEMORY, then the frame is finished and the fsm can be in the STATE_END;

- STATE_END: In this status the working of the Architecture is completed then the "DONE" signal is asserted and it is a output of the Architecture, because this signal is put in the test_bench for the clock gating in AND with the clock signal to reduce the power consumation when the working is finished;

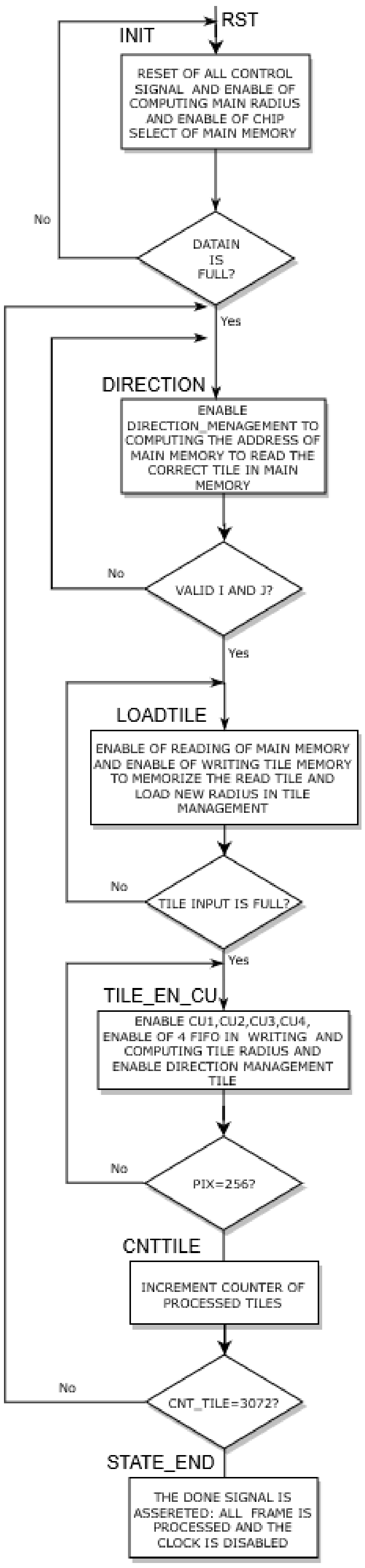

RST
INIT
RESET OF ALL CONTROL SIGNAL AND ENABLE OF COMPUTING MAIN RADIUS AND ENABLE OF CHIP SELECT OF MAIN MEMORY
No
DATAIN IS FULL?
Yes
DIRECTION
ENABLE DIRECTION_MENAGEMENT TO COMPUTING THE ADDRESS OF MAIN MEMORY TO READ THE CORRECT TILE IN MAIN MEMORY
No
VALID I AND J?
Yes
LOADTILE
ENABLE OF READING OF MAIN MEMORY AND ENABLE OF WRITING TILE MEMORY TO MEMORIZE THE READ TILE AND LOAD NEW RADIUS IN TILE MANAGEMENT
No
TILE INPUT IS FULL?
Yes
TILE_EN_CU
ENABLE CU1,CU2,CU3,CU4, ENABLE OF 4 FIFO IN WRITING AND COMPUTING TILE RADIUS AND ENABLE DIRECTION MANAGEMENT TILE
No
PIX=256?
CNTTILE
INCREMENT COUNTER OF PROCESSED TILES
No
CNT_TILE=3072?
STATE_END
THE DONE SIGNAL IS ASSERETED: ALL FRAME IS PROCESSED AND THE CLOCK IS DISABLED

Figura 3.14. The main control unit

When the four enable signals are activated in TILE_EN_CU status the four parallel fsm start to work. The secondary state machines provide all the signals for the management of the whole part that works in parallel. In particular, it is focused on the four units that work in parallel to manage the warping. These signals manage the reading the four inputs fifo (FIFO1, FIFO2, FIFO3, FIFO4), select the correct outputs of the multiplexers inside the unit called TILE WARPING in the WARPING block, manage the writing and reading of the DATA_MEMORY, when the entire tile is processed, fill the FIFO_DATA and its emptying and then the writing of the REORDER MEMORY and the consequent reading. So, the reading of the FIFO_OUT_RC that provides the correct reading addresses of the REORDER MEMORY to provide a correctly ordered data in compliance with the occlusion. The read data are ordered and the htey are ready to be written in the cache. The implemented cache that takes care to keep in this smaller and faster memory the tiles that are filled from time to time, as well as the parallelism of the output of the MAIN MEMORY is of a tile also into the MAIN OUT MEMORY the input parallelism will be the tile,a memory block. The filling of FIFO_DATA could be much faster than the reading and then when this fifo is full the "FULL" signal is asserted by FIFO_DATA and this signal is sent to the control unit, that stops the working of all block except the COMPUTING TILE RADIUS, the TILE MANAGEMENT , the FIFO_OUT_RC, reading of FIFO_DATA and the working of REORDER MEMORY and CACHE memory. In this way the throughput is assured and output continues to be data as if everything continues to work, but the initial part of the architecture is stopped until the signal "FULL" of the fifo does not return low to make sure there is no loss of data. Each secondary state machine has five states so organized:

- T_INIT: In this status the machine resets all counters and all enable of counters and gets ready the selection signals of multiplexer and the SUB/ADDER, a adder that can add or subtract its two inputs, of Warping block. In particular, to further optimize the execution already from this state, the selection signals of both the multiplexers and the SUB / ADDER are already sets as those necessary for calculating the position of the first pixel of the new tile, an operation managed by FIRST status, but the counters in this state are not enabled. When the enable is asserted by the main control unit the machine can pass to the FIRST status;

- FIRST: Considering that in T_INIT the selection signals are set as ones of this status, only in this status, since the working of fsm is correctly started, the enables of the counter of general computed position of pixel and of the counter of computed position of pixel in the same row are asserted. After the one clock cycle the machine goes to H_UP status;

- H_UP: In this status the processed pixels are of the same row and the number of column is increased one by one up to the end of the row, when output of counter of columns is equal to fifteen. The selection signals of multiplexers and SUB/ADDER are changed as explained in the WARPING section. The enable of the counter of general computed position of pixel and of the counter of computed position of pixel in the same row are activated. When all sixteen positions of pixels in same row are finished the machine can go to the R_PLUS status;

- R_PLUS: In this status since the columns with the same row are finished, the enable of counter of the row is increased by one, but the enable of counter of column is set to zero and then the number of the column remains that of the last and the enable of the counter of general computed position of pixel is asserted. If the output of counter of rows is a odd number,one of three,the machine goes to H_DOWN status,instead, if the output of counter of rows is even, then 2, the machine goes to H_UP status;

- H_DOWN: In this status the processed pixels are of the same row and the number of column is decresed one by one up to the end of the row, then output of counter of columns is equal to zeros. The selection signals of multiplexers and SUB/ADDER are changed as explained in the WARPING section. The enable of the counter of general computed position of pixel and of the counter of computed position of pixel in the same row are activated. When all sixteen positions of pixels in same row are finished there are two options: if the output of counter of general computed position of pixel is equal to 64, then all positions of pixels for the entire tile are computed, the machine goes to T_INIT status and it waits for the new enable. If, instead, the output of counter of general computed position of pixel is lower than 64, the machine goes to R_PLUS status to start the new row.

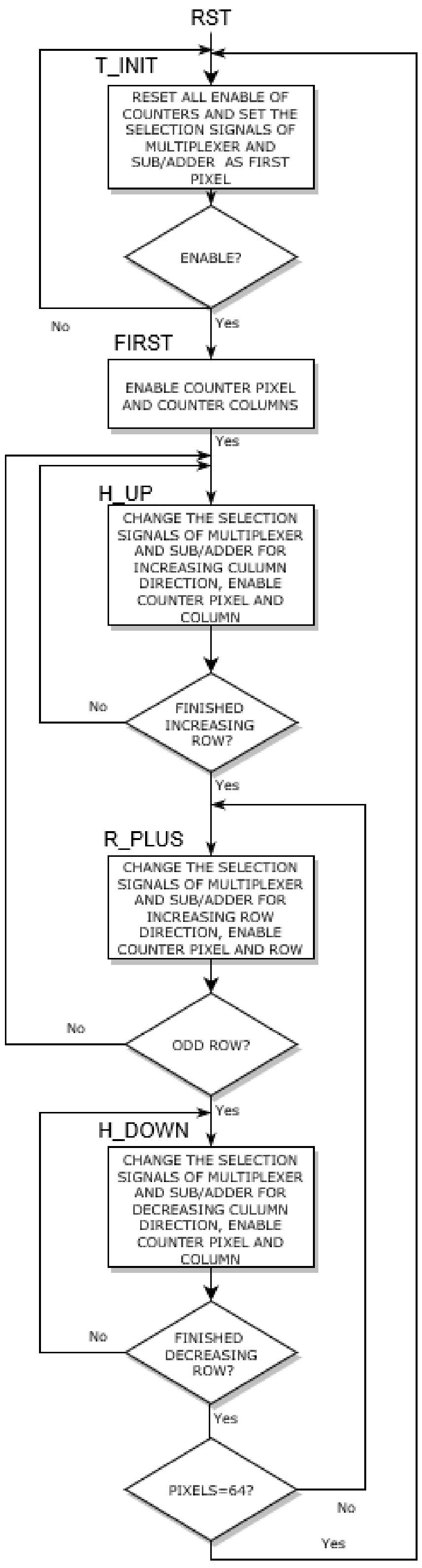

RST
T_INIT
RESET ALL ENABLE OF COUNTERS AND SET THE SELECTION SIGNALS OF MULTIPLEXER AND SUB/ADDER AS FIRST PIXEL
ENABLE?
No
Yes
FIRST
ENABLE COUNTER PIXEL AND COUNTER COLUMNS
Yes
H_UP
CHANGE THE SELECTION SIGNALS OF MULTIPLEXER AND SUB/ADDER FOR INCREASING CULUMN DIRECTION, ENABLE COUNTER PIXEL AND COLUMN
No
FINISHED INCREASING ROW?
Yes
R_PLUS
CHANGE THE SELECTION SIGNALS OF MULTIPLEXER AND SUB/ADDER FOR INCREASING ROW DIRECTION, ENABLE COUNTER PIXEL AND ROW
No
ODD ROW?
Yes
H_DOWN
CHANGE THE SELECTION SIGNALS OF MULTIPLEXER AND SUB/ADDER FOR DECREASING CULUMN DIRECTION, ENABLE COUNTER PIXEL AND COLUMN
No
FINISHED DECREASING ROW?
Yes
PIXELS=64?
No
Yes

Figura 3.15. The secondary finite state machine

3.8 Output Cache

In order to choose the correct dimension of the output cache, the behaviour of the warping is been studied. To understand how many output tiles were covered by a only one input tile the plots of behaviour are producted. The only figures 3.16, 3.8 and 3.18 are reported, because they result the only different from zero. From figures it is clear that the pixels of one only input tile are located in the three different output tile, in biggest number result located in the output tile with same address of input tile, some located in the output tile with same column addresses but with row addresses one tile higher than input tile, others located in the output tile with same row addresses respect to input tile but with culumn addresses one tile higher than input tile. It means that the output cache has to contain three different tile in same time and they are the processed tile, a tile processed by previous input tile and ones covered by next forty-eighth input tile respect to the order of warping of this dataset 2.

In following images these results are shown.

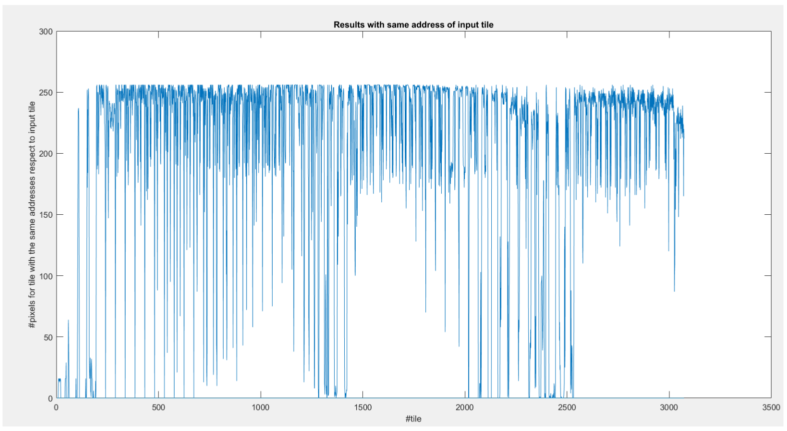


Figura 3.16. Number of pixels for tile with adresses equal to input tile

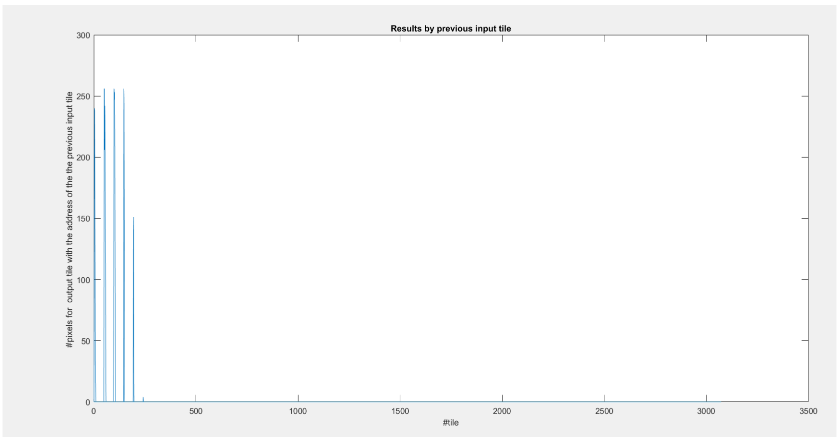


Figura 3.17. Number of pixels for tile with adresses equal to previous input tile

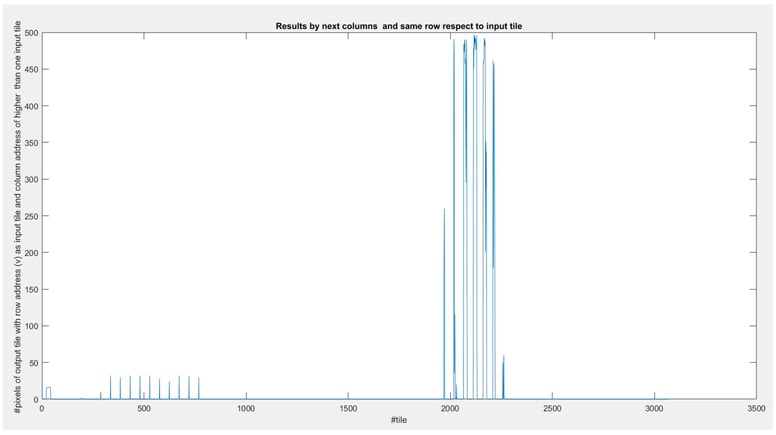


Figura 3.18. Number of pixels for tile with row adresses equal to input tile and column one tile higher than input tile

The previous tile has to be mantained in memory only for first ten processed tile to avoid high miss rate, as it is shown in the figure 3.8, because in the other processed tile the warping pixels remain located in the other two possible output tile. The chosen type of cache is the full-associative cache, because the block of the used main memory block could be different by the study computed only on the data set and the number of necessary tile could be higher than three for the generic data set. For all these considerations the cache is with full associative mapping and with 8 lines then 8 tile and in case of miss the tecnique used to the replacement is LRU, then the replaced line is one that is not used by more time. The managment of the address of cache is obtained with the z-order. Morton's order maps multidimensional data to one dimension while preserving locality of the data points. The address in multidimensions is simply calculated by interleaving the binary representations of its coordinate values. In 3.1 it is reported the organization of the implemented cache containing four filds and they are the VALID, TAG, DATA, LRU.

The first field is VALID, it is a bit set to one when the corresponding

Tabella 3.1. Cache Organization

VALID(1 bit)	TAG (12 bit)	DATA(TILE)	LRU(integer)
VALID0	TAG0	LINE0	LRU0
VALID1	TAG1	LINE1	LRU1
VALID2	TAG2	LINE2	LRU2
VALID3	TAG3	LINE3	LRU3
VALID4	TAG4	LINE4	LRU4
VALID5	TAG5	LINE5	LRU5
VALID6	TAG6	LINE6	LRU6
VALID7	TAG7	LINE7	LRU7

line is valid and it is set to zero when not. The second field is TAG and it is the address to identify the line in the the cache, when the new value has to be writen in the cache tha tag of the input are compared with the tag of all line to detect if there is a hit condition or the miss condition. When there is a hit consition, the real location of input data is composed by the offset value that detect the position inside the line. The third field is the DATA, that rapresents the lines of the cache, in this case a tile. The last field is the LRU and it rappresent a counter associated to the each line to save the age of each tile. When the write back has to be made, in order to understand which line must be writen in the main out memory, so that the locations occupied by that tile can be overwritten, the LRU field is read and the LINE with the highest age are chosen.

