A Software and FPGA Based Smoke Simulator

CSEE 4840 Embedded System Design

Yun Fei
Jonathan Chang
Tianming Miao
Guanduo Li

5/15/2014

Table of Content

1. Project Introduction

Building animation tools for fluid-like motions is an important and challenging problem with many applications in computer graphics, but we want to challenge the FPGA Sockit Board to work with the numerous athematic. In this project, we are going to implement a smoke simulator in both hardware and software, more specifically, the system will generate a smoke that has a sequence of different states and it will exhibit natural-looking smoke-like behavior [1]. Moreover, the smoke will be in its initial state, and then diffuse towards the shape like real smoke. An overview of the project is shown in figure 1. Throughout the project, we implement floating point calculation in 38-bit fixed point for hardware design.

The algorithm will be based on Raanan Fattal and Dani Lischinski's paper published in 2004 [1]. The paper claims that typical smoke animations are generally created by the inviscid Euler equations:

$$u_t = -u \cdot \nabla u - \nabla p + f \quad (1)$$

$$\nabla u = 0 \quad (2)$$

In above equations, u_t is the smoke fluid velocity vector's temporal derivative. P is the hydrostatic pressure and f is the external forces. Equation (1) is basically Newton's second law. Although this describes smoke behavior well, it does not provide good control of it. The paper brings an improved algorithm based on the previous one and allows us to control the movement of smoke animation.

$$u_t = -u \cdot \nabla u - \nabla p + v_f F(\rho, \rho^*) - v_d u \quad (3)$$

In equation (3), we have a new term F, which is the driving force term to carry the smoke to the defined target density ρ^*. v_f and v_d are control parameters that tune the strength of the driving force and attenuate momentum.

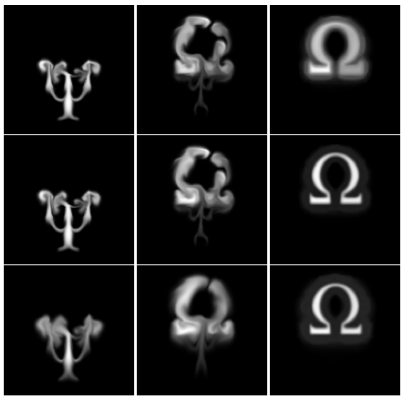

Figure 1: Simulation result

The Navier-Stokes Equation is our main functional mode and algorithm.

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} + v\nabla^2\mathbf{u} - \frac{1}{d}\nabla p + \mathbf{f}$$

The equation indicates four momentum conservation conditions:

a. Advection/Conversion process

b. Diffusion(damping) process

c. Pressure process

d. External force (gravity, etc.) process

(1) Add Force

The easiest term to solve is the addition of the external force f. If we assume that the force does not vary considerably during the time step, then:

$$w_1(x) = w_0(x) + \Delta t f(x,t)$$

is a good approximation of the effect of the force on the field over the time step $\triangle$t. In an interactive system this is a good approximation, since forces are only applied at the beginning of each time step.

(2) Advection

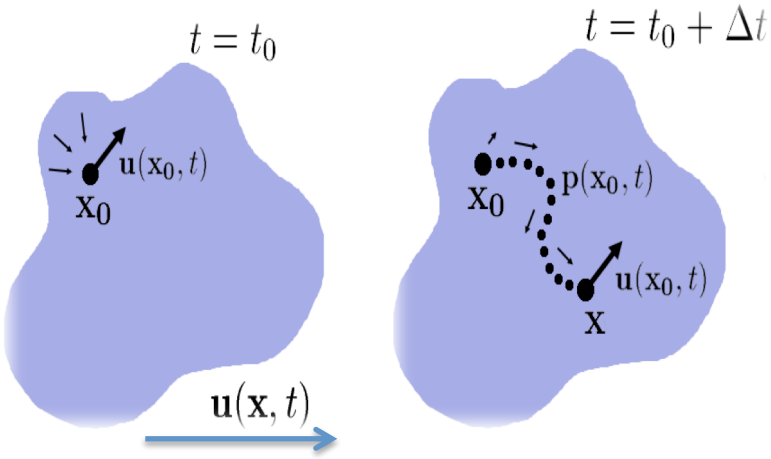

Figure2: Advection

With the pressure $p(x_0,t)$, after $\triangle t$, the position of x advects from x_0 to x, so there is some advected path when node x_0 is moving.

(3) Diffusion

The third step solves the effect of viscosity and is equivalent ot a diffusion equation: $\frac{\partial w_2}{\partial t} = \nu \nabla^2 w_2$. This is a standard equation for which many numerical procedures have been developed. The most straightforward way of solving this equation is to discretize the diffusion operator ∇^2 and then to do an explicit time step as Foster and Metaxas did.

(4) Projection step

The fourth step involves the projection step, which makes the resulting field divergence free. As pointed out in the previous subsection this involves the resolution of the Poisson Problem defined by $\nabla^2 q = \nabla w_3$; $w_4 = w_3 - \nabla q$

2. Design Flow

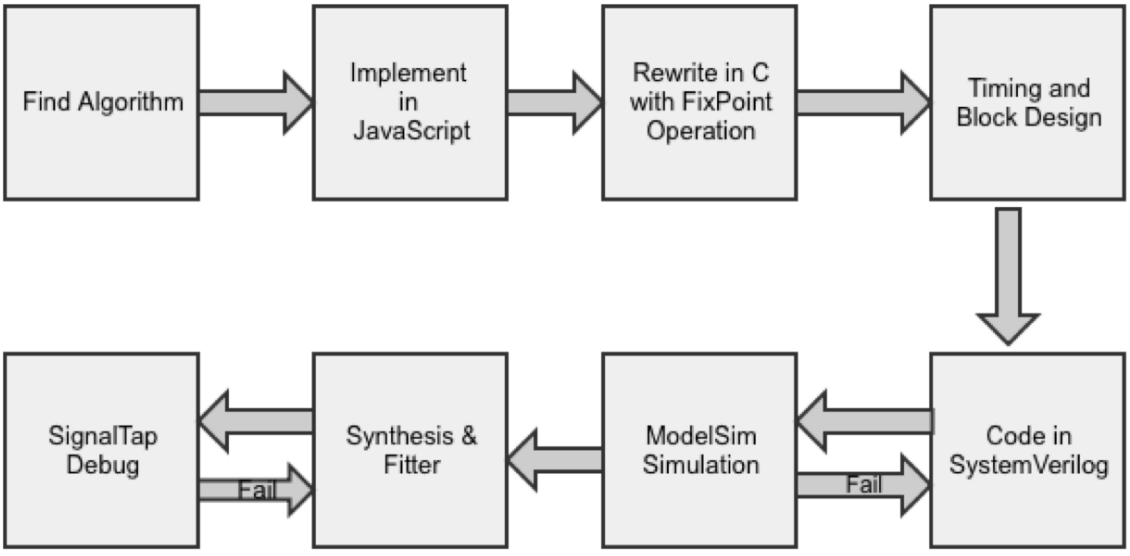

Figure 3: Design Flow

Based on the algorithm that has been discussed above, we implemented this smoke simulation in JavaScript and C++ (See code below). We did the timing and block design and then coded everything in SystemVerilog. We debugged the design with ModelSim and SignalTap.

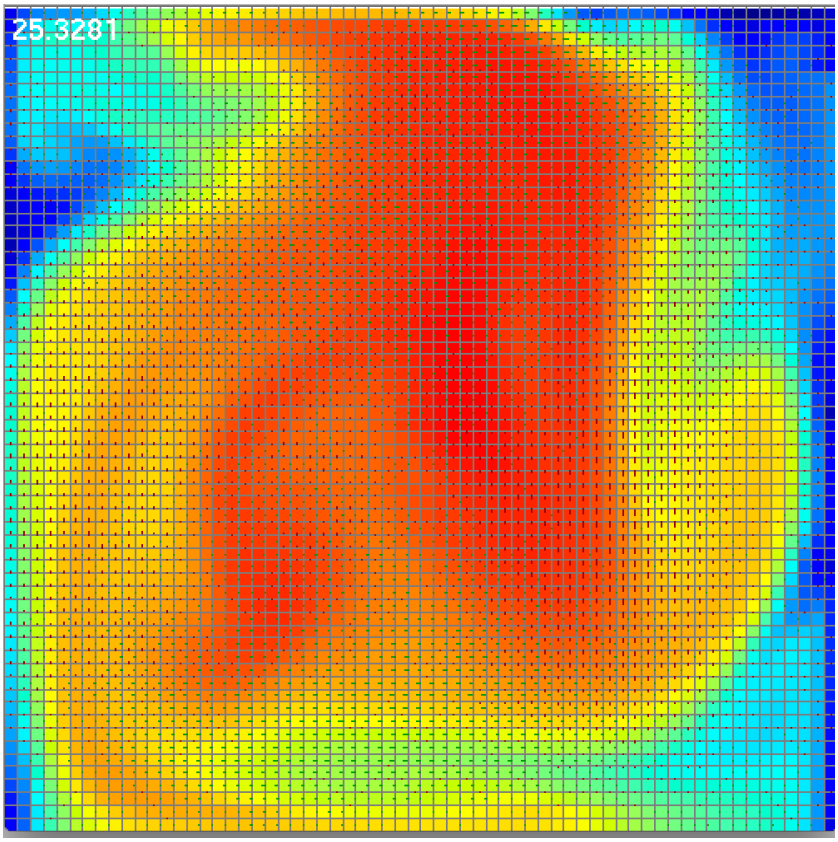

Figure 4: C++ Software Simulation

3. FPGA Hardware Design Architecture

The following figure includes block diagrams of the FPGA based system. The main blocks are shown below in Figure 5. The Processing Unit is designed in SystemVerilog to be a state machine that controls the access to different RAMs. The Processing Unit module executes the algorithm and eventually transfers data to the framebuffer.

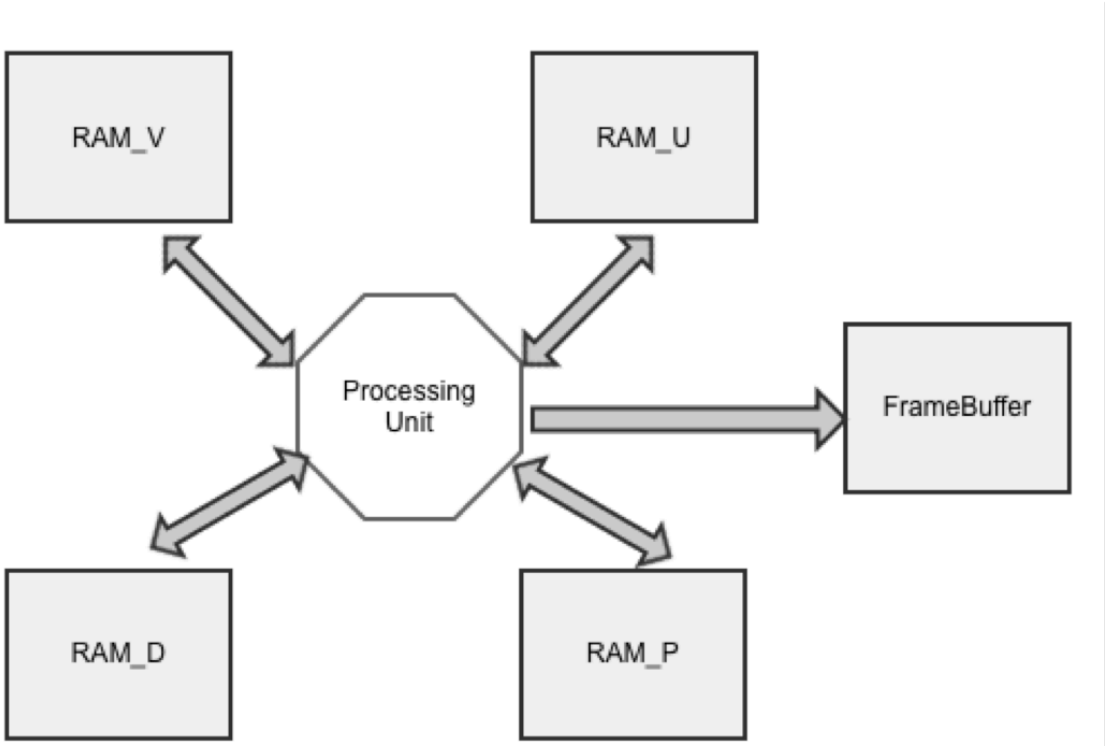

Figure 5: System Architecture

4. FPGA Hardware Design Timing

The monitor resolution is 640 x 480 pixels. Since we had a 50MHz clock, we can easily compute the time that is needed to refresh each frame, which is about 6.144ms. For a 64 x 64 smoke animation, the time that is allowed to calculate each pixel is roughly 65 clock cycles. The headroom is plenty so that we implemented this algorithm in serial.

5. FPGA Hardware Computation Order

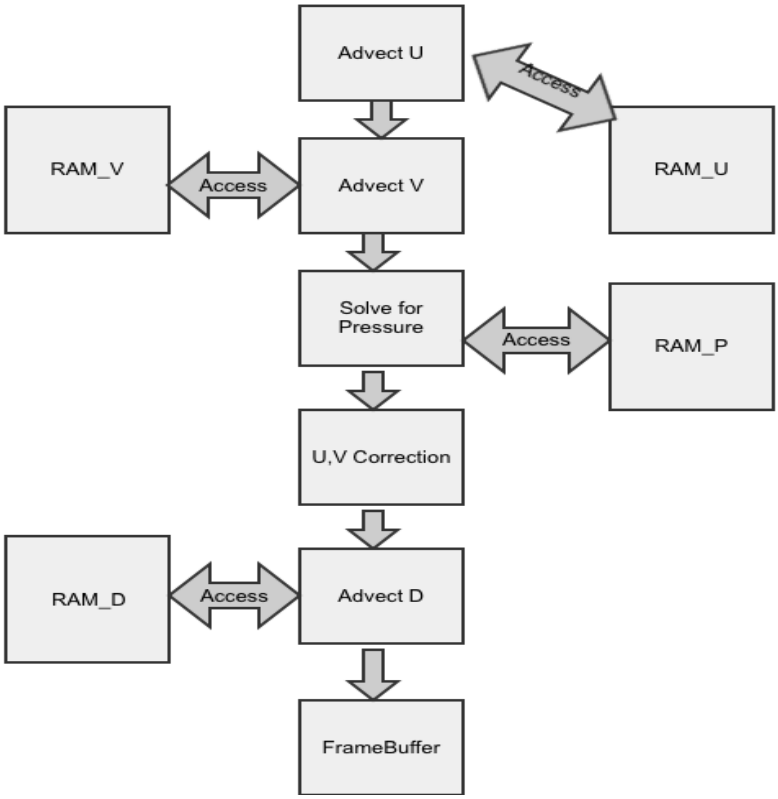

Figure 6: Hardware Implementation Order

Figure 6 shows the dataflow of the hardware. The order is the following: the Processing Unit takes the last values (or initial) from the memories and calculates horizontal and vertical velocities for the next frame and writes results back to memories. Then it executes the Projection step, which calculates pressure for each pixel based on results from the previous step. Then the system computes the more accurate U and V based on pressure. At last, the density for the smoke behavior is calculated based on accurate U and V.

6. FPGA Hardware Implementation

1. Specification

   Top module : Smoke simulation
   a. Input pins: clock, reset
   b. Output pins: VGA_R, VGA_G, VGA_B
   c. RAM using: Four 36 words x 8192 columns with two read/write port
   d. Sub-module:
      InterpolateV, InterpolateU, Project, Correction, InterpolateD, Display

2. Design Discription

   a. In ADVECT_U state, we start to calculate the advectU part, using two counters I and J to represent the for-loop in C code. The first cycle we generate the address for memory, and then we use two cycle to read the U&V value from RAM_U and RAM_V, and throw into the InterpolateV, InterpolateU module for calculating the bi and bj, since there are six 36bits x 36bits multiplexers and twelve 14-bit shifter, so we give them three cycles to calculate the answer.

      After having the bi and bj value from pervious procedure, we throw then into InterpolateU module again to have the U value at (bi, bj) coordinate interpolate with the adjacent value, finally we write back into RAM_U, finish the first point. The whole state will calculate 64x65 pixels and each pixels need twelve cycles to calculate.

   b. In ADVECT_V state, the procedure is almost the same with ADVECT_U, except the range is 65x64 pixels, and the input offset is at horizontal direction.

   c. In PROJECT state, since we need to take the boundary condition in to consideration, we add the pressure condition to have the correct U&V value. In this state, we check if the current U&V value is at the boundary, if is, we add the opposite force to let it slow down and move to opposite direction, so the smoke will not flow out of the boundary. This state will do 64x64 pixels, each pixel will take six cycles for calculation.

      This procedure is the most crucial part in the whole simulation, and we need to less than ten times iterations, if we iterate less than ten times, the smoke will diffused very quickly, but if we iterate too many times, we cannot meet the timing constrain of the display.

   d. The CORRECT state is to update the U&V value after adding the pressure condition, read the updated pressure value from RAM_P and write back to

RAM_U & RAM_V after correction.

e. The state ADVECT_D which is using the U&V value to interpolate the density, and then output the value. Since the maximum value of density is 1, so we need to time 255 to meet the RGB format.

f. To display the density, we generate the 64x64 frame buffer to save the value, and we enlarge the display range to 128x128 by interpolating one density point and its adjacent value.

