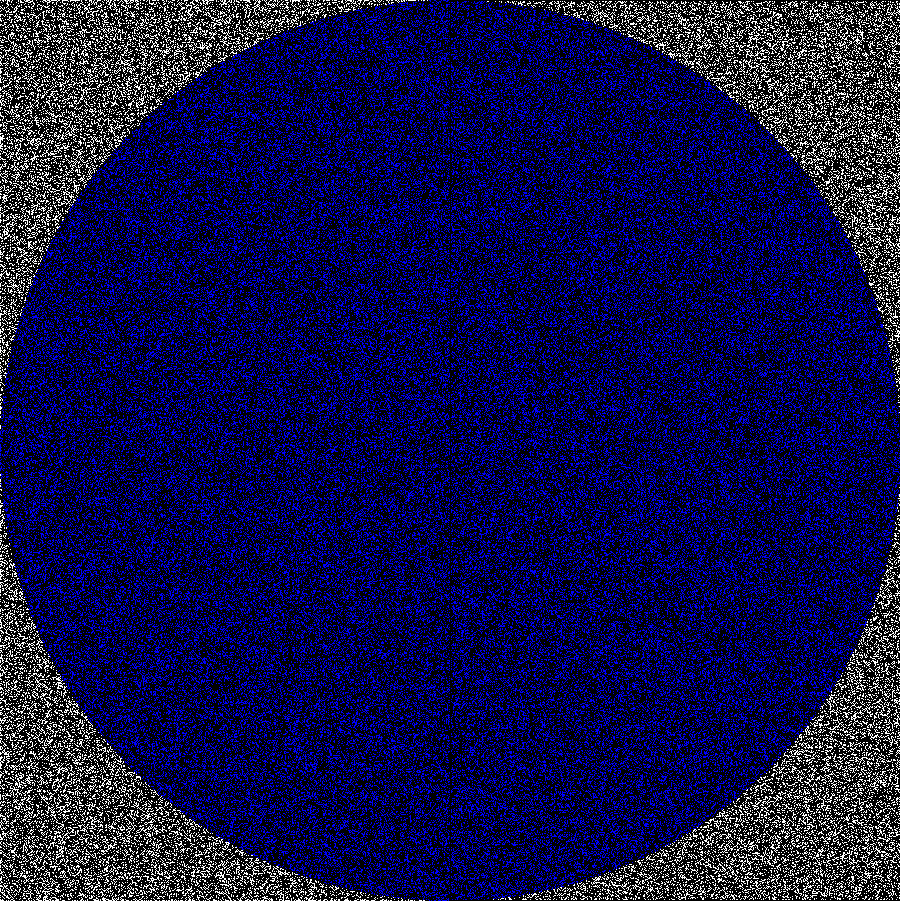
**ELEC 204 FINAL PROJECT**

**MURAT GÜÇ 68967**

**HASAN EFE CAN 72661**

**Pi Approximation**

**Introduction and Objectives**

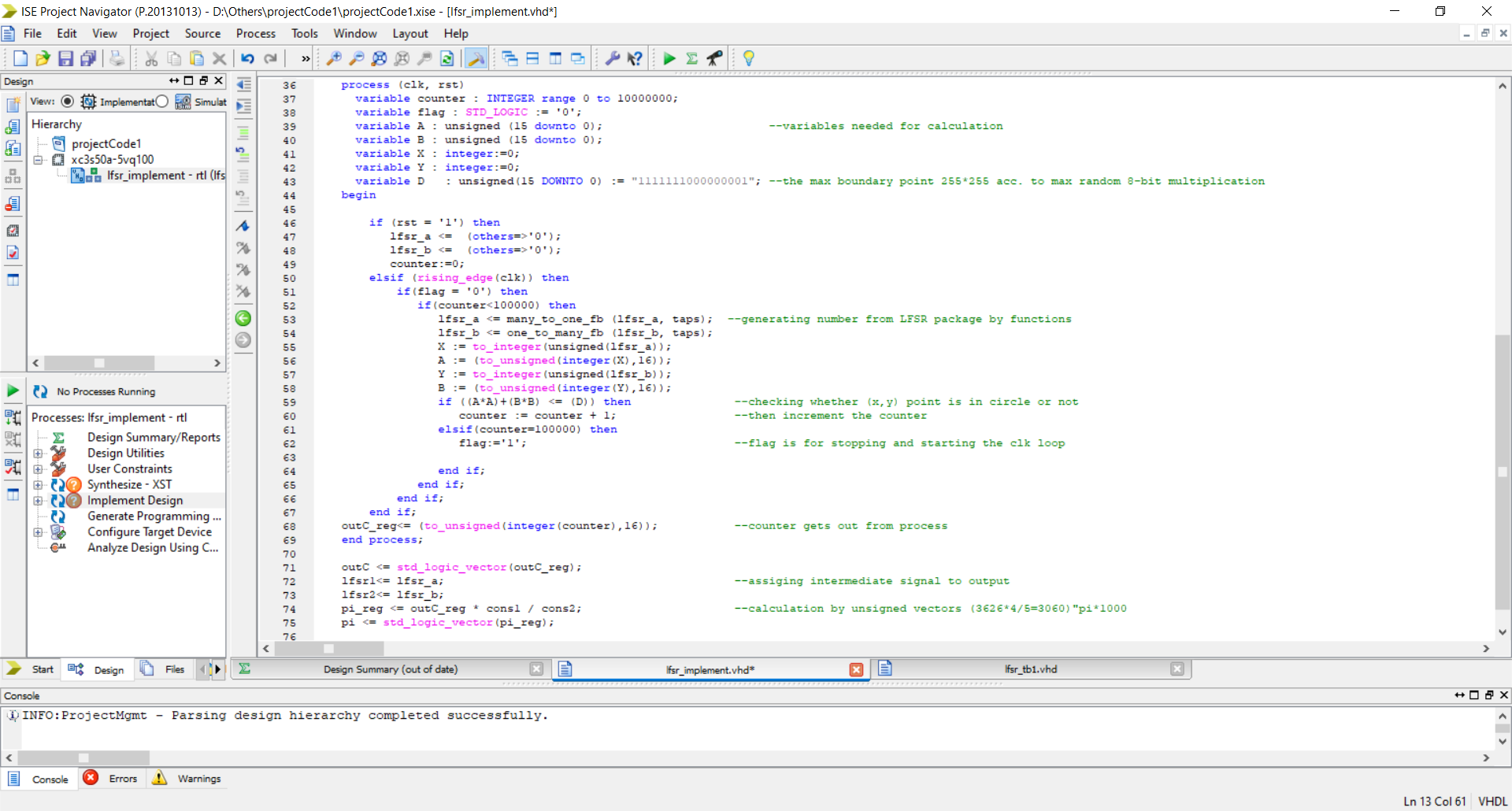
