International Journal of Reconfigurable and Embedded Systems (IJRES)
Vol. 1, No. 3, November 2012, pp. 108~122
ISSN: 2089-4864

Design and Development of Texture Filtering Architecture for GPU Application Using Reconfigurable Computing

Krishna Bhushan Vutukuru*, Sanket Dessai **
* Department of Computer Engineering, M. S. Ramaiah School of Advanced Studies, Bengaluru
** Department of Computer Engineering, M. S. Ramaiah School of Advanced Studies, Bengaluru

Article Info

Article history:

Received Jun 17, 2012
Revised Oct 11, 2012
Accepted Oct 28, 2012

Keyword:

Texture Filtering
Texel
GPU
Pixel
Reconfigurable

ABSTRACT

Graphical Processing Units (GPUs) have become an integral part of today's mainstream computing systems. They are also being used as reprogrammable General Purpose GPUs (GP-GPUs) to perform complex scientific computations. Reconfigurability is an attractive approach to embedded systems allowing hardware level modification. Hence, there is a high demand for GPU designs based on reconfigurable hardware. The texture filter unit is designed to process geometric data like vertices and convert these into pixels on the screen. This process involves number of operations, like circle and cube generation, rotator, and scaling. The texture filter unit is designed with all necessary hardware to deal with all the different filtering operations. The designed texture filtering units are modelled in Verilog on Altera Quartus II and simulated using ModelSim tools. The functionality of the modelled blocks is verified using test inputs in the simulator.Circle and cube coordinates are generated for circle and cube generation. The work can form the basis for designing a complete reconfigurable GPU.

Corresponding Author:

Sanket Dessai,
Department of Computer Engineering,
M.S.Ramaiah School of Advanced Studies,
#470-P,Peenya Industrial Area,Peenya 4th Phase,Bengaluru-560058
Karnataka, India.
Email: sanketdessai@gmail.com

1. INTRODUCTION

Computer graphics has become an important technique in many applications such as CAD tools, game, film, virtual reality and etc. Although many techniques are used in 3D Computer graphics, texture mapping is one of the most successful and popular techniques in high-quality image synthesis [5]. Especially, texture mapping creates the appearance of complexity without the tedium of modelling and rendering every 3D detail of a surface. Moreover, texture mapping is a basis of other mapping techniques such as shadow mapping, environment mapping, bump mapping and etc. However, the greatest weakness of the texture mapping is that it requires high memory bandwidth to fetch the texture image data. The use of a cache is important in improving processing speed of a system. A well-tuned cache hierarchy and organization can induce the increase of system performance and bandwidth saving in a system bus.During the texture mapping process, a 'texture lookup' takes place to find out where on the texture each pixel centre falls. Since the textured surface may be at an arbitrary distance and orientation relative to the viewer, one pixel does not usually correspond directly to one texel [25]. Some form of filtering has to be applied to determine the best color for the pixel. Insufficient or incorrect filtering will show up in the image as artefacts (errors in the image), such as 'blockings', jaggies, or shimmering.

Texture unit is one of the major components in GPU, and consists of three main parts-address generator(s), texture cache(s), and texture filter(s). A reconfigurable architecture is chosen due to the following two reasons: (1) Area reduction is better achieved using all-purpose texture filters because they are not idle as the required filter type varies during run-time; (2) the reconfiguration overhead is low because the configuration does not switch very often.

2. TEXTURE FILTERING CONCEPTS

In 3D computer graphics, surfaces of a 3D object are represented into sum of triangles. Drawing of a 2D image onto the surface is texture mapping. The image drawn onto the surface is called a texture map and its individual element, texture pixel, is called a texel. The texture mapping consists of two steps: the first is a transform from the 2D texture space to the 3D object space and the second is a transform from the 3D object space to the 2D screen space [17]. The composition of two transforms is denoted as a rational linear projective transform as shown in Equation (1). The *xs*, *ys* and *u*, *v* are coordinate values of a pixel in the screen space and a corresponding texel in the texture space. And, *a* and *i* are constants.

$$x_s = \frac{au+bv+c}{gu+hv+i} \qquad y_s = \frac{du+ev+f}{gu+hv+i} \tag{1}$$

All the triangles creating a 3D object are decomposed into spans in a span rasterizer for displaying. Span means a set of successive horizontal pixels in a triangle. By mapping, a span is mapped into a random-directed straight line on a texture image as shown in Figure 1. This line conservation property is well explained in the following. When an arbitrary line in the screen space, $y_s = Ax_s + B$, is mapped into texture image space, a corresponding line, $v = A`u + B`$, is obtained by substituting x_s and y_s in Equation (1) and rearrangement [17].

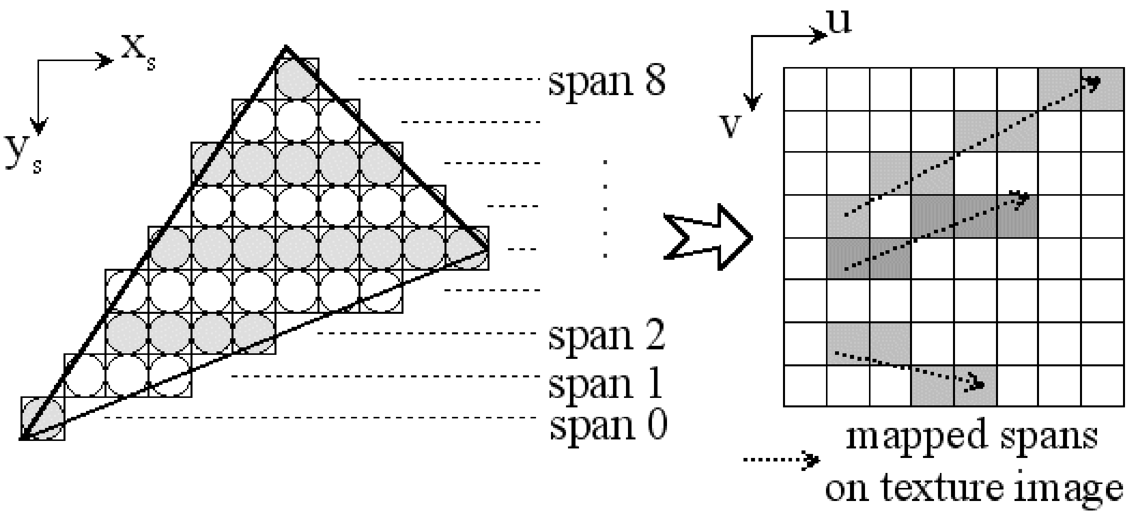


Figure 1. Mapping of Triangle Spans

2.1 Texture Cache And Triangle Span

The efficiency of cache memory depends on region of location in data accesses. Both spatial and temporal regions are present in texture mapping [13]. Mipmap filtering increases spatial locality in texture access since the level of the map is selected to closely match the level-of-detail that is being drawn on the screen. That is, due to the mipmap filtering, one pixel movement in screen space is nearly mapped to one texel movement in texture space. One texture image can be mapped to several polygons of single frame or consecutive frames. Therefore, temporal locality in texel access is also present. In the cases of bilinear or trilinear filtering, multiple texels are needed for single pixel. It also contributes to temporal locality because some of the multiple texels for a pixel are appropriate to overlap with some texels for neighbouring pixels. Due to the locality of texture, texture cache can be used to improve system performance and to save the required bandwidth in system bus.

A rendered scene can be decomposed into a set of spans. Cache replacement can be divided based on the triangle span. If cache replacement occurs between texels residing in the same span, then it is called as intra-span replacement. If replacement occurs between texels of two different spans, we call it inter-span replacement. In cache operation, there is one cache replacement when one cache miss occurs excluding cold miss. So, cache miss can be calculated by counting cache replacement occurring in the cache.

2.2 Geomatrical Methods

The differentiation of texture analysis techniques that falls under the title of geometrical processes is distinguished by their definition of texture as being composed of “texture elements” or primitives. The process of analysis generally depends upon the geometric properties of these texture elements. Once the texture elements are identified in the image, there are two major approaches to analyzing the texture. One computes statistical properties from the extracted texture elements and utilizes these as texture features. The other tries to extract the placement rule that describes the texture. The latter approach may involve geometric or syntactic methods of analyzing texture.

Each stream client may access its dedicated stream buffer every cycle if there is data available to be read or space available to be written. The eight stream buffers serving the clusters are accessed eight words at a time, one word per cluster. The eight stream buffers serving the network interface are accessed two words at a time [20]. The other six stream buffers are accessed a single word at a time. The peak bandwidth of the stream buffers is therefore 86 words per cycle, allowing peak stream demand to exceed the SRF bandwidth during short transients. Stream buffers are bidirectional, but may only be used in a single direction for the duration of each logical stream transfer.

2.3 VORONOI TESSELLATION FEATURES

Voronoi tessellation has been proposed because of its desirable properties in defining local spatial neighborhoods and because the local spatial distributions of tokens are reflected in the shapes of the Voronoi polygons. First, texture tokens are extracted and then the tessellation is constructed. Tokens can be as simple as points of high gradient in the image or complex structures such as line segments or closed boundaries.