3. Simulation Result (Repeat)

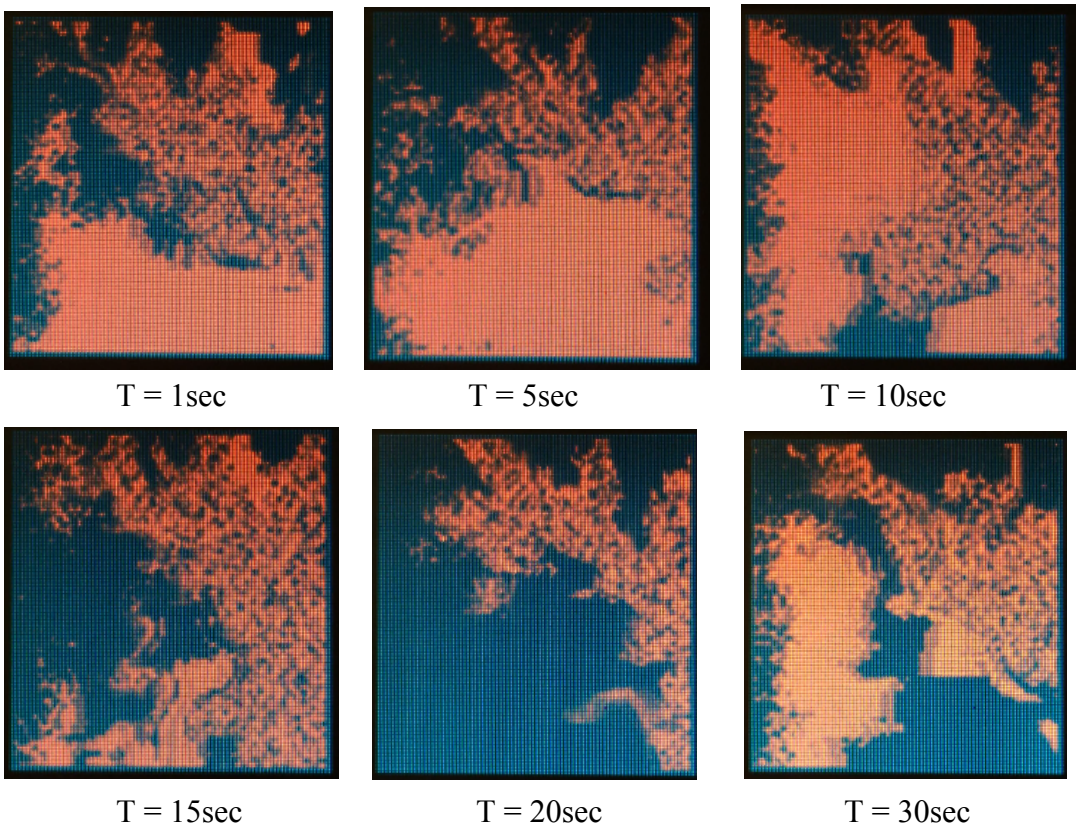

Figure 7: FPGA Based Simulation

7. Summary

To sum up, this project proves the theory described in Fattal, Raanan, and Dani Lischinski's paper. Simulation results that were obtained through both software and FPGA hardware were success and in good shapes. However, this

algorithm requires tremendous amount of iterative (for loop) operations and this number grows exponentially with the resolution of the animation. Meanwhile, implementing this algorithm in hardware is tedious. A processor with good parallel computation ability is necessary.

8. Reference

[1] Fattal, Raanan, and Dani Lischinski. "Target-driven Smoke Animation." ACM Transactions on Graphics 23.3 (2004): 441. Print.

9.C++ Code

Software in C++

StableFluidsSim.cpp

```
#include "StableFluidsSim.h"
#include <Eigen/LU>
#include <fstream>
#include <string>

using namespace std;

string DecToBin2(long long number)
{
    string result = "";

    int i = 36;
    do
    {
        if ( (number & 1L) == 0L )
            result += "0";
        else
            result += "1";

        number >>= 1L;
        i--;
    } while ( i );

    reverse(result.begin(), result.end());
    return result;
}

StableFluidsSim::StableFluidsSim( const int& rows, const int& cols)
:
m_N(rows)
, m_p(m_N + 2, m_N + 2)
, m_d(m_N + 2, m_N + 2)
, m_u(m_N + 2, m_N + 1)
, m_v(m_N + 1, m_N + 2)
{
```

```
    assert(rows==cols);

    clear();
}

StableFluidsSim::~StableFluidsSim()
{

}

////////////////////////////////////////////////////////////////////////////////
/////
void StableFluidsSim::SWAP(ArrayFXs *& x1, ArrayFXs *& x2)
{
    ArrayFXs * t = x1;
    x1 = x2;
    x2 = t;
}

void StableFluidsSim::add_source(int N, ArrayFXs * x, ArrayFXs * x0, FixedReal dt)
{
    // nothing to do. adding marker/velocity is handled in StableFluidsEnsemble.
    *x = *x0;
}

FixedReal StableFluidsSim::interpolateD(ArrayFXs * d, FixedReal i, FixedReal j)
{
    int i1 = CLAMP(i.toInt(), 0, m_N);
    int i2 = i1 + 1;

    int j1 = CLAMP(j.toInt(), 0, m_N);
    int j2 = j1 + 1;

    FixedReal s = CLAMP(i - i1, FixedReal(0), FixedReal(1));
    FixedReal t = CLAMP(j - j1, FixedReal(0), FixedReal(1));
    FixedReal os = FixedReal(1.0f) - s;
    FixedReal ot = FixedReal(1.0f) - t;

    return ((*d)(i1, j1) * os + (*d)(i2, j1) * s) * ot +
    ((*d)(i1, j2) * os + (*d)(i2, j2) * s) * t;
}
```

```
FixedReal StableFluidsSim::interpolateU(ArrayFXs * u, FixedReal i, FixedReal j)
{
    int i1 = CLAMP(i.toInt(), 0, m_N);
    int i2 = i1 + 1;

    int j1 = CLAMP((j - FixedReal(0.5f)).toInt(), 0, m_N - 1);
    int j2 = j1 + 1;

    FixedReal s = CLAMP(i - i1, FixedReal(0), FixedReal(1));
    FixedReal t = CLAMP(j - FixedReal(0.5f) - j1, FixedReal(0), FixedReal(1));
    FixedReal os = FixedReal(1.0f) - s;
    FixedReal ot = FixedReal(1.0f) - t;

    return ((*u)(i1, j1) * os + (*u)(i2, j1) * s) * ot +
    ((*u)(i1, j2) * os + (*u)(i2, j2) * s) * t;
}

FixedReal StableFluidsSim::interpolateV(ArrayFXs * v, FixedReal i, FixedReal j)
{
    int i1 = CLAMP((i - FixedReal(0.5f)).toInt(), 0, m_N - 1);
    int i2 = i1 + 1;

    int j1 = CLAMP(j.toInt(), 0, m_N);
    int j2 = j1 + 1;

    FixedReal s = CLAMP(i - FixedReal(0.5f) - i1, FixedReal(0), FixedReal(1));
    FixedReal t = CLAMP(j - j1, FixedReal(0), FixedReal(1));
    FixedReal os = FixedReal(1.0f) - s;
    FixedReal ot = FixedReal(1.0f) - t;

    return ((*v)(i1, j1) * os + (*v)(i2, j1) * s) * ot +
    ((*v)(i1, j2) * os + (*v)(i2, j2) * s) * t;
}

void StableFluidsSim::advectD(int N, ArrayFXs * x, ArrayFXs * x0, ArrayFXs * u,
ArrayFXs * v, FixedReal dt)
{
    for (int i = 1; i <= N; i++)
    {
        for (int j = 1; j <= N; j++)
```

```
        {
            FixedReal bi = FixedReal(i) - interpolateV(v, FixedReal(i), FixedReal(j))
* (dt * N);
            FixedReal bj = FixedReal(j) - interpolateU(u, FixedReal(i), FixedReal(j))
* (dt * N);

            int i1 = CLAMP(bi.toInt(), 0, m_N);
            int i2 = i1 + 1;

            int j1 = CLAMP(bj.toInt(), 0, m_N);
            int j2 = j1 + 1;

            FixedReal s = CLAMP(bi - i1, FixedReal(0), FixedReal(1));
            FixedReal t = CLAMP(bj - j1, FixedReal(0), FixedReal(1));
            FixedReal os = FixedReal(1.0f) - s;
            FixedReal ot = FixedReal(1.0f) - t;

            (*x)(i, j) = ((*x0)(i1, j1) * os + (*x0)(i2, j1) * s) * ot +
            ((*x0)(i1, j2) * os + (*x0)(i2, j2) * s) * t;
        }
    }
}

void StableFluidsSim::advectU(int N, ArrayFXs * x, ArrayFXs * x0, ArrayFXs * u,
ArrayFXs * v, FixedReal dt)
{
    FixedReal xoffset(0.5f);
    for (int i = 1; i <= N; i++)
    {
        for (int j = 0; j <= N; j++)
        {
            FixedReal bi = FixedReal(i) - interpolateV(v, FixedReal(i), FixedReal(j)
+ xoffset) * (dt * N);
            FixedReal bj = FixedReal(j) + xoffset - interpolateU(u, FixedReal(i),
FixedReal(j) + xoffset) * (dt * N);

            (*x)(i, j) = interpolateU(x0, bi, bj);
        }
    }

}
```

```
void StableFluidsSim::advectV(int N, ArrayFXs * x, ArrayFXs * x0, ArrayFXs * u,
ArrayFXs * v, FixedReal dt)
{
    FixedReal yoffset(0.5f);
    for (int i = 0; i <= N; i++)
    {
        for (int j = 1; j <= N; j++)
        {
            FixedReal bi = FixedReal(i) + yoffset - interpolateV(v, FixedReal(i) +
yoffset, FixedReal(j)) * (dt * N);
            FixedReal bj = FixedReal(j) - interpolateU(u, FixedReal(i) + yoffset,
FixedReal(j)) * (dt * N);

            (*x)(i, j) = interpolateV(x0, bi, bj);
        }
    }

}

void StableFluidsSim::project(int N, ArrayFXs * u, ArrayFXs * v, ArrayFXs * u0,
ArrayFXs * v0)
{

    // set solid boundary conditions
    for (int i = 0; i <= N + 1; i++)
    {
        (*u0)(i, 0) = FixedReal(0);
        (*u0)(i, N) = FixedReal(0);
        (*v0)(0, i) = FixedReal(0);
        (*v0)(N, i) = FixedReal(0);
    }

    for (int i = 0; i <= N + 1; i++)
    {
        for (int j = 0; j <= N + 1; j++)
        {
            m_p(i, j) = FixedReal(0);
        }
    }
```

```
    for (int i = 0; i <= N; i++)
    {
        (*u0)(0, i) = FixedReal(0);
        (*u0)(N + 1, i) = FixedReal(0);
        (*v0)(i, 0) = FixedReal(0);
        (*v0)(i, N + 1) = FixedReal(0);
    }

    // compute divergence of the velocity field

    // solve for pressure
    for (int k = 0; k < 10; k++)
    {
        for (int i = 1; i <= N; i++)
        {
            for (int j = 1; j <= N; j++)
            {
                FixedReal numer = ((*v0)(i, j) - (*v0)(i - 1, j) + (*u0)(i, j) - (*u0)(i,
j - 1));
                int denom = -4;

                if (i == 1) denom++;
                else       numer -= m_p(i - 1, j);
                if (i == N) denom++;
                else       numer -= m_p(i + 1, j);
                if (j == 1) denom++;
                else       numer -= m_p(i, j - 1);
                if (j == N) denom++;
                else       numer -= m_p(i, j + 1);

                switch (denom) {
                    case -4:
                        m_p(i, j) = -numer * FixedReal(0.25f);
                        break;
                    case -3:
                        m_p(i, j) = -numer * FixedReal(0.3333333333f);
                        break;
                    case -2:
                        m_p(i, j) = -numer * FixedReal(0.5f);
                        break;
                    case -1:
```

```
                    m_p(i, j) = -numer;
                    break;
                default:
                    break;
            }
        }
    }


    // apply pressure to correct velocities
    (*u) = (*u0);
    (*v) = (*v0);
    for (int i = 1; i <= N; i++)
    {
        for (int j = 1; j < N; j++)
        {
            (*u)(i, j) = (*u0)(i, j) - (m_p(i, j + 1) - m_p(i, j));
            (*v)(i, j) = (*v0)(i, j) - (m_p(i + 1, j) - m_p(i, j));
        }
    }

}

void StableFluidsSim::dens_step(int N, ArrayFXs * x, ArrayFXs * x0, ArrayFXs * u,
ArrayFXs * v, FixedReal dt)
{
    advectD(N, x, x0, u, v, dt);

}

void StableFluidsSim::vel_step(int N, ArrayFXs * u, ArrayFXs * v, ArrayFXs * u0,
ArrayFXs * v0, FixedReal dt)
{
    *u = *u0;
    *v = *v0;

    advectU(N, u0, u, u, v, dt);
    advectV(N, v0, v, u, v, dt);

    project(N, u, v, u0, v0);
```

```
}

void StableFluidsSim::stepSystem( const FixedReal& dt)
{
    ArrayFXs new_d(m_N + 2, m_N + 2);
    ArrayFXs new_u(m_N + 2, m_N + 1);
    ArrayFXs new_v(m_N + 1, m_N + 2);

    new_d.setZero();
    new_u.setZero();
    new_v.setZero();

    vel_step(m_N, &new_u, &new_v, &m_u, &m_v, dt);
    dens_step(m_N, &new_d, &m_d, &new_u, &new_v, dt);

    m_d = new_d;
    m_u = new_u;
    m_v = new_v;
}

const ArrayFXs& StableFluidsSim::getMarkerDensities() const
{
    return m_d;
}

ArrayFXs& StableFluidsSim::getMarkerDensities()
{
    return m_d;
}

ArrayFXs& StableFluidsSim::getHorizontalVelocities()
{
    return m_u;
}

const ArrayFXs& StableFluidsSim::getHorizontalVelocities() const
{
    return m_u;
}
```

```
ArrayFXs& StableFluidsSim::getVerticalVelocities()
{
    return m_v;
}

const ArrayFXs& StableFluidsSim::getVerticalVelocities() const
{
    return m_v;
}

int StableFluidsSim::physicalRows() const
{
    return m_N;
}

int StableFluidsSim::physicalCols() const
{
    return m_N;
}

void StableFluidsSim::clear()
{
    m_d.setZero();
    m_u.setZero();
    m_v.setZero();
}

void StableFluidsSim::setPrescribedVelocity(int p)
{
    switch (p)
    {
        case 0:
            break;
        case 1:
            for (int i = m_N * 0.2; i <= m_N * 0.8; i++)
                for (int j = m_N * 0.2; j <= m_N * 0.8; j++)
                    m_u(i, j) = m_v(i, j) = FixedReal(0.8);
            for (int i = m_N * 0.4; i <= m_N * 0.6; i++)
                for (int j = m_N * 0.4; j <= m_N * 0.6; j++)
                    m_d(i, j) = FixedReal(1.0);
            break;
```

```
    }
}


void StableFluidsSim::copyState( const StableFluidsSim& otherscene )
{
    m_N = otherscene.m_N;
    m_d = otherscene.m_d;
    m_u = otherscene.m_u;
    m_v = otherscene.m_v;
}

ArrayFXs& StableFluidsSim::getHorizontalVelocitiesAdvect()
{
    return m_uAfterAdvect;
}

const ArrayFXs& StableFluidsSim::getHorizontalVelocitiesAdvect() const
{
    return m_uAfterAdvect;
}

ArrayFXs& StableFluidsSim::getVerticalVelocitiesAdvect()
{
    return m_vAfterAdvect;
}

const ArrayFXs& StableFluidsSim::getVerticalVelocitiesAdvect() const
{
    return m_vAfterAdvect;
}

ArrayFXs& StableFluidsSim::getHorizontalVelocitiesDiffusion()
{
    return m_uAfterDiffusion;
}

const ArrayFXs& StableFluidsSim::getHorizontalVelocitiesDiffusion() const
{
    return m_uAfterDiffusion;
}
```

```
ArrayFXs& StableFluidsSim::getVerticalVelocitiesDiffusion()
{
    return m_vAfterDiffusion;
}

const ArrayFXs& StableFluidsSim::getVerticalVelocitiesDiffusion() const
{
    return m_vAfterDiffusion;
}

ArrayXb& StableFluidsSim::getHasFluid()
{
    return m_all_ones;
}

const ArrayXb& StableFluidsSim::getHasFluid() const
{
    return m_all_ones;
}
```

MathDefines.h

```
template<long long E>
struct BasicFixedReal
{
    typedef BasicFixedReal self;
    static const long long factor = 1L << E;
    BasicFixedReal( ) : m(0) { }
    BasicFixedReal(double d) : m(static_cast<long long>(d * factor)) { }
    BasicFixedReal(float d) : m(static_cast<long long>(d * factor)) { }
    BasicFixedReal(long long d, bool RAW = false) : m(RAW ? d : (d << E)) { }
    BasicFixedReal(int d, bool RAW = false) : m(RAW ? static_cast<long long>(d) :
(static_cast<long long>(d) << E)) { }
    self& operator+=(const self& x) { m += x.m; return *this; }
    self& operator-=(const self& x) { m -= x.m; return *this; }
    self& operator*=(const self& x) { m >>= (E >> 1); m *= (x.m >> (E >> 1)); return
*this; }
    self& operator/=(const self& x) { m /= x.m; m *= factor; return *this; }
    self& operator*=(int x) { m *= x; return *this; }
    self& operator/=(int x) { m /= x; return *this; }
    self operator-( ) { return self(-m, true); }
```

```
    double toDouble( ) const { return double(m) / factor; }
    float toFloat( ) const { return float(m) / factor; }
    int toInt( ) const {return m >> E; }
    long long toLongLong( ) const {return m >> E; }
    // friend functions
    friend self operator+(self x, const self& y) { return x += y; }
    friend self operator-(self x, const self& y) { return x -= y; }
    friend self operator*(self x, const self& y) { return x *= y; }
    friend self operator/(self x, const self& y) { return x /= y; }
    friend self operator>>(self x, const int& y) { return self(x.m >> y, true); }
    friend self operator<<(self x, const int& y) { return self(x.m << y, true); }
    // comparison operators
    friend bool operator==(const self& x, const self& y) { return x.m == y.m; }
    friend bool operator!=(const self& x, const self& y) { return x.m != y.m; }
    friend bool operator>(const self& x, const self& y) { return x.m > y.m; }
    friend bool operator<(const self& x, const self& y) { return x.m < y.m; }
    friend bool operator>=(const self& x, const self& y) { return x.m >= y.m; }
    friend bool operator<=(const self& x, const self& y) { return x.m <= y.m; }
    friend std::ostream &operator<<(std::ostream &out, const self& c)    //output
    {
        out << c.m;
        return out;
    }

    long long m;
};
typedef BasicFixedReal<28L> FixedReal;
```

