EEE102 PROJECT FINAL

SIMULATION OF GRAVITATION

Student Name: Yiğit Türkmen

Department: EE & Phys

ID: 22102518

Course Code: EEE-102

Section: 01

Video Link:

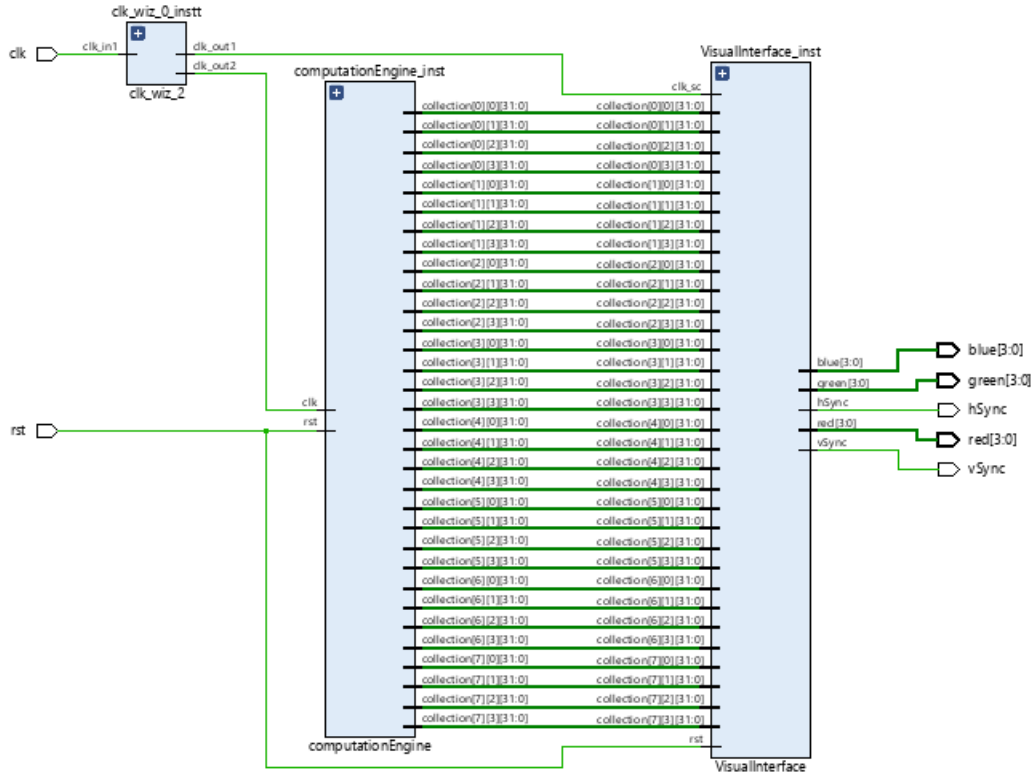
https://www.youtube.com/watch?v=3TJQeo8Bq1s&t=5s&ab_channel=Yi%C4%9FitT%C3%BCrkmen

Description of the project:

In this project, the aim was creating gravity simulation up to 32 particles by showing it on VGA monitor using only FPGA and VHDL. However, even though simulations were accurate and convenient, monitor didn't show that particles moving because, my board Basys-3 was not suitable for to do all calculations there was not enough hardware in Basys-3 for this purpose. However, I simulated the same simulation with exactly same logic using python and expected results can be seen in the video. Since very few pages report is wanted I don't go into details in the report, I think I explained very well in the video.

Design of the Project:

My design includes two main modules: ComputationEngine and VisualInterface.



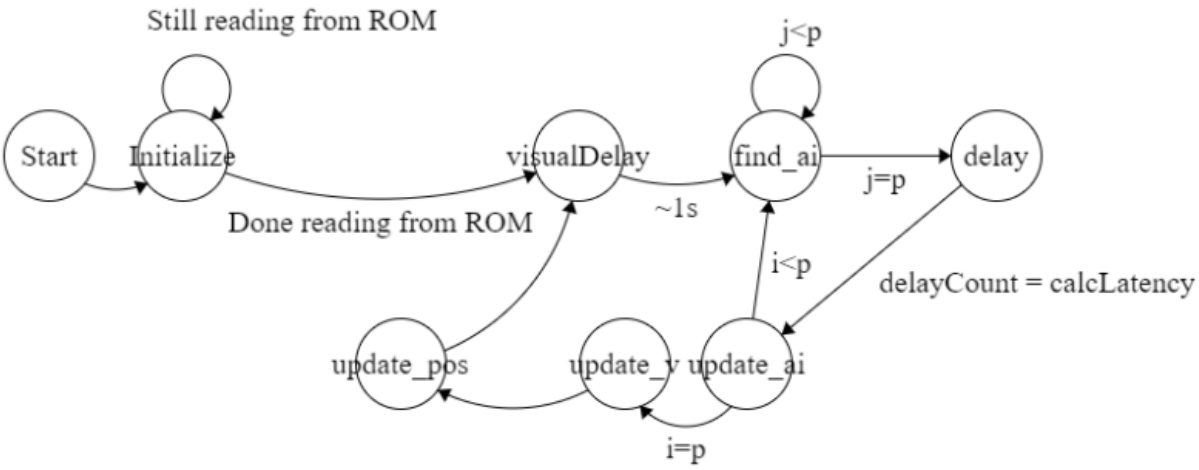
RTL schematic.

ComputationEngine (1):

Role of computatioEngine is calculating and updating new positions of particles and then sending updated collectionList to the VisualInterface.

StateMachine:

I used state machine for this purpose which is shown below. I explained it in the video.

