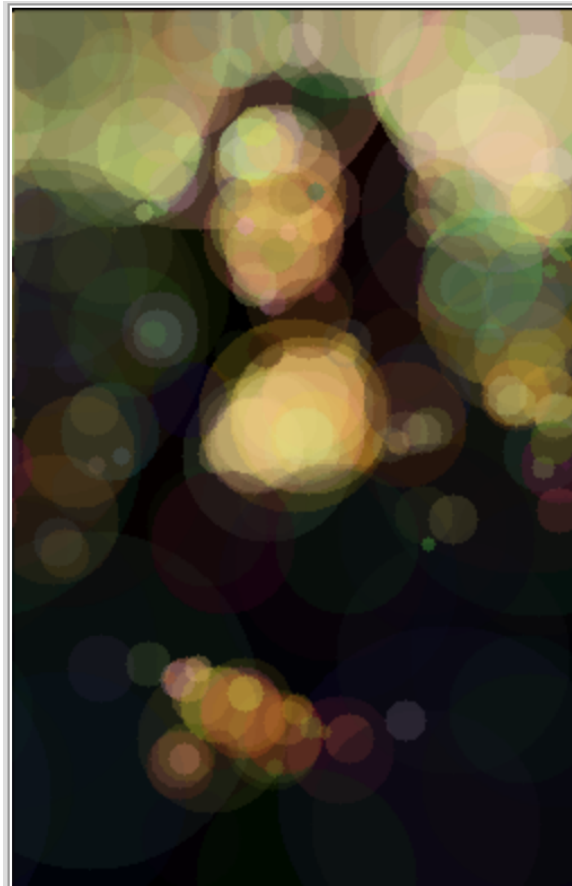


# **The Evolution of a Smile**

## **A Genetic Algorithm with FPGA Implementation**

Jihua Li - jl4345  
Wenbei Yu - wy2228  
Yini Zhou - yz2719  
Jian Jiao - jj2756



# Overview

In recent years a great concern focuses on accelerating time-consuming algorithms that solve large combinatorial optimization problems [1]. One is Genetic Algorithms.

Genetic algorithms, known as one of robust heuristic algorithms for complex optimization problems in various fields of engineering, find application in bioinformatics, computational science, economics and many other fields. It is a powerful technique that implements the principles nature uses in biological evolution to optimize a multidimensional nonlinear problem and provides robust capability of exploring in the solution space of a given problem [2].

However, one big problem of this algorithm is the computation time. The amount of computations and iterations required for this method is enormous. As a result, software implementations of GA can become extremely slow for large circuit partitioning problems. To reduce the execution time of GA, hardware implementation of GA has been proposed[3].

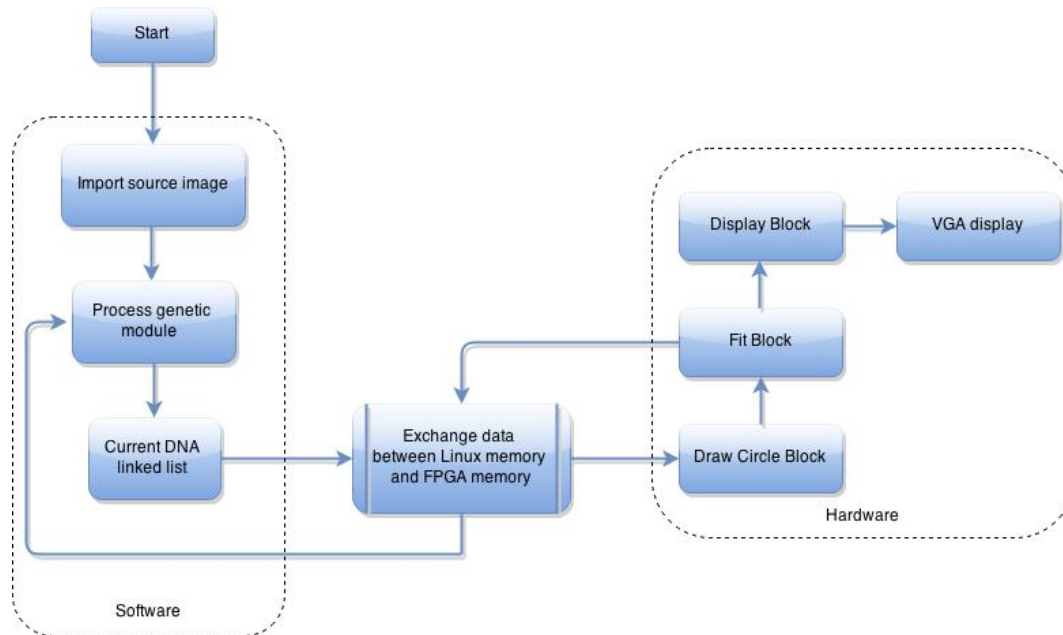
In this project, we design an accelerator for Genetic Algorithm to generate Mona Lisa or any other images with circles which are generated randomly in DNA sequence. The goal of this project is to demonstrate Genetic Algorithm and to speed up the algorithm with the FPGA implementation when compared to running on a regular CPU.

Based on the purpose of this project, we choose to implement the function of drawcircle, fitting and padding in hardware instead of software to achieve higher speed, and remain to use software for mutation and overall control. The whole project can moderately accelerate the algorithm and is much faster than software implementation.

# Design and Implementation

## Architecture Overview

The architecture of our project is shown as bellow. Details of software, hardware and interface are discussed in the following parts.



## Software - Overview

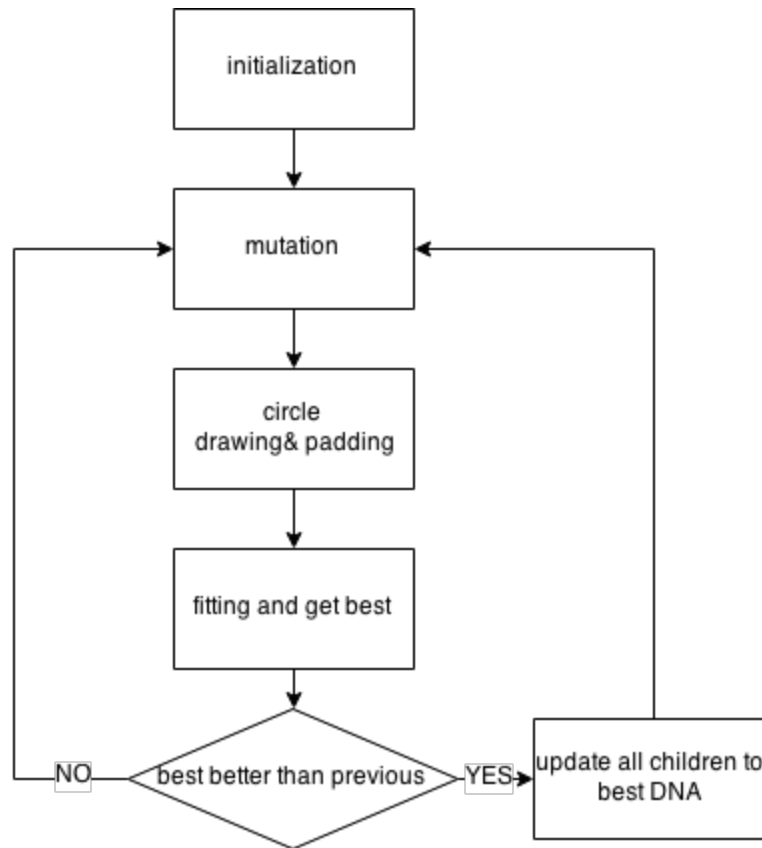
The genetic algorithm is running on on-board linux soc platform using resources from hardware accelerator to boost its running speed. A device driver is written to allow software communicating with hardware interface.

Software structure can be divided into several parts listed below.

- Algorithm
  - circle drawing and padding
  - mutation
  - fitting and comparison
  - best child in generation updating
- Hardware accelerator interface
  - start signal triggering
  - data writing
  - data reading
  - status register reading
  - hardware mode switching

## Software - Details

### ASM of genetic algorithm



### Data structure for DNA

DNA in this genetic algorithm are in form of linked list. Each node of the linked list is saving a certain information of a circle including x,y position, the radius, color and opacity of the circle.

```
typedef struct circ {
    int x, y, rad, color;
    int opacity;
    struct circ* next;
} circle;
--codes in test.h in the software part of the project
```

Above is the node definition of the linked list mentioned. Because color is composed by three RGB values each of which have a bit length of 8 bits, we use color to represent all RGB values from the 24th bit down to the 1st bit.

For opacity, we used large integer to represent floating points since floating points cannot be directly processed by FPGA hardware.

## Initialization

### Initialize and allocate memory for linked list with 100 children.

Each child have a start node of the circle linked list and an array is adopted for saving all the starting nodes of all the start nodes of the circle linked list. In our codes, this array is represented by the name of “data”.

data[i] is namely the i-th child of the generation.

### Srand initialization

srand was initialized for further random number generation. The randomized number in the following programme are used to generate equal chance for decision of the evolution direction.

### Best fit values initialization

Initialization of the current best child and best fit value in name of “best” and ”diff” respectively. These value will be updated whenever a better fit comes out.

### Device driver interface initialization

Device driver for communicating with hardware interface will also be initialized.

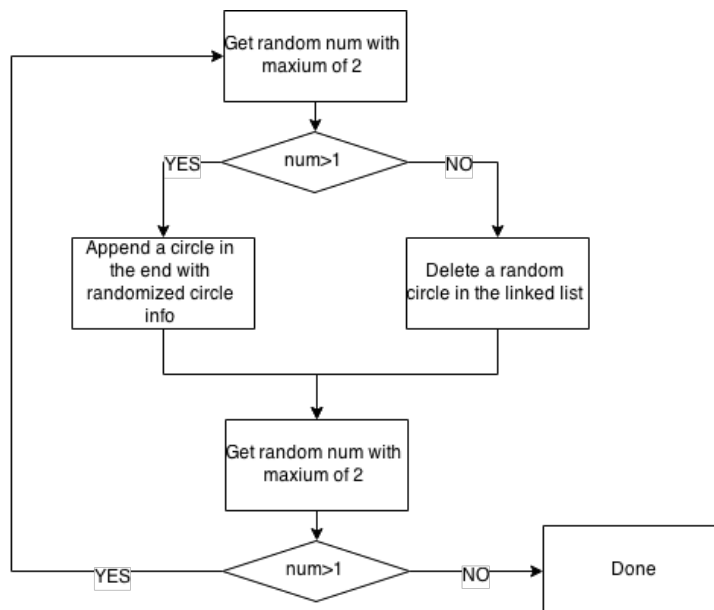
### Monalisa original picture loading

A mona lisa picture data was loaded from a local file named “mldata” which was generated by Matlab. All pixel data was loaded and put in an array. The format of mldata is in hex consisting 60000 length data of pixels in format of integer(32 bits).

### Hardware original picture initialization

The loaded mona lisa pixel data are written into hardware through interface

## Mutation



As can be shown from the mutation ASM above, with equal chance, the function either append a circle or delete a circle node from the linked list. Then also with equal chance, the function either jump out or doing recursively.

### **Circle drawing and padding**

This part was originally implemented in software but later moved to hardware for acceleration. After hardware implementation, software just need to write circle information and then give a start signal.

In hardware, the circle drawing status is set by writing 0x10004 to hardware. Circle information of x,y,r are given in a single integer being writing to address 0x10001 in hardware. 0x10002 for opacity and 0x10003 for color. Start signal of drawing circle are triggered by reading 0x00003 from the hardware.