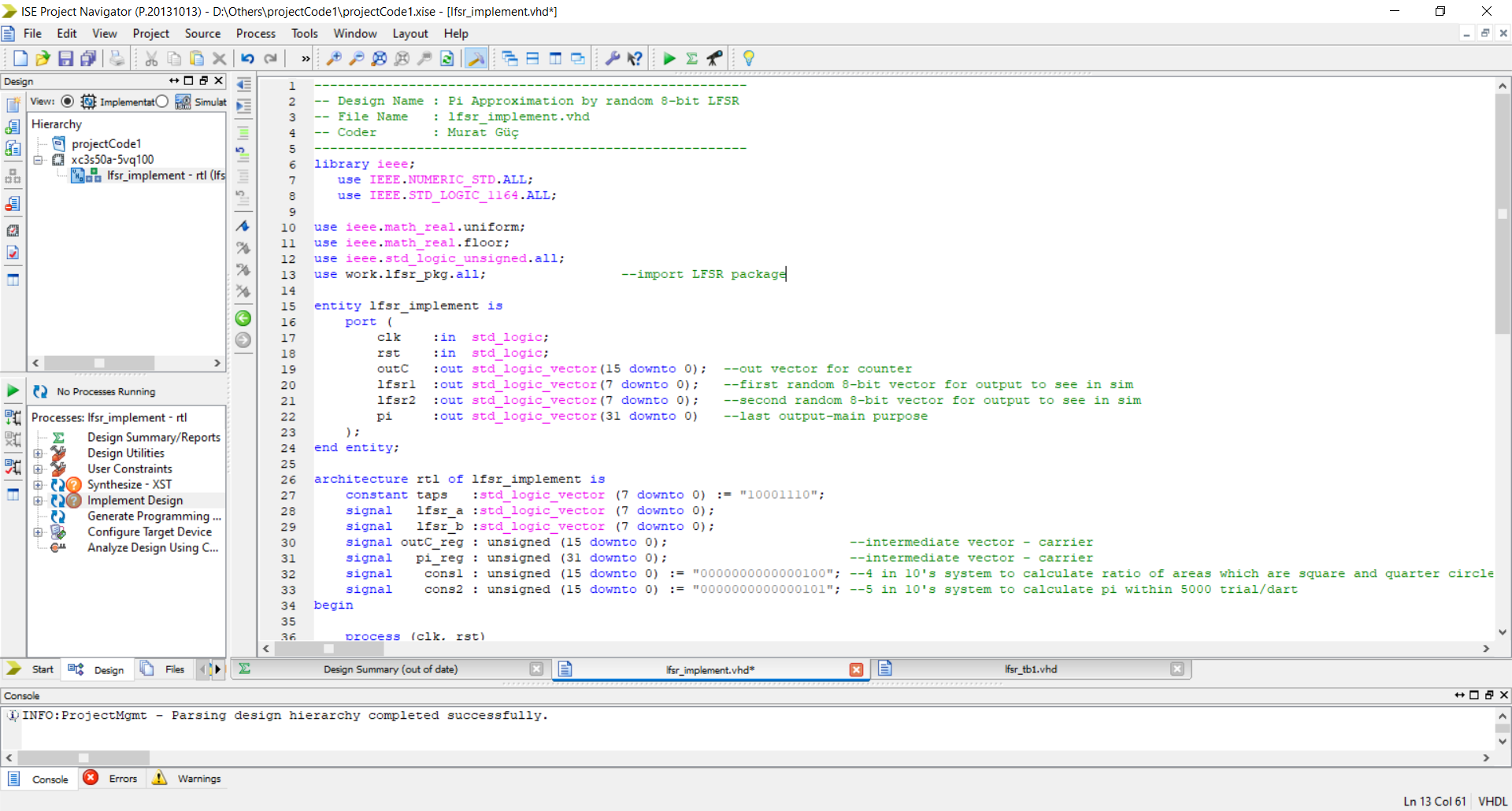
The objective is to calculate pi. The project uses The Monte Carlo Integration algorithm. This algorithm chooses random points from a 2x2 square. There is also a circle that is concentric to the square. The points should be distributed proportional to their area because there is a pi constant in the circles area it can be calculated. 