10. Javascript

Software in HTML5/JavaScript

Index.html

```
<html><head>
<title>2D fluid dynamics</title>
<link href="./css/bootstrap.min.css" rel="stylesheet">

<script src="base64.js"></script>
<script src="./js/fluids_main.js"></script>

</head>
<body onload="main()" style="background: #000">
   <div class="container">
     <center><canvas id="myCanvas" width="1024" height="512"></canvas></center>
   </div><!-- /.container -->

   <div class="navbar navbar-inverse navbar-fixed-bottom" role="navigation">
     <div class="container">
       <div class="navbar-header">
         <a class="navbar-brand" href="#" id = "fileshow">A Brief Review of Modern
Computer Graphics</a>
       </div>
       <div class="collapse navbar-collapse">
         <ul class="nav navbar-nav">
           <li><a href="#" onclick="run()">Stop</a></li>
           <li      class="active"><a      href="#">itr.</a><input      value="10"
onchange="setIt( this.value )"></li>
           <li     class="active"><a     href="#">&beta;</a><input     value="10"
onchange="setBu( this.value )"></li>
           <li      class="active"><a      href="#">delay</a><input      value="0"
onchange="setDelay( this.value )"></li>
           <li class="active"><a href="#">fps</a><input id="framerate"></li>
           <li  class="active"><a  href="#">interpolation</a><input  type="range"
id="slidertract" min="0" max="40" value="3" onchange="setTract( this.value )"><a
href="#" id="intervalue">
             3
           </a></li>
```

```
          <li><a href="#" onclick="nextImage()">Switch</a></li>
          <li><a href="#" onclick="saveCanvasToBMP()">Save</a></li>
          <li id = "animControl"></li>
        </ul>
      </div><!--/.nav-collapse -->
    </div>
  </div>

  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js"></scrip
t>
  <script src="./js/bootstrap.min.js"></script>
</body>
</html>
```

Fluids_main.js

```
var curimgidx = 0;
var intensities =
[102,
394,
116
];

var addc = 0.0;

// our shaders base path
loadShaders.base = "shader/";

// our shaders loader
function loadShaders(gl, shaders, callback) {
   // (C) WebReflection - Mit Style License
   function onreadystatechange() {
      var
         xhr = this,
         i = xhr.i
      ;
      if (xhr.readyState == 4) {
         shaders[i] = gl.createShader(
            shaders[i].slice(0, 2) == "fs" ?
               gl.FRAGMENT_SHADER :
```

```
                gl.VERTEX_SHADER
            );
            gl.shaderSource(shaders[i], xhr.responseText);
            gl.compileShader(shaders[i]);
            if (!gl.getShaderParameter(shaders[i], gl.COMPILE_STATUS))
               throw gl.getShaderInfoLog(shaders[i])
            ;
            !--length && typeof callback == "function" && callback(shaders);
         }
      }
      for (var
         shaders = [].concat(shaders),
         asynchronous = !!callback,
         i = shaders.length,
         length = i,
         xhr;
         i--;
      ) {
         (xhr = new XMLHttpRequest).i = i;
         xhr.open("get", loadShaders.base + shaders[i] + ".c", asynchronous);
         if (asynchronous) {
            xhr.onreadystatechange = onreadystatechange;
         }
         xhr.send(null);
         onreadystatechange.call(xhr);
      }
      return shaders;
}

var gl, prog_advec, prog_force, prog_Source, prog_p, prog_div, prog_gather,
prog_show, prog_convert, prog_rgb2lab,
   FBO, FBO1, FBO2, FBO3, FBO6, texture, texture1, texture2, texture3, texture6,
texture30, texture7, textureSave, bgImage, bgImage0, bgImage2, bgImage7,
prog_gaussianx, prog_gaussiany,
   timer, delay = 0, it = 10, frames = 0, time, animation, accuframes = 0, tforce
= 5.0, startSave = 0,
   nX = 1024, nY = 512, sampLoc, sampLoc_gather, rvalueLoc;

var anim_grp = [], anim_pos = 0;
var flow_grp = [];
var anim_timer = null;
```

```
function main() {
   var c = document.getElementById("myCanvas");
   var err = "Your browser does not support ";
   if (!window.WebGLRenderingContext){
     alert(err+"WebGL. See http://get.webgl.org");
     return;}
   try { gl = c.getContext("experimental-webgl");
   } catch(e) {}
   if ( !gl ) {alert("Can't get WebGL"); return;}
   var ext;
   try { ext = gl.getExtension("OES_texture_float");
   } catch(e) {}
   if ( !ext ) {alert(err + "OES_texture_float extension"); return;}

   var shaders = loadShaders(gl, [
    "vs/shader",
    "fs/advec",
    "fs/convert",
    "fs/div",
    "fs/force",
    "fs/gather",
    "fs/gaussianx",
    "fs/gaussiany",
    "fs/p",
    "fs/rgb2lab",
    "fs/show",
    "fs/source"
    ]);


   prog_force  = gl.createProgram();
   gl.attachShader(prog_force, shaders[0] );
   gl.attachShader(prog_force, shaders[4] );
   gl.linkProgram(prog_force);
   gl.useProgram(prog_force);
   gl.uniform1f(gl.getUniformLocation(prog_force, "c"), .001*.5*10 );
   gl.uniform1i(gl.getUniformLocation(prog_force, "samp"), 1);
   gl.uniform1i(gl.getUniformLocation(prog_force, "samp3"), 3);

   prog_advec  = gl.createProgram();
```

```
gl.attachShader(prog_advec, shaders[0] );
gl.attachShader(prog_advec, shaders[1] );
gl.linkProgram(prog_advec);
gl.useProgram(prog_advec);
gl.uniform1i(gl.getUniformLocation(prog_advec, "samp3"), 3);
gl.uniform1i(gl.getUniformLocation(prog_advec, "samp30"), 4);

prog_div  = gl.createProgram();
gl.attachShader(prog_div, shaders[0] );
gl.attachShader(prog_div, shaders[3] );
gl.linkProgram(prog_div);
gl.useProgram(prog_div);
gl.uniform1i(gl.getUniformLocation(prog_div, "samp"), 1);

prog_convert  = gl.createProgram();
gl.attachShader(prog_convert, shaders[0] );
gl.attachShader(prog_convert, shaders[2] );
gl.linkProgram(prog_convert);
gl.useProgram(prog_convert);
gl.uniform1i(gl.getUniformLocation(prog_convert, "samp"), 0);

prog_rgb2lab  = gl.createProgram();
gl.attachShader(prog_rgb2lab, shaders[0] );
gl.attachShader(prog_rgb2lab, shaders[9] );
gl.linkProgram(prog_rgb2lab);
gl.useProgram(prog_rgb2lab);
gl.uniform1i(gl.getUniformLocation(prog_rgb2lab, "samp30"), 4);

prog_gaussianx  = gl.createProgram();
gl.attachShader(prog_gaussianx, shaders[0] );
gl.attachShader(prog_gaussianx, shaders[6] );
gl.linkProgram(prog_gaussianx);
gl.useProgram(prog_gaussianx);
gl.uniform1i(gl.getUniformLocation(prog_gaussianx, "samp30"), 3);

prog_gaussiany  = gl.createProgram();
gl.attachShader(prog_gaussiany, shaders[0] );
gl.attachShader(prog_gaussiany, shaders[7] );
gl.linkProgram(prog_gaussiany);
gl.useProgram(prog_gaussiany);
gl.uniform1i(gl.getUniformLocation(prog_gaussiany, "samp30"), 6);
```

```
prog_gather  = gl.createProgram();
gl.attachShader(prog_gather, shaders[0] );
gl.attachShader(prog_gather, shaders[5] );
gl.linkProgram(prog_gather);
gl.useProgram(prog_gather);
sampLoc_gather  = gl.getUniformLocation(prog_gather, "samp");
gl.uniform1f(gl.getUniformLocation(prog_gather, "nudt"), .001*.00000025 );
gl.uniform1i(gl.getUniformLocation(prog_gather, "samp3"), 3);
gl.uniform1i(gl.getUniformLocation(prog_gather, "samp30"), 4);
gl.uniform1i(gl.getUniformLocation(prog_gather, "samp7"), 7);

prog_p  = gl.createProgram();
gl.attachShader(prog_p, shaders[0] );
gl.attachShader(prog_p, shaders[8] );
gl.linkProgram(prog_p);
gl.useProgram(prog_p);
sampLoc  = gl.getUniformLocation(prog_p, "samp");

prog_Source  = gl.createProgram();
gl.attachShader(prog_Source, shaders[0] );
gl.attachShader(prog_Source, shaders[11] );
gl.linkProgram(prog_Source);
gl.useProgram(prog_Source);
gl.uniform1i(gl.getUniformLocation(prog_Source, "samp2"), 2);
gl.uniform1f(gl.getUniformLocation(prog_Source, "c"), addc);
rvalueLoc = gl.getUniformLocation(prog_Source, "rvalue");

prog_show  = gl.createProgram();
gl.attachShader(prog_show, shaders[0]);
gl.attachShader(prog_show, shaders[10]);
gl.linkProgram(prog_show);
gl.useProgram(prog_show);
gl.uniform1i(gl.getUniformLocation(prog_show, "samp30"), 3);

gl.useProgram(prog_advec);
var aPosLoc = gl.getAttribLocation(prog_advec, "aPos");
var aTexLoc = gl.getAttribLocation(prog_advec, "aTexCoord");
gl.enableVertexAttribArray( aPosLoc );
gl.enableVertexAttribArray( aTexLoc );
var data = new Float32Array([-1,-1, 0,0,  1,-1, 1,0,  -1,1, 0,1,
```

```
  1,1, 1,1]);
gl.bindBuffer(gl.ARRAY_BUFFER, gl.createBuffer());
gl.bufferData(gl.ARRAY_BUFFER, data, gl.STATIC_DRAW);
gl.vertexAttribPointer(aPosLoc, 2, gl.FLOAT, gl.FALSE, 16, 0);
gl.vertexAttribPointer(aTexLoc, 2, gl.FLOAT, gl.FALSE, 16, 8);

var pixels = [],  hX = 2/nX, hY = 2/nY, T, v;
for(var i = 0; i<nY; i++)
  for(var j = 0; j<nX; j++){
    pixels.push( 0, 0, 0, 0 );
  }
texture2 = gl.createTexture();
gl.activeTexture(gl.TEXTURE2);
gl.bindTexture(gl.TEXTURE_2D, texture2);
gl.pixelStorei(gl.UNPACK_ALIGNMENT, 1);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, nX, nY, 0,
  gl.RGBA, gl.FLOAT, new Float32Array(pixels));
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);



texture = gl.createTexture();
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, texture);
gl.pixelStorei(gl.UNPACK_ALIGNMENT, 1);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, nX, nY, 0,
  gl.RGBA, gl.FLOAT, new Float32Array(pixels));
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);

texture1 = gl.createTexture();
gl.activeTexture(gl.TEXTURE1);
gl.bindTexture(gl.TEXTURE_2D, texture1);
gl.pixelStorei(gl.UNPACK_ALIGNMENT, 1);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, nX, nY, 0,
  gl.RGBA, gl.FLOAT, new Float32Array(pixels));
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);

texture3 = gl.createTexture();
gl.activeTexture(gl.TEXTURE3);
```

```
  gl.bindTexture(gl.TEXTURE_2D, texture3);
  gl.pixelStorei(gl.UNPACK_ALIGNMENT, 1);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, nX, nY, 0,
    gl.RGBA, gl.FLOAT, new Float32Array(pixels));
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);

  texture6 = gl.createTexture();
  gl.activeTexture(gl.TEXTURE6);
  gl.bindTexture(gl.TEXTURE_2D, texture6);
  gl.pixelStorei(gl.UNPACK_ALIGNMENT, 1);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, nX, nY, 0,
    gl.RGBA, gl.FLOAT, new Float32Array(pixels));
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);

  texture30 = gl.createTexture();
  bgImage0 = new Image();
  bgImage0.src = './origin/' + curimgidx + '.png';
  bgImage0.onload = function() {
    gl.activeTexture(gl.TEXTURE4);
    gl.bindTexture(gl.TEXTURE_2D, texture30);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
         gl.UNSIGNED_BYTE, bgImage0);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D,                    gl.TEXTURE_MIN_FILTER,
gl.LINEAR_MIPMAP_NEAREST);
    gl.generateMipmap(gl.TEXTURE_2D);
  }

  // Load first empty flow
  texture7 = gl.createTexture();
  bgImage7 = new Image();
  bgImage7.src = './flow/0_0.png';
  bgImage7.onload = function() {
     flow_grp.push(bgImage7);
    gl.activeTexture(gl.TEXTURE7);
    gl.bindTexture(gl.TEXTURE_2D, texture7);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA,
         gl.UNSIGNED_BYTE, bgImage7);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
```

```
     gl.texParameteri(gl.TEXTURE_2D,                      gl.TEXTURE_MIN_FILTER,
gl.LINEAR_MIPMAP_NEAREST);
     gl.generateMipmap(gl.TEXTURE_2D);
   }

   textureSave = gl.createTexture();
   gl.activeTexture(gl.TEXTURE5);
   gl.bindTexture(gl.TEXTURE_2D, textureSave);
   gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, nX, nY, 0, gl.RGBA, gl.UNSIGNED_BYTE,
null);
   gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
   gl.texParameteri(gl.TEXTURE_2D,                      gl.TEXTURE_MIN_FILTER,
gl.LINEAR_MIPMAP_NEAREST);
   gl.generateMipmap(gl.TEXTURE_2D);


   FBO = gl.createFramebuffer();
   gl.bindFramebuffer(gl.FRAMEBUFFER, FBO);
   gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
      gl.TEXTURE_2D, texture, 0);
   FBO1 = gl.createFramebuffer();
   gl.bindFramebuffer(gl.FRAMEBUFFER, FBO1);
   gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
      gl.TEXTURE_2D, texture1, 0);
   FBO2 = gl.createFramebuffer();
   gl.bindFramebuffer(gl.FRAMEBUFFER, FBO2);
   gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
      gl.TEXTURE_2D, textureSave, 0);
   FBO3 = gl.createFramebuffer();
   gl.bindFramebuffer(gl.FRAMEBUFFER, FBO3);
   gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
      gl.TEXTURE_2D, texture3, 0);
   FBO6 = gl.createFramebuffer();
   gl.bindFramebuffer(gl.FRAMEBUFFER, FBO6);
   gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
      gl.TEXTURE_2D, texture6, 0);


   if( gl.checkFramebufferStatus(gl.FRAMEBUFFER) != gl.FRAMEBUFFER_COMPLETE)
     alert(err + "FLOAT as the color attachment to an FBO");
```

```
   timer = setInterval(fr, 500);
   time = new Date().getTime();
   animation = "animate";
//   draw();
   anim();
}
function draw(){
  //Convert image from RGB to LAB Color space (shader/fs/rgb2lab.c)
   gl.bindFramebuffer(gl.FRAMEBUFFER, FBO3);
   gl.useProgram(prog_rgb2lab);
   gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
   gl.flush();

  //Gaussian Blur in X direction (shader/fs/gaussianx.c)
   gl.bindFramebuffer(gl.FRAMEBUFFER, FBO6);
   gl.useProgram(prog_gaussianx);
   gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
   gl.flush();

  //Gaussian Blur in Y direction (shader/fs/gaussiany.c)
   gl.bindFramebuffer(gl.FRAMEBUFFER, FBO3);
   gl.useProgram(prog_gaussiany);
   gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
   gl.flush();

  //Gaussian Blur in Y direction (shader/fs/gaussiany.c)
   gl.bindFramebuffer(gl.FRAMEBUFFER, FBO1);
   gl.useProgram(prog_Source);
   if(accuframes > 100)
    gl.uniform1f(gl.getUniformLocation(prog_Source, "c"), 0);
   else
    gl.uniform1f(gl.getUniformLocation(prog_Source, "c"), addc);

   gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
   gl.flush();

  //Add force (shader/fs/force.c)
   gl.bindFramebuffer(gl.FRAMEBUFFER, FBO);
   gl.useProgram(prog_force);
   gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
   gl.flush();
```

```
  //Advect fluids (shader/fs/advec.c)
   gl.bindFramebuffer(gl.FRAMEBUFFER, FBO1);
   gl.useProgram(prog_advec);
   gl.uniform1f(gl.getUniformLocation(prog_advec, "tractforce"), tforce * 0.0001);
   gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
   gl.flush();

  //Gauss-Seidel solving for incompressibility (shader/fs/p.c)
   gl.useProgram(prog_p);
   for(var i = 0; i < it; i++){
    gl.uniform1i(sampLoc, 1);
    gl.bindFramebuffer(gl.FRAMEBUFFER, FBO);
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
    gl.flush();

    gl.uniform1i(sampLoc, 0);
    gl.bindFramebuffer(gl.FRAMEBUFFER, FBO1);
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
    gl.flush();
   }

  //Apply the corrected pressure to velocity (shader/fs/div.c)
   gl.bindFramebuffer(gl.FRAMEBUFFER, FBO);
   gl.useProgram(prog_div);
   gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
   gl.flush();

  //Gather the smoke towards the input image (shader/fs/gather.c)
   gl.useProgram(prog_gather);
   gl.uniform1f(gl.getUniformLocation(prog_gather,   "tractforce"),   tforce   *
0.00001);
   for(var i = 0; i < it; i++){
    gl.uniform1i(sampLoc_gather, 0);
    gl.bindFramebuffer(gl.FRAMEBUFFER, FBO1);
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
    gl.flush();

    gl.uniform1i(sampLoc_gather, 1);
    gl.bindFramebuffer(gl.FRAMEBUFFER, FBO);
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
```

```
    gl.flush();
   }

   //useless
   gl.useProgram(prog_convert);
   gl.bindFramebuffer(gl.FRAMEBUFFER, FBO2);
   gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
   gl.flush();
/*
   if(startSave)
    saveCanvasToBMP();
*/

  //Combine the smoke in the L channel with A-B channels (shader/fs/show.c)
   gl.useProgram(prog_show);
   gl.bindFramebuffer(gl.FRAMEBUFFER, null);
   gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
   frames++;
   accuframes++;
}
function startS() {
  startSave = 1;
}
function anim(){
   draw();
   switch ( animation ){
     case "animate":
       if (delay == 0) requestAnimationFrame(anim);
       else setTimeout("requestAnimationFrame(anim)", delay);
       break;
     case "stop":
       break;
   }
}
function run(v) {
  if( animation == "animate" ){
    animation = "stop";
    document.getElementById('runBtn').value = "Run ";}
  else{
    animation = "animate";
    document.getElementById('runBtn').value = "Stop";
```

```
    anim();
  }
}
function reset() {
  if( animation == "stop" ){
    animation = "reset";
    document.getElementById('runBtn').value = "Stop";
    anim();}
  else animation = "reset";
}
function fr(){
  var ti = new Date().getTime();
  var fps = Math.round(1000*frames/(ti - time));
  document.getElementById("framerate").value = fps;
  frames = 0;  time = ti;
}
function setDelay(val) {
  delay = parseInt(val);
}
function setIt(val) {
  it = parseInt(val);
}
function setBu(v) {
  var bu = v.valueOf();
  gl.useProgram(prog_force);              //    c = dt*b/2
  gl.uniform1f(gl.getUniformLocation(prog_force, "c"), .001*.5*bu );
}

function animImage() {
    if(anim_pos <= anim_grp.length) {

        if(anim_pos < anim_grp.length) {
            console.dir('Switch Image ' + anim_pos);
            gl.activeTexture(gl.TEXTURE4);
            gl.bindTexture(gl.TEXTURE_2D, texture30);
            gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE,
anim_grp[anim_pos]);
            gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
            gl.texParameteri(gl.TEXTURE_2D,               gl.TEXTURE_MIN_FILTER,
gl.LINEAR_MIPMAP_NEAREST);
            gl.generateMipmap(gl.TEXTURE_2D);
```

```
        }

        gl.activeTexture(gl.TEXTURE7);
        gl.bindTexture(gl.TEXTURE_2D, texture7);
        gl.texImage2D(gl.TEXTURE_2D,  0,  gl.RGBA,  gl.RGBA,  gl.UNSIGNED_BYTE,
flow_grp[anim_pos]);
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
        gl.texParameteri(gl.TEXTURE_2D,                   gl.TEXTURE_MIN_FILTER,
gl.LINEAR_MIPMAP_NEAREST);
        gl.generateMipmap(gl.TEXTURE_2D);
        anim_pos++;
    }
}

function imageExists(image_url){

   var http = new XMLHttpRequest();

   http.open('HEAD', image_url, false);
   http.send();

   return http.status != 404;

}

function startAnim() {
    var sec = anim_grp.length * 4.0 / 9.0;
    sec = Math.max(sec, 20);
    var dur = sec * 1000 / anim_grp.length;
//  console.dir('Timer Duration: ' + dur);
    anim_timer = setInterval(animImage, dur);
}

function loadAnimFile(i) {
    var cname = './origin/' + curimgidx + '_' + i + '.png';
    var flowname = './flow/' + curimgidx + '_' + (i - 1) + '.png';

    var animImg = new Image();

    animImg.src = cname;
```

```
    animImg.onload = function() {
        anim_grp.push(animImg);
        var flowImg = new Image();
        flowImg.src = flowname;

        flowImg.onload = function() {
            flow_grp.push(flowImg);
            loadAnimFile(i + 1);
        }
    }

    animImg.onerror = function() {
        if(anim_grp.length > 0) {
            document.getElementById('animControl').innerHTML   =   '<a   href="#"
onclick="startAnim()"><b>Start Anim</b></a>';
        } else {
            document.getElementById('animControl').innerHTML = '';
        }
        var flowImg = new Image();
        flowImg.src = flowname;
        flowImg.onload = function() {
            flow_grp.push(flowImg);
        }
    }
}

function checkAnimImage() {
    var i = 1;

    if(anim_timer != null)
        clearInterval(anim_timer);

    anim_grp = [];
    flow_grp = [];
    anim_pos = 0;

    loadAnimFile(1);
}

function nextImage() {
  var lastidx = curimgidx;
```

```
  curimgidx = (curimgidx + 1) % intensities.length;

  texture30 = gl.createTexture();
  bgImage0 = new Image();
  bgImage0.src = './origin/' + curimgidx + '.png';
  bgImage0.onload = function() {
    texture2 = gl.createTexture();
    bgImage2 = new Image();
    bgImage2.src = './tracker/' + curimgidx + '.png';
    bgImage2.onload = function() {
      gl.activeTexture(gl.TEXTURE2);
      gl.bindTexture(gl.TEXTURE_2D, texture2);
      gl.texImage2D(gl.TEXTURE_2D,   0,   gl.RGBA,   gl.RGBA,   gl.UNSIGNED_BYTE,
bgImage2);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
      gl.texParameteri(gl.TEXTURE_2D,                      gl.TEXTURE_MIN_FILTER,
gl.LINEAR_MIPMAP_NEAREST);
      gl.generateMipmap(gl.TEXTURE_2D);

      gl.activeTexture(gl.TEXTURE4);
      gl.bindTexture(gl.TEXTURE_2D, texture30);
      gl.texImage2D(gl.TEXTURE_2D,   0,   gl.RGBA,   gl.RGBA,   gl.UNSIGNED_BYTE,
bgImage0);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
      gl.texParameteri(gl.TEXTURE_2D,                      gl.TEXTURE_MIN_FILTER,
gl.LINEAR_MIPMAP_NEAREST);
      gl.generateMipmap(gl.TEXTURE_2D);

      addc = (intensities[curimgidx] - intensities[lastidx] + 10) * 512.0 * 5e-05
* 0.01;
      accuframes = 0;
      gl.useProgram(prog_Source);
      var angle = Math.random() * Math.PI * 2.0;
      var rx = Math.cos(angle);
      var ry = Math.sin(angle);

      gl.uniform4f(rvalueLoc, rx, ry, Math.random(), Math.random());

      //Load Last Flow Image
      if(flow_grp.length == 0) {
         console.dir('Something wrong with flow_grp!');
```

```
     } else {
        gl.activeTexture(gl.TEXTURE7);
        gl.bindTexture(gl.TEXTURE_2D, texture7);
        gl.texImage2D(gl.TEXTURE_2D,  0,  gl.RGBA,  gl.RGBA,  gl.UNSIGNED_BYTE,
flow_grp[flow_grp.length - 1]);
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
        gl.texParameteri(gl.TEXTURE_2D,                    gl.TEXTURE_MIN_FILTER,
gl.LINEAR_MIPMAP_NEAREST);
        gl.generateMipmap(gl.TEXTURE_2D);
      }

     checkAnimImage();

     document.getElementById('fileshow').innerHTML = 'File Name: ' + bgImage2.src;
    }
  }
}

function setTract(v) {
  tforce = v * v;
  document.getElementById('intervalue').innerHTML = '' + tforce;
}

var createBMP = function(oData, iWidth, iHeight) {
    var aHeader = [];

    aHeader.push(0x42); // magic 1
    aHeader.push(0x4D);

    var iFileSize = iWidth*iHeight*3 + 54; // total header size = 54 bytes
    aHeader.push(iFileSize % 256); iFileSize = Math.floor(iFileSize / 256);
    aHeader.push(iFileSize % 256); iFileSize = Math.floor(iFileSize / 256);
    aHeader.push(iFileSize % 256); iFileSize = Math.floor(iFileSize / 256);
    aHeader.push(iFileSize % 256);

    aHeader.push(0); // reserved
    aHeader.push(0);
    aHeader.push(0); // reserved
    aHeader.push(0);

    aHeader.push(54); // dataoffset
```

```
    aHeader.push(0);
    aHeader.push(0);
    aHeader.push(0);

    var aInfoHeader = [];
    aInfoHeader.push(40); // info header size
    aInfoHeader.push(0);
    aInfoHeader.push(0);
    aInfoHeader.push(0);

    var iImageWidth = iWidth;
    aInfoHeader.push(iImageWidth % 256); iImageWidth = Math.floor(iImageWidth /
256);
    aInfoHeader.push(iImageWidth % 256); iImageWidth = Math.floor(iImageWidth /
256);
    aInfoHeader.push(iImageWidth % 256); iImageWidth = Math.floor(iImageWidth /
256);
    aInfoHeader.push(iImageWidth % 256);

    var iImageHeight = iHeight;
    aInfoHeader.push(iImageHeight % 256); iImageHeight = Math.floor(iImageHeight /
256);
    aInfoHeader.push(iImageHeight % 256); iImageHeight = Math.floor(iImageHeight /
256);
    aInfoHeader.push(iImageHeight % 256); iImageHeight = Math.floor(iImageHeight /
256);
    aInfoHeader.push(iImageHeight % 256);

    aInfoHeader.push(1); // num of planes
    aInfoHeader.push(0);

    aInfoHeader.push(24); // num of bits per pixel
    aInfoHeader.push(0);

    aInfoHeader.push(0); // compression = none
    aInfoHeader.push(0);
    aInfoHeader.push(0);
    aInfoHeader.push(0);

    var iDataSize = iWidth*iHeight*3;
    aInfoHeader.push(iDataSize % 256); iDataSize = Math.floor(iDataSize / 256);
```

```
    aInfoHeader.push(iDataSize % 256); iDataSize = Math.floor(iDataSize / 256);
    aInfoHeader.push(iDataSize % 256); iDataSize = Math.floor(iDataSize / 256);
    aInfoHeader.push(iDataSize % 256);

    for (var i=0;i<16;i++) {
        aInfoHeader.push(0);    // these bytes not used
    }

    var iPadding = (4 - ((iWidth * 3) % 4)) % 4;

    var aImgData = oData;

    var strPixelData = "";
    var y = iHeight;
    do {
        var iOffsetY = iWidth*(y-1)*4;
        var strPixelRow = "";
        for (var x=0;x<iWidth;x++) {
            var iOffsetX = 4*x;

            strPixelRow += String.fromCharCode(aImgData[iOffsetY+iOffsetX+2]);
            strPixelRow += String.fromCharCode(aImgData[iOffsetY+iOffsetX+1]);
            strPixelRow += String.fromCharCode(aImgData[iOffsetY+iOffsetX]);
        }
        for (var c=0;c<iPadding;c++) {
            strPixelRow += String.fromCharCode(0);
        }
        strPixelData += strPixelRow;
    } while (--y);

    var     strEncoded     =     encodeData(aHeader.concat(aInfoHeader))     +
encodeData(strPixelData);

    return strEncoded;
}

var makeDataURI = function(strData, strMime) {
    return "data:" + strMime + ";base64," + strData;
}

var saveFile = function(strData) {
```

```
    document.location.href = strData;
}

var encodeData = function(data) {
    var strData = "";
    if (typeof data == "string") {
        strData = data;
    } else {
        var aData = data;
        for (var i=0;i<aData.length;i++) {
            strData += String.fromCharCode(aData[i]);
        }
    }
    return btoa(strData);
}

function saveCanvasToBMP() {
    var strDownloadMime = "image/octet-stream";
    gl.bindFramebuffer(gl.FRAMEBUFFER, FBO2);
    var pixels = new Uint8Array(4 * nX * nY);
    gl.readPixels(0,0,nX,nY,gl.RGBA,gl.UNSIGNED_BYTE,pixels);
    var strImgData = createBMP(pixels, nX, nY);

    gl.bindFramebuffer(gl.FRAMEBUFFER, null);
}
```