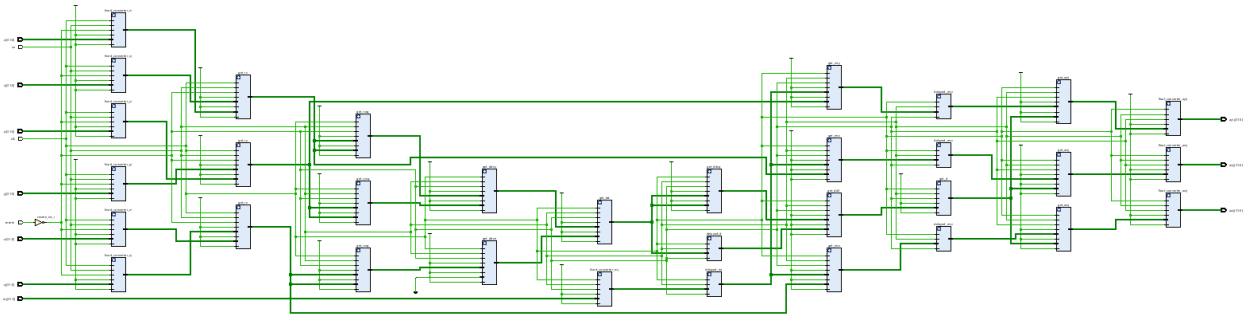


1.1 AccelerationCalculator:

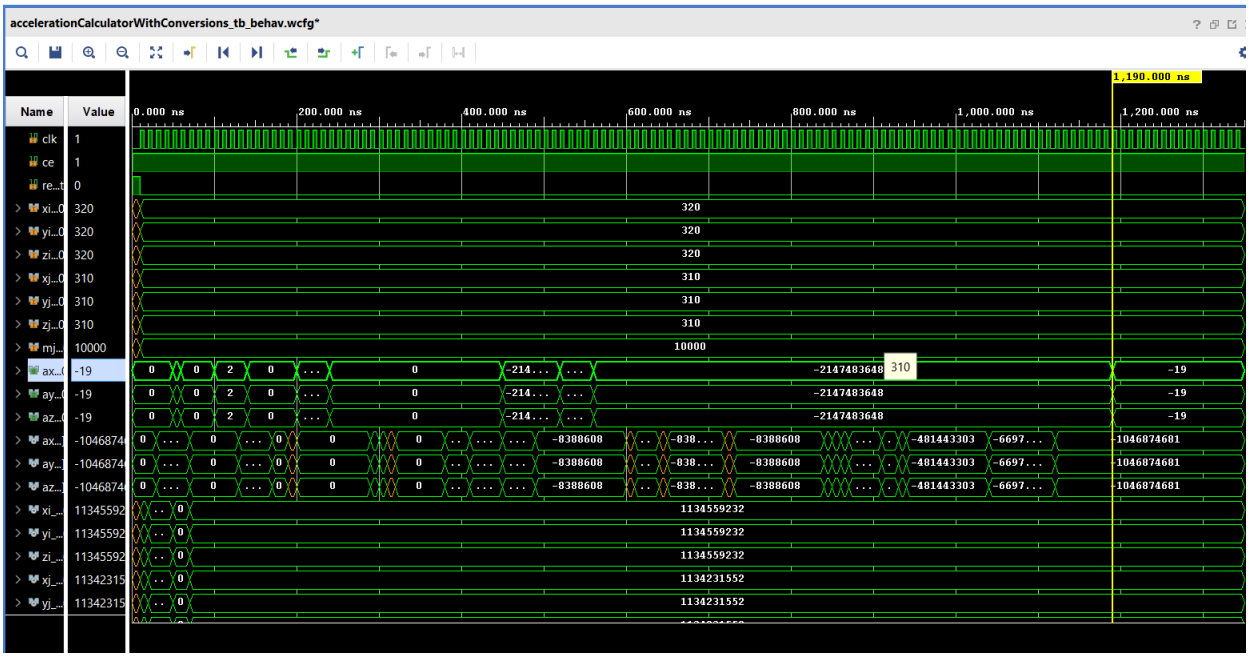
This component was responsible for calculating acceleration occurred on i by j. Floating point ip cores were responsible for this purpose.

```
for (i = 0; i < N; i++) {
   for (j = 0; j < N; j++) {
      if (j != i) {

         rx = p[j].x - p[i].x;
         ry = p[j].y - p[i].y;
         rz = p[j].z - p[i].z;

         dd = rx*rx + ry*ry + rz*rz + EPS;
         d = 1 / sqrtf(dd * dd * dd);

         s = p[j].m * d;

         a[i].x += rx * s;
         a[i].y += ry * s;
         a[i].z += rz * s;
      }
   }
}
```

One iteration of this pseudo-code is what AccelerationCalculator does.



RTL schematic of accCalculator, each of these components are floating point ip cores.



It takes 119 clock cycles to calculate accelerations for one iteration. As it can be seen in the simulation.

1.2 ROM:

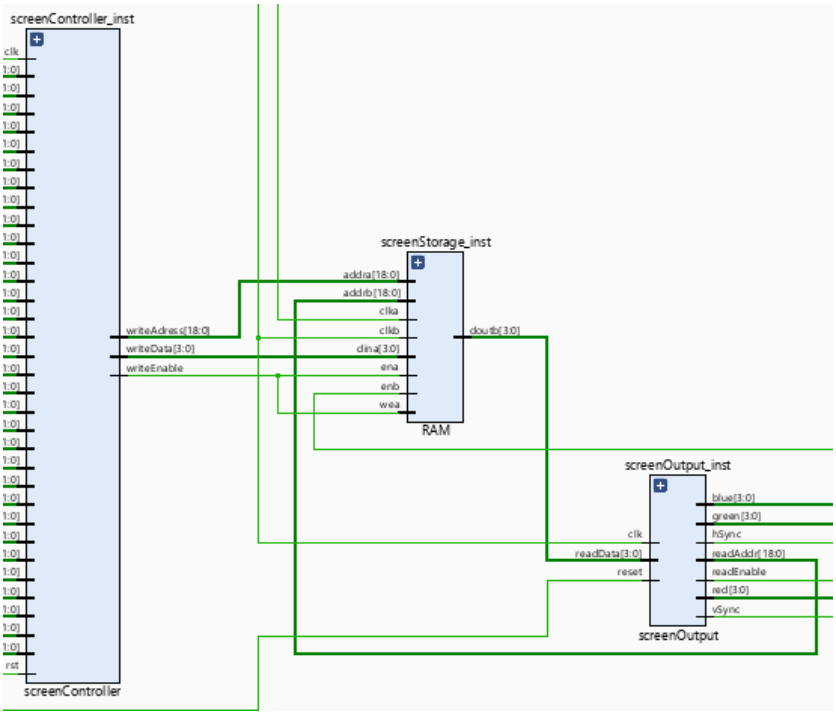
Rom was responsible for taking initial data from the initial.coe. Since hardware of Basys3 is not much good, only 32 particles could be shown on the screen due to the ROM.

```
memory_initialization_radix=16;
memory_initialization_vector=
00000140000001400000000000003E8,
00000136000001360000000000003E8,
0000012A0000012A00000000000003E8,
0000011A0000011A00000000000003E8,
00000101000001010000000000003E8,
00000000000000000000000000000000,
00000000000000000000000000000000,
00000000000000000000000000000000;
```

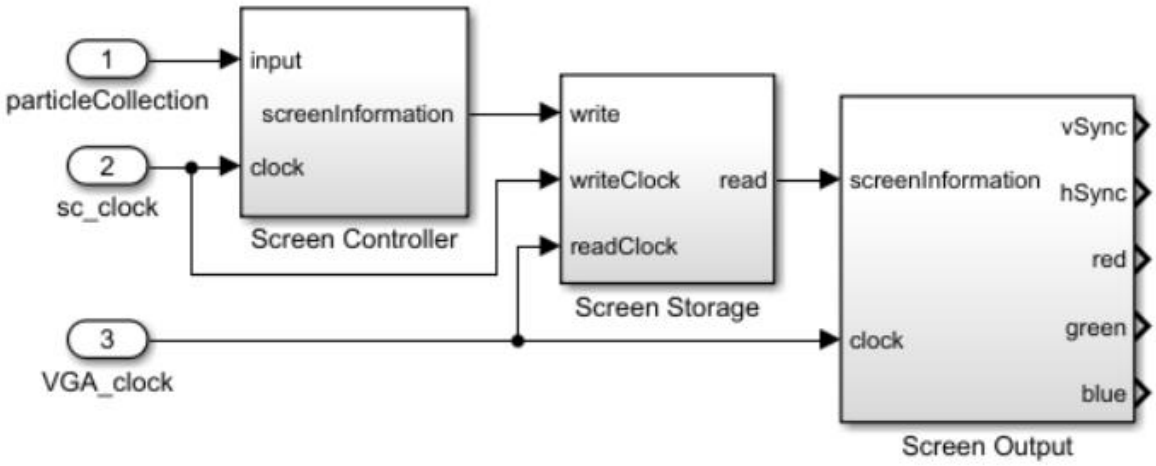
Initial.coe.