In order to apply geometrical methods to gray level images, first extraction of tokens from images has to be performed. Following simple algorithms are used to extract tokens from input gray level textural images.

1. Apply a Laplacian-of-Gaussian (LoG or $\nabla 2G$) filter to the image. For computational efficiency, the $\nabla 2G$ filter can be approximated with a difference of Gaussians (DoG) filter. The size of the DoG filter is determined by the sizes of the two Gaussian filters.
2. Select those pixels that lie on a local intensity maximum in the filtered image. A pixel in the filtered image is said to be on a local maximum if its magnitude is larger than six or more of its eight nearest neighbors. This results in a binary image.
3. Perform a connected component analysis on the binary image using eight nearest neighbors. Each connected component defines a texture primitive (token).

The texture features based on Voronoi polygons have been used for segmentation of textured images. The segmentation algorithm is edge based, using a statistical comparison of the neighboring collections of tokens. A large dissimilarity among the texture features is evidence for a texture edge.

3. REQUIREMENT ANALYSIS

The requirement analysis for the texture filtering architecture modelling includes on FPGA devices and its components. A texture filtering unit needs to be verified, the output can be monitored on a terminal for the multiprocessing tasks. In order to debug the processor and monitor the processor state a ‘JTAG Debugger’ must be present in the system. Since the texture filtering unit model consumes huge number of slices, the FPGA device of advanced version is required. Hence to bring about the mentioned requirements for professional model of texture filtering unit architecture the available FPGA Altera Cyclone II has been decided upon.

3.1 Voronoi Tessellation Features

The design consists of developing a programmable graphics processing unit with as many aspects as possible to be coded in hardware, even with object and edge generation. Texture mapping is a shading technique for image synthesis in which a texture image is mapped onto a surface in a three dimensional scene, much as wallpaper is applied to a wall. If a table is need to be modelled, a rectangular box for the table top, and four cylinders for the legs is used. Unadorned, this table model would look quite dull when rendered. The realism of the rendered image can be enhanced immensely by mapping a wood grain pattern onto the table top, using the values in the texture to define the colour at each point of the surface. The advantage of texture mapping is that it adds much detail to a scene while requiring only a modest increase in rendering time. Texture mapping does not affect hidden surface elimination, but merely adds a small incremental cost to the shading process. The technique generalizes easily to curved surfaces.

Texture mapping can be used to define many surface parameters besides color. These includes the perturbation of surface normal vectors to simulate bumpy surfaces, transparency mapping to modulate the opacity of a translucent surface, specularity mapping to vary the glossiness of a surface, and illumination mapping to model the distribution of incoming light in all directions.

In all of the varieties of texture mapping mentioned above, geometric mappings are fundamental. Two-dimensional mappings are used to define the parameterization of a surface and to describe the transformation between the texture coordinates system and the screen coordinates system.

The graphics unit takes operations in a very-long instruction word format that has a one-to-one representation to a high-level scripting language, which provides a mean to moving objects and features in a scene to dynamically during run-time. The high-level design shares many similarities multi-cycle pipelines, such as intermediate memories and registers. However, unlike a regular processor, the co-processor has one pipeline that operates on multiple pieces of data in parallel; much like a vector processor does in a single-instruction multiple-data fashion.

There are three components of the circuit: an object generation pipeline to generate edges of the target shape; a transformation pipeline that performed transformations on the unit objects1; and a rasterizing pipeline that generates the points for the VGA controller to display. The design has made certain tradeoffs due to the constraints imposed by the FPGA to synthesize the circuit.

First, the transformation pipeline does not employ a generalized 4x4 matrix multiply because the limited number of multipliers on the FPGA. Instead, the transformation pipeline is currently designed as an operate-and-accumulate module, with intermediate data values stored in registers. Alternatively with more available multipliers, by first generating a reduced matrix transformation, one data set can be transformed in one cycle. Second, the available memory on the FPGA is limited to 8.5 megabytes at most, of which about 512 kilobytes are available memory that are designed to be read from within a single cycle of exerting the desired address.

The graphics platform was designed to be able to generate shapes and objects by computing the edges for the object, perform a number of transformations, and to rasterize the scene into a VGA buffer. The module houses three concurrent pipelines: one to generate an edge list, one to compute the transformations, and one to rasterize the transformed points. These three pipelines are implicit consumer-producer constructs, with one waiting for the completion of the previous before continuing.

3.2 High Level Design

The graphics platform was designed to be able to generate shapes and objects by computing the edges for the object, perform a number of transformations, and to rasterize the scene into a VGA buffer. The module houses three concurrent pipelines: one to generate an edge list, one to compute the transformations, and one to rasterize the transformed points. These three pipelines are implicit consumer-producer constructs, with one waiting for the completion of the previous before continuing.

4 HARDWARE DESIGN

As described in the high level design, there are three pipelines to generate the edges to objects, to compute transformations on the points, and to rasterize the computed elements, the first two pipelines need to store all its data in M4K blocks in order to make them to the pipeline downstream. Since the last pipeline also serves the VGA controller, the third pipeline stores its data in SRAM.

Due to the time constraint, circle, and cube generator have been implemented as part of the project. The other sections which play an important role in texture filtering unit are scaling and rotator units. The pipeline selects only one generator; the positive clock edge initializes the values to the generator, and raises the initialize line for that generator. The generator will raise its 'busy line' when there is a valid edge existing through its output lines [(Ax, Ay, Az), (Bx, By, Bz)], and raise 'done' when it is done.

4.1 Circle Generator Unit

The circle generator unit draws the two-dimensional circles at the specified origin and scale. The flow chart of this unit is shown in Figure 2. This unit will be active when respective arguments are available at positive edge clock cycle. If the reset signal is high then all registers available in the circle generator unit will get cleared. During done_looping case, querying for sine and cosine table will be done, when init signal is high or low. During looping case, the circle will be drawn depending on the sine and cosine values.