This is the main equation that helps with deriving pi. When number of points get higher there should be a better approximation in theory.

**Description**

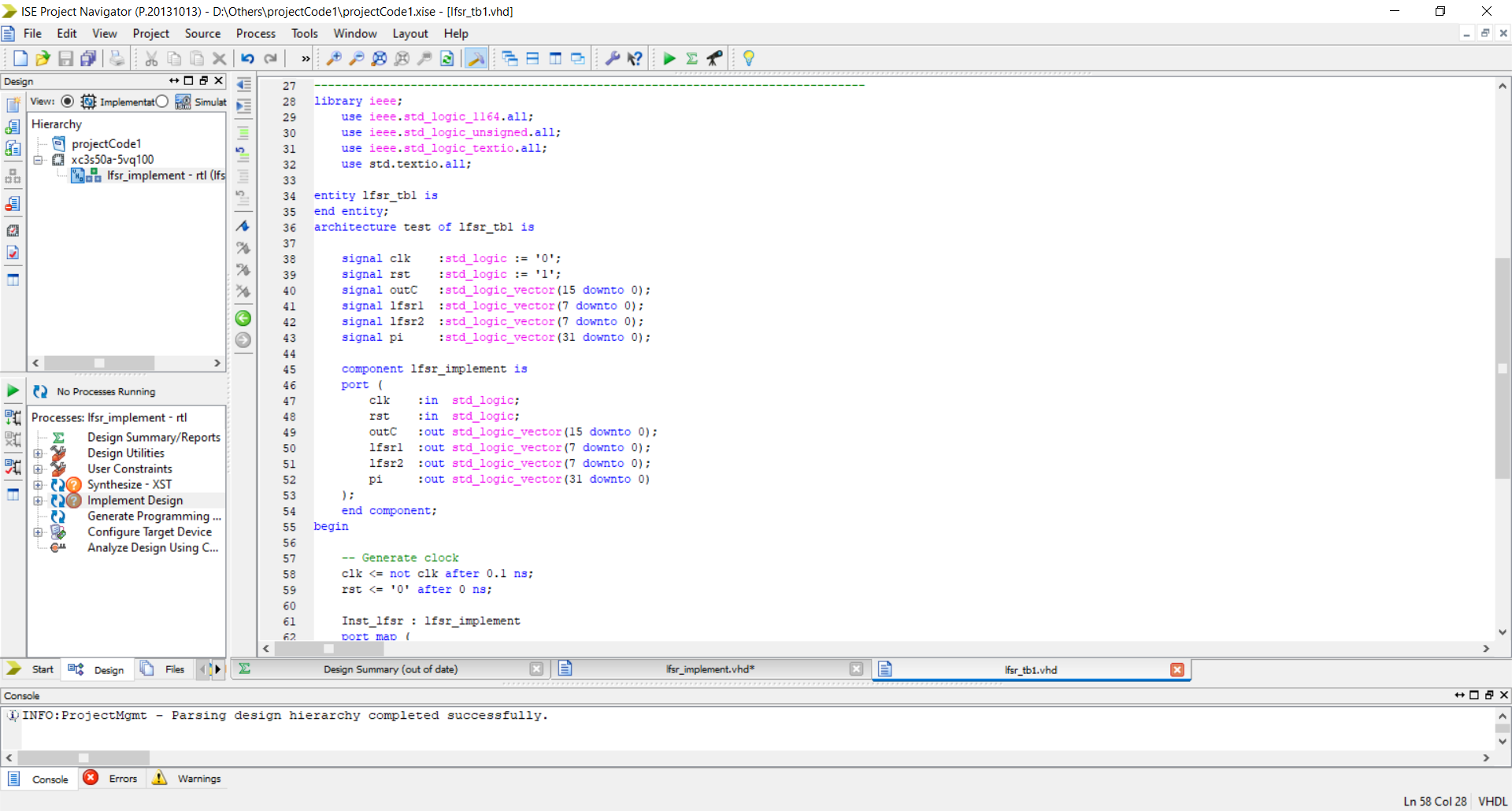
There needs to be a function that can determine on a random number between two numbers These two numbers will be thought as the x-coordinate and y-coordinate. The square will be between lines x=1 and x=-1 on the vertical and between y=1 and y=-1 on the horizontal. Each point that are determined by the two random points will be checked if it satisfies the circles inequality.