VisualInterface (2):

VisualInterface was basically responsible for giving particles a shape in the monitor and creating images on the monitor.



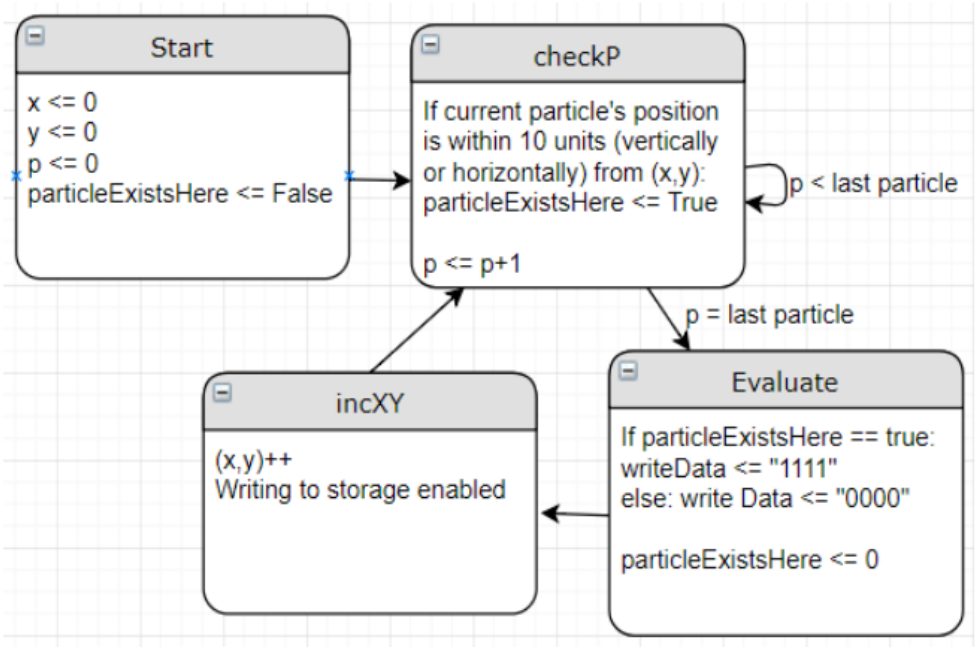
RTL schematic of VisualInterface.



Block diagram for VisualInterface.

2.1 ScreenController:

The screen controller's role involves assigning and storing pixel values on the output screen according to the data received from the particle collection. State diagram utilized for this purpose which is shown below.



2.2

ScreenStorage:

RAM is utilized for screenstorage. The RAM unit manages data writing and flow, whereas the output unit oversees the data reading process.

2.3 ScreenOutput:

The VGA timing controller is the core of the screen output unit. It traverses each pixel on the screen, generating sync signals for monitor alignment. Its current position becomes the read address for the screen storage, controlling system outputs such as VGA sync signals and color values.

Components:

-Basys3 board.

-VGA monitor.

Conclusion:

It was extremely hard project to do but I think I managed it well, I learned lots of things about FPGA and VHDL, and generally to speak my perception is increased.

Important Codes:

ConstantList.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

PACKAGE constantList is
--Particle constants
constant numberOfParticles : INTEGER := 32;
constant xComponent : INTEGER :=0;
constant yComponent  : INTEGER :=1;
constant zComponent  : INTEGER :=2;
constant massComponent  : INTEGER :=3;

--Width of each particle on the screen
constant particleDisplayWidth : INTEGER :=3;

--640x480 basys3
    --Horizontal
constant H_DISPLAY_cste : INTEGER := 640; --Nb Active pixels per line
constant H_FP_cste : INTEGER := 16; --Nb clocks front proch
constant H_PULSE_cste : INTEGER := 96; --Nb clocks horizontal proch
constant H_BP_cste : INTEGER := 48; --Nb clocks back proch
constant H_SIZE : INTEGER := 10;
    --Vertical
constant V_DISPLAY_cste : INTEGER := 480; --Nb Active line per frame
constant V_FP_cste : INTEGER := 10; --Nb lines front proch
constant V_PULSE_cste : INTEGER := 2; --Nb lines horizontal proch
constant V_BP_cste : INTEGER := 33; --Nb lines back proch
constant V_SIZE: INTEGER := 9;
--VGA computations
constant H_START_PULSE_cste : INTEGER := H_DISPLAY_cste + H_FP_cste;
constant H_END_PULSE_cste : INTEGER := H_START_PULSE_cste + H_PULSE_cste ;
```



```
constant V_START_PULSE_cste : INTEGER := V_DISPLAY_cste + V_FP_cste ;
constant V_END_PULSE_cste : INTEGER := V_START_PULSE_cste + V_PULSE_cste;
constant H_PERIOD_cste : INTEGER := H_DISPLAY_cste + H_FP_cste + H_PULSE_cste + H_BP_cste ;
constant V_PERIOD_cste : INTEGER := V_DISPLAY_cste + V_FP_cste + V_PULSE_cste + V_BP_cste ;

--used in the screen storage RAM
constant ADDR_WIDTH : INTEGER := 19; -- "-1" for rounding errors

--width of the x,y,z,mass data
constant componentWidth: INTEGER := 32;

--These latencies are recorded to obtain correct number of delays
constant subLatency : INTEGER := 12;
constant multLatency : INTEGER := 9;
constant addLatency : INTEGER := 12;
constant invSqRootLatency : INTEGER := 33;
constant fixedToFloatLatency : INTEGER := 7;
constant floatToFixedLatency : INTEGER := 7;