vs/shader.c

```
  attribute vec2 aPos;
  attribute vec2 aTexCoord;
  varying   vec2 tc;
void main(void) {
   gl_Position = vec4(aPos, 0., 1.);
   tc = aTexCoord;
}
```

fs/advec.c

```
precision highp float;
  uniform sampler2D samp;
  varying vec2 tc;
  uniform sampler2D samp3;
```

```
  uniform sampler2D samp30;
  uniform float tractforce;
  const float hX = 1./1024.;
  const float hY = 1./512.;
  const float dt = .001;
  const float tauX = .5*dt/hX;
  const float tauY = .5*dt/hY;
void main(void) {
   vec2 D = -vec2(tauX,tauY)*vec2(
    texture2D(samp, tc).r + texture2D(samp, vec2(tc.r - hX, tc.g)).r,
    texture2D(samp, tc).g + texture2D(samp, vec2(tc.r, tc.g - hY)).g );
   vec2 Df = floor(D),   Dd = D - Df;
   vec2 tc1 = tc + Df*vec2(hX, hY);
   vec3 newr =
    (texture2D(samp, tc1).rgb*(1. - Dd.g) +
     texture2D(samp, vec2(tc1.r, tc1.g + hY)).rgb*Dd.g)*(1. - Dd.r) +
    (texture2D(samp, vec2(tc1.r + hX, tc1.g)).rgb*(1. - Dd.g) +
     texture2D(samp, vec2(tc1.r + hX, tc1.g + hY)).rgb*Dd.g)*Dd.r;
    float tstar =
    (texture2D(samp30, tc1).r*(1. - Dd.g) +
     texture2D(samp30, vec2(tc1.r, tc1.g + hY)).r*Dd.g)*(1. - Dd.r) +
    (texture2D(samp30, vec2(tc1.r + hX, tc1.g)).r*(1. - Dd.g) +
     texture2D(samp30, vec2(tc1.r + hX, tc1.g + hY)).r*Dd.g)*Dd.r;
    newr.b += (tstar - newr.b) * tractforce;
   gl_FragColor = vec4( newr, texture2D(samp, tc).a );
}
```

fs/div.c

```
precision highp float;
  uniform sampler2D samp;
  varying vec2 tc;
  const float n = 512., h = 1./n;
void main(void) {
   vec4 t = texture2D(samp, tc);
   t.r -= (texture2D(samp, vec2(tc.r + h, tc.g)).a - t.a)*n;
   t.g -= (texture2D(samp, vec2(tc.r, tc.g + h)).a - t.a)*n;
   gl_FragColor = t;
}
```

fs/p.c

```
precision highp float;
```

```
  uniform sampler2D samp;
  varying vec2 tc;
  const float h = 1. / 512.;
void main(void) {
   vec4 t = texture2D(samp, tc);
   t.a =
     (texture2D(samp, vec2(tc.r - h, tc.g)).a +
      texture2D(samp, vec2(tc.r + h, tc.g)).a +
      texture2D(samp, vec2(tc.r, tc.g - h)).a +
      texture2D(samp, vec2(tc.r, tc.g + h)).a -
     (
      (t.r - texture2D(samp, vec2(tc.r - h, tc.g)).r) * h +
      (t.g - texture2D(samp, vec2(tc.r, tc.g - h)).g) * h
      )) *.25;
   gl_FragColor = t;
}
```

fs/source.c

```
precision highp float;
  uniform sampler2D samp;
  uniform sampler2D samp2;
  uniform vec4 rvalue;
  uniform float c;
  varying vec2 tc;
  varying float S;
void main(void) {
   vec4 t = texture2D(samp, tc);
   vec4 tu = texture2D(samp2, tc);
   vec2 rv = (tu.rg * 2.0 - 1.0) * abs(c) * 3.0;
   t.rgb += vec3(rv.r, rv.g, tu.b * c);
   gl_FragColor = t;
}
```