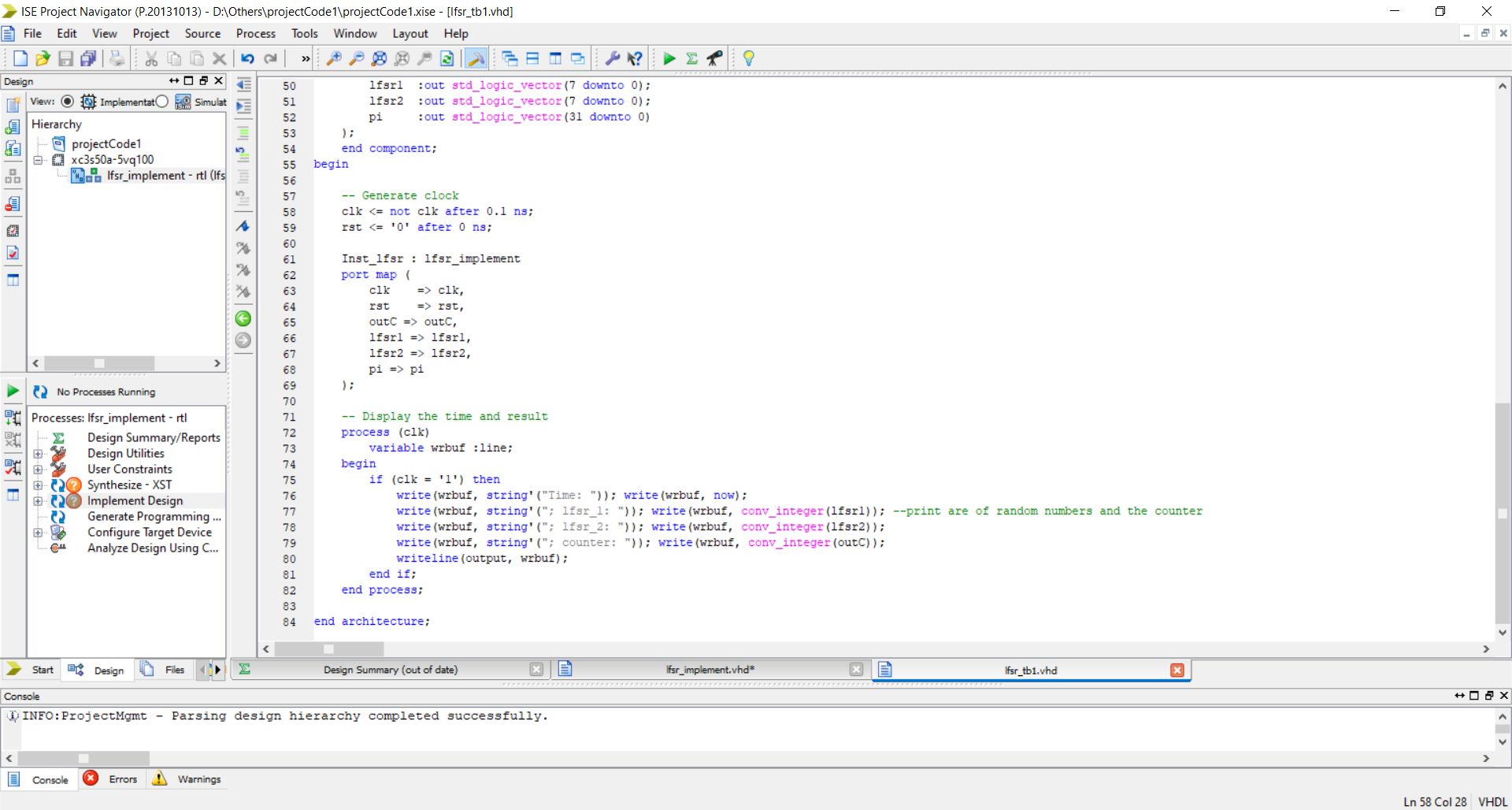
Counter will increment by 1 for every point that is in the circle or satisfies the inequality. When all the iteration or the loops are done the final value of the counter will be equal to Nc or the points that are in the circle and the number of iterations will be equal to Np or all the sample points. When the iterations are all done Nc and Np will be determined and the approximate value of pi can be calculated by the equation mentioned above.