--Visual delay of 1 second, the "1/1" can be modified to increase/decrease the visual delay
constant compEngineFreq : INTEGER := 100000000;
constant visualDelayMax : INTEGER := compEngineFreq / 1;
end constantList;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
library work;
use work.constantList.all;

entity computationEngine is
 Port (clk: in std_logic;
       rst: in std_logic;
       collection: out particleList);
       --collectionUpdated: out std_logic );
end computationEngine;

architecture Behavioral of computationEngine is

component accelerationCalculatorWithConversions

Port(clk: in std_logic;
    ce: in std_logic;
    resetn: in std_logic;
    xi: in std_logic_vector(componentWidth-1 downto 0);
    yi:in std_logic_vector(componentWidth-1 downto 0);
    zi:in std_logic_vector(componentWidth-1 downto 0);
    xj:in std_logic_vector(componentWidth-1 downto 0);
    yj:in std_logic_vector(componentWidth-1 downto 0);
    zj:in std_logic_vector(componentWidth-1 downto 0);
    mj:in std_logic_vector(componentWidth-1 downto 0);
    axij:out std_logic_vector(componentWidth-1 downto 0);
    ayij:out std_logic_vector(componentWidth-1 downto 0);
    azij:out std_logic_vector(componentWidth-1 downto 0));

end component;
```

```
component ROM
port (clka: in std_logic;
    ena:in std_logic;
    addra: in std_logic_vector(4 downto 0);
    douta: out std_logic_vector(127 downto 0);
    clkb:in std_logic;
    enb:in std_logic;
    addrb: in std_logic_vector(4 downto 0);
    doutb: out std_logic_vector(127 downto 0));
end component;

constant accCalcLatency: INTEGER := subLatency + multLatency + addLatency +
    invSqRootLatency  + fixedToFloatLatency + floatToFixedLatency;

signal positionWithMass: particlelist;
signal velocity, acceleration: particleVectorList;

type state is (start, initialize, visualDelay, find_ai, update_ai, update_v, update_pos, delay);

signal pState: state;
signal i,j: INTEGER range 0 to (numberOfParticles - 1);
signal ROMen: std_logic;
--ROM was made for 32 particles, hence 5 bits for addressing
signal ROMaddra, ROMaddrb : std_logic_vector(4 DOWNTO 0);
signal ROMdouta, ROMdoutb : Std_logic_vector(4*componentWidth-1 downto 0);

signal initEN: std_logic;
constant initmax: INTEGER:= (numberOfParticles/2 - 1);
signal xi,yi,zi,xj,yj,zj,mj,axij,ayij,azij: std_logic_vector(componentWidth-1 downto 0);
signal accCalcReset: std_logic;
signal accCalcEn: std_logic;
signal delayCount: INTEGER range 0 to visualDelayMax;
```

1

```
signal axi, ayi, azi: SIGNED(componentWidth-1 downto 0);

begin
collection <= positionWithMass;

initializationStorage_Inst: ROM
port map(clka => clk,
    ena => ROMen,
    addra =>ROMaddra,
    douta => ROMdouta,
    clkb => clk,
    enb => ROMen,
    addrb => ROMaddrb,
    doutb => ROMdoutb);


accCalc_Inst: accelerationCalculatorWithConversions
port map ( clk => clk,
    ce=> accCalcEn,
    resetn=> accCalcReset,
    xi=>xi,
    yi=>yi,
    zi=>zi,
    xj=>xj,
    yj=>yj,
    zj=>zj,
    mj=>mj,
    axij=>axij,
    ayij=>ayij,
    azij=>azij);
```

2

```
ROMen <= '1' when pState = initialize else '0';
ROMaddra <= std_logic_vector(to_unsigned(i*2,5));
ROMaddrb <= std_logic_vector(to_unsigned(i*2 + 1,5));

xi <= positionWithMass(i)(xComponent) when pSTATE = find_ai
 else (others => '0');
yi <= positionWithMass(i)(yComponent) when pSTATE = find_ai
 else (others => '0');
zi <= positionWithMass(i)(zComponent) when pSTATE = find_ai
 else (others => '0');
xj <= positionWithMass(j)(xComponent) when pSTATE = find_ai
 else (others => '0');
yj <= positionWithMass(j)(yComponent) when pSTATE = find_ai
 else (others => '0');
zj <= positionWithMass(j)(zComponent) when pSTATE = find_ai
 else (others => '0');
mj <= positionWithMass(j)(massComponent) when pSTATE = find_ai
 else (others => '0');


accCalcEn <= '1' when pState = find_ai or pState = visualDelay or pState = delay else '0';
accCalcReset <= '1' when pState = start or pState = initialize else '0';

) process
 begin
    wait until rising_edge(clk);
)   if rst = '1' then
        pState <= start;
    else
)       case pState is
)       when start =>
```

3

```
when start =>

i <= 0;
j <= 0;
initEn <= '0';
delayCount <= 0;
velocity <= (others => (others => (others => '0')));

axi <= (others => '0');
ayi <= (others => '0');
azi <= (others => '0');

--collectionUpdated <= '0';  -- Reset the collectionUpdated signal

pState <= initialize;

when initialize =>
    --x(2j)
    if initEn = '1' then
    positionWithMass (2*j)(xComponent) <= ROMdouta (4*componentWidth-1 downto 3*componentWidth);
    --y(2j)
    positionWithMass (2*j)(yComponent) <= ROMdouta (3*componentWidth-1 downto 2*componentWidth);
    --z(2j)
    positionWithMass (2*j)(zComponent) <= ROMdouta (2*componentWidth-1 downto componentWidth);
    --m(2j)
    positionWithMass (2*j)(massComponent) <= ROMdouta (componentWidth-1 downto 0);

    --x(2j+1)
    positionWithMass (2*j+1)(xComponent) <= ROMdoutb (4*componentWidth-1 downto 3*componentWidth);
    --y(2j+1)
    positionWithMass (2*j+1)(yComponent) <= ROMdoutb (3*componentWidth-1 downto 2*componentWidth);
    --z(2j+1)
```

4

```
    positionWithMass (2*j+1)(zComponent) <= ROMdoutb (2*componentWidth-1 downto componentWidth);
    --m(2j+1)
    positionWithMass (2*j+1)(massComponent) <= ROMdoutb (componentWidth-1 downto 0);

    end if;

    -- after the first cycle, j is always one behind i
    -- this means j will write what i just read from
    -- initEn confirms that j won't write until
    -- after first cycle

    if (j < initMax) then
        i <= i + 1;
        j <= i;
        initEn <= '1';
        pState <= initialize;
    else
        i <= 0;
        j <= 0;
        initEn <= '0';
        pState <= visualDelay;

    end if;

when visualDelay =>

    if delayCount < VisualDelayMax then
        delayCount <= delayCount + 1;
        pState <= VisualDelay;
    else
        delayCount <= 0;
        pState <= find_ai;
```

5

```
            end if;

        --when find_ai =>
    -- Calculate acceleration components...
            --axi <= axi + SIGNED(axij);
            --ayi <= ayi + SIGNED(ayij);
            --azi <= azi + SIGNED(azij);
            -- Check if j is less than numberOfParticles - 1
           -- if j < numberOfParticles - 1 then
                -- If so, increment j and transition to delay_after_calculation state
               -- j <= j + 1;
              --  pState <= delay_after_calculation;
           -- else
                -- If not, reset j to 0 and transition to update_ai state
               -- j <= 0;
               -- pState <= update_ai;
           -- end if;

        when find_ai =>

          if j < numberOfParticles - 1 then
              j <= j + 1;
              pState <= find_ai;
         else
              j <= 0;
              pState <= delay;
          end if;

          axi <= axi + SIGNED(axij);
          ayi <= ayi + SIGNED(ayij);
          azi <= azi + SIGNED(azij);
```

6