### **Circle drawing pipelining**

To boost up speed of drawing circles, pipelining methodology is adopted. In code, instead of checking and waiting for one circle to complete then exit, we read and wait for the previous circle drawing to finish and then put present circle information into hardware and then give start signal. Then software is free to generate next circle information while hardware is drawing circle on the ram.

### **Fitting and get best**

Fitting is also implemented in software originally but later on moved to hardware for acceleration. For hardware transplanted operation, write to address 0x10007 for changing mode to fitting mode on hardware. write address 0x10000 to give fitting part the number of children in each generation. Reading of address 0x00006 gives a start signal for fitting module.

Because we give the children number to hardware, the fit module will get best index and difference value done automatically after fitting all children of a generation. Hence, after fitting of each generation, the best value and minimal difference can be read from address 0x00000 and 0x00001 respectively.

### **Update best values**

Each time when a new best index and smaller difference found, the best values are being updated. In the algorithm, we first update the best index and difference value, then we free all the circle linked list except for the best one. Later on, the best circle nodes are copied to all the children of a generation to be the basis where next mutation is performed.

## **Hardware accelerator interface**

### **start signal triggering**

Start signals are triggered by reading registers from hardware, these registers are actually virtual, return data are not of our interest. The address for different start triggering is listed below.

```
#define CIRC_START      3
#define CLEAN_START    4
```

```
#define CP_START 5
#define FIT_START 6
--codes in test.h showing the address for different start signals
```

### **data writing**

Few data need to be transferred to hardware to perform correct function. The number of children for each generation is needed by fitting module. Besides, all circle data is needed in circle drawing module. The addresses for writing different data are listed below.

```
#define GEN_NUM 0x10000
#define CIRC_DIM 0X10001
--Includes x,y,rad information
--x,y are 9 bits each and rad is 7 bits
#define CIRC_OPA 0X10002
--opacity 24 bits
#define CIRC_COLOR 0X10003
--color 24 bits, RGB 8 bits each in format {R,G,B}
```

### **data reading**

Best values need to be read from fit module for updating operation. Different address are listed below.

```
#define FIT_DIFF 0
Minimal difference value each generation.
#define BEST_GEN 1
Best child of each generation
```

### **status register reading**

Status register can be read from address 0x00002. Certain bit fields of the status register are listed below for further information

```
#define STATUS 2
#define CIRC_READY 0X8
#define CLEAN_READY 0X4
#define CP_READY 0X2
#define FIT_READY 0X1
```

### **hardware mode switching**

Hardware works like a state machine, so each time when we want to change to another function, we have to change status on the hardware.

This function can be further improved to achieve faster speed by automatically switching modes based on knowledge of done and ready signals.

Address listed below can be written to change to different states on hardware.

```
#define DRAW 0X10004
#define CLEAN 0X10005
#define COPY 0x10006
```

```
#define FIT            0x10007
#define WRITE_LEFT    0x10008
#define WRITE_RIGHT   0x10009
#define WRITE_TMP     0x1000A
```

## **Device driver**

Driver of this project is different from lab3 before because instead of only writing data to hardware, we also read data and return to user through avalon bus.

Also, in order to achieve virtual address on hardware, the width of a address would be 17. 16 bit are used for writing chunks of data to a ram with size of 60000. One extra bit indicates whether it's writing to a memory or other operation.

The data bus width we need from hardware is 24, but from software perspective, we adopted 32 bits to read and write data with `ioread32` and `iowrite32`.

In dts file, the address of the hardware is changed to range from `0x0000` to `0x7fff`.

## **Memory Requirement**

Originally, 4 rams are needed to perform a pipelined operation of drawing and fitting and displaying but the on-chip ram won't fit 4 rams of size 60000. Therefore, three rams are used to store our data.

All rams we used are dual-port ram.

Ram `ram_ping` is a temporary ram for saving pixels of circles that are drawn so that circle drawing module can read and pad circles on top of existing circles. Both of the ports are used for circle drawing in parallel.

`left_ram` is used to store best generation's circle pixels. It's manipulated by both copy function and display module. One of the ports are always connected to display module allowing VGA fetching data at anytime. The other port is used for copy module to copy the internal temp ram to this display one.

`right_ram` is used to store source image data and allow display module to read and show on VGA the two images.

Each image consists of 200\*300 pixel. Each pixel is comprised of three bytes, R, G, B. So the total memory usage is 540 KB.

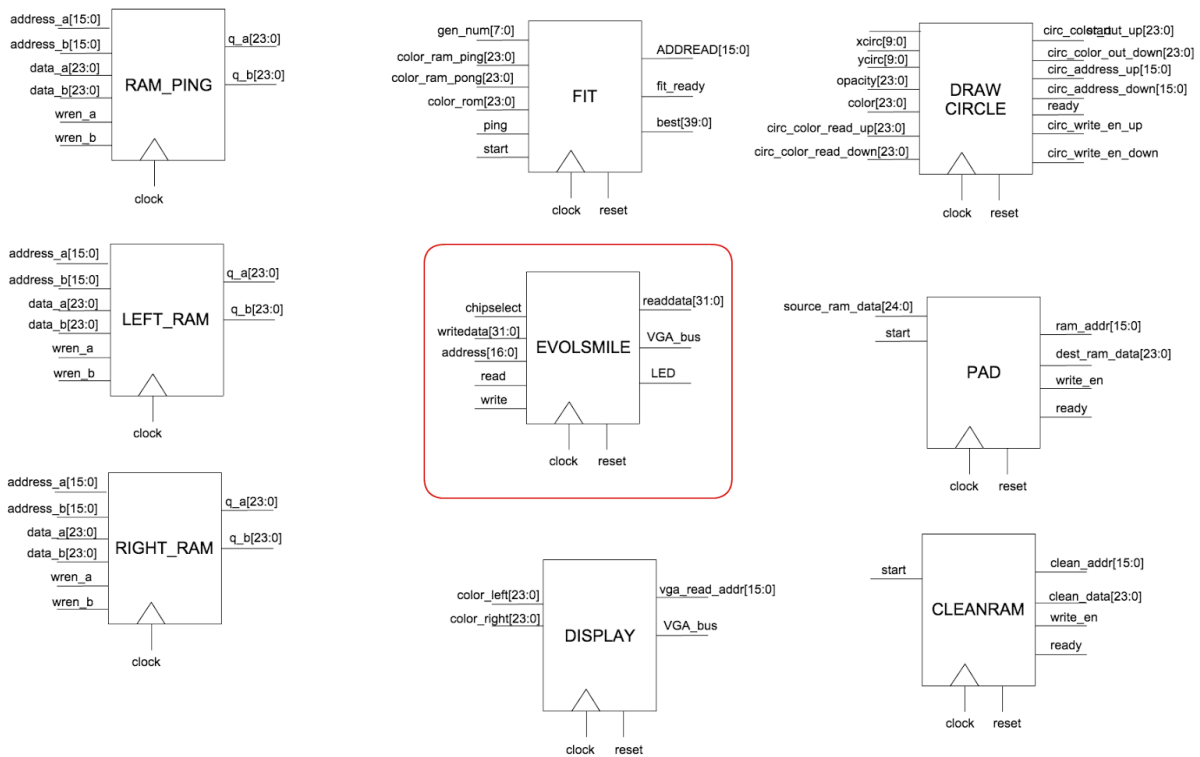


# Hardware - Overview

The hardware components of this project are primarily responsible for the following:

1. Display and vga\_emulator module display source image and generated image on the VGA screen.
2. Drawcircle module stores each generation's circles data in ram\_ping.
3. Fit module compares value between ram\_ping and right\_ram to find out the best generation.
4. Pad module copies best generation's data to left\_ram to display it on the screen.
5. Clean module cleans the temp ram for further drawing.

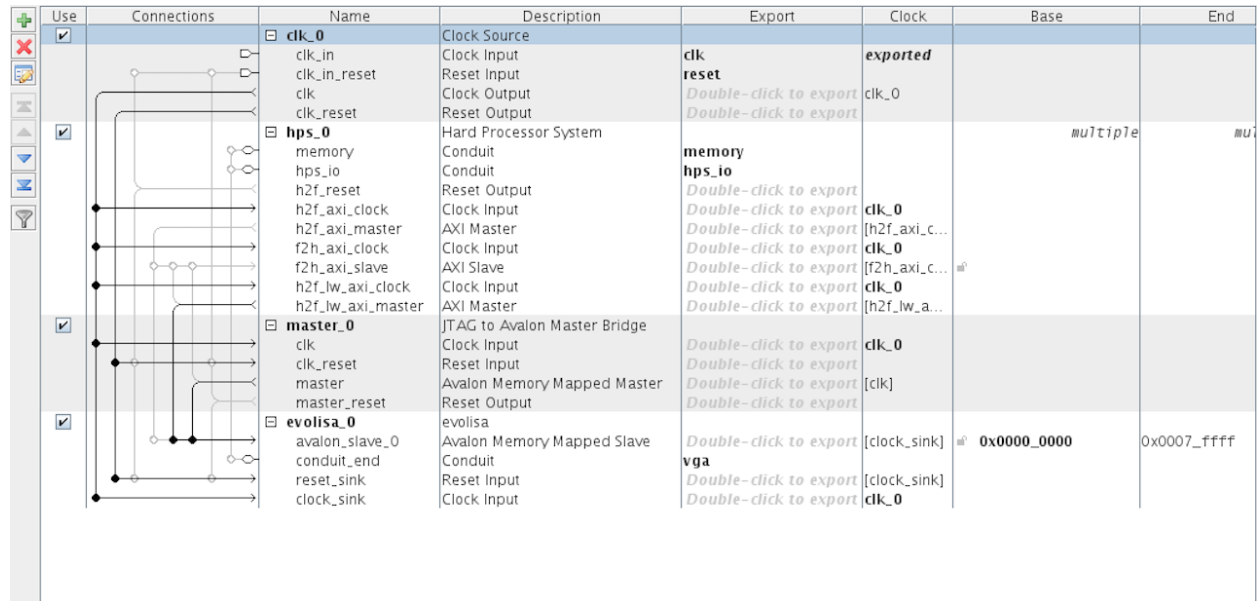
The figure below shows the block diagram for each module used in hardware.