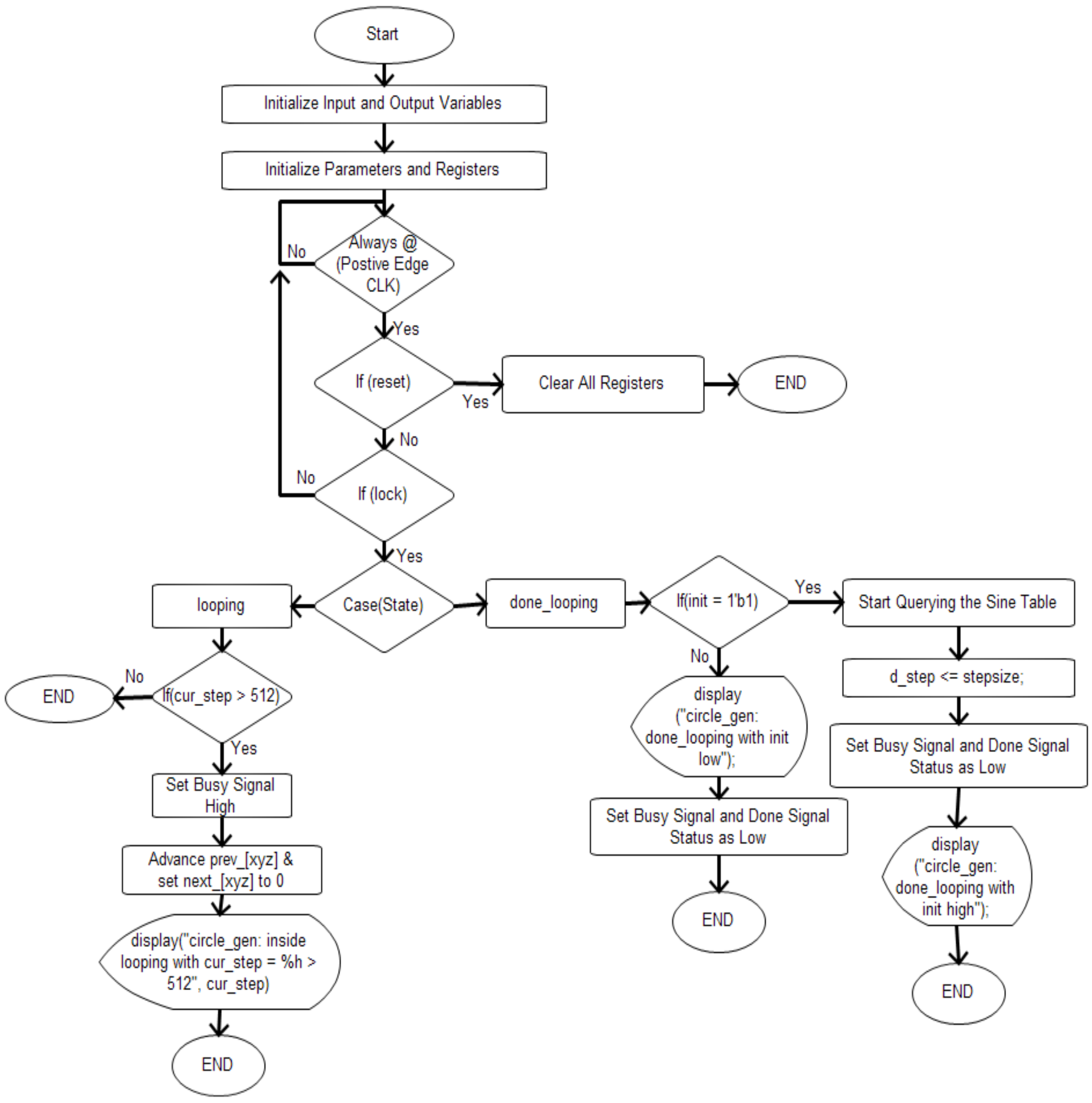

Start
Initialize Input and Output Variables
Initialize Parameters and Registers
Always @ (Postive Edge CLK)
No
Yes
If (reset)
Yes
Clear All Registers
END
No
If (lock)
No
Yes
Case(State)
looping
done_looping
If(cur_step > 512)
No
END
Yes
Set Busy Signal High
Advance prev_[xyz] & set next_[xyz] to 0
display("circle_gen: inside looping with cur_step = %h > 512", cur_step)
END
If(init = 1'b1)
Yes
Start Querying the Sine Table
d_step <= stepsize;
Set Busy Signal and Done Signal Status as Low
display ("circle_gen: done_looping with init high");
END
No
display ("circle_gen: done_looping with init low");
Set Busy Signal and Done Signal Status as Low
END

Figure 2. Flow Chart for Circle Generator Unit

4.2 Cube Generator Unit

The cube generator unit draws the cubes at the specified origin and scale. The flow chart of this unit is shown in Figure 4. This unit will be active when respective arguments are available at positive edge clock cycle. If the reset signal is high then all registers available in the cube generator unit will get cleared. For these model eleven possible combinational cases has been written. A basic model of the cube with has been considered initially as shown in Figure 3 (a). The dimensions of the cube depend on three main axis [(Ax, Ay, Az)] and three imaginary axis [(Bx, By, Bz)]. Whenever, Ay axis is high the cube will change accordingly in y-axis direction as shown in Figure 3 (b), Ax axis is high the cube will change its shape accordingly in x-axis direction as shown in Figure 3 (c), and if Az axis is high the cube will change its shape accordingly in z-axis direction as shown in Figure 3 (d) respectively. Similarly, for other combinations the cube shape will be changed accordingly. The loop will run till all possible cases are checked.

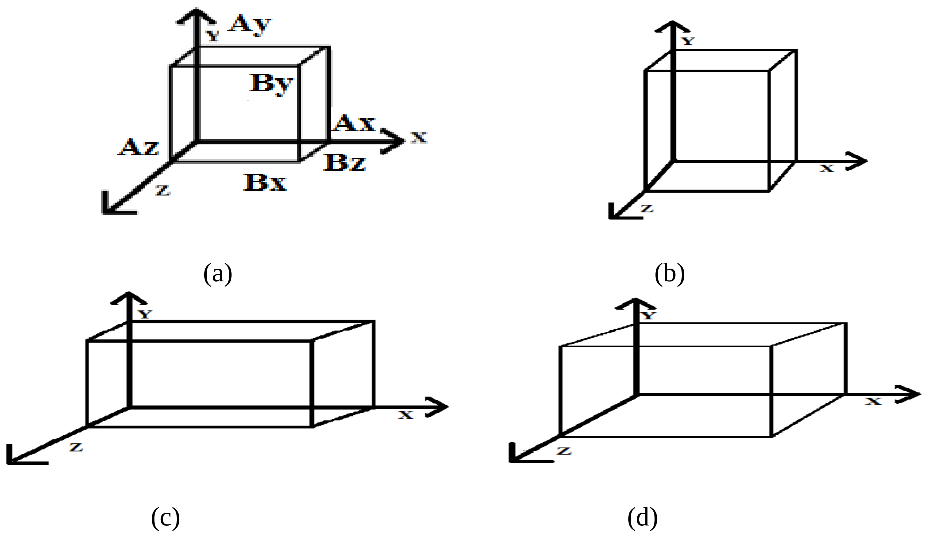


Figure 3. Stages of Cube Generation

4.3 Scaling Unit

Scaling is the process of resizing a digital image. Scaling is a non-trivial process that involves a trade-off between efficiency, smoothness and sharpness. As the size of an image is increased, so the pixels which comprise the image become increasingly visible, making the image appears "soft". The flow chart of scaling unit shown in Figure 5 is explains: reducing an image will tend to enhance its smoothness and apparent sharpness. Apart from fitting a smaller display area, image size is most commonly decreased in order to produce thumbnails. Enlarging an image is generally common for making smaller imagery fit a bigger screen in full screen mode, for example. In “zooming” an image, it is not possible to discover any more information in the image than already exists, and image quality certainly suffers. However, there are several methods of increasing the number of pixels that an image contains, which evens out the appearance of the original pixels.

4.4 Rotator Unit

The rotation process is used to rotate or turn an object based on the angle of rotation required by the user. A rotation transformation is generated by specifying a rotation axis and rotation angle. Parameters are the rotation angle θ and a position (X_r, Y_r) called the rotation point about which the object is to be rotated as shown in Figure 6. And the flow chat is represented in Figure 7.

5. VERIFICATION, TESTING AND VALIDATION

Testing of software or hardware is conducted on a complete system to evaluate the system's agreement with its specified requirements.Testing involves operation of a system or application under controlled conditions and evaluating the results.

5.1 TEST VECTOR FOR PIPELINE MULTIPLIER

The test case result of the pipelined multiplier is shown in Figure 8. The time taken to perform the compilation by using pipelined adder is 60 ps. In this model each bit of the two numbers are multiplied parallely.