3.8.1 Cache control

In order to control the working of the cache the control unit is created. The cache control is composed by 4 states:

- INIT : in this status the cache does not work, but wait the arrive of the signal, in this case writing signal. When signal arrives the cache control goes in the compare status;
- COMPARE : In this status the machine compares the input tag with each tags in the cache. If the tile is in cache the HIT signal becomes one and the system goes to CNT_AGE status, if the tile is not in cache, there is the MISS situation and then the system goes to WRITE_BACK status;
- WRITE_BACK:In this status the ages of tiles are controlled and the tile with highest age is written in the main out memory. In this way, the content of line can be rewritten without loss;
- CNT_AGE: This status is the status where the system increments the counter of each line and resets the couter of the line just written(LRU);

In figure 3.19 the control unit of the cache si reported.

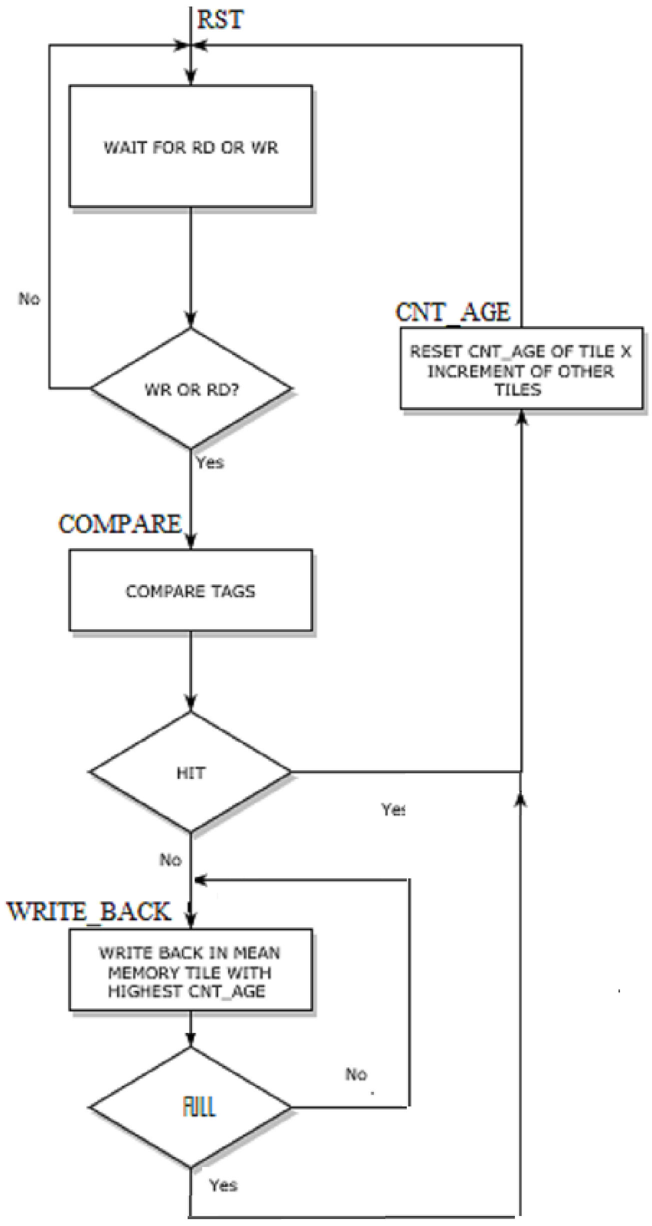

RST
WAIT FOR RD OR WR
No
WR OR RD?
Yes
CNT_AGE
RESET CNT_AGE OF TILE X INCREMENT OF OTHER TILES
COMPARE
COMPARE TAGS
HIT
Yes
No
WRITE_BACK
WRITE BACK IN MEAN MEMORY TILE WITH HIGHEST CNT_AGE
FULL
No
Yes

Figura 3.19. The cache control fsm

Capitolo 4

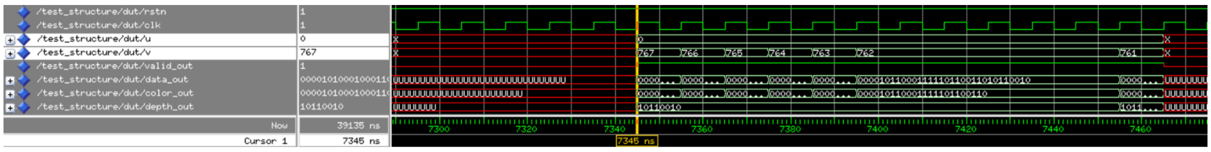
Simulations

This section shows all the operational checks carried out by simulation with modelsim. The test_bench created to test the architecture is hierarchically divided into two parts: a part in which the architecture interacts with the external input and output memories, called IN_OUT_structure and a part in which the parameters are assigned and in which the writing of the main memoryis enabled, called Test_structure. In IN_OUT_structure the main input memory is first filled by reading four files provided by MATLAB with 786432 data each, 3 files for the colors with respect to RGB model and 1 file for depth. These 4 integer values read from files, are converted into unsigned on 8 bits and then concatenated forming a 32 bit input, where the last 8 bits are relative to the depth and therefore necessary for warping. The main architecture called, RENDERING_L, is connected to output of the main memory already filled and with the output memory. The data that come out ready by the architecture are written in cache, as explained in the appropriate section and then, when a tile is complete, is written in the main output memory. During these steps the data provided by the RENDERING_L are also written to the output files. These output files are two, one with the color data and one with the data related to the depth, these data are then inserted into a MATLAB script and compared with the data produced by the software to also graphically verify the correct functioning of the architecture . In Test_structure there is a input generation. In particular, in a first period the writing of the main memory is activated, in such a way that the other part can fill it by reading the file. In a second step the writing of the main input memory is disabled and all the parameter values are supplied at the input of the architecture that will contain them in the input registers. In table 4.1 the values of the parameters obtained from MATLAB and transformed into fixed point are shown. In the last column of the table, there are the real values obtained by the convertion into a fixed point which deviates from the value obtained in floating point in double precision on MATLAB. As can be seen in 4.1 increasing values the error obtained decreases. Then the result obtained by the simulation are correct and similar to ones of MATLAB.

Tabella 4.1. Simulation parameters

PARAM	MATLAB VAL	FIXED POINT RAP.	REAL VAL
DEN_5L	4.5532	00000000000000000100.100011	4.5469 $(291 * 2^{-6})$
DEN_6L	-0.5409	11111111111111111111.011110	0.5313 $(-34 * 2^{-6})$
DEN_7L	1.0554E+05	0011001110001000100.000000	10554 $(6754560 * 2^{-6})$
U_PAR1L	5.1415E+04	0000001010000010101.100000	5141.5 $(329056 * 2^{-6})$
U_PAR2L	2.1419	0000000000000000010.001001	2.1406 $(137 * 2^{-6})$
U_PAR3L	0.0506	0000000000000000000.000011	0.0469 $(3 * 2^{-6})$
U_PAR4L	-212.5915	1111111111100101011.011011	-212.5781 $(-13605 * 2^{-6})$
V_PAR1L	-1.5239E+03	1111111101000001100.000111	-1523.9 $(-97529 * 2^{-6})$
V_PAR2L	0.7784	0000000000000000000.110001	0.7656 $(49 * 2^{-6})$
V_PAR3L	-70.6859	1111111111110111001.010101	-70.6719 $(-4523 * 2^{-6})$
V_PAR4L	-811.0867	1111111110011010100.111011	-811.0781 $(-51909 * 2^{-6})$

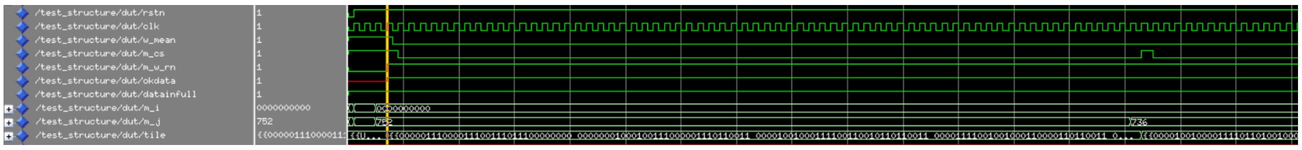
As shown in the table 4.1 the parameters are reffered to the only left camera. The simulation is done for the left frame, the only differece with right frame is the parameters, but the behaviour of structure is the same. To obtain the final fused image of virtual frame it is neccessary only put two istances of the structure and then fusing the images. Observing the values of the coordinates of the epipole, corresponding to the values called in 4.1 and in the rest of the work V_PAR1L and U_PAR1L , it is reasonabe to locate the epipole about in (-1523, 51415). Then considering the frame rappresentation 3.5 and the corner coordinates in 3.3, it is evident that the epipole is located at a point outside the frame and in particular, considering that the value of the coordinates v decreases along the VL direction, because 0 is the v coordinate of START_UP and 767 is the coordinate of START_L, and that the value of the coordinates u increases along the HU direction, because 0 is the u coordinate of START_UP and 1023 is the coordinate of START_R, the epipole is located in a point very far by the frame and the nearest point of the frame respect to the epipole is the START_R and the farest point is START_L. So by the analysis of the data it turns out that the direction that the architecture, in particular, the DIRECTION MANAGEMENT block for the frame and therefore the reading of the main memory and TILE MANAGEMENT to reorder the output tile must be VL and TVL at the level of internal tile. So the direction covered by the same radius should only be the one for both levels. So at the frame level the radius should decrease by 16 times for 48 times and the architecture should then traverse 48 times the VL direction. The same considerations apply to the internal level of the tile, but this time the radius should decrease by one for 16 times following each time the TVL direction.



```
/test_structure/dut/rstn          1
/test_structure/dut/clk           1
/test_structure/dut/u             0
/test_structure/dut/v             767
/test_structure/dut/valid_out     1
/test_structure/dut/data_out      000010100010001100
/test_structure/dut/color_out     000010100010001100
/test_structure/dut/depth_out     10110010
                           Now    39135 ns
                      Cursor 1    7345 ns
```

Figura 4.1. The output reordered values written in the cache

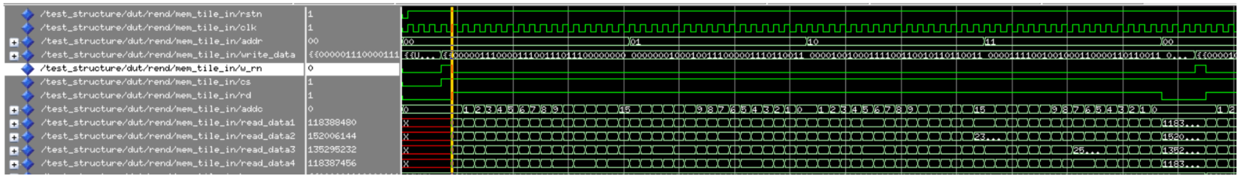
For all considerations jet explained,in figure 4.1 itis shown the correct working of the Architecture.



```
/test_structure/dut/rstn          1
/test_structure/dut/clk           1
/test_structure/dut/w_mean        1
/test_structure/dut/m_cs          1
/test_structure/dut/m_w_rn        1
/test_structure/dut/okdata        1
/test_structure/dut/datainfull    1
/test_structure/dut/m_i           0000000000
/test_structure/dut/m_j           752
/test_structure/dut/tile          {{00000111000011
```

Figura 4.2. Reading and Writing of the main memory

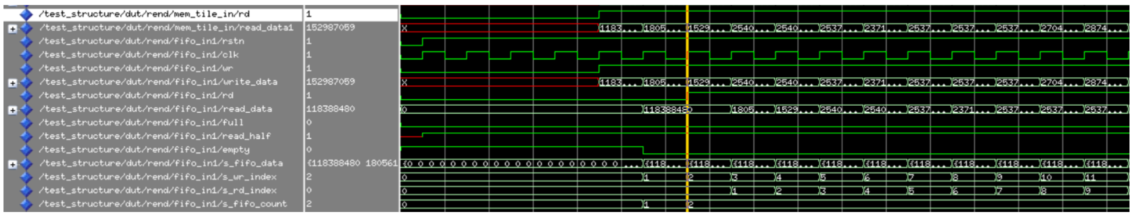
In figure 4.2 the working of main memory is shown. In particular, being the MAIN MEMORY external from the architecture, the attention is for the reading and writing controlled by the MAIN CONTROL UNIT and the address computed in the architecture by the DIRECTION MANAGEMENT and put in input of the main memory by the control unit. The output of the MAIN MEMORY is a tile, that a matrix of 256 pixel(16x16x32) bits.



```
/test_structure/dut/rend/mem_tile_in/rstn          1
/test_structure/dut/rend/mem_tile_in/clk           1
/test_structure/dut/rend/mem_tile_in/addr          00
/test_structure/dut/rend/mem_tile_in/write_data    {{00000111000001111
/test_structure/dut/rend/mem_tile_in/w_rn          0
/test_structure/dut/rend/mem_tile_in/cs            1
/test_structure/dut/rend/mem_tile_in/rd            1
/test_structure/dut/rend/mem_tile_in/addc          0
/test_structure/dut/rend/mem_tile_in/read_data1    118388480
/test_structure/dut/rend/mem_tile_in/read_data2    152006144
/test_structure/dut/rend/mem_tile_in/read_data3    135295232
/test_structure/dut/rend/mem_tile_in/read_data4    118387456
```

Figura 4.3. Reading and Writing of the tile memory

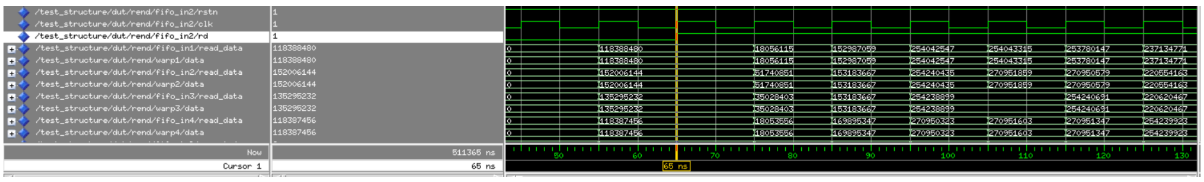
The read tile is writen in the tile memory, that, in this first version of the architecture, is considered with a tile input parallelism. The load of tile in tile memory and the reading of them is shown in 4.3.The address for the reading is generated by the control unit. In this first version the outputs of tile memory are four, one for each input fifo as shown in figure 4.4.



```
/test_structure/dut/rend/mem_tile_in/rd             1
/test_structure/dut/rend/mem_tile_in/read_data1     152987059
/test_structure/dut/rend/fifo_in1/rstn              1
/test_structure/dut/rend/fifo_in1/clk               1
/test_structure/dut/rend/fifo_in1/wr                1
/test_structure/dut/rend/fifo_in1/write_data        152987059
/test_structure/dut/rend/fifo_in1/rd                1
/test_structure/dut/rend/fifo_in1/read_data         118388480
/test_structure/dut/rend/fifo_in1/full              0
/test_structure/dut/rend/fifo_in1/read_half         1
/test_structure/dut/rend/fifo_in1/empty             0
/test_structure/dut/rend/fifo_in1/s_fifo_data       {118388480 180561
/test_structure/dut/rend/fifo_in1/s_wr_index        2
/test_structure/dut/rend/fifo_in1/s_rd_index        0
/test_structure/dut/rend/fifo_in1/s_fifo_count      2
```

Figura 4.4. Data transition between tile memory and fifo

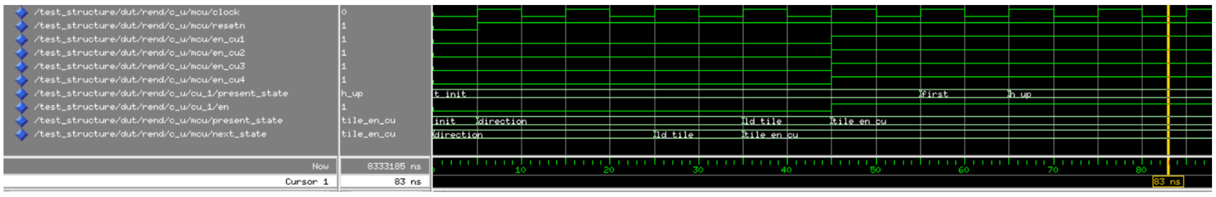
When the data from the fifo are reading the warping can start to work. The correct transition between fifo e warping block is shown in 4.5.



```
/test_structure/dut/rend/fifo_in2/rstn              1
/test_structure/dut/rend/fifo_in2/clk               1
/test_structure/dut/rend/fifo_in2/rd                1
/test_structure/dut/rend/fifo_in1/read_data         118388480
/test_structure/dut/rend/warp1/data                 118388480
/test_structure/dut/rend/fifo_in2/read_data         152006144
/test_structure/dut/rend/warp2/data                 152006144
/test_structure/dut/rend/fifo_in3/read_data         135295232
/test_structure/dut/rend/warp3/data                 135295232
/test_structure/dut/rend/fifo_in4/read_data         118387456
/test_structure/dut/rend/warp4/data                 118387456
                                          Now       511365 ns
                                     Cursor 1           65 ns
```

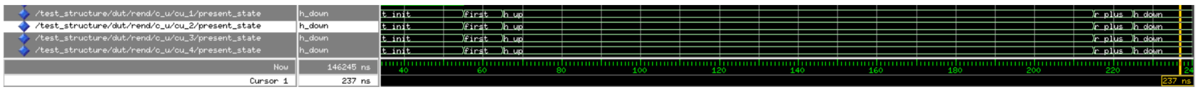
Figura 4.5. Data transition between fifo and warping

In figures 4.6 and 4.7 are reported the transitions of the main and the secondary fsm. In the first, it is shown that the main fsm starts by the init status and then it stays in direction status for three clock cycles, to compute the correct address for main main memory and then in ld_tile reading the correct tile by main memory and load it in tile memory. At this point, when the data are ready, they are writen in the fifo and ready for the warping block. So, in tile_en_cu status ,as shown, the enable signals of all four secondary control units are asserted and after one clock cycle there is a transition in the secondary fsm from t_init status to first status, to compute the first pixel position. In 4.7 it is proved that the secondary fsm work in the same way.