11. SystemVerilog for FPGA

```
module Smoke_Simulation(

input logic clk50, reset,

output logic [7:0] VGA_R, VGA_G, VGA_B,

output logic VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n

);

//////////////////////////////// ----- RAM -----
////////////////////////////////////////

RAM_U2

RAM_U1(.address_a(U_addr_a1), .address_b(U_addr_b1), .clock(cl

k50), .data_a(U_da1),

.data_b(U_db1), .wren_a(U_wen_a1), .wren_b(U_wen_b1), .q_a(U_qa1

), .q_b(U_qb1));

RAM_V2
```

```
RAM_V1(.address_a(V_addr_a1), .address_b(V_addr_b1), .clock(cl
k50), .data_a(V_da1),

.data_b(V_db1), .wren_a(V_wen_a1), .wren_b(V_wen_b1), .q_a(V_qa1
), .q_b(V_qb1));

RAM_D2

RAM_D1(.address_a(D_addr_a1), .address_b(D_addr_b1), .clock(cl
k50), .data_a(D_da1),

.data_b(D_db1), .wren_a(D_wen_a1), .wren_b(D_wen_b1), .q_a(D_qa1
), .q_b(D_qb1));

RAM_P2

RAM_P1(.address_a(P_addr_a1), .address_b(P_addr_b1), .clock(cl
k50), .data_a(P_da1),

.data_b(P_db1), .wren_a(P_wen_a1), .wren_b(P_wen_b1), .q_a(P_qa1
), .q_b(P_qb1));
```

```
//////////////////////////// ----- Frame Buffer -----
////////////////////////////////

logic [7:0] D_temp [0:4355];

logic Dt_wen;

logic [12:0] Dt_addr;

logic [7:0] Dt_d, Dt_q;

always_ff @ (posedge clk50) begin

end

if (Dt_wen) D_temp[Dt_addr] <= Dt_d;

else D_temp <= D_temp;

assign Dt_q = (Dt_wen == 0) ? D_temp[Dt_addr] : 8'd0;

always_ff @ (posedge clk50) begin

if(reset) Dt_addr <= 13'd0;

else begin

if(state == DISPLAY1 | state == DISPLAY2) begin
```

```
end

if (endOfField) Dt_addr <= 13'd0;

else if(hcount[0] == 1) Dt_addr <= Dt_addr;

else if((hcount <= 263 ) & (vcount <= 131)) begin

end

end

if(J_count == 8'd1) Dt_addr <= Dt_addr + 13'd1;

else Dt_addr <= Dt_addr;

else if(vcount <= 131 & hcount == 264) begin

if (I_count == 8'd0) Dt_addr <= Dt_addr - 13'd66;

else Dt_addr <= Dt_addr;

else Dt_addr <= Dt_addr;

else begin

if (endOfField) Dt_addr <= 13'd0;

else if(hcount[0] == 1) Dt_addr <= Dt_addr;

else if((hcount <= 263 ) & (vcount <= 131)) begin

end
```

```
if(J_count == 8'd1) Dt_addr <= Dt_addr + 13'd1;

else Dt_addr <= Dt_addr;

end

end

end

else if(vcount <= 131 & hcount == 264) begin

end

if (I_count == 8'd0) Dt_addr <= Dt_addr - 13'd66;

else Dt_addr <= Dt_addr;

else Dt_addr <= Dt_addr;

always_comb begin

if(state == DISPLAY1) begin

if(D_qa2[29] == 1) Dt_d = 8'd255;

else if(D_qa2[28:21] > 8'd200) Dt_d = 8'd255;

else Dt_d = D_qa2[28:21] - 8'd100;

end

else if(state == DISPLAY2) begin
```

```
if(D_qa1[29] == 1) Dt_d = 8'd255;

else if(D_qa1[28:21] > 8'd200) Dt_d = 8'd255;

else Dt_d = D_qa1[28:21] - 8'd100;

end

end

else Dt_d = 8'd0;

always_comb begin

if(state == DISPLAY1 | state == DISPLAY2) begin


if (hcount[0] == 0 & I_count == 8'd0 & J_count == 8'd0) begin


end

if((hcount <= 263 ) & (vcount <= 131)) Dt_wen = 1;


else Dt_wen = 0;


else Dt_wen = 0;


end
```

```
else Dt_wen = 0;

end

//////////////////////////////////// ----- logic -----
////////////////////////////////////

logic [20:0]TOTAL_count; logic [3:0]state,n_state;

logic [12:0]U_addr_a1,U_addr_b1; //8192


logic [35:0]U_da1,U_db1,U_qa1,U_qb1; //36bits


logic [12:0]U_addr_a2,U_addr_b2;


logic [35:0]U_da2,U_db2,U_qa2,U_qb2;


logic U_wen_b1,U_wen_b2; logic U_wen_a1,U_wen_a2;

---

logic [12:0]V_addr_a1,V_addr_b1;


logic [35:0]V_da1,V_db1,V_qa1,V_qb1;


logic [12:0]V_addr_a2,V_addr_b2;
```

```
logic [35:0]V_da2,V_db2,V_qa2,V_qb2;

logic V_wen_b1,V_wen_b2;logic V_wen_a1,V_wen_a2;

---

logic [12:0]D_addr_a1,D_addr_b1;

logic [35:0]D_da1,D_db1,D_qa1,D_qb1;

logic [12:0]D_addr_a2,D_addr_b2;

logic [35:0]D_da2,D_db2,D_qa2,D_qb2;

logic D_wen_b1,D_wen_b2;logic D_wen_a1,D_wen_a2;

---

logic [12:0]P_addr_a1,P_addr_b1;

logic [35:0]P_da1,P_db1,P_qa1,P_qb1;

logic P_wen_b1,P_wen_b2;logic P_wen_a1,P_wen_a2;

---
```

```
logic [12:0] READ_count,ADVECT_count;

logic [3:0] ITP_count;

logic [7:0] I_count,J_count;

logic [4:0] ITER_count;

logic signed [35:0] U0_i,U0_j; logic signed [35:0] V0_i,V0_j;
logic signed [35:0] D0_i,D0_j;

---

logic signed [7:0] P0_i,P0_j; logic signed [7:0] C0_i,C0_j;

---

logic signed [12:0] U_dir,V_dir,P_dir,D_dir;

logic [7:0] U0_i1,U0_i2;

logic [15:0] U0_i3,U0_i4;

logic [12:0] U0_j1,U0_j2;

logic signed [35:0] U0_s,U0_t,U0_os,U0_ot,U0_stemp,U0_ttemp;
```

```
logic signed [35:0] U0_s_w,U0_t_w,U0_os_w,U0_ot_w;

logic signed [35:0] insertU0_1,insertU0_2;

logic signed [35:0] U0_jtemp;

logic signed [35:0] u0,u0_w,u1_w;

logic [12:0] V0_i1,V0_i2,V0_j1,V0_j2;


logic signed [35:0] V0_s,V0_t,V0_os,V0_ot,V0_stemp,V0_ttemp;


logic signed [35:0] V0_s_w,V0_t_w,V0_os_w,V0_ot_w;


logic signed [35:0] insertV0_1,insertV0_2;


logic signed [35:0] V0_itemp;


logic signed [35:0] v0,v0_w,v1_w;


logic [12:0] D0_i1,D0_i2,D0_j1,D0_j2;


logic signed [35:0] D0_s,D0_t,D0_os,D0_ot,D0_stemp,D0_ttemp;
```

```
logic signed [35:0] D0_s_w, D0_t_w, D0_os_w, D0_ot_w;

logic signed [35:0] insertD0_1,insertD0_2;

logic signed [35:0] d0, d0_w;

logic signed [35:0] bi, bj;

logic [12:0] P0_i1, P0_i2, P0_i3; logic [12:0] P0_j1, P0_j2,
P0_j3;
```

```
logic signed [35:0] numer1, numer2, numer3;

logic [1:0] denom;

logic signed [35:0] p0, p0_w;

logic [12:0] C0_i1, C0_i2; logic [12:0] C0_j1, C0_j2;
```

```
logic signed [35:0] c0_u, c0_v;
```

```
logic Dswitch;

logic [12:0]U_temp_1,U_temp_2,U_temp_3,U_temp_4;logic
[12:0]V_temp_1,V_temp_2,V_temp_3,V_temp_4;logic [12:0]
D_temp_1,D_temp_2,D_temp_3,D_temp_4;

logic [12:0]P_temp_v1,P_temp_v2,P_temp_u1,P_temp_u2,
P_temp_p1,P_temp_p2,P_temp_p3,

P_temp_p4;

logic [12:0]C_temp_v,C_temp_u,C_temp_p1,C_temp_p2,
C_temp_p3,C_temp_p4;

logic signed [35:0]U_ans_1,U_ans_2,U_ans_3,U_ans_4;logic
signed [35:0]V_ans_1,V_ans_2,V_ans_3,V_ans_4;logic signed
[35:0]D_ans_1,D_ans_2,D_ans_3,D_ans_4;logic signed [35:0]
P_ans_1,P_ans_2,P_ans_3,P_ans_4;


parameter IDLE = 4'd0;

parameter ADVECT_U = 4'd1;

parameter INITIALIZE_V = 4'd2;
```

```
parameter ADVECT_V = 4'd3;

parameter INITIALIZE_P = 4'd4;

parameter PROJECT = 4'd5;

parameter INITIALIZE_C = 4'd6;

parameter CORRECTION = 4'd7;

parameter INITIALIZE_D = 4'd8;

parameter ADVECT_D1 = 4'd9;

parameter INITIALIZE_F1 = 4'd10;

parameter DISPLAY1 = 4'd11;

parameter ADVECT_D2 = 4'd12;

parameter INITIALIZE_F2 = 4'd13;

parameter DISPLAY2 = 4'd14;

//////////////////////////////------ State Machine ------
//////////////////////////////

always_ff @(posedge clk50 or posedge reset) begin //state


end
```

```
if (reset) state <= IDLE;

else state <= n_state;

always_comb begin //n_state

case(state)

IDLE: begin

end

n_state = ADVECT_U;

ADVECT_U: begin

if(ADVECT_count == 13'd4159 & ITP_count == 4'd12) n_state =
INITIALIZE_V;

else n_state = state;

end
```

```
INITIALIZE_V : n_state = ADVECT_V;

INITIALIZE_P : n_state = PROJECT;

INITIALIZE_C : n_state = CORRECTION;

ADVECT_V: begin

end

if(ADVECT_count == 13'd4159 & ITP_count == 4'd12) n_state =
INITIALIZE_P;

else n_state = state;

PROJECT: begin

if(ADVECT_count == 13'd4095 & ITP_count == 4'd6) begin

if(ITER_count == 5'd12) n_state = INITIALIZE_C;

else n_state = INITIALIZE_P;

end

end
```

```
else n_state = state;

CORRECTION: begin

end

if(ADVECT_count == 13'd4095 & ITP_count == 4'd6) n_state =
INITIALIZE_D;

end

else n_state = state;

INITIALIZE_D : begin

end

if(Dswitch) n_state = ADVECT_D2;

else n_state = ADVECT_D1;

ADVECT_D1: begin
```

```
if(ADVECT_count == 13'd4095 & ITP_count == 4'd12) n_state =
INITIALIZE_F1;

else n_state = state;

INITIALIZE_F1 : begin

if(vcount > 128) n_state = DISPLAY1;

else n_state = state;

end

DISPLAY1: begin

if(vcount == 128) n_state = IDLE;

else n_state = state;

end

ADVECT_D2: begin
```

```
if(ADVECT_count == 13'd4095 & ITP_count == 4'd12) n_state =
INITIALIZE_F2;

else n_state = state;

end

INITIALIZE_F2 : begin

if(vcount > 128) n_state = DISPLAY2;

else n_state = state;

end

DISPLAY2: begin

if(vcount == 128) n_state = IDLE;

else n_state = state;

end
```

```
default: n_state = IDLE;

endcase

end

////////////////////////////------ COUNTERS ------
////////////////////////////

always_ff @(posedge clk50 or posedge reset) begin //Dswitch


if (reset) Dswitch <= 1'd0;

else begin

if(state == ADVECT_D1) Dswitch <= 1'd1;

else if(state == ADVECT_D2) Dswitch <= 1'd0;

else Dswitch <= Dswitch;

end

end

always_ff @(posedge clk50 or posedge reset) begin //ADVECT_count
```

```
if (reset) ADVECT_count <= 13'd0;

else begin

if(state == IDLE | state == INITIALIZE_V | state == INITIALIZE_P
|

state == INITIALIZE_D | state == INITIALIZE_C)

ADVECT_count <= 13'd0;

else if(state == ADVECT_U | state == ADVECT_V | state == ADVECT_D1
| state ==

ADVECT_D2) begin

if (ITP_count == 4'd12) ADVECT_count <= ADVECT_count + 13'd1;

end

else ADVECT_count <= ADVECT_count;

else if(state == PROJECT | state == CORRECTION) begin

if (ITP_count == 4'd6) ADVECT_count <= ADVECT_count + 13'd1;

else ADVECT_count <= ADVECT_count;

end

else ADVECT_count <= ADVECT_count;
```

```
end
end
always_ff @(posedge clk50 or posedge reset) begin //ITP_count
if (reset) ITP_count <= 4'd15;
else begin
if (state == ADVECT_U | state == ADVECT_V | state == ADVECT_D1
| state == ADVECT_D2)
begin //0~12
if(ITP_count == 4'd12) ITP_count <= 4'd0;
else ITP_count <= ITP_count + 4'd1;
end
end
end
else if (state == PROJECT | state == CORRECTION) begin //0~6
end
if(ITP_count == 4'd6) ITP_count <= 4'd0;
else ITP_count <= ITP_count + 4'd1;
else ITP_count <= 4'd15;
```

```
always_ff @(posedge clk50 or posedge reset) begin //ITER_count

if (reset) ITER_count <= 5'd0;

else begin

if (state == IDLE) ITER_count <= 5'd0;

else if (state == PROJECT & ADVECT_count == 13'd4095 &

ITP_count == 4'd6) ITER_count

<= ITER_count + 5'd1;

else ITER_count <= ITER_count;

end

end

always_ff @(posedge clk50 or posedge reset) begin //I_count


if (reset) I_count <= 8'd0;


else begin


case(state)


IDLE: I_count <= 8'd1;
```

```
INITIALIZE_V: I_count <= 8'd0;

INITIALIZE_P: I_count <= 8'd1;

INITIALIZE_C: I_count <= 8'd1;

INITIALIZE_D: I_count <= 8'd1;

ADVECT_U: begin

1~64

end

if(J_count == 8'd64 & ITP_count == 4'd11) I_count <= I_count +

8'd1; //i =


else I_count <= I_count;

0~64

end

ADVECT_V: begin

if(J_count == 8'd64 & ITP_count == 4'd11) I_count <= I_count +

8'd1; //i =
```

```
else I_count <= I_count;

end

PROJECT: begin

if(J_count == 8'd64 & ITP_count == 4'd5) I_count <= I_count +
8'd1; //i =

1~64

else I_count <= I_count;

CORRECTION: begin

if(J_count == 8'd64 & ITP_count == 4'd5) I_count <= I_count +
8'd1; //i =

1~64

else I_count <= I_count;
```

```
end

ADVECT_D1: begin

if(J_count == 8'd64 & ITP_count == 4'd11) I_count <= I_count +
8'd1; //i =

1~64

else I_count <= I_count;

end

1~64

ADVECT_D2: begin

if(J_count == 8'd64 & ITP_count == 4'd11) I_count <= I_count +
8'd1; //i =

else I_count <= I_count;

end
```

```
INITIALIZE_F1: I_count <= 8'd0;

INITIALIZE_F2: I_count <= 8'd0;

DISPLAY1: begin

if(vcount > 131) I_count <= 8'd0;

else if(hcount == 264) begin

end

end

if(I_count == 8'd1) I_count <= 8'd0; //j = 1~64

else I_count <= 8'd1;

else I_count <= I_count;

DISPLAY2: begin

end

if(vcount > 131) I_count <= 8'd0;
```

```
else if(hcount == 264) begin

end

if(I_count == 8'd1) I_count <= 8'd0; //j = 1~64

else I_count <= 8'd1;

else I_count <= I_count;

default: I_count <= I_count;


endcase

end

end

always_ff @(posedge clk50 or posedge reset) begin //J_count


if (reset) J_count <= 8'd0;

else begin

case(state)


IDLE: J_count <= 8'd0;
```

```
INITIALIZE_V: J_count <= 8'd1;

INITIALIZE_P: J_count <= 8'd1;

ADVECT_U: begin


if(ITP_count == 4'd11) begin


if(J_count == 8'd64) J_count <= 8'd0; //j = 0~64


else J_count <= J_count + 8'd1;


end


else J_count <= J_count;


end


ADVECT_V: begin


if(ITP_count == 4'd11) begin


if(J_count == 8'd64) J_count <= 8'd1; //j = 1~64
```

```
else J_count <= J_count + 8'd1;

end

else J_count <= J_count;

end

PROJECT: begin

if(ITP_count == 4'd5) begin

if(J_count == 8'd64) J_count <= 8'd1; //j = 1~64

else J_count <= J_count + 8'd1;

end

else J_count <= J_count;

end

INITIALIZE_C: J_count <= 8'd1;
```

```
INITIALIZE_D: J_count <= 8'd1;

CORRECTION: begin

end

if(ITP_count == 4'd5) begin

if(J_count == 8'd64) J_count <= 8'd1; //j = 1~64

else J_count <= J_count + 8'd1;

end

else J_count <= J_count;

ADVECT_D1: begin

if(ITP_count == 4'd11) begin

if(J_count == 8'd64) J_count <= 8'd1; //j = 1~64

else J_count <= J_count + 8'd1;
```

```
end

else J_count <= J_count;

end

ADVECT_D2: begin

if(ITP_count == 4'd11) begin

end

if(J_count == 8'd64) J_count <= 8'd1; //j = 1~64

else J_count <= J_count + 8'd1;

else J_count <= J_count;

end

INITIALIZE_F1: J_count <=

8'd0;

INITIALIZE_F2: J_count <=
```

```
8'd0;

DISPLAY1: begin

end

if(hcount[0] == 0) begin

end

if(hcount > 263 | vcount > 131) J_count <= 8'd0;

else if (J_count == 8'd1) J_count <= 8'd0;

else J_count <= 8'd1;

else J_count <= J_count;

DISPLAY2: begin


if(hcount[0] == 0) begin


if(hcount > 263 | vcount > 131) J_count <= 8'd0;


end

else if (J_count == 8'd1) J_count <= 8'd0;


else J_count <= 8'd1;
```

```
else J_count <= J_count;

end

default: J_count <= J_count;

endcase

end

end

////////////////////////////////------ U_RAM1 ------
////////////////////////////////

IDLE: U_addr_a1 <= 13'd0; ADVECT_U: begin

end

always_ff @(posedge clk50 or posedge reset) begin //U_addr_a1


if (reset) U_addr_a1 <= 13'd0;

else begin

case(state)
```

```
if(ITP_count == 4'd0) U_addr_a1 <= U_temp_1 ; //first address

else if(ITP_count == 4'd1) U_addr_a1 <= U_temp_3 ; //third