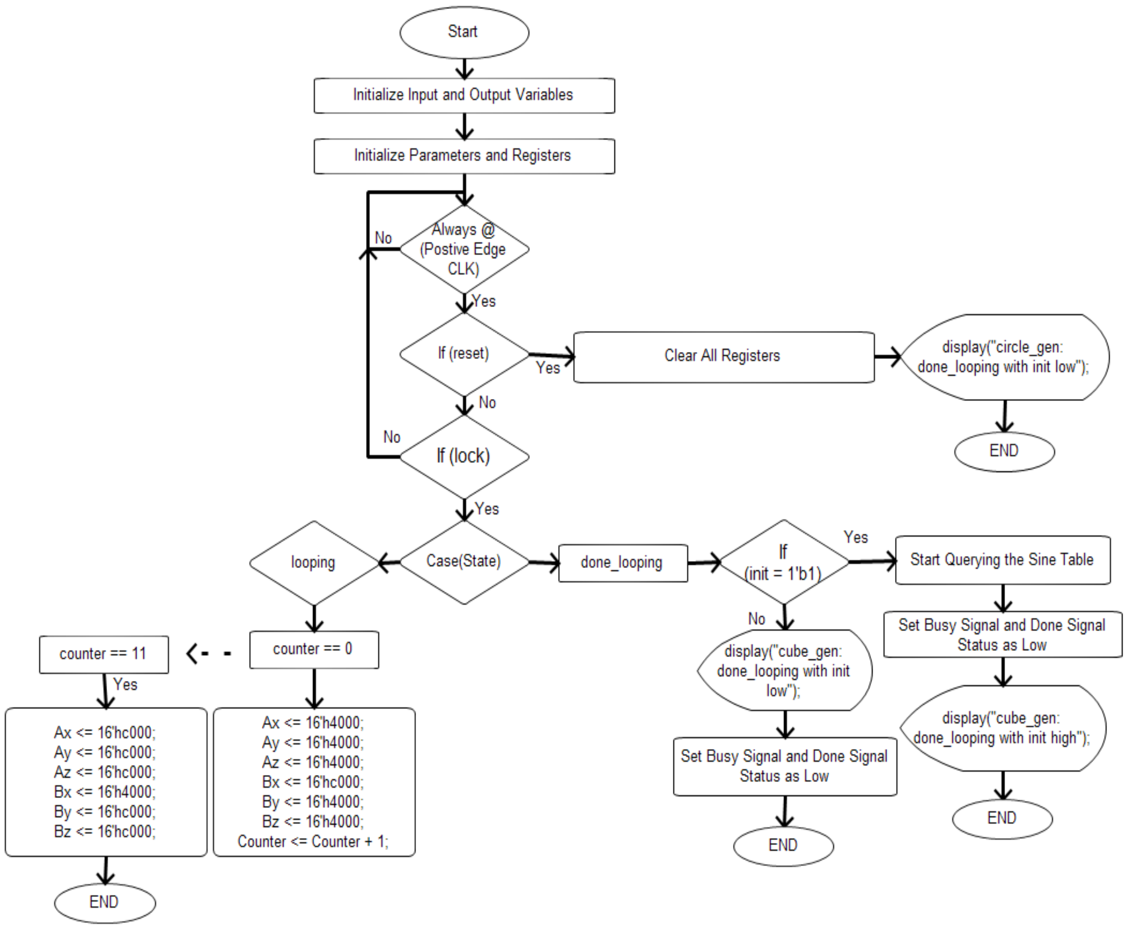

Start
Initialize Input and Output Variables
Initialize Parameters and Registers
Always @ (Postive Edge CLK)
No
Yes
If (reset)
Yes
Clear All Registers
display("circle_gen: done_looping with init low");
END
No
If (lock)
No
Yes
Case(State)
looping
done_looping
If (init = 1'b1)
Yes
No
Start Querying the Sine Table
Set Busy Signal and Done Signal Status as Low
display("cube_gen: done_looping with init high");
END
display("cube_gen: done_looping with init low");
Set Busy Signal and Done Signal Status as Low
END
counter == 0
counter == 11
Yes
Ax <= 16'hc000; Ay <= 16'hc000; Az <= 16'hc000; Bx <= 16'h4000; By <= 16'hc000; Bz <= 16'hc000;
Ax <= 16'h4000; Ay <= 16'h4000; Az <= 16'h4000; Bx <= 16'hc000; By <= 16'h4000; Bz <= 16'h4000; Counter <= Counter + 1;
END

Figure 4. Flow Chart for Cube Generator Unit

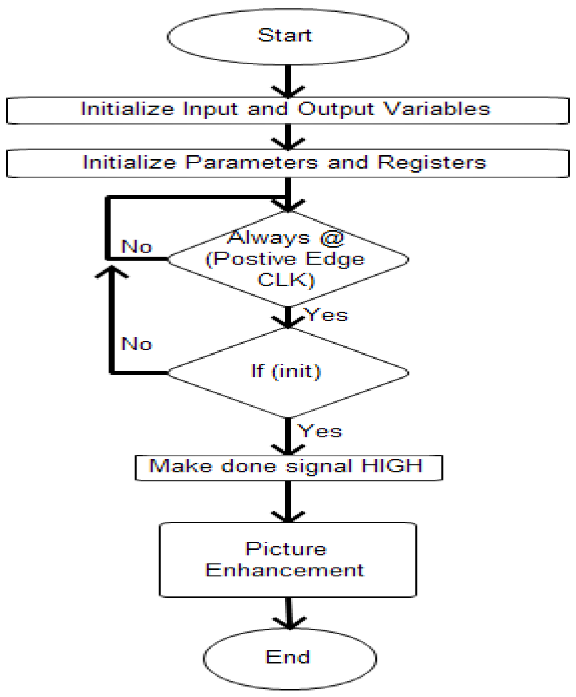

Start
Initialize Input and Output Variables
Initialize Parameters and Registers
Always @ (Postive Edge CLK)
No
Yes
If (init)
No
Yes
Make done signal HIGH
Picture Enhancement
End

Figure 5. Flow Chart for Scaling Unit

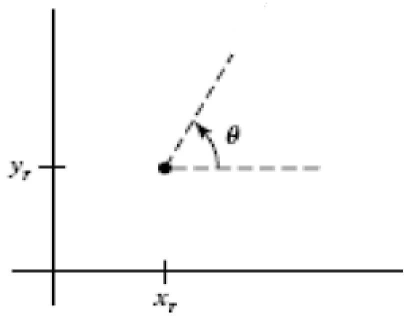
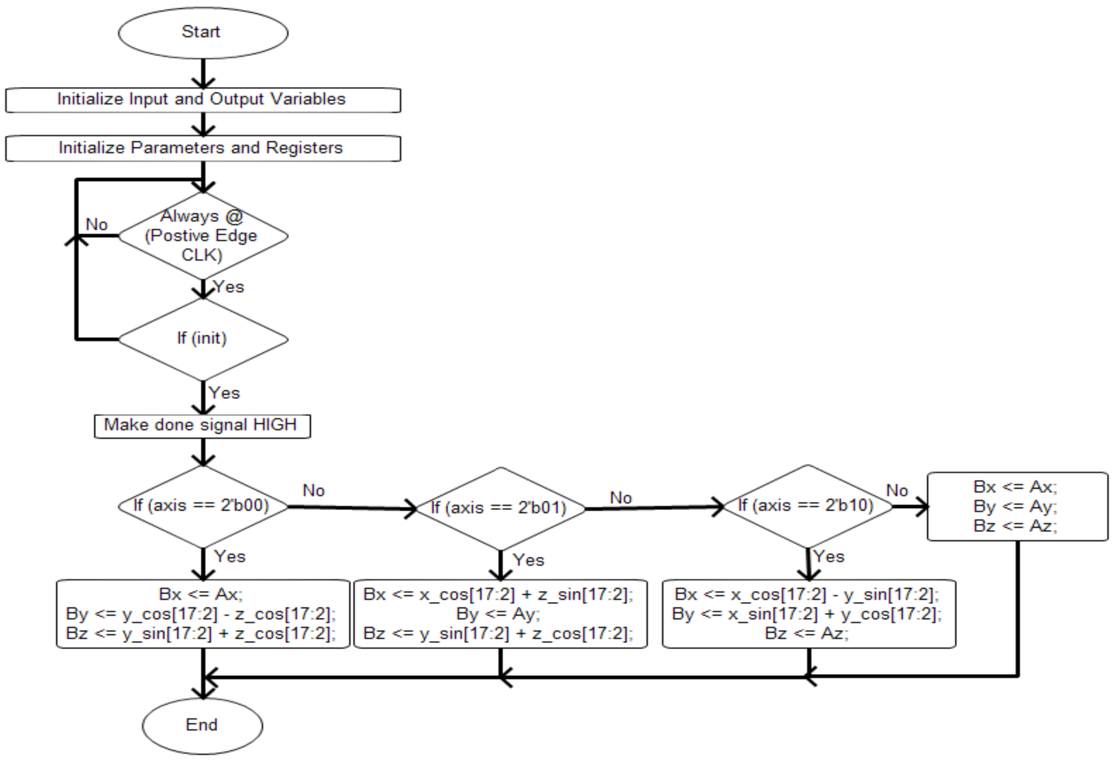
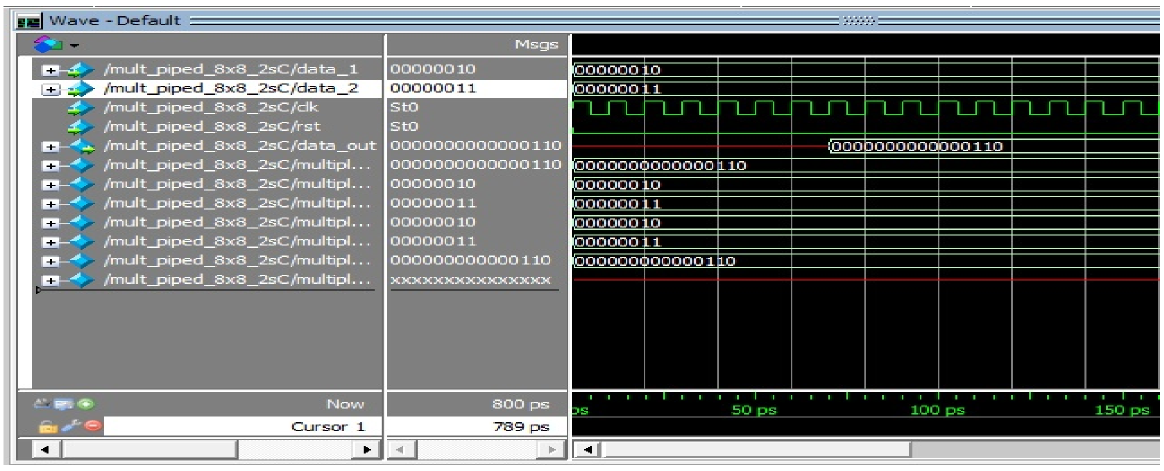