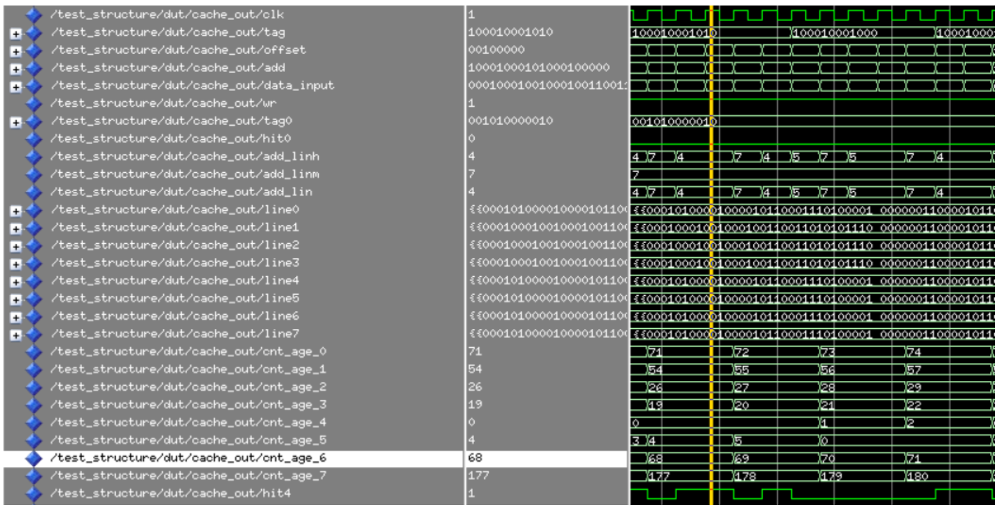
```
/test_structure/dut/rend/c_u/mcu/clock              0
/test_structure/dut/rend/c_u/mcu/resetn             1
/test_structure/dut/rend/c_u/mcu/en_cu1             1
/test_structure/dut/rend/c_u/mcu/en_cu2             1
/test_structure/dut/rend/c_u/mcu/en_cu3             1
/test_structure/dut/rend/c_u/mcu/en_cu4             1
/test_structure/dut/rend/c_u/cu_1/present_state     h_up
/test_structure/dut/rend/c_u/cu_1/en                1
/test_structure/dut/rend/c_u/mcu/present_state      tile_en_cu
/test_structure/dut/rend/c_u/mcu/next_state         tile_en_cu
                                          Now       8333185 ns
                                     Cursor 1            83 ns
```

Figura 4.6. Relationship between main and secondary states machines



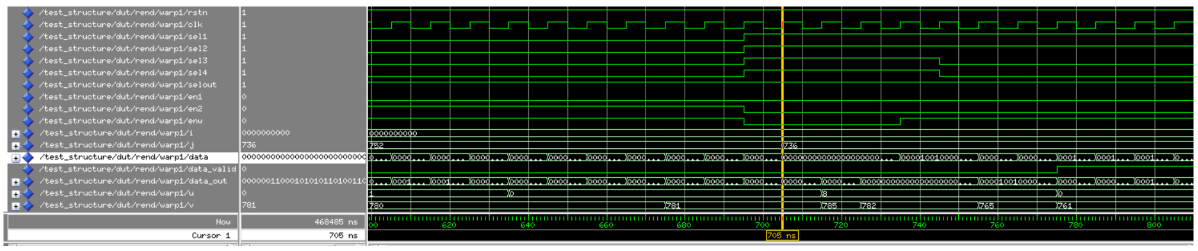
```
/test_structure/dut/rend/c_u/cu_1/present_state   h_down
/test_structure/dut/rend/c_u/cu_2/present_state   h_down
/test_structure/dut/rend/c_u/cu_3/present_state   h_down
/test_structure/dut/rend/c_u/cu_4/present_state   h_down
                                            Now   146245 ns
                                       Cursor 1   237 ns
```

Figura 4.7. Transiotions of the states in the secondary fsm



```
/test_structure/dut/cache_out/clk          1
/test_structure/dut/cache_out/tag          10001000101O
/test_structure/dut/cache_out/offset       00100000
/test_structure/dut/cache_out/add          1000100010100010000O
/test_structure/dut/cache_out/data_input   000100010010001001100 1
/test_structure/dut/cache_out/wr           1
/test_structure/dut/cache_out/tag0         00101000001O
/test_structure/dut/cache_out/hit0         0
/test_structure/dut/cache_out/add_linh     4
/test_structure/dut/cache_out/add_linm     7
/test_structure/dut/cache_out/add_lin      4
/test_structure/dut/cache_out/line0        {{000101000010000101100
/test_structure/dut/cache_out/line1        {{000100010010001001100
/test_structure/dut/cache_out/line2        {{000100010010001001100
/test_structure/dut/cache_out/line3        {{000100010010001001100
/test_structure/dut/cache_out/line4        {{000101000010000101100
/test_structure/dut/cache_out/line5        {{000101000010000101100
/test_structure/dut/cache_out/line6        {{000101000010000101100
/test_structure/dut/cache_out/line7        {{000101000010000101100
/test_structure/dut/cache_out/cnt_age_0    71
/test_structure/dut/cache_out/cnt_age_1    54
/test_structure/dut/cache_out/cnt_age_2    26
/test_structure/dut/cache_out/cnt_age_3    19
/test_structure/dut/cache_out/cnt_age_4    0
/test_structure/dut/cache_out/cnt_age_5    4
/test_structure/dut/cache_out/cnt_age_6    68
/test_structure/dut/cache_out/cnt_age_7    177
/test_structure/dut/cache_out/hit4         1
```

Figura 4.8. Working of Cache



```
/test_structure/dut/rend/warp1/rstn         1
/test_structure/dut/rend/warp1/clk          1
/test_structure/dut/rend/warp1/sel1         1
/test_structure/dut/rend/warp1/sel2         1
/test_structure/dut/rend/warp1/sel3         1
/test_structure/dut/rend/warp1/sel4         1
/test_structure/dut/rend/warp1/selout       1
/test_structure/dut/rend/warp1/en1          0
/test_structure/dut/rend/warp1/en2          0
/test_structure/dut/rend/warp1/enw          0
/test_structure/dut/rend/warp1/i            0000000000
/test_structure/dut/rend/warp1/j            736
/test_structure/dut/rend/warp1/data         000000000000000000000000000
/test_structure/dut/rend/warp1/data_valid   0
/test_structure/dut/rend/warp1/data_out     0000001100010101011010011 0
/test_structure/dut/rend/warp1/u            0
/test_structure/dut/rend/warp1/v            781
                                     Now    468485 ns
                                Cursor 1    705 ns
```

Figura 4.9. Results of Warping of first warping block

In figure 4.9 the behaviour of the first Warping block is shown. In particular, as shown by the cursor, the attention is given to the change of the tile. In fact, the value of j goes from 752 to 736, so a new tile with address (736, 0) is read by MAIN MEMORY and until the new tile is ready to be read in TILE MEMORY and the FIFO1 is written the data, remains constant and "enw", that is high when the secondary fsm is doing working, is in this case low. The fact that the tile is ready is signaled by the "enw" signal which returns high. Compared to what in the section 3.5.3, the warping outputs, with respect to the input data, are delayed by two clock cycles, in fact, from the figure, we can see how the first result obtained by the new tile is not valid, but from the second tile the signal validation signal "data_valid" becomes high, consequently the output data corresponding to the calculated coordinates (761, 0) is valid and then written in DATA_MEMORY, where the write signal provided by the control unit is exactly the same as the "data_valid" output signal.

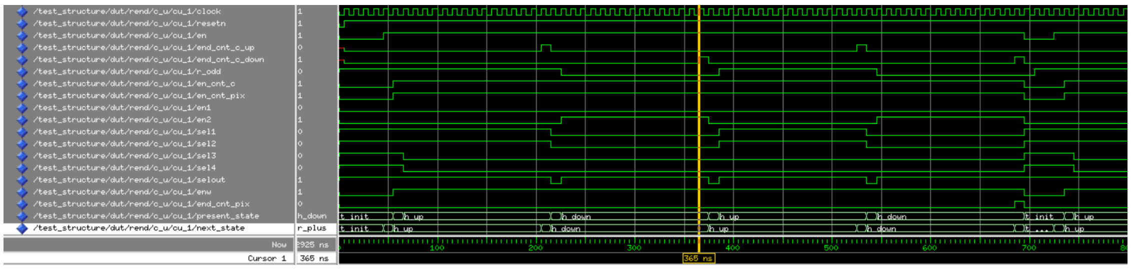

/test_structure/dut/rend/c_u/cu_1/clock 1
/test_structure/dut/rend/c_u/cu_1/resetn 1
/test_structure/dut/rend/c_u/cu_1/en 1
/test_structure/dut/rend/c_u/cu_1/end_cnt_c_up 0
/test_structure/dut/rend/c_u/cu_1/end_cnt_c_down 1
/test_structure/dut/rend/c_u/cu_1/r_odd 0
/test_structure/dut/rend/c_u/cu_1/en_cnt_c 1
/test_structure/dut/rend/c_u/cu_1/en_cnt_pix 1
/test_structure/dut/rend/c_u/cu_1/en1 0
/test_structure/dut/rend/c_u/cu_1/en2 1
/test_structure/dut/rend/c_u/cu_1/sel1 0
/test_structure/dut/rend/c_u/cu_1/sel2 0
/test_structure/dut/rend/c_u/cu_1/sel3 0
/test_structure/dut/rend/c_u/cu_1/sel4 0
/test_structure/dut/rend/c_u/cu_1/selout 1
/test_structure/dut/rend/c_u/cu_1/enw 1
/test_structure/dut/rend/c_u/cu_1/end_cnt_pix 0
/test_structure/dut/rend/c_u/cu_1/present_state h_down
/test_structure/dut/rend/c_u/cu_1/next_state r_plus
Now 2925 ns
Cursor 1 365 ns
t_init
h_up
h_down
365 ns

Figura 4.10. Working of Secondary control unit reffered to the first Warping block

In figure 4.10 the behaviour of Secondary control unit is shown. In this case it is shown the case of the Secondary control unit, that control the first warping block, but the behaviour is the same for all four warping blocks. In particular, the correct transition of the states. The first status is the T_INIT, after there is FIRST status and then the h_up, because the first row is processed from left to right, when the pixel is the last of the row, the row counter is increased in the r_plus status that is shown by the cursor, and then the second row is processed decreasing the column and this operation is made by the h_down status. The multiplexers of the Warping tile block is controlled to do the currect position generation and all signals of selection are shown in the figure in relationship to relative status.

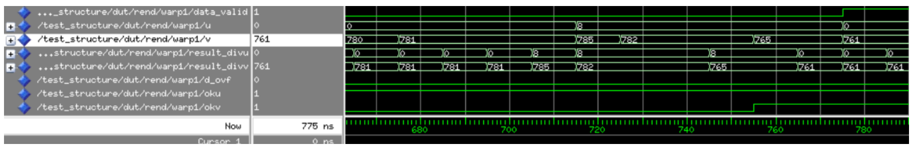
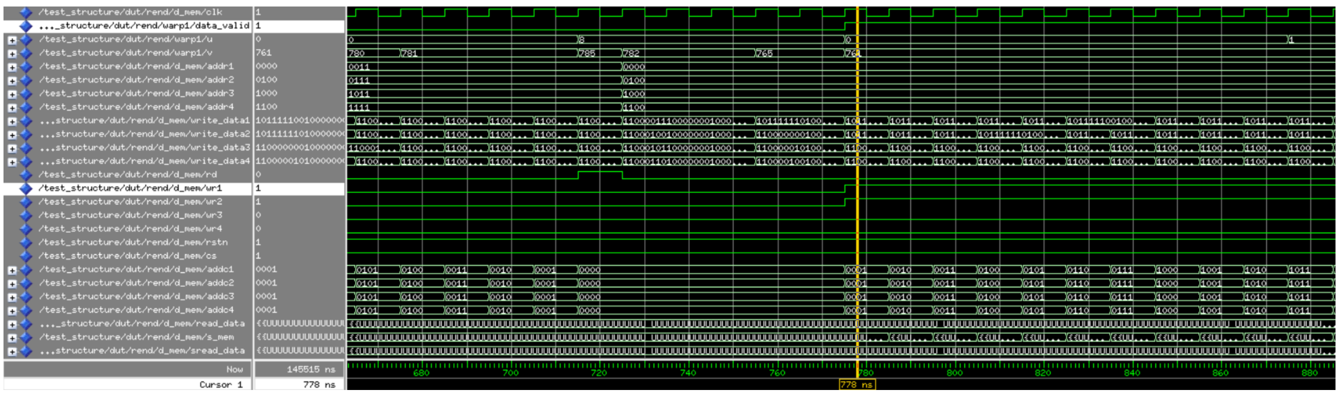

..._structure/dut/rend/warp1/data_valid 1
/test_structure/dut/rend/warp1/u 0
/test_structure/dut/rend/warp1/v 761
...structure/dut/rend/warp1/result_divu 0
...structure/dut/rend/warp1/result_divv 761
/test_structure/dut/rend/warp1/d_ovf 0
/test_structure/dut/rend/warp1/oku 1
/test_structure/dut/rend/warp1/okv 1
Now 775 ns
Cursor 1 0 ns

Figura 4.11. Output of Warping unit

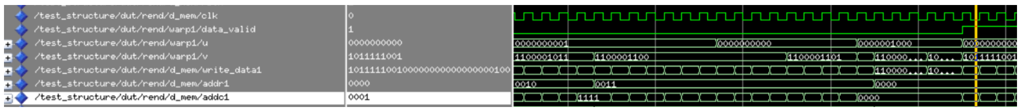
In figure 4.11 the outputs of each warping block are shown. In particular, it is shown the first valid position of coordinates (761,0). These coordinates are valid because , as shown in figure, oku is '1' , okv is '1', and d_ovf is '0'. oku is the signal that is asserted when the u coordinate is in the range [0,1023]. okv is the signal that is asserted when the v coordinate is in the range [0,767].d_ovf is the overflow signal that cames from shift unit, then when this signal is '0' there is not overflow and the data could be valid. datavalid is the signal asserted when the outputs coordinates are valid then when oku is '1' AND okv is '1' AND d_ovf is '0'.



```
/test_structure/dut/rend/d_mem/clk              1
..._structure/dut/rend/warp1/data_valid         1
/test_structure/dut/rend/warp1/u                0
/test_structure/dut/rend/warp1/v                761
/test_structure/dut/rend/d_mem/addr1            0000
/test_structure/dut/rend/d_mem/addr2            0100
/test_structure/dut/rend/d_mem/addr3            1000
/test_structure/dut/rend/d_mem/addr4            1100
...structure/dut/rend/d_mem/write_data1         1011111001000000(
...structure/dut/rend/d_mem/write_data2         1011111101000000(
...structure/dut/rend/d_mem/write_data3         1100000001000000(
...structure/dut/rend/d_mem/write_data4         1100000101000000(
/test_structure/dut/rend/d_mem/rd               0
/test_structure/dut/rend/d_mem/wr1              1
/test_structure/dut/rend/d_mem/wr2              1
/test_structure/dut/rend/d_mem/wr3              0
/test_structure/dut/rend/d_mem/wr4              0
/test_structure/dut/rend/d_mem/rstn             1
/test_structure/dut/rend/d_mem/cs               1
/test_structure/dut/rend/d_mem/addc1            0001
/test_structure/dut/rend/d_mem/addc2            0001
/test_structure/dut/rend/d_mem/addc3            0001
/test_structure/dut/rend/d_mem/addc4            0001
..._structure/dut/rend/d_mem/read_data          {{UUUUUUUUUUUUUUU
/test_structure/dut/rend/d_mem/s_mem            {{UUUUUUUUUUUUUUU
...structure/dut/rend/d_mem/sread_data          {{UUUUUUUUUUUUUUU
                                        Now     145515 ns
                                   Cursor 1     778 ns
```