address

else if(ITP_count == 4'd6) U_addr_a1 <= U_temp_1 ; else

if(ITP_count == 4'd7) U_addr_a1 <= U_temp_3 ;

else U_addr_a1 <= U_addr_a1 ;

ADVECT_V: begin


end

end

end

if(ITP_count == 4'd0) U_addr_a1 <= U_temp_1 ;


else if(ITP_count == 4'd1) U_addr_a1 <= U_temp_3 ;


else U_addr_a1 <= U_addr_a1 ;


ADVECT_D1: begin


if(ITP_count == 4'd0) U_addr_a1 <= U_temp_1 ;
```

```
else if(ITP_count == 4'd1) U_addr_a1 <= U_temp_3 ;

else U_addr_a1 <= U_addr_a1 ;

ADVECT_D2: begin

if(ITP_count == 4'd0) U_addr_a1 <= U_temp_1 ;

else if(ITP_count == 4'd1) U_addr_a1 <= U_temp_3 ;

else U_addr_a1 <= U_addr_a1 ;

end

end

default: U_addr_a1 <= 13'd0;

endcase

assign U_dir = {5'd0,I_count} * 13'd65 + {5'd0,J_count};

IDLE: U_addr_b1 <= 13'd0; ADVECT_U: begin

always_ff @(posedge clk50 or posedge reset) begin //U_addr_b1
```

```
if (reset) U_addr_b1 <= 13'd0;

else begin

case(state)

if(ITP_count == 4'd0) U_addr_b1 <= U_temp_2 ; //second address

else if(ITP_count == 4'd1) U_addr_b1 <= U_temp_4 ; //fourth
address

else if(ITP_count == 4'd6) U_addr_b1 <= U_temp_2 ; else

if(ITP_count == 4'd7) U_addr_b1 <= U_temp_4 ;

else U_addr_b1 <= U_addr_b1 ;

end

ADVECT_V: begin

end

assign U_wen_a1 = 1'd0; always_comb begin

assign U_da1 = 36'd0;
```

```
    //U_wen_a1
//U_wen_b1
          //U_da1 (no use)
if(ITP_count == 4'd0) U_addr_b1 <= U_temp_2 ;

else if(ITP_count == 4'd1) U_addr_b1 <= U_temp_4 ;

else U_addr_b1 <= U_addr_b1 ;

CORRECTION: begin

if(ITP_count == 4'd4) U_addr_b1 <= U_dir ; //write back

else U_addr_b1 <= U_addr_b1 ;

end

ADVECT_D1: begin

if(ITP_count == 4'd0) U_addr_b1 <= U_temp_2 ;

end

else if(ITP_count == 4'd1) U_addr_b1 <= U_temp_4 ;
```

```
else U_addr_b1 <= U_addr_b1 ;

ADVECT_D2: begin

if(ITP_count == 4'd0) U_addr_b1 <= U_temp_2 ;

else if(ITP_count == 4'd1) U_addr_b1 <= U_temp_4 ;

else U_addr_b1 <= U_addr_b1 ;

end

default: U_addr_b1 <= 13'd0;

end

end

endcase

if(state == CORRECTION) begin //write back after correction

if(ITP_count == 4'd5) U_wen_b1 = 1'd1;

else U_wen_b1 = 1'd0;
```

```
end

else U_wen_b1 = 1'd0;

end

always_comb begin //U_db1

if (state == CORRECTION) begin

if(ITP_count == 4'd5) begin //write back

U_db1 = c0_u;

end

end

else U_db1 = 36'd0;

end

else U_db1 = 36'd0;
```

```
////////////////////////////////------ U_RAM2 ------
////////////////////////////////

always_ff @(posedge clk50 or posedge reset) begin //U_addr_a2


if (reset) U_addr_a2 <= 13'd0;


else begin


case(state)


PROJECT: begin


if(ITP_count == 4'd0) U_addr_a2 <= P_temp_u1 ;


else U_addr_a2 <= U_addr_a2 ;


end


CORRECTION: begin


if(ITP_count == 4'd0) U_addr_a2 <= C_temp_u ;


else U_addr_a2 <= U_addr_a2 ;
```

```
end

default: U_addr_a2 <= 13'd0;

endcase

end

end

always_ff @(posedge clk50 or posedge reset) begin //U_addr_b2


if (reset) U_addr_b2 <= 13'd0;

else begin

case(state)


IDLE: U_addr_b2 <= 13'd0;

ADVECT_U: begin


if(ITP_count == 4'd10) U_addr_b2 <= U_dir; //write address
```

```
end

else U_addr_b2 <= U_addr_b2;

PROJECT: begin

end

if(ITP_count == 4'd0) U_addr_b2 <= P_temp_u2 ;

else U_addr_b2 <= U_addr_b2;

CORRECTION: begin

U_addr_b2 <= U_addr_b2;

end

default: U_addr_b2 <= 13'd0;

endcase

end

end
```

```
assign U_wen_a2 = 1'd0; //U_wen_a2

always_comb begin //U_wen_b2

end

if(state == ADVECT_U) begin //write back after advect

if(ITP_count == 4'd11) U_wen_b2 = 1'd1;

else U_wen_b2 = 1'd0;

end

else U_wen_b2 = 1'd0;

assign U_da2 = 36'd0;

//U_da2 (no use)
always_comb begin //U_db2

if (state == ADVECT_U) begin

if(ITP_count == 4'd11) begin //write back

if(J_count == 8'd0 | J_count == 8'd64 ) //if it's boundry (j=0
or j=64),
```

```
set to zero

U_db2 = 36'd0;

end

else U_db2 = u1_w;

end

else U_db2 = 36'd0;

else U_db2 = 36'd0;

end
//////////////////////////////// ----- U_ans -----

////////////////////////////////////

always_ff @(posedge clk50 or posedge reset) begin //U_ans_1

if (reset) U_ans_1 <= 36'd0;

else begin

if (state == ADVECT_U) begin

if(ITP_count == 4'd2 | ITP_count == 4'd8) U_ans_1 <= U_qa1;

else U_ans_1 <= U_ans_1;

end
```

```
else if (state == ADVECT_V | state == ADVECT_D1 | state ==
ADVECT_D2) begin

if(ITP_count == 4'd2) U_ans_1 <= U_qa1;

end

else U_ans_1 <= U_ans_1;

else if (state == PROJECT | state == CORRECTION) begin

if(ITP_count == 4'd2) U_ans_1 <= U_qa2; // read from U_RAM2

else U_ans_1 <= U_ans_1;

end

else U_ans_1 <= U_ans_1;

end

end

always_ff @(posedge clk50 or posedge reset) begin //U_ans_2

if (reset) U_ans_2 <= 36'd0;

else begin

if (state == ADVECT_U) begin

if(ITP_count == 4'd2 | ITP_count == 4'd8) U_ans_2 <= U_qb1;
```

```
else U_ans_2 <= U_ans_2;

end

else if (state == ADVECT_V | state == ADVECT_D1 | state ==
ADVECT_D2) begin

if(ITP_count == 4'd2) U_ans_2 <= U_qb1;

end

else U_ans_2 <= U_ans_2;

else if (state == PROJECT) begin

if(ITP_count == 4'd2) U_ans_2 <= U_qb2; // read from U_RAM2

else U_ans_2 <= U_ans_2;

end

else U_ans_2 <= U_ans_2;

end

end

always_ff @(posedge clk50 or posedge reset) begin //U_ans_3

if (reset) U_ans_3 <= 36'd0;

else begin
```

```
if (state == ADVECT_U) begin

if(ITP_count == 4'd3 | ITP_count == 4'd9) U_ans_3 <= U_qa1;

else U_ans_3 <= U_ans_3;

end

else if (state == ADVECT_V | state == ADVECT_D1 | state ==
ADVECT_D2) begin

if(ITP_count == 4'd3) U_ans_3 <= U_qa1;

end

else U_ans_3 <= U_ans_3;

else if (state == PROJECT) begin

if(ITP_count == 4'd3) U_ans_3 <= U_qa2; // read from U_RAM2

else U_ans_3 <= U_ans_3;

end

else U_ans_3 <= U_ans_3;

end

end

always_ff @(posedge clk50 or posedge reset) begin //U_ans_4
```

```
if (reset) U_ans_4 <= 36'd0;

else begin

if (state == ADVECT_U) begin

if(ITP_count == 4'd3 | ITP_count == 4'd9) U_ans_4 <= U_qb1;

else U_ans_4 <= U_ans_4;

end

else if (state == ADVECT_V | state == ADVECT_D1 | state ==
ADVECT_D2) begin

if(ITP_count == 4'd3) U_ans_4 <= U_qb1;

end

else U_ans_4 <= U_ans_4;

else if (state == PROJECT) begin

if(ITP_count == 4'd3) U_ans_4 <= U_qb2; // read from U_RAM2

else U_ans_4 <= U_ans_4;

end

else U_ans_4 <= U_ans_4;

end
```

```
end

////////////////////////////////------ advectU ------
//////////////////////////////////

always_ff @(posedge clk50 or posedge reset) begin //U0_i

if (reset) U0_i <= 36'd0;

else begin

case(state)

IDLE: U0_i <= {8'd1,28'd0};

ADVECT_U: begin

if(ITP_count == 4'd12) U0_i <= {I_count,28'd0}; //FixedReal(i),
for bj

else if(ITP_count == 4'd5) begin

end

end

U0_i <= bi;


end


else U0_i <= U0_i;


end
```

```
INITIALIZE_V: U0_i <= {8'd0,1'd1,27'd0};

INITIALIZE_D: U0_i <= {8'd1,28'd0};

ADVECT_V: begin


if(ITP_count == 4'd12) U0_i <= {I_count,1'd1,27'd0};
//FixedReal(i) +

yoffset

else U0_i <= U0_i;

end

ADVECT_D1: begin

if(ITP_count == 4'd12) U0_i <= {I_count,28'd0}; // only for bj

else U0_i <= U0_i;

ADVECT_D2: begin

end
```

```
if(ITP_count == 4'd12) U0_i <= {I_count,28'd0}; // only for bj

else U0_i <= U0_i;

default: U0_i <= U0_i;

endcase

end

always_ff @(posedge clk50 or posedge reset) begin //U0_j

if (reset) U0_j <= 36'd0;

else begin

case(state)

IDLE: U0_j <= {8'd0,1'd1,27'd0}; //for first case

U0_j <= bj;

ADVECT_U: begin
```

```
if(ITP_count == 4'd12) U0_j <= {J_count,1'd1,27'd0};
//FixedReal(j) + xoffset


else if(ITP_count == 4'd5) begin

end

else U0_j <= U0_j;

end

INITIALIZE_V: U0_j <= {8'd1,28'd0};

INITIALIZE_D: U0_j <= {8'd1,28'd0};

ADVECT_V: begin

if(ITP_count == 4'd12) U0_j <= {J_count,28'd0}; // FixedReal(j)


else U0_j <= U0_j;

end

ADVECT_D1: begin
```

```
if(ITP_count == 4'd12) U0_j <= {J_count,28'd0}; // FixedReal(j)

else U0_j  <= U0_j;

end

ADVECT_D2: begin

if(ITP_count == 4'd12) U0_j <= {J_count,28'd0}; // FixedReal(j)

end

endcase

else U0_j  <= U0_j;

default: U0_j  <= U0_j;

end

end

`ifdef TEST

logic [7:0] U0_i_D,U0_j_D;
```

```
assign U0_i_D = U0_i[35:28]; assign U0_j_D = U0_j[35:28];

---

`endif

always_ff @(posedge clk50 or posedge reset) begin //U0_s

if (reset) U0_s <= 36'd0;

else begin

if (state == ADVECT_U) begin

end

if(ITP_count == 4'd0 | ITP_count == 4'd6) U0_s <= U0_s_w;

else U0_s <= U0_s;

else if (state == ADVECT_V | state == ADVECT_D1 | state ==
ADVECT_D2) begin

if(ITP_count == 4'd0) U0_s <= U0_s_w;

else U0_s <= U0_s;

end

end

end
```

```
else U0_s <= U0_s;

always_ff @(posedge clk50 or posedge reset) begin //U0_t


if (reset) U0_t <= 36'd0;


else begin


if (state == ADVECT_U) begin


if(ITP_count == 4'd0 | ITP_count == 4'd6) U0_t <= U0_t_w;

else U0_t <= U0_t;

end

else if (state == ADVECT_V | state == ADVECT_D1 | state ==
ADVECT_D2) begin

if(ITP_count == 4'd0) U0_t <= U0_t_w;

end

else U0_t <= U0_t;


else U0_t <= U0_t;

end
```

```
end

always_ff @(posedge clk50 or posedge reset) begin //U0_os

if (reset) U0_os <= 36'd0;

else begin

if (state == ADVECT_U) begin

end

if(ITP_count == 4'd0 | ITP_count == 4'd6) U0_os <= U0_os_w;

else U0_os <= U0_os;

else if (state == ADVECT_V | state == ADVECT_D1 | state ==
ADVECT_D2) begin

if(ITP_count == 4'd0) U0_os <= U0_os_w;

else U0_os <= U0_os;

end

end

end

else U0_os <= U0_os;

always_ff @(posedge clk50 or posedge reset) begin //U0_ot
```

```
if (reset) U0_ot <= 36'd0;

else begin

if (state == ADVECT_U) begin

end

if(ITP_count == 4'd0 | ITP_count == 4'd6) U0_ot <= U0_ot_w;

else U0_ot <= U0_ot;

else if (state == ADVECT_V | state == ADVECT_D1 | state ==
ADVECT_D2) begin

if(ITP_count == 4'd0) U0_ot <= U0_ot_w;

else U0_ot <= U0_ot;

end

end

end

else U0_ot <= U0_ot;

always_ff @(posedge clk50 or posedge reset) begin //u0


if (reset) u0 <= 36'd0;


else begin
```

```
if(state == ADVECT_U && ITP_count == 4'd11) u0 <= u1_w;

else u0 <= u0;

end

end

RAM_U2

RAM_U2(.address_a(U_addr_a2), .address_b(U_addr_b2), .clock(cl
k50), .data_a(U_da2),

.data_b(U_db2), .wren_a(U_wen_a2), .wren_b(U_wen_b2), .q_a(U_qa2
), .q_b(U_qb2));

RAM_V2

RAM_V2(.address_a(V_addr_a2), .address_b(V_addr_b2), .clock(cl
k50), .data_a(V_da2),

.data_b(V_db2), .wren_a(V_wen_a2), .wren_b(V_wen_b2), .q_a(V_qa2
), .q_b(V_qb2));
```

```
RAM_D2

RAM_D2(.address_a(D_addr_a2), .address_b(D_addr_b2), .clock(cl
k50), .data_a(D_da2),

.data_b(D_db2), .wren_a(D_wen_a2), .wren_b(D_wen_b2), .q_a(D_qa2
), .q_b(D_qb2));

////////////////////////////////------ V_RAM ------
////////////////////////////////

IDLE: V_addr_a1 <= 13'd0;

always_ff @(posedge clk50 or posedge reset) begin //V_addr_a1

if (reset) V_addr_a1 <= 13'd0;

else begin

case(state)

ADVECT_U: begin

if(ITP_count == 4'd0) V_addr_a1 <= V_temp_1 ;
```

```
else if(ITP_count == 4'd1) V_addr_a1 <= V_temp_3 ;

else V_addr_a1 <= V_addr_a1 ;

end

ADVECT_V: begin

if(ITP_count == 4'd0) V_addr_a1 <= V_temp_1 ;

else if(ITP_count == 4'd1) V_addr_a1 <= V_temp_3 ;

else if(ITP_count == 4'd6) V_addr_a1 <= V_temp_1 ; else
if(ITP_count == 4'd7) V_addr_a1 <= V_temp_3 ;

else V_addr_a1 <= V_addr_a1 ;

end

ADVECT_D1: begin

if(ITP_count == 4'd0) V_addr_a1 <= V_temp_1 ;

else if(ITP_count == 4'd1) V_addr_a1 <= V_temp_3 ;
```

```
else V_addr_a1 <= V_addr_a1;

end

ADVECT_D2: begin

if(ITP_count == 4'd0) V_addr_a1 <= V_temp_1 ;

else if(ITP_count == 4'd1) V_addr_a1 <= V_temp_3 ;

else V_addr_a1 <= V_addr_a1;

end

default: V_addr_a1 <= 13'd0;

endcase

end

end

assign V_dir = {5'd0,I_count} * 13'd66 + {5'd0,J_count};

always_ff @(posedge clk50 or posedge reset) begin //V_addr_b1
```

```
if (reset) V_addr_b1 <= 13'd0;

else begin

case(state)

IDLE: V_addr_b1 <= 13'd0;

ADVECT_U: begin

end

if(ITP_count == 4'd0) V_addr_b1 <= V_temp_2 ;

else if(ITP_count == 4'd1) V_addr_b1 <= V_temp_4 ;

else V_addr_b1 <= V_addr_b1 ;

ADVECT_V: begin

if(ITP_count == 4'd0) V_addr_b1 <= V_temp_2 ;

else if(ITP_count == 4'd1) V_addr_b1 <= V_temp_4 ;
```

```
else if(ITP_count == 4'd6) V_addr_b1 <= V_temp_2 ; else

if(ITP_count == 4'd7) V_addr_b1 <= V_temp_4 ;

else V_addr_b1 <= V_addr_b1 ;

end

CORRECTION: begin


end

if(ITP_count == 4'd4) V_addr_b1 <= V_dir ; //write back


else V_addr_b1 <= V_addr_b1;


ADVECT_D1: begin


if(ITP_count == 4'd0) V_addr_b1 <= V_temp_2 ;


else if(ITP_count == 4'd1) V_addr_b1 <= V_temp_4 ;


else V_addr_b1 <= V_addr_b1 ;


end
```

```
ADVECT_D2: begin

if(ITP_count == 4'd0) V_addr_b1 <= V_temp_2 ;

else if(ITP_count == 4'd1) V_addr_b1 <= V_temp_4 ;

else V_addr_b1 <= V_addr_b1 ;

end

default: V_addr_b1 <= 13'd0;

endcase

end

end

assign V_wen_a1 = 1'd0; //V_wen_a1 always_comb begin
//V_wen_b1

if(state == CORRECTION) begin

if(ITP_count == 4'd5) V_wen_b1 = 1'd1;
```

```
else V_wen_b1 = 1'd0;

end

end

else V_wen_b1 = 1'd0;

assign V_da1 = 36'd0; //V_da1 (no use)

always_comb begin //V_db1

if (state == CORRECTION) begin

if(ITP_count == 4'd5) begin //write back

case(state)

V_db1 = c0_v;

end

else V_db1 = 36'd0;

end
```

```
end

else V_db1 = 36'd0;


////////////////////////////////------ V_RAM2 ------
////////////////////////////////

IDLE: V_addr_a2 <= 13'd0;

always_ff @(posedge clk50 or posedge reset) begin //V_addr_a2


if (reset) V_addr_a2 <= 13'd0;

else begin

PROJECT: begin


if(ITP_count == 4'd0) V_addr_a2 <= P_temp_v1 ;


else V_addr_a2 <= V_addr_a2;

end

CORRECTION: begin

end
```

```
if(ITP_count == 4'd0) V_addr_a2 <= C_temp_v ;

else V_addr_a2 <= V_addr_a2 ;

default: V_addr_a2 <= 13'd0;

endcase

end

end

always_ff @(posedge clk50 or posedge reset) begin //V_addr_b2

if (reset) V_addr_b2 <= 13'd0;

else begin

case(state)

IDLE: V_addr_b2 <= 13'd0;

ADVECT_V: begin
```

```
if(ITP_count == 4'd10) V_addr_b2 <= V_dir ;


end

assign V_wen_a2 = 1'd0; always_comb begin

assign V_da2 = 36'd0;