Figure 6. Rotation of an Object through Angle θ about the Position X_r, Y_r



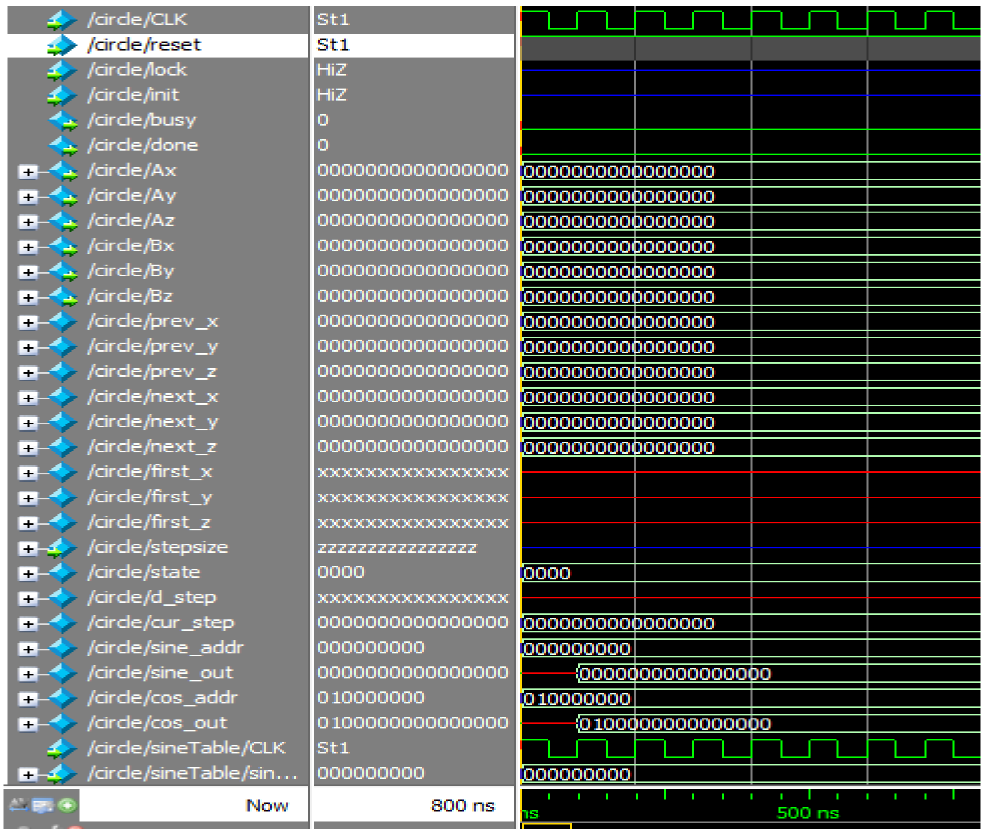
Start
Initialize Input and Output Variables
Initialize Parameters and Registers
Always @ (Postive Edge CLK)
No
Yes
If (init)
Yes
Make done signal HIGH
If (axis == 2'b00)
No
If (axis == 2'b01)
No
If (axis == 2'b10)
No
Bx <= Ax; By <= Ay; Bz <= Az;
Yes
Bx <= Ax; By <= y_cos[17:2] - z_cos[17:2]; Bz <= y_sin[17:2] + z_cos[17:2];
Yes
Bx <= x_cos[17:2] + z_sin[17:2]; By <= Ay; Bz <= y_sin[17:2] + z_cos[17:2];
Yes
Bx <= x_cos[17:2] - y_sin[17:2]; By <= x_sin[17:2] + y_cos[17:2]; Bz <= Az;
End

Figure 7. Flow Chart for Rotator Unit



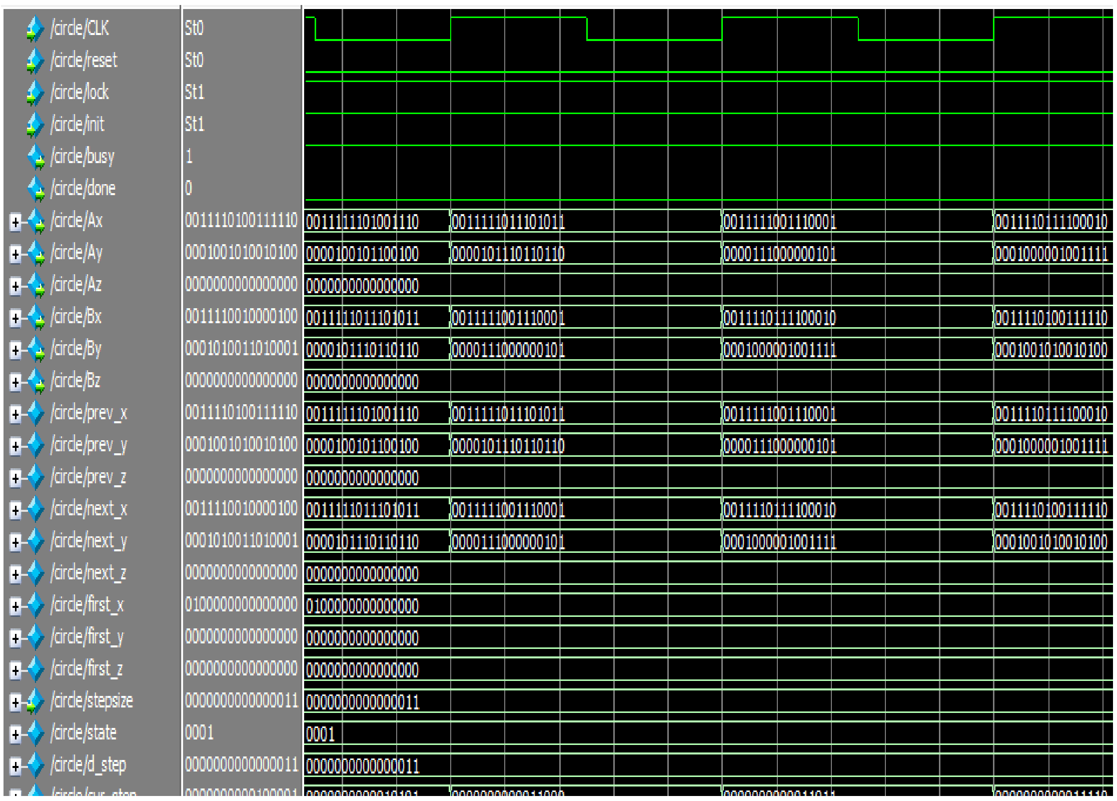
```
/mult_piped_8x8_2sC/data_1       00000010          00000010
/mult_piped_8x8_2sC/data_2       00000011          00000011
/mult_piped_8x8_2sC/clk          St0
/mult_piped_8x8_2sC/rst          St0
/mult_piped_8x8_2sC/data_out     0000000000000110  0000000000000110
/mult_piped_8x8_2sC/multipl...   0000000000000110  0000000000000110
/mult_piped_8x8_2sC/multipl...   00000010          00000010
/mult_piped_8x8_2sC/multipl...   00000011          00000011
/mult_piped_8x8_2sC/multipl...   00000010          00000010
/mult_piped_8x8_2sC/multipl...   00000011          00000011
/mult_piped_8x8_2sC/multipl...   000000000000110   0000000000000110
/mult_piped_8x8_2sC/multipl...   xxxxxxxxxxxxxxx

Now        800 ps      ps   50 ps   100 ps   150 ps
Cursor 1   789 ps
```

Figure 8. Test Case Result of Pipelined Multiplier



```
/circle/CLK             St1
/circle/reset           St1
/circle/lock            HiZ
/circle/init            HiZ
/circle/busy            0
/circle/done            0
/circle/Ax              0000000000000000   0000000000000000
/circle/Ay              0000000000000000   0000000000000000
/circle/Az              0000000000000000   0000000000000000
/circle/Bx              0000000000000000   0000000000000000
/circle/By              0000000000000000   0000000000000000
/circle/Bz              0000000000000000   0000000000000000
/circle/prev_x          0000000000000000   0000000000000000
/circle/prev_y          0000000000000000   0000000000000000
/circle/prev_z          0000000000000000   0000000000000000
/circle/next_x          0000000000000000   0000000000000000
/circle/next_y          0000000000000000   0000000000000000
/circle/next_z          0000000000000000   0000000000000000
/circle/first_x         xxxxxxxxxxxxxxxx
/circle/first_y         xxxxxxxxxxxxxxxx
/circle/first_z         xxxxxxxxxxxxxxxx
/circle/stepsize        zzzzzzzzzzzzzzzz
/circle/state           0000               0000
/circle/d_step          xxxxxxxxxxxxxxxx
/circle/cur_step        0000000000000000   0000000000000000
/circle/sine_addr       000000000          000000000
/circle/sine_out        0000000000000000   0000000000000000
/circle/cos_addr        010000000          010000000
/circle/cos_out         0100000000000000   0100000000000000
/circle/sineTable/CLK   St1
/circle/sineTable/sin...  000000000        000000000
Now                     800 ns             ns   500 ns
```

Figure 9. Test Case Result of Circle Generator when Reset Signal is High



```
/circle/CLK        St0
/circle/reset      St0
/circle/lock       St1
/circle/init       St1
/circle/busy       1
/circle/done       0
/circle/Ax         0011110100111110  0011111101001110  0011111011101011  0011111001110001  0011110111100010
/circle/Ay         0001001010010100  0000100101100100  0000101110110110  0000111000000101  0001000001001111
/circle/Az         0000000000000000  0000000000000000
/circle/Bx         0011110010000100  0011111011101011  0011111001110001  0011110111100010  0011110100111110
/circle/By         0001010011010001  0000101110110110  0000111000000101  0001000001001111  0001001010010100
/circle/Bz         0000000000000000  0000000000000000
/circle/prev_x     0011110100111110  0011111101001110  0011111011101011  0011111001110001  0011110111100010
/circle/prev_y     0001001010010100  0000100101100100  0000101110110110  0000111000000101  0001000001001111
/circle/prev_z     0000000000000000  0000000000000000
/circle/next_x     0011110010000100  0011111011101011  0011111001110001  0011110111100010  0011110100111110
/circle/next_y     0001010011010001  0000101110110110  0000111000000101  0001000001001111  0001001010010100
/circle/next_z     0000000000000000  0000000000000000
/circle/first_x    0100000000000000  0100000000000000
/circle/first_y    0000000000000000  0000000000000000
/circle/first_z    0000000000000000  0000000000000000
/circle/stepsize   0000000000000011  0000000000000011
/circle/state      0001              0001
/circle/d_step     0000000000000011  0000000000000011
```