Figura 4.12. Writing of the DATA MEMORY, ADDRESS and DATA_VALID

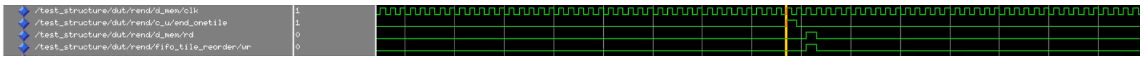
When a computed data of warping is ready and valid, it is saved in the DATA MEMORY. In figure 4.12, as previous figure, the case of first valid value is shown. Then when the valid value is prodeced, the write signal, related to the corresponding port in the memory, is asserted. When the data valid is produced by fisrt warping block, WR1 in data memory is asserted. The obtained coordinates value and color bits and depth bits are saved in data memory in the location corresponding to the input pixel lacations that has generated this value. The correspondig address of data memory is computed by the control unit.



```
/test_structure/dut/rend/d_mem/clk              0
/test_structure/dut/rend/warp1/data_valid       1
/test_structure/dut/rend/warp1/u                0000000000
/test_structure/dut/rend/warp1/v                1011111001
/test_structure/dut/rend/d_mem/write_data1      101111100100000000000000000100
/test_structure/dut/rend/d_mem/addr1            0000
/test_structure/dut/rend/d_mem/addc1            0001
```

Figura 4.13. Verification of the correctness of the written data

In figure 4.12 it can see that in the data memory is writen the correct value, because the writen value is obtained by the concatenation of v, u , and data and in the "write_data1" there is a correspondig value. When a tile is all processed, the data memory contains all the locations correspondig to this input tile, then the data memory is read and content is saved in the fifo tile reorder. In this way, there is not a lost of information when the new tile is loaded.



```
/test_structure/dut/rend/d_mem/clk                  1
/test_structure/dut/rend/c_u/end_onetile            1
/test_structure/dut/rend/d_mem/rd                   0
/test_structure/dut/rend/fifo_tile_reorder/wr       0
```

Figura 4.14. Synchronization between read signal data memory and writing signal fifo tile reorder

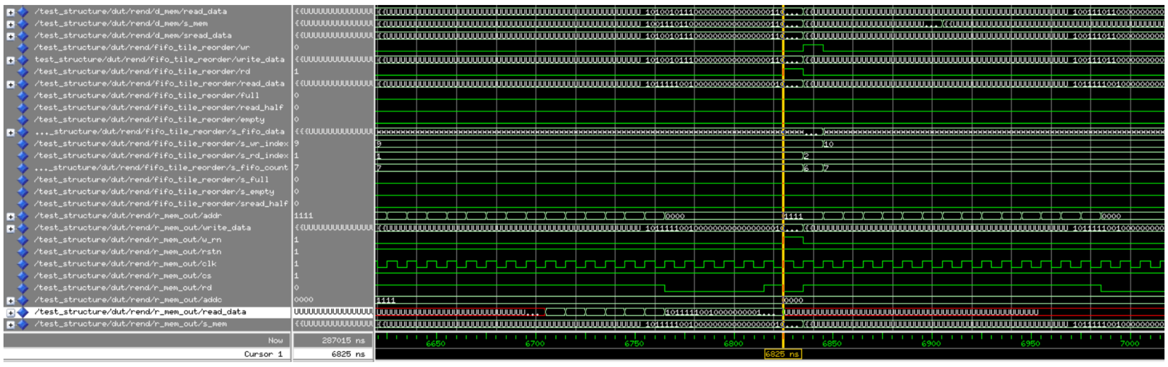

/test_structure/dut/rend/d_mem/read_data
/test_structure/dut/rend/d_mem/s_mem
/test_structure/dut/rend/d_mem/sread_data
/test_structure/dut/rend/fifo_tile_reorder/wr
test_structure/dut/rend/fifo_tile_reorder/write_data
/test_structure/dut/rend/fifo_tile_reorder/rd
/test_structure/dut/rend/fifo_tile_reorder/read_data
/test_structure/dut/rend/fifo_tile_reorder/full
/test_structure/dut/rend/fifo_tile_reorder/read_half
/test_structure/dut/rend/fifo_tile_reorder/empty
..._structure/dut/rend/fifo_tile_reorder/s_fifo_data
/test_structure/dut/rend/fifo_tile_reorder/s_wr_index
/test_structure/dut/rend/fifo_tile_reorder/s_rd_index
..._structure/dut/rend/fifo_tile_reorder/s_fifo_count
/test_structure/dut/rend/fifo_tile_reorder/s_full
/test_structure/dut/rend/fifo_tile_reorder/s_empty
/test_structure/dut/rend/fifo_tile_reorder/sread_half
/test_structure/dut/rend/r_mem_out/addr
/test_structure/dut/rend/r_mem_out/write_data
/test_structure/dut/rend/r_mem_out/w_rn
/test_structure/dut/rend/r_mem_out/rstn
/test_structure/dut/rend/r_mem_out/clk
/test_structure/dut/rend/r_mem_out/cs
/test_structure/dut/rend/r_mem_out/rd
/test_structure/dut/rend/r_mem_out/addc
/test_structure/dut/rend/r_mem_out/read_data
/test_structure/dut/rend/r_mem_out/s_mem
Now 287015 ns
Cursor 1 6825 ns

Figura 4.15. Transitions data between data memory, fifo tile reorder and reorder memory

In figure 4.15 the transition of the data between data memory, fifo tile reorder and then reorder memory are shown. The data are writen in reorder memory and the read data by the reorder memory are the data reordered respect to the occlusion compatibility. For this purpose the addresses of this memory are computed by the tile menagement and then saved in fifo rc.

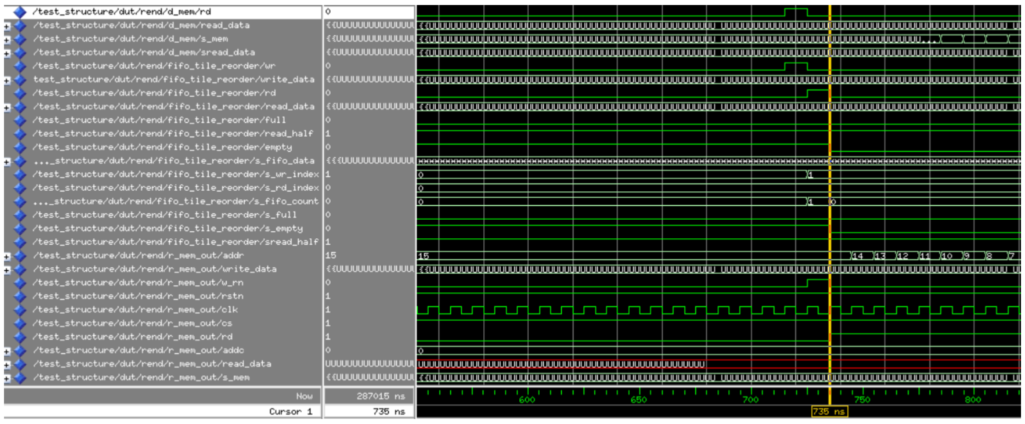

/test_structure/dut/rend/d_mem/rd
/test_structure/dut/rend/d_mem/read_data
/test_structure/dut/rend/d_mem/s_mem
/test_structure/dut/rend/d_mem/sread_data
/test_structure/dut/rend/fifo_tile_reorder/wr
test_structure/dut/rend/fifo_tile_reorder/write_data
/test_structure/dut/rend/fifo_tile_reorder/rd
/test_structure/dut/rend/fifo_tile_reorder/read_data
/test_structure/dut/rend/fifo_tile_reorder/full
/test_structure/dut/rend/fifo_tile_reorder/read_half
/test_structure/dut/rend/fifo_tile_reorder/empty
..._structure/dut/rend/fifo_tile_reorder/s_fifo_data
/test_structure/dut/rend/fifo_tile_reorder/s_wr_index
/test_structure/dut/rend/fifo_tile_reorder/s_rd_index
..._structure/dut/rend/fifo_tile_reorder/s_fifo_count
/test_structure/dut/rend/fifo_tile_reorder/s_full
/test_structure/dut/rend/fifo_tile_reorder/s_empty
/test_structure/dut/rend/fifo_tile_reorder/sread_half
/test_structure/dut/rend/r_mem_out/addr
/test_structure/dut/rend/r_mem_out/write_data
/test_structure/dut/rend/r_mem_out/w_rn
/test_structure/dut/rend/r_mem_out/rstn
/test_structure/dut/rend/r_mem_out/clk
/test_structure/dut/rend/r_mem_out/cs
/test_structure/dut/rend/r_mem_out/rd
/test_structure/dut/rend/r_mem_out/addc
/test_structure/dut/rend/r_mem_out/read_data
/test_structure/dut/rend/r_mem_out/s_mem
Now 287015 ns
Cursor 1 735 ns

Figura 4.16. Passing data between DATA MEMORY, FIFO_TILE_REORDER and REORDER MEMORY

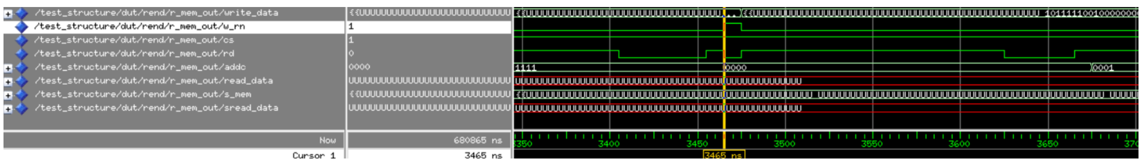
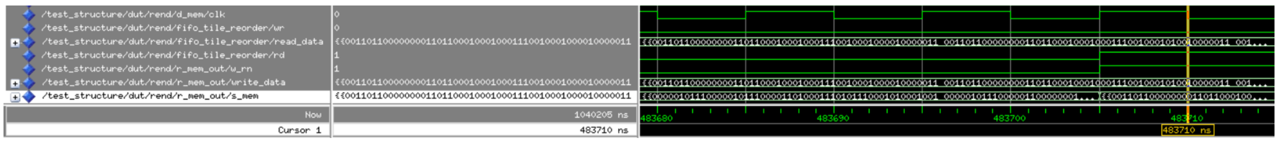

/test_structure/dut/rend/r_mem_out/write_data
/test_structure/dut/rend/r_mem_out/w_rn
/test_structure/dut/rend/r_mem_out/cs
/test_structure/dut/rend/r_mem_out/rd
/test_structure/dut/rend/r_mem_out/addc
/test_structure/dut/rend/r_mem_out/read_data
/test_structure/dut/rend/r_mem_out/s_mem
/test_structure/dut/rend/r_mem_out/sread_data
Now 680865 ns
Cursor 1 3465 ns

Figura 4.17. Writing of REORDER_MEMORY

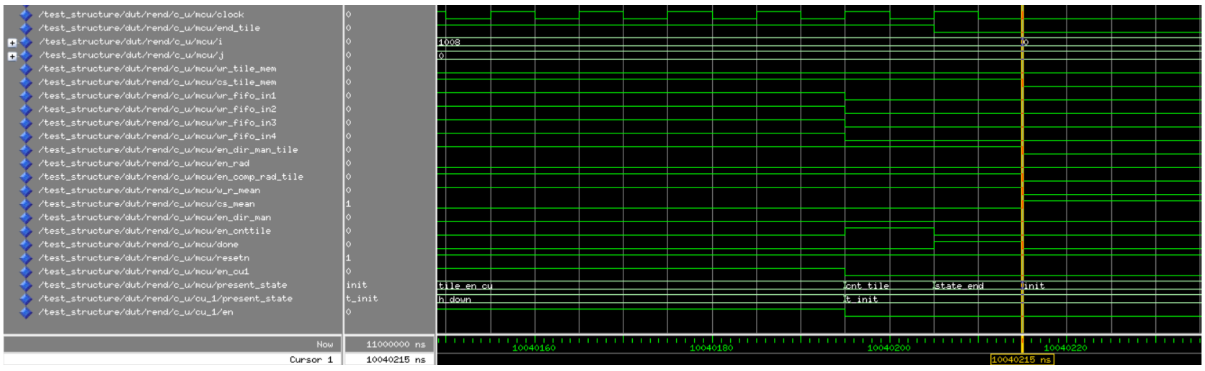
In the figure 4.17 the Writing of REORDER_MEMORY is shown. It demonstrates how despite the WRITE_DATA changes to the memory input the writing, being synchronous is really done only when there is the relative signal "WR".



```
/test_structure/dut/rend/d_mem/clk                      0
/test_structure/dut/rend/fifo_tile_reorder/wr           0
/test_structure/dut/rend/fifo_tile_reorder/read_data    {{0011011000000000110110001000100011100100010000100000011
/test_structure/dut/rend/fifo_tile_reorder/rd           1
/test_structure/dut/rend/r_mem_out/w_rn                 1
/test_structure/dut/rend/r_mem_out/write_data           {{0011011000000000110110001000100011100100010000100000011
/test_structure/dut/rend/r_mem_out/s_mem                {{0011011000000000110110001000100011100100010000100000011
                                                Now     1040205 ns
                                           Cursor 1     483710 ns
```

Figura 4.18. Writing REORDER MEMORY and reading FIFO_TILE_REORDER

In figure 4.19 the end of the working of architecture is shown. When all tiles are processed, the clock gating turn off the clock and the value of input tile address that, as shown, for the last tile was (1008,0) is resetted and the main fsm, after state_end status, returns to the init status to wait the start of the new frame.



```
/test_structure/dut/rend/c_u/mcu/clock                  0
/test_structure/dut/rend/c_u/mcu/end_tile               0
/test_structure/dut/rend/c_u/mcu/i                      0
/test_structure/dut/rend/c_u/mcu/j                      0
/test_structure/dut/rend/c_u/mcu/wr_tile_mem            0
/test_structure/dut/rend/c_u/mcu/cs_tile_mem            0
/test_structure/dut/rend/c_u/mcu/wr_fifo_in1            0
/test_structure/dut/rend/c_u/mcu/wr_fifo_in2            0
/test_structure/dut/rend/c_u/mcu/wr_fifo_in3            0
/test_structure/dut/rend/c_u/mcu/wr_fifo_in4            0
/test_structure/dut/rend/c_u/mcu/en_dir_man_tile        0
/test_structure/dut/rend/c_u/mcu/en_rad                 0
/test_structure/dut/rend/c_u/mcu/en_comp_rad_tile       0
/test_structure/dut/rend/c_u/mcu/w_r_mean               0
/test_structure/dut/rend/c_u/mcu/cs_mean                1
/test_structure/dut/rend/c_u/mcu/en_dir_man             0
/test_structure/dut/rend/c_u/mcu/en_cnttile             0
/test_structure/dut/rend/c_u/mcu/done                   0
/test_structure/dut/rend/c_u/mcu/resetn                 1
/test_structure/dut/rend/c_u/mcu/en_cu1                 0
/test_structure/dut/rend/c_u/mcu/present_state          init
/test_structure/dut/rend/c_u/cu_1/present_state         t_init
/test_structure/dut/rend/c_u/cu_1/en                    0
                                                Now     11000000 ns
                                           Cursor 1     10040215 ns
```

Figura 4.19. Done signal and relative behaviour of the structure

In figures 4.20 and 4.21 it is shown the correct behaviour of the cache. In the first, it is shown the writing of the first tile and then, as shown by the fsm, there is a compare status that detects the miss situation (hit='0') and then there is a write back status. At this point, the line_add of the line that has to be writen back in the main memory is the line 0 with all zeros tag value. When there is the write back, the line 0 have the new tag value equal to the previous tag input and the value are saved, updating the new count age for the lsu to choose the line to be replaced in write back status.