    //V_wen_a2
//V_wen_b2
else V_addr_b2 <= V_addr_b2 ;


end

PROJECT: begin

if(ITP_count == 4'd0) V_addr_b2 <= P_temp_v2 ;


else V_addr_b2 <= V_addr_b2;


end

end

default: V_addr_b2 <= 13'd0;


endcase
```

```
if(state == ADVECT_V) begin //write back after advect

if(ITP_count == 4'd11) V_wen_b2 = 1'd1;

else V_wen_b2 = 1'd0;

end

else V_wen_b2 = 1'd0;

end

////////////////////////////// ----- V_ans -----
////////////////////////////////////////

always_ff @(posedge clk50 or posedge reset) begin //V_ans_1

//V_da2 (no use)
always_comb begin //V_db2

if (state == ADVECT_V) begin

if(ITP_count == 4'd11) begin //write back

if(I_count == 8'd0 | I_count == 8'd64 ) //if it's boundry (i=0
or i=64),
set to zero
```

```
V_db2 = 36'd0;

end

else V_db2 = v1_w;

end

else V_db2 = 36'd0;

else V_db2 = 36'd0;

end

if (reset) V_ans_1 <= 36'd0;

else begin

if (state == ADVECT_V) begin

if(ITP_count == 4'd2 | ITP_count == 4'd8) V_ans_1 <= V_qa1;

else V_ans_1 <= V_ans_1;

end

else if (state == ADVECT_U | state == ADVECT_D1 | state ==
ADVECT_D2) begin
```

```
if(ITP_count == 4'd2) V_ans_1 <= V_qa1;

else V_ans_1 <= V_ans_1;

end

else if (state == PROJECT | state == CORRECTION) begin

if(ITP_count == 4'd2) V_ans_1 <= V_qa2; //read from V_RAM2

else V_ans_1 <= V_ans_1;

end

else V_ans_1 <= V_ans_1;

end

end

always_ff @(posedge clk50 or posedge reset) begin //V_ans_2

if (reset) V_ans_2 <= 36'd0;

else begin

if (state == ADVECT_V) begin

if(ITP_count == 4'd2 | ITP_count == 4'd8) V_ans_2 <= V_qb1;

else V_ans_2 <= V_ans_2;

end
```

```
else if (state == ADVECT_U | state == ADVECT_D1 | state ==
ADVECT_D2) begin

if(ITP_count == 4'd2) V_ans_2 <= V_qb1;

end

else V_ans_2 <= V_ans_2;

else if (state == PROJECT) begin

if(ITP_count == 4'd2) V_ans_2 <= V_qb2; //read from V_RAM2

else V_ans_2 <= V_ans_2;

end

else V_ans_2 <= V_ans_2;

end

end

always_ff @(posedge clk50 or posedge reset) begin //V_ans_3

if (reset) V_ans_3 <= 36'd0;

else begin

if (state == ADVECT_V) begin

if(ITP_count == 4'd3 | ITP_count == 4'd9) V_ans_3 <= V_qa1;
```

```
else V_ans_3 <= V_ans_3;

end

else if (state == ADVECT_U | state == ADVECT_D1 | state ==
ADVECT_D2) begin

if(ITP_count == 4'd3) V_ans_3 <= V_qa1;

end

else V_ans_3 <= V_ans_3;

else if (state == PROJECT) begin

if(ITP_count == 4'd3) V_ans_3 <= V_qa2; //read from V_RAM2

else V_ans_3 <= V_ans_3;

end

else V_ans_3 <= V_ans_3;

end

end

always_ff @(posedge clk50 or posedge reset) begin //V_ans_4


if (reset) V_ans_4 <= 36'd0;


else begin
```

```
if (state == ADVECT_V) begin

if(ITP_count == 4'd3 | ITP_count == 4'd9) V_ans_4 <= V_qb1;

else V_ans_4 <= V_ans_4;

end

else if (state == ADVECT_U | state == ADVECT_D1 | state ==
ADVECT_D2) begin

if(ITP_count == 4'd3) V_ans_4 <= V_qb1;

else V_ans_4 <= V_ans_4;

end

else if (state == PROJECT) begin

if(ITP_count == 4'd3) V_ans_4 <= V_qb2; //read from V_RAM2

else V_ans_4 <= V_ans_4;

end

end

end

else V_ans_4 <= V_ans_4;
```

```
////////////////////////////////------ advectV ------
////////////////////////////////

always_ff @(posedge clk50 or posedge reset) begin //V0_i


yoffset

if (reset) V0_i <= 36'd0;


else begin


case(state)


INITIALIZE_V: V0_i <= {8'd0,1'd1,27'd0}; //for first case


ADVECT_V: begin


end

if(ITP_count == 4'd12) V0_i <= {I_count,1'd1,27'd0};
//FixedReal(i) +


else if(ITP_count == 4'd5) begin


end

ADVECT_U: begin
```

```
V0_i <= bi;

else V0_i <= V0_i;

IDLE: V0_i <= {8'd1,28'd0};

INITIALIZE_D: V0_i <= {8'd1,28'd0};

end

if(ITP_count == 4'd12) V0_i <= {I_count,28'd0}; // FixedReal(i),
for bj

else V0_i <= V0_i;

ADVECT_D1: begin

end

if(ITP_count == 4'd12) V0_i <= {I_count,28'd0}; // only for bj

else V0_i <= V0_i;

end

end
```

```
ADVECT_D2: begin

end

if(ITP_count == 4'd12) V0_i <= {I_count,28'd0}; // only for bj

else V0_i <= V0_i;

default: V0_i <= V0_i;

endcase

always_ff @(posedge clk50 or posedge reset) begin //V0_j

if (reset) V0_j <= 36'd0;

else begin

case(state)

INITIALIZE_V: V0_j <= {8'd1,28'd0}; // FixedReal(j)

IDLE: V0_j <= {8'd0,1'd1,27'd0};

INITIALIZE_D: V0_j <= {8'd1,28'd0};
```

```
ADVECT_V: begin

if(ITP_count == 4'd12) V0_j <= {J_count,28'd0}; // FixedReal(j)

else if(ITP_count == 4'd5) begin

V0_j <= bj;

end

end

else V0_j <= V0_j;

ADVECT_U: begin

if(ITP_count == 4'd12) V0_j <= {J_count,1'd1,27'd0}; //
FixedReal(j) +

xoffset

end

else V0_j <= V0_j;
```

```
ADVECT_D1: begin

if(ITP_count == 4'd12) V0_j <= {J_count,28'd0}; // only for bj

else V0_j <= V0_j;

end

ADVECT_D2: begin

if(ITP_count == 4'd12) V0_j <= {J_count,28'd0}; // only for bj

end

else V0_j <= V0_j;

default: V0_j <= V0_j;

endcase

end

end

`ifdef TEST
```

```
logic [7:0] V1_i_D, V1_j_D;

assign V0_i_D = V0_i[35:28]; assign V0_j_D = V0_j[35:28];

---

`endif

always_ff @(posedge clk50 or posedge reset) begin //V0_s

if (reset) V0_s <= 36'd0;

else begin

if (state == ADVECT_V) begin

if(ITP_count == 4'd0 | ITP_count == 4'd6) V0_s <= V0_s_w;

end

else V0_s <= V0_s;

else if (state == ADVECT_U | state == ADVECT_D1 | state ==
ADVECT_D2) begin
```

```
if(ITP_count == 4'd0) V0_s <= V0_s_w;

else V0_s <= V0_s;

end

else V0_s <= V0_s;

end

end

always_ff @(posedge clk50 or posedge reset) begin //V0_t

if (reset) V0_t <= 36'd0;

else begin

if (state == ADVECT_V) begin

end

if(ITP_count == 4'd0 | ITP_count == 4'd6) V0_t <= V0_t_w;

else V0_t <= V0_t;

else if (state == ADVECT_U | state == ADVECT_D1 | state ==

ADVECT_D2) begin
```

```
if(ITP_count == 4'd0) V0_t <= V0_t_w;

else V0_t <= V0_t;

end

end

end

else V0_t <= V0_t;

always_ff @(posedge clk50 or posedge reset) begin //V0_os

if (reset) V0_os <= 36'd0;

else begin

if (state == ADVECT_V) begin

end

if(ITP_count == 4'd0 | ITP_count == 4'd6) V0_os <= V0_os_w;

else V0_os <= V0_os;

else if (state == ADVECT_U | state == ADVECT_D1 | state ==
ADVECT_D2) begin

if(ITP_count == 4'd0) V0_os <= V0_os_w;

else V0_os <= V0_os;
```

```
end

end

end

else V0_os <= V0_os;

always_ff @(posedge clk50 or posedge reset) begin //V0_ot

if (reset) V0_ot <= 36'd0;

else begin

if (state == ADVECT_V) begin

end

if(ITP_count == 4'd0 | ITP_count == 4'd6) V0_ot <= V0_ot_w;

else V0_ot <= V0_ot;

else if (state == ADVECT_U | state == ADVECT_D1 | state ==
ADVECT_D2) begin

if(ITP_count == 4'd0) V0_ot <= V0_ot_w;

else V0_ot <= V0_ot;

end

end

end
```

```
else V0_ot <= V0_ot;

always_ff @(posedge clk50 or posedge reset) begin //v0

if (reset) v0 <= 36'd0;

else begin

end

if(state == ADVECT_V & ITP_count == 4'd11) v0 <= v1_w;

else v0 <= v0;

end

////////////////////////////////------ D_RAM1 ------
////////////////////////////////

always_ff @(posedge clk50 or posedge reset) begin //D_addr_a1

if (reset) D_addr_a1 <= 13'd0;

else begin

case(state)
```

```
ADVECT_D1: begin

if(ITP_count == 4'd6) D_addr_a1 <= D_temp_1 ;

else if(ITP_count == 4'd7) D_addr_a1 <= D_temp_3 ;

else D_addr_a1 <= D_addr_a1 ;

end

INITIALIZE_F1: D_addr_a1 <= 13'd0;

DISPLAY2: begin

if(hcount[0] == 1) D_addr_a1 <= D_addr_a1;

else if((hcount <= 263 ) & (vcount <= 131)) begin

end

if(J_count == 8'd1) D_addr_a1 <= D_addr_a1 + 13'd1;

else D_addr_a1 <= D_addr_a1;

else if(vcount <= 131 & hcount == 264) begin

if (I_count == 8'd0) D_addr_a1 <= D_addr_a1 - 13'd66;
```

```
end

end

end

end

else D_addr_a1 <= D_addr_a1;

else D_addr_a1 <= D_addr_a1;

default: D_addr_a1 <= 13'd0;

endcase

assign D_dir = {5'd0,I_count} * 13'd66 + {5'd0,J_count};

always_ff @(posedge clk50 or posedge reset) begin //D_addr_b1

if (reset) D_addr_b1 <= 13'd0;

else begin

case(state)

ADVECT_D1: begin

if(ITP_count == 4'd6) D_addr_b1 <= D_temp_2 ;

else if(ITP_count == 4'd7) D_addr_b1 <= D_temp_4 ;

else D_addr_b1 <= D_addr_b1 ;
```

```
end

ADVECT_D2: begin


default: D_addr_b1 <= 13'd0;


endcase

end

end


assign D_wen_a1 = 1'd0; always_comb  begin

endassign D_da1 = 36'd0;

    //D_wen_a1
//D_wen_b1
if(ITP_count == 4'd10) D_addr_b1 <= D_dir; //write back


else D_addr_b1 <= D_addr_b1 ;


end

if(state == ADVECT_D2) begin //write back after advect


if(ITP_count == 4'd11) D_wen_b1 = 1'd1;
```

```
else D_wen_b1 = 1'd0;

end

else D_wen_b1 = 1'd0;

always_comb begin //D_db1

if (state == ADVECT_D2) begin

if(ITP_count == 4'd11) D_db1 = d0_w;

end

end

else D_db1 = 36'd0;

else D_db1 = 36'd0;

always_ff @(posedge clk50 or posedge reset) begin //D_addr_a2

if (reset) D_addr_a2 <= 13'd0;
```

```
else begin

case(state)

ADVECT_D2: begin

if(ITP_count == 4'd6) D_addr_a2 <= D_temp_1 ;

else if(ITP_count == 4'd7) D_addr_a2 <= D_temp_3 ;

else D_addr_a2 <= D_addr_a2 ;

end

INITIALIZE_F2: D_addr_a2 <= 13'd0;

//D_da1 (no use)
////////////////////////////////------ D_RAM2 ------
////////////////////////////////

DISPLAY1: begin

if (endOfField) D_addr_a2 <= 13'd0;

else if(hcount[0] == 1) D_addr_a2 <= D_addr_a2;

else if((hcount <= 263 ) & (vcount <= 131)) begin
```

```
end
if(J_count == 8'd1) D_addr_a2 <= D_addr_a2 + 13'd1;
else D_addr_a2 <= D_addr_a2;
else if(vcount <= 131 & hcount == 264) begin
if (I_count == 8'd0) D_addr_a2 <= D_addr_a2 - 13'd66;
else D_addr_a2 <= D_addr_a2;
end
else D_addr_a2 <= D_addr_a2;
end
default: D_addr_a2 <= 13'd0;
endcase
end
end
always_ff @(posedge clk50 or posedge reset) begin //D_addr_b2
if (reset) D_addr_b2 <= 13'd0;
else begin
case(state)
```

```
ADVECT_D1: begin

if(ITP_count == 4'd10) D_addr_b2 <= D_dir; //write back

else D_addr_b2 <= D_addr_b2 ;

end

end

end

ADVECT_D2: begin


endcase

if(ITP_count == 4'd6) D_addr_b2 <= D_temp_2 ;


else if(ITP_count == 4'd7) D_addr_b2 <= D_temp_4 ;


else D_addr_b2 <= D_addr_b2 ;


end


default: D_addr_b2 <= 13'd0;


assign D_wen_a2 = 1'd0; //D_wen_a2 always_comb begin
//D_wen_b2
```

```
if(state == ADVECT_D1) begin //write back after advect

if(ITP_count == 4'd11) D_wen_b2 = 1'd1;

else D_wen_b2 = 1'd0;

end

else D_wen_b2 = 1'd0;

end

assign D_da2 = 36'd0; //D_da2 (no use)

always_comb begin //D_db2

if (state == ADVECT_D1) begin

end

if(ITP_count == 4'd11) D_db2 = d0_w;

else D_db2 = 36'd0;
```

```
else D_db2 = 36'd0;


end

////////////////////////////////////// --- ANS_D -----
////////////////////////////////////

always_ff @(posedge clk50 or posedge reset) begin //D_ans_1

if (reset) D_ans_1 <= 36'd0;

else begin

if (state == ADVECT_D1) begin

end

if(ITP_count == 4'd8) D_ans_1 <= D_qa1;

else D_ans_1 <= D_ans_1;

else if (state == ADVECT_D2) begin

if(ITP_count == 4'd8) D_ans_1 <= D_qa2;

else D_ans_1 <= D_ans_1;

end

end

end
```

```
else D_ans_1 <= D_ans_1;

always_ff @(posedge clk50 or posedge reset) begin //D_ans_2

if (reset) D_ans_2 <= 36'd0;

else begin

if (state == ADVECT_D1) begin

end

if(ITP_count == 4'd8) D_ans_2 <= D_qb1;

else D_ans_2 <= D_ans_2;

else if (state == ADVECT_D2) begin

if(ITP_count == 4'd8) D_ans_2 <= D_qb2;

else D_ans_2 <= D_ans_2;

end

end

end

else D_ans_2 <= D_ans_2;

always_ff @(posedge clk50 or posedge reset) begin //D_ans_3

if (reset) D_ans_3 <= 36'd0;
```

```
else begin

if (state == ADVECT_D1) begin

end

if(ITP_count == 4'd9) D_ans_3 <= D_qa1;

else D_ans_3 <= D_ans_3;

else if (state == ADVECT_D2) begin

if(ITP_count == 4'd9) D_ans_3 <= D_qa2;


else D_ans_3 <= D_ans_3;


end


end

else D_ans_3 <= D_ans_3;


end


always_ff @(posedge clk50 or posedge reset) begin //D_ans_4

if (reset) D_ans_4 <= 36'd0;

else begin
```

```
if (state == ADVECT_D1) begin

end

if(ITP_count == 4'd9) D_ans_4 <= D_qb1;

else D_ans_4 <= D_ans_4;

else if (state == ADVECT_D2) begin

if(ITP_count == 4'd9) D_ans_4 <= D_qb2;

else D_ans_4 <= D_ans_4;

end

end

end

else D_ans_4 <= D_ans_4;

////////////////////////////////------ advectD ------
////////////////////////////////

always_ff @(posedge clk50 or posedge reset) begin //D0_i

if (reset) D0_i <= 36'd0;

else begin

if(state == ADVECT_D1 | state == ADVECT_D2) begin

if(ITP_count == 4'd5) begin
```

```
D0_i <= bi;
end
else D0_i <= D0_i;
end
else D0_i <= D0_i;
end
end
always_ff @(posedge clk50 or posedge reset) begin //D0_j
if (reset) D0_j <= 36'd0;
else begin
if(state == ADVECT_D1 | state == ADVECT_D2) begin
if(ITP_count == 4'd5) begin
D0_j <= bj;
end
else D0_j <= D0_j;
end
else D0_j <= D0_j;
end
```

```
end
`ifdef TEST

logic [7:0] D0_i_D, D0_j_D;

assign D0_i_D = D0_i[35:28]; assign D0_j_D = D0_j[35:28];

---

`endif

always_ff @(posedge clk50 or posedge reset) begin //D0_s

if (reset) D0_s <= 36'd0;

else begin

if (state == ADVECT_D1 | state == ADVECT_D2) begin

end

if(ITP_count == 4'd6) D0_s <= D0_s_w;

else D0_s <= D0_s;

end
```

```
end

always_ff @(posedge clk50 or posedge reset) begin //D0_t

if (reset) D0_t <= 36'd0;

else begin

if (state == ADVECT_D1 |state == ADVECT_D2) begin

if(ITP_count == 4'd6) D0_t <= D0_t_w;

else D0_t <= D0_t;

end

end

end

always_ff @(posedge clk50 or posedge reset) begin //D0_os

if (reset) D0_os <= 36'd0;

else begin
```

```
if (state == ADVECT_D1 | state == ADVECT_D2) begin

if(ITP_count == 4'd6) D0_os <= D0_os_w;

else D0_os <= D0_os;

end

end

end

always_ff @(posedge clk50 or posedge reset) begin //D0_ot

if (reset) D0_ot <= 36'd0;

else begin

if (state == ADVECT_D1 | state == ADVECT_D2) begin

if(ITP_count == 4'd6) D0_ot <= D0_ot_w;

end
```

```
else D0_ot <= D0_ot;

end

end

always_ff @(posedge clk50 or posedge reset) begin //d0

if (reset) d0 <= 36'd0;

else begin

if((state == ADVECT_D1 | state == ADVECT_D2) & ITP_count ==
4'd11) d0 <= d0_w;

else d0 <= d0;

end

end

////////////////////////////////------ P_RAM ------
////////////////////////////////

always_ff @(posedge clk50 or posedge reset) begin //P_addr_a1