Figure 10. Test Case Result of Circle Generator When Init Signal is High

5.2 Test Vector for Circle Generator

This unit is used to draw the circular objects in an image. Initially, when the reset signal is high, and all other signals are low, registers in the unit will get cleared as shown in Figure 9. When reset signal is low and lock, init signals are high, at this time step size should be mentioned to draw a circle. It has been observed in the Figure 10 that the outputs [(Ax, Ay, Az), (Bx, By, Bz)] of the block carry the coordinates of the circle. Is has also been observed that for the given step size 0000000000000011, the shape of the circle is changed along Ax, Ay, Bx, and By axis.

5.3 Test Vector for Cube Generator

This unit is used to draw the cubical objects in an image. Initially, when the reset signal is high and all other signals are low, registers in the unit will get cleared as shown in Figure 11. When reset signal is low and lock, init signals are high, at this time step size should be mentioned to draw a cube shape objects. It can be observed in the Figure 12 that the coordinates of the cube changes according to the counter.

5.4 Test Vector for Rotator

This unit is used to rotate the objects in an image. When reset signal is low and lock, init signals are high, at this time the coordinates of the objects in an image which are needed to be rotated are received from the respective functions as shown in Figure 13. The coordinates of the objects in images changes according to the axis.

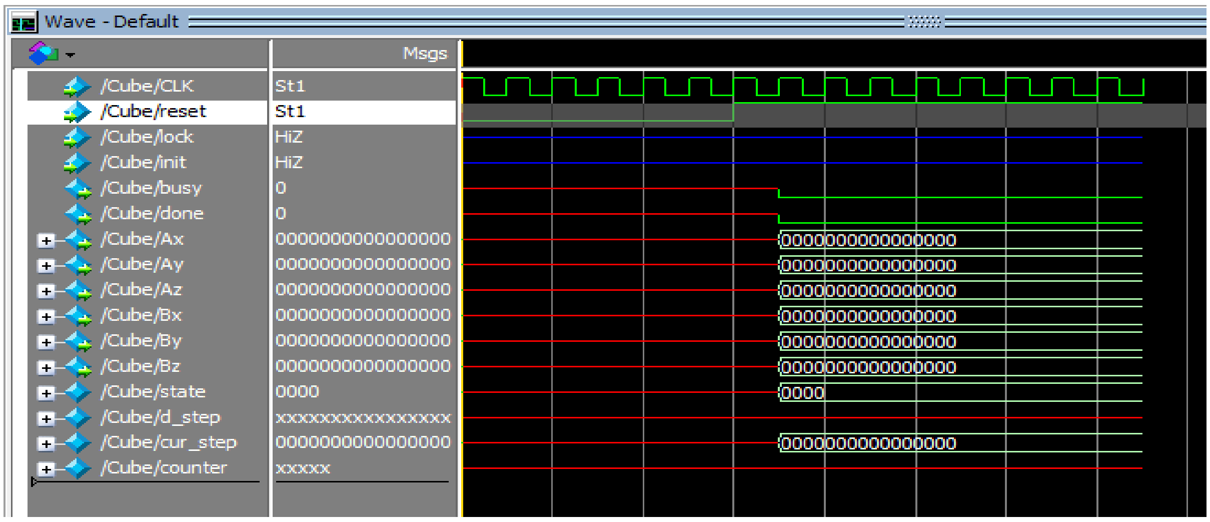

Wave - Default
Msgs
/Cube/CLK St1
/Cube/reset St1
/Cube/lock HiZ
/Cube/init HiZ
/Cube/busy 0
/Cube/done 0
/Cube/Ax 0000000000000000
/Cube/Ay 0000000000000000
/Cube/Az 0000000000000000
/Cube/Bx 0000000000000000
/Cube/By 0000000000000000
/Cube/Bz 0000000000000000
/Cube/state 0000
/Cube/d_step xxxxxxxxxxxxxxxx
/Cube/cur_step 0000000000000000
/Cube/counter xxxxx

Figure 11. Test Case Result When Reset Signal is High For Cube

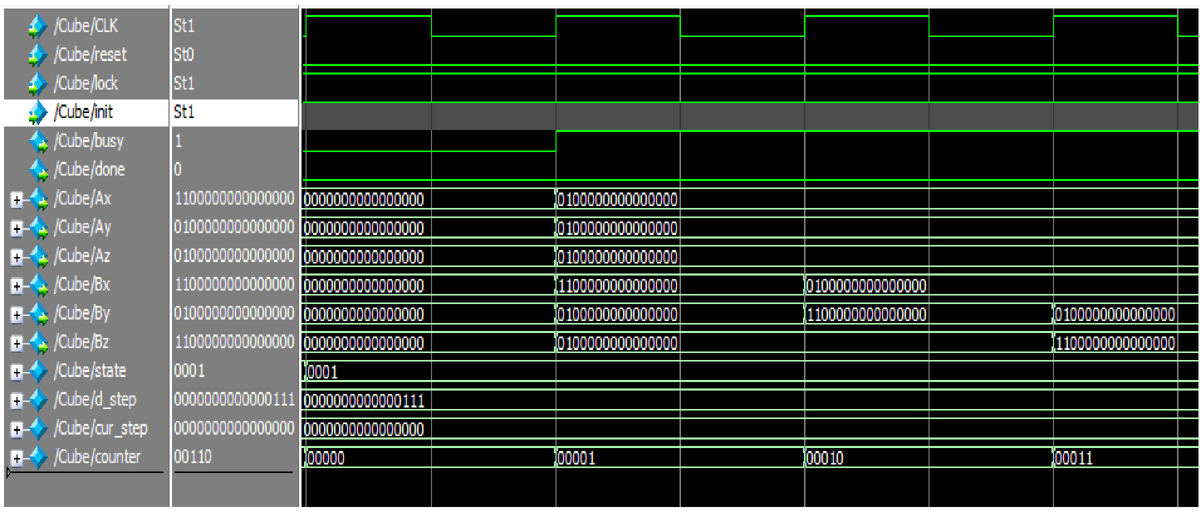

/Cube/CLK St1
/Cube/reset St0
/Cube/lock St1
/Cube/init St1
/Cube/busy 1
/Cube/done 0
/Cube/Ax 1100000000000000 0000000000000000 0100000000000000
/Cube/Ay 0100000000000000 0000000000000000 0100000000000000
/Cube/Az 0100000000000000 0000000000000000 0100000000000000
/Cube/Bx 1100000000000000 0000000000000000 1100000000000000 0100000000000000
/Cube/By 0100000000000000 0000000000000000 0100000000000000 1100000000000000 0100000000000000
/Cube/Bz 1100000000000000 0000000000000000 0100000000000000 1100000000000000
/Cube/state 0001 0001
/Cube/d_step 0000000000000111 0000000000000111
/Cube/cur_step 0000000000000000 0000000000000000
/Cube/counter 00110 00000 00001 00010 00011

Figure 12. Test Case Result When Init Signal is High For Cube

6. RESULTS AND DISCUSSION

Texture filtering unit architecture IP is designed by using verilog programming language, Altera Quartus II, ModelSim, and Active HDL synthesis and simulation tools. In the generated architecture of the texture filtering unit important sections are shown partially in Figure 14. The texture filtering unit is designed using geometrical modelling. The functions of each block of the texture filtering unit are explained below.

6.1 Circle Generator Unit

The circle generator unit draws circles shapes available in an image. This unit has been designed using Bresenham Line Algorithm. This block would generate a two-dimensional circle at the specified origin and scale, so ellipse can be realized.

6.2 Cube Generator Unit

The cube generator unit shown in Figure 15 draws cube shapes available in an image. This unit has also been modelled using Bresenham Line Algorithm. This block would generate a cube at the specified origin and scale. Whenever, there is a change in the output of the respective axis [(Ax, Ay, Az), (Bx, By, Bz)] the shape of the cube will be changed accordingly. The three transformations currently supported in this model are scaling, translation, and rotation on x, y, z axis.