```
        when delay =>

            if delayCount < 120 then
               delayCount <= delayCount + 1;
                pState <= delay;
            else
                delayCount <= 0;
               pState <= update_ai;
            end if;

            axi <= axi + SIGNED(axij);
            ayi <= ayi + SIGNED(ayij);
            azi <= azi + SIGNED(azij);
          -- when delay_after_calculation =>
              -- if delayCount < accCaloLatency + 100 then
                   --delayCount <= delayCount + 1;
                   --pState <= delay_after_calculation;
              -- else
                  -- delayCount <= 0;
                   -- Transition back to find_ai state (or to next state where you want to perform a calculation)
                   --pState <= find_ai;
               --end if;

        when update_ai =>

            acceleration(i)(xComponent) <= axi;
            acceleration(i)(yComponent) <= ayi;
            acceleration(i)(zComponent) <= azi;

            axi <= (others => '0');
            ayi <= (others => '0');
            azi <= (others => '0');
```

7

```
if i < numberOfParticles then
    i <= i + 1;
    pState <= find_ai;
else
    i <= 0;
    pState <= update_v;
end if;

when update_v =>

    --for each particle in each dimension
    --velocity = velocity + acceleration
    for p in 0 to (numberOfParticles-1) loop
        velocity(p)(xComponent) <= velocity(p)(xComponent) + acceleration(p)(xComponent);
        velocity(p)(yComponent) <= velocity(p)(yComponent) + acceleration(p)(yComponent);
        velocity(p)(zComponent) <= velocity(p)(zComponent) + acceleration(p)(zComponent);
    end loop;

    pState <= update_pos;

when update_pos =>

    --for each particle in each dimension
    --position = position + velocity
    for p in 0 to (numberOfParticles-1) loop
        positionWithMass(p)(xComponent) <= std_logic_vector(SIGNED(positionWithMass(p)(xComponent)) + velocity(p)(xComponent));
        positionWithMass(p)(yComponent) <= std_logic_vector(SIGNED(positionWithMass(p)(yComponent)) + velocity(p)(yComponent));
        positionWithMass(p)(zComponent) <= std_logic_vector(SIGNED(positionWithMass(p)(zComponent)) + velocity(p)(zComponent));
        --collectionUpdated <= '1';  -- Set the collectionUpdated signal to indicate the collection list update
    end loop;

    pState <= visualDelay;
```

8

9

```
                pState <= visualDelay;

            end case;
        end if;
    end process;
end Behavioral;
```

10

AccelerationCalculator.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.MATH_REAL.ALL;

entity accelerationCalculatorWithConversions is
  Port (
    clk: in std_logic;
    ce: in std_logic;
    resetn: in std_logic;
    xi: in std_logic_vector(31 downto 0);
    yi: in std_logic_vector(31 downto 0);
    zi: in std_logic_vector(31 downto 0);
    xj: in std_logic_vector(31 downto 0);
    yj: in std_logic_vector(31 downto 0);
    zj: in std_logic_vector(31 downto 0);
    mj: in std_logic_vector(31 downto 0);
    axij: out std_logic_vector(31 downto 0);
    ayij: out std_logic_vector(31 downto 0);
    a
    );
  );
end a

archi
  com
  por
  )

    component subtractor is
      Port(

      aclk : IN STD_LOGIC;
      aclken : IN STD_LOGIC;
      aresetn : IN STD_LOGIC;
      s_axis_a_tvalid : IN STD_LOGIC;
      s_axis_a_tready : OUT STD_LOGIC;
      s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
      s_axis_b_tvalid : IN STD_LOGIC;
      s_axis_b_tready : OUT STD_LOGIC;
      s_axis_b_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
      m_axis_result_tvalid : OUT STD_LOGIC;
      m_axis_result_tready : IN STD_LOGIC;
      m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
```

```
    );
    END COMPONENT;

    COMPONENT floatToFixed
      PORT (                                                    ; is
        aclk : IN STD_LOGIC;
        aclken : IN STD_LOGIC;
        aresetn : IN STD_LOGIC;
        s_axis_a_tvalid : IN STD_LOGIC;
        s_axis_a_tready : OUT STD_LOGIC;
        s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        m_axis_result_tvalid : OUT STD_LOGIC;
        m_axis_result_tready : IN STD_LOGIC;
        m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
      );
    END COMPONENT;

end component;
component delay2_dd is
    port (
        clk: in std_logic;
        en : in std_logic;
        reset: in std_logic;
        input: in std_logic_vector(31 downto 0);
        output: out std_logic_vector(31 downto 0)
    );
end component;

component delay3_mr is
    port (
        clk: in std_logic;
        en : in std_logic;
        reset: in std_logic;
        input: in std_logic_vector(31 downto 0);
        output: out std_logic_vector(31 downto 0)
    );
end component;

  COMPONENT fixedToFloat
  PORT (
    aclk : IN STD_LOGIC;
    aclken : IN STD_LOGIC;
    aresetn : IN STD_LOGIC;
    s_axis_a_tvalid : IN STD_LOGIC;
    s_axis_a_tready : OUT STD_LOGIC;
    s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    m_axis_result_tvalid : OUT STD_LOGIC;
    m_axis_result_tready : IN STD_LOGIC;
    m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
```

1

2

```
                );
  end component subtractor;

  COMPONENT multiplier
  PORT (
    aclk : IN STD_LOGIC;
    aclken : IN STD_LOGIC;
    aresetn : IN STD_LOGIC;
    s_axis_a_tvalid : IN STD_LOGIC;
    s_axis_a_tready : OUT STD_LOGIC;
    s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    s_axis_b_tvalid : IN STD_LOGIC;
    s_axis_b_tready : OUT STD_LOGIC;
    s_axis_b_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    m_axis_result_tvalid : OUT STD_LOGIC;
    m_axis_result_tready : IN STD_LOGIC;
    m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
  );
END COMPONENT;

COMPONENT adder
  PORT (
    aclk : IN STD_LOGIC;
    aclken : IN STD_LOGIC;
    aresetn : IN STD_LOGIC;
    s_axis_a_tvalid : IN STD_LOGIC;
    s_axis_a_tready : OUT STD_LOGIC;
    s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    s_axis_b_tvalid : IN STD_LOGIC;
    s_axis_b_tready : OUT STD_LOGIC;
    s_axis_b_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    m_axis_result_tvalid : OUT STD_LOGIC;
```

```
    m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
  );

END component;

 COMPONENT invSqroot
  PORT (
    aclk : IN STD_LOGIC;
    aclken : IN STD_LOGIC;
    aresetn : IN STD_LOGIC;
    s_axis_a_tvalid : IN STD_LOGIC;
    s_axis_a_tready : OUT STD_LOGIC;
    s_axis_a_tdata : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    m_axis_result_tvalid : OUT STD_LOGIC;
    m_axis_result_tready : IN STD_LOGIC;
    m_axis_result_tdata : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
  );
END COMPONENT;
  signal xint, yint, zint: integer range -10000 to 10000;
  signal axij_conv, ayij_conv, azij_conv: std_logic_vector(31 downto 0);
  signal xi_conv, yi_conv, zi_conv, xj_conv, yj_conv, zj_conv, mj_conv: std_logic_vector(31 downto 0);
  signal rx_i,ry_i,rz_i,rx_sq, ry_sq, rz_sq, dd_xy, dd_ze, dd, ddsq, mrx, mry, mrz, dd3, d: std_logic_vector(31 downto 0);
  signal EPS: std_logic_vector (31 downto 0) := x"2D1BC3B8";
  signal resetn_inv : std_logic;
  signal mj_delayed, mrx_delayed, mry_delayed, mrz_delayed, d_delayed: std_logic_Vector(31 downto 0);

begin
resetn_inv <= not resetn;
--xint <= TO_INTEGER(SIGNED(axij));
--yint <= TO_INTEGER(SIGNED(ayij));
--zint <= TO_INTEGER(SIGNED(azij));
    fixed_converter_xi: fixedToFloat
```