```

```
if (reset) P_addr_a1 <= 13'd0;

else begin

case(state)

PROJECT: begin

end

if(ITP_count == 4'd0) P_addr_a1 <= P_temp_p1 ;

else if(ITP_count == 4'd1) P_addr_a1 <= P_temp_p3 ;

else P_addr_a1 <= P_addr_a1 ;

CORRECTION: begin

if(ITP_count == 4'd0) P_addr_a1 <= C_temp_p1 ;

else if(ITP_count == 4'd1) P_addr_a1 <= C_temp_p3 ;

else P_addr_a1 <= P_addr_a1 ;
```

```
end

default: P_addr_a1 <= 13'd0;

endcase

end

end

assign P_dir = {5'd0,I_count} * 13'd66 + {5'd0,J_count};

always_ff @(posedge clk50 or posedge reset) begin //P_addr_b1

if (reset) P_addr_b1 <= 13'd1;

else begin

case(state)

PROJECT: begin

if(ITP_count == 4'd0) P_addr_b1 <= P_temp_p2 ;

end
```

```
else if(ITP_count == 4'd1) P_addr_b1 <= P_temp_p4;

else if(ITP_count == 4'd4) P_addr_b1 <= P_dir; //write

else P_addr_b1 <= P_addr_b1 ;

CORRECTION: begin

end

if(ITP_count == 4'd0) P_addr_b1 <= C_temp_p2 ;

else if(ITP_count == 4'd1) P_addr_b1 <= C_temp_p4;

else P_addr_b1 <= P_addr_b1 ;

ADVECT_D1: begin

end

if(ITP_count == 4'd0) P_addr_b1 <= P_dir; //clear the P_RAM

else P_addr_b1 <= P_addr_b1 ;
```

```
ADVECT_D2: begin

if(ITP_count == 4'd0) P_addr_b1 <= P_dir; //clear the P_RAM

end

end

else P_addr_b1 <= P_addr_b1 ;

end

default: P_addr_b1 <= 13'd0;

endcase

assign P_wen_a1 = 1'd0; //P_wen_a1

assign P_da1 = 36'd0; //P_da1 (no use)

always_comb begin //P_wen_b1

if(state==PROJECT&ITP_count==4'd5)P_wen_b1=1'd1;

//writebackafteradvect else if((state == ADVECT_D1 |state ==

ADVECT_D2) & ITP_count == 4'd1) P_wen_b1 =
```

```
1'd1; //clear

else P_wen_b1 = 1'd0;

end

always_comb begin //P_db1

if(state == PROJECT & ITP_count == 4'd5) P_db1 = p0_w;

else if((state == ADVECT_D1 | state == ADVECT_D2) ) P_db1 = 36'd0;
//input 0

else P_db1 = 36'd0;

end

always_ff @(posedge clk50 or posedge reset) begin //P_ans_1

if (reset) P_ans_1 <= 36'd0;

else begin

if (state == PROJECT | state == CORRECTION) begin
```

```
if(ITP_count == 4'd2) P_ans_1 <= P_qa1;

end

else P_ans_1 <= P_ans_1;

else P_ans_1 <= P_ans_1;

end

end

always_ff @(posedge clk50 or posedge reset) begin //P_ans_2

if (reset) P_ans_2 <= 36'd0;

else begin

if (state == PROJECT | state == CORRECTION) begin

if(ITP_count == 4'd2) P_ans_2 <= P_qb1;

end

else P_ans_2 <= P_ans_2;

else P_ans_2 <= P_ans_2;

end

end

always_ff @(posedge clk50 or posedge reset) begin //P_ans_3
```

```
if (reset) P_ans_3 <= 36'd0;

else begin

if (state == PROJECT | state == CORRECTION) begin

if(ITP_count == 4'd3) P_ans_3 <= P_qa1;

end

else P_ans_3 <= P_ans_3;

else P_ans_3 <= P_ans_3;

end

end

always_ff @(posedge clk50 or posedge reset) begin //P_ans_4

if (reset) P_ans_4 <= 36'd0;

else begin

if (state == PROJECT | state == CORRECTION) begin

if(ITP_count == 4'd3) P_ans_4 <= P_qb1;

end

else P_ans_4 <= P_ans_4;

else P_ans_4 <= P_ans_4;
```

```
end

end

////////////////////////////////------ PROJECT ------
////////////////////////////////

always_ff @(posedge clk50 or posedge reset) begin //P0_i

if (reset) P0_i <= 8'd1;

else begin

if(state == INITIALIZE_P) P0_i <= 8'd1;

else if (state == PROJECT) begin

if(ITP_count == 4'd6) P0_i <= I_count;

else P0_i <= P0_i;

end

else P0_i <= P0_i;

end

end

always_ff @(posedge clk50 or posedge reset) begin //P0_j

if (reset) P0_j <= 8'd1;

else begin
```

```
if(state == INITIALIZE_P) P0_j <= 8'd1;

else if (state == PROJECT) begin

if(ITP_count == 4'd6) P0_j <= J_count;

else P0_j <= P0_j;

end

else P0_j <= P0_j;

end

always_comb begin //project


P0_i1 = {5'd0,P0_i};


P0_i2 = P0_i1 + 13'd1; P0_i3 = P0_i1 - 13'd1;

---


P0_j1 = {5'd0,P0_j};


P0_j2 = P0_j1 - 13'd1;


P0_j3 = P0_j1 + 13'd1;
```

```
P_temp_v1 = P0_i1 * 13'd66 + P0_j1; //v(i, j)

P_temp_v2 = P0_i3 * 13'd66 + P0_j1; //v(i-1, j)

P_temp_u1 = P0_i1 * 13'd65 + P0_j1; //u(i, j)

P_temp_u2 = P0_i1 * 13'd65 + P0_j2; //u(i, j-1)

P_temp_p1 = P_temp_v2; //p(i-1, j)

P_temp_p2 = P0_i2 * 13'd66 + P0_j1; //p(i+1, j) P_temp_p3 =

P0_i1 * 13'd66 + P0_j3; //p(i, j-1) P_temp_p4 = P0_i1 * 13'd66

+ P0_j2; //p(i, j+1)

endassign numer1 = V_ans_1 - V_ans_2 + U_ans_1 - U_ans_2;

always_comb begin

if(P0_i == 8'd1) begin // i == 1

if(P0_j == 8'd1) begin // j == 1

numer2=numer1-P_ans_2-P_ans_4;
//numer=numer-m_p(i-1,j)-m_p(i,j-1)

denom = 2'd2;

end
```

```
else if(P0_j == 8'd64) begin // j == 64

numer2=numer1-P_ans_2-P_ans_3;
//numer=numer-m_p(i-1,j)-m_p(i,j+1)

denom = 2'd2;

end

else begin

numer2 = numer1 - P_ans_2 - P_ans_3 - P_ans_4; //numer = numer
- m_p(i-1, j)

denom = 2'd1;

end

end

else if(P0_i == 8'd64) begin // i == 64

if(P0_j == 8'd1) begin // j == 1

numer2 = numer1 - P_ans_1 - P_ans_4; //numer = numer - m_p(i+1,
j) - m_p(i, j-1)

denom = 2'd2;

end

else if(P0_j == 8'd64) begin // j == 64
```

```
numer2 = numer1 - P_ans_1 - P_ans_3; //numer = numer - m_p(i+1,
j) - m_p(i, j+1)

denom = 2'd2;

end

else begin

numer2 = numer1 - P_ans_1 - P_ans_3 - P_ans_4; //numer = numer
- m_p(i+1, j)

denom = 2'd1;

end

end

else begin

if(P0_j == 8'd1) begin // j == 1

numer2 = numer1 - P_ans_1 - P_ans_2 - P_ans_4; //numer = numer
- m_p(i, j-1)

denom = 2'd1;

end

else if(P0_j == 8'd64) begin // j == 64

numer2 = numer1 - P_ans_1 - P_ans_2 - P_ans_3; //numer = numer
- m_p(i, j+1)

end
```

```
denom = 2'd1;

end

else begin

numer2=numer1-P_ans_1-P_ans_2-P_ans_3-P_ans_4;
//numer=numer-m_p(i-1,

j) - m_p(i, j-1)
denom = 2'd0;

end

end

numer3 = ~numer2 + 36'd1;

case(denom)

2'd0: begin

p0_w = (numer3>>>14)<<12;

end
```

```
2'd1: begin

p0_w = (numer3>>>14) * {22'd0,14'b01_0101_0101_0101};

end

2'd2: begin


p0_w = (numer3>>>14)<<13;

end

default: p0_w = numer3;

endcase

always_ff @(posedge clk50 or posedge reset) begin //p0

if (reset) p0 <= 36'd0;

else begin

if(state == PROJECT && ITP_count == 4'd5) p0 <= p0_w;
```

```
else p0 <= p0;

end

end

////////////////////////////////------ CORRECTION ------
/////////////////////////////

always_ff @(posedge clk50 or posedge reset) begin //C0_i

if (reset) C0_i <= 8'd1;

else begin

if(state == INITIALIZE_C) C0_i <= 8'd1;

else if (state == CORRECTION) begin

if(ITP_count == 4'd6) C0_i <= I_count;

else C0_i <= C0_i;

end

else C0_i <= C0_i;

end

end

always_ff @(posedge clk50 or posedge reset) begin //C0_j
```

```
if (reset) C0_j <= 8'd1;

else begin

if(state == INITIALIZE_C) C0_j <= 8'd1;

else if (state == CORRECTION) begin

always_comb begin

if(ITP_count == 4'd6) C0_j <= J_count;

else C0_j <= C0_j;

end

end

end

else C0_j <= C0_j;

C0_i1 = {5'd0,C0_i};

C0_i2 = C0_i1 + 13'd1;
```

```
C0_j1 = {5'd0,C0_j};

C0_j2 = C0_j1 + 13'd1;

C_temp_v = C0_i1 * 13'd66 + C0_j1; //v(i, j) C_temp_u = C0_i1
* 13'd65 + C0_j1; //u(i, j)

---

C_temp_p1 = C0_i1 * 13'd66 + C0_j2; //p(i, j+1)

C_temp_p2 = C_temp_v; //p(i, j)

C_temp_p3 = C0_i2 * 13'd66 + C0_j1; //p(i+1, j)

C_temp_p4 = C_temp_p2;

end

////////////////////////////////---- interpolateV0 ----
//////////////////////////// //8-bit integer 26-bit
fraction always_comb begin //interpolateV block, input:
V0_i, V0_j, output: v0_w

assign c0_u = U_ans_1 - (P_ans_1 - P_ans_2); assign c0_v = V_ans_1
- (P_ans_3 - P_ans_4);
```

```
V0_itemp = V0_i - 36'd134217728; // V0_i - FixedReal(0.5f)
// FixedReal(0.5f) = {8'b0,1'b1,27'b0} = 36'd134217728
if(V0_itemp[35] == 1'b1 ) V0_i1 = 13'd0;
else if (V0_itemp[35:28] >= 8'd63) V0_i1 = 13'd63; //i1 = 0~63
else V0_i1 = {5'd0,V0_itemp[35:28]};
V0_i2 = V0_i1 + 13'd1;
if(V0_j[35] == 1'b1) V0_j1 = 13'd0;

else if (V0_j[35:28] >= 8'd64) V0_j1 = 13'd64; //j1 = 0~64

else V0_j1 = {5'd0,V0_j[35:28]};

V0_j2 = V0_j1 + 13'd1; //V0_j2 = 1~65

V_temp_1 = V0_i1 * 13'd66 + V0_j1;
end
V_temp_2 = V0_i2 * 13'd66 + V0_j1; V_temp_3 = V0_i1 * 13'd66
+ V0_j2; V_temp_4 = V0_i2 * 13'd66 + V0_j2;
```

```
V0_stemp = V0_itemp - {V0_i1[7:0],28'd0};

if (V0_stemp[35] == 1) V0_s_w = 36'd0;

else if (V0_stemp[35:28] >= 8'd1) V0_s_w = 36'd268435456;
//FixedReal(1)>>14

else V0_s_w = (V0_stemp);

V0_ttemp = V0_j - {V0_j1[7:0],28'd0};

if (V0_ttemp[35] == 1) V0_t_w = 36'd0 ;

else if (V0_ttemp[35:28] >= 8'd1) V0_t_w = 36'd268435456;
//FixedReal(1)>>14

else V0_t_w = (V0_ttemp);

V0_os_w = (36'd268435456 - V0_s_w);

V0_ot_w = (36'd268435456 - V0_t_w);
```

```
assign insertV0_1=(((V_ans_1>>>14)*(V0_os>>>14)
+(V_ans_2>>>14)*(V0_s>>>14))
>>>14 ) * (V0_ot>>>14);

assign insertV0_2=(((V_ans_3>>>14)*(V0_os>>>14)
+(V_ans_4>>>14)*(V0_s>>>14))
>>>14 ) * (V0_t>>>14);

assign v1_w = insertV0_1 + insertV0_2;

assign v0_w = (v1_w>>>14)<<13;

assign bi = V0_i - v0_w; //V0_i = FixedReal(i) in advectU & D,
V0_i = FixedReal(i) +
yoffset in advectV
//* {22'd0,14'b01_0101_0101_0101};
`ifdef TEST

logic [7:0] v0_w_D;

logic [7:0] inV0_1,inV0_2,inV0_3,inV0_4;

assign inV0_1 = V_ans_1[35:28]; assign inV0_2 = V_ans_2[35:28];
assign inV0_3 = V_ans_3[35:28]; assign inV0_4 = V_ans_4[35:28];
```

```
assign v0_w_D = v1_w[35:28];

`endif

////////////////////////////////---- interpolateU0 ----
////////////////////////////// always_comb begin
//interpolateU block, input: U0_i, U0_j, output: u0_w
//i

if(U0_i[35] == 1'b1) U0_i1 = 8'd0;

else if (U0_i[35:28] >= 8'd64) U0_i1 = 8'd64; //U0_i1 = 0~64

else U0_i1 = U0_i[35:28];

U0_i2 = U0_i1 + 8'd1; //U0_i1 = 1~65

//j

U0_jtemp = U0_j - 36'd134217728; // U0_j - FixedReal(0.5f)

end
```

```
if(U0_jtemp[35] == 1'b1) U0_j1 = 13'd0;

else if (U0_jtemp[35:28] >= 8'd63) U0_j1 = 13'd63; //U0_j1 = 0~63

else U0_j1 = {5'd0,U0_jtemp[35:28]};

U0_j2 = U0_j1 + 13'd1; //U0_j2 = 1~64

U0_i3 = U0_i1 * 8'd65; U0_i4 = U0_i2 * 8'd65;

---

U_temp_1 = U0_i3[12:0] + U0_j1; U_temp_2 = U0_i4[12:0] + U0_j1;

U_temp_3 = U0_i3[12:0] + U0_j2; U_temp_4 = U0_i4[12:0] + U0_j2;

---

//s

U0_stemp = U0_i - {U0_i1,28'd0};

if (U0_stemp[35] == 1) U0_s_w = 34'd0 ;

else if (U0_stemp[35:28] >= 8'd1) U0_s_w = 36'd268435456;
//FixedReal(1)>>13 2^28
```

```
else U0_s_w = (U0_stemp);

//t

U0_ttemp = U0_jtemp - {U0_j1[7:0],28'd0};

if (U0_ttemp[35] == 1) U0_t_w = 36'd0 ;

else if (U0_ttemp[35:28] >= 8'd1) U0_t_w = 36'd268435456;
//FixedReal(1)>>13

else U0_t_w = (U0_ttemp);

U0_os_w = (36'd268435456 - U0_s_w); U0_ot_w = (36'd268435456 -
U0_t_w);
```

```
assigninsertU0_1=(((U_ans_1>>>14)*(U0_os>>>14) +
(U_ans_2>>>14)*(U0_s>>>14))

>>>14  ) * (U0_ot>>>14);
```

```
assigninsertU0_2=(((U_ans_3>>>14)*(U0_os>>>14) +

(U_ans_4>>>14)*(U0_s>>>14))


>>>14  ) * (U0_t>>>14);


assign u1_w  = insertU0_1  + insertU0_2;


assign u0_w  = (u1_w>>>14)<<13;


assignbj=U0_j-u0_w;
//U0_j=FixedReal(j)+xoffseinadvectU,U0_j=FixedReal(j)

in advectV

`ifdef TEST


logic [7:0] U0_w_D;


logic [7:0] inU0_1,inU0_2,inU0_3,inU0_4;


logic [7:0] advectU_bj_D,advectV_bj_D;
```

```
assign inU0_1 = U_ans_1[35:28]; assign inU0_2 = U_ans_2[35:28];
assign inU0_3 = U_ans_3[35:28]; assign inU0_4 = U_ans_4[35:28];

---

assign u0_D = u0[35:28];

`endif

////////////////////////////////---- interpolateD ----
////////////////////////////// always_comb begin
//interpolateD block, input: D0_i, D0_j, output: d0_w
if(D0_i[35] == 1'b1) D0_i1 = 13'd0;

else if (D0_i[35:28] >= 8'd64) D0_i1 = 13'd64; //D0_i1 = 0~64

else D0_i1 = {5'd0,D0_i[35:28]};

D0_i2 = D0_i1 + 13'd1; //D0_i2 = 1~65

if(D0_j[35] == 1'b1) D0_j1 = 13'd0;

else if (D0_j[35:28] >= 8'd64) D0_j1 = 13'd64; //D0_j1 = 0~64
```

```
else D0_j1 = {5'd0,D0_j[35:28]};

D0_j2 = D0_j1 + 13'd1; //D0_j2 = 1~65

D_temp_1 = D0_i1 * 13'd66 + D0_j1; D_temp_2 = D0_i2 * 13'd66
+ D0_j1; D_temp_3 = D0_i1 * 13'd66 + D0_j2; D_temp_4 = D0_i2 *
13'd66 + D0_j2;


D0_stemp = D0_i - {D0_i1[7:0],28'd0};

if (D0_stemp[35] == 1) D0_s_w = 36'd0 ;

else if (D0_stemp[35:28] >= 8'd1) D0_s_w = 36'd268435456;

else D0_s_w = D0_stemp;

D0_ttemp = D0_j - {D0_j1[7:0],28'd0};

if (D0_ttemp[35] == 1) D0_t_w = 36'd0 ;
```

```
else if (D0_ttemp[35:28] >= 8'd1) D0_t_w = 36'd268435456;

else D0_t_w = D0_ttemp;

end

D0_os_w = (36'd268435456 - D0_s_w); D0_ot_w = (36'd268435456 -
D0_t_w);

---

assign insertD0_1 = ( ( (D_ans_1>>>14) * (D0_os>>>14) +
(D_ans_2>>>13) *

(D0_s>>>14) )>>>14 ) * (D0_ot>>>14);

assign insertD0_2 = ( ( (D_ans_3>>>14) * (D0_os>>>14) +
(D_ans_4>>>13) *

(D0_s>>>14) )>>>14 ) * (D0_t>>>14);

assign d0_w = insertD0_1 + insertD0_2;

`ifdef TEST
```

```
logic [7:0] d0_w_D;

logic [7:0] inD0_1,inD0_2,inD0_3,inD0_4;

assign inD0_1 = D_ans_1[35:28]; assign inD0_2 = D_ans_2[35:28];

assign inD0_3 = D_ans_3[35:28];

assign inD0_4 = D_ans_4[35:28];

assign d0_w_D = d0_w[35:28];

`endif

//////////////////////////////////// ----- VGA -----
////////////////////////////////////

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;
```

```
assign VGA_HS = !( (hcount[10:7] == 4'b1010) & (hcount[6] |
hcount[5]));

assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

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

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

always_comb begin

VGA_B = 8'hff; VGA_G = 8'hff;

if(state == DISPLAY1) begin

if ((hcount <= 263 ) & (vcount <= 131)) begin

VGA_B = 8'h80; VGA_G = 8'h80;
```

```
if(D_qa2[29]) VGA_R = 8'd255;

else VGA_R = D_qa2[28:21];

end

else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};

end

else if(state == DISPLAY2) begin

if ((hcount <= 263 ) & (vcount <= 131)) begin

VGA_B = 8'h80; VGA_G = 8'h80;

if(D_qa1[29]) VGA_R = 8'd255;

else VGA_R = D_qa1[28:21];

end

else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};

end

else begin

if ((hcount <= 263 ) & (vcount <= 131)) begin

VGA_B = 8'h80; VGA_G = 8'h80;

VGA_R = Dt_q;
```

```
end

end

end

else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};

endmodule
```