## Hardware - Details

### Peripherals

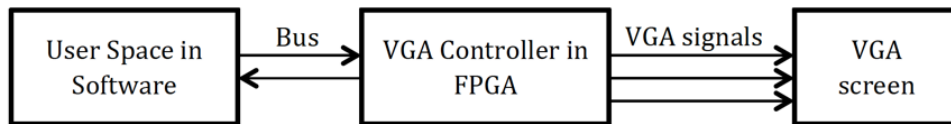
Peripheral evolisa is used in this project. The following screenshot from Qsys shows the peripheral connections:



Use	Connections	Name	Description	Export	Clock	Base	End
<input checked="" type="checkbox"/>		<b>clk_0</b>	Clock Source				
		clk_in	Clock Input	clk	exported		
		clk_in_reset	Reset Input	reset			
		clk	Clock Output	Double-click to export	clk_0		
		clk_reset	Reset Output	Double-click to export			
<input checked="" type="checkbox"/>		<b>hps_0</b>	Hard Processor System			multiple	mu
		memory	Conduit	memory			
		hps_io	Conduit	hps_io			
		h2f_reset	Reset Output	Double-click to export			
		h2f_axi_clock	Clock Input	Double-click to export	clk_0		
		h2f_axi_master	AXI Master	Double-click to export	[h2f_axi_c...		
		f2h_axi_clock	Clock Input	Double-click to export	clk_0		
		f2h_axi_slave	AXI Slave	Double-click to export	[f2h_axi_c...		
		h2f_lw_axi_clock	Clock Input	Double-click to export	clk_0		
		h2f_lw_axi_master	AXI Master	Double-click to export	[h2f_lw_a...		
<input checked="" type="checkbox"/>		<b>master_0</b>	JTAG to Avalon Master Bridge				
		clk	Clock Input	Double-click to export	clk_0		
		clk_reset	Reset Input	Double-click to export			
		master	Avalon Memory Mapped Master	Double-click to export	[clk]		
		master_reset	Reset Output	Double-click to export			
<input checked="" type="checkbox"/>		<b>evolisa_0</b>	evolisa				
		avalon_slave_0	Avalon Memory Mapped Slave	Double-click to export	[clock_sink]	0x0000_0000	0x0007_ffff
		conduit_end	Conduit	vga			
		reset_sink	Reset Input	Double-click to export	[clock_sink]		
		clock_sink	Clock Input	Double-click to export	clk_0		

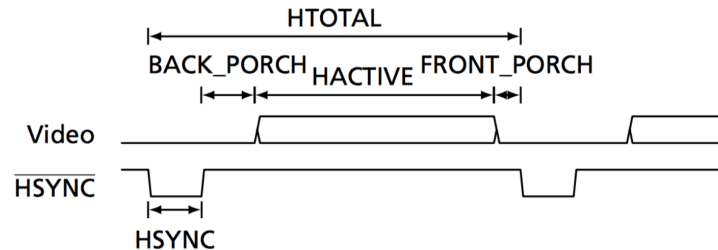
### VGA

The VGA controller generates the VGA signals to control VGA display, just like what we did in lab3. In this module, we need to generate clock signals such as VGA\_CLK, achieve display in any coordinates of the screen and control color values for each pixel. This module should communicate with user through Linux device driver, so the user can control the coordinates, shape and color displaying on the VGA screen.



More specifically, the image display is handled in hardware by display.sv and vga\_emulator.sv. There are two images to be displayed in the screen. The left image is the best generation circles of total 100 generations in one evolution. The right image is the source image. The size of each image is 200\*300 pixels. The position of left image is (80, 90) and the position of right image is (360, 90). The vga\_emulator module change coordinate (x,y) to next pixel on the rising edge of each clock cycle. The display module checks if this coordinate is in the field of one of these images and if so, detect which image field this coordinate belongs to. If this coordinate is in the left image field, the display module accesses left\_ram to get RGB values of this coordinate. Since the address in the ram is linear and the coordinate is two dimension. The

address is decided by the equation  $vga\_read\_addr = (x - XL) * DIMY + (y - YL)$ , where (XL, YL) is the first point of the left field and DIMY is the height of the field. The process of right image is the same. Then vga\_emulator module displays correct RGB values of this pixel. The timing issue of vga\_emulator is shown as follow.



## Draw Circle

### 1. Overview

In our project, we achieve Genetic Algorithm by mutated circles padding, which consumes a lot of time and will be implemented on the hardware.

The function of draw-circle module is to write the information of generated circles onto the RAM in FPGA, which will be used to compare with the target image in Fitness module. In this module, we need to assign the RGB values of each circle to the correct address of RAM, and achieve the circle overlap with opacity of circles.

The input of this module is all the information about a circle sent from software, including the location (coordinates of the center) and radius (one byte) of the circle, its RGB values (one byte each) and opacity (three bytes). These information are generated by the function of mutate in software and saved in a linked list. The output of this module is the address in this RAM where values will be changed and the corresponding values (RGB values) to write data into the RAM.

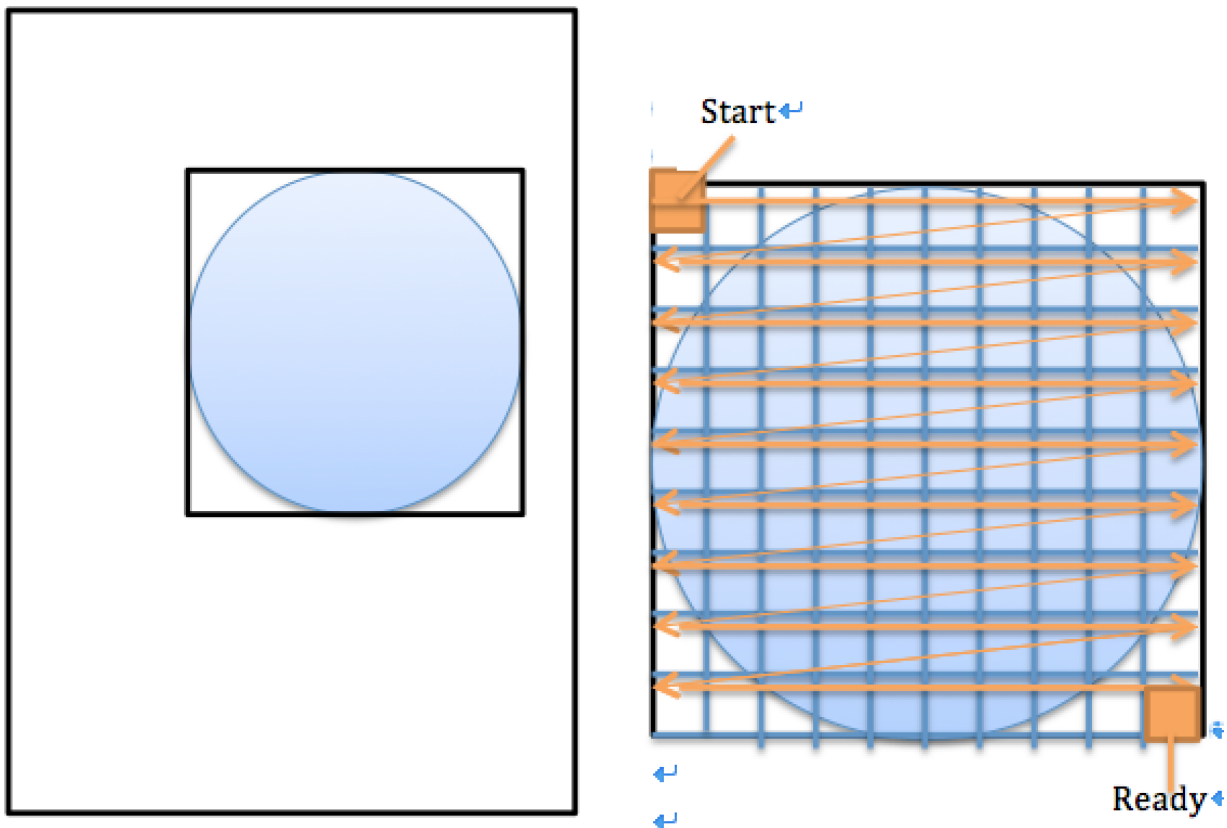
When a circle has been mutated in the software, the circle information will be sent to the draw-circle module along with the Start signal. And when a circle is finished, the module will return a Ready signal to software, which means that it is ready to draw next circle.

### 2. Design

To achieve padding circles on a RAM, we need to do as follows:

- 1) Locate the center of circle on the corresponding address in the RAM using the x, y coordinates of center of circle sent from software.
- 2) Judge whether other addresses in the RAM are in the circle or not using the radius.
- 3) For those addresses in the circle, overlap the circle on the background. That is, add the new RGB values of this pixel to its original value with the ratio defined by opacity.

To locate the circle in the RAM, we use Circle Equation to judge whether an address is in circle or not, which is the simplest way to draw circle. But there is no need to judge all the addresses because we already know the radius and center of circle, and can define a boundary for calculating. So in this module, we only judge whether the address is in the circle in a district of square, which circumscribes the circle, as shown below. Meanwhile read and write data in these addresses one by one.



The whole function of drawing circle is designed using Finite-State Machine. When received Start signal, the coordinates should be located at the up-left corner of the square, then move right to the next address. For each address, we need to judge whether it is in circle. If in circle, read old RGB values from that address and calculate the new ones with the input RGB and opacity of the circle. When getting to the end of line, the coordinates should go to the next line and start from the left. Finally when

having gone through all the addresses in this square, a Ready signal will be generated. The whole procedure is shown above.

So roughly there should be six states: before start (s\_start), start of line (s\_sol), mid of line and read (s\_molr), mid of line and write (s\_molw), end of line (s\_eol), done (s\_done).

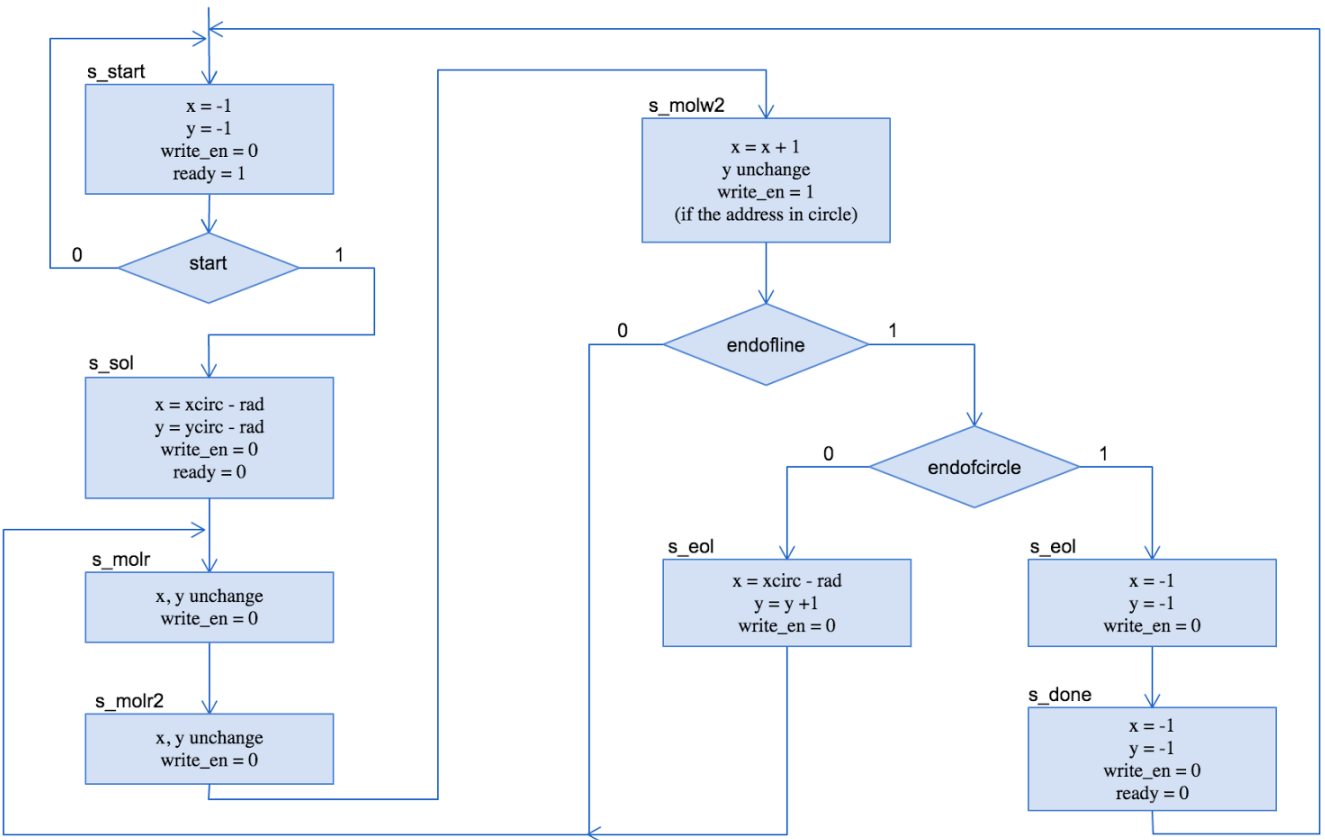
### 3. Basic Implementation

According to the design, we add another two registers x, y to point to the address now reading or writing. For each line in the square, let x add 1 when skipping to the next address, and y add 1 when skipping to the next line. We need total three clock cycles to achieve reading and writing to one address. The first cycle to locate to the exact address, the second one to read values from that address and the third one to write the new values calculated. Thus, we actually need two cycles for read in mid of line.