3

```
 fixed_converter_xi: fixedToFloat
 PORT MAP (
 aclk => clk,
 aclken => ce,
 aresetn => resetn_inv,
 s_axis_a_tvalid => '1',
 s_axis_a_tdata => xi ,
 m_axis_result_tready => '1',
 m_axis_result_tdata => xi_conv
);
 fixed_converter_yi: fixedToFloat
 PORT MAP (
 aclk => clk,
 aclken => ce,
 aresetn => resetn_inv,
 s_axis_a_tvalid => '1',
 s_axis_a_tdata => yi ,
 m_axis_result_tready => '1',
 m_axis_result_tdata => yi_conv
);
 fixed_converter_zi: fixedToFloat
 PORT MAP (
 aclk => clk,
 aclken => ce,
 aresetn => resetn_inv,
 s_axis_a_tvalid => '1',
 s_axis_a_tdata => zi ,
 m_axis_result_tready => '1',
 m_axis_result_tdata => zi_conv
);
 fixed_converter_xj: fixedToFloat
 PORT MAP (
```

4

5

6

```
 fixed_converter_xj: fixedToFloat
  PORT MAP (
  aclk => clk,
  aclken => ce,
  aresetn => resetn_inv,
  s_axis_a_tvalid => '1',
  s_axis_a_tdata => xj ,
  m_axis_result_tready => '1',
  m_axis_result_tdata => xj_conv
);
 fixed_converter_yj: fixedToFloat
  PORT MAP (
  aclk => clk,
  aclken => ce,
  aresetn => resetn_inv,
  s_axis_a_tvalid => '1',
  s_axis_a_tdata => yj ,
  m_axis_result_tready => '1',
  m_axis_result_tdata => yj_conv
);
 fixed_converter_zj: fixedToFloat
  PORT MAP (
  aclk => clk,
  aclken => ce,
  aresetn => resetn_inv,
  s_axis_a_tvalid => '1',
  s_axis_a_tdata => zj ,
  m_axis_result_tready => '1',
  m_axis_result_tdata => zj_conv
);
fixed_converter_mj: fixedToFloat
  PORT MAP (
```

7

```
fixed_converter_mj: fixedToFloat
  PORT MAP (
  aclk => clk,
  aclken => ce,
  aresetn => resetn_inv,
  s_axis_a_tvalid => '1',
  s_axis_a_tdata => mj ,
  m_axis_result_tready => '1',
  m_axis_result_tdata => mj_conv
);

get_rx: subtractor
  port map(  aclk => clk,
    aclken => ce,
    aresetn => resetn_inv,
    s_axis_a_tvalid => '1',
    m_axis_result_tready => '1',
    s_axis_b_tvalid => '1',

    s_axis_a_tdata => xj_conv,
    s_axis_b_tdata => xi_conv,
    m_axis_Result_tdata => rx_i
              );

get_ry: subtractor
  port map(  aclk => clk,
    aclken => ce,
    aresetn => resetn_inv,
    s_axis_a_tvalid => '1',
    m_axis_result_tready => '1',
    s_axis_b_tvalid => '1',
```

8

```
      s_axis_b_tdata => yi_conv,
      m_axis_Result_tdata => ry_i
                );

  get_rz: subtractor
    port map(  aclk => clk,
      aclken => ce,
      aresetn => resetn_inv,
      s_axis_a_tvalid => '1',
      m_axis_result_tready => '1',
      s_axis_b_tvalid => '1',

      s_axis_a_tdata => zj_conv,
      s_axis_b_tdata => zi_conv,
      m_axis_Result_tdata => rz_i
                );

    get_rxsq: multiplier
    PORT MAP (
    aclk => clk,
      aclken => ce,
      aresetn => resetn_inv,
      s_axis_a_tvalid => '1',
      m_axis_result_tready => '1',
      s_axis_b_tvalid => '1',

      s_axis_a_tdata => rx_i,
      s_axis_b_tdata => rx_i,
      m_axis_Result_tdata => rx_sq
                );
```

```
    get_rysq: multiplier
    PORT MAP (
    aclk => clk,
      aclken => ce,
      aresetn => resetn_inv,
      s_axis_a_tvalid => '1',
      m_axis_result_tready => '1',
      s_axis_b_tvalid => '1',

      s_axis_a_tdata => ry_i,
      s_axis_b_tdata => ry_i,
      m_axis_Result_tdata => ry_sq
                );

  get_rzsq: multiplier
    PORT MAP (
    aclk => clk,
      aclken => ce,
      aresetn => resetn_inv,
      s_axis_a_tvalid => '1',
      m_axis_result_tready => '1',
      s_axis_b_tvalid => '1',

      s_axis_a_tdata => rz_i,
      s_axis_b_tdata => rz_i,
      m_axis_Result_tdata => rz_sq
                );

get_ddxy: adder

  PORT MAP (
    aclk => clk,
```

```
    aclk => clk,
      aclken => ce,
      aresetn => resetn_inv,
      s_axis_a_tvalid => '1',
      m_axis_result_tready => '1',
      s_axis_b_tvalid => '1',

      s_axis_a_tdata => rx_sq,
      s_axis_b_tdata => ry_sq,
      m_axis_Result_tdata => dd_xy
                );

 get_ddze: adder

PORT MAP (
  aclk => clk,
    aclken => ce,
    aresetn => resetn_inv,
    s_axis_a_tvalid => '1',
    m_axis_result_tready => '1',
    s_axis_b_tvalid => '1',

    s_axis_a_tdata => rz_sq,
    s_axis_b_tdata => EPS,
    m_axis_Result_tdata => dd_ze
                );

get_dd: adder

PORT MAP (
  aclk => clk,
    aclken => ce,
```