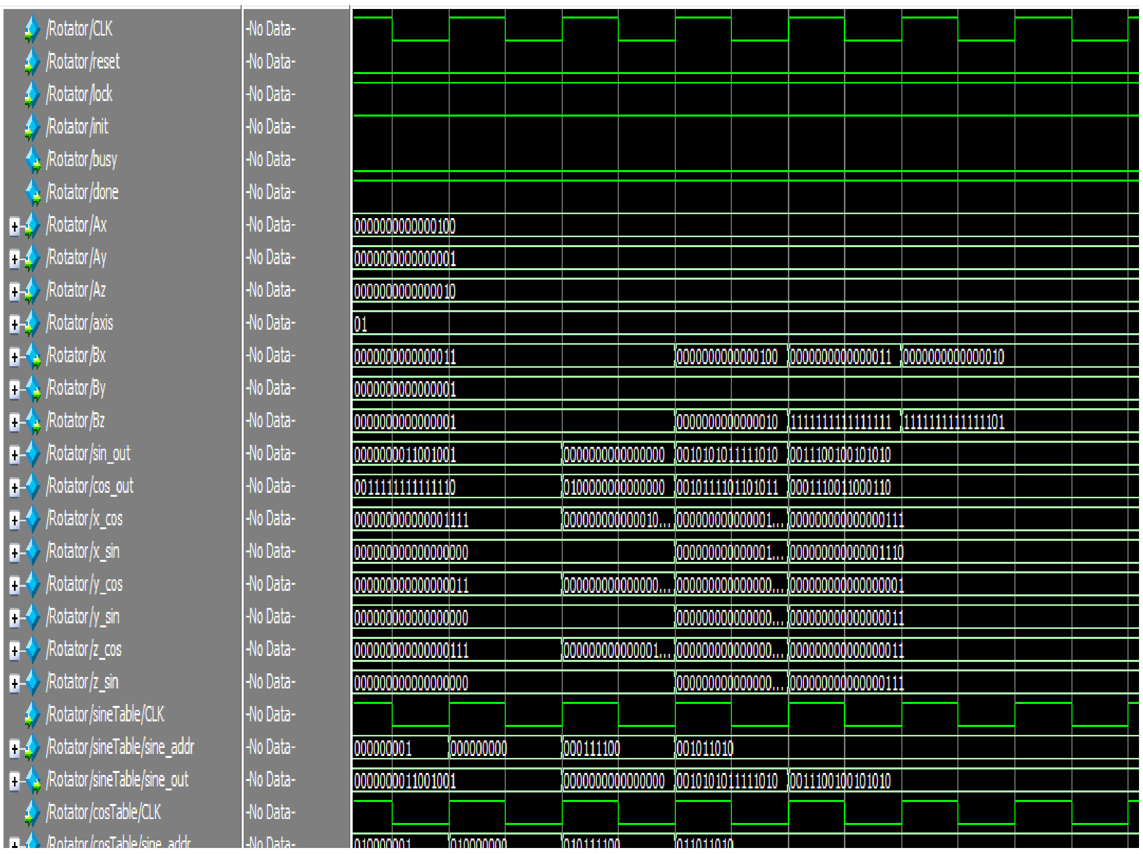


Figure 13. Test Case Result For Rotator

6.3 Rotate Unit

The rotation process is used to rotate or turn an object based on the angle of rotation required by the user. A rotation transformation is generated by specifying a rotation axis and rotation angle. Parameters are the rotation angle θ and a position (X_r, Y_r) called the rotation point about which the object is to be rotated. The generated block of the rotate unit is shown in Figure 16. It has inputs of 16-bit [(Ax, Ay, Az)] and theta. The changed coordinates of the object are sent through the imaginary points [(Bx, By, Bz)] respectively.

6.4 Scale Unit

The unit shown in Figure 17 helps to scale the edges of the objects to the specified origin and scale. This helps in adding effects to the overall object being rendered by the GPU and more specifically by the texture filter. The object rendering is carried out by the GPU and enhancement of effects is done by other units. Texture scaling is used to enhance effects on the object for example shadows, edges, brightness etc. This adds to the overall look and feel.