Along with the two reading cycles, when the address (x, y coordinates) is stable,  $x^2$ ,  $y^2$  and square of radius are calculated and compared to judge whether the position belongs to the circle. If the address is in circle, the write\_en signal will be set to 1, then enable writing. After reading from the address, the new RGB values can be calculated by multiplying RGB values of the new circle with opacity and add to old values, which can be expressed by this formula.

New RGB values = circle RGBs \* opacity + old RGBs from RAM \*(1 - opacity)

Our specific algorithm is shown below.



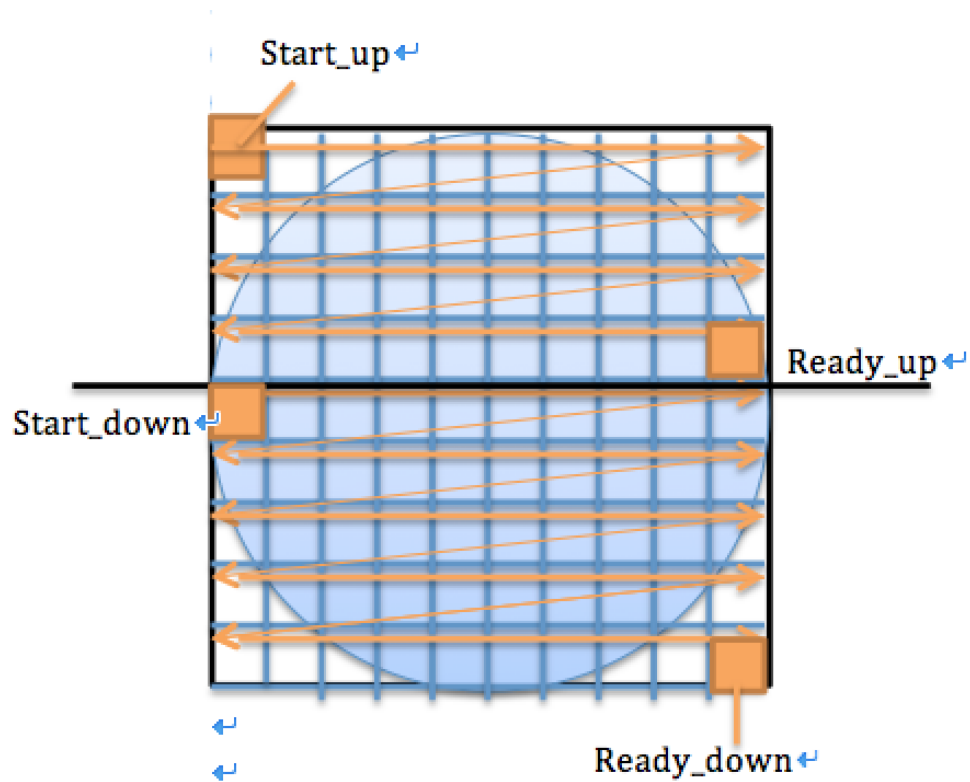
#### 4. Optimization

To speed up the procedure of drawing circle for better performance, we use dual-port RAM to draw the upside and downside of the circle at the same time.

Here we made a little change to our basic implementation. We copy the original “drawcircle” module to two modules “drawup” and “drawdown”, and change the condition of the start and done for each module.

For module “drawup”, when start, x, y will point to the upon left corner of the square, and when finish drawing the upside of the circle, a signal “readyup” will be sent back. For module “drawdown”, y will start from the y coordinate of the center of the circle, and end at the bottom right corner, also a signal “readydown” will be generated. When both parts send back ready signal, a ready signal will be sent from the upper level “drawcircle” to software.

With this optimization, the speed of drawing circle on the RAM is much higher than before.



## 5. Further Improvement

The time for drawing circle can be even shortened by using Bresenham Algorithm of drawing circle. This algorithm utilizing the geometric symmetry of a circle, can decide which pixel belongs to this circle. The speed of this method is very fast, and will save the time wasted for those pixels that don't belong to the circle in our implementation. This can be a direction for our furthermore optimization.

### Clean RAM

This function is also achieved by Finite-state machine, but only two states. One state is the default state with no operation, and the other is the clean state.

The clean operation is actually a writing operation that writes zeros into all the addresses in a RAM. So when the module receives the Start signal, it will go to the clean state, which will start from the first bit of the RAM and swipe till the last bit, set the write\_enable to 1 and write all 0s for RGB values in RAM.

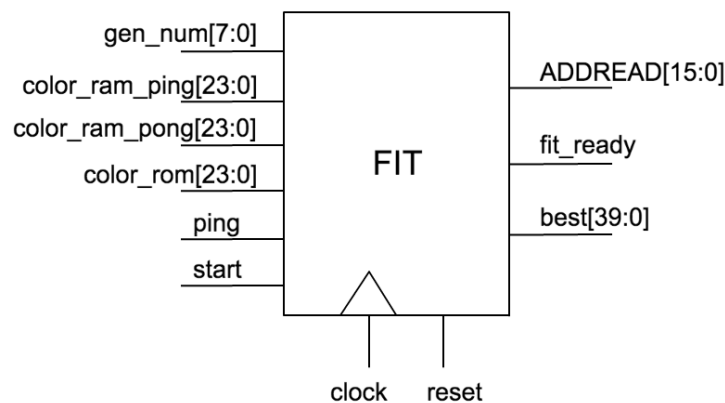
## Fit Module

The Fit module aims at implementing the Fitness function in the original code into hardware. The C code of this function is shown in Figure 1.

```
/*
 * A fitness function to compare between 2 images.
 */
int fitness(int* im, int objcount, int* targ, int size){
    unsigned int diff = 0;
    int i;
    int r,g,b;
    for (i=0; i<size; ++i){
        r = ((targ[i]>>16) % 256) - ((im[i]>>16) % 256);
        g = ((targ[i]>>8) % 256) - ((im[i]>>8) % 256);
        b = (targ[i] % 256) - (im[i] % 256);
        diff += r*r + g*g + b*b;
    }
    return diff; // * (1+objcount/100);
}
```

### 1. Basic implementation

The Module block diagram is shown in figure 2.



**Figure 2 Fit Block Diagram**

The function of Fitness is to compare the difference of two pictures. Since the pixel value consists of R, G, B, the difference is calculated by these three difference summing up together. In the C code, because a pixel value is saved as int, which is a 32-bit data type, a shift operation is used to get different part of the pixel.

A similar idea is applied in the hardware design. The basic operation is to calculate the difference of each pixel and sum up together. The following code is corresponding to this operation in the original C code, in which color\_ram\_ping is a 24-bit output from the padded RAM and color\_rom is another 24-bit output from the RAM which saved the original image.

```
diff_r_ping = (color_ram_ping[7:0] - color_rom[7:0])*(color_ram_ping[7:0] - color_rom[7:0]);
diff_g_ping = (color_ram_ping[15:8] - color_rom[15:8])*(color_ram_ping[15:8] - color_rom[15:8]);
diff_b_ping = (color_ram_ping[23:16] - color_rom[23:16])*(color_ram_ping[23:16] - color_rom[23:16]);
```

```
diff <= diff + diff_b_ping + diff_g_ping +diff_r_ping;
```



We set the logic signal 'diff' as a 32-bit signal although in the extreme case, the difference of two 200x300 images should overflow. We set this length because in our real case, we believe this length is reasonable to represent the difference cases and we don't need to spend more hardware resources on this.

The main logic in the hardware fit module is how to traverse the whole picture, read out the data from the two rams and calculate the difference then add it to diff. I use two parameters, x and y to represent the coordinates of the current pixel. So that x and y indicate the index of the current in the image array. We can use these two parameters to get the read address of the RAM, which is shown below:

$$\text{addread} = x * \text{DIMY} + y;$$

The addread is a 16-bit signal, corresponding to the input address of the RAM.

In order to get the difference of the two images, the fit module is also expected to know what is the best one in a single generation. And it need to return the index of the optimal image to the software, so that the software can know which one should be chosen to be based on for the next generation.

This function is realized by using a 40-bit output signal best to save the smallest difference and the corresponding index num. best[39:32] is used to save the index number and best[31:0] is used to save the smallest difference. A intermediate 8-bit logic signal 'num' is used to track the index of the current image in one generation. Every time it finishes comparing one picture, it will comparing the diff with best[31:0], if smaller, save the num to best[39:32] and diff to best[31:0].

The input 'gen\_num' is used to tell the module how many images in a single generation. With this configuration information, the module is able to compare the whole generation and start over by itself.

At the very beginning, we planned to pipeline the Fit operation with the Drawcircle operation, so that a ping-pong RAM is needed. As a result, this module is designed with color\_ram\_ping and color\_ram\_pong as input. These two signals are the data read out by the ping-pong RAM and a 1-bit input ping indicates whether the module need to compare the ping RAM with original image or the pong RAM with the original one.

## 2. Challenge Issue

After I first finished this implementation, I use the software testbench to test the function correctness. While some bugs happened:

(1) If there's a difference in the first pixel of two images, the result difference will be calculated 3 times as output.

(2) If there's difference in the last two pixels of the two images, the result achieved by the module is not able to detect this difference.

While given the test result that the module can catch the difference in other pixel positions well, we think that the problem may due to the timing issue.

I re-examine the code and find that I neglect a fact that once the address is provided to the RAM, we need to wait for a clock cycle to get the output data. But in my original implementation, I assumed that the data can be read out once the address is provided. So this leading to the fact that I recalculate the first pixel's difference and can not catch the final pixel's difference during the operation.

So I add another logic signal 'count' to separate the read-data and add-difference behavior into two clock cycle. Signal 'last' is also used to solve this problem by extending one clock cycle for the last pixel to finish the operation.

### 3. Improvement

Actually this module is able to make two improvement:

- a. Optimize the difference calculation by using addition instead of multiplication. This can be easily realized by adding an 'if' sentence to compare the magnitude of the two pixel values and use the bigger one subtracting the smaller one. Once we replace the multiplication with addition, the clock period can be expected to be shorter so that the whole performance can be improved.
- b. Another improvement is realizing parallelism in the whole operation. We can set the dimension of the block to half size of the image. Two fit block can operate in parallel to calculate the difference of the top half and the below half of the image separately. Using a top module to collect the difference result, adding them together to get the final difference.