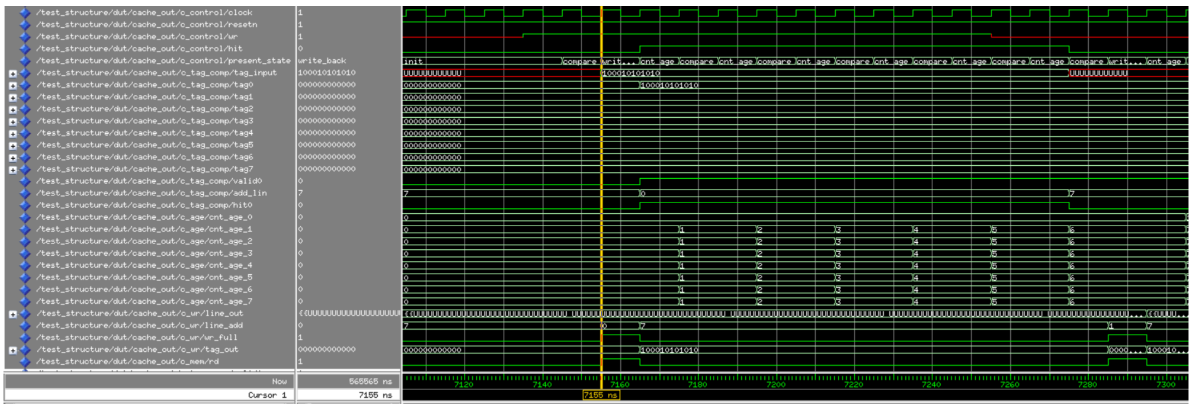

/test_structure/dut/cache_out/c_control/clock
/test_structure/dut/cache_out/c_control/resetn
/test_structure/dut/cache_out/c_control/wr
/test_structure/dut/cache_out/c_control/hit
/test_structure/dut/cache_out/c_control/present_state
/test_structure/dut/cache_out/c_tag_comp/tag_input
/test_structure/dut/cache_out/c_tag_comp/tag0
/test_structure/dut/cache_out/c_tag_comp/tag1
/test_structure/dut/cache_out/c_tag_comp/tag2
/test_structure/dut/cache_out/c_tag_comp/tag3
/test_structure/dut/cache_out/c_tag_comp/tag4
/test_structure/dut/cache_out/c_tag_comp/tag5
/test_structure/dut/cache_out/c_tag_comp/tag6
/test_structure/dut/cache_out/c_tag_comp/tag7
/test_structure/dut/cache_out/c_tag_comp/valid0
/test_structure/dut/cache_out/c_tag_comp/add_lin
/test_structure/dut/cache_out/c_tag_comp/hit0
/test_structure/dut/cache_out/c_age/cnt_age_0
/test_structure/dut/cache_out/c_age/cnt_age_1
/test_structure/dut/cache_out/c_age/cnt_age_2
/test_structure/dut/cache_out/c_age/cnt_age_3
/test_structure/dut/cache_out/c_age/cnt_age_4
/test_structure/dut/cache_out/c_age/cnt_age_5
/test_structure/dut/cache_out/c_age/cnt_age_6
/test_structure/dut/cache_out/c_age/cnt_age_7
/test_structure/dut/cache_out/c_wr/line_out
/test_structure/dut/cache_out/c_wr/line_add
/test_structure/dut/cache_out/c_wr/wr_full
/test_structure/dut/cache_out/c_wr/tag_out
/test_structure/dut/cache_out/c_mem/rd
Now 565565 ns
Cursor 1 7155 ns

Figura 4.20. Correct work of the cache respect to the cache control

In figure 4.21 the correct writing and reading respect to the tag and offset is shown.

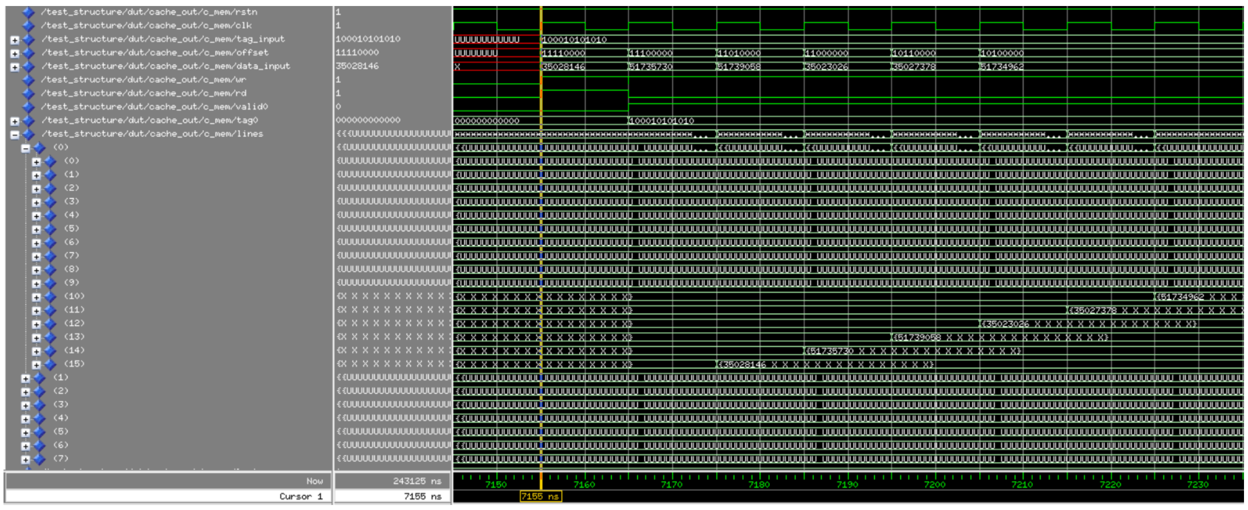

/test_structure/dut/cache_out/c_mem/rstn
/test_structure/dut/cache_out/c_mem/clk
/test_structure/dut/cache_out/c_mem/tag_input
/test_structure/dut/cache_out/c_mem/offset
/test_structure/dut/cache_out/c_mem/data_input
/test_structure/dut/cache_out/c_mem/wr
/test_structure/dut/cache_out/c_mem/rd
/test_structure/dut/cache_out/c_mem/valid0
/test_structure/dut/cache_out/c_mem/tag0
/test_structure/dut/cache_out/c_mem/lines
Now 243125 ns
Cursor 1 7155 ns

Figura 4.21. Correct writing and reading of the cache

Capitolo 5

Critical path and Optimization

In order to synthesize the architecture with synopsys, considering the large number of necessary memories, the each block of the structure without memory are separately synthesized. Firstly, the maximum clock frequency of each block design is found, setting in the clock constraints the clock period equal to zero. The minimum period of each block is reported in table 5.1. From the analysis of the obtained results in the table, as expected, the critical path is obtained in the propagation of the carry out in the divisor. Being the minimum period of divisor equal to 28 ns, it is considered necessary to modify the divisor, pipeling it, in order to reduce the critical path and to obtain, also, a divisor with a minimum period similar to the other minimum period of blocks of the architecture.

Tabella 5.1. Study of critical path

BLOCK	CRITICAL PATH [ns]	AREA[μm^2]
DIVIDER	28.28	520046
TILE_MANAGEMENT	2.03	1899405.3
WARPING	25.09	54699.4
DIRECTION_MANAGEMENT	1.66	6949.4
CU	1.09	17325.3
COMPUTING RADIUS	2.23	4068

5.1 Optimization

5.1.1 Pipelined Divisor

To optimize the architecture, firstly, the divisor is pipelined as shown in figure 5.2. In the figure 5.2 it is reported only a example of pipelined divisor, because the generic implemented divisor is composed by the twenty-five columns for eleven rows, since the parallelism is on 35 bits for the numerator and 25 bits for denominator. The made cutsets are as in figure introducing a level of pipelined in each row, in this way the resulted number of pipeline levels in the divisor are ten and the expected minimum period for the divisor should be about a tenth of the previous one. The structure with a new divisor works correctly and in figure 5.1 it is reported the behaviour of the output data that result corrects. It is reported only a significant simulation to verify that the the pipelined divisor and the entire structure work correctly. In particular to synchronize the signals of control of the parts of the architecture placed after the pipelined divisor.

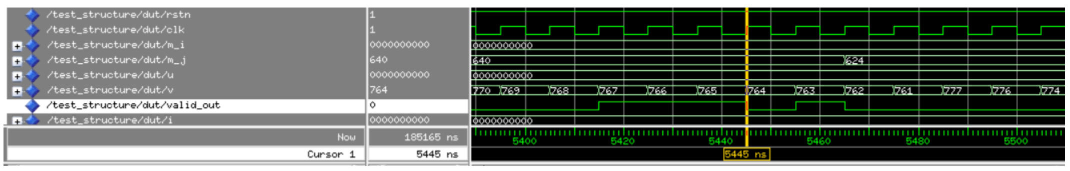


Figura 5.1. The output reordered values written in the cache

Then in order to verify that the applied pipeline confirms the expected result, the synthesis of the new divisor block is carried out, setting in the clock constraints the clock period equal to zero.

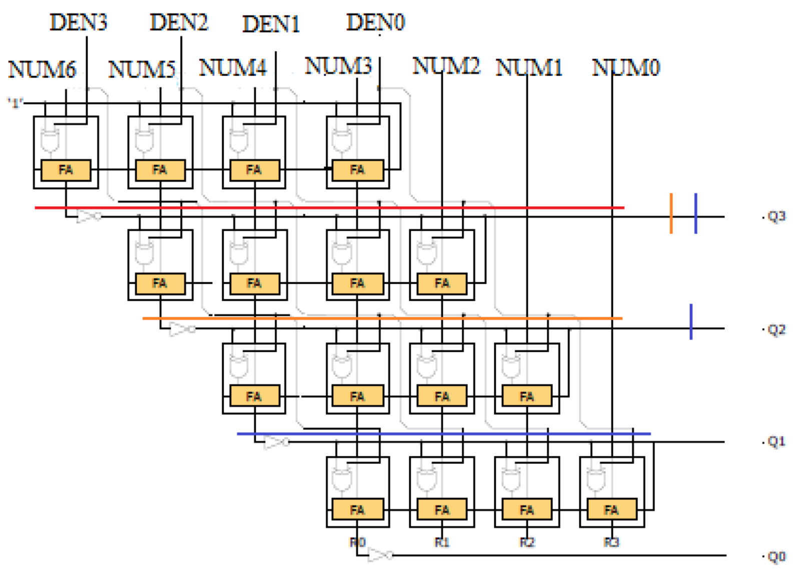


Figura 5.2. Pipelined divisor

In the table 5.2 the results of the synthesys of the divisor and of warping are reported. In both cases the results are as expected. Then with the pipilined divisor the critical path results, in the some way, due to the divisor path, but it is similar to the minimum periods of other blocks. With pipelined divisor there is a notewor-

Tabella 5.2. Study of critical path for optimizated architecture

BLOCK	CRITICAL PATH [ns]	AREA[μm^2]
PIPELINED DIVIDER	2.79	12337.20
WARPING	3.5	84332.9

thy improvement of the critical path and then of the maximum frequency of the structure. In fact,being the critical path due to the divisor in the previous version it was equal to 28 ns and in this version 2.8 ns then the maximum frequency, very important in this work for the real-time requirements, should be about ten time higher than the previous one, from about 35 MHz to about 350 MHz. However, this considerable improvement is paid, for obvious reasons, in terms of area due to the introduction of a considerable number of registers. Considering only the obtained area of warping, it changes from 54699.4 mu^2 to 84332.9 mu^2.

Capitolo 6

Optimizated Architecture

Starting from the synthesis of the previous described architecture, in the hypothesis of using real memories during the synthesis of the entire architecture, the typology of necessary memories has been studied and some optimizations are done.

6.1 The Modified Structure

Considering figure 3.7 the necessary memory for the TILE MEMORY shoud be a four port memory and the parallelism of input is a TILE. Then thinking on a memory that is easier to find and faster, the TILE MEMORY is replaced with four sram memory of 256 byte with a input and output parallelism on 32 bits. Applying this replacement the following four fifos are not necessary. In fact the data written in the memory on the clock fall edge is immediately read on the next rising edge of the clock and continuously supplied as input to the warping, in this way there is no latency due to the fact that with this solution is written and read one data on 32 bits per clock cycle. The initial latency will remain, instead, that due to the pipeline levels in the warping block. In fact the data is read, as soon as the address of the tile to be read in the MAIN MEMORY is valid, as it was in the previous version, but the position obtained from the warping of which data will be ready only 10 clock cycles later respect to the previous time. The warping output is on 53 bit, which consists of the corresponding input data (color and depth)on 32 bits concatenated to the output frame position on 20 bits and validity of the data on lsb, computed by WARPING, is no longer saved in the DATA MEMORY and then passes into the FIFO_DATA which provided the tile to the REORDER MEMORY which, by reading the correct order in respect of occusion compatibility, provided a data output ready to be written to the cache as in previous versione. All these intermediate memories, needed for synchronization, should have been very large memories and very often with a parallelism on tile, then to avoid it, some some changes have been made. The output of each WARPING block on 53 bits is saved in a fifo, there are four fifo, one corresponding to each warping block. DATA MEMORY, FIFO DATA have been eliminated leading only to the introduction of a second REAORDER MEMORY necessary to ensure a given output every clock cycle. The writing of the FIFOs is enabled as soon as the warping produces the first data, then 10 clock cycles after the first reading of the TILE MEMORY. In the case of FIFO1, the reading is done after one clock cycle respect to writing and, consequently, the data is immediately written to the first REORDER MEMORY. After sixty-four data have been written in the REORDER MEMORY in the location with addresses coming from the control unit, the latter it will activate the reading of FIFO2 and it will select as output of the multiplexer the its input data coming from FIFO 2, every 64 clock cycles the control unit changes input following in loop the order FIFO1, FIFO2, FIFO3, FIFO4. When the sixty-fourth data from the FIFO4 has been written into the REORDER MEMORY, the writing of the current memory is interrupted because the memory is full and all the tile data can, therefore, be read. The control unit enables the reading of the REORDER MEMORY as soon as written, selecting the addresses coming from the FIFO RC in the address selection mutiplexer and simultaneously enabling the writing of the other REORDER MEMORY so as not to lose any clock cycle. The data is written alternately one tile in one and one tile in the other memory. When the writing in the other one is taking place, in the other one the reading will take place. Before the cache, there is a multiplexer that selects from which REORDER MEMORY the data, in input to the cache will come. The data read by the REORDER MEMORY will already be sorted with respect to occusion compatibility and therefore ready to be written to the cache. The amount of memory required is much lower and the data transferred will be at most on 53 bits. TILE MEMORIES are small 256-byte sram memories while the two REORDER MEMORIES have to contain the data of the entire tile. Then their dimension is just under 2kbytes (256x53 bits). The entire architecture is illustrated in figure 6.2.

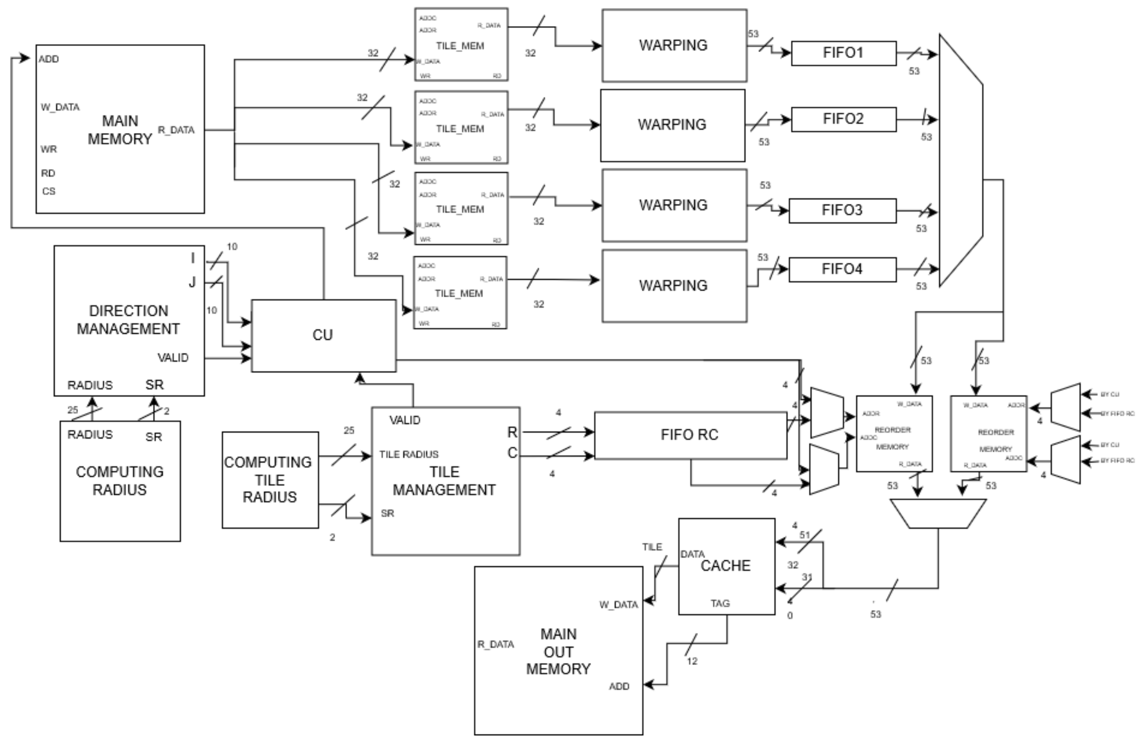


Figura 6.1. Optimized architecture block

In this optimizated structure the warping block is changed. In particular, as it is clear in figure 6.2, the pipeline registers are added to syncrhonized all the data that go out in parallel with result of the divider. In order to obtain the correct outputs of the warping, the data, read from TILE MEMORY in input of block, corresponding to the color and depth charateristics, are delayed by ten clock cycles. In the same way, all data necessary to validate the output are delayed by ten clock cycle to be synchronize with the output of divisor, and these data are :

- ovf that signals if there is overflow in the shift of numerators;
- MSB_DEN that is the most significant bit, then the sign bit, of the denominator;
- MSB_U that is the most significant bit, then the sign bit, of the numerator u;
- MSB_V that is the most significant bit, then the sign bit, of the numerator v;

In Warping block, it is insereted a new signal, END_LATENCY in figure, that signals when the latency due to the pipeline is finished and then the output could be valid.