**Methodology**

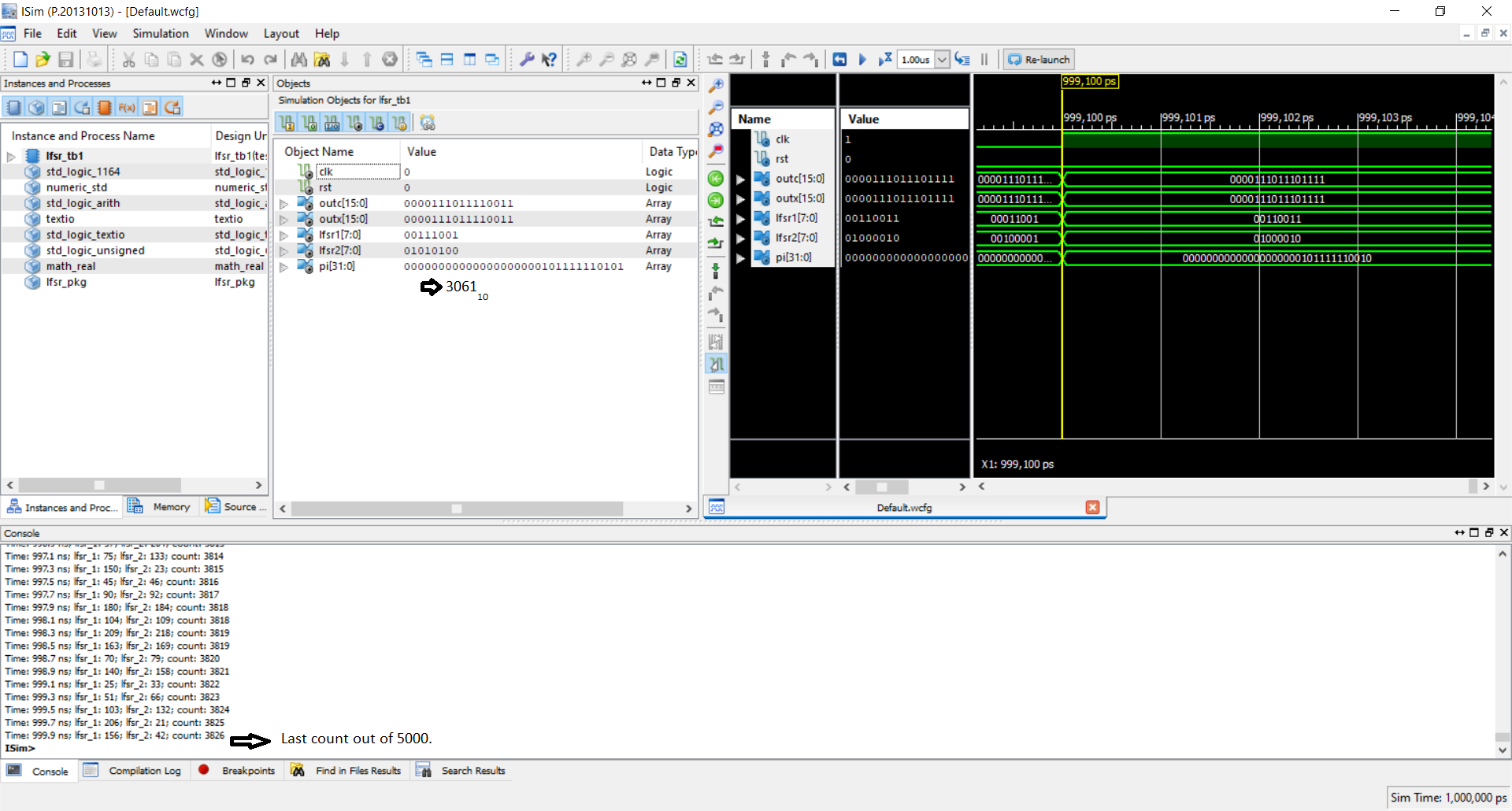
The signal cons1 and cons2 are constant signals and which are respectively 4 and 5. These are the constant numbers mentioned above, 4 can be thought as the are of the square and 5 is equal to Np/1000, for Np = 5000, which gives the final solution pi as pi\*1000.