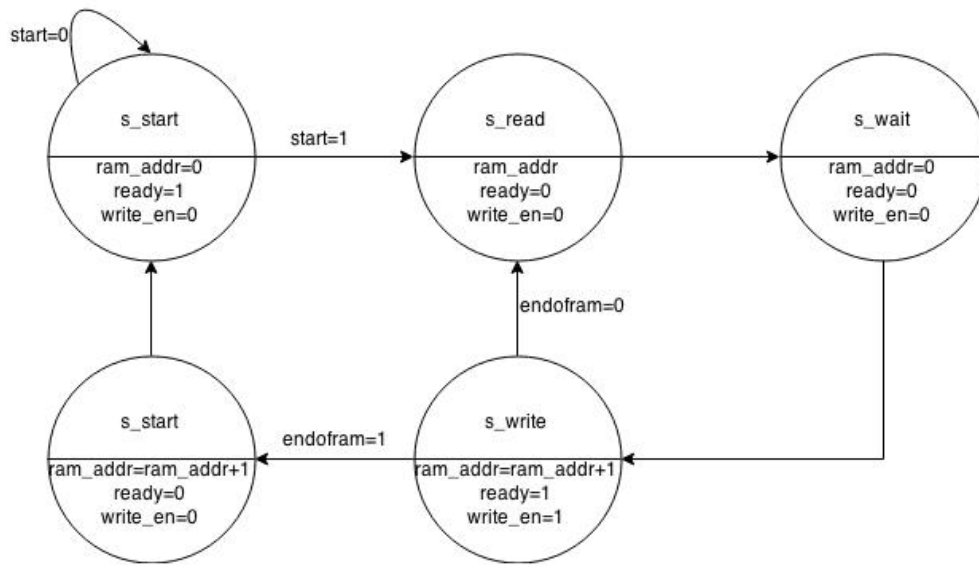
For improvement a, since in the whole hardware system, the multiplication is needed in Drawcircle module, so the the clock frequency can not be improved just by replacing the multiplication in this module. So we keep these multiplication to make it agree with the original algorithm.

As for the improvement b, unfortunately, due to the hardware limitation, we can not apply this to the implementation either. Detail challenge will be discussed in the Challenges part.

### **Pad Module**

Pad module is used to copy the best generation's data out of 100 generations to left\_ram to display it on the screen. When fit module finishes and passes the number of best generation to software, software controls pad module to read the data in ram\_ping and write it into the left\_ram.

The pad module is a Moore machine. There are five states, {s\_start, s\_read, s\_wait, s\_write, s\_done}. Signal start is controlled by software. When start is asserted to 1, pad module starts to copy data. It takes three clocks to copy one item of data. The first clock is to read data from source ram, ram\_ping. Wait for one clock to complete read process, and the third clock is to write date to destination ram, left\_ram. The number of address to read and write are the same. Signal endofram is asserted high when ram\_addr equals 16'd59999, the last useful address in the ram. Details are shown in the following FSM.



## Evolsmile

Evolsmile is the top module of the whole hardware system. In this module, it connects the 3 RAMs, drawcircle, fit, pad, cleanram and display together. At the same time, it defines the interface with the software.

The block diagram of Evolsmile is shown in Figure 3.

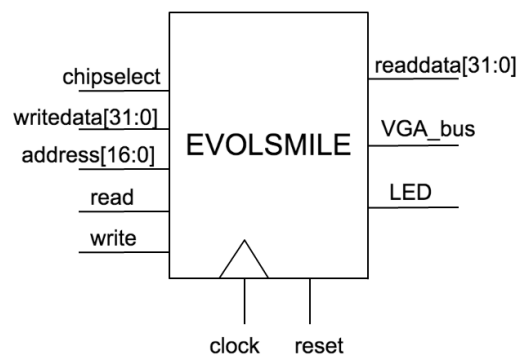


Figure 3

The whole module behaviors as a moore state machine, which have the following states:

```

typedef enum{s_draw_on_ping,
            s_clean_ping,
            s_cp_ping_disp,
            s_fit_ping,
            s_write_left,
            s_write_right,
            s_write_tmp} state;
  
```

At state `s_draw_on_ping`, `ram_ping`'s input ports will be connected to `drawcircle` and the generated circles will be written to this RAM.

At state `s_clean_ping`, `ram_ping`'s input will be connected to `cleanram` and the image in the ram will be flushed away.

At state `s_cp_ping_disp`, `ram_ping` and `left_ram` (which is used to display the optimal image) will be connected to `pad` and the image in the `ram_ping` will be copied to the `left_ram` and be displayed on the screen.

At state `s_fit_ping`, `ram_ping` and `right_ram` will be connected to `fit`, and their pixel difference will be calculated by `fit`.

At state `s_write_right`, `right_ram` will be connected to directly to the software input and the software is able to write data into the ram.

State `s_write_left` and `s_write_tmp`, `left_ram` are similar to `s_write_right` and in these two states, software is able to write data into the `left_ram` and `ram_ping` directly.

Actually, when running the algorithm, some of the states may not be used at all. But we still need them to debug and they can give us the insight of what's going on in the hardware.

For the convenience of debugging, we light different LED to indicate the current state so that we can know which state the hardware is stuck at and this is helpful to locate the bug.

As for the interface with the software, we use input signal `read` and `write` to determine whether receiving data from the software or returning data back to software.

In order to indicate which state and signal software sending in, we use one extra bit on the address bus to realize this. That's why our address bus is 17-bit long.

Since the control signal of the function modules are rising-edge triggered or need a pulse to trigger, we need to realize a pulse control signal in top module and sends them to the corresponding units. So we keep the control logic low whenever there's no input command from software.

# Performance

Performance of the whole system is evaluated both by time consuming and memory consuming and is presented in comparison between hardware accelerated version and pure software version.

## Time consuming test by gprof

### Pure software version

Each sample counts as 0.01 seconds.

%	cumulative	self	self	total			
time	seconds	seconds	calls	ms/call	ms/call	name	
81.56	1.15	1.15	100	11.50	11.50	fitness	
10.64	1.30	0.15	100	1.50	1.50	allocateImage	
3.55	1.35	0.05	85815	0.00	0.00	resolveColor	
2.13	1.38	0.03	89	0.34	0.90	drawCircle	
0.71	1.39	0.01	124400	0.00	0.00	write_reg	
0.71	1.40	0.01	1	10.00	15.00	loadTarget	
0.71	1.41	0.01	1	10.00	11.80	redraw	
0.00	1.41	0.00	793	0.00	0.00	rnd	
0.00	1.41	0.00	193	0.00	0.00	countCircles	
0.00	1.41	0.00	103	0.00	0.00	cloneCircles	
0.00	1.41	0.00	100	0.00	0.90	mutate	
0.00	1.41	0.00	99	0.00	0.00	cloneImage	
0.00	1.41	0.00	99	0.00	0.00	freeCircles	
0.00	1.41	0.00	1	0.00	0.00	init	
0.00	1.41	0.00	1	0.00	5.00	writebest	
0.00	1.41	0.00	1	0.00	5.00	writetest	

### Accelerated version

%	cumulative	self	self	total			
time	seconds	seconds	calls	ms/call	ms/call	name	
40.00	0.02	0.02	339011	0.00	0.00	read_reg	
40.00	0.04	0.02	1	20.00	20.00	loadTarget	
20.00	0.05	0.01	102	0.10	0.16	clean	
0.00	0.05	0.00	60660	0.00	0.00	write_reg	
0.00	0.05	0.00	793	0.00	0.00	rnd	

0.00	0.05	0.00	102	0.00	0.16	redraw
0.00	0.05	0.00	100	0.00	0.00	cloneCircles
0.00	0.05	0.00	100	0.00	0.13	fit
0.00	0.05	0.00	100	0.00	0.00	mutate
0.00	0.05	0.00	99	0.00	0.00	freeCircles
0.00	0.05	0.00	93	0.00	0.00	countCircles
0.00	0.05	0.00	89	0.00	0.00	draw
0.00	0.05	0.00	1	0.00	0.19	copy
0.00	0.05	0.00	1	0.00	0.00	init
0.00	0.05	0.00	1	0.00	0.00	writeright

As can be shown, fit and drawcircle function is fast accelerated and read\_reg function appears consuming most of the time. This also indicates if all the states switching was handled from the internal of hardware, the time consume can be further minimalized.

For load target function we don't care how much time it takes because it runs only once.

## Memory analysis

### Pure software version

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM
1039	root	20	0	26384	24m	376	R	99.0	2.4

### Accelerated version

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM
1034	root	20	0	1676	828	348	R	94.0	0.1

As can be shown from the top command above, the total percent of memory using of the programme we running is much improved after using hardware acceleration.

1033752 is total memory in KB. After calculation, the accelerated version saved 23776KB namely almost 23Mb in total.

# Challenges

## Parallelism between Drawcircle and Fit

At the very beginning, we plan to realize parallelism between drawcircle and fit module. For this kind of parallelism, a pair of ping-pong RAM are needed when drawcircle is drawing the circle on ping and fit is comparing the pong with the original image at the same time.

While considering that we need two RAM to display two 200x300 images on the screen to examine the evolution process of the algorithm, we have not enough RAM space left to save another 2 200x300 images. Instead, we only have 1 RAM space left to do the operation.

## Fit Parallelizing

As we mentioned in the Fit part, although two fit module can operate in parallel to achieve higher performance, we are not able to realize this in our real implementation. This is due to the RAM port restriction.

There are two RAMs fit need to read from during operation. One is the RAM\_ping, in which saves our generated image. Another is the RAM which contains the original image. During operation, fit need to give the same address to these two ram and compare their output data. If we parallel the fit part and two RAM ports are needed for each RAM. However, in order to display our result on the screen, vga needs to keep reading the data from the RAM which contains the original image and display it on the screen. As a result, we have only one port left for this RAM that can be used by other modules.

## Reference

- [1] D. Abramson, P. Logothetis, A. Postula, and M. Randall, "Application Specific Computers for Combinatorial Optimization," in Australian Computer Architecture Conference. Sydney, Australia: Springer-Verlag, 1997, pp. 29–44.
- [2] S. Wakabayashi, T. Koidez, N. Toshiney, M. Yamaney and H. Uenoy: "Genetic Algorithm Accelerator GAA-II," Design Automation Conference, 2000. Proceedings of the ASP-DAC 2000. Asia and South Pacific, pp. 9-10 (June. 2000)
- [3] S. D. Scott, A. Samal and S. Seth: "HGA: A hardware-based genetic algorithm," Proc. ACM/SIGDA 3rd International Symposium on FPGA, pp.53–59 (1995).