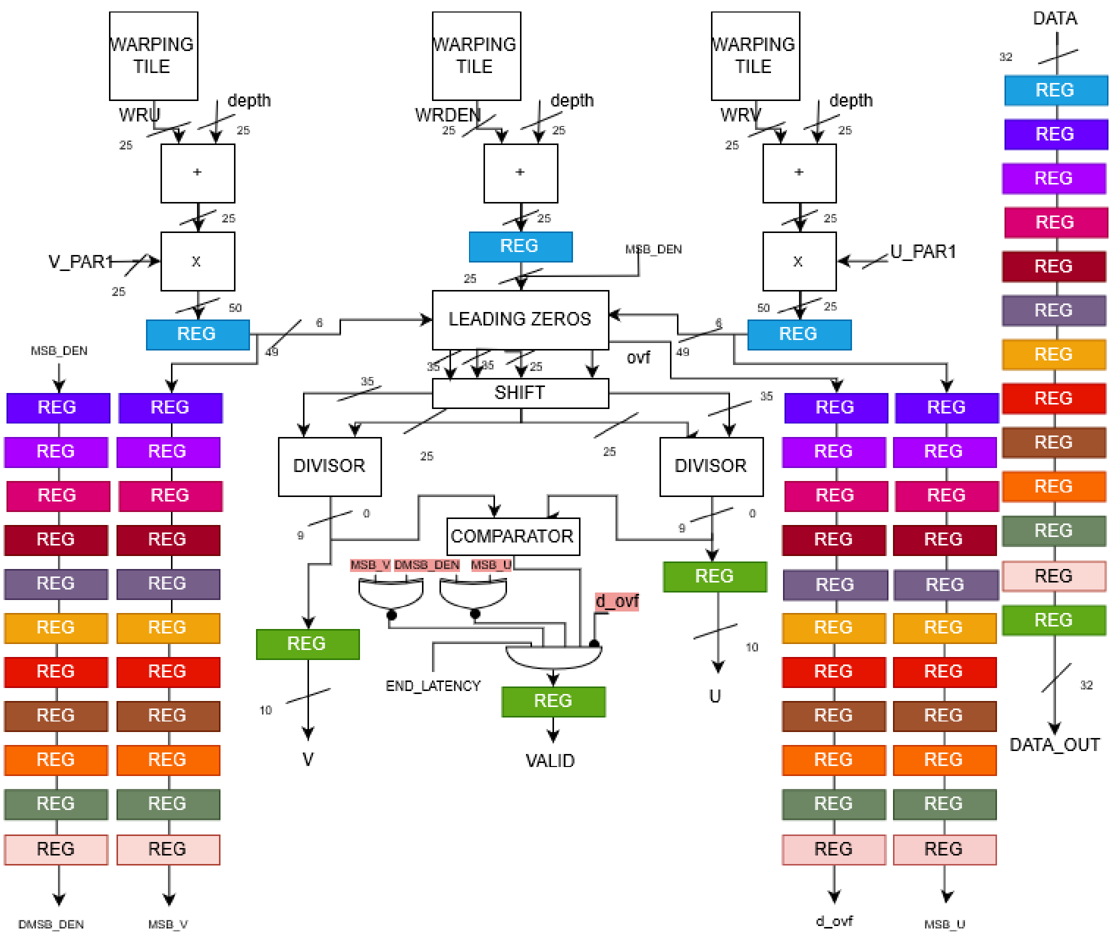


Figura 6.2. Pipelined warping block

The pipeline levels introduced have conditioned, also, some control signals in the control unit, which have also been delayed by 10 clock cycles. In the control unit new commands have been inserted for enabling the two REORDER MEMORIES and the enabling signals of all those unnecessary memories have been deleted. As for the main control unit, which also deals with memory management, for obvious reasons, it changes slightly. In particular, the signals that enables the reading and writing of the TILE MEMORIES. To be sure that the architecture loses the least possible clock cycle for the filling of these memories, so the reading of the TILE MEMORIES is always active during the warping, therefore from the state of TILE_EN_CU the reading signal is always set to one.

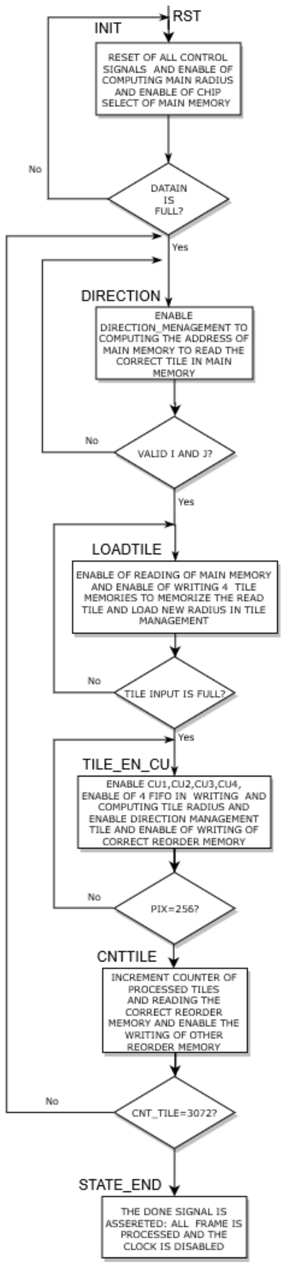

RST
INIT
RESET OF ALL CONTROL SIGNALS AND ENABLE OF COMPUTING MAIN RADIUS AND ENABLE OF CHIP SELECT OF MAIN MEMORY
No
DATAIN IS FULL?
Yes
DIRECTION
ENABLE DIRECTION_MENAGEMENT TO COMPUTING THE ADDRESS OF MAIN MEMORY TO READ THE CORRECT TILE IN MAIN MEMORY
No
VALID I AND J?
Yes
LOADTILE
ENABLE OF READING OF MAIN MEMORY AND ENABLE OF WRITING 4 TILE MEMORIES TO MEMORIZE THE READ TILE AND LOAD NEW RADIUS IN TILE MANAGEMENT
No
TILE INPUT IS FULL?
Yes
TILE_EN_CU
ENABLE CU1,CU2,CU3,CU4, ENABLE OF 4 FIFO IN WRITING AND COMPUTING TILE RADIUS AND ENABLE DIRECTION MANAGEMENT TILE AND ENABLE OF WRITING OF CORRECT REORDER MEMORY
No
PIX=256?
CNTTILE
INCREMENT COUNTER OF PROCESSED TILES AND READING THE CORRECT REORDER MEMORY AND ENABLE THE WRITING OF OTHER REORDER MEMORY
No
CNT_TILE=3072?
STATE_END
THE DONE SIGNAL IS ASSERETED: ALL FRAME IS PROCESSED AND THE CLOCK IS DISABLED

Figura 6.3. New main control unit

In TILE_EN_CU state, as shown in figure 6.3 the writing signals of the fifos and the correct REORDER MEMORY are also enabled, but these signals are then correctly delayed by twelve clock cycles in the upper hierarchical block, called Control unit. They are twelve because, since the reading of the first valid value from the TILE MEMORY, the warping employs ten clock cycles due to the divider pipeline and two clock cycle due to the WARPING pipeline as can be seen from previous section 3.5.3. Also the signals enabled in the CNTTILE state have changed, in fact, besides an updated the counter of the processed tiles, in this state, the reading of the correct REORDER MEMORY is also enabled and consequently the writing of the this MEMORY REORDER, that has to be read, is disabled and in the same way, instead, writing of the other MEMORY REORDER is enabled and read desabled.

6.2 Simulations

For the simulation of this part a different hierarchy of test_bench has been developed. In fact, in view of the simulation of the netlist the TOP test entity, called test_rend, was realized in verilog, while the other parts that manage the memory and the preparation of the input data are, instead, realized in vhdl. Specifically, there are two modules underlying the ts_rend that have been called test_structure and IN_OUT_STRUCTURE. The first deals with generating the clock signal with a period of 10 ns and an asynchronous reset and also takes care of providing all the parameters necessary to the structure and the filling of the main memories, that is not managed by the control unit, because it would be a very long process considering that the main memory is constituted by (768x1024x32) bits, therefore in the hypothesis of writing a whole word from 32 bits per clock cycle, there would have taken 787 clock cycles only to fill the main memory, which is considered external to the developed architecture. As far as the clock is concerned, it is in this module that the clock gating is carried out, since the input to the entity is provided by the control unit a done signal, which is then delayed by 10 clock cycles to respect the delay dictated by the levels of pipeline inserted in the structure. IN_OUT_STRUCTURE takes care of all memories management. It manages the distribution of data read from the MAIN MEMORY to the structure. In fact the main memory, as already explained in 3.5.2, is written and read in tile, while in the architecture it enters a data of 32 bits per clock cycle, dimension corresponding to a pixel. This data enters the TILE MEMORY and it is then placed at the input to the WARPING and the warping produces a data on 53 bits, containing the 32 bits of data of the pixel plus its address, given that, it is supplied outgoing to the architecture together with the reading signals and writing the fifos contained in an entity inferior to the IN_OUTPUT_STRUCTURE. The data read by the fifos are then passed to the architecture by the latter. It also deals with the writing of the ready and reordered data in the CACHE also contained in the IN_OUTPUT_STRUCTURE. In figure 6.4 it is shown the correct reordered data that are put in input to the CACHE. They are the same of the other structures, because as explain in the previous parts the correct order for position are: for u, ascending order, from 0 to 1023 and for each value of u the valid v values, in descending order, are from 767 to 0.

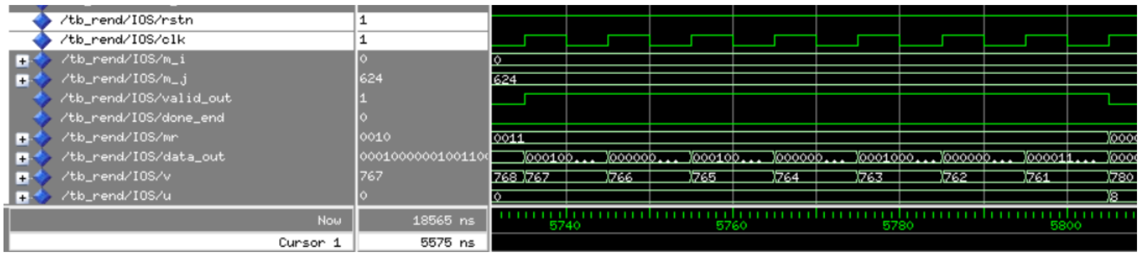


Figura 6.4. Correct data in output of the structure

In figure 6.5 a significant portion of the outputs of the control unit is shown. In particular, the focus is on the regular performance of the secondary control unit, in particular of the CU1, and of the main control unit. Also the addresses, put in input to the REORDER MEMORIES, are underlined . In fact, as already explained, the addresses of the REORDER MEMORIES are chosen through a multiplexer between the addresses, seen in the figure, generated by the control unit and the addresses generated by the TILE MENAGEMENT and saved in the FIFO RC. The addresses provided by the control unit are delayed by twelve clock cycles with respect to the first state in which the first data enters the WARPING. There are four different addresses in the figure (co_1, co_2, co_3, co_4) as compared to which fifo is supplying data to the REORDER MEMORY, the correct address is selected. Two new counters are inserted in the control unit to understand which REORDER MEMORY to select for writing and which to read. In particular,one, called "who", counts how many pixels have been processed, as shown in the figure 6.5, this counter is incremented together with the addresses provided to the REORDER MEMORIES. The figure shows, also, the initial functioning of the control signals of the two REORDER MEMORIES. It is noted, in fact, that for the first tile it is actived only the writing of the first REORDER MEMORY, in the figure called "wr_r_mem1", together with the chip select of both memories.

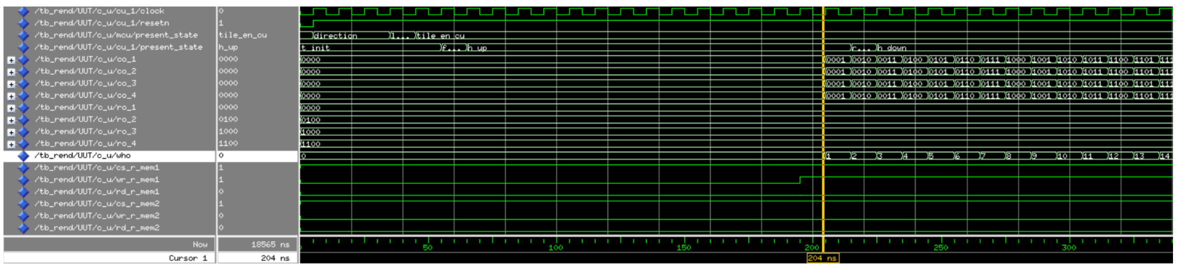
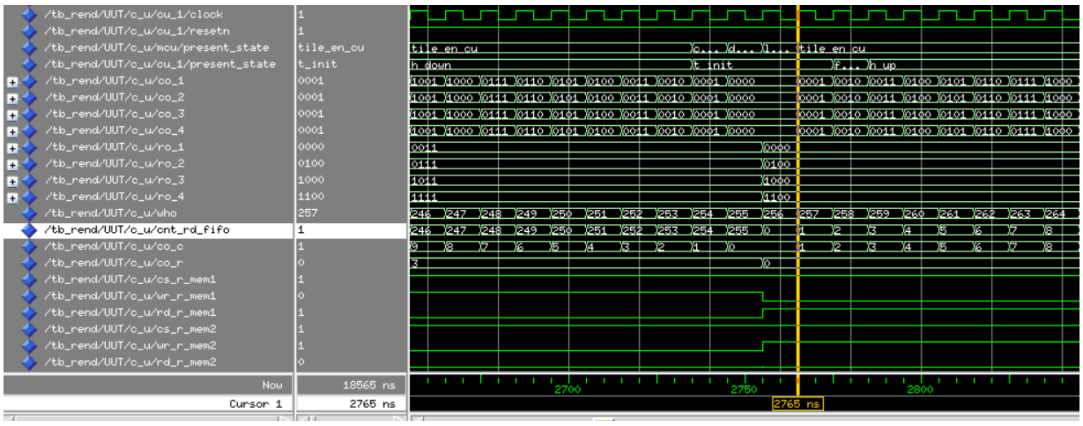

/tb_rend/UUT/c_u/cu_1/clock
/tb_rend/UUT/c_u/cu_1/resetn
/tb_rend/UUT/c_u/mcu/present_state
/tb_rend/UUT/c_u/cu_1/present_state
/tb_rend/UUT/c_u/co_1
/tb_rend/UUT/c_u/co_2
/tb_rend/UUT/c_u/co_3
/tb_rend/UUT/c_u/co_4
/tb_rend/UUT/c_u/ro_1
/tb_rend/UUT/c_u/ro_2
/tb_rend/UUT/c_u/ro_3
/tb_rend/UUT/c_u/ro_4
/tb_rend/UUT/c_u/who
/tb_rend/UUT/c_u/cs_r_mem1
/tb_rend/UUT/c_u/wr_r_mem1
/tb_rend/UUT/c_u/rd_r_mem1
/tb_rend/UUT/c_u/cs_r_mem2
/tb_rend/UUT/c_u/wr_r_mem2
/tb_rend/UUT/c_u/rd_r_mem2
Now
Cursor 1
204 ns

Figura 6.5. Working of the Control units and new signals

The second counter, called cnt_rd_fifo in figure 6.6, counts how many pixels have been processed in a single tile. In fact, as shown in the figure, it is reset when its value is equal to two hundred and fifty-sixth. While the first "who" counter continues to count up to five hundred and twelveth pixels, number corresponding to the pixels contained in two tiles. While the "cnt_rd_fifo" counts the pixels to manage which fifo must be read, "who" is necessary to decide which REORDER MEMORY to write or read. Figure 6.6 shows both the functioning of the alternate reading and writing of the two REORDER MEMORIES, when pixels of a tile are all processed, and the outputs of two counters used by the control unit to supply the reading address to the memories, called co_c and co_r.



```
/tb_rend/UUT/c_u/cu_1/clock              1
/tb_rend/UUT/c_u/cu_1/resetn             1
/tb_rend/UUT/c_u/mcu/present_state       tile_en_cu
/tb_rend/UUT/c_u/cu_1/present_state      t_init
/tb_rend/UUT/c_u/co_1                    0001
/tb_rend/UUT/c_u/co_2                    0001
/tb_rend/UUT/c_u/co_3                    0001
/tb_rend/UUT/c_u/co_4                    0001
/tb_rend/UUT/c_u/ro_1                    0000
/tb_rend/UUT/c_u/ro_2                    0100
/tb_rend/UUT/c_u/ro_3                    1000
/tb_rend/UUT/c_u/ro_4                    1100
/tb_rend/UUT/c_u/who                     257
/tb_rend/UUT/c_u/cnt_rd_fifo             1
/tb_rend/UUT/c_u/co_c                    1
/tb_rend/UUT/c_u/co_r                    0
/tb_rend/UUT/c_u/cs_r_mem1               1
/tb_rend/UUT/c_u/wr_r_mem1               0
/tb_rend/UUT/c_u/rd_r_mem1               1
/tb_rend/UUT/c_u/cs_r_mem2               1
/tb_rend/UUT/c_u/wr_r_mem2               1
/tb_rend/UUT/c_u/rd_r_mem2               0
                                   Now   18565 ns
                              Cursor 1   2765 ns
```

Figura 6.6. New counters to swap between two REORDER MEMORIES

The figure shows the operation of the TILE MEMORIES, in particular of the first, and the corresponding operation of the warping in relation to the input data provided by the TILE MEMORIES. We can well see how reading and writing of the TILE MEMORIES are synchronous and the control signals of them are contemporary. But as it can be seen from the data to be able to work and make sure that the reading data is the desired one, the writing is sensitive to the clock falling edge while the reading to the rising edge. The data read by the TILE MEMORIES are placed at the entrance to the WARPING block, indicated in the figure as "data". The WARPING success data corresponding to the data read by the TILE MEMORY will be available in output only after twelve clock shots due to the pipeline, as explained in previous time.