lfsr\_a and lfsr\_b signals use the functions that use Linear Feedback Shift Register (LFSR) to randomize and these randomized 7 bit signals are used to determine the X and Y signals which are the corresponding coordinates of random points or darts. X and Y signals are unsigned integer versions of random 7-bit signals, so they are upper bounded by 255 and lower bounded by 0. Random points can be on the range of 0-255 on both x and y axis, to check if they are in the quarter circle if sum of X^2 and Y^2 is lower than 255^2 which is 65025 or 1111111000000001 in binary which is the constant signal D in the code.

If the above condition of the darts landing in the circle or the random point being inside the circle is satisfied counter increments itself by 1 and at the end of all the iterations the final value of the counter is assigned to outC signal which is Nc or the points that were in the circle.

**Simulation**

For the simulation first the clock is generated then it is given as input to the lfsr\_implement function which is introduced in the above methodology part. Then lfsr\_a, lfsr\_b and outC signals are printed out for each iteration.

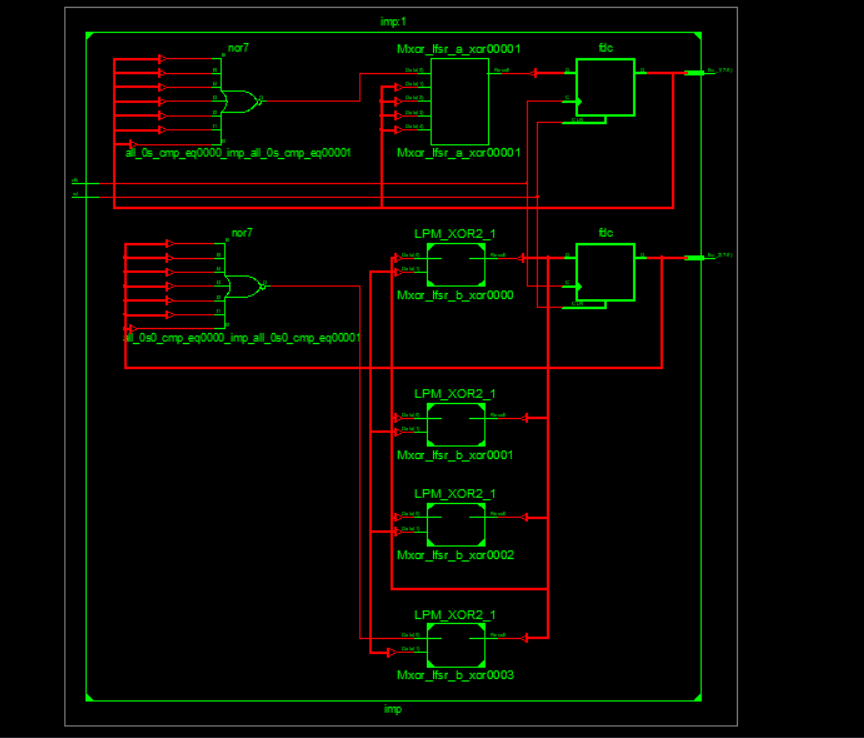
The simulation timing diagram is given. The output values can change at every rising edge of the clock.

At every clock edge lfsr\_a and lfsr\_b becomes another random number.

Outc increases 1 in every loop or rising clock edge.