# Source Code

## Software Code

### ball.h

```
#ifndef _BALL_H
#define _BALL_H

#include <linux/ioctl.h>

#define BALL_REG_CNT 65536*2

typedef struct {
    unsigned int addr; /* from 0 to 3 representing x,y,b,r */
    unsigned int reg_data; /* integer data comes in */
} ball_arg_t;

#define BALL_MAGIC 'q'

/* ioctls and their arguments */
#define BALL_WRITE_REG_IOW(BALL_MAGIC, 1, ball_arg_t *)
#define BALL_READ_REG_IOWR(BALL_MAGIC, 2, ball_arg_t *)

#endif
```

### ball.c

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
```

```

#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "ball.h"

#define DRIVER_NAME "ball"

/*
 * Information about our device
 */
struct ball_dev {
    struct resource res; /* Resource: our registers */
    void __iomem *virtbase; /* Where registers can be accessed in memory */
    unsigned int data[BALL_REG_CNT];
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
//x,y,b,r
//elliptical formular with a=1 b on nominator default 10
//each parameter counts for 2 add
static void write_reg(int addr,unsigned int data)
{
    iowrite32(data, dev.virtbase + addr*4);
    dev.data[addr] = data;
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */
static unsigned int read_reg(unsigned int addr)
{
    unsigned int data;
    data=ioread32(dev.virtbase+addr*4);
}

```

```

        return data;
    }
static long ball_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
    ball_arg_t balla;

    switch (cmd) {
    case BALL_WRITE_REG:
        if (copy_from_user(&balla, (ball_arg_t *) arg,
                           sizeof(ball_arg_t)))
            return -EACCES;
        if (balla.addr > BALL_REG_CNT)
            return -EINVAL;
        ///////////////////////////////////////////////////////////////////
        write_reg(balla.addr, balla.reg_data);
        break;

    case BALL_READ_REG:
        if (copy_from_user(&balla, (ball_arg_t *) arg,
                           sizeof(ball_arg_t)))
            return -EACCES;
        if (balla.addr > BALL_REG_CNT)
            return -EINVAL;
        balla.reg_data = read_reg(balla.addr);
        if (copy_to_user((ball_arg_t *) arg, &balla,
                          sizeof(ball_arg_t)))
            return -EACCES;
        break;

    default:
        return -EINVAL;
    }

    return 0;
}

```

```

/* The operations our device knows how to do */

```

```

static const struct file_operations ball_fops = {
    .owner      = THIS_MODULE,

```

```

        .unlocked_ioctl = ball_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice ball_misc_device = {
    .minor      = MISC_DYNAMIC_MINOR,
    .name       = DRIVER_NAME,
    .fops       = &ball_fops,
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init ball_probe(struct platform_device *pdev)
{
    int ret;

    ret = misc_register(&ball_misc_device);

    /* Get the address of our registers from the device tree */
    ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
    if (ret) {
        ret = -ENOENT;
        goto out_deregister;
    }

    /* Make sure we can use these registers */
    if (request_mem_region(dev.res.start, resource_size(&dev.res),
        DRIVER_NAME) == NULL) {
        ret = -EBUSY;
        goto out_deregister;
    }

    /* Arrange access to our registers */
    dev.virtbase = of_iomap(pdev->dev.of_node, 0);
    if (dev.virtbase == NULL) {
        ret = -ENOMEM;
        goto out_release_mem_region;
    }
}

```

```

    }
    //////////////////////////////////////
    /* Display a welcome message */

    return 0;

out_release_mem_region:
    release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
    misc_deregister(&ball_misc_device);
    return ret;
}

/* Clean-up code: release resources */
static int ball_remove(struct platform_device *pdev)
{
    iounmap(dev.virtbase);
    release_mem_region(dev.res.start, resource_size(&dev.res));
    misc_deregister(&ball_misc_device);
    return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id ball_of_match[] = {
    { .compatible = "altr,ball" },
    {}
};
MODULE_DEVICE_TABLE(of, ball_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver ball_driver = {
    .driver = {
        .name = DRIVER_NAME,
        .owner = THIS_MODULE,
        .of_match_table = of_match_ptr(ball_of_match),
    },
    .remove = __exit_p(ball_remove),
};

```

```

};

/* Called when the module is loaded: set things up */
static int __init ball_init(void)
{
    pr_info(DRIVER_NAME ": init\n");
    return platform_driver_probe(&ball_driver, ball_probe);
}

/* Called when the module is unloaded: release resources */
static void __exit ball_exit(void)
{
    platform_driver_unregister(&ball_driver);
    pr_info(DRIVER_NAME ": exit\n");
}

module_init(ball_init);
module_exit(ball_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Jihua Li Wenbei Yu");
MODULE_DESCRIPTION("BALL Emulator");

```

### **ball.mod.c**

```

#include <linux/module.h>
#include <linux/vermagic.h>
#include <linux/compiler.h>

MODULE_INFO(vermagic, VERMAGIC_STRING);

struct module __this_module
__attribute__((section(".gnu.linkonce.this_module"))) = {
    .name = KBUILD_MODNAME,
    .init = init_module,
#ifdef CONFIG_MODULE_UNLOAD
    .exit = cleanup_module,
#endif

```

```
        .arch = MODULE_ARCH_INIT,  
};  
  
static const char __module_depends[]  
__used  
__attribute__((section(".modinfo"))) =  
"depends=";  
  
MODULE_ALIAS("of:N*T*Caltr,ball*");
```

### test.h

```
#ifndef _TEST_H_  
#define _TEST_H_  
#define POPULATION 100  
#define DIM 200*300  
#define DIMX 200  
#define DIMY 300  
#define MIN(x,y) (x>y?y:x)  
  
#define GEN_NUM 0x10000  
#define CIRC_DIM 0X10001  
#define CIRC_OPA 0X10002  
#define CIRC_COLOR 0X10003  
#define DRAW 0X10004  
#define CLEAN 0X10005  
#define COPY 0x10006  
#define FIT 0x10007  
#define WRITE_LEFT 0x10008  
#define WRITE_RIGHT 0x10009  
  
#define FIT_DIFF 0  
#define BEST_GEN 1  
#define STATUS 2  
#define CIRC_START 3  
#define CLEAN_START 4  
#define CP_START 5
```

```
#define FIT_START 6

#define CIRC_READY    0X8
#define CLEAN_READY  0X4
#define CP_READY     0X2
#define FIT_READY    0x1

#endif
```

```
typedef struct circ {
    int x, y, rad, color;
    float opacity;
    struct circ* next;
} circle;
```

### **test.c**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "ball.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <math.h>
#include "test.h"

/*****FOR THE DRIVER*****/
int ball_fd;
```



```

void write_reg(int addr,int color)
{
    ball_arg_t balla;
    balla.addr = addr;
    balla.reg_data = color;
    if (ioctl(ball_fd, BALL_WRITE_REG, &balla)) {
        printf("ioctl(ball_WRITE_REG) failed");
        return;
    }
}

unsigned int read_reg(int addr)
{
    ball_arg_t balla;
    balla.addr=addr;
    ioctl(ball_fd,BALL_READ_REG,&balla);
    return balla.reg_data;
}

void init()
{
    static const char filename[] = "/dev/ball";
    if ( (ball_fd = open(filename, O_RDWR)) == -1) {
        fprintf(stderr, "could not open %s\n", filename);
    }
}

/*****FOR FUNC WE DEFINED*****/

void clean(){
    write_reg(0x10005,0);
    read_reg(4);
    while(!(read_reg(2)&0x00000004)){};
}

void copy(){
    while(!(read_reg(2)&0x00000008)){};
    write_reg(0x10006,0);
    read_reg(5);
    while(!(read_reg(2)&0x00000002)){};
}

```

```

void fit(){
    write_reg(0x10007,0);
    write_reg(0x10000,100);
    read_reg(6);
    while(!(read_reg(2)&0x00000001)){};
}
//load data from local file mldata
void writerright(int* data){
    int i;
    write_reg(0x10009,0);
    for(i=0;i<60000;i++)
        write_reg(i,data[i]);
}
int* loadTarget(){
    void* fp = fopen("mldata", "r");
    int i=0;
    int j=0;
    char x;
    int* data = malloc(DIM * sizeof(int));
    for(i=0;i<DIM;i++) {
        for(j=0;j<4;j++){
            fscanf(fp,"%c",&x);
            *(data+i)|=(0x000000ff&x);
            if(j!=3)
                *(data+i)<<=8;
        }
    }
    fclose(fp);
    return data;
}
int countCircles(circle* e){
    if (e == NULL)
        return 0;
    return countCircles(e->next) + 1;
}

int rnd(int max){
    return rand() % max + 1;
}

```

```

}
void draw (int x, int y, int r, int opacity, int color){
    while(!(read_reg(2)&0x00000008)){};
    write_reg(0x10004,0);
    write_reg(0x10001,((x<<16)+(y<<7)+r));
    write_reg(0x10002,opacity);
    write_reg(0x10003,color);
    read_reg(3);
}
void redraw(circle *c){
    circle* f;
    clean();
    for(f=c;f!=NULL;f=f->next){
        draw(f->x,f->y,f->rad,f->opacity,f->color);
    }
}
circle* mutate( circle* c){
    if (rnd(2) > 1){
        /* Add Random Circle */
        circle* d = (circle *) malloc(sizeof(circle));
        d->x = rnd(DIMX);
        d->y = rnd(DIMY);
        d->color = rnd(0xffffffff);
        d->rad = rnd(50);
        d->opacity = rnd(0x555555);
        d->next = NULL;

        // Put on end so doesn't screw prev image
        if (c!=NULL){
            circle* f;
            circle* p;
            for(f=c;f!=NULL;f = f->next){
                p=f;
            }
            p->next = d;
        }else{
            c = d;
            c->next = NULL;
        }
    }
}

```

```

draw(d->x,d->y,d->rad,d->opacity,d->color);

}else{
    /* Delete Random Circle */
    circle* f = c;
    int num = countCircles(c)-1;

    if (num >1){
        int ind = rand() % num;
        circle* old;

        if (ind==0){
            old = c;
            c = c->next;

        }else{

            int i;
            for (i=0; i<ind; i++){
                f = f->next;
            }

            old = (circle*) f->next;
            f->next = f->next->next;

        }
        old->next =NULL;
        free(old);
        redraw(c);

    }else{
        if(c!=NULL){
            free(c);
        }
        c = NULL;
    }
}
if (rnd(2)>1)
    c = mutate( c);
return c;

```

```

}

void freeCircles(circle* e){
    if (e != NULL){
        if (e->next){
            freeCircles(e->next);
        }
        e->next = NULL;
        free(e);
    }
}

circle* cloneCircles(circle* e){
    if (e == NULL){
        return NULL;
    }

    circle* d = (circle *) malloc(sizeof(circle));
    d->x = e->x;
    d->y = e->y;
    d->color = e->color;
    d->rad = e->rad;
    d->opacity =e->opacity;

    if (e->next != NULL){
        d->next = cloneCircles(e->next);
    }else{
        d->next = NULL;
    }
    return d;
}

int main(){

    srand(1);
    init();
    int i;
    unsigned int min_ind = 0;
    unsigned int min=0xffffffff;
    int* dat = loadTarget();

```

```

//write right plane
writeright(dat);
circle** data = malloc(POPULATION*sizeof(circle*));
for(i=0;i<POPULATION; i++){
    data[i] = NULL;
}
unsigned int prev = min;
circle* best = NULL;
int itrs =0;
    min=read_reg(0);
    min_ind=read_reg(1);
    printf("inner:%d,%d\n",min,min_ind);
while (1){
    for(i =0; i<POPULATION; i++){
        redraw(data[i]);
        data[i] = mutate(data[i]);
        fit();
    }
    min=read_reg(0);
    min_ind=read_reg(1);
    printf("min=%d,min_ind=%d\n",min,min_ind);
    //sleep(1);
    best=cloneCircles(data[min_ind]);

    if (min<prev){
        printf("new best: min=%d - %d circles\n", min,
countCircles(data[min_ind]));
        prev = min;
        redraw(data[min_ind]);
        copy();
    }
    //print every tem generations completed
    if(itrs%10==0){
        printf("\t->%d, %d \n" , itrs, prev);
    }
    //free all bad children and copy the best to each child and redo mutation on this
basis
    for(i =0; i<POPULATION; i++){

```

```

        if (i != min_ind){
            freeCircles(data[i]);
            data[i] = cloneCircles(best);
        }
    }
    itrns ++;
}
return 0;
}