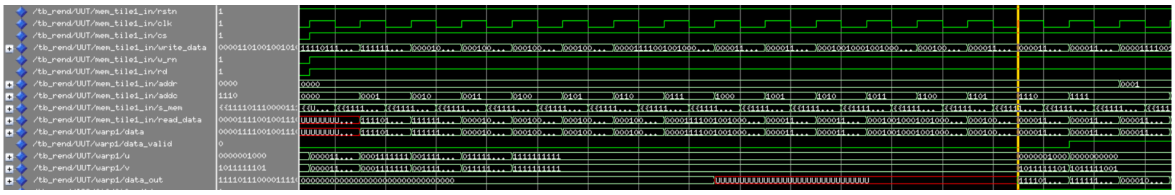

/tb_rend/UUT/mem_tile1_in/rstn 1
/tb_rend/UUT/mem_tile1_in/clk 1
/tb_rend/UUT/mem_tile1_in/cs 1
/tb_rend/UUT/mem_tile1_in/write_data 0000110100100101
/tb_rend/UUT/mem_tile1_in/w_rn 1
/tb_rend/UUT/mem_tile1_in/rd 1
/tb_rend/UUT/mem_tile1_in/addr 0000
/tb_rend/UUT/mem_tile1_in/addc 1110
/tb_rend/UUT/mem_tile1_in/s_mem {{1111011100001111
/tb_rend/UUT/mem_tile1_in/read_data 0000111100100111
/tb_rend/UUT/warp1/data 0000111100100111
/tb_rend/UUT/warp1/data_valid 0
/tb_rend/UUT/warp1/u 0000001000
/tb_rend/UUT/warp1/v 1011111101
/tb_rend/UUT/warp1/data_out 1111011100001111

Figura 6.7. Writing and Reading of the first input tile memory and input data of the Warping

In figure 6.8 is shown the hebaviour of the WARPING block. In particular, it is introduced a new signal, called "end_lat", that is equal to one when the latency due to pipeline is finished and the output data could be valid. The signal of "data_valid" is obtained from the logic AND between:

- the XNOR logic between nu and nd, in figure, corresponding to sign of u numerator and sign of denominator to detect that the sign of two terms of division are the same;
- the XNOR logic between nv and nd, in figure, corresponding to sign of v numerator and sign of denominator to detect that the sign of two terms of division are the same;
- end_lat, in figure, to detect when the latency is finished;
- oku, in figure, which is equal to one when the coordinate u in output of division is cantained in the correct u range;
- okv, in figure, which is equal to one when the coordinate v in output of division is cantained in the correct v range;
- the NOT logic of dovf, in figure, that is equal to one when at least one overflow in the inputs of the division is detect;

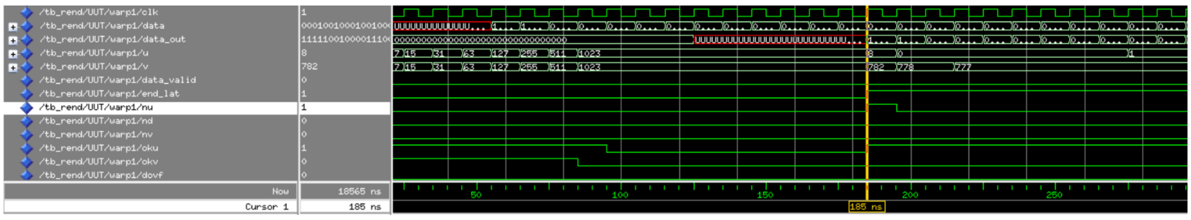


Figura 6.8. Genaration of signal validation of output data in WARPING

The figure 6.9 shows the passage of data between the first input TILE MEMORY, the WARPING block, the FIFO1, the multiplexer of selection of the input data of the first REORDER MEMORY and then REORDER MEMORY. In the upper part of the simulation in the figure, we see how, in order not to lose any data and be able to easily reconstruct the positions of the data in the REORDER MEMORY, the data exiting from the warping are however written in the fifo whether they are valid or not. The information on the validity of the data is contained in the last bit of the 53 saved in the fifo. The FIFO1, as it is underlined in the figure, starts to be read a clock cycle after the activation of writing, because the writing of the fifo is synchronous. What is shown in the figure with mux is the multiplexer that is immediately after the four fifos and deals with producing the right data coming from one of the four fifos. In this case the beginning of the writing of the first tile is being analyzed and consequently the multiplexer will select in output, thanks to the selection signal provided by the control unit, correctly the first input (ing1) connected to the FIFO1. The output of mux, out_1, corresponds perfectly to the input of the REORDER MEMORY, mem_buf1, from the figure, it is also clear that the writing is synchronous, in fact, smem is updated to the new value in input only after a clock cycle.The output signal is still undefined because the read signal has not yet been enabled, in fact it will be actived only when the content of the whole tile is written.

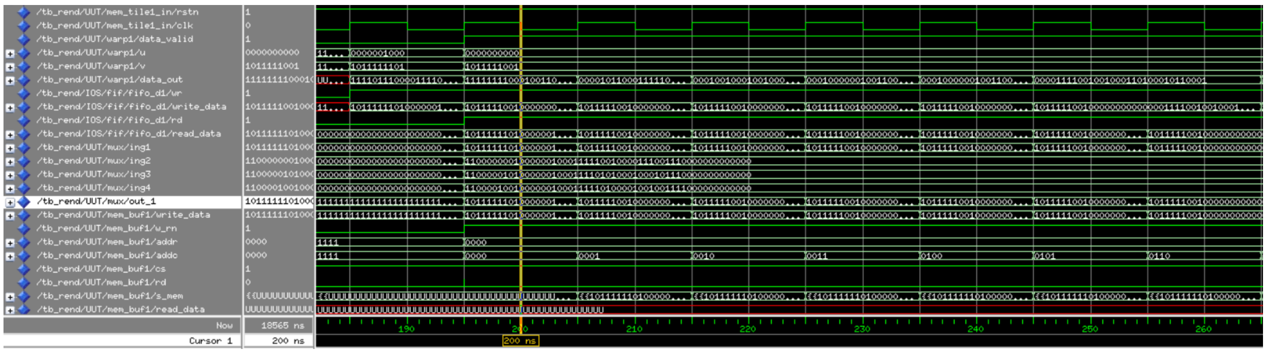

/tb_rend/UUT/mem_tile1_in/rstn
/tb_rend/UUT/mem_tile1_in/clk
/tb_rend/UUT/warp1/data_valid
/tb_rend/UUT/warp1/u
/tb_rend/UUT/warp1/v
/tb_rend/UUT/warp1/data_out
/tb_rend/IOS/fif/fifo_d1/wr
/tb_rend/IOS/fif/fifo_d1/write_data
/tb_rend/IOS/fif/fifo_d1/rd
/tb_rend/IOS/fif/fifo_d1/read_data
/tb_rend/UUT/mux/ing1
/tb_rend/UUT/mux/ing2
/tb_rend/UUT/mux/ing3
/tb_rend/UUT/mux/ing4
/tb_rend/UUT/mux/out_1
/tb_rend/UUT/mem_buf1/write_data
/tb_rend/UUT/mem_buf1/w_rn
/tb_rend/UUT/mem_buf1/addr
/tb_rend/UUT/mem_buf1/addc
/tb_rend/UUT/mem_buf1/cs
/tb_rend/UUT/mem_buf1/rd
/tb_rend/UUT/mem_buf1/s_mem
/tb_rend/UUT/mem_buf1/read_data
Now
18565 ns
Cursor 1
200 ns

Figura 6.9. Passing DATA between WARPING1, FIFO1 and REORDER MEMORY

The figure 6.11 shows the correctness of the read and write signals of the fifos comparing the first two fifos, FIFO1 and FIFO2. Specifically starting from the data provided by the WARPING1 block, they are written in the right fifo, FIFO1, while the data found in input to FIFO2 are different respect to ones because they are those produced by WARPING2. Furthermore the two entries are activated at the same time because, once the data produced by the WARPING blocks are ready, they must be written in their respective fifos so not to lose any of them. The fifo reading, on the other hand, is rightly different in the various fifos. In fact, while in the first fifo, FIFO1, it is immediately activated in order not to lose further clock cycles, and immediately write the data in the correct REORDER MEMORY, in the case of reading it is read first for 64 clock cycles, corresponding to the number of pixels of a tile processed by a WARPING, FIFO1 and for the 64 following FIFO2, then FIFO3 and then FIFO4.

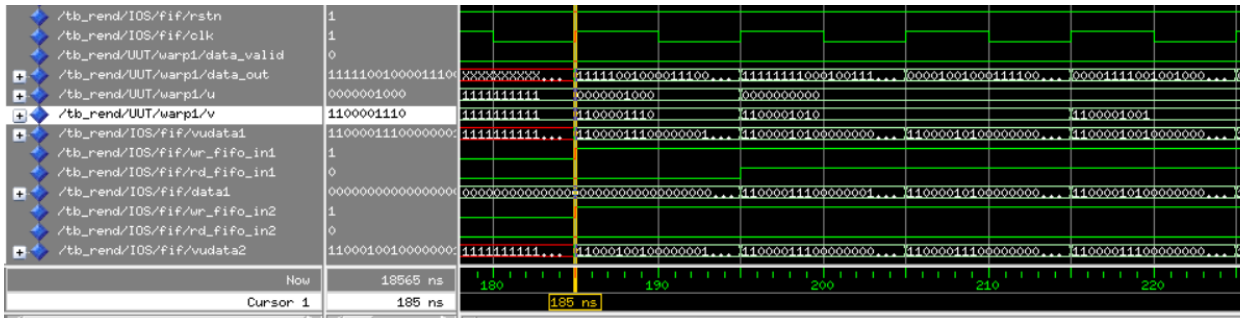

/tb_rend/IOS/fif/rstn 1
/tb_rend/IOS/fif/clk 1
/tb_rend/UUT/warp1/data_valid 0
/tb_rend/UUT/warp1/data_out
/tb_rend/UUT/warp1/u 0000001000
/tb_rend/UUT/warp1/v 1100001110
/tb_rend/IOS/fif/vudata1
/tb_rend/IOS/fif/wr_fifo_in1 1
/tb_rend/IOS/fif/rd_fifo_in1 0
/tb_rend/IOS/fif/data1
/tb_rend/IOS/fif/wr_fifo_in2 1
/tb_rend/IOS/fif/rd_fifo_in2 0
/tb_rend/IOS/fif/vudata2
Now 18565 ns
Cursor 1 185 ns
180
190
200
210
220
185 ns

Figura 6.10. Correct writing and reading in FIFO1 and in FIFO2

In figure 6.10 the above is just shown, in particular we note that the fifo writing signal is high for 64 clock cycles and low only for 4 considering a whole period of 68 clock cycles, the four clock cycles are due to the main control unit and needed to load the new tile. Considering this the reading signal respects what has just been said as for the first 64 clock cycles the reading of the FIFO1 is active, then of the FIFO2, then of the FIFO3, then of the FIFO4 and then starting from the FIFO1.

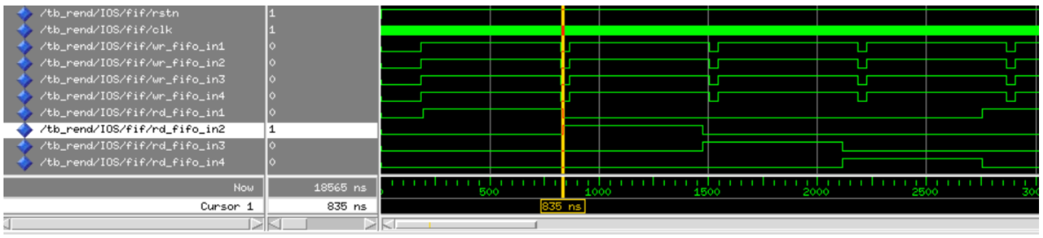

/tb_rend/IOS/fif/rstn 1
/tb_rend/IOS/fif/clk 1
/tb_rend/IOS/fif/wr_fifo_in1 0
/tb_rend/IOS/fif/wr_fifo_in2 0
/tb_rend/IOS/fif/wr_fifo_in3 0
/tb_rend/IOS/fif/wr_fifo_in4 0
/tb_rend/IOS/fif/rd_fifo_in1 0
/tb_rend/IOS/fif/rd_fifo_in2 1
/tb_rend/IOS/fif/rd_fifo_in3 0
/tb_rend/IOS/fif/rd_fifo_in4 0
Now 18565 ns
Cursor 1 835 ns
500
1000
1500
2000
2500
835 ns

Figura 6.11. Writing and alternate reading in FIFO1, FIFO2, FIFO3, FIFO4

In figure 6.12 it is shown how the same identical behavior is respected also by the selection of "mux" and therefore by the correct writing of the REORDER MEMORY in a general vision.

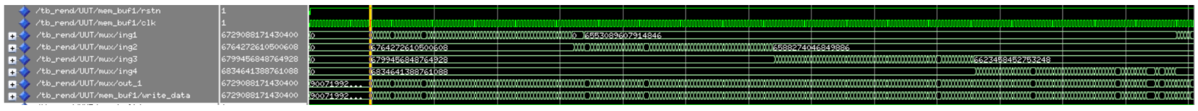
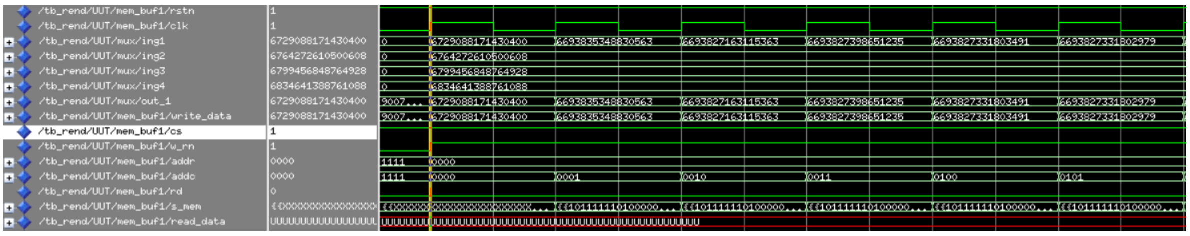

/tb_rend/UUT/mem_buf1/rstn 1
/tb_rend/UUT/mem_buf1/clk 1
/tb_rend/UUT/mux/ing1 6729088171430400
/tb_rend/UUT/mux/ing2 6764272610500608
/tb_rend/UUT/mux/ing3 6799456848764928
/tb_rend/UUT/mux/ing4 6834641388761088
/tb_rend/UUT/mux/out_1 6729088171430400
/tb_rend/UUT/mem_buf1/write_data 6729088171430400