Pi increases if integer versions of lfsr\_a and lfsr\_b which are X and Y satisfy the equation X^2 + Y^2 < D

**RTL Schematics of LFSR:**



Generator of LFSR package uses Galois LFSR’s procedure with feedback system one\_to\_many and many\_to\_one methods which are used in project. In the Galois configuration, when the system is clocked, bits that are not taps are shifted one position to the right unchanged. The taps, on the other hand, are XORed with the output bit before they are stored in the next position which means pattern. We used that as “10001110”. The new output bit is the next input bit. The effect of this is that when the output bit is zero, all the bits in the register shift to the right unchanged, and the input bit becomes zero. When the output bit is one, the bits in the tap positions all flip (if they are 0, they become 1, and if they are 1, they become 0), and then the entire register is shifted to the right and the input bit becomes 1.

Here is the package code:

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

package lfsr\_pkg is

-- LFSR Feedback for 2\*\*n

function many\_to\_one\_fb (DATA, TAPS :std\_logic\_vector) return std\_logic\_vector;

function one\_to\_many\_fb (DATA, TAPS :std\_logic\_vector) return std\_logic\_vector;

end;

package body lfsr\_pkg is

function many\_to\_one\_fb (DATA, TAPS :std\_logic\_vector) return std\_logic\_vector is

variable xor\_taps :std\_logic;

variable all\_0s :std\_logic;

variable feedback :std\_logic;

begin

-- Validate if lfsr = to zero (Prohibit Value)

if (DATA(DATA'length-2 downto 0) = 0) then

all\_0s := '1';

else

all\_0s := '0';

end if;

xor\_taps := '0';

for idx in 0 to (TAPS'length-1) loop

if (TAPS(idx) = '1') then

xor\_taps := xor\_taps xor DATA(idx);

end if;

end loop;

feedback := xor\_taps xor all\_0s;

return DATA((DATA'length-2) downto 0) & feedback;

end function;

function one\_to\_many\_fb (DATA, TAPS :std\_logic\_vector) return std\_logic\_vector is

variable xor\_taps :std\_logic;

variable all\_0s :std\_logic;

variable feedback :std\_logic;

variable result :std\_logic\_vector (DATA'length-1 downto 0);

begin

-- Validate if lfsr = to zero (Prohibit Value)

if (DATA(DATA'length-2 downto 0) = 0) then

all\_0s := '1';

else

all\_0s := '0';

end if;

feedback := DATA(DATA'length-1) xor all\_0s;

-- XOR the taps with the feedback

result(0) := feedback;

for idx in 0 to (TAPS'length-2) loop

if (TAPS(idx) = '1') then

result(idx+1) := feedback xor DATA(idx);

else

result(idx+1) := DATA(idx);

end if;

end loop;

return result;

end function;

end package body;

**Conclusion**

Monte Carlo Integration is used in this project and this algorithm gives better output with more iterations. This algorithm prints out 3.06 as the pi approximation better approximation can be made with more loops. Linear feedback shift register is used to take the random point. Random points are determined by two random numbers and those numbers are checked if it is in the circles area. With the equation that connects probability and area, pi can be approximately calculated. The equation depends on probability so when more random operations are done, better approximation can be made.

**Better Approximation**

LFSR (Linear feedback shift register) is used in this code for achieving random numbers but, this is not really the case. LFSR gives pseudo random numbers meaning there is a correlation between number changes, but the pattern is complicated and random enough to consider it random. Another reason decreasing the accuracy is the number of sample points or darts thrown, in this program only 5000 points are chosen randomly if the number of points increase accuracy of the program will also increase correspondingly. We know theoretically the last result must be equal to pi but in practice because the algorithm depends on luck and probability. To decrease the difference between experimental probability and theoretical probability, the number of tries or experiments must increase.

**Sources**:

1. <http://www.asic-world.com/>
2. <https://en.wikipedia.org/wiki/Linear-feedback_shift_register>
3. <https://www.easics.com/crctool/>
4. <https://www.nandland.com/vhdl/tips/tip-convert-numeric-std-logic-vector-to-integer.html>