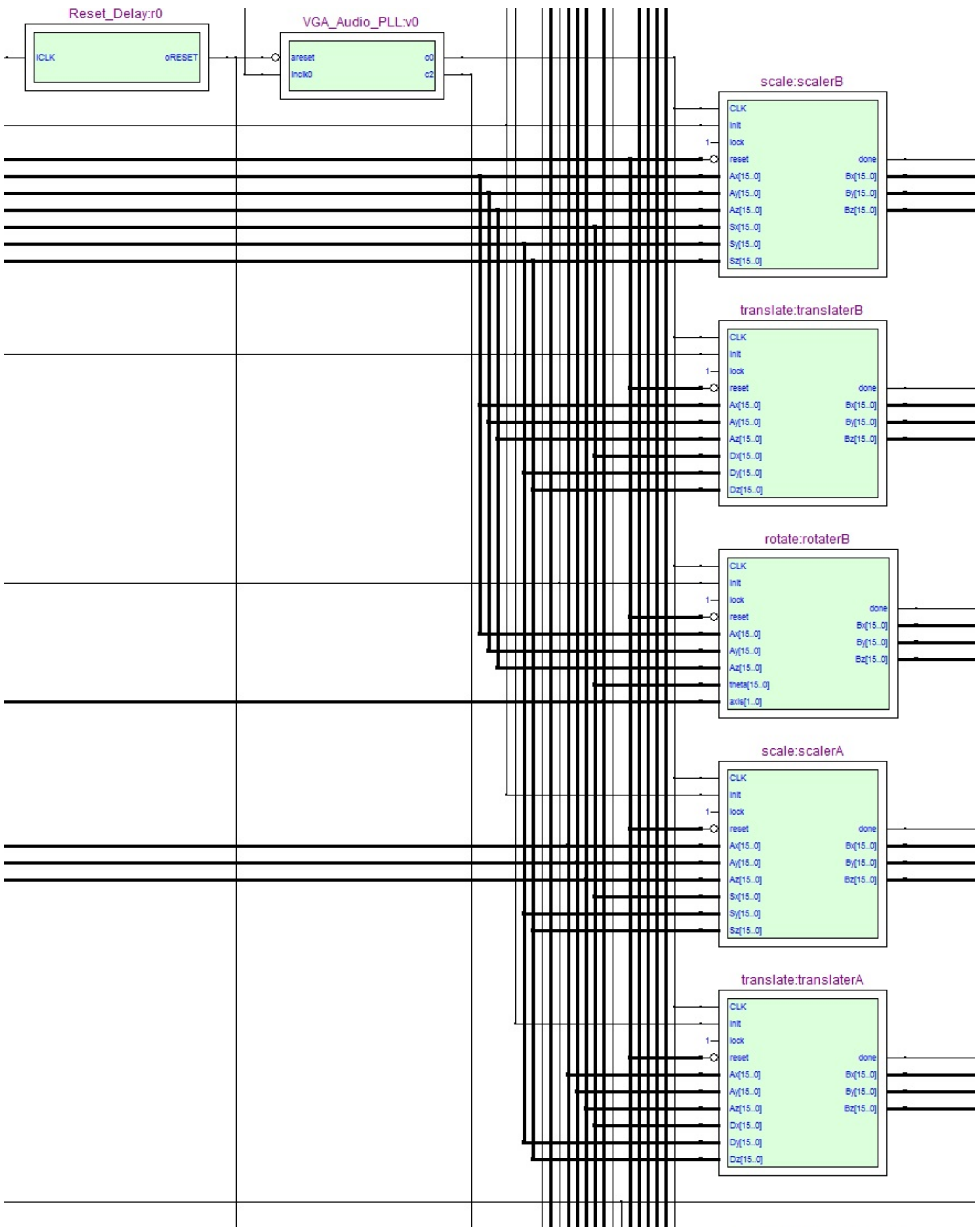


Figure 14. Partial Generated Circuit of the Texture Filtering Unit

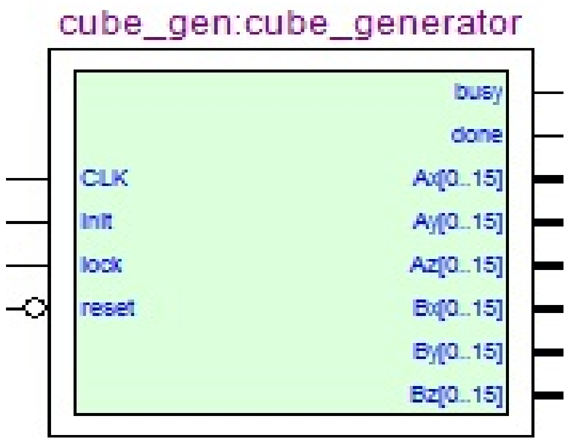


Figure 15. Generated Block Diagram of the Cube Unit

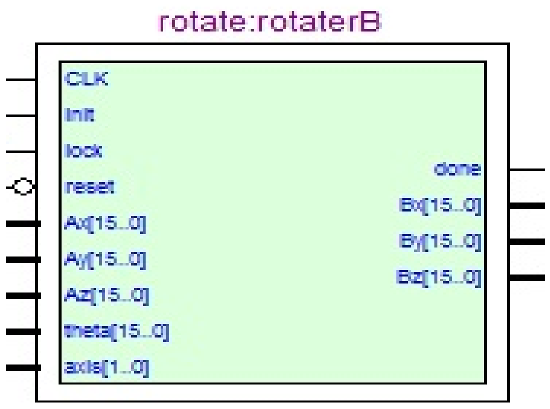


Figure 16. Generated Block Diagram of the Rotate Unit

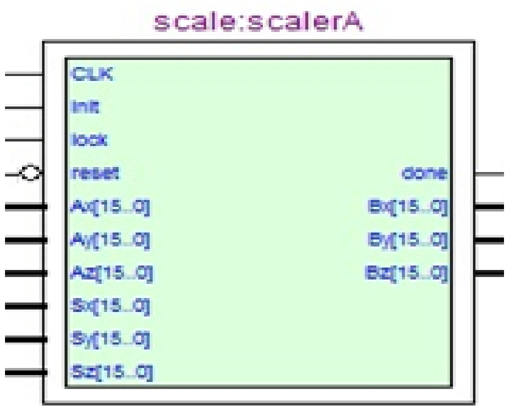


Figure 17. Generated Block for Scaling Unit

7. CONCLUSIONS

Based on design specifications, texture filtering architecture had been designed and verified.but due to certain resources constraints mapping to FPGA had not been achived. The available memory on the FPGA is limited to 8.5 megabytes at most, of which about 512 kilobytes are available memory that are designed to be read from within a single cycle of exerting the desired address. A reconfigurable architecture is chosen due to the following two reasons (1) Area reduction is better achieved using all-purpose texture filters because they are not idle as the required filter type varies during run-time; (2) the reconfiguration overhead is low because the configuration does not switch very often.

REFERENCES

[1] Altera Corporation, "Introduction to Simulation of Verilog Design Using ModelSim Graphical Waveform", February 2012.ftp://ftp.altera.com/up/pub/Altera_Material/9.1/Tutorials/Verilog/ModelSim_GUI_Introduction.pdf [13 February 2012]

[2] Arvo J., "Graphics Gems II" *USA:Academic Press Inc*.

[3] S.Banerjia,E.Ozer,and T.M.Conte, "Unified Assign and Schedule:A New Approach to Scheduling for Clustered Registered File Microarchitectures, *Proceeding of IEEE International Symposium on Microarchitecture,* pp308-315,2009

[4] Q.K.Chen and J.K Zhang, "A Stream Processor Cluster Architecture Model with the Hybrid Technology of MPI and CUDA", *Proceedings of IEEE International Conference on Information Science and Engineering,page nos 86-89,*2009

[5] J.Chittarmuru, J.Euh and W.Burleson, "A Low-Power Content-Adaptive Texture Mapping Architecture for Real-Time 3D Graphics",University of Massachusetts Amherst,http://www.citeseerx.ist.psu.edu[December 2012]

[6] Doulos Inc., "OVM Golden Reference Guide,Version 2.0", *Hamspire:Doulos*,2008

[7] Doulos Inc., "The Verilog Golden Reference Guide, Version 1.0",*Hamspire:Doulos*,1996

[8] J.D.Foley,A.V.Dam,S.K Feiner, and J.K.Hughes, "Computer Graphics:Principles and Practice in C, 2nd edition.",*USA:Addison-Wesley*.,1990

[9] K.S. Fu, "Syntactic Pattern Recognition and Application", *New Jersey:Prentice-Hall.*,1982

[10] P.N.Glaskowsky, "NVIDIA's Fermi: The First Complete GPU Comuting Architecture, <http://www.nvidia.com/ content/pdf/fermi_white_papers/P.Glaskowsky_Nvidia's_fermi_the_first_complete_GPU_architecture. pdf>[8 December 2011]

[11] A.Greß and G. Zachmann, "GPU-ABiSort: Optimal Parallel Sorting on Stream Architectures" *Proceeding of IEEE Conference on Graphics Hardware*,2006

[12] M.Gschwind,V.Salapura., "A VHDL Design Methodology for FPGA",*Proceeding of FPL`95 of the 5th International Workshop on Field-Programmable Logic and Application*,pp 208-217,1995

[13] A.Gupta and Z.S Hakura, "The Design and Analysis of a Cache Architecture for Texture Mapping" <http://www.cs.cmu.edu/afs/ cs/academics/class/15869-f11/www/readings/hakura97_texcaching.pdf> [8 December 2011].

[14] P.S.Heckbert, "Survey of Texture Mapping" *IEEE Transcation of Computer Graphics and Applications*,6,pages nos.56-67,1986

[15] Intel's Next Generation Integrated Graphics Architecture – *Intel Graphics Media Accelerator X3000 and 3000,2006* http://www.intel.com/products/chipsets/gma3000/gma3000.pdf [16 November 2011]

[16] E.Kilgariff, R.Fernando, "The GeForce 6 Series GPU Architecture", http://www.nvidia.com/page/geforce6.html> [8 December 2011]

[17] C.H.Kim and L.S.Kim, "Adaptive Selection of an Index in a Texture Cache", *In Proceedings of IEEE International Conference on Computer Design:VLSI in Computers and Processors*,pp 295-300,2004

[18] Lattice Semiconductor Corporation, " FPGA Design Guide" http://www.lattticesemi.com/lit/docs/manuals/fpga_design_guide.pdf, [5 January 2012]

[19] J.McDonald, "Insider Guide:FPGAs,Tools and Boards," 2008, http://www.eg3.com/report-fpga[30 September 2011]

[20] NVIDIA, "GeForce 8800 GPU Architecture Overview",*White paper TB-02787-001_v0.9*, http://www.nvidia.com/object/IO_37_100.html, [16 November 2011]

[21] NVIDIA, "GeForce GTX 200 GPU Architecture Overview ",*White paper TB-04044-001_v01*, http://www.nvidia.com/docs/IO/55506/GeForce_GTX_200_GPU_Technical_Brief.pdf [16 November 2011] object/IO_37_100.html, [16 November 2011]

[22] S.Rajagopal, J.R.Cavallaro and S.Rixner, "Design Space Exploration for Real-Time Embedded Stream Processors" *Proceedings of IEEE Computer Society*,2004

[23] F.Remond, "The Work Flow of a Block-Based Design Team,Integrated System Design ",2004, http://www.eedesign.com/editorial/2000/designtools0012.html, [16 November 2011]

[24] J.Sanders, E.Kandrot and J.Dongarra, "CUDA by Example an Introduction to General-Purpose GPU Programming", *Boston:Addison-Wesley*,2011

[25] H.C.Shin,J.A,Lee and L.S.Kim, "A Cost-Effective VLSI Architecture for Anisotropic Texture Filtering in Limited Memory Bandwidth",*IEEE Transactions on VLSI Systems*,14,pp 254-257,2006

[26] Stream Processors Inc., "Stream Processing :Enabling the New Generation of Easy to Use,High-performance DSPs" ,*White Paper,USA*,2008

[27] F.Tomita,S.Tsuji, "Computer Analysis of Visual Textures",*Boston:Kluwer Academic Publishers*,1990

[28] H.Voorhees and T.Poggio, "Detecting Textons and texture Boundaries in natural Images",*Proceedings of the First International Conference on Computer Vision,London*, pp250-258,1987

[29] W.T.Wang, Y.C.Chen,C.P.Chung, “A Run-Time Reconfigurable Fabric for 3D texture Filtering”, *Proceedings of IEEE International Conference on Application Specific Systems,Architectures and Processors*,pp 180-185, 2007
[30] C.M.Witternbrink,E.Kilgariff and A.Prabhu, “FERMI GF100 GPU Architecture”,*IEEE MICRO,31*,pp 50-59,2011
[31] Xilinx, “System Generator for DSP User Guide Release 10.1”, http://www.xilinx.com/support/sw_manuals/sysgen_user.pdf, [20 December 2011]
[32] S.W.Zucker, “Toward a model of Texture”,*Computer Graphics and Image Processing,5*,pp 190-202, 1976

BIOGRAPHIES OF AUTHORS

Krishna Bhushan Vutukuru received his BSc, MSc degree in Electronics and MSc[Engg] in Real-Time Embedded Systems from Osmania University, Acharya Nagarjuna University, India in 2005 ,2007 and Coventry University,UK in 2012.His research interests include the field of System on Chip Design, Digital Design and FPGA and Embedded Systems.

Sanket Dessai received his BSc, MSc degree in Physics and MSc[Engg] in Real-Time Embedded Systems from Goa University,India in 2001 ,2004 and Coventry University,UK in 2007.Since 2007 he has been a Assistant Professor at M .S.Ramaiah School of Advanced Studies(MSRSAS),Bengaluru in collaboration with Coventry University.His research interests include the field of System on Chip Design,Embedded Systems, MEMS/NEMS Engineering,Nanophysics and Nanotechnology,Solid State Physics and Engineering and Photonics