```

## Hardware Code

### vga\_emulator.sv

```

module vga_emulator(
input logic      clk50, reset,
input logic [23:0]  color,
output logic  [9:0]  x,y,

output logic [7:0]  VGA_R, VGA_G, VGA_B,
output logic      VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n,
VGA_SYNC_n
);
//change to next pixel on rising edge of clock
/*
* 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
*
*HCOUNT 1599 0      1279      1599 0
*
* _____| Video | _____| Video
*
*
* |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
*
* _____| _____|
* |____|   VGA_HS   |____|
*/

parameter HACTIVE = 11'd 1280,

```

```

    HFRONT_PORCH = 11'd 32,
    HSYNC       = 11'd 192,
    HBACK_PORCH = 11'd 96,
    HTOTAL      = HACTIVE + HFRONT_PORCH + HSYNC + HBACK_PORCH;
//1600

parameter VACTIVE   = 10'd 480,
    VFRONT_PORCH = 10'd 10,
    VSYNC       = 10'd 2,
    VBACK_PORCH = 10'd 33,
    VTOTAL      = VACTIVE + VFRONT_PORCH + VSYNC + VBACK_PORCH; //525

logic [10:0]          hcount; // Horizontal counter
logic                endOfLine;

    /***/

always_ff @(posedge clk50 or posedge reset)
    if (reset)      hcount <= 0;
    else if (endOfLine) hcount <= 0;
    else            hcount <= hcount + 11'd 1;

assign endOfLine = hcount == HTOTAL - 1;

// Vertical counter
logic [9:0]          vcount;
logic                endOfField;

always_ff @(posedge clk50 or posedge reset)
    if (reset)      vcount <= 0;
    else if (endOfLine)
        if (endOfField) vcount <= 0;
    else            vcount <= vcount + 10'd 1;

assign endOfField = vcount == VTOTAL - 1;

// Horizontal sync: from 0x520 to 0x57F
// 101 0010 0000 to 101 0111 1111

```



```

assign VGA_HS = !( hcount[10:7] == 4'b1010) & (hcount[6] | hcount[5]);
assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) >> 1);
//modified here

assign VGA_SYNC_n = 1; // For adding sync to video signals; not used for VGA

// Horizontal active: 0 to 1279   Vertical active: 0 to 479
// 101 0000 0000 1280           01 1110 0000 480
// 110 0011 1111 1599           10 0000 1100 524
assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
                    !( vcount[9] | (vcount[8:5] == 4'b1111) );

assign VGA_CLK = hcount[0]; // 25 MHz clock: pixel latched on rising edge

```

```

assign VGA_R=color[23:16];
assign VGA_G=color[15:8];
assign VGA_B=color[7:0];

```

```

always_comb begin
    if(hcount>11'd1279)
        x=10'd0;
    else
        x=hcount[10:1];
    if(vcount>10'd479)
        y=10'd0;
    else
        y=vcount;
end
endmodule // VGA_LED_Emulator

```

### display.sv

```

module display(input logic    clock, reset,
               input logic [23:0] color_left,
               input logic [23:0] color_right,

```

```

        output logic [15:0] vga_read_addr,
        output logic [7:0] VGA_R, VGA_G, VGA_B,
        output logic      VGA_CLK, VGA_HS, VGA_VS,
VGA_BLANK_n,
        output logic      VGA_SYNC_n);

//dimention of every single block of picture to display, maxium dim to be 65535
    parameter DIMX=200;
    parameter DIMY=300;

//position of left picture and right picture
    parameter XL=80;
    parameter XR=360;
    parameter YL=90;
    parameter YR=90;

    logic [9:0] x,y;
    logic [23:0] color;
    vga_emulator vga(.clk50(clock), .*);

always_comb begin
    //display of left picture
        if(x>=XL && x<DIMX+XL && y>=YL && y<DIMY+YL) begin
            color=color_left;
            vga_read_addr=(x-XL)*DIMY+(y-YL);
        end
        else if(x>=XR && x<DIMX+XR && y>=YR && y<DIMY+YR) begin
            color=color_right;
            vga_read_addr=(x-XR)*DIMY+(y-YR);
        end
        else begin
            color=24'h00ffff;
            vga_read_addr=16'd0;
        end
    end
end

endmodule

```

## drawcircle.sv

```
//using pipeline methodology
//circle drawing when in mutate function, draw circle when excuting next circle generation untill
all circles drawn
//need a bit for ram clear

module drawcircle(input logic    clock,                //should also drive writing port of the
ra,
                input logic    reset,
                //200*311 need x,y bit width of 9, radius maxium
fixed to be 128 7 bits, color 24 bits, opacity 24 bits, 3*24 bits in total
                input logic signed [9:0] xcirc,ycirc,
//one extra for sign
                input logic signed [7:0] rad,
                input logic    [23:0] opacity,
                //one extra for sign
                input logic    [23:0] color,
                //color out put
                input logic    [23:0] circ_color_read_up,
                input logic    [23:0]
circ_color_read_down,
                input logic    start,
                output logic    [23:0] circ_color_out_up,
                output logic    [23:0] circ_color_out_down,
                output logic    [15:0] circ_address_up,
                output logic    [15:0] circ_address_down,
                output logic    ready,
                output logic    circ_write_en_up,
                output logic    circ_write_en_down
);

/*****PARAMETERS*****/
*****/
```

```

//dimention of every single block of picture to display, maxium dim to be 65535
    parameter DIMX=200;
    parameter DIMY=300;
    wire ready_up,ready_down;
    assign ready=ready_up&ready_down;

//updating fsm state
drawup      up(*,
                .circ_color_read(circ_color_read_up),
                .circ_color_out(circ_color_out_up),
                .circ_address(circ_address_up),
                .circ_write_en(circ_write_en_up),
                .ready(ready_up));

drawdown    down(*,
                .circ_color_read(circ_color_read_down),
                .circ_color_out(circ_color_out_down),
                .circ_address(circ_address_down),
                .circ_write_en(circ_write_en_down),
                .ready(ready_down));

endmodule

```

### **Fit.sv**

```

// This module is responsible for comparing the padded picture with the original picture, choose
the best one
//implemented by reading fixed number of ramdata from both ram and compare a difference and
accumulate with the last row.
//also pipeline methodology, when circles padded, start fitting while next circle is drawing.
module fit( input logic clock, reset,
            input logic [7:0]gen_num,
            input logic [23:0] color_ram_ping,
            input logic [23:0] color_ram_pong,
            input logic [23:0] color_rom,
            input logic ping, // decide which ram to compare
            input logic start,

            output logic [15:0] addread,
            output logic fit_ready,

```

```
output logic[39:0] best);
```

```
//dimention of every single block of picture to display, maxium dim to be 65535
```

```
parameter DIMX=200;
```

```
parameter DIMY=300;
```

```
logic [31:0] diff;
```

```
logic [9:0] x=0,y=0;
```

```
logic [7:0] num=0;
```

```
logic [31:0] diff_r_ping;
```

```
logic [31:0] diff_g_ping;
```

```
logic [31:0] diff_b_ping;
```

```
logic [31:0] diff_r_pong;
```

```
logic [31:0] diff_g_pong;
```

```
logic [31:0] diff_b_pong;
```

```
logic fit=0; // indicate the ram can be used to fit
```

```
logic pad; // indicate the best is found and can be pad to the display ram
```

```
logic [1:0] last=0;
```

```
logic [7:0] num_max;
```

```
logic count=0;
```

```
always_ff @(posedge clock or posedge reset) begin
```

```
    //display of left picture
```

```
        if (reset) begin
```

```
            best <= 40'hfffffffff;
```

```
            diff <= 32'd 0;
```

```
            x <= 10'd 0;
```

```
            y <= 10'd 0;
```

```
            num <= 8'd 0;
```

```
            fit <= 1'd 0;
```

```
            fit_ready <= 1'd 1;
```

```
            last <=0;
```

```
            count <=0;
```

```
            num_max <= 0;
```

```
        end
```

```
        else if (start) begin
```

```
            if (num == num_max) begin
```

```
                fit <=1'd 1;
```

```
                fit_ready <= 1'd 0;
```

```

        diff <= 32'd 0;
        x <= 10'd 0;
        y <=10'd 0;
        last <=0;
        num_max <= gen_num;
        best <= 40'hffffffff;
        num <= 0;
        count <=0;
        end
else begin
    fit <=1'd 1;
    fit_ready <= 1'd 0;
    diff <= 32'd 0;
    x <= 10'd 0;
    y <=10'd 0;
    last <=0;
    count<=0;
    num_max <= gen_num;
    end

end

else if (fit == 1) begin
    if (count ==1) begin
        count <=0;
        if(y==(DIMY-1) && x == (DIMX-1)) begin
            y <= 10'd 0;
            x <= 10'd 0;
            last <=last+1;
        end
    end
    else if (x==0 && y==0 && last==1) begin
        num <= num +8'd 1;//num-1
        fit <= 0;
        fit_ready <= 1'd1;
        last <=0;
        if (diff < best[31:0]) begin
            best[31:0] <= diff;
            best[39:32] <= num;
            pad <=1;
        end
    end
end

```

```

        end
        else if (y == (DIMY-1)) begin
            y <= 10'd 0;
            x <= x + 10'd 1;
        end
        else begin
            y <= y + 10'd 1;
        end
    end
    if (count==0) begin
        if (ping) diff <= diff + diff_b_ping + diff_g_ping
+diff_r_ping;
        else diff <= diff + diff_b_pong + diff_g_pong +
diff_r_pong;
        count <=1;
    end
end
end
end

```

```

always_comb begin
    address = x * DIMY + y;
    diff_r_ping = (color_ram_ping[7:0] -
color_rom[7:0]) * (color_ram_ping[7:0] - color_rom[7:0]);
    diff_g_ping = (color_ram_ping[15:8] -
color_rom[15:8]) * (color_ram_ping[15:8] - color_rom[15:8]);
    diff_b_ping = (color_ram_ping[23:16] -
color_rom[23:16]) * (color_ram_ping[23:16] - color_rom[23:16]);
    diff_r_pong = (color_ram_pong[7:0] -
color_rom[7:0]) * (color_ram_pong[7:0] - color_rom[7:0]);
    diff_g_pong = (color_ram_pong[15:8] -
color_rom[15:8]) * (color_ram_pong[15:8] - color_rom[15:8]);
    diff_b_pong = (color_ram_pong[23:16] -
color_rom[23:16]) * (color_ram_pong[23:16] - color_rom[23:16]);

    end
endmodule

```

## pad.sv

```
module pad(  input logic clock, reset,
            input logic [23:0] source_ram_data,
            input logic  start,

            output logic [15:0] ram_addr,
            output logic write_en,
            output logic [23:0] dest_ram_data,
            output logic  ready);

//dimention of every single block of picture to display, maxium dim to be 65535
    parameter DIMX=200;
    parameter DIMY=300;

    logic endofram;

    typedef enum {s_start,s_read,s_wait,s_write,s_done} state;
//moore

    state current_state, next_state;

    logic [15:0] x,nx;

    always_ff @(posedge clock or posedge reset) begin
        if (reset) begin
            current_state<=s_start;
            x<=0;
        end
        else begin
            current_state <= next_state;
            x <= nx;
        end
    end

    always_comb begin
        case(current_state)
```



```

s_start: begin
    if (start)
        next_state = s_read;
    else
        next_state = s_start;
        nx=0;;
        ready=1;
        write_en=0;
    end
s_read: begin //just
start drawing line, put x,y pos in to states
    next_state = s_wait;
    nx = x;
    ready=0;
    write_en=0;
end
//////////
s_wait: begin
    next_state=s_write;
    nx=x;
    ready=0;
    write_en=0;
end
s_write: begin
    if(endofram)
        next_state= s_done;
    else
        next_state = s_read;
        nx=x+12'sd1;
        ready=0;
        write_en=1;
    end
s_done: begin
    next_state = s_start;
    nx=x+1;;
    ready=0;
    write_en=0;