9

```
    aresetn => resetn_inv,
    s_axis_a_tvalid => '1',
    m_axis_result_tready => '1',
    s_axis_b_tvalid => '1',

    s_axis_a_tdata => dd_xy,
    s_axis_b_tdata => dd_ze,
    m_axis_Result_tdata => dd
              );

  get_ddsq: multiplier
  PORT MAP (
  aclk => clk,
    aclken => ce,
    aresetn => resetn_inv,
    s_axis_a_tvalid => '1',
    m_axis_result_tready => '1',
    s_axis_b_tvalid => '1',

    s_axis_a_tdata => dd,
    s_axis_b_tdata => dd,
    m_axis_Result_tdata => ddsq
              );


   delayed_mj: delay1_mj
   port map(
      clk => clk,
      en => ce,
      reset => resetn_inv,
      input => mj_conv,
      output => mj_delayed
  );
```

10

```
    delayed_d : delay2_dd
    port map(
        clk => clk,
        en => ce,
        reset => resetn_inv,
        input => dd,
        output => d_delayed
    );

    delayed_mrx : delay3_mr
    port map(
        clk => clk,
        en => ce,
        reset => resetn_inv,
        input => mrx,
        output => mrx_delayed
    );
    delayed_mry : delay3_mr
    port map(
        clk => clk,
        en => ce,
        reset => resetn_inv,
        input => mry,
        output => mry_delayed
    );
    delayed_mrz : delay3_mr
    port map(
        clk => clk,
        en => ce,
        reset => resetn_inv,
        input => mrz,
        output => mrz_delayed
    );
```

11

```
  get_mrx: multiplier
  PORT MAP (
  aclk => clk,
    aclken => ce,
    aresetn => resetn_inv,
    s_axis_a_tvalid => '1',
    m_axis_result_tready => '1',
    s_axis_b_tvalid => '1',

    s_axis_a_tdata => mj_delayed,
    s_axis_b_tdata => rx_i,
    m_axis_Result_tdata => mrx
              );

  get_mry: multiplier
  PORT MAP (
  aclk => clk,
    aclken => ce,
    aresetn => resetn_inv,
    s_axis_a_tvalid => '1',
    m_axis_result_tready => '1',
    s_axis_b_tvalid => '1',

    s_axis_a_tdata => mj_delayed,
    s_axis_b_tdata => ry_i,
    m_axis_Result_tdata => mry
              );

  get_mrz: multiplier
  PORT MAP (
  aclk => clk,
    aclken => ce,
```

12

```
    s_axis_a_tvalid => '1',
    m_axis_result_tready => '1',
    s_axis_b_tvalid => '1',

    s_axis_a_tdata => mj_delayed,
    s_axis_b_tdata => rz_i,
    m_axis_Result_tdata => mrz
              );

  get_dd3: multiplier
  PORT MAP (
  aclk => clk,
    aclken => ce,
    aresetn => resetn_inv,
    s_axis_a_tvalid => '1',
    m_axis_result_tready => '1',
    s_axis_b_tvalid => '1',

    s_axis_a_tdata => ddsq,
    s_axis_b_tdata => d_delayed,
    m_axis_Result_tdata => dd3
              );

get_d : invSqroot
PORT MAP (
  aclk => clk,
  aclken => ce,
  aresetn => resetn_inv,
  s_axis_a_tvalid => '1',
  s_axis_a_tdata => dd3,
  m_axis_result_tready => '1',
  m_axis_result_tdata => d
);
```

13

```
get_axij: multiplier
 PORT MAP (
 aclk => clk,
   aclken => ce,
   aresetn => resetn_inv,
   s_axis_a_tvalid => '1',
   m_axis_result_tready => '1',
   s_axis_b_tvalid => '1',

   s_axis_a_tdata => mrx_Delayed,
   s_axis_b_tdata => d,
   m_axis_Result_tdata => axij_conv
             );

  get_ayij: multiplier
 PORT MAP (
 aclk => clk,
   aclken => ce,
   aresetn => resetn_inv,
   s_axis_a_tvalid => '1',
   m_axis_result_tready => '1',
   s_axis_b_tvalid => '1',

   s_axis_a_tdata => mry_delayed,
   s_axis_b_tdata => d,
   m_axis_Result_tdata => ayij_conv
             );

 get_azij: multiplier
 PORT MAP (
 aclk => clk,
   aclken => ce,
```

14

```
    aresetn => resetn_inv,
    s_axis_a_tvalid => '1',
    m_axis_result_tready => '1',
    s_axis_b_tvalid => '1',

    s_axis_a_tdata => mrz_delayed,
    s_axis_b_tdata => d,
    m_axis_Result_tdata => azij_conv
              );
  float_converter_axij: floatToFixed
  PORT MAP (
  aclk => clk,
  aclken => ce,
  aresetn => resetn_inv,
  s_axis_a_tvalid => '1',
  s_axis_a_tdata => axij_conv ,
  m_axis_result_tready => '1',
  m_axis_result_tdata => axij
);
 float_converter_ayij: floatToFixed
  PORT MAP (
  aclk => clk,
  aclken => ce,
  aresetn => resetn_inv,
  s_axis_a_tvalid => '1',
  s_axis_a_tdata => ayij_conv ,
  m_axis_result_tready => '1',
  m_axis_result_tdata => ayij
);
 float_converter_azij: floatToFixed
  PORT MAP (
  aclk => clk,
```

17 **18** **19**

```
    float_converter_azij: floatToFixed
     PORT MAP (
     aclk => clk,
     aclken => ce,
     aresetn => resetn_inv,
     s_axis_a_tvalid => '1',
     s_axis_a_tdata => azij_conv,
     m_axis_result_tready => '1',
     m_axis_result_tdata => azij
    );
end Behavioral;
```

20

ScreenController.vhd:

```
writeEnable <= '1' when pState = incXY else '0';

writeAdress <= std_logic_vector(TO_UNSIGNED(x + y*H_PERIOD_cste, ADDR_WIDTH));

particleX <= TO_INTEGER(signed(collection(p)(xComponent)));
particleY <= TO_INTEGER(signed(collection(p)(yComponent)));
particleM <= TO_INTEGER(signed(collection(p)(massComponent)));


PROCESS
BEGIN
    WAIT UNTIL RISING_EDGE(clk);

    If (rst='1') then
    pState <= start;

    writedata <= "0000";

    else

        Case pState is
            When start =>
                x <= 0;
                y <= 0;
                p <= 0;
                particleExistsHere <= '0';
                pState <= checkP;

            When checkP =>
                IF ( (x > particleX - particleDisplayWidth) and
                 (x < particleX + particleDisplayWidth) and
                (y > particleX - particleDisplayWidth) and
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
library work;
use work.constantList.all;

entity screenController is
  Port (clk : in STD_LOGIC;
        rst : in std_logic;
        collection: in particleList;
        writeEnable: out STD_logic;
        writeAdress: out STD_LOGIC_VECTOR(ADDR_WIDTH-1 downto
        writeData: out STD_LOGIC_VECTOR(3 downto 0));
        --collectionUpdated: in std_logic );

end screenController;

architecture controller of screenController is

SIGNAL x: INTEGER range 0 to (H_PERIOD_cste - 1);
SIGNAL y: INTEGER range 0 to (V_PERIOD_cste - 1);
SIGNAL p: INTEGER range 0 to (numberOfParticles - 1);
SIGNAL particleExistshere: STD_LOGIC;

TYPE state is (start, checkP, evaluate, incXY);

SIGNAL pState: state;
SIGNAL particleX, particleY, particleM: INTEGER;

begin
```