Figura 6.12. Working of multiplexer for selection of FIFO

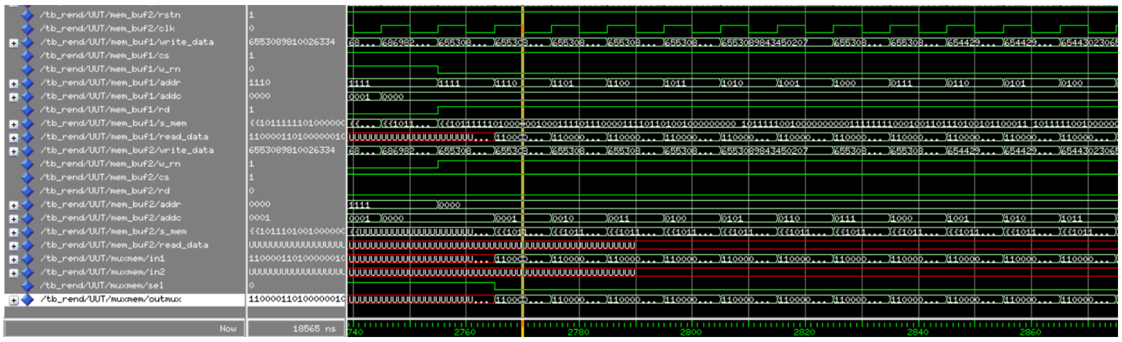
The figure 6.13 shows the correctness from the data point of view. In fact, for convenience and clarity the data are reported in decimal to better appreciate that the output data from "mux" are those of the correct FIFO, FIFO1, and the same are correctly written with the correct addresses in the first REORDER MEMORY.



```
/tb_rend/UUT/mem_buf1/rstn         1
/tb_rend/UUT/mem_buf1/clk          1
/tb_rend/UUT/mux/ing1              6729088171430400
/tb_rend/UUT/mux/ing2              6764272610500608
/tb_rend/UUT/mux/ing3              6799456848764928
/tb_rend/UUT/mux/ing4              6834641388761088
/tb_rend/UUT/mux/out_1             6729088171430400
/tb_rend/UUT/mem_buf1/write_data   6729088171430400
/tb_rend/UUT/mem_buf1/cs           1
/tb_rend/UUT/mem_buf1/w_rn         1
/tb_rend/UUT/mem_buf1/addr         0000
/tb_rend/UUT/mem_buf1/addc         0000
/tb_rend/UUT/mem_buf1/rd           0
/tb_rend/UUT/mem_buf1/s_mem        {{XXXXXXXXXXXXXXXX
/tb_rend/UUT/mem_buf1/read_data    UUUUUUUUUUUUUUUUUUUL
```

Figura 6.13. Passing data between the multiplexer after FIFOs and REORDER MEMORIES

The figure 6.14 shows the operation of the final part of the architecture composed of the two REORDER MEMORIES and the output multiplexer, called "muxmem". "muxmem" selects either the first input if the REORDER MEMORY being read is the first or second input in the case where the memory being read is the second REORDER MEMORY, since the first input, in1, is connected to the output of the first REORDER MEMORY and the second input, in2, to the output of the second one.



```
/tb_rend/UUT/mem_buf2/rstn         1
/tb_rend/UUT/mem_buf2/clk          0
/tb_rend/UUT/mem_buf1/write_data   6553089810026334
/tb_rend/UUT/mem_buf1/cs           1
/tb_rend/UUT/mem_buf1/w_rn         0
/tb_rend/UUT/mem_buf1/addr         1110
/tb_rend/UUT/mem_buf1/addc         0000
/tb_rend/UUT/mem_buf1/rd           1
/tb_rend/UUT/mem_buf1/s_mem        {{101111110100000C
/tb_rend/UUT/mem_buf1/read_data    110000110100000010
/tb_rend/UUT/mem_buf2/write_data   6553089810026334
/tb_rend/UUT/mem_buf2/w_rn         1
/tb_rend/UUT/mem_buf2/cs           1
/tb_rend/UUT/mem_buf2/rd           0
/tb_rend/UUT/mem_buf2/addr         0000
/tb_rend/UUT/mem_buf2/addc         0001
/tb_rend/UUT/mem_buf2/s_mem        {{101110100100000C
/tb_rend/UUT/mem_buf2/read_data    UUUUUUUUUUUUUUUUUUUL
/tb_rend/UUT/muxmem/in1            110000110100000010
/tb_rend/UUT/muxmem/in2            UUUUUUUUUUUUUUUUUUUL
/tb_rend/UUT/muxmem/sel            0
/tb_rend/UUT/muxmem/outmux         110000110100000010

                              Now  18565 ns
```

Figura 6.14. Writing and Reading of two REORDER MEMORIES and output data

The same osservation is done in 6.15, but in this case the simulation is at a later time, in which both REORDER MEMORIES are working and it is shown the end of reading the first and the beginning of reading the second one. Just as it is shown the end of the writing of the second and the beginning of the writing of the first. Particular attention is paid to the output of the "muxmem", which continues to have valid data passing from the last data read from the first memory to the first data read from the second memory without any interruption.

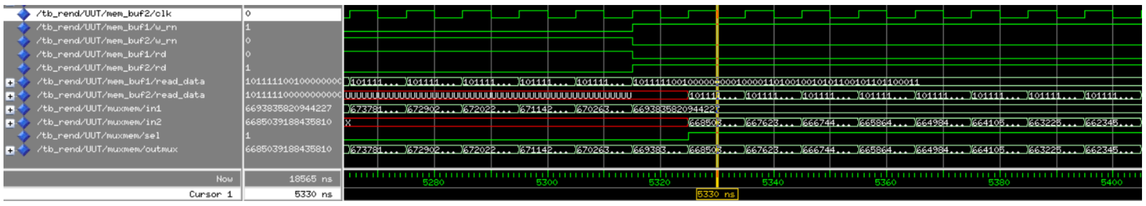

/tb_rend/UUT/mem_buf2/clk
/tb_rend/UUT/mem_buf1/w_rn
/tb_rend/UUT/mem_buf2/w_rn
/tb_rend/UUT/mem_buf1/rd
/tb_rend/UUT/mem_buf2/rd
/tb_rend/UUT/mem_buf1/read_data
/tb_rend/UUT/mem_buf2/read_data
/tb_rend/UUT/muxmem/in1
/tb_rend/UUT/muxmem/in2
/tb_rend/UUT/muxmem/sel
/tb_rend/UUT/muxmem/outmux
Now
18565 ns
Cursor 1
5330 ns

Figura 6.15. Swap of two REORDER MEMORIES

The data output from the "muxmem" are the data now reordered in compatibility with the occlusion and therefore ready to be written in the cache and then filled a tile in the MAIN OUT MEMORY. In the figure you can see how the data on 53 bits coming from the multiplexer are divided according to the meaning. Thus from the most significant bit the first 10 bits correspond to the coordinate v, the next 10 to the coordinate u the next 32 to the data relating to color and depth of the pixel and the last bit the lsb, to the validity of the data. In fact only if the data is valid, that is lsb equal to ’1’, it is written in cache and therefore "wr" which would be the cache writing signal becomes high. The data to be written are the 32 bits, called data_out, while the two coordinates u and v are used for determining the tag and the offset.

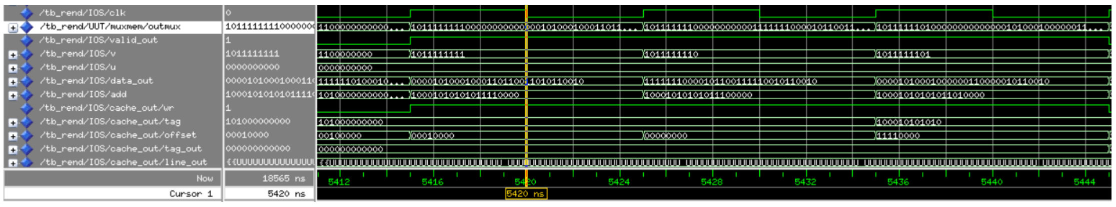

/tb_rend/IOS/clk
/tb_rend/UUT/muxmem/outmux
/tb_rend/IOS/valid_out
/tb_rend/IOS/v
/tb_rend/IOS/u
/tb_rend/IOS/data_out
/tb_rend/IOS/add
/tb_rend/IOS/cache_out/wr
/tb_rend/IOS/cache_out/tag
/tb_rend/IOS/cache_out/offset
/tb_rend/IOS/cache_out/tag_out
/tb_rend/IOS/cache_out/line_out
Now
18565 ns
Cursor 1
5420 ns

Figura 6.16. Reaordered data in input to the CACHE and working of cache

The tag realized, as explained in the relative section 3.8, affixed with the z-order addressing technique is obtained from the six most significant bits of v interlaced to the six most significant bits of u, while the four remaining bits of v of u are used for the formation of the offset. Instead with regards to the tag_out and the line_out they correspond to the data written in WRITE BACK in the MAIN OUT MEMORY, in particular the tagout is all zeros since the cache is initially considered to be full and that is the tag inserted by default to the first line and the line_out corresponds to a tile full of undefined.

6.3 Synthesis

In order to synthesize the entire architecture and generate the relative netlist it was decided to consider, only in testbench and therefore not to hear, the FIFOs and the cache together with the two main input and output memories MAIN MEMORY and MAIN OUT MEMORY, as they are too large to synthesize with synopsys. At first a clock period equal to 0 ns was applied, in order to understand which was the negative slack and therefore the least applicable clock period. A minimum period of 4.36 ns was thus obtained with an area equal to 2993749.6 μm^2 and a power eximation equal to 3.4 mW as reported in relative table 6.1. With what

Tabella 6.1. Area and Power for the minimum clock period

Tmin [ns]	AREA[μm^2]	POWER [mW]
4.36	2993749.59	3.4

has been obtained it was therefore decided to set a clock period equal to 5 ns, consequently the architecture can work at a maximum frequency of 200 MHz. The

Tabella 6.2. Area

CLOCK PERIOD [ns]	AREA[μm^2]
5	2935857

area occupated by the obtained netlist with this clock constraint is equal to 2935857 μm^2 as reported in relative table 6.2.

The estimate of power consumption obtained from synopsys from the generation of the netlist is equal to 371.47 mW obtained by 364.96 mW of internal power, switching power equal to 6.3 mW and 0.23 mW of leakege power. To reduce the dynamic power, in fact, as explained in the relative sections, numerous measures have been implemented to save it. As in the INIT state of the secondary control unit that most of the signals have already been pre-set as in the next state, so as to decrease the switching activity due to them. Despite this, the architecture turns out to be very large and full of blocks to control, so much so that the control unit generates lots of signals and consequently the dynamic power can only be affected. The components that require the highest power consumtion are the registers, because the small memories necessary for the structure are synthetized with the flip-flop, then the quantity of memory elements in the netlist is very high, while the consumption of the combinatorial part is equal to 10 mW only 3 % of total power.

Tabella 6.3. POWER EXIMATION for optimizated architecture

INTERNAL [mW]	SWITCHING [mW]	LEAKAGE [pW]	TOTAL [mW]
364.96	6.3	2.29 e+8	371.47

6.4 Results

The results stored in the cache are written in files from teshbench. These files are read from the script MATLAB to plot with video graphic tecniques of Matlab the obtained numeric values. The resulted image is ones of 6.17. Respect to 2.5, obtained in MATLAB with the values computing from developed MATLAB model, the result image of the architecture is less precise and with many more black pixels. These image can be improved with inpaint and filtering.

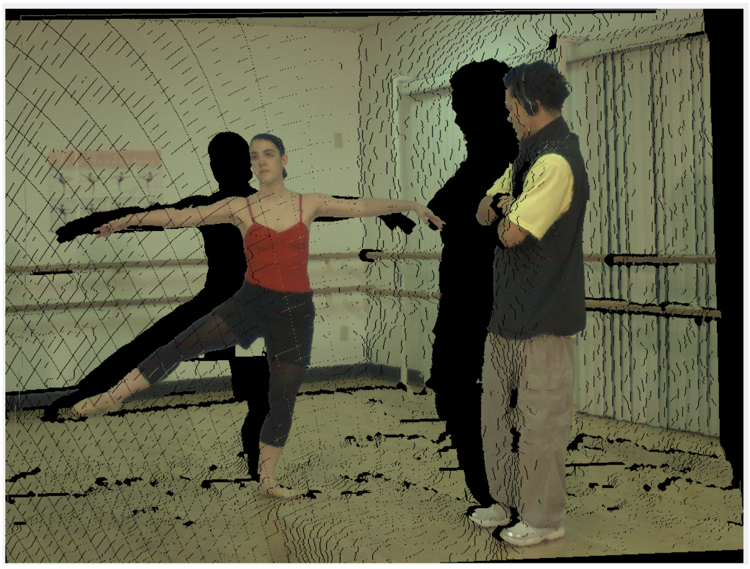


Figura 6.17. Results in output of structure

The results are not always equal to ones of MATLAB, bacause as explained in different points of this work, the type of number considered are the fixed point numbers, then the obtained positions of each input pixel in the output frame could be a little different respect to ones obtained by the MATLAB model in double precision.

Capitolo 7

Conclusions and possible future improvements

In future implementation it could be realized a structure in floating point even if this requires a greater complexity at the architectural level. In order to improve the error due to the truncation with which the results of the WARPING is obtained, it could be realized a block that manages the rounding which also takes into account the bits relating to the fractional part. As far as this truncation is concerned, the error due to it is not very high, since at most the error introduced could be equal to 1 lsb, since in truncation the only bits eliminated are fractional ones, therefore using a rounding this could be taken into account and if the most significant bit of the fractional part is equal to one, the latter could be added to the result. A further improvement could be to insert real memories at the synthesis level, since the internal ones are mostly quite small since the memories that make up the four TILE MEMORIES could be SRAM of 256 Bytes each, while the memories needed for the REORDER MEMORIES could be two SRAMs for less than 2kBytes, then in the synthesys level these internal memories were synthesized as composed by a set of flip-flops. The final quality depends on the availability of color information, in particular in occluded zones. The very important role for a high quality of final image is, also, the post-processing part after the fusion of the two images obtained by the architecture developed in this work, one referred to the left view and one reffered to the right view. The fused image in the post-processing could be filtered and the impaint is necessary to cover the black pixels. A first step is the filtering. One possible filter, as proposed in 1 is a modified Hybrid Median Filter. It performs the median filter on the five elements in an horizontal cross shape, then on a diagonal cross shape and, finally, looks for the last median value between these two values and the central pixel. The idea here is to take advantage of all the available information to find the output value. Another important step is the inpaint process, it can be handled directly inside this process. The key point of this process is that the previous results are directly used for the inpaint task. The missed part of information is coming from the pixel that will be processed in the next time. In this case there could be black pixels to discard. In particular,if both the views presents a black pixel in that location, leave a black value, instead if one of the two pixels is black and the other is valid take the valid one as the pixel value, or if both the pixels present valid values take the average of the two. Now that the number of black pixels should have been reduced to the minimum, it is possible to apply the HMF filter but a crucial step is to not consider the remaining black pixels to compute the medians. In particular the color values have to be sorted from the smallest to the greatest one, the blank pixels should not be considered because they will simply be the in the lowest value positions and the median will be taken from the remaining ones, leaving a black value if all the pixels are black. Considering that this approach work with two consecutive pixel during the warping, it is very important considering that the proposed architecture of this work is parallelized.

Bibliografia

[1] R. Peloso, *"Design and optimization of a VLSI architecture for Depth Image-Based Rendering"*, Politecnico di Torino, 2016.

[2] Microsoft Research. MSR 3D Video Dataset.

[3] Wang, Baokang ; Fukazawa, Yuki ; Kondo, Toshio ; Sasaki, Takahiro, *" Tile/line access cache memory based on a multi-level Z-order tiling data layout"*, Practice and Experience, 10 May 2018, Vol.30(9).

[4] Wikipedia, *" Z-order curve"*, Wikipedia, the free encyclopedia

[5] Y. R. Horng, Y. C. Tseng, and T. S. Chang *"VLSI Architecture for Real- Time HD1080p View Synthesis Engine"*, IEEE Transactions on Circuits and Systems for Video Technology 21.9 (Sept. 2011).

[6] *"Appendix B, Mathematical Computations on Fixed-Point Processors"*,Digital Media Processing. Ed. by Hazarathaiah Malepati. Boston: Newnes, 2010.

[7] Krishna Rao Vijayanagar et al. *"Efficient view synthesis for multi-view video plus depth"*,IEEE. 2013.

[8] Y. R. Horng, Y. C. Tseng, and T. S. Chang. *"VLSI Architecture for Real- Time HD1080p View Synthesis Engine»"*EEE Transactions on Circuits and Systems for Video Technology 21.9, 2011.

[9] F. J. Chang, Y. C. Tseng, and T. S. Chang. *"A 94fps view synthesis engine for HD1080p video». Visual Communications and Image Processing (VCIP)"*IEEE, 2011

[10] J. Wang and L. A. Rønningen. *"Real time believable stereo and virtual view synthesis engine for autostereoscopic display"*International Conference on 3D Imaging (IC3D),2012.

[11] M. Schaffner et al. *"MADmax: A 1080p stereo-to-multiview rendering ASIC in 65 nm CMOS based on image domain warping"* ESSCIRC (ESSCIRC),2013.

[12] Wikipedia *"Geometria_epipolare"*, Wikipedia, the free encyclopedia .

[13] Richard Hartley, Andrew Zisserman, Multiple View Geometry *"Computer Vision"*Cambridge University Press, 2004.

[14] P. K. Tsung et al. *"A 216fps 4096x2160p 3DTV set-top box SoC for freeview-point 3DTV applications"*IEEE International Solid-State Circuits Conference, 2011.

[15] Samira Sayedsalehia, Mostafa Rahimi Azghadib, Shaahin Angizic, Keivan Navic " *Restoring and non-restoring array divider designs in Quantum-dot Cellular Automata* "Information Sciences Volume 311, 1 August 2015, P. 86-10