```

```

end
endcase
end
assign endofram = (x==16'd59999);
assign ram_addr=x;
assign dest_ram_data=source_ram_data;
endmodule

```

### **cleanram.sv**

```

module cleanram(
    input logic clock, reset,
    input logic start,
    output logic [15:0] clean_addr,
    output logic write_en,
    output logic ready,
    output logic [23:0] clean_data);

```

//dimention of every single block of picture to display, maxium dim to be 65535

```
parameter DIMX=200;
```

```
parameter DIMY=300;
```

```
logic endofram;
```

```
typedef enum {s_start,s_clean} state; //moore
```

```
state current_state, next_state;
```

```
logic [15:0] x,nx;
```

```
always_ff @(posedge clock or posedge reset) begin
```

```
if (reset) begin
```

```
current_state<=s_start;
```

```
x<=16'd0;
```

```
end
```

```
else begin
```

```
current_state <= next_state;
```

```
x <= nx;
```

```

        end
    end

    always_comb begin
        case(current_state)
            s_start: begin
                if (start)
                    next_state = s_clean;

                else
                    next_state = s_start;
                    nx=16'd0;
                    ready=1;
            end
            s_clean: begin
                if(endofram)
                    next_state = s_start;
                else
                    next_state = s_clean;
                    nx=x+1;
                    ready=0;
                end
            endcase
        end
        assign write_en=1;
        assign clean_data=24'd0;
        assign clean_addr = x;
        assign endofram = (x==16'd59999);
    endmodule

```

### **evolsmile.sv**

```

module evolsmile(input logic    clock,
                 input logic    reset,
                 input logic [31:0] writedata,

```

```

        output logic [31:0]          readdata,
        input logic                 read,
        input logic                 write,
        input                       chipselect,
        input logic [16:0]          address,
        output logic [7:0]          VGA_R, VGA_G, VGA_B,
        output logic                VGA_CLK, VGA_HS, VGA_VS,
VGA_BLANK_n,
        output logic                VGA_SYNC_n,
        output logic [3:0]          LED
    );
    wire [15:0] ping_address_a, ping_address_b;
    wire [23:0] ping_data_a, ping_data_b, ping_q_a, ping_q_b;
    wire ping_wren_a, ping_wren_b ;
    ramlisa ram_ping( .clock(clock),
                    .address_a(ping_address_a),
                    //write/read address for drawcircle
                    .data_a(ping_data_a),
                    .wren_a(ping_wren_a),
                    .q_a(ping_q_a),
                    .address_b(ping_address_b),
                    .data_b(ping_data_b),
                    .wren_b(ping_wren_b),
                    .q_b(ping_q_b));
    wire [15:0] left_addr;
    wire [23:0] left_data, left_q;
    reg left_wren;
    ramlisa left_ram( .clock(clock),
                    .address_a(left_addr),
                    .data_a(left_data),
                    .wren_a(left_wren),
                    .q_a(left_q),
                    .address_b(vga_read_addr),
                    .data_b(0),
                    .wren_b(0),
                    .q_b(color_left));
    wire [15:0] right_addr;
    wire [23:0] right_data, right_q;
    reg right_wren;

```

```

ramlisa right_ram( .clock(clock),
                  .address_a(right_addr),
                  .data_a(right_data),
                  .wren_a(right_wren),
                  .q_a(right_q),
                  .address_b(vga_read_addr),
                  .data_b(0),
                  .wren_b(0),
                  .q_b(color_right));

logic signed [9:0]    xcirc,ycirc;
logic signed [7:0]    rad;
logic                [23:0] opacity;
logic                [23:0] color;
logic                circ_start,circ_ready;
wire                 circ_write_en_up,circ_write_en_down;
wire                 [15:0] circ_address_up,circ_address_down;
wire                 [23:0]
circ_color_out_up,circ_color_out_down,circ_color_read_up,circ_color_read_down;
drawcircle draw ( .*,
                  .xcirc(xcirc),
                  .ycirc(ycirc),
                  .rad(rad),
                  .start(circ_start),
                  .ready(circ_ready));

wire                 [23:0] cp_source_data;
wire                 [23:0] cp_dest_data;
wire                 [15:0] cp_ram_addr;
reg                  cp_start,cp_ready,cp_write_en;

pad                  copy( .*,
                  .source_ram_data(cp_source_data),
                  .start(cp_start),
                  .ram_addr(cp_ram_addr),
                  .write_en(cp_write_en),
                  .dest_ram_data(cp_dest_data),
                  .ready(cp_ready));

```

```

wire          [23:0] clean_data;
wire          [15:0] clean_addr;
reg
cleanram      clean(.*,
                                clean_start,clean_ready,clean_write_en;
                                .start(clean_start),
                                .clean_addr(clean_addr),
                                .write_en(clean_write_en),
                                .ready(clean_ready),
                                .clean_data(clean_data));

reg          [7:0] gen_num;
wire        [15:0] fit_addr;
reg
fit compare_ram(.*,
                                fit_ping,fit_ready,fit_start;
                                .gen_num(gen_num),
                                .color_ram_ping(ping_q_a),
                                .color_ram_pong(0),
                                .color_rom(right_q),
                                .ping(1), // decide which ram to compare
                                .start(fit_start),
                                .address(fit_addr),
                                .fit_ready(fit_ready),
                                .best(fit_best));

wire          [23:0] color_left,color_right;
wire          [15:0] vga_read_addr;
display disp(.*);

typedef enum{s_draw_on_ping,
            s_clean_ping,
            s_cp_ping_disp,
            s_fit_ping,
            s_write_left,
            s_write_right,
            s_write_tmp} state;

state current_state;

```

```

always_ff @(posedge clock or posedge reset) begin
    if(reset) begin
        current_state<=s_write_right;
        fit_ping<=0;
        gen_num<=200;
        circ_start<=0;
        clean_start<=0;
        cp_start<=0;
        fit_start<=0;
        xcirc<=100;
        ycirc<=100;
        opacity<=24'hffffff;
        color<=24'hff00ff;
        end
    else if(chipselect&&write)
        case(address)
            17'h10000 : gen_num<=writedata[7:0];
            17'h10001 : {xcirc[8:0],ycirc[8:0],rad[6:0]}<=writedata[24:0];
            17'h10002 : opacity<=writedata[23:0];
            17'h10003 : color<=writedata[23:0];
            17'h10004 : current_state<=s_draw_on_ping;
            17'h10005 : current_state<=s_clean_ping;
            17'h10006 : current_state<=s_cp_ping_disp;
            17'h10007 : current_state<=s_fit_ping;
            17'h10008 : current_state<=s_write_left;
            17'h10009 : current_state<=s_write_right;
            17'h1000A : current_state<=s_write_tmp;
        endcase
    else if(chipselect&&read)
        case(address)
            17'h0 : readdata<=fit_best[31:0];
            17'h1 : readdata<=fit_best[39:32];
            17'h2 :
readdata[3:0]<={circ_ready,clean_ready,cp_ready,fit_ready};
            17'h3 : circ_start<=1;
            17'h4 : clean_start<=1;
            17'h5 : cp_start<=1;
            17'h6 : fit_start<=1;
        endcase

```

```

else begin
    circ_start<=0;
    clean_start<=0;
    cp_start<=0;
    fit_start<=0;
end
end
always_comb begin
    case (current_state)
    s_draw_on_ping: begin
        ping_address_a=circ_address_up;
        ping_data_a=circ_color_out_up;
        ping_wren_a=circ_write_en_up;

        ping_address_b=circ_address_down;
        ping_data_b=circ_color_out_down;
        ping_wren_b=circ_write_en_down;

        left_addr=0;
        left_data=0;
        left_wren=0;

        right_addr=0;
        right_data=0;
        right_wren=0;

        circ_color_read_up=ping_q_a;
        circ_color_read_down=ping_q_b;
        cp_source_data=0;
        LED=0;
    end
    s_clean_ping: begin
        ping_address_a=clean_addr;
        ping_data_a=clean_data;
        ping_wren_a=clean_write_en;

        ping_address_b=0;
        ping_data_b=0;
        ping_wren_b=0;
    end
end

```



```

    left_addr=0;
    left_data=0;
    left_wren=0;

    right_addr=0;
    right_data=0;
    right_wren=0;

    circ_color_read_up=0;
    circ_color_read_down=0;
    cp_source_data=0;
    LED=1;
    end
s_cp_ping_disp: begin
    ping_address_a=cp_ram_addr;
    ping_data_a=0;
    ping_wren_a=0;

    ping_address_b=0;
    ping_data_b=0;
    ping_wren_b=0;

    left_addr=cp_ram_addr;
    left_data=cp_dest_data;
    left_wren=cp_write_en;

    right_addr=0;
    right_data=0;
    right_wren=0;

    circ_color_read_up=0;
    circ_color_read_down=0;
    cp_source_data=ping_q_a;
    LED=2;
    end
s_fit_ping: begin
    ping_address_a=fit_addr;
    ping_data_a=0;

```

```
ping_wren_a=0;
```

```
ping_address_b=0;
```

```
ping_data_b=0;
```

```
ping_wren_b=0;
```

```
left_addr=0;
```

```
left_data=0;
```

```
left_wren=0;
```

```
right_addr=fit_addr;
```

```
right_data=0;
```

```
right_wren=0;
```

```
circ_color_read_up=0;
```

```
circ_color_read_down=0;
```

```
cp_source_data=0;
```

```
LED=3;
```

```
end
```

```
s_write_left: begin
```

```
ping_address_a=0;
```

```
ping_data_a=0;
```

```
ping_wren_a=0;
```

```
ping_address_b=0;
```

```
ping_data_b=0;
```

```
ping_wren_b=0;
```

```
left_addr=address[15:0];
```

```
left_data=writedata[23:0];
```

```
left_wren=write;
```

```
right_addr=0;
```

```
right_data=0;
```

```
right_wren=0;
```

```
circ_color_read_up=0;
```

```
circ_color_read_down=0;
```

```

        cp_source_data=0;
        LED=4;
    end
s_write_right: begin
    ping_address_a=0;
    ping_data_a=0;
    ping_wren_a=0;

    ping_address_b=0;
    ping_data_b=0;
    ping_wren_b=0;

    left_addr=0;
    left_data=0;
    left_wren=0;

    right_addr=address[15:0];
    right_data=writedata[23:0];
    right_wren=write;

    circ_color_read_up=0;
    circ_color_read_down=0;
    cp_source_data=0;
    LED=5;
end
s_write_tmp: begin
    ping_address_a=address[15:0];
    ping_data_a=writedata[23:0];
    ping_wren_a=write;

    ping_address_b=0;
    ping_data_b=0;
    ping_wren_b=0;

    left_addr=0;
    left_data=0;
    left_wren=0;

    right_addr=0;

```

```
right_data=0;
right_wren=0;

circ_color_read_up=0;
circ_color_read_down=0;
cp_source_data=0;
LED=5;
end
endcase
end
endmodule
```