1

2

```
(y < particleX + particleDisplayWidth) and
(particl                                    else
particle                                        y <= 0;
--keeps                                     end if;
END IF;                                 end if;
if (p =
    p <=                                if x < H_PERIOD_cste-1 then
    pSta                                    x <= x + 1;
else                                    else
    p <=                                    x <= 0;
    pSta                                end if;
END if;
                                        pState <= checkP;
When evaluat
    if parti
        writ
    else
        writ
    end if;                         end case;
    particle                end if;
    pState <=

                            end process;
When incXY =>               end controller;
    if x = H
        if y
            y <= y + 1;
        else
```

3

4

ScreenOutput.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
library work;
use work.constantList.all;

entity screenOutput is
 Port (clk : in std_logic;
reset : in std_logic;
readData : in std_logic_vector(3 downto 0);
readAddr : out std_logic_vector(ADDR_width -1 downto 0);
readEnable : out std_logic;
vSync : out std_logic;
hSync : out std_logic;
red : out std_logic_vector(3 downto 0);
green : out std_logic_vector(3 downto 0);
blue : out std_logic_vector(3 downto 0)
  );
end screenOutput;

architecture Behavioral of screenOutput is

component VGA_timing_controller
     port(
        clk: in STD_LOGIC;
        rst_inp: in STD_LOGIC;
        hsync_outp: out STD_LOGIC;
        vsync_outp: out STD_LOGIC;
        video_active_outp: out STD_LOGIC;
        H_position: out std_logic_vector(h_size-1 downto 0);
        V_position: out std_logic_vector(v_size-1 downto 0)
         );
```

1

```
end component;

signal vide_active: std_logic;
signal xVector: std_logic_vector(h_size-1 downto 0);
signal yVector: std_logic_vector(v_size-1 downto 0);
signal xInt: INTEGER range 0 to (h_period_cste-1);
signal yInt: INTEGER range 0 to (v_period_cste-1);

begin

VGA_timing_controller_0: VGA_timing_controller
port map (
        clk => clk,
        rst_inp => reset,
        hsync_outp => hSync,
        vsync_outp => vSync,
        video_active_outp => vide_active,
        H_position => xVector,
        V_position => yVector);

xInt <= TO_INTEGER(unsigned(xVector));
yInt <= TO_INTEGER(unsigned(yVector));

process(xINT, yINT)
begin

if ((xInt = h_period_cste-1) and (yInt = v_period_cste-1)) then
    readAddr <= (others => '0');

else
    readAddr <= std_logic_vector(TO_UNSIGNED(xInt + yInt*H_period_cste + 1,Addr_Width));
```

2

```
process(xINT, yINT)
begin

if ((xInt = h_period_cste-1) and (yInt = v_period_cste-1)) then
    readAddr <= (others => '0');

else
    readAddr <= std_logic_vector(TO_UNSIGNED(xInt + yInt*H_period_cste + 1,Addr_Width));
end if;

end process;

readEnable <= '1';

red <= readData when vide_active = '1' else "0000";
green <= readData when vide_active = '1' else "0000";
blue <= readData when vide_active = '1' else "0000";

end Behavioral;
```


Python code for GravitySimulation:

```
        self.x_pos = start_position[0]
        self.y_pos = start_position[1]

        self.x_vel = x_vel
        self.x_acc = 0
        self.y_vel = y_vel
        self.y_acc = 0

    def set_y_vel(self, value):
        self.y_vel = value

    def set_x_vel(self, value):
        self.x_vel = value

    def set_y_acc(self, value):
        self.y_acc = value

    def set_x_acc(self, value):
        self.x_acc = value

    def change_x_pos(self, value):
        self.x_pos += value

    def change_y_pos(self, value):
        self.y_pos += value

    def change_x_vel(self, value):
        self.x_vel += value

    def change_y_vel(self, value):
        self.y_vel += value
```

```python
import pygame
import itertools
from sys import exit
import random
import math


pygame.init()
WINDOW_SIZE = (1000,1000)
WINDOW = pygame.display.set_mode(WINDOW_SIZE)
CLOCK = pygame.time.Clock()

BACKGROUND = pygame.Surface(WINDOW_SIZE)
BACKGROUND.fill("Black")
BACKGROUND_RECT = BACKGROUND.get_rect(center=(500,500))

G = 1

class Body(pygame.sprite.Sprite):

    def __init__(self, mass, radius, start_position, color, x_vel, y_vel):
        super().__init__()

        self.image = pygame.Surface((radius*2, radius*2))
        self.image.fill("black")
        self.rect = self.image.get_rect(center = start_position)
        pygame.draw.circle(self.image, color, (radius, radius), radius, 0)

        self.mass = mass
        self.radius = radius
        self.x_pos = start_position[0]
```

1

2

```python
def update_pos(self):
    self.rect.center = (round(self.x_pos), round(self.y_pos))

def animate(self):
    self.change_x_vel(self.x_acc)
    self.change_y_vel(self.y_acc)

    self.change_x_pos(self.x_vel)
    self.change_y_pos(self.y_vel)

    self.update_pos()

def gravitate(self, otherbody):
    dx = abs(self.x_pos - otherbody.x_pos)
    dy = abs(self.y_pos - otherbody.y_pos)

    if dx < self.x_pos*2 and dy < self.radius*2:
        pass

    else:
        try:
            r = math.sqrt(dx**2 + dy**2)
            a = G*otherbody.mass/(r**2)
            theta = math.asin(dy/r)

            if self.y_pos > otherbody.y_pos:
                self.set_y_acc(-math.sin(theta)*a)
            else:
                self.set_y_acc(math.sin(theta)*a)
```

3

```python
                else:
                    self.set_y_acc(math.sin(theta)*a)
                if self.x_pos > otherbody.x_pos:
                    self.set_x_acc(-math.cos(theta)*a)
                else:
                    self.set_x_acc(math.cos(theta)*a)

            except ZeroDivisionError:
                pass

body_group = pygame.sprite.Group()
BODYCOUNT = 10
for i in range(BODYCOUNT):
    #if i == 0:
     #   body_group.add(Body(100, 10, (500,500), "cyan", 0, 0))

    body_group.add(Body(1, 3, (random.randrange(100,900), random.randrange(100,900)), "white", 0, 0))

body_list = list(body_group)
body_pairs = list(itertools.combinations(body_list, 2))



while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            exit()
    WINDOW.blit(BACKGROUND, BACKGROUND_RECT)
    body_group.draw(WINDOW)
```

4

```
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            exit()
    WINDOW.blit(BACKGROUND, BACKGROUND_RECT)
    body_group.draw(WINDOW)

    for body, otherbody in body_pairs:
        body.gravitate(otherbody)
        otherbody.gravitate(body)
        body.animate()
        otherbody.animate()

    pygame.display.update()
    CLOCK.tick